GUILHERME BARROSO MAINIERI

HEURÍSTICAS PARA A MINIMIZAÇÃO DO ATRASO TOTAL NO
AMBIENTE *FLOWSHOP* COM MÚLTIPLOS PROCESSADORES

Orientadora: Prof.^a Dr.^a Débora Pretti Ronconi

Dissertação apresentada à Escola
Politécnica da Universidade de São Paulo,
como parte dos requisitos exigidos para
obtenção do título de mestre em
Engenharia de Produção.

São Paulo
2009

FICHA CATALOGRÁFICA

Mainieri, Guilherme Barroso

**Heurística para a minimização do atraso total no ambiente
flowshop com múltiplos processadores / G.B. Mainieri. -- São
Paulo, 2009.
p.65**

**Dissertação (Mestrado) - Escola Politécnica da Universidade
de São Paulo. Departamento de Engenharia de Produção.**

**1. Scheduling 2. Heurística 3. Sequenciação da produção
4. Pesquisa operacional 5. Programação da produção I. Univer-
sidade de São Paulo. Escola Politécnica. Departamento de
Engenharia de Produção II. t.**

DEDICATÓRIA

Aos meus pais e a todos que me ensinaram a importância do estudo.

AGRADECIMENTOS

À minha família, pela dedicação e apoio.

À professora Débora, pela orientação, pelo apoio, pelo conhecimento passado e pela motivação dados durante o desenvolvimento do trabalho.

Aos meus amigos, colegas, professores e todos os outros que me ajudaram no desenvolvimento deste trabalho.

À Geun C. Lee por gentilmente fornecer as instâncias utilizadas neste trabalho.

Esta pesquisa teve o apoio financeiro da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) e da Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP).

RESUMO

Neste trabalho será estudado um ambiente de produção que é freqüentemente encontrado na prática: o *flowshop* com múltiplos processadores. No caso estudado existem estágios em série e em cada estágio existe um número de máquinas idênticas em paralelo. Todas as tarefas devem ser processadas por todos os estágios. O objetivo é minimizar o atraso das tarefas.

Primeiramente o problema foi abordado através de um método que programa as tarefas por estágio e em ordem direta, ou seja, do primeiro para o último estágio. Em seguida, foram desenvolvidas duas novas regras que utilizam o mesmo método de programação, porém consideram o ambiente como uma série de problemas de máquinas em paralelo. Uma das regras desenvolvidas tem como característica principal considerar estados futuros do sistema.

Também foi desenvolvido um novo método de programação em ordem inversa, no qual as tarefas são programadas do último para o primeiro estágio. Este método apresenta melhor desempenho se comparado com o método de programação em ordem inversa da literatura.

Por último foi desenvolvido um método de programação com foco no estágio gargalo, visto que este estágio pode impedir um bom fluxo das tarefas pelo sistema e resultar em uma conclusão tardia das mesmas. Este método é mais simples, rápido e tem resultados competitivos frente ao método com foco no gargalo da literatura.

ABSTRACT

This work considers a production environment that is frequently found in practice: the flexible flowshop. In the case studied, there are stages in series and in each stage there are a number of identical parallel machines. All jobs must be processed by all stages. The objective is to minimize the tardiness of jobs.

First the problem was addressed by a method in which jobs are schedule forward, that is, from first to last stage. Two new rules were developed using this same method, but considering the environment as a series of parallel machines problems. One of the rules is able to consider future states of the system.

It was also developed a new method in which jobs are scheduled backward, i.e., from last to first stage. This method shows better performance compared to the literature method.

At last, it was developed a method that focus on the bottleneck stage scheduling (since this stage may prevent a good flow of jobs throughout the system and result in late completions). This method is simpler, faster and competitive next to the literature method.

SUMÁRIO

INTRODUÇÃO	1
1. CARACTERIZAÇÃO DO PROBLEMA	3
1.1. Definição do problema	3
1.2. Objetivo	7
1.3. Formulação matemática.....	9
1.4. Revisão bibliográfica.....	12
2. HEURÍSTICA CONSTRUTIVA	18
2.1. Método de programação por lista de prioridade.....	18
2.2. Método de programação em ordem inversa	23
3. RESULTADOS COMPUTACIONAIS	32
3.1. Instâncias teste.....	32
3.2. Testes preliminares	33
3.3. Método de programação por lista de prioridades	34
3.4. Método de programação em ordem inversa	36
3.5. Comparações entre os métodos de programação.....	38
3.6. Comparações com métodos da literatura.....	39
4. ALGORITMO COM FOCO NO GARGALO	43
4.1. Método proposto.....	43
4.2. Algoritmo para o método proposto.....	46
4.3. Exemplo ilustrativo	48
5. RESULTADOS FINAIS	54
5.1. Testes preliminares	54
5.2. Comparação entre as regras.....	54

5.3.	Comparação entre os três métodos	56
5.4.	Comparação com a literatura.....	56
6.	CONCLUSÕES.....	60
	REFERÊNCIAS BIBLIOGRÁFICAS.....	62

LISTA DE TABELAS

Tabela 1 - Parâmetros para o exemplo do método de programação em ordem inversa.	26
Tabela 2 - Programação obtida através do método de programação por lista de prioridade.....	27
Tabela 3 - Problema inverso formulado no Passo 1.	28
Tabela 4 - Programação do problema inverso.	29
Tabela 5 - Programação final obtida através do método de programação em ordem inversa.....	31
Tabela 6 - Média do RDI para o método de programação por lista de prioridades.....	35
Tabela 7 - Tempo computacional (em ms) para as regras de despacho consideradas.....	36
Tabela 8 - Média do RDI para o método de programação em ordem inversa.....	37
Tabela 9 - Tempo computacional (em ms) para o método de programação em ordem inversa....	37
Tabela 10 - Média do RDI para as cinco regras e para os dois métodos de programação.	38
Tabela 11 – Média do RDI para os dois métodos de programação.....	39
Tabela 12 - Comparação para o método de programação por lista de prioridade (RDI médio). ...	39
Tabela 13 - Comparação entre os métodos de programação em ordem inversa (RDI médio).....	41
Tabela 14 - Comparação em relação ao atraso total.....	42
Tabela 15 - Método de decomposição do algoritmo com foco no gargalo.	45
Tabela 16 - Parâmetros para o exemplo do algoritmo com foco no gargalo.....	48
Tabela 17 - Programação completa para a 1ª iteração do algoritmo com foco no gargalo.	50
Tabela 18 - Método RT3.	51
Tabela 19 - Programação completa para a 2ª iteração.....	52
Tabela 20 - RDI médio para o algoritmo com foco no gargalo.....	55
Tabela 21 – Média do tempo computacional para o algoritmo (em ms).....	55
Tabela 22 - Média do RDI para os três métodos.	56
Tabela 23 - Média do RDI para a comparação com o método da literatura.....	57

Tabela 24 - Média do RDI em função da posição do gargalo.	58
Tabela 25 - Tempo computacional para o algoritmo com foco no gargalo.	58

LISTA DE FIGURAS

Figura 1 - Exemplo do ambiente <i>flowshop</i> flexível.	4
Figura 2 - Exemplo de Gráfico de Gantt para o ambiente <i>flowshop</i> flexível.	4
Figura 3 - Grafo $G(\pi)$ e caminho crítico para o exemplo de <i>flowshop</i> flexível.	7
Figura 4 - Programação gerada pelo método da lista de prioridades (PRTTa).	21
Figura 5 – Fluxograma para o algoritmo com foco no gargalo.	46

LISTA DE GRÁFICOS

Gráfico 1 - Gráfico de Gantt para o método de programação por lista de prioridade.	27
Gráfico 2 - Gráfico de Gantt para programação do problema inverso.	29
Gráfico 3 - Gráfico de Gantt para a programação (Passo 5)	30
Gráfico 4 - Gráfico de Gantt para a programação final.	31
Gráfico 5 - Comportamento do atraso total em função de α	33
Gráfico 6 – Gráfico de Gantt para o problema de máquinas em paralelo do gargalo.	49
Gráfico 7 - Programação para os problemas dos estágios anteriores ao gargalo e do gargalo.	49
Gráfico 8 - Gráfico de Gantt para a 1ª iteração do algoritmo com foco no gargalo.	50
Gráfico 9 - Programação dos estágios anteriores e do gargalo.	51
Gráfico 10 - Programação completa para a 2ª iteração.	52

INTRODUÇÃO

De maneira geral os problemas de programação de tarefas, conhecidos na língua inglesa como *scheduling*, consistem na alocação de recursos para a execução de tarefas. Segundo Pinedo (2002), os recursos podem ser, por exemplo: máquinas, pessoas, pistas de aeroportos, unidades de processamento de computadores, entre outros, assim como tarefas podem ser: operações de processos produtivos, atividades diárias do cotidiano, aeronaves que devem pousar e/ou decolar, execução de softwares, etc. Estas tarefas podem possuir diversas características como datas de entrega, prioridade, instantes de liberação e recursos utilizados. Os problemas podem estar sujeitos a restrições que devem ser respeitadas, e possuir mais de um critério para sua otimização. Os critérios mais comuns estudados na literatura são: minimização do instante do término da última tarefa na última máquina (*makespan*), minimização do número de tarefas atrasadas, minimização do atraso máximo, minimização do atraso total, minimização do adiantamento e do atraso e minimização do desvio em relação às datas de entregas.

Nos ambientes produtivos este problema é conhecido como *scheduling* em máquinas, no qual os recursos são máquinas e as tarefas, ordens de produção. Basicamente, nestes ambientes as disposições das máquinas podem ser: uma única máquina, máquinas em paralelo, *flowshop* e *job shop*. Uma única máquina e máquinas em paralelo são as disposições mais simples possíveis. Porém para esses dois casos já é possível observar a natureza combinatória dos problemas de *scheduling*. Por exemplo, para programar n tarefas em uma única máquina existem n fatorial ($n!$) possíveis soluções.

Du e Leung (1990) mostraram que o problema da minimização do atraso total para o caso particular de uma única máquina, com datas de entrega distintas, já é *NP-hard*, ou seja, não existem algoritmos que conseguem resolver este problema em tempo polinomial.

O *flowshop* é uma configuração na qual existem m máquinas em série. Todas as tarefas têm o mesmo roteiro, ou seja, devem ser processadas primeiro na máquina 1, depois na máquina 2 e assim por diante, até a última máquina. Já no *job shop* cada tarefa tem seu próprio roteiro para passar pelas máquinas.

O *flowshop* com múltiplos processadores, que será o caso estudado, é uma generalização do *flowshop*. Também referido como *flowshop* híbrido ou como *flowshop* flexível, neste ambiente existem máquinas em paralelo organizadas em estágios. Todas as tarefas devem passar por todos os estágios, na mesma seqüência e uma única vez, começando no primeiro, passando para o segundo e assim por diante até o último estágio. Em cada estágio cada tarefa deve ser processada por uma única máquina.

O próximo capítulo apresenta o problema de *flowshop* flexível para minimização do atraso total que será considerado neste trabalho. Existem três razões principais que motivaram a escolha deste problema:

- i. Este ambiente produtivo existe em situações reais de produção. Por exemplo, o *flowshop* flexível é muito comum na indústria eletrônica na produção de Placas de Circuito Impresso (PCI). Jin *et al.*(2002) afirmam que uma típica linha de produção de PCI possui vários estágios e em cada estágio uma série de máquinas em paralelo, que são características fundamentais de um *flowshop* flexível;
- ii. O atraso é uma medida de desempenho cada vez mais importante, pois tem impacto direto na satisfação dos clientes. Segundo Lee *et al.* (2003), atender as necessidades dos clientes (em termos de quantidade e data de entrega) é tão importante como outros objetivos como qualidade e tempo de passagem pelo sistema;
- iii. Conforme Linn e Zhang (1999) e Quadts e Kuhn (2007), o atraso é uma medida de desempenho pouco explorada em comparação com outras medidas como, por exemplo, o *makespan*.

Este trabalho está estruturado da seguinte forma. No próximo capítulo é feita a descrição do problema estudado e a revisão bibliográfica. O capítulo 3 apresenta a heurística construtiva proposta para o problema em questão. No quarto capítulo são apresentados os resultados computacionais obtidos com a heurística proposta. O último capítulo apresenta as conclusões e os próximos passos.

1. CARACTERIZAÇÃO DO PROBLEMA

Este capítulo apresenta uma definição formal do problema, assim como, uma estrutura em grafo e uma formulação matemática para a minimização do atraso total no ambiente *flowshop* flexível. Por último, é apresentada a revisão bibliográfica sobre o assunto.

1.1. Definição do problema

Em situações práticas, os problemas de *flowshop* flexível têm aparecido mais freqüentemente nos ambientes produtivos, pois quando a carga de trabalho de um estágio aumenta, é necessário existir mais de uma máquina por estágio, o que acaba resultando na configuração de máquinas do *flowshop* flexível (LOGENDRAN *et al.*, 2006).

Este ambiente pode ser encontrado em diversos tipos de sistemas de manufaturas. Um exemplo é o caso da produção de sacos de papel (PINEDO, 2002). A fábrica produz sacos de papel multicamadas para cimento, carvão vegetal, comida pra cachorro, etc. A matéria prima é composta de rolos de papel. O processo de produção tem 3 estágios. No primeiro estágio os rótulos são impressos na camada externa. No segundo estágio são colados os lados do papel de tal maneira que ele adquira a forma de um tubo com as duas extremidades abertas. No último estágio são feitas as costuras no fundo do tubo ou nas duas extremidades, de acordo com o tipo de saco produzido. Em cada estágio há um número de máquinas em paralelo. Essas máquinas podem diferir na velocidade em que elas operam bem como em outros parâmetros de produção.

No *flowshop* flexível existem n tarefas representadas por $N = \{1, \dots, n\}$, e s estágios em série representados por $M = \{1, 2, \dots, s\}$. Em cada estágio l existem m_l máquinas. O conjunto de máquinas do estágio l é denotado por $M_l = \{1, \dots, m_l\}$. O tempo de processamento de uma tarefa j ($p_{jl}, j \in N$) depende apenas da tarefa e do estágio, ou seja, em cada estágio, as máquinas são idênticas. Todas as tarefas devem passar por todos os estágios, primeiro pelo estágio 1, depois pelo estágio 2, assim por diante até o último estágio s . Cada tarefa deve ser processada por apenas uma máquina de cada estágio. Um programa viável é definido como um conjunto de pares $(i_{jl}, C_{jl}), j \in N, l \in M$, sendo $i_{jl} \in M_l$ a máquina que processa a tarefa j no estágio l , e $C_{jl} \geq 0$ é o

instante de término da tarefa j no estágio l (na máquina i_{jl}), tal que as restrições acima sejam satisfeitas.

A Figura 1 mostra um exemplo de *flowshop* flexível com três estágios ($s = 3$), três máquinas nos estágios 1 e 3 ($m_1 = m_3 = 3$) e duas máquinas no estágio 2 ($m_2 = 2$). As setas entre os estágios representam as possíveis rotas das tarefas dentro do sistema.

Figura 1 - Exemplo do ambiente *flowshop* flexível.

Um exemplo de uma possível programação de quatro tarefas, no ambiente do exemplo acima, está ilustrado no gráfico de Gantt da Figura 2.

Figura 2 - Exemplo de Gráfico de Gantt para o ambiente *flowshop* flexível.

As tarefas 1 e 2, nesta ordem, entram no sistema através da máquina 1A do primeiro estágio. As tarefas 3 e 4 entram através das máquinas 1B e 1C, respectivamente. Assim que concluídas no primeiro estágio, as tarefas 1 e 4 são programadas na máquina 2A do segundo estágio e as tarefas 3 e 2, na máquina 2B. No último estágio as máquinas 3A e 3B processam as tarefas 3 e 1, respectivamente. Já a máquina 3C processa as tarefas 2 e 4. Os instantes de conclusão das tarefas 1, 2, 3 e 4 são 40, 40, 55 e 80, respectivamente.

Este problema também pode ser representado através de um grafo. Com este propósito são introduzidas duas novas noções: grupos e ordem de processamento. Um grupo é um subconjunto de tarefas designado a uma máquina em um estágio. Em cada estágio l , o conjunto de N tarefas é particionado em m_l grupos $N_{li} \subset N, n_{li} = |N_{li}|; i \in M_l, l \in M$.

A ordem de processamento das tarefas do grupo N_{li} pode ser expressa por uma permutação $\pi_{li} = (\pi_{li}(1), \dots, \pi_{li}(n_{li})) \in P(N_{li})$, sendo $\pi_{li}(k)$ o elemento de N_{li} que ocupa a posição k em π_{li} , e $P(N_{li})$ o conjunto de todas as permutações do conjunto N_{li} . Assim, o processamento das tarefas no estágio l pode ser completamente determinado pelo conjunto de m_l permutações $\pi_l = \{\pi_{l1}, \dots, \pi_{lm_l}\}$. A ordem de processamento das tarefas é definida pela s -tupla $\pi = (\pi_1, \dots, \pi_s)$. O conjunto de todas as ordens de processamento é representado por:

$$\Pi = \left\{ \pi = (\pi_1, \dots, \pi_s) \mid \pi_l = \{\pi_{l1}, \dots, \pi_{lm_l}\}, (\pi_{li} \in P(N_{li}), i \in M_l) \right\} \quad (2.1)$$

Para uma dada permutação $\pi \in \Pi$, os instantes de término das tarefas podem ser calculados em $O(ns)$ usando a seguinte fórmula recursiva para $k=1, \dots, n_{li}; i=1, \dots, m_l; l=1, \dots, s$

$$C_{\pi_{li}(k), l} = \max \left\{ C_{\pi_{li}(k-1), l}, C_{\pi_{li}(k), l-1} \right\} + p_{\pi_{li}(k), l} \quad (2.2)$$

sendo, $\pi_{li}(0) = 0, i = 1, \dots, m_l; C_{0, l} = 0, l = 1, \dots, s; C_{j, 0} = 0, j = 1, \dots, n$.

Esta representação é bastante usada em problemas que apresentam o *makespan* como medida de desempenho (veja SACCHI, 1997). O valor de *makespan* associado a π é denotado por $C_{\max}(\pi)$. Assim, $C_{\max}(\pi) = \max_{j \in N} C_{js}$. Para determinar o programa factível a partir da ordem de processamento π , é preciso alocar grupos $N_{li}, i \in M_l$, a máquinas no estágio $l, l \in M$. Já que as

máquinas nos estágios são idênticas, cada grupo pode ser alocado em qualquer uma das máquinas do respectivo estágio. Assim um programa factível (i_{jl}, C_{jl}) , $j \in N$, $l \in M$, pode ser obtido da ordem de processamento π através do seguinte procedimento: faz-se a alocação $i_{jl} \leftarrow i$, sendo $i \in M_l$, $j \in N_{li}$, e encontra-se C_{jl} através de (2.2).

Para o exemplo apresentado, são criados três grupos $N_{11}=\{1,2\}$, $N_{12}=\{3\}$ e $N_{13}=\{4\}$ no estágio 1; dois grupos $N_{21}=\{1,4\}$ e $N_{22}=\{2,3\}$ no estágio 2 e três grupos $N_{31}=\{3\}$, $N_{32}=\{1\}$ e $N_{33}=\{2,4\}$ no estágio 3. Os grupos são processados nas ordens dadas por $\pi_{11}=\{1,2\}$, $\pi_{12}=\{3\}$, $\pi_{13}=\{4\}$, $\pi_{21}=\{1,4\}$, $\pi_{22}=\{3,2\}$, $\pi_{31}=\{3\}$, $\pi_{32}=\{1\}$, $\pi_{33}=\{2,4\}$, respectivamente. A ordem de processamento das tarefas é $\pi=(\pi_1, \pi_2, \pi_3)$, sendo $\pi_1=(\pi_{11}, \pi_{12}, \pi_{13})$, $\pi_2=(\pi_{21}, \pi_{22})$, $\pi_3=(\pi_{31}, \pi_{32}, \pi_{33})$. Modificando a alocação dos grupos, esta ordem de processamento gera $3! \cdot 2! \cdot 3! = 72$ programas factíveis com o mesmo valor de *makespan*. Um destes programas é mostrado na Figura 2.

É conveniente fazer a análise através de um grafo associado a uma ordem processamento estabelecida. Esta estrutura em grafo foi utilizada por Nowicki e Smutnicki (1998). Para uma dada permutação $\pi \in \Pi$ cria-se o grafo $G(\pi) = (O, E^* \cup E(\pi))$ com um conjunto de nós $O = M \times N$ e um conjunto de arcos $E^* \cup E(\pi)$. Os arcos do conjunto E^* representam as rotas das tarefas pelos estágios, enquanto os arcos do conjunto $E(\pi)$ representam a ordem de processamento dos grupos. Cada nó $(l, j) \in O$ representa a l -ésima operação da tarefa j e tem peso p_{jl} . Todo arco tem peso zero. O instante de término da tarefa C_{jl} é encontrado através de (2.2) e é igual ao comprimento do maior caminho entre $(1, \pi_{1i}(1))$, $i \in M_1$, até o nó (l, j) , incluindo p_{jl} em $G(\pi)$. O *makespan* é igual ao maior caminho (caminho crítico) em $G(\pi)$.

O grafo $G(\pi)$ para a ordem de processamento π , do exemplo da Figura 2, é mostrado na Figura 3. Os arcos E^* são pontilhados e os arcos de $E(\pi)$ são desenhados com traçado contínuo. Os nós em cor cinza são os nós pertencentes ao caminho crítico.

Figura 3 - Grafo $G(\pi)$ e caminho crítico para o exemplo de *flowshop* flexível.

No ambiente considerado, a data de entrega de uma tarefa j (d_j) depende apenas da tarefa. Os métodos de otimização decidem quais tarefas devem ser alocadas em quais máquinas e qual a programação que elas devem seguir em cada estágio.

1.2. Objetivo

Este trabalho visa propor uma heurística para o problema de minimização do atraso total no ambiente *flowshop* flexível no qual as tarefas têm diferentes tempos de processamento em cada estágio. Este problema é *NP-hard*, visto que a minimização do atraso total considerando uma única máquina é *NP-hard* (DU e LEUNG, 1990). Portanto, o objetivo deste trabalho é fornecer uma heurística capaz de gerar soluções com qualidade de resultados num tempo satisfatório para problemas de maiores dimensões.

Será considerado o caso determinístico estático, ou seja, as datas de entrega e os tempos de processamento das tarefas são conhecidos e não variantes, todas as tarefas e prazos estão definidos na data de início da operação e todas as tarefas e máquinas estão disponíveis no começo da operação.

Este problema tem importância em casos reais de produção. Primeiramente, como apresentado, o *flowshop* flexível é um ambiente de produção comum em vários tipos de sistemas de manufatura, como, por exemplo, na produção de placas de circuito impresso (JIN *et al.*, 2002 e LEE *et al.*, 2003). Segundo, porque a minimização do atraso total é atualmente um objetivo de importância estratégica cada vez maior. Davis e Kanet (1993) salientam as conseqüências que o atraso de pedidos pode apresentar como perda da reputação em relação ao cliente, custos de perda de vendas e até penalidade financeira por contrato de venda. Sen e Gupta (2004) ressaltam as mesmas conseqüências e reforçam que a perda de reputação pode não somente resultar em perda do cliente, mas também no afastamento de novos clientes. Deste modo a resolução deste problema através da heurística proposta pode melhorar a satisfação dos clientes, o que atualmente é cada vez mais relevante.

Sendo C_j o tempo no qual uma tarefa é concluída no último estágio, então o atraso de uma tarefa é definido como sendo:

$$T_j = \max \{0, C_j - d_j\}$$

E assim, calcula-se o atraso total: $\sum_{j=1}^n T_j$.

1.3. Formulação matemática

Para formalizar o problema, a seguir é apresentada a formulação matemática linear inteira mista adaptada de Guinet *et al.* (1996) e Yang (1998). Guinet *et al.* (1996) apresentam a formulação para o mesmo ambiente, porém considerando a minimização do *makespan*. Já Yang (1998) analisa o mesmo ambiente, no entanto considera diferentes instantes de liberação das tarefas.

Índices:

j, k : índice para tarefas $(0, \dots, n+1)$;

l : índice para estágio $(1, \dots, s)$;

i : índice para máquina $(1, \dots, m_l)$.

Parâmetros:

n : número de tarefas;

s : número de estágios;

m_l : número de máquinas do estágio l ;

p_{jl} : tempo de processamento da tarefa j no estágio l ;

d_j : data de entrega da tarefa j ;

Variáveis:

x_{jkil} : assume o valor 1 se a tarefa k é designada após a tarefa j na máquina i do estágio l , caso contrário, assume valor zero;

C_{jl} : instante de conclusão da tarefa j ao término do processamento no estágio l ;

T_j : atraso da tarefa j .

Formulação:

$$\text{Minimizar } \sum_{j=1}^n T_j \quad (1)$$

Sujeito a :

$$\sum_{i=1}^{m_l} \sum_{j=0}^n x_{jkil} = 1 \quad l = 1, \dots, s; \quad k = 1, \dots, n. \quad (2)$$

$$\sum_{k=1}^{n+1} x_{jkil} \leq 1 \quad l = 1, \dots, s; \quad j = 0, \dots, n; \quad i = 1, \dots, m_l. \quad (3)$$

$$\sum_{k=1}^{n+1} x_{jkil} - \sum_{k=0}^n x_{kjil} = 0 \quad l = 1, \dots, s; \quad j = 1, \dots, n; \quad i = 1, \dots, m_l. \quad (4)$$

$$T_j \geq C_{js} - d_j \quad j = 1, \dots, n. \quad (5)$$

$$T_j \geq 0 \quad j = 1, \dots, n. \quad (6)$$

$$C_{kl} - C_{jl} \geq p_{kl} \sum_{i=1}^{m_l} x_{jkil} + M \sum_{i=1}^{m_l} (x_{jkil} - 1) \quad l = 1, \dots, s; \quad j, k = 1, \dots, n; \quad j \neq k. \quad (7)$$

$$C_{j,l+1} - C_{jl} \geq p_{j,l+1} \quad j = 1, \dots, n; \quad l = 1, \dots, s-1. \quad (8)$$

$$C_{j1} \geq p_{j1} \quad j = 1, \dots, n. \quad (9)$$

$$x_{jjil} = 0 \quad j = 0, \dots, n; \quad i = 1, \dots, m_l; \quad l = 1, \dots, s. \quad (10)$$

$$x_{jkil} \in \{0, 1\} \quad j = 0, \dots, n; \quad k = 1, \dots, n+1; \quad i = 1, \dots, m_l; \quad l = 1, \dots, s. \quad (11)$$

Considere que a variável x_{0kil} assume o valor 1 se a tarefa k está programada como a primeira tarefa na máquina i do estágio l , e $x_{j(n+1)il}$ assume o valor 1, se a tarefa j está programada como a última tarefa na máquina i do estágio l . Ou seja, as tarefas $n+1$ e zero são tarefas fictícias e representam, respectivamente, a tarefa posterior a última e a tarefa anterior a primeira.

O instante de conclusão da tarefa j no sistema é igual ao instante de conclusão da tarefa j no último estágio ($C_j = C_{js}$). A constante $\mathbf{M}$ é escolhida como um escalar maior que o maior instante de término de qualquer solução factível.

A equação (1) representa a minimização da soma dos atrasos das tarefas. O conjunto de restrições (2) garante que apenas uma tarefa j preceda uma tarefa k em alguma das máquinas do estágio l . As restrições (3) garantem que no máximo uma tarefa k suceda uma tarefa j em uma certa máquina i do estágio l . As restrições (4) garantem que cada tarefa tenha uma tarefa antecessora e uma sucessora na máquina i do estágio l na qual ela seja programada. Os conjuntos de restrições (5) e (6) garantem que, para cada tarefa j , o atraso seja $\max\{0, C_{js} - d_j\}$. As restrições (7) garantem que se uma tarefa j e outra k estão designadas para a mesma máquina de um certo estágio, então a tarefa j deve ser processada antes ou depois da tarefa k . Caso contrário, não existe restrição de precedência entre as duas tarefas. As restrições (8) garantem a precedência de uma tarefa entre estágios, ou seja, uma tarefa não pode ser processada no estágio $l+1$ antes de ter sido processada no estágio l . As restrições (9) garantem que o instante de conclusão de uma tarefa no primeiro estágio seja maior ou igual ao seu tempo de processamento. As restrições (10) garantem que uma tarefa não seja precedida por ela mesma e as restrições (11) garantem que as variáveis sejam binárias.

1.4. Revisão bibliográfica

A literatura sobre *scheduling* tem dado bastante atenção aos ambientes de *job shop* e *flowshop* em comparação com o *flowshop* flexível. Ao mesmo tempo, conforme Linn e Zhang (1999) e Quadrt e Kuhn (2007), a maioria dos autores trata o problema de *flowshop* flexível com medidas de desempenho relacionadas à passagem das tarefas pelo sistema, tais como minimização do *makespan* ou do tempo médio de fluxo. A minimização do atraso total é pouco freqüente e mesmo assim fica restrita a problemas de *flowshop*.

A seguir, são descritos estudos sobre *flowshop* flexível com atraso total ou atraso médio como medida de desempenho, visto que ambos são equivalentes, visando mostrar a oportunidade de pesquisa identificada. Primeiramente serão mostrados trabalhos que também consideram o *flowshop* flexível, porém com algumas características que os diferenciam do ambiente a ser analisado neste trabalho (*e.g.*, diferentes instantes de liberação das tarefas, lotes reentrantes, etc.)

Choi *et al.* (2005) consideram o *flowshop* flexível com produtos reentrantes no sistema já que alguns produtos podem ser processados mais de uma vez em um estágio. Um pedido de um produto pode ser dividido em dois ou mais lotes independentes, isso acontece quando o número de produtos associados a eles é suficientemente grande. A medida de desempenho é o atraso total, os autores propõem duas heurísticas considerando os lotes e duas considerando produtos. Estas quatro heurísticas utilizam o método de programação por lista de prioridades: quando uma máquina fica disponível para processar uma tarefa, seleciona-se a tarefa disponível com maior prioridade para ser processada naquela máquina. A primeira heurística considerando lotes utiliza o procedimento de inserção da heurística NEH proposta por Nawaz *et al.* (1983) para a minimização do *makespan* no ambiente *flowshop*. Sendo a data de entrega de um lote igual à data de entrega do pedido a que ele pertence, inicialmente os lotes são ordenados conforme a regra EDD (*Earliest Due Date*) e então são inseridos em todas as posições possíveis da sequência parcial construída previamente. A próxima iteração utiliza sequência construída com menor atraso total. O atraso dos produtos é calculado com base apenas nos lotes que já estão na sequência. Ao final deste procedimento é fornecida a programação das tarefas.

A segunda heurística é similar a primeira, no entanto, o método para avaliação de uma sequência é diferente: a cada iteração, os lotes que já foram sequenciados são programados, assim como na primeira heurística. A diferença é que, depois de feita esta programação, os lotes que ainda não foram sequenciados são programados pelo método da lista de prioridades, com a prioridade definida de acordo com a regra EDD. Isto é feito para que o atraso total, utilizado nas avaliações das sequências, sempre considere todos os lotes e não apenas os lotes já sequenciados.

Os dois algoritmos que consideram produtos são adaptações das duas últimas heurísticas. Nestes algoritmos atribui-se prioridade aos produtos e então os lotes recebem a mesma prioridade do respectivo produto. Nos testes computacionais são considerados problemas com até 20 tarefas e 8 estágios. Os resultados das quatro heurísticas são comparados com oito regras de despacho. As heurísticas com base em produtos apresentam melhor desempenho em relação às heurísticas com base em lotes, assim como em relação às regras de despacho.

Yang *et al.* (2000) consideram um *flowshop* flexível no qual as tarefas são todas conhecidas no instante zero e têm diferentes tempos de liberação, datas de entrega, tempos de processamento e pesos. O objetivo é a minimização do atraso total ponderado pelos pesos previamente conhecidos. Estes pesos podem indicar prioridades entre as tarefas. No artigo são apresentadas três heurísticas. A primeira é uma heurística de decomposição que pode ser dividida em quatro passos:

1. Detecção de gargalos: calcula-se a carga de trabalho (W_l) por máquina para todos os estágios. É considerado como gargalo o estágio ainda não programado com maior carga de trabalho;
2. Formulação do subproblema: o estágio considerado no passo anterior é visto como um problema de máquinas em paralelo no qual as tarefas chegam em diferentes instantes de tempo, para isto estes instantes são estimados, assim como as datas de entrega;
3. Solução do subproblema: através da regra de despacho Custo do Atraso Aparente (ATC – regra que faz uma estimativa do atraso usando uma função exponencial da folga que a tarefa tem para ser concluída antes de sua data de entrega), a programação

do subproblema (formulado no passo anterior) é feita através do método da lista de prioridades. Com isto é gerada uma solução inicial para o subproblema de máquinas em paralelo que passará por uma busca na vizinhança que procura uma realocação das tarefas que tem alto atraso, da máquina mais carregada do estágio para a menos carregada. A alocação da nova tarefa é feita na posição que resulta em menor acréscimo no atraso e em seguida são feitas trocas entre duas tarefas verificando se elas resultam em uma melhora na função objetivo. O critério de parada é um número máximo de iterações;

4. Reotimização: toda vez que um novo estágio é programado, todos os outros estágios, que já foram programados, têm seus instantes de liberação e datas de entrega estimados novamente e são reprogramados utilizando a heurística apresentada no passo 3. Ao fim desta etapa, volte para o passo 1 até que todos os estágios estejam programados.

A segunda heurística é uma heurística de busca local. Cada iteração da busca local consiste em duas fases. A primeira fase utiliza o mesmo método usado na solução de subproblemas da primeira heurística proposta, alterando a carga de trabalho e gerando novas seqüências para as máquinas. A segunda fase tenta procurar por melhores seqüências para cada máquina em cada estágio através do método *large step optimization (LSO)*. O método consiste em dar passos largos, para procurar por novas regiões de soluções, e passos curtos para intensificar a busca na região. O critério de parada desta segunda heurística é o tempo. Em sua versão curta, o tempo é ajustado como o mesmo tempo da heurística de decomposição. Já em sua versão longa, o limite é cinco vezes o mesmo tempo.

E por último é proposta uma heurística que é uma mistura das duas primeiras: a solução final da heurística de decomposição é usada como solução inicial para a heurística de busca local. O seu critério de parada é o mesmo da versão longa da heurística de busca local. O trabalho é concluído com uma análise comparativa das três heurísticas e como esperado a terceira heurística teve melhor desempenho que as duas primeiras, mostrando a importância de uma boa solução inicial para a segunda heurística. São testados problemas com até 50 tarefas, 7 estágios e 4 máquinas em paralelo em cada estágio e os resultados comparados com a regra de despacho

Custo do Atraso Aparente (ATC). Os resultados mostram que os valores das soluções fornecidas por esta regra ficaram bem acima dos melhores resultados encontrados, 291,6% em média.

Outros trabalhos consideram situações particulares do *flowshop* flexível. O trabalho de Barman (1997) e Jayamohan e Rajendran (2000) consideram o desempenho de uma série de regras de despacho em um *flowshop* flexível com três estágios, duas máquinas em cada estágio e as tarefas têm diferentes tempos de liberação. Barman considera três medidas de desempenho enquanto Jayamohan e Rajendran consideraram sete medidas de desempenho diferentes, em ambos os trabalhos o atraso médio está entre as medidas de desempenho consideradas. Barman conclui que os melhores resultados para o atraso médio são obtidos quando diferentes regras de despacho são consideradas para os três estágios (SPT, EDD e FIQ - *First in queue rule*). Jayamohan e Rajendran comparam seus resultados com o de Barman e conseguem um melhor desempenho com uma única regra de despacho para os três estágios, esta regra considera tempo de processamento, instante de liberação e a soma dos tempos de processamento de todas as tarefas na fila do próximo estágio.

Lee *et al.* (2003) consideram um *flowshop* flexível com oito estágios em uma fábrica de placas de circuito impresso. Os autores desenvolveram um método no qual lotes e programação são determinados simultaneamente por meio de uma regra de despacho. Estas regras de despachos foram desenvolvidas através de uma combinação de regras de prioridade clássicas (MDD e SLACK) com modelos de tamanho de lote (*e.g.* Quantidade econômica de produção - EPQ). Os resultados são comparados com um método frequentemente usado neste tipo de fabricação: o método com duas fases no qual na primeira fase são montados os lotes e na segunda é feita a programação com base no tamanho dos lotes determinados na primeira fase.

Um dos primeiros trabalhos que considera o ambiente abordado neste trabalho é o artigo de Brah (1996). O autor analisa o desempenho de dez regras de prioridade em termos de atraso médio e máximo atraso. O autor também examina os efeitos de características do problema, como, por exemplo, número de tarefas, número de estágios, número de máquinas em paralelo em cada estágio, considerando no máximo 50 tarefas, 20 estágios e 10 máquinas em paralelo. O autor conclui que as características do problema têm um grande impacto em ambas as medidas de desempenho. O artigo mostra que o atraso decresce linearmente com o número de estágios. No

entanto, o atraso cresce linearmente com o número de tarefas. Existe uma relação inversa entre número de máquinas e o atraso: um aumento no número de máquinas diminui o atraso e vice-versa. Já o impacto de uma regra de despacho é relativamente pequeno se comparado com o impacto das características do problema, mas mesmo assim, o autor chega a algumas conclusões em relação às regras de despacho. A mais importante delas é que regras relativamente simples como Data de Entrega mais cedo (EDD) e Data de Entrega Modificada (MDD) apresentam os melhores desempenhos para máximo atraso e atraso médio, respectivamente. As demais regras apresentaram desempenhos alternantes.

O primeiro trabalho que procura encontrar uma solução ótima para o problema considerado é apresentado por Lee e Kim (2004). Os autores propõem um algoritmo *branch-and-bound* para o qual foram desenvolvidos dois limitantes inferiores. O ambiente considerado possui dois estágios com apenas uma máquina no primeiro estágio e até cinco máquinas no segundo. Os resultados mostram que o algoritmo acha a solução exata para problemas com até 15 tarefas em um tempo aceitável, variando entre 337,7s e 2.368,7s.

Lee *et al.* (2004) propõem uma heurística visando a minimização do atraso total com foco no estágio com maior carga de trabalho. O primeiro passo é identificar o estágio gargalo, em seguida deve-se estimar os instantes de liberação no gargalo. Para esta estimativa, os autores utilizam três métodos denominados por RT1 (emprega um limitante inferior e é usada apenas na primeira estimativa), RT2 (método iterativo usado nos casos de inviabilidade) e RT3 (usado para melhorar uma solução). Posteriormente o estágio gargalo é programado como um problema de máquinas em paralelo (com diferentes instantes de liberação) por meio de uma heurística baseada nos métodos propostos por Panwalkar *et al.* (1993), conhecida como PSK, e por Koulamas (1994), conhecida como KPM. Em seguida, são determinadas as programações para os outros estágios, com base na programação do gargalo. Para os estágios anteriores ao gargalo, a programação é feita em ordem inversa, ou seja, do estágio imediatamente anterior ao gargalo até o primeiro estágio. Se esta programação for inviável com a do gargalo, aplica-se o método RT2 e o algoritmo volta para o passo da programação do estágio gargalo até que se encontre uma programação viável. Caso contrário, os estágios posteriores são programados através do método de programação por lista de prioridade. O procedimento é iterativo, ou seja, depois de encontrada

uma solução viável, o método RT3 é aplicado sucessivas vezes até que se encontre uma solução pior que a melhor encontrada ou até alcançar um máximo de iterações.

Nos experimentos computacionais são considerados três níveis para a quantidade de tarefas e para o número de estágios (30, 50 e 100 tarefas e 5, 10 e 30 estágios) e quatro níveis para a posição do estágio gargalo no sistema (primeiro, segundo, terceiro e quarto quartos). Os resultados são comparados com três métodos distintos: programação pelo método da lista de prioridades (do primeiro para o último estágio - *forward*), programação em ordem inversa (*backward*) e uma programação que mistura as duas primeiras (*multipass*). Cada um dos métodos é avaliado em conjunto com oito regras de despacho. O autor conclui mostrando que a heurística com foco no estágio gargalo apresenta os melhores resultados independentemente da regra de despacho considerada e, independentemente do método de programação, as regras de despacho ATC e COVERT (*Cost over time* – regra que combina SPT com uma estimativa do atraso) apresentam os melhores resultados. Dentre as outras seis regras de despacho que não dependem de parâmetros de ajuste, a regra MDD apresentou os melhores resultados através dos quatro métodos de programação.

Influenciado principalmente pelo trabalho de Lee *et al.* (2004), a proposta deste trabalho é contribuir com o estudo deste problema através da elaboração de novas heurísticas construtivas para o problema e de um novo método de programação em ordem inversa.

No próximo capítulo, é apresentada a heurística construtiva proposta para o problema considerado.

2. HEURÍSTICA CONSTRUTIVA

Neste capítulo serão apresentados dois métodos heurísticos de programação para minimização do atraso total no ambiente *flowshop* flexível. O primeiro método programa as tarefas na ordem de passagem das mesmas pelo sistema, o segundo considera a ordem inversa. Juntamente com estes métodos são apresentadas regras de despachos conhecidas e outras são desenvolvidas para o ambiente considerado.

2.1. Método de programação por lista de prioridade

Em muitos ambientes de produção, especialmente em sistemas complexos como o considerado neste trabalho, o método de programação por lista de prioridade (*forward list scheduling*) é amplamente utilizado para operações de programação. Neste algoritmo, sempre que uma máquina de certo estágio torna-se disponível, seleciona-se a tarefa de maior prioridade que pode ser processada na máquina. Esta prioridade é determinada por uma regra de despacho. Existem diversas regras de despacho conhecidas na literatura para problemas de programação de tarefas. Neste trabalho serão consideradas cinco regras. Duas delas por serem simples e por apresentarem bom desempenho (veja LEE *et al.*, 2004): SPT, EDD. A primeira atribui maior prioridade para as tarefas com menor tempo de processamento no estágio considerado. A segunda prioriza as tarefas com menor data de entrega.

A terceira regra não é tão simples quanto as duas primeiras, porém de acordo com Brah (1996) e Lee *et al.* (2004), dentre as regras que não possuem parâmetros a serem ajustados, a regra MDD apresenta os melhores resultados para a minimização do atraso total no ambiente *flowshop* flexível. A regra MDD foi originalmente proposta por Baker e Bertrand (1982) para o problema de uma única máquina e foi adaptada para o *flowshop* flexível da seguinte maneira. Seja t o tempo no qual a regra é aplicada para uma tarefa j num certo estágio l , MDD é o máximo entre a data de entrega e a soma dos tempos de processamentos da tarefa nos $s-l+1$ estágios restantes, ou seja, $\max\left(d_j, t + \sum_{u=l}^s p_{ju}\right)$. Atribui-se maior prioridade para tarefas com menor MDD.

Dado o bom desempenho da regra MDD para o ambiente considerado, veja Brah (1996) e Lee *et al.* (2004), as duas regras desenvolvidas neste trabalho estão relacionadas com ela.

A quarta regra é uma nova adaptação da regra MDD para o *flowshop* flexível e, portanto, será denominada por MDDa. Tendo em vista que o *flowshop* flexível pode ser visto como uma série de problemas de máquinas em paralelo, é promissor desenvolver uma regra que considere esta propriedade deste ambiente de máquinas. Para isso é necessário definir a data de entrega para cada um dos problemas de máquinas em paralelo. Assim sendo, a data de entrega da tarefa j no estágio l (d_{jl}) é definida como a data de entrega real menos o trabalho remanescente, ou seja:

$$d_{jl} = d_j - \sum_{u=l+1}^s p_{ju}. \quad (2.1)$$

Por fim, define-se a regra MDDa como: $MDDa(j, l, t) = \max(d_{jl}, t + p_{jl})$, sendo j a tarefa considerada, l o estágio, t o instante de aplicação da regra.

A escolha da última regra considerada se deve ao fato dela apresentar relação com as regras MDD e MDDa. A regra PRTT, proposta por Chu e Portmann (1992), foi originalmente proposta para minimizar o atraso total em um ambiente com uma única máquina e tarefas que possuem diferentes instantes de liberação para iniciar os seus processamentos. Na regra PRTT as tarefas que estarão disponíveis em instantes posteriores ao atual também são consideradas. Dado que podemos analisar o ambiente *flowshop* flexível como uma série de problemas isolados de máquinas em paralelo, nos quais as tarefas ficam disponíveis em diferentes instantes de tempo, o estudo de funções de prioridades que consideram estados futuros do sistema é relevante. Assim sendo, a adaptação da regra PRTT também considera o *flowshop* flexível como uma série de máquinas em paralelo, assim como a regra MDDa. Portanto, ela utiliza a mesma definição de data de entrega da tarefa j no estágio l (d_{jl}). A regra PRTT adaptada para o *flowshop* flexível (PRTTa) foi definida como:

$PRTTa(j, l, t) = \alpha \cdot \max(r_{jl}, t) + \max[\max(r_{jl}, t) + p_{jl}, d_{jl}]$, sendo j a tarefa considerada, l o estágio, t o instante de aplicação da regra, r_{jl} o instante de liberação da tarefas j no estágio l e α um parâmetro constante que pondera a prioridade do primeiro termo da regra.

O primeiro termo da soma, $\alpha \max(r_{jl}, t)$, procura penalizar tarefas que não estão disponíveis no instante t ($r_{jl} > t$), ou seja, estas tarefas receberão um valor maior para este termo do que tarefas disponíveis para programação ($r_{jl} \leq t$). O segundo termo procura simultaneamente considerar o princípio de priorização da regra MDDa e penalizar as tarefas não disponíveis. Note que se $\max(r_{jl}, t) = t$, o segundo termo torna-se idêntico a regra MDDa. Esta característica é valiosa para a resolução do problema considerado, visto que a regra MDD original apresenta bons resultados para a minimização do atraso total no ambiente *flowshop* flexível.

No método de programação por lista de prioridades, em todo instante t , no qual existe uma máquina disponível para processar uma tarefa, programa-se a tarefa disponível com maior prioridade para ser processada naquela máquina, sendo esta prioridade dada através da regra de despacho considerada. A regra proposta (PRTTa) exige uma adaptação no método de programação por lista de prioridades tradicional. Nesta regra, quando tarefas que não estão disponíveis no instante t são programadas, ocorre a inclusão de tempo ocioso. Um procedimento para o preenchimento deste tempo ocioso é proposto a seguir.

• **Método de programação por lista de prioridades - PRTTa:** em todo instante t no qual existe uma máquina disponível, programe a tarefa j ainda não programada com menor PRTTa, desempate pelo menor instante de conclusão na máquina considerada. Caso a tarefa j a ser programada ainda não esteja disponível no instante t , preencha o tempo ocioso gerado com tarefas que possam ser programadas antes do início da tarefa j , sem alterar o instante de início de processamento da tarefa j . Se existirem tais tarefas, deve-se programar a tarefa com menor instante de liberação (r_{jl}), desempate pelo menor PRTTa, até que não existam mais tarefas que possam ser programadas no tempo ocioso.

A Figura 4 apresenta um exemplo de uma programação gerada através do método proposto. Este exemplo possui quatro tarefas e dois estágios com duas máquinas no primeiro e uma máquina no segundo estágio. As datas de entrega das tarefas 1, 2, 3 e 4 são 4, 14, 1 e 15, respectivamente. A programação no estágio 1 é feita como a programação por lista de prioridades original, visto que todas as tarefas estão disponíveis no instante inicial. A programação no instante $t=2$ do estágio 2 ilustra a diferença entre os dois métodos de programação. Neste instante

apenas a tarefa 2 está disponível, no entanto, decidiu-se programar a tarefa 1 no instante de sua disponibilidade ($t=3$). Esta programação foi realizada, pois a tarefa 1 apresenta um menor valor para a regra PRTTa no instante $t=2$ do estágio 2, sendo $\alpha=1$. Com isso, foi inserido um tempo ocioso de 2 até 3, portanto deve-se procurar por tarefas que possam ser programadas antes da tarefa 1. A tarefa 2 é a única tarefa ainda não programada que apresenta disponibilidade durante este período de tempo, no entanto ela não é programada, pois seu tempo de processamento é maior que o tempo ocioso. Como não foram encontradas tarefas que possam preencher este período de tempo, o método de programação segue para o próximo instante de disponibilização de máquina no estágio ($t=6$). As demais tarefas foram programadas seguindo o mesmo procedimento.

Figura 4 - Programação gerada pelo método da lista de prioridades (PRTTa).

O atraso total da programação ilustrada na Figura 4 é igual a 20. Através da comparação com o valor ótimo da soma dos atrasos para este exemplo (obtida através da resolução da formulação apresentada nos softwares Xpress e What's Best) verificamos que a heurística proposta encontrou a solução ótima.

Este mesmo exemplo foi resolvido através de três regras de despacho: SPT, EDD, MDD e MDDa. O atraso total foi, respectivamente, 24, 22, 24, 23, resultados superiores ao valor encontrado pela heurística proposta, ilustrando que a inserção de tempo ocioso pode ser importante para a qualidade da solução encontrada. Conforme Kanet e Sridharan (2000) quando existem tarefas com diferentes instantes de liberação, como no problema considerado, a inserção de tempo ocioso na programação pode ser vantajosa.

É importante ressaltar que, através do método de programação por lista de prioridades tradicional, não é possível chegar à programação da Figura 4, pois qualquer regra através do método tradicional programaria a tarefa 2 no instante $t=2$ do estágio 2

O método de programação apresentado nesta seção considera a programação na sequência de passagem das tarefas pelo sistema, ou seja, do primeiro para o último estágio. Na próxima seção será apresentado um método inverso de programação, que programa do último para o primeiro estágio.

2.2. Método de programação em ordem inversa

Em geral a operação de programar tarefas é feita de maneira direta, ou seja, na sequência na qual tarefas passam pelo sistema. No entanto, existe um método alternativo no qual as tarefas são programadas em ordem inversa, ou seja, do último para o primeiro estágio em um horizonte de tempo invertido.

Este método é conhecido como método inverso de programação (*backward list scheduling*). Para obter uma programação através do método inverso, primeiro deve-se construir um problema invertido a partir do problema original, no qual as tarefas têm que ser processadas na ordem inversa. Uma vez que o problema invertido está construído, deve-se resolver o problema utilizando o método de programação por lista de prioridade. Depois que o problema invertido está resolvido, a programação resultante deve ser invertida de volta para o horizonte de tempo real, para que possamos obter uma programação para o problema original. Quando a programação é invertida, as operações são deslocadas para esquerda (tempos de início e conclusão das tarefas são deslocados para o instante de tempo mais cedo possível) no primeiro estágio. A programação dos estágios restantes é obtida com uma última aplicação do método de programação por lista de prioridade.

Quando o problema inverso é construído, não se deve apenas inverter a relação de precedência entre os estágios, mas também redefinir as datas de entrega e os instantes de liberação das tarefas. Quando o problema é visto através de um horizonte de tempo invertido, o instante de conclusão no problema original está relacionado com o instante de liberação no problema inverso. Kim (1995) apresenta vários métodos para estas redefinições. Neste trabalho consideramos as mesmas redefinições utilizadas por Lee *et al.* (2004), tendo em vista que os autores testaram várias redefinições e estas apresentaram os melhores resultados para a minimização do atraso total no ambiente *flowshop* flexível. Nas redefinições escolhidas, os instantes de liberação (r_j^*) e as datas de entrega (d_j^*) do problema inverso são definidos como:

$$r_j^* = \max \{C_{\max}, d_{\max}\} - \max \{C_j, d_j\} \quad (2.2)$$

$$d_j^* = d_{\max} \quad (2.3)$$

Sendo d_j , d_{max} , C_j e C_{max} , a data de entrega da tarefa j , o maior valor de data de entrega dentre todas as tarefas, o instante de conclusão da tarefa j no último estágio e o maior tempo de conclusão de todas as tarefas, respectivamente. C_j e C_{max} são obtidos através do método de programação por lista de prioridades no problema original.

A interpretação da formulação do problema inverso é de fundamental importância para entender o bom funcionamento deste método. A idéia geral do instante de liberação (r_j^*) é fazer com que tarefas atrasadas comecem mais cedo no problema original, o que corresponde a começar mais tarde no problema inverso. Do mesmo jeito, r_j^* procura fazer com que tarefas adiantadas comecem mais tarde no problema original. Para isto, o primeiro termo de r_j^* é definido de tal modo que ele sempre seja maior ou igual que qualquer valor que o segundo termo possa assumir, ou seja, o termo $\max\{C_{max}, d_{max}\}$ garante que $r_j^* \geq 0, \forall j$. Já o segundo termo baseia-se no princípio de que o instante de conclusão de uma tarefa no problema original está relacionado com o instante de liberação no problema inverso. Tendo isto em vista, o segundo termo poderia ser apenas função do instante de conclusão (C_j). No entanto, r_j^* também considera a data de entrega da tarefa j no problema original (d_j), pois se uma tarefa terminou antes da sua data de entrega ($\max\{C_j, d_j\}=d_j$) significa que ela está adiantada e pode começar mais tarde no problema original (ou mais cedo no problema invertido). Já caso ($\max\{C_j, d_j\}=C_j$) significa que a tarefa está atrasada e, portanto, deve começar mais cedo no problema original (ou mais tarde no problema invertido).

Por exemplo, sejam duas tarefas com mesmo C_j ($C_1=C_2=10$) e datas de entrega diferentes ($d_1=8$ e $d_2=12$), portanto, $\max\{C_{max}, d_{max}\}=12$, a tarefa 1 está atrasada (já que $C_1 > d_1$) e a tarefa 2 adiantada ($C_2 < d_2$). Calculando os instantes de liberação (r_j^*) têm-se: $r_1^* = 12 - \max\{10, 8\} = 2$ e $r_2^* = 12 - \max\{10, 12\} = 0$. Portanto, uma vez que $r_1^* > r_2^*$, a tarefa 1 vai começar mais tarde no problema invertido e, conseqüentemente, mais cedo no problema original, o que é interessante visto que ela é uma tarefa atrasada. Já tarefa 2 (adiantada) vai começar mais cedo no problema invertido e, conseqüentemente, mais tarde no problema original, o que também é interessante.

De acordo com Kim (1995), a data de entrega no problema inverso está relacionada com o instante de liberação do problema original. Tendo em vista esta afirmação, as datas de entrega de

todas as tarefas têm que ser iguais, já que os instantes de liberação são todos iguais a zero no problema original. Deste modo define-se, $d_j^* = d_{max}, \forall j$.

O algoritmo para o método inverso de programação é resumido a seguir.

• **Método de programação em ordem inversa**

Passo 0. Faça o método de programação por lista de prioridade para o problema original e obtenha uma programação inicial.

Passo 1. Construa o problema inverso através da inversão da relação de precedência entre os estágios e da redefinição das datas de entrega e dos tempos de liberação usando a informação obtida com a programação inicial do Passo 0.

Passo 2. Programe as tarefas através da regra de maior C_{jl} (tarefas com maior C_{jl} tem maior prioridade) considerando os C_{jl} obtidos através da programação do Passo 0.

Passo 3. Inverta a seqüência das tarefas em cada máquina do primeiro estágio.

Passo 4. Para o primeiro estágio, ache uma programação através do deslocamento das tarefas para a esquerda que não apresente tempo ocioso e que não viole as relações de precedências e seqüências obtidas no Passo 3.

Passo 5. Para os estágios restantes, obtenha a programação utilizando o método de programação por lista de prioridade respeitando os instantes de liberação do primeiro estágio.

Passo 6. Para obter a programação final faça: respeitando a ordem de programação atual, realoque as tarefas às máquinas do primeiro estágio, verificando se elas podem começar mais cedo. Retire os possíveis tempos ociosos gerados nos estágios subseqüentes.

A seguir é apresentado um exemplo para um melhor entendimento do método de programação em ordem inversa. Para fins didáticos o exemplo é uma instância com pequenas dimensões: seis tarefas e três estágios, com duas máquinas nos dois primeiros estágios e uma no terceiro. A Tabela 1 mostra os parâmetros da instância, ou seja, as datas de entrega (d_j) e os

tempos de processamentos (p_{jl}) para as tarefas j no estágio l . A regra de despacho escolhida para este exemplo foi a regra EDD (menor data de entrega), portanto a ordem de priorização é $\{5,3,2,4,1,6\}$.

Tabela 1 - Parâmetros para o exemplo do método de programação em ordem inversa.

tarefas j	d_j	p_{jl}		
		$l=1$	$l=2$	$l=3$
1	14	2	5	4
2	11	1	5	3
3	10	3	4	1
4	11	1	3	5
5	7	2	3	1
6	18	4	5	5

A programação resultante do Passo 0 através do método de programação por lista de prioridades com a regra EDD é mostrada de duas maneiras: em um gráfico de Gantt (Gráfico 1) e em uma tabela (Tabela 2). Na Tabela 2 são mostrados os instantes de programação, ou seja, o instante no qual a tarefa começou a ser processada no estágio, os instantes de conclusão (C_{jl}) e o respectivo atraso (T_j) de cada tarefa, assim como o atraso total ($\sum T_j$). Pode-se notar que as primeiras tarefas escolhidas para a programação no primeiro estágio são as tarefas 5 e 3 por apresentarem as menores datas de entrega, $d_5=7$ e $d_3=10$. A ordem priorização de menor data de entrega é mantida para todo o sistema e obteve-se a programação através do método de programação por lista de prioridade do Passo 0.

Gráfico 1 - Gráfico de Gantt para o método de programação por lista de prioridade.

Tabela 2 - Programação obtida através do método de programação por lista de prioridade.

tarefas j	d_j	p_{jl}			Instante da programação			C_{jl}			T_j
		$l=1$	$l=2$	$l=3$	$l=1$	$l=2$	$l=3$	$l=1$	$l=2$	$l=3$	
1	14	2	5	4	3	10	18	5	15	22	8
2	11	1	5	3	2	5	10	3	10	13	2
3	10	3	4	1	0	3	7	3	7	8	0
4	11	1	3	5	3	7	13	4	10	18	7
5	7	2	3	1	0	2	5	2	5	6	0
6	18	4	5	5	4	10	22	8	15	27	9
ΣT_j										26	

No Passo 1 deve-se construir o problema inverso com base nas informações da programação do Passo 0, ou seja, com base nos resultados expressos na Tabela 2 e utilizando-se as redefinições dos instantes de liberação (r_j^*) e das datas de entrega (d_j^*) através das fórmulas (2.2) e (2.3). A Tabela 3 mostra o problema inverso e seus parâmetros já calculados. Nesta mesma tabela é importante destacar a inversão da relação de precedência entre as tarefas, representada nos tempos de processamento que estão dispostos em ordem inversa, ou seja, do estágio $l=3$ ao

estágio $l=1$. As datas de entrega (d_j^*) são todas iguais a 18, visto que a maior data de entrega é $d_6=18$. Para a tarefa 1, o seu instante de liberação (r_1^*) no problema inverso foi calculado através da equação (3.1) da seguinte maneira: da Tabela 2 têm-se, $C_{max}=C_6=27$; $d_{max}=d_6=18$; $C_1=22$; $d_1=14$, portanto $r_1^* = \max\{27,18\} - \max\{22,14\} = 27 - 22 = 5$, e assim segue para as demais tarefas, obtendo-se os valores apresentados na Tabela 3.

Tabela 3 - Problema inverso formulado no Passo 1.

tarefas j	d_j^*	p_{jl}			r_j^*
		$l=3$	$l=2$	$l=1$	
1	18	4	5	2	5
2	18	3	5	1	14
3	18	1	4	3	17
4	18	5	3	1	9
5	18	1	3	2	20
6	18	5	5	4	0

No passo seguinte (Passo 2) deve-se fazer a programação do problema inverso. A regra de despacho para programação neste passo é a regra de maior instante de conclusão no estágio (C_{jl}) obtido na programação do Passo 0. Com isso, obteve-se a programação para o problema inverso, mostrada no Gráfico 2 e na Tabela 4. Como a tarefa 6 é a única tarefa disponível no instante zero, ela é a primeira a ser programada, em seguida foi programada a tarefa 1, pois era a única disponível no instante 5 e assim por diante. Através deste exemplo numérico é possível notar que devido ao fato das tarefas possuírem diferentes instantes de liberação, é menos freqüente o uso da regra de despacho para decidir qual tarefa que deve ser priorizada. Se em algum momento tivesse existido mais tarefas disponíveis do que máquinas livres o desempate seria feito pela regra de maior instante de conclusão (C_{jl}), os quais estão presentes na Tabela 2.

Gráfico 2 - Gráfico de Gantt para programação do problema inverso.

Tabela 4 - Programação do problema inverso.

tarefas j	Instante da programação			C_{jt}		
	3	2	1	3	2	1
1	5	9	14	9	14	16
2	14	17	22	17	22	23
3	17	18	22	18	22	25
4	9	14	17	14	17	18
5	20	22	25	21	25	27
6	0	5	10	5	10	14

No Passo 3, utilizando as informações da programação do passo anterior, deve-se inverter a sequência das tarefas nas máquinas do primeiro estágio, neste caso as máquinas 1A e 1B. No problema inverso, a sequência da programação das tarefas na máquina 1A é {6,1,4,3,5} portanto esta ordem é invertida pra {5,3,4,1,6} e será a ordem de programação final para esta máquina. A ordem de programação da máquina 1B continua mesma, pois ela processa apenas uma tarefa. No passo seguinte (Passo 4) deve-se deslocar as tarefas para a esquerda, respeitando as restrições de uma programação factível. Por exemplo, para a máquina 1A a primeira tarefa ($j=5$) começará a ser processada no instante zero, visto que no problema original todas as tarefas estão disponíveis

no instante zero. A próxima tarefa desta máquina ($j=3$) pode ser deslocada apenas até o instante 2, visto que este é o instante que a máquina é liberada pela tarefa 5. Este procedimento segue para as demais tarefas. Concluído este passo (Passo 4), obtém-se uma programação para o primeiro estágio. A programação para os estágios restantes (Passo 5) é feita através da mesma regra de despacho do passo 0, neste caso a regra EDD. A programação obtida é apresentada no Gráfico 3.

Gráfico 3 - Gráfico de Gantt para a programação (Passo 5)

A programação final é obtida através do Passo 6, no qual deve-se verificar quais tarefas do primeiro estágio podem começar mais cedo através da realocação às máquinas. Para isso deve-se começar pela tarefa com menor instante de programação (desde que maior que zero). No caso do exemplo, a tarefa 3. Esta tarefa pode começar mais cedo se for realocada à máquina 1B. A realocação é feita, a próxima tarefa a ser considerada é a tarefa 4. O procedimento continua até que não seja mais possível diminuir o instante de programação de nenhuma tarefa do primeiro estágio. Feito isso, algumas tarefas podem ser deslocadas para a esquerda nos estágios subsequentes. Se isso for possível, deve-se deslocá-las e a programação final é obtida como mostrada no Gráfico 4 e na Tabela 5.

Gráfico 4 - Gráfico de Gantt para a programação final

Tabela 5 - Programação final obtida através do método de programação em ordem inversa

tarefas j	d_j	p_{jl}			Instante da programação			C_{jl}			T_j
		1	2	3	1	2	3	1	2	3	
1	14	2	5	4	3	9	15	5	14	19	5
2	11	1	5	3	0	1	6	1	6	9	0
3	10	3	4	1	1	5	9	4	9	10	0
4	11	1	3	5	2	6	10	3	9	15	4
5	7	2	3	1	0	2	5	2	5	6	0
6	18	4	5	5	4	9	19	8	14	24	6
ΣT_j											15

Analisando os resultados obtidos através do método de programação por lista de prioridade (Tabela 2) e por meio do método de programação em ordem inversa (Tabela 5), pode-se perceber uma redução significativa no valor da função objetivo (ΣT_j) de 26 para 15 (veja Tabela 2), ilustrando a eficácia do segundo método em relação ao primeiro.

O próximo capítulo apresenta os testes preliminares e os resultados computacionais.

3. RESULTADOS COMPUTACIONAIS

A seguir é apresentada uma breve descrição de como foram geradas as instâncias, os testes preliminares, as comparações entre as regras, entre os métodos e com os métodos da literatura.

3.1. Instâncias teste

Para verificar o desempenho das heurísticas propostas, os resultados foram comparados entre si e com os resultados obtidos por Lee *et al.* (2004). As instâncias foram disponibilizadas pelos próprios autores. O número de máquinas por estágio foi gerado de acordo com uma distribuição uniforme discreta (entre 1 e 10). Os tempos de processamento também foram gerados de acordo com uma distribuição uniforme, mas de duas maneiras diferentes: entre $1 \cdot m_l$ e $50 \cdot m_l$ para o estágio gargalo e entre $1 \cdot m_l \cdot L$ e $50 \cdot m_l \cdot L$ para os demais estágios, sendo L o nível de diferença da carga de trabalho entre o estágio gargalo e os demais estágios. As datas de entrega foram geradas por uma distribuição uniforme dentro do intervalo: $P\left(1 - T - \frac{R}{2}\right) \leq d_j \leq P\left(1 - T + \frac{R}{2}\right)$, no qual T e R são fatores de atraso e amplitude, respectivamente, e P é o limitante inferior do *makespan* para o mesmo problema proposto por Santos *et al.* (1995).

Foram geradas 4.320 instâncias de teste, 10 problemas para cada combinação de três níveis do número de tarefas (30, 50, 100), três níveis do número de estágios (5, 10, 30), quatro níveis da posição do estágio gargalo no sistema (primeiro, segundo, terceiro e quarto quartos do sistema), três níveis de diferença L da carga de trabalho entre o estágio gargalo e os demais estágios (0,5, 0,7 e 0,9), dois níveis para o fator de atraso T (0,1 e 0,5) e dois níveis para a amplitude da data de entrega R (0,8 e 1,8).

O estudo de Lee *et al.* (2004) foi escolhido para comparação por ser o estudo mais atual que trata exatamente do mesmo problema: o *flowshop* flexível para minimização do atraso total.

O algoritmo foi desenvolvido em C++ e implementado em um AMD Athlon 64 X2 4200+ 2,36GHz com 2,5Gb de memória RAM.

3.2. Testes preliminares

Primeiramente foram realizados testes para selecionar um valor para o parâmetro α que pondera a importância do primeiro termo da regra PRTTa. Estes testes consistiram em variar α e registrar o resultado da média do atraso total das 4.320 instâncias para cada valor de α . O Gráfico 5 apresenta o resultado destes testes.

Gráfico 5 - Comportamento do atraso total em função de α .

Através da análise do Gráfico 5, pode-se perceber que a regra apresenta melhor desempenho para $\alpha = 12$. No entanto é importante observar que resultados similares são obtidos para $\alpha \geq 5$. Este comportamento mostra a robustez da metodologia no que se refere à escolha do parâmetro, ou seja, não são necessários grandes esforços para ajustá-lo.

Na próxima seção são apresentadas comparações entre as regras de despacho selecionadas associadas com os diferentes métodos de programação.

3.3. Método de programação por lista de prioridades

Devido às dificuldades de encontrar a solução ótima para os problemas considerados, foi usado o RDI (*Relative Deviation Index*) para medir o desempenho das diferentes regras consideradas. O RDI é calculado da seguinte maneira: $RDI = \frac{S_a - S_b}{S_w - S_b}$, no qual S_a é a solução do método avaliado, S_b e S_w são, respectivamente, a melhor e pior soluções encontradas. Quanto menor o valor do RDI, melhor o desempenho do método avaliado. Esta medida de desempenho também foi utilizada por Lee *et al.* (2004), Ronconi(2004) e Kim (1993). Neste trabalho a solução é o valor do atraso total calculado para a instância considerada.

A Tabela 6 apresenta a média do RDI obtida com o método de programação por lista de prioridades para as cinco regras consideradas (SPT, EDD, MDD, MDDa e PRTTa). A primeira coluna refere-se à regra SPT e a primeira linha refere-se a todas as instâncias que possuem 5 estágios e 30 tarefas (480 instâncias). Por exemplo, considerando a regra SPT, a média do RDI é de 0,81 para as 480 instâncias de 5 estágios e 30 tarefas. A última linha mostra a média do RDI calculada em relação às 4320 instâncias. Os valores destacados em cinza indicam os melhores resultados.

Confirmando os resultados da literatura, dentre as regras conhecidas e que não dependem de parâmetros ajustáveis, a regra MDD apresenta o menor valor de RDI (0,29) para a minimização do atraso total no ambiente *flowshop* flexível em oposição a 0,84 da regra SPT e 0,53 da regra EDD. A regra MDD também apresenta os menores valores de RDI quando considerado o número de estágios e o número de tarefas das instâncias, exceto pelas 480 instâncias com 30 estágios e 30 tarefas (MDD apresenta um RDI igual a 0,44 em oposição a 0,41 da regra EDD).

No entanto, as duas regras desenvolvidas neste trabalho (MDDa e PRTTa) superam as três regras conhecidas consideradas. As regras propostas superam amplamente as demais regras na média total do RDI, 0,06 (MDDa) e 0,04 (PRTTa). Ambas também superam em todas as médias que consideram número de estágios e o número de tarefas das instâncias. Considerando o número de melhores soluções, a regra PRTTa também supera as demais com 2.973 instâncias nas quais

esta regra atingiu a melhor solução, em contraposição a 1.040 da regra MDDa, 451 da regra MDD, 240 da EDD e 3 da SPT.

Comparando entre si apenas as duas regras desenvolvidas, percebe-se um melhor desempenho da regra PRTTa em termos de RDI total e em termos das médias que consideram número de estágio e número de tarefas. Em alguns casos a regra MDDa aproxima-se e até mesmo empata com a regra PRTTa, mas em outros casos a regra que possui visão do futuro supera bastante a regra MDDa.

Tabela 6 - Média do RDI para o método de programação por lista de prioridades.

Estágios	Tarefas	SPT	EDD	MDD	MDDa	PRTTa
5	30	0,81	0,51	0,31	0,08	0,06
	50	0,78	0,55	0,19	0,07	0,07
	100	0,88	0,44	0,08	0,03	0,03
10	30	0,91	0,45	0,39	0,06	0,02
	50	0,77	0,59	0,31	0,07	0,05
	100	0,67	0,65	0,14	0,08	0,08
30	30	0,97	0,41	0,44	0,08	0,00
	50	0,95	0,51	0,42	0,06	0,00
	100	0,81	0,63	0,31	0,04	0,02
Média		0,84	0,53	0,29	0,06	0,04

A Tabela 7 apresenta os tempos de execução (em ms) de cada regra de despacho. A estrutura desta tabela é a mesma da Tabela 6. Os resultados apresentados nesta tabela mostram que os tempos gastos pelas regras SPT, EDD, MDD e MDDa são de mesma ordem de grandeza. Já os tempos da regra com visão do futuro (PRTTa) são relativamente mais elevados. A principal razão para isto deve ao fato de que em todos os instantes de programação o método de programação da regra avalia todas as tarefas e não apenas as tarefas disponíveis. No entanto estes tempos são considerados satisfatórios, devido a sua ordem de grandeza, 10^{-3} ·s que é suficiente para a maioria das aplicações práticas, e devido ao ganho que esta regra apresenta na qualidade de suas soluções.

Tabela 7 - Tempo computacional (em ms) para as regras de despacho consideradas.

Estágios	Tarefas	SPT	EDD	MDD	MDDa	PRTTa
5	30	0,10	0,13	0,13	0,16	0,16
	50	0,13	0,30	0,32	0,26	0,49
	100	0,33	0,39	1,24	1,14	3,12
10	30	0,16	0,10	0,16	0,16	0,16
	50	0,33	0,35	0,29	0,42	0,91
	100	1,05	1,11	1,17	1,60	6,54
30	30	0,40	0,39	0,45	0,43	0,75
	50	0,65	0,84	0,85	0,81	2,51
	100	2,70	3,19	3,32	3,27	19,43
Média		0,65	0,76	0,88	0,92	3,79

Na próxima seção são feitas comparações entre as regras de despacho consideradas com a programação feita através do método de programação em ordem inversa.

3.4. Método de programação em ordem inversa

A Tabela 8, apresentada a seguir, mostra o desempenho das regras de despacho consideradas no método de programação em ordem inversa e possui a mesma estrutura da Tabela 6. Seus resultados mostram que as regras desenvolvidas (MDDa e PRTTa) apresentam um desempenho bem superior se comparado com o desempenho das regras conhecidas na literatura (SPT, EDD, MDD). Comparando os resultados das duas melhores regras, percebe-se que a regra com visão do futuro supera amplamente a segunda melhor regra (MDDa) em termos de RDI médio. Considerando o número de melhores soluções, a regra PRTTa supera a regra MDDa com 2.735 instâncias nas quais esta regra atingiu a melhor solução, em contraposição a 1.204 da regra MDDa.

Tendo em vista o maior esforço computacional da regra PRTTa em comparação com a regra MDDa, como mostra a Tabela 9, pode-se concluir que a segunda regra é vantajosa em casos nos quais a rapidez seja a prioridade. Caso contrário, a regra PRTTa é mais vantajosa devido aos seus melhor resultados.

Tabela 8 - Média do RDI para o método de programação em ordem inversa.

Estágios	Tarefas	SPT	EDD	MDD	MDDa	PRTTa
5	30	0,64	0,65	0,31	0,11	0,08
	50	0,68	0,64	0,18	0,07	0,06
	100	0,83	0,51	0,08	0,03	0,02
10	30	0,65	0,68	0,38	0,12	0,05
	50	0,62	0,75	0,28	0,08	0,06
	100	0,61	0,73	0,14	0,06	0,06
30	30	0,68	0,72	0,44	0,17	0,02
	50	0,66	0,82	0,39	0,15	0,01
	100	0,61	0,85	0,29	0,07	0,01
Média		0,66	0,71	0,28	0,10	0,04

Tabela 9 - Tempo computacional (em ms) para o método de programação em ordem inversa.

Estágios	Tarefas	SPT	EDD	MDD	MDDa	PRTTa
5	30	0,23	0,32	0,20	0,26	0,39
	50	0,39	0,52	0,65	0,62	1,04
	100	1,66	1,69	2,41	2,21	6,61
10	30	0,29	0,43	0,36	0,49	0,68
	50	0,98	1,00	0,75	1,10	2,15
	100	3,19	3,32	4,07	3,88	13,80
30	30	1,10	1,14	1,08	1,01	1,82
	50	2,21	2,57	2,44	2,77	5,56
	100	9,31	9,57	10,52	24,66	41,45
Média		2,15	2,28	2,50	4,11	8,17

A Tabela 9 mostra que a média de tempo para a resolução de uma instância através do método de programação em ordem inversa, utilizando-se a regra PRTTa, é de 8,17ms em contraposição a 4,11ms da regra MDDa. Vale destacar que o maior tempo computacional da regra PRTTa para este método se deve ao fato da necessidade deste método programar através de seu método particular de programação por lista de prioridades (apresentado na seção 2.1) e, conforme mostra a Tabela 7, este método particular da regra PRTTa é computacionalmente mais custoso por considerar, em todo o instante de decisão, todas as tarefas e não apenas as disponíveis.

Na seção seguinte são feitas comparações entre o método de programação por lista de prioridades e o método de programação em ordem inversa.

3.5. Comparações entre os métodos de programação

A Tabela 10 apresenta a média do RDI para as cinco regras consideradas e para os dois métodos apresentados. O método de programação em ordem inversa é indicado com a sigla BW, do inglês *backward*. Por exemplo, a média do RDI para as 4.320 instâncias utilizando o método de programação em ordem inversa com a regra SPT é igual a 0,34. Este valor está localizado na última linha da coluna SPT-BW. Os valores destacados em cinza indicam as menores médias do RDI.

Tabela 10 - Média do RDI para as cinco regras e para os dois métodos de programação.

Estágios	Tarefas	SPT	SPT-BW	EDD	EDD-BW	MDD	MDD-BW	MDDa	MDDa-BW	PRTTa	PRTTa-BW
5	30	0,82	0,32	0,53	0,45	0,35	0,15	0,13	0,06	0,12	0,05
	50	0,78	0,44	0,56	0,51	0,22	0,12	0,10	0,05	0,10	0,05
	100	0,88	0,57	0,45	0,43	0,09	0,06	0,04	0,02	0,04	0,02
10	30	0,91	0,25	0,47	0,37	0,42	0,15	0,10	0,05	0,06	0,03
	50	0,78	0,32	0,60	0,52	0,34	0,15	0,11	0,05	0,10	0,04
	100	0,67	0,43	0,66	0,60	0,17	0,10	0,11	0,05	0,10	0,05
30	30	0,97	0,20	0,41	0,29	0,45	0,13	0,10	0,05	0,02	0,01
	50	0,95	0,24	0,51	0,39	0,43	0,14	0,09	0,05	0,03	0,01
	100	0,81	0,29	0,64	0,53	0,34	0,15	0,07	0,03	0,05	0,01
Total		0,84	0,34	0,54	0,45	0,31	0,13	0,09	0,05	0,07	0,03

Comparando os dois métodos, independentemente da regra de despacho, os resultados mostram uma considerável melhora do método de programação em ordem inversa se comparado com o método de programação por lista de prioridades. A regra EDD apresenta o menor ganho com o método inverso (redução de 16% - 0,54 para 0,45), já a regra SPT apresenta o maior ganho (redução de 60% - 0,84 para 0,34). As regras desenvolvidas também proporcionam bons ganhos com reduções de 49% (MDDa) e 58% (PRTTa). O maior ganho da regra SPT é explicado pelo fato de que esta regra nunca considera as datas de entrega no método de programação por lista de prioridade. No entanto, os instantes de liberação do método de programação em ordem inversa introduz o fato das tarefas estarem atrasadas ou adiantadas.

Também foi calculada a média do RDI agrupando os resultados das cinco regras (SPT, EDD, MDD, MDDa e PRTTa). Os resultados são apresentados na Tabela 11. O método de programação por lista de prioridades é indicado com a sigla FW, do inglês *forward*. Por exemplo, a primeira coluna (FW) indica a média do RDI para as cinco regras sendo a programação feita através do método de programação por lista de prioridades. Portanto a média do RDI

considerando os resultados obtidos com este método e com as cinco regras é de 0,37 em contraposição a 0,20 do método de programação em ordem inversa, ratificando a superioridade deste método frente ao primeiro. Analisando os resultados pelo número de estágios e de tarefas, pode-se perceber que a superioridade do segundo método existe em todas as dimensões das instâncias.

Tabela 11 – Média do RDI para os dois métodos de programação.

Estágios	Tarefas	FW	BW
5	30	0,39	0,21
	50	0,35	0,23
	100	0,30	0,22
10	30	0,39	0,17
	50	0,38	0,22
	100	0,34	0,25
30	30	0,39	0,13
	50	0,40	0,17
	100	0,38	0,20
Total		0,37	0,20

3.6. Comparações com métodos da literatura

Nesta seção será feita a comparação com os métodos de Lee *et al.* (2004). Começando com o método de programação por lista de prioridade, a Tabela 12 foi construída para comparação das três melhores regras através deste método de programação (MDD, MDDa e PRTTa) com as duas melhores regras de Lee *et al.* (2004), ATC e COVERT.

Tabela 12 - Comparação para o método de programação por lista de prioridade (RDI médio).

Estágios	Tarefas	MDD	MDDa	PRTTa	ATC	COVERT
5	30	0,86	0,20	0,17	0,39	0,34
	50	0,77	0,26	0,25	0,45	0,32
	100	0,67	0,21	0,22	0,39	0,37
10	30	0,95	0,18	0,07	0,23	0,45
	50	0,88	0,20	0,16	0,28	0,41
	100	0,75	0,30	0,30	0,35	0,34
30	30	0,98	0,21	0,01	0,22	0,58
	50	0,99	0,18	0,01	0,19	0,60
	100	0,97	0,14	0,07	0,15	0,51
Média		0,87	0,21	0,14	0,29	0,43

Analisando os resultados da Tabela 12, pode-se perceber que as regras MDDa e PRTTa tem desempenho superior as demais regras e ainda apresentam a vantagem de possuírem fórmulas para os seus cálculos relativamente mais simples do que as regras ATC e COVERT. Além disso, as regras ATC e COVERT apresentam a desvantagem de necessitarem de dois parâmetros de difícil ajuste. Apesar da regra PRTTa apresentar um parâmetro de ajuste (α), este ajuste mostrou-se relativamente simples. A superioridade também está presente no número de melhores soluções: 351(MDD), 759(MDDa), 2607(PRTTa), 520(ATC), 468 (COVERT).

A seguir comparamos o método de programação em ordem inversa apresentado neste trabalho com o método de programação em ordem inversa (*backward*) de Lee *et al.* (2004). De todas as regras consideradas por Lee *et al.* (2004), serão consideradas as duas regras que apresentaram os melhores resultados em relação a média do RDI para as 4.320 instâncias, ou seja, as regras ATC e COVERT que serão denotadas como ATC (BW-Lee) e COVERT (BW-Lee). Também serão considerados os resultados das duas melhores regras apresentadas neste trabalho: MDDa-BW e PRTTa-BW.

Neste momento é interessante ressaltar a diferença entre o método de programação em ordem inversa proposto por Lee *et al.* (2004) e o método inverso proposto neste trabalho. A principal diferença encontra-se na programação do problema inverso (Passo 2 - vide seção 2.2). Enquanto Lee *et al.* (2004) fazem esta programação utilizando a mesma regra de despacho do Passo 0, no método proposto utiliza-se a regra de maior instante de conclusão, sendo estes instantes os instantes obtidos na programação do Passo 0. A justificativa desta regra é o fato de que em caso de empate deve-se escolher a tarefa que mantenha a programação obtida no Passo 0, visto que esta programação é uma boa programação. A maneira encontrada para satisfazer esta condição foi considerar o maior instante de conclusão, visto que nesta etapa do método está se programando em um horizonte de tempo invertido. Outra melhora do método proposto é o procedimento do Passo 6. Lee *et al.* não fazem nenhum tipo de realocação no primeiro estágio e conseqüentes remoções de tempos ociosos nos estágios seguintes.

A Tabela 13 mostra o RDI médio considerando as quatro regras supracitadas. Os resultados mostram que ambas as regras desenvolvidas neste trabalho superam as duas melhores regras de

Lee *et al.* (2004). Além disso, as regras deste trabalho apresentam a vantagem de serem mais simples que as regras propostas por Lee, visto que uma delas (MDDa) não necessita de parâmetro de ajuste e a outra (PRTTa) possui apenas um parâmetro que mostrou-se de fácil ajuste. Já as regras ATC e COVERT necessitam de dois parâmetros de ajuste cada. Ao mesmo tempo, as regras MDDa e PRTTa possuem fórmulas de cálculo bem menos complexas se comparadas com as regras ATC e COVERT.

A superioridade das regras também é representada no número de melhores soluções. Ao mesmo tempo em que PRTTa-BW e MDDa-BW apresentam, respectivamente, 2.541 e 1.178 melhores soluções, ATC(BW-Lee) e COVERT (BW-Lee) apresentam, respectivamente, 461 e 535. Analisando os resultados, também se percebe a superioridade da regra PRTTa-BW em relação ao RDI médio (0,15) e em relação ao número de melhores soluções (2.541) que são muito superiores às outras regras. A superioridade também aparece considerando o número de estágios e o número de tarefas das instâncias, visto que a regra PRTTa-BW superara as demais regras em todas as nove combinações estágios/tarefas.

Tabela 13 - Comparação entre os métodos de programação em ordem inversa (RDI médio).

Estágios	Tarefas	MDDa-BW	PRTTa-BW	ATC(BW-Lee)	COVERT (BW-Lee)
5	30	0,43	0,33	0,57	0,54
	50	0,27	0,24	0,68	0,60
	100	0,19	0,15	0,60	0,68
10	30	0,46	0,22	0,61	0,64
	50	0,33	0,18	0,68	0,67
	100	0,12	0,09	0,76	0,81
30	30	0,52	0,09	0,75	0,77
	50	0,60	0,07	0,71	0,77
	100	0,31	0,02	0,82	0,75
Total		0,36	0,15	0,69	0,69

Para aprimorar a comparação com os resultados da literatura, também será feita uma comparação considerando os valores da medida de desempenho (atraso total) para as 4.320 instâncias propostas por Lee *et al.* (2004). Esta comparação tem a finalidade de mostrar a diferença entre os dois métodos (independentemente da regra de despacho). Os resultados são apresentados na Tabela 14. Nesta tabela a primeira coluna refere-se ao método de programação em ordem inversa para a regra SPT-BW. A última linha indica a média do atraso total para as

4.320 instâncias propostas por Lee *et al.* (2004). Os valores destacados em cinza indicam os melhores resultados.

Tabela 14 - Comparação em relação ao atraso total.

Estágios	Tarefas	SPT-BW	SPT(BW-Lee)	EDD-BW	EDD(BW-Lee)	MDD-BW	MDD(BW-Lee)
5	30	7.163	7.044	7.484	7.609	6.902	6.901
	50	14.537	14.574	15.527	16.323	13.127	13.525
	100	34.781	34.336	34.098	36.675	26.244	28.015
10	30	11.221	11.012	11.469	11.509	11.049	10.953
	50	22.407	22.875	24.043	25.065	21.593	22.030
	100	61.628	70.570	71.634	78.942	55.037	60.417
30	30	28.324	28.126	28.497	28.444	28.207	28.106
	50	49.717	49.034	50.568	50.526	49.245	48.886
	100	114.997	118.603	122.924	133.382	111.956	113.443
Total		38.308	39.575	40.694	43.164	35.929	36.919

De acordo com a Tabela 14, pode-se perceber o desempenho superior do método em ordem inversa proposto neste trabalho, visto que para as três regras a média do atraso total é menor utilizando o método deste trabalho. Por exemplo, a média do atraso total da regra MDD-BW é de 35.929, já para MDD(BW-Lee) a mesma média é de 36.919. Analisando o número de estágios e tarefas a regra MDD-BW apresenta o melhor desempenho frente a regra MDD(BW-Lee) em cinco dos nove cenários (5 estágios/50 tarefas, 5 estágios/100 tarefas, 10 estágios/50 tarefas, 10 estágios/100 tarefas e 30 estágios/100 tarefas)

No próximo capítulo é apresentado o algoritmo com foco no gargalo que visa primeiro programar o estágio gargalo para depois programar os outros estágios do sistema. Este método baseia-se na idéia de que o programa das tarefas no gargalo é essencial para um fluxo apropriado das mesmas por todos os estágios.

4. ALGORITMO COM FOCO NO GARGALO

Neste capítulo é apresentado um novo método de programação chamado: algoritmo com foco no gargalo. Este método tem como base o método desenvolvido por Lee *et al.* (2004) tendo em vista seus bons resultados e o fato de tratar-se do mesmo problema. Este método é fundamentado no princípio de que uma boa programação para o estágio gargalo é essencial para um bom fluxo das tarefas através do sistema. Para isto, neste capítulo é apresentada uma idéia geral do método proposto, o seu algoritmo e por fim um exemplo ilustrativo.

4.1. Método proposto

De acordo com Yang *et al.* (2000) e Lee *et al.* (2004), os melhores resultados obtidos para a minimização do atraso total no ambiente *flowshop* flexível procuram primeiro encontrar uma boa programação para o estágio gargalo. No método proposto, primeiramente é construída uma programação para o estágio gargalo, para em seguida programar os outros estágios com base nos resultados da programação do gargalo. Para começar programando as tarefas no estágio gargalo, deve-se resolver um problema de máquinas em paralelo. Como o estágio gargalo pode não ser o primeiro estágio do sistema, devem-se estimar os instantes de liberação das tarefas para este estágio. Neste método os instantes de liberação no gargalo são atualizados de maneira iterativa visando uma boa programação final para o sistema. No método iterativo, é usada uma heurística para a programação do estágio gargalo e então é feita a programação dos demais estágios utilizando os métodos de programação por lista de prioridades em ordem direta e inversa.

No método de programação com foco no gargalo primeiramente deve-se identificá-lo. Para esta identificação será considerado o mesmo critério utilizado por Yang *et al.* (2000) e Lee *et al.* (2004), ou seja, o estágio gargalo é o estágio com maior carga de trabalho (W_i) calculada como:

$$W_i = \frac{\sum_{j=1}^n p_{ji}}{m_i} \quad (4.1)$$

Identificado o estágio gargalo, deve-se construir o problema de máquinas em paralelo estimando os instantes de liberação das tarefas, já que o estágio gargalo pode não ser o primeiro

estágio, e redefinindo as datas de entrega das mesmas. Para estimar os instantes de liberação serão utilizados dois métodos utilizados por Lee *et al.* (2004), denominados por RT1 e RT3 (RT de *ready time* da língua inglesa). O primeiro método é utilizado apenas uma vez e é um limitante inferior para os instantes de liberação, já o RT3 necessita de uma programação viável, visto que ele utiliza atrasos e adiantamentos, e é usado para melhorar a programação encontrada. A redefinição das datas de entrega para o problema de máquinas em paralelo do estágio gargalo é feita da mesma maneira utilizada pelas regras desenvolvidas (MDDa e PRTTa): obtém-se a data de entrega (d_{ji}) apenas subtraindo o trabalho remanescente da data de entrega real (d_j).

Para resolver o problema de máquinas em paralelo do estágio gargalo, foi escolhida a regra PRTTa desenvolvida neste trabalho. As três principais razões desta escolha são: o fato de ser uma regra desenvolvida para trabalhar com o problema de máquinas em paralelo, que considera diferentes instantes de liberação e apresenta um bom desempenho para a minimização do atraso total no ambiente *flowshop* flexível.

Depois de resolvido o problema de máquinas em paralelo do estágio gargalo, deve-se achar uma programação para os estágios anteriores ao gargalo que respeite os instantes de programação das tarefas no estágio gargalo obtidos na etapa anterior. Nesta etapa os estágios anteriores ao gargalo são vistos como um *flowshop* flexível independente do resto do sistema. Para este *flowshop* flexível são definidas as datas de entrega d_j' como o instante de programação da tarefa j no estágio gargalo, visto que pretende-se respeitar estes instantes de programação, ou seja, para que a programação deste *flowshop* flexível dos estágios anteriores seja viável com a programação do estágio gargalo, todas as tarefas devem cumprir suas datas de entrega. Para resolver o problema de *flowshop* flexível dos estágios anteriores foi escolhido o método de programação em ordem inversa devido ao seu bom desempenho.

Caso a programação dos estágios anteriores ao gargalo respeite os instantes de programação do estágio gargalo, a programação do gargalo é viável e mantém-se. Caso contrário, é necessário um método para viabilização da solução do gargalo. Testes preliminares mostraram que o melhor método para a viabilização da programação do gargalo é o método de programação por lista de prioridade com a prioridade dada pelo instante de programação da solução inviável. Esta priorização é interessante, pois com esta regra obtém uma programação que mais se aproxima da

programação original (inviável) que era uma programação promissora, visto que foi obtida com uma regra de bom desempenho (PRTTa).

A última etapa é obter uma programação para os estágios posteriores ao gargalo. Esta programação é feita pelo método de programação por lista de prioridade, respeitando os instantes de liberação do estágio gargalo. A Tabela 15 mostra um resumo do método de decomposição utilizado pelo algoritmo com foco no gargalo.

Tabela 15 - Método de decomposição do algoritmo com foco no gargalo.

	Estágios Anteriores	Gargalo	Estágios Posteriores
Problema	<i>Flow shop</i> flexível	Máquinas em paralelo com diferentes instantes de liberação	<i>Flow shop</i> flexível com diferentes instantes de liberação
Método de Resolução	Programação em ordem inversa	Programação por lista de prioridade	Programação por lista de prioridade
Regra	SPT, EDD, MDD, MDDa ou PRTTa	PRTTa	Mesma regra dos estágios anteriores
Data de Entrega	d'_j	d_{jl}	d_j

Obtida uma programação completa para o sistema, visando melhorar a solução já encontrada, é feita uma programação em ordem inversa com base na programação encontrada. Por fim, os instantes de liberação das tarefas no estágio gargalo são ajustados com base em seus adiantamentos ou atrasos da programação atual e o algoritmo volta para a resolução de um novo problema de máquinas em paralelo do estágio gargalo. Este procedimento é repetido até que um número máximo de iterações seja alcançado.

O algoritmo completo para a programação com foco no estágio gargalo é apresentado no fluxograma da Figura 5. As setas verdes contínuas indicam que a condição foi satisfeita, caso contrário as setas são vermelhas e tracejadas.

[illegible]

- FO_{inc} – atraso total referente a melhor solução encontrada até o momento;
- FO – atraso total referente à solução atual;
- $MaxIt$ – número máximo de iterações;
- $Cont$ – contador para o número de iterações.

O algoritmo com foco no gargalo trabalha com duas soluções: a solução atual, que é construída e modificada ao longo do algoritmo, e a solução incumbente, que é a solução com menor valor para o atraso total encontrada até o momento. A solução incumbente é retornada no fim do algoritmo como a melhor solução encontrada.

Por fim, devem-se definir os dois métodos para estimar os instantes de liberação no gargalo (RT1 e RT3). Conforme dito anteriormente estes são os mesmo métodos utilizados por Lee *et al.* (2004). O RT1 é um limitante inferior para os instantes de liberação, ou seja, igual a soma dos tempos de processamento dos estágios anteriores ao gargalo.

Já o RT3, utilizado depois que já se tem uma programação viável, utiliza os atrasos e adiantamentos desta programação. Sendo assim, este método faz o instante de liberação de uma tarefa j igual ao último instante de liberação estimado para a tarefa mais o adiantamento ou menos o atraso, desde que esse valor não seja menor que o limitante inferior (RT1) para a tarefa j . Este método baseia-se na idéia de que se uma tarefa está atrasada ela pode começar mais cedo e, do mesmo modo, se ela estiver adiantada, pode começar mais tarde.

Para formalizar estes dois métodos, considere Π o conjunto de estágios anteriores ao gargalo, b o estágio gargalo e r'_{jb} o instante de liberação estimado para tarefa j no estágio b . Desta maneira, têm-se:

$$\text{RT1: } r'_{jb} = \sum_{k \in \Pi} p_{jk} \quad (4.2)$$

$$\text{RT3: } r'_{jb} = \max \left\{ r'_{jb} - (C_j - d_j), \sum_{k \in \Pi} p_{jk} \right\} \quad (4.3)$$

Vale destacar que se $C_j - d_j > 0$, a tarefa está atrasada e o valor do atraso será subtraído do último instante de liberação estimado (r'_{jb}). Caso contrário ($C_j - d_j < 0$), a tarefa está adiantada e esse valor de adiantamento será somado ao último r'_{jb} .

4.3. Exemplo ilustrativo

Para fins didáticos, será considerado um exemplo de pequenas dimensões para ilustrar o algoritmo com foco no gargalo. Considere um *flowshop* flexível com três estágios, quatro tarefas, uma máquina no segundo estágio e duas máquinas nos outros dois estágios. As datas de entrega e tempos de processamento são mostrados na Tabela 16.

Tabela 16 - Parâmetros para o exemplo do algoritmo com foco no gargalo.

tarefas	d_j	p_{jl}		
		$l=1$	$l=2$	$l=3$
$j=1$	16	8	1	4
$j=2$	19	7	1	7
$j=3$	11	3	3	3
$j=4$	20	4	9	3

O algoritmo começa identificando o estágio gargalo. Para isso calcule as cargas de trabalho (W_l) para os três estágios. Utilizando a fórmula (4.1), têm-se: $W_1 = \frac{8+7+3+4}{2} = 11$, $W_2 = \frac{1+1+3+9}{1} = 14$ e $W_3 = \frac{4+7+3+3}{2} = 8,5$. Portanto, o segundo estágio é o gargalo do sistema ($b=2$). Em seguida, estime os instantes de liberação para o problema de máquinas em paralelo do segundo estágio através do método RT1. Assim sendo, $r'_{12}=8$, $r'_{22}=7$, $r'_{32}=3$ e $r'_{42}=4$. Também são necessárias as datas de entrega (d_{j2}) calculadas através da fórmula (2.1): $d_{12} = 16 - 4 = 12$, $d_{22} = 19 - 7 = 12$, $d_{32} = 11 - 3 = 8$, $d_{42} = 20 - 3 = 17$. Com isso, já é possível resolver o problema de máquinas em paralelo do estágio gargalo. Para isso, faça o método de programação por lista de prioridade para a regra PRTTa. O Gráfico 6 mostra o gráfico de Gantt para a programação feita através da regra PRTTa.

Gráfico 6 – Gráfico de Gantt para o problema de máquinas em paralelo do gargalo.

O próximo passo é resolver o *flowshop* flexível dos estágios anteriores ao gargalo, ou seja, o estágio $l=1$. Para isso, defina os d'_j como o instante de programação do estágio gargalo, ou seja, $d'_3=3$, $d'_4=6$, $d'_2=15$ e $d'_1=16$. Resolva este problema com o método de programação em ordem inversa. A regra escolhida foi a EDD por ser a mais simples, tendo em vista que a finalidade do exemplo é apenas ilustrar o funcionamento do algoritmo. O resultado final deste método para este problema está representado através no Gráfico 7, juntamente com a programação do gargalo obtida anteriormente.

Gráfico 7 - Programação para os problemas dos estágios anteriores ao gargalo e do gargalo.

Como a programação do gargalo é viável com a programação dos estágios anteriores, verifique se existem tempos ociosos entre os estágio 1 e 2. Como não existem, o algoritmo segue para a próxima etapa que é programar os estágios posteriores através do método de programação por lista de prioridades (EDD), obtendo assim uma programação completa para o sistema. Esta programação é mostrada no Gráfico 8 e na Tabela 17.

Gráfico 8 - Gráfico de Gantt para a 1ª iteração do algoritmo com foco no gargalo.

Tabela 17 - Programação completa para a 1ª iteração do algoritmo com foco no gargalo.

tarefas	d_j	p_{jl}			Instante da programação			C_{jl}			T_j	$C_j - d_j$
		$l=1$	$l=2$	$l=3$	$l=1$	$l=2$	$l=3$	$l=1$	$l=2$	$l=3$		
$j=1$	16	8	1	4	3	16	18	11	17	22	6	6
$j=2$	19	7	1	7	4	15	16	11	16	23	4	4
$j=3$	11	3	3	3	0	3	6	3	6	9	0	-2
$j=4$	20	4	9	3	0	6	15	4	15	18	0	-2
$\sum T_j$											10	

A última coluna da Tabela 17 mostra a diferença $C_j - d_j$ que será utilizada no método RT3. A penúltima coluna da mesma tabela mostra o atraso (T_j) de cada tarefa e na última linha está calculado o atraso total ($\sum T_j$) que é a função objetivo FO da solução atual. Como o atraso total ($\sum T_j = FO = 10$) da programação atual é menor que o atraso total da solução incumbente ($FO_{inc} = \infty$), esta solução passa a ser a incumbente e $FO_{inc} = 10$. Em seguida, com base na programação atual

(presente na Tabela 17), faça o método de programação em ordem inversa. O resultado obtido foi o mesmo em termos de atraso total ($FO=10$), portanto, como $FO_{inc}=10$, o algoritmo segue sem substituir a solução incumbente.

De posse de uma programação completa, o algoritmo aplica o método RT3 que estima os instantes de liberação através de (4.3). O método RT3 é ilustrado na Tabela 18. A primeira coluna refere-se aos últimos instantes estimados. A segunda coluna é a diferença utilizada em (4.3) que foi retirada da programação completa da Tabela 17. A terceira coluna são os instantes de liberação finais estimados pelo método, calculados através de (4.3).

Tabela 18 - Método RT3.

tarefas	r'_{j2}	$C_j - d_j$	$r'_{j2} \text{ (RT3)}$
$j=1$	8	6	8
$j=2$	7	4	7
$j=3$	3	-2	5
$j=4$	4	-2	6

Feito isso, têm-se um novo problema de máquinas em paralelo. Resolva-o através do método de programação por lista de prioridade com a regra PRTTa. Em seguida, programe o *flowshop* flexível dos estágios anteriores ao gargalo. Os resultados dessas duas programações são mostrados no.

Gráfico 9 - Programação dos estágios anteriores e do gargalo.

Como a programação do gargalo não é viável, refaça a programação do gargalo através do método de programação por lista de prioridades com a prioridade dada pelos instantes de programação da solução inviável. Em seguida, programe os estágios posteriores por meio da regra EDD. Com isso, obtém-se uma segunda programação completa, mostrada no Gráfico 10 e na Tabela 19.

Gráfico 10 - Programação completa para a 2ª iteração.

Tabela 19 - Programação completa para a 2ª iteração.

tarefas	d_j	p_{jl}			Instante da programação			C_{jl}			T_j	$C_j - d_j$
		$l=1$	$l=2$	$l=3$	$l=1$	$l=2$	$l=3$	$l=1$	$l=2$	$l=3$		
$j=1$	16	8	1	4	3	11	12	11	12	16	0	0
$j=2$	19	7	1	7	0	7	8	7	8	15	0	-4
$j=3$	11	3	3	3	0	3	6	3	6	9	0	-2
$j=4$	20	4	9	3	7	12	21	11	21	24	4	4
ΣT_j											4	

Como o atraso total da programação atual ($FO=4$) é menor que o atraso total da solução incumbente ($FO_{inc}=10$), esta solução passa a ser a incumbente e $FO_{inc}=4$. Em seguida, com base na programação atual (presente na Tabela 19), faça o método de programação em ordem inversa. O resultado obtido foi o mesmo em termos de atraso total ($FO=4$), portanto, como $FO_{inc}=4$, o

algoritmo segue sem substituir a solução incumbente até que seja atingido o número máximo de iterações.

Neste exemplo de pequenas dimensões, é interessante ressaltar o funcionamento dos métodos de estimar os instantes de liberação no gargalo. Através do RT1, foi obtido um atraso total igual a 10. Na primeira iteração do RT3, este método já conseguiu uma melhora significativa do atraso total de 10 para 4.

O próximo capítulo apresenta os resultados computacionais do algoritmo com foco no gargalo.

5. RESULTADOS FINAIS

Para verificar o desempenho do algoritmo com foco no estágio gargalo proposto, os resultados deste algoritmo, através das cinco regras de despacho, foram comparados entre si e com os resultados obtidos pelo algoritmo com foco no gargalo de Lee *et al.* (2004).

5.1. Testes preliminares

O algoritmo com foco no gargalo necessita do dimensionamento do parâmetro que limita o número máximo de iterações (MaxIt). Para isto foram feitos diversos testes preliminares e foi observado que o algoritmo proposto consegue melhoras mais expressivas para instâncias que possuem o estágio gargalo mais no começo do sistema. Tendo isto em vista, o número máximo de iterações foi definido de acordo com o parâmetro das instâncias que está relacionado com a posição do estágio gargalo no sistema. Lee *et al.* (2004) construíram 1080 instâncias para cada um dos quatro níveis da posição do estágio gargalo no sistema (primeiro, segundo, terceiro e quarto quartos). Por exemplo, para uma instância com 30 estágios, o gargalo está localizado no primeiro quarto do sistema, se e somente se ele estiver localizado do primeiro ao sétimo estágio.

Portanto, com base em diversos testes computacionais, o número máximo de iterações (MaxIt) foi definido para o primeiro, segundo, terceiro e quarto quartos respectivamente como: 100, 50, 30 e 30. Estes valores foram determinados de acordo com o tempo limite para o algoritmo que será explicado na seção 5.4.

5.2. Comparação entre as regras

A Tabela 20 apresenta a média do RDI obtida por meio do algoritmo com foco no gargalo através das cinco regras consideradas. Este método será denominado por meio da sigla BN do inglês *bottleneck*, portanto as regras através deste método serão representadas por SPT-BN, EDD-BN, MDD-BN, MDDa-BN e PRTTa-BN. A estrutura das tabelas desta seção é a mesma das tabelas do capítulo 3.

Analisando os resultados da Tabela 20, pode-se notar, novamente, o desempenho superior da regra desenvolvida que considera estados futuros do sistema (PRTTa). A outra regra

desenvolvida neste trabalho (MDDa) também apresenta resultados superiores, se comparada com as regras tradicionais da literatura. Considerando as nove combinações de estágios/tarefas as duas regras superam as demais em todas as categorias. Em relação ao número de melhores soluções, o desempenho também é superior. A regra PRTTa-BN possui o maior número de melhores soluções (2.330), em contraposição a 1.084 da regra MDDa-BN, 957 (EDD), 791(MDD) e 293 (SPT). O número de melhores soluções apenas confirma o desempenho superior das regras propostas.

Tabela 20 - RDI médio para o algoritmo com foco no gargalo.

Estágios	Tarefas	SPT-BN	EDD-BN	MDD-BN	MDDa-BN	PRTTa-BN
5	30	0,73	0,34	0,40	0,26	0,26
	50	0,84	0,28	0,30	0,23	0,25
	100	0,83	0,17	0,19	0,14	0,17
10	30	0,78	0,36	0,42	0,29	0,14
	50	0,90	0,34	0,41	0,27	0,18
	100	0,93	0,27	0,22	0,14	0,17
30	30	0,82	0,36	0,40	0,39	0,11
	50	0,96	0,36	0,42	0,39	0,05
	100	0,99	0,31	0,36	0,23	0,08
Média		0,87	0,31	0,35	0,26	0,16

A Tabela 21 mostra a média do tempo computacional para o algoritmo com foco no gargalo. Os resultados são mais elevados que os tempos dos outros métodos, mas mesmo assim ainda são satisfatórios para aplicação em casos reais. Conforme esperado a regra PRTTa-BN é mais custosa que as demais, mas seus resultados justificam o maior esforço computacional.

Tabela 21 – Média do tempo computacional para o algoritmo (em ms).

Estágios	Tarefas	SPT-BN	EDD-BN	MDD-BN	MDDa-BN	PRTTa-BN
5	30	23	23	24	24	30
	50	46	45	47	48	74
	100	147	131	154	153	377
10	30	39	39	41	40	56
	50	85	81	91	89	156
	100	273	255	288	290	909
30	30	95	94	99	96	143
	50	216	206	232	227	436
	100	692	646	712	714	3.675
Média		180	169	188	187	651

5.3. Comparação entre os três métodos

Também foi calculada a média do RDI agrupando os resultados das cinco regras (SPT, EDD, MDD, MDDa e PRTTa) para os três métodos apresentados neste trabalho. Os resultados são apresentados na Tabela 22. O método de programação por lista de prioridades é indicado com a sigla FW, o método de programação em ordem inversa por BW e o algoritmo com foco no gargalo por BN. Por exemplo, a terceira coluna (BN) indica a média do RDI para as cinco regras sendo a programação feita através do algoritmo com foco no gargalo. Portanto a média do RDI considerando os resultados obtidos com este método e com as cinco regras é de 0,04 em contraposição a 0,23 do método de programação em ordem inversa e 0,39 do método de programação por lista de prioridade. Analisando os resultados pelo número de estágios e de tarefas, pode-se perceber a superioridade do método com foco no gargalo em todas as combinações de estágios/tarefas.

Tabela 22 - Média do RDI para os três métodos.

Estágios	Tarefas	FW	BW	BN
5	30	0,42	0,25	0,04
	50	0,38	0,27	0,04
	100	0,31	0,23	0,03
10	30	0,42	0,21	0,03
	50	0,41	0,26	0,04
	100	0,37	0,28	0,04
30	30	0,41	0,16	0,03
	50	0,42	0,19	0,03
	100	0,40	0,23	0,04
Média		0,39	0,23	0,04

5.4. Comparação com a literatura

Nesta seção será feita a comparação do algoritmo com foco no gargalo com o método com foco no gargalo (*bottleneck focused* - BF) de Lee *et al.* (2004). Serão consideradas as três regras que apresentaram melhor desempenho MDD, ATC e COVERT que serão denotadas como MDD (BF-Lee), ATC (BF-Lee) e COVERT (BF-Lee). Também serão considerados os resultados das cinco regras apresentadas neste trabalho: SPT-BN, EDD-BN, MDD-BN, MDDa-BN e PRTTa-BN.

Os resultados estão apresentados na Tabela 23. Analisando os resultados, nota-se que a média do RDI da melhor regra deste trabalho (PRTTa-BN) é igual a 0,41, superando a melhor regra da literatura, ATC(BF-Lee) que apresentou um RDI médio de 0,43. Outra vantagem do método proposto é que ele apresenta melhor desempenho nas 2.880 instâncias de 5 e 10 estágios, ou seja, as duas melhores regras (MDDa-BN e PRTTa-BN) apresentam melhores resultados para 6 das 9 combinações estágios/tarefas. A desvantagem do método proposto aparece no número de melhores soluções, PRTTa-BN apresenta 1.216 em contraposição a 1.549 da ATC(BF-Lee).

Tabela 23 - Média do RDI para a comparação com o método da literatura.

Estágios	Tarefas	SPT-BN	EDD-BN	MDD-BN	MDDa-BN	PRTTa-BN	MDD(BF-Lee)	ATC(BF-Lee)	COVERT(BF-Lee)
5	30	0,56	0,37	0,39	0,35	0,34	0,58	0,58	0,58
	50	0,68	0,36	0,36	0,34	0,35	0,56	0,56	0,55
	100	0,70	0,23	0,25	0,22	0,24	0,48	0,48	0,49
10	30	0,62	0,46	0,48	0,44	0,39	0,50	0,47	0,48
	50	0,72	0,48	0,50	0,45	0,41	0,51	0,49	0,49
	100	0,79	0,42	0,40	0,36	0,38	0,43	0,43	0,43
30	30	0,74	0,62	0,63	0,62	0,55	0,32	0,30	0,32
	50	0,83	0,66	0,67	0,66	0,56	0,29	0,27	0,28
	100	0,89	0,63	0,65	0,58	0,54	0,31	0,29	0,30
Média		0,73	0,47	0,48	0,45	0,42	0,44	0,43	0,44

A Tabela 24 tem a finalidade de ilustrar o melhor desempenho do algoritmo em instâncias que possuem o gargalo no início do sistema, como foi afirmado no começo desta seção. Posição gargalo igual a 1 refere-se as instâncias que possuem o gargalo no primeiro quarto do sistema. Por exemplo, o valor da primeira linha e da primeira coluna (0,44) representa a média do RDI através da regra SPT-BN para as 1.080 instâncias que possuem o gargalo no primeiro quarto do sistema.

Através da análise dos resultados da Tabela 24 percebe-se o desempenho bem superior do método proposto para instâncias que possuem o gargalo no primeiro e segundo quarto. Em contraposição, o desempenho é inferior nas instâncias que possuem o gargalo no terceiro e quarto quarto. Uma solução para tentar melhorar o desempenho nestes quartos seria aumentar o número máximo de iterações do algoritmo. Porém, testes preliminares mostraram que este aumento não reflete em uma melhora muito significativa da média do RDI, além de existir a limitação do tempo computacional. Por isso foi escolhido concentrar os esforços computacionais nas instâncias que possuem o gargalo no começo do sistema.

Tabela 24 - Média do RDI em função da posição do gargalo.

Posição Gargalo	SPT-BN	EDD-BN	MDD-BN	MDDa-BN	PRTTa-BN	MDD(BF-Lee)	ATC(BF-Lee)	COVERT(BF-Lee)
1	0,47	0,31	0,30	0,26	0,22	0,72	0,68	0,70
2	0,78	0,45	0,45	0,42	0,39	0,43	0,42	0,42
3	0,83	0,55	0,57	0,53	0,51	0,33	0,33	0,33
4	0,83	0,58	0,59	0,57	0,56	0,30	0,30	0,30
Média	0,73	0,47	0,48	0,45	0,42	0,44	0,43	0,44

As médias do tempo computacional (em ms) para o algoritmo com foco no gargalo são apresentadas na Tabela 25. Devido à difícil tarefa de comparar tempos computacionais de métodos executados em diferentes computadores, foi utilizada a seguinte estratégia: para determinar o tempo limite do algoritmo foi feita uma equivalência com os tempos do método de programação por lista de prioridades, tendo em vista que os resultados de Lee *et al.* (2004) para este método foram exatamente reproduzidos neste trabalho.

Para Lee *et al.* (2004) a média do método em ordem direta foi de 0,032s e a média para o método com foco no gargalo foi de 10,707s. Portanto, o método com foco no gargalo é 335 vezes mais custoso que o método em ordem direta $\left(\frac{10,707}{0,032} \approx 335\right)$. Ou seja, para este trabalho o algoritmo com foco no gargalo não deve ter tempos médios mais que 335 vezes maiores que os tempos médios do método de programação por lista de prioridade.

De acordo com a Tabela 7 a média do tempo computacional para as regras que utilizam o método de programação por lista de prioridade (SPT, EDD, MDD e MDDa) é de 0,8ms. Portanto, o tempo limite para o algoritmo com foco no gargalo são de 268ms ($335 \times 0,8 = 268$).

Tabela 25 - Tempo computacional para o algoritmo com foco no gargalo.

Estágios	Tarefas	SPT-BN	EDD-BN	MDD-BN	MDDa-BN	PRTTa-BN
5	30	23	23	24	24	30
	50	46	45	47	48	74
	100	147	131	154	153	377
10	30	39	39	41	40	56
	50	85	81	91	89	156
	100	273	255	288	290	909
30	30	95	94	99	96	143
	50	216	206	232	227	436
	100	692	646	712	714	3.675
Média		180	169	188	187	651

Tendo isto em vista, analisando os resultados da Tabela 25, pode-se notar que o algoritmo com foco no gargalo proposto neste trabalho é computacionalmente menos custoso que o método

de Lee *et al.* (2004). Já a regra PRTTa-BN possui um média maior que o valor limite. Como explicado anteriormente, isto acontece porque o método de programação para esta regra é mais custoso.

No entanto, o algoritmo deste trabalho é mais simples que o de Lee *et al.* (2004), principalmente porque os autores utilizam um método bem complexo para a programação do estágio gargalo, baseado nos métodos KPM (KOULAMAS *et al.*, 1994) e PSK (PANWALKAR *et al.*, 1993) e adaptado para o problema de máquinas em paralelo com diferentes instantes de liberação. Além disso, os autores utilizam três métodos de estimar instantes de liberação no gargalo, sendo um desses métodos bastante custoso, pois estima os tempos de liberação no gargalo através de diversas programações através do método em ordem direta. Neste trabalho foi proposta a utilização de apenas dois, sendo estes os dois mais simples.

No próximo capítulo são apresentadas as conclusões e próximos passos para este trabalho.

6. CONCLUSÕES

O objetivo deste trabalho foi propor heurísticas para o problema de minimização do atraso total em um ambiente de *flowshop* flexível. Ao longo do trabalho, foi reproduzido o método de programação por lista de prioridades com regras bem conhecidas na literatura (SPT, EDD e MDD). Além disso, foram desenvolvidas duas novas regras para este mesmo método de programação: MDDa e uma nova regra que possui visão de estados futuros do sistema (PRTTa). Para isto também foi necessário desenvolver um novo método de programação para esta última regra. Ambas as regras (MDDa e PRTTa) tratam o problema de *flowshop* flexível como uma série de problemas de máquinas em paralelo. Esta estratégia mostrou-se eficaz no que diz respeito a minimização do atraso total neste tipo de ambiente de máquinas.

A regra MDDa possui uma fórmula de cálculo relativamente simples, não possui parâmetros de ajuste e apresentou resultados melhores em comparação com regras que eram conhecidas como as melhores regras para a minimização do atraso total no ambiente *flowshop* flexível (ATC e COVERT). Sua média do RDI foi de 0,21 em contraposição a 0,29 da ATC e 0,43 da COVERT. Além disso, ATC e COVERT possuem fórmulas de cálculo mais complexas e apresentam dois parâmetros de difícil ajuste. A regra PRTTa apresentou resultados ainda melhores (média do RDI igual a 0,14) e apesar de depender de um parâmetro de ajuste(α), esse parâmetro mostrou-se de fácil ajuste, ao mesmo tempo, PRTTa também possui uma fórmula relativamente simples.

Comparando o desempenho das duas melhores regras propostas, concluiu-se que a regra MDDa é mais simples, rápida e apresenta resultados competitivos com a regra PRTTa, portanto ela é mais vantajosa em casos nos quais as prioridades sejam a facilidade de implementação e rapidez.

Também foi desenvolvido um novo método de programação em ordem inversa. Este método formula um problema inverso utilizando as informações de uma programação através do método de programação por lista de prioridade. Esta estratégia já era conhecida na literatura, mas este trabalho propõe um novo método de se fazer a programação do problema inverso, no qual são utilizadas informações da programação direta.

A programação em ordem inversa para as regras MDDa-BW e PRTTa-BW também obtiveram um melhor desempenho do que as duas melhores regras de Lee *et al.* (2004) através da programação em ordem inversa, ATC (BW-Lee) e COVERT (BW-Lee). MDDa e PRTTa apresentam um RDI médio de, respectivamente, 0,36 e 0,15, em contraposição a 0,69 das regras ATC (BW-Lee) e COVERT (BW-Lee). Além disso, as regras propostas apresentam as vantagens já mencionadas de simplicidade de cálculo e de ajuste de parâmetros.

Já o algoritmo com foco no gargalo deste trabalho utiliza o método de programação por lista de prioridades da regra PRTTa para resolver o problema de máquinas em paralelo com diferentes instantes de liberação. Este método é mais simples do que o método de Lee *et al.* (2004), no qual são utilizadas adaptações dos métodos KPM e PSK. Além disso, o algoritmo desenvolvido utiliza apenas dois (RT1 e RT3) dos três métodos para a estimativa dos instantes de liberação no gargalo, sendo estes dois métodos bem mais simples que o outro método utilizado na literatura (RT2).

Os resultados do algoritmo com foco no gargalo mostraram-se competitivos com o método equivalente de Lee *et al.* (2004), com a vantagem de apresentar resultados superiores para problemas com 5 e 10 estágios e para problemas com o estágio gargalo localizado na primeira metade do sistema, além de ser relativamente mais simples e computacionalmente menos custoso (para as regras SPT, EDD, MDD e MDDa). Considerando a média do RDI para as 4.320 instâncias, a melhor regra (PRTTa-BN) apresentou um RDI médio (0,42), superando a melhor regra (ATC) de de Lee *et al.* (2004) que apresentou um RDI médio de 0,43.

Os próximos passos para este trabalho são: desenvolver uma estratégia para os problemas que possuem o estágio gargalo na segunda metade do sistema, visando melhorar a qualidade de resultados para as instâncias deste tipo; utilizar a representação em grafo apresentada e desenvolver uma meta-heurística para o problema; verificar se um método de prioridade, entre as máquinas de um estágio, traz melhoras significativas para os três métodos de programação desenvolvidos neste trabalho, por exemplo, quando, num instante de decisão, se tem mais máquinas do que tarefas, criar uma regra de priorização que decida em quais máquinas que as tarefas devem ser programadas.

REFERÊNCIAS BIBLIOGRÁFICAS

BAKER, K.R., BERTRAND, J.W.M., A dynamic priority rule for scheduling against due-dates. *Journal of Operations Management*, 3, 37-42, 1982.

BARMAN, S., Simple priority rule combinations: an approach to improve both flow time and tardiness. *International Journal of Production Research*, 35, 2857-2870, 1997.

BRAH, S. A., A comparative analysis of due date based job sequencing rules in a flow shop with multiple processors. *Production Planning and Control*, 7, 362-373, 1996.

CHOI, S.W., KIM, Y.D., LEE G.C., Minimizing total tardiness of orders with reentrant lots in a hybrid flowshop. *International Journal of Production Research*, 43, 2149-2167, 2005.

CHU, C., PORTMANN, M.-C., Some new efficient methods to solve the $n/1/r_i/\sum T_i$ scheduling problem, *European Journal of Operational Research*, 58, 404-413, 1992.

DU, J., LEUNG J.Y.T., Minimizing total tardiness on one machine is NP-hard. *Mathematics of Operations Research*, 15, 483-495, 1990.

DAVIS, J.S., KANET, J.J., Single machine scheduling with early and tardy completion costs. *Naval Research Logistics*, 40, 85-101, 1993.

GUINET, A., SOLOMON, M. M., KEDIA, P. K., DUSSAUCHOY, A., A computational study of heuristics for two-stage flexible flowshops. *International Journal of Production Research*, 34, 1399-1415, 1996.

HALL, N.G., KUBIAK, W., SETHI, S.P., Earliness-tardiness scheduling problems: II. Deviation of completion times about a restrictive common due date. *Operations Research*, 39, 847-856, 1991.

JAYAMOHAN, M. S., RAJENDRAN, C., A comparative analysis of two different approaches to scheduling in flexible flow shops. *Production Planning & Control*, 11, 572-580, 2000.

JIN, Z. H., OHNO, K., ITO, T., ELMAGHRABY, S.E., Scheduling hybrid flowshops in printed circuit board assembly lines. *Production and Operations Management*, 11, 216-230, 2002.

KANET J.J., SRIDHARAN V., Scheduling with Inserted Idle Time: Problem Taxonomy and Literature Review, *Operations Research*, 48, 99-110, 2000.

KIM, Y. D., Heuristics for flowshop scheduling problems minimizing mean tardiness. *Journal of the Operational Research Society*, 44, 19-28, 1993.

KIM, Y. D., A backward approach in list scheduling algorithms for multi-machine tardiness problems. *Computers and Operations Research*, 22, 307-319, 1995.

KOULAMAS, C. P., The total tardiness problem: review and extensions. *Operations Research*, 42, 1025-1041, 1994.

LAWRENCE, S.R., MORTON, T.E., Resource-constrained multi-project scheduling with tardy costs: Comparing myopic, bottleneck, and resource pricing heuristics. *European Journal of Operational Research*, 64, 168-187, 1993.

LENSTRA, J.K., RINNOOY KAN, A.H.G., BRUCKER, P., Complexity of machine scheduling problems. *Annals of Discrete Applied Mathematics*, 1, 343-362, 1977.

LEE, G.C., KIM, Y.D., A branch-and-bound algorithm for a two-stage hybrid flowshop scheduling problem minimizing total tardiness. *International Journal of Production Research*, 42, 4731-4743, 2004.

LEE, G.C., KIM, Y.D., CHOI, S.W., Bottleneck-focused scheduling for a hybrid flowshop. *International Journal of Production Research*, 42, 165-181, 2004.

LEE, G.C., KIM, Y.D., KIM, J.G., CHOI, S.H., A dispatching rule-based approach to production scheduling in a printed circuit board manufacturing system. *Journal of the Operational Research Society*, 54, 1038-1049, 2003.

LINN, R., ZHANG, W., Hybrid flow shop scheduling: a survey. *Computers and Industrial Engineering*, 37, 57-61, 1999.

LOGENDRAN, R., DeSZOEKE, P., BARNARD, F., Sequence-dependent group scheduling problems in flexible flow shops. *International Journal of Production Economics*, 102, 66-86, 2006.

NAWAZ, M., ENSCORE, E.E., HAM, I., A heuristic algorithm for the m-machine, n-job flowshop scheduling problem. *Omega*, 11, 91-95, 1983.

NOWICKI, E., SMUTNICKI, C., The flow shop with parallel machines: A tabu search approach. *European Journal of Operational Research*. 106, 226-253, 1998.

PANWALKAR, S. S., SMITH, M. L., KOULAMAS, C. P., A heuristic for the single machine tardiness problem. *European Journal of Operational Research*, 70, 304-310, 1993.

PINEDO, M. Scheduling: Theory Algorithms and Systems. Prentice-Hall, Englewood Cliffs, NJ. 2002.

QUADT, D., KUHN, H., A taxonomy of flexible flow line scheduling procedures. *European Journal of Operations Research*, 178, 686-698, 2007.

RACHAMADUGU, R., MORTON, V., A note on the weighted tardiness problem. *Operations Research*, 35, 450-452. 1982.

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

SEN, T., GUPTA S.K., A state-of-art survey of static scheduling research involving due dates. *OMEGA*, 12, 63-76, 1984.

SACCHI, L.H., Algoritmos Genéticos para minimização do *makespan* em um *flow shop* flexível. Dissertação de mestrado, UNICAMP, SP, 1997.

SANTOS, D. L., HUNSUCKER, J. L., DEAL, D. E., Global lower bound for flow shops with multiple processors. *European Journal of Operational Research*, 80, 112-120, 1995.

YANG, Y., Optimization and heuristic algorithms for flexible flow shop scheduling. Ph.D. dissertation, Columbia University, NY, 1998.

YANG, Y., KREIPL, S., PINEDO, M., Heuristics for minimizing total weighted tardiness in flexible flow shops. *Journal of Scheduling*, 3, 89-108, 2000.