UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

THAYS JOSYANE PERASSOLI BOIKO

Métodos heurísticos para a programação em *Flow Shop* Permutacional com tempos de *setup* separados dos tempos de processamento e independentes da seqüência de tarefas

São Carlos
2008

THAYS JOSYANE PERASSOLI BOIKO

Métodos heurísticos para a programação em *Flow Shop* Permutacional com tempos de *setup* separados dos tempos de processamento e independentes da seqüência de tarefas

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

Área de Concentração: Pesquisa Operacional
Orientador: Prof. Titular João Vitor Moccellin

São Carlos
2008

Ficha catalográfica preparada pela Seção de Tratamento da Informação do Serviço de Biblioteca – EESC/USP

Boiko, Thays Josyane Perassoli

B678m Métodos heurísticos para a programação em *flow shop* permutacional com tempos de *setup* separados dos tempos de processamento e independentes da seqüência de tarefas / Thays Josyane Pessaroli Boiko ; orientador João Vitor Moccellin . –- São Carlos, 2008.

Dissertação (Mestrado-Programa de Pós-Graduação em Engenharia de Produção e Área de Concentração em Pesquisa Operacional) -- Escola de Engenharia de São Carlos da Universidade de São Paulo.

1. Programação da produção. 2. *Flow Shop* permutacional. 3. Tempos de *Setup*. I. Título.

FOLHA DE JULGAMENTO

Candidata: Engenheira: **THAYS JOSYANE PERASSOLI BOIKO**

Dissertação defendida e julgada em 11/06/2008 perante a Comissão Julgadora:

APROVADA

Prof. Titular **JOÃO VITOR MOCCELLIN (Orientador)**
(Escola de Engenharia de São Carlos/USP)

Aprovada

Prof. Dr. **MARCELO SEIDO NAGANO**
(Escola de Engenharia de São Carlos/USP)

APROVADA

Profª. Drª. **FRANKLINA MARIA BRAGION DE TOLEDO**
(Instituto de Ciências Matemáticas e de Computação/USP)

Prof. Associado **REGINALDO TEIXEIRA COELHO**
Coordenador do Programa de Pós-Graduação em
Engenharia de Produção

Prof. Associado **GERALDO ROBERTO MARTINS DA COSTA**
Presidente da Comissão da Pós-Graduação da EESC

A toda a minha família, pois cada um, a sua maneira, têm me apoiado em todos os momentos da minha vida. Em especial à minha querida mãe *Leonice*, sem o esforço dela nada seria possível. Ao meu amor *Marlus*, que, nestes três anos de mestrado, virou meu marido, meu companheiro, psicólogo, massagista e que, mesmo assim, eu tive de deixar de lado, em muitos, muitos momentos. A Pofessora *Zueleide*, que foi a primeira a me mostrar o mundo acadêmico, das pesquisas, mas principalmente, porque o projeto do mestrado, foi um projeto que começamos a sonhar juntas, há muitos anos atrás. E, ao meu orientador, o Professor *Moccellin*, simplesmente, por tudo, por cada conselho, pela compreensão e pelos merecidos "puxões de orelha", dados sempre com uma delicadeza e sabedoria sem igual.

AGRADECIMENTOS

Muitas são as pessoas e instituições a agradecer neste momento. Pessoas e instituições que de alguma maneira contribuíram, ao longo de toda a minha formação pessoal e profissional, desde o tempo do ensino médio, passando pela graduação até esta conquista. No entanto, para citar todos esses nomes muitas páginas seriam utilizadas. Assim, aqui vou agradecer aquelas que contribuíram diretamente para a conclusão do meu mestrado.

Meu primeiro e mais especial agradecimento eu dedico ao Professor Moccellin, meu orientador, em primeiro lugar pela oportunidade a mim concedida, por cada um de seus preciosos conselhos, tanto profissionais quanto pessoais, pela confiança nos meus momentos de ausência, minha mais sincera gratidão.

A Universidade de São Paulo (USP), através do Departamento de Engenharia de Produção (DEP) e da Coordenação da Pós-Graduação da Escola de Engenharia de São Carlos (EESC), meus agradecimentos, pela oportunidade, por sua infra-estrutura e pela Bolsa do CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico), sem as quais não seria possível a conclusão desta pesquisa.

A todos os funcionários e professores do DEP, em especial ao Zé Luiz, pela constante atenção e, sobretudo, pela amizade, e ao Professor Fábio, pelas correções e sugestões feitas durante a qualificação.

A amiga, madrinha e colega de trabalho Márcia, por ter sido companheira de estudo para a seleção do mestrado, por entrarmos no mestrado e dividirmos o mesmo “ap”, por sermos da mesma linha de pesquisa, termos o mesmo orientador, feito as mesmas disciplinas, pela época de estudo do concurso, por termos sido companheiras de estrada, entregado a dissertação no mesmo dia, e, pelas, outras tantas coisas boas e ruins que compartilhamos.

Aos amigos feitos durante a época de mestrado Renata, Cristiane, Flávia, Rafael, Mariana e tantos outros, cada um com sua contribuição única e mais que especial. E também, aos amigos de Linha de Pesquisa e de Laboratório Hélio e Fábio, pelas ajudas, esclarecimentos, sugestões e pela companhia.

A Faculdade Estadual de Ciências de Campo Mourão, em especial a cada um dos meus colegas de trabalho do Departamento de Engenharia de Produção, pela paciência nestes últimos meses de finalização da dissertação.

Ao professor Roberto, pela ajuda com o programa Delphi.

A minha irmã Ligia, a minha prima Lohayne e ao meu cunhado Diego, pela ajuda nos dias de “transpiração”.

A banca examinadora deste trabalho, muito obrigada.

Agradecimentos especiais ao Professor Nagano, por cada um de seus esclarecimentos e conselhos e, principalmente, por sua tese de doutorado ter sido fonte de inspiração.

Obrigada a todos, serei sempre grata.

RESUMO

BOIKO, T. J. P. **Métodos heurísticos para a programação em *Flow Shop* Permutacional com tempos de *setup* separados dos tempos de processamento e independentes da seqüência de tarefas**. 2008. Dissertação (Mestrado) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2008.

Este trabalho dedica-se ao problema de programação em *Flow Shop* Permutacional com tempos de *setup* separados dos tempos de processamento e independentes da seqüência de execução das tarefas com o objetivo de minimizar a duração total da programação (*Makespan*). Por intermédio de investigações realizadas sobre as características estruturais do problema de programação e sua solução, uma propriedade deste problema é apresentada. Esta propriedade, denominada "Propriedade LBY", considerando quaisquer duas tarefas adjacentes J_u e J_v (J_u imediatamente precede J_v) independentemente de suas posições na seqüência de tarefas, fornece, um limitante inferior do tempo de espera para a tarefa J_v entre o fim do seu processamento na máquina M_k e o início do seu processamento na máquina seguinte. Dois novos métodos heurísticos são desenvolvidos, com base na propriedade apresentada e no procedimento de inserção de tarefas dos conhecidos métodos N&M e NEH: um construtivo, denominado BM_c; e, um melhorativo, denominado BM_m. Os métodos heurísticos propostos são comparados com os métodos heurísticos melhorativos de Cao; Bedworth (1992) e Rajendran; Ziegler (1997), através de um grande número de problemas gerados aleatoriamente. Os tempos de processamento são distribuídos no intervalo [1, 99] e os tempos de *setup* nos intervalos de [1, 49], [1, 99], [51, 149] e [101, 199]. Os métodos são avaliados quanto à porcentagem de sucesso em obter a melhor solução, ao desvio relativo médio e o tempo médio de computação. Os resultados da experimentação computacional mostram a qualidade do método construtivo BM_c e a melhor performance do método melhorativo BM_m. Estes resultados são apresentados e discutidos.

Palavras-chave: Programação da Produção. *Flow Shop* Permutacional. Tempos de *Setup*.

ABSTRACT

BOIKO, T. J. P. **Heuristic methods for the Permutation Flow Shop scheduling problem with separated, non-batch, and sequence-independent setup times**. 2008. Dissertation (Mastership) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2008.

This work addresses the Permutation Flow Shop scheduling problem with separated, non-batch, and sequence-independent setup times with the objective of minimizing the total time to complete the schedule (Makespan). Following an investigation of problem structural characteristics and your solution a property of this scheduling problem is presented. This property, denoted by "Property LBY", given any two adjacent jobs J_u e J_v (J_u immediately precedes J_v), regardless of their position in the sequence of jobs, provides an lower bound of the waiting time for job J_v between the end of its operations on the machine M_k and the beginning on machine $M_{(k+1)}$. Two news heuristics methods are development, on the basis of the presented property and in the job insertion procedure of the known methods named N&M and NEH: one constructive, denote by BM_c; and, one improvement, denote by BM_m. The proposed heuristics methods are compared with the improvement heuristics methods of Cao; Bedworth (1992) and Rajendran; Ziegler (1997), by a large number of randomly generated problems. The processing time are sampled from a distribution ranging from [1, 99] and, the setup times are sampled from distributions ranging from [1, 49], [1, 99], [51, 149] and [101, 199]. The methods are evaluated by the percentage of success in find the best solution, the average relative deviation and the average computation time. The results of the computational investigation show the quality of the constructive heuristic method BM_c and that the improvement heuristic method BM_c outperforms all others. These results are presented and discussed.

Keywords: Production Scheduling. Permutation Flow Shop. Setup Times.

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

$\sum C_j$	Soma das datas de término das tarefas (em inglês, *Total Completion Time*)
$\sum C_j/n$	Data média de término das tarefas
$\sum f_j$	Soma dos tempos de fluxo das tarefas (em inglês, *Total Flow Time*)
$\sum f_j/n$	Tempo médio de fluxo das tarefas
$\sum Tj$	Tempo total de atraso das tarefas (em inglês, *Total Tardiness*)
$\%S_h$	Porcentagem de sucesso de um método *h*
σ	$J_{[1]}, J_{[2]}, ..., J_{[j]}, ..., J_{[n]}$ = uma seqüência qualquer, entre as *n*! possíveis seqüências
Cj	Data de término da tarefa J_j (em inglês, *Completion Time*)
C_{max}	Duração Total da Programação (em inglês, *Makespan*)
dj	Data de entrega da tarefa J_j (em inglês, *due date*)
D	*Makespan* obtido
D_{BM}	*Makespan* obtido pelo método proposto, designado BM
D_{CB}	*Makespan* obtido pelo método CB
D_{hp}	*Makespan* obtido por um método *h*, em um determinado problema *p*
D_p*	Melhor *Makespan* obtido, para um determinado problema *p*, por todos os métodos analisados
D_{RZ1}	*Makespan* obtido pelo método RZ_1
D_{RZ2}	*Makespan* obtido pelo método RZ_2
F_j	Tempo de Fluxo da tarefa J_j (em inglês, *Flow Time*)
J_j	Tarefa *j* do conjunto de *n* tarefas

$J_{[j]}$	Tarefa que ocupa a j-ésima posição na seqüência σ
K	Número de estágios de produção
k_i	Número de máquinas no estágio *i* de produção
l_j	Data de liberação da tarefa J_j (em inglês, *release date*)
L_{max}	*Maximum Lateness*
m	Número de máquinas
M_k	Máquina *k* do conjunto de *m* máquinas
n	Número de tarefas a serem programadas
N_T	Número de tarefas em atraso
op_{kj}	Operação da tarefa J_j na máquina M_k
P	Número total de problemas resolvidos
p_{kj}	Tempo de processamento da tarefa J_j na máquina M_k
$p_{k[j]}$	Tempo de processamento da tarefa $J_{[j]}$ na máquina M_k
q_h	Quantidade de vezes em que o método *h* forneceu a melhor solução, isoladamente ou não
r_{kj}	Tempo de remoção da tarefa J_j da máquina M_k
$r_{k[j]}$	Tempo de remoção da tarefa $J_{[j]}$ da máquina M_k
s_{kj}	Tempo de *setup* da máquina M_k para a execução da tarefa J_j
$s_{k[j]}$	Tempo de *setup* da máquina M_k para a execução da tarefa $J_{[j]}$
$s_{k\,xy}$	Tempo de *setup* da máquina M_k para a execução da tarefa J_y, quando a tarefa J_x é sua precedente direta
T	Total de problemas analisados
T_j	Tempo de atraso da tarefa J_j (em inglês, *tardiness*)

t_{kj} Tempo de transferência da tarefa J_j entre as máquinas M_k e $M_{(k+1)}$

$t_{k[j]}$ Tempo de transferência da tarefa $J_{[j]}$ entre as máquinas M_k e $M_{(k+1)}$

TC_{hp} Tempo de computação de um determinado método *h*, em um determinado problema *p*

TMC_h Tempo médio de computação de um determinado método *h*

W_j Tempo total de espera da tarefa J_j

w_{kj} Tempo de espera da tarefa J_j após o fim de seu processamento na máquina M_k e o início de seu processamento na máquina $M_{(k+1)}$

[*x*] Determinado problema de uma classe em que um determinado método *h* obteve desvio relativo diferente de zero

$X_{[j]k}$ Intervalo de tempo entre o término da preparação da máquina M_k para a execução da $J_{[j]}$ e o início do processamento de $J_{[j]}$ nesta máquina, ou seja, é o tempo ocioso na máquina M_k entre o fim do *setup* de $J_{[j]}$ e o início do processamento desta tarefa

$X_{[x\ y]k}$ Intervalo de tempo na máquina M_k entre o término de uma atividade (*setup*/processamento) da tarefa J_x e o início de uma outra atividade (processamento da tarefa J_x /*setup* da tarefa J_y), quando a tarefa J_x é precedente direta da tarefa J_y

$Y_{[j]k}$ Intervalo de tempo entre o término da operação da tarefa $J_{[j]}$ na máquina M_k e o início da operação da mesma tarefa na máquina $M_{(k+1)}$

$Y_{[x\ y]k}$ Intervalo de tempo entre o término da atividade (*setup*/processamento) da tarefa J_y na máquina M_k e o início da atividade (*setup*/processamento) da mesma tarefa na máquina $M_{(k+1)}$, com a tarefa J_x precedendo diretamente a tarefa J_y

LISTA DE ABREVIATURAS E SIGLAS

B&B	*Branch & Bound*
BM	Boiko; Moccellin
BM_c	Boiko; Moccellin Construtivo
BM_m	Boiko; Moccellin Melhorativo
CB	Cao; Bedworth
DRM	Desvio relativo médio
EDD	*Earliest Due Date* (do inglês para o português, Primeira Data de Término)
F	*Flow Shop*
F2	*Flow Shop* com duas máquinas
F3	*Flow Shop* com três máquinas
F*m*	*Flow Shop* com *m* máquinas
G	Hipóteses sobre tarefas
H	Hipóteses sobre políticas de operação
IDp_{kj}	Intervalo de distribuição dos tempos de processamento
IDs_{kj}	Intervalo de distribuição dos tempos de *setup*
J	*Job Shop*
LB	*Lower Bound* (do inglês para o português, Limitante Inferior)
LBY	Limitante Inferior da variável $Y_{[j]k}$ (Tempo de Espera)
MSEO	Melhor seqüência até então obtida
N	Número de grupos (lotes) de tarefas a serem programadas
NP	Não Polinomial
O	*Open Shop*

P	Polinomial
PCP	Planejamento e Controle da Produção
PF	*Flow Shop* Permutacional
R	Hipóteses sobre máquinas
RZ	Rajendran; Ziegler
s.n-b.i.ST	*Separated, non-batch, and sequence-independent setup times* (do inglês para o português, tempos de *setup* separados dos tempos de processamento e independentes da seqüência de execução das tarefas)
TMC	Tempo médio de computação
UB	*Upper Bound* (do inglês para o português, Limitante Superior)

SUMÁRIO

1 INTRODUÇÃO

A produção pode ser definida como um processo, ou um conjunto de processos, pelos quais os produtos ou serviços são gerados. Desta maneira, entende-se por sistema de produção, de acordo com Monks (1987), a transformação de insumos (mão-de-obra, matérias-primas, capital, materiais, por exemplo) em produtos ou serviços, através de processos e operações. A responsabilidade pelo planejamento, execução e controle das atividades realizadas no sistema de produção, ou seja, pela operacionalização do sistema, é do Planejamento e Controle da Produção (PCP).

O PCP é definido, conforme Sá Motta In Machline (1972), como a função administrativa que tem por objetivo desenvolver os planos que irão orientar a produção, bem como realizar o controle da produção, com base nestes planos. De forma simplificada, pode-se dizer que, o PCP seria então a atividade de determinar "o que" vai ser produzido, "em qual quantidade" vai ser produzido, "como" e "onde" vai ser produzido, "quem" vai produzir e "quando" vai ser produzido. A partir desta definição pode-se perceber que o PCP, como destaca Zaccarelli (1979), possui três funções básicas: i) estoques; ii) planejamento do processo de produção; iii) programação e controle da produção. Estas funções do PCP são exercidas, segundo Artiba; Elmaghraby In Artiba; Elmaghraby (1979), nos níveis estratégico, tático e operacional de tomada de decisão. Estes níveis de tomada de decisão são refletidos no enfoque de Hax; Candea (1984) de hierarquia do planejamento da produção, também utilizada por Gaither; Frazier (2002). De acordo com esta hierarquia, o problema global de produção se inicia com um problema agregado de tipos de produtos, que são subseqüentemente desagregados em um problema de planejamento de família de produtos que, então, serão desagregados em nível de planejamento de itens individuais.

No nível de planejamento de itens individuais, com base em um plano de produção, previamente estabelecido, cabe à atividade de Programação da Produção, como afirmam Corrêa; Gianesi; Caon (2001), decidir "quais" atividades detalhadas (ordens, instruções de trabalho, ou seja, tarefas), "quando" (momento de início ou prioridade) e com quais recursos (máquinas[1]) devem ser realizadas, para que esse plano seja atendido.

1.1 PROGRAMAÇÃO DA PRODUÇÃO

A Programação da Produção pode ser definida, de maneira geral, de acordo com Baker (1974), como a alocação de recursos através do tempo. Esta definição geral leva a duas diferentes considerações. Primeiro, a Programação é uma função de tomada de decisão, ou seja, é o processo de determinar uma programação. Segundo, a Programação é um "corpo" de teoria, uma vez que está envolta em um grupo de princípios, modelos, técnicas e conclusões que promovem entradas para a função de programação.

Desta maneira, a essência das tomadas de decisão pela Programação, como ressalta Baker (1974), ocasiona: i) decisões de alocação de recursos; ii) decisões de seqüenciamento de tarefas.

Assim, segundo MacCarthy; Liu (1993), a Programação pode ser definida como a alocação de recursos através do tempo para a realização de tarefas, para melhor satisfazer um grupo de critérios pré-definidos, como acrescentam Yang; Liao (1999).

Um problema geral de programação pode ser entendido, então, conforme Taillard (1993), como um problema de n tarefas $\{J_1,J_2,... J_j, ..., J_n\}$ que devem ser processadas em m

[1] Neste trabalho, entende-se por recursos matérias-primas, mão-de-obra, ferramentas, equipamentos, centros de trabalho, então, usa-se genericamente o termo "máquinas".

máquinas $\{M_1, M_2, ..., M_k, ..., M_m\}$ que estão disponíveis. O processamento de uma tarefa J_j numa máquina M_k é chamado uma operação, designada de op_{kj}. Para cada operação op_{kj} existe um tempo de processamento p_{kj} associado. Para cada tarefa J_j deve existir uma *release date* (l_j)[2], data de liberação, a partir da qual a tarefa pode ser executada, e uma *due date* (dj), que corresponde à data em que a tarefa deve estar concluída, ou seja, data de entrega da tarefa. A programação neste contexto, conforme MacCarthy; Liu (1993), corresponde a uma designação de tarefas, através do tempo, às máquinas.

1.1.1 Classificação dos problemas de programação

Um problema de programação é especificado, segundo Maccarthy; Liu (1993), em termos das restrições tecnológicas do ambiente de produção onde as tarefas devem ser realizadas e dos objetivos da programação.

As restrições tecnológicas são determinadas principalmente pelo padrão de fluxo das tarefas nas máquinas. Desta maneira, os problemas de programação podem ser classificados conforme segue:

1) *Job Shop* (J): as tarefas têm um fluxo individual ou uma rota específica de processamento nas máquinas, existindo apenas uma máquina em cada estágio de produção;

[2] Também conhecida nas literaturas inglesas e portuguesas como *release time*, *ready time*.

2) *Flow Shop* (F): as tarefas têm um idêntico fluxo padrão, ou seja, as tarefas possuem o mesmo roteiro de processamento em todas as máquinas e o número de máquinas em cada estágio de produção é igual a um;
3) *Open Shop* (O): não existe fluxo padrão especificado para nenhuma tarefa e cada estágio de produção possui apenas uma máquina;
4) *Flow Shop* Permutacional: um *Flow Shop* em que a ordem de processamento das tarefas em todas as máquinas é a mesma;
5) Máquina Única: existe apenas um estágio de produção com uma única máquina disponível;
6) Máquinas Paralelas: existe mais de uma máquina disponível em um único estágio de produção, onde cada tarefa necessita de apenas uma destas máquinas;
7) *Job Shop* com máquinas múltiplas: um *Job Shop* no qual existem diversas máquinas paralelas em cada estágio de produção, sendo que cada tarefa é processada em apenas uma máquina em cada estágio de produção;
8) *Flow Shop* com máquinas múltiplas: um *Flow Shop* no qual existem diversas máquinas paralelas em cada estágio de produção, onde cada tarefa exige apenas uma máquina em cada estágio de produção.

Os ambientes de produção, onde ocorrem os problemas de programação, e seus relacionamentos podem ser visualizados na Figura 1.1, adaptada de MacCarthy; Liu (1993, p. 62).

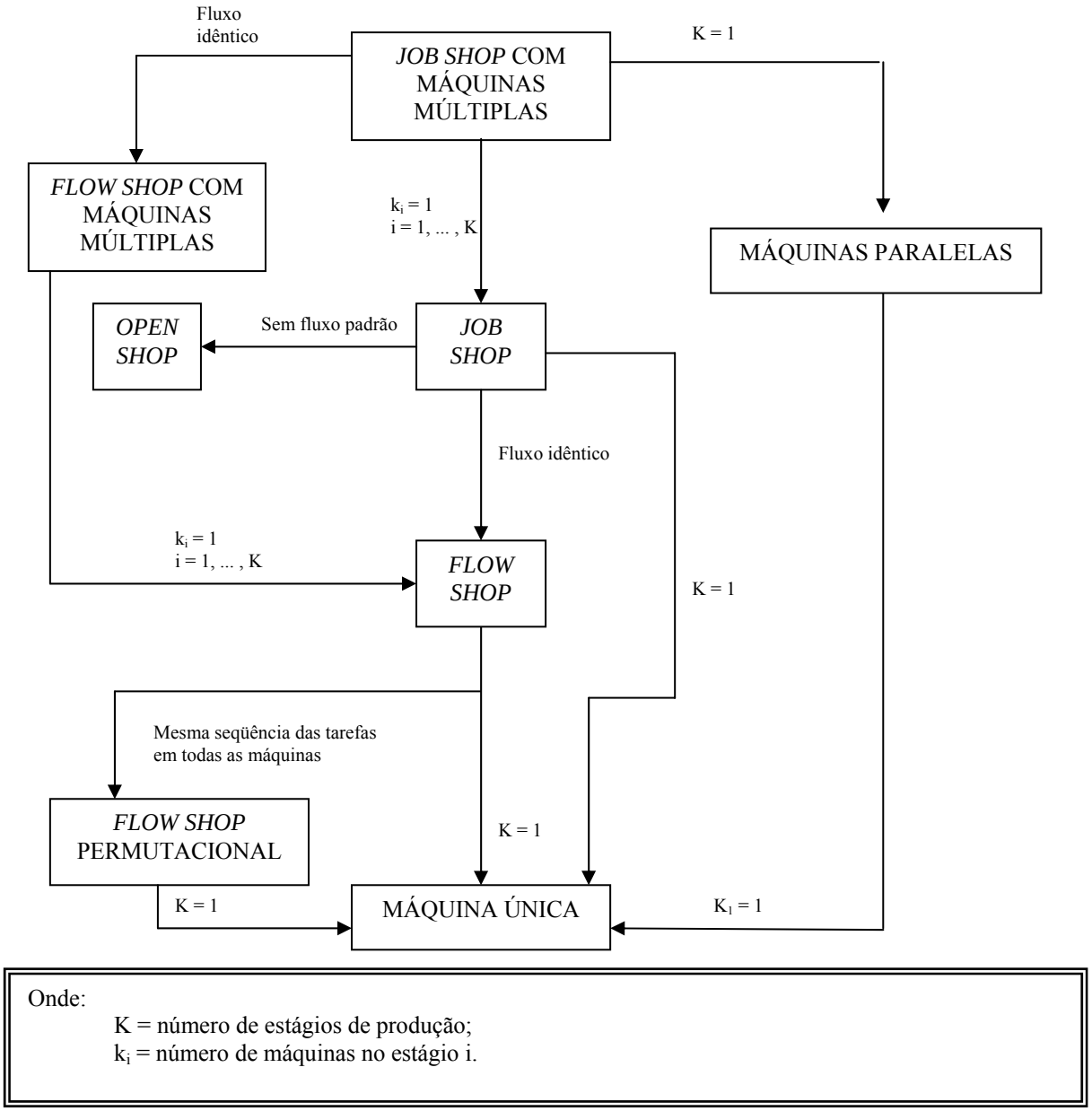

Figura 1.1. Ambientes de produção e seus relacionamentos

O problema de programação tratado neste trabalho é o *Flow Shop* Permutacional.

No *Flow Shop*, segundo Baker (1974), cada operação, depois da primeira, tem exatamente uma precedente direta e cada operação, antes da última, tem exatamente uma sucessora direta, como pode ser visualizado na Figura 1.2, adaptada de Baker (1974, p. 136), que representa uma determinada tarefa J_j.

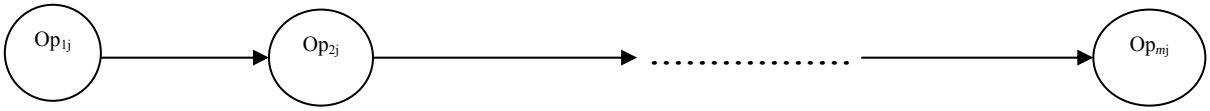

Figura 1.2. Estrutura de precedência linear para uma determinada tarefa J_j

Assim, cada tarefa no *Flow Shop* requer uma específica seqüência de operações a serem cumpridas para que a tarefa seja completada, assim pode-se afirmar, conforme Taillard (1993), que as operações de todas as *n* tarefas devem ser processadas nas *m* máquinas $\{M_1, M_2, ..., M_k, ..., M_m\}$, nesta ordem. Este tipo de estrutura pode ser chamado de estrutura linear de precedência. Os processos industriais químicos, de óleo e tintas são exemplos de *Flow Shop*, conforme citam Fogarty; Blackstone Jr.; Hoffmann (1983).

No *Flow Shop* Permutacional, de acordo com Taillard (1993) e Moccellin (1995), a ordem de processamento de todas as *n* tarefas nas máquinas é a mesma em todas as *m* máquinas do *Flow Shop*.

1.1.2 Ambientes de programação

A programação pode ser realizada em um ambiente estático ou em um ambiente dinâmico. Desta maneira podem existir sistemas estáticos de programação, bem como sistemas dinâmicos de programação. Nos sistemas estáticos, segundo Baker (1974), o grupo de tarefas disponíveis para a programação não muda o tempo todo. Enquanto que, nos sistemas dinâmicos, novas tarefas chegam durante todo o tempo.

1.1.3 Medidas de desempenho

Em cada um dos ambientes de produção, a programação pode ser realizada buscando atingir um objetivo ou critério diferente, as chamadas medidas de desempenho, que caracterizam a natureza do problema de programação.

As seguintes medidas de desempenho, listadas por MacCarthy; Liu (1993), são descritas matematicamente segundo French (1982) e explicadas segundo Bedworth; Bailey (1987):

- Tempo de Espera (w_{kj}): tempo de espera da operação op_{kj}, após ter sido completada a operação anterior $op_{(k-1)j}$. Quando k=1 (máquina M_1), w_{1j} é o tempo de espera a partir da data de liberação l_j para executar a primeira operação da tarefa J_j;
- Tempo Total de Espera (W_j): somatória dos tempos de espera da tarefa J_j desde a primeira operação até a última:

$$W_j = \sum_{k=1}^{m} w_{kj};$$

- Data de término da tarefa J_j ou, em inglês, *Completion Time* (C_j): corresponde ao período desde o início da programação na data zero até ao momento em que a tarefa J_j é finalizada. Ou Seja:

$$C_j = l_j + \sum_{k=1}^{m} \left(w_{kj} + p_{kj} \right);$$

- Tempo de Fluxo da tarefa J_j ou, em inglês, *Flow Time* (f_j): é o tempo entre o momento que a tarefa está disponível para ser processada e o momento em que ela é completada, correspondendo, então, ao tempo que a tarefa J_j permanece no centro de trabalho. Assim:

$$f_j = C_j - l_j \quad \text{ou} \quad f_j = \sum_{k=1}^{m} \left(w_{kj} + p_{kj} \right);$$

- *Lateness* (L_j): corresponde ao desvio entre o *Completion Time* da tarefa e sua *due date*. Uma tarefa pode ter um *lateness* positivo se for completada depois da *due date* e um *lateness* negativo se for completada antes. Então:

$$L_j = C_j - d_j;$$

- *Tardiness* (T_j): corresponde ao atraso na execução da tarefa, em relação à sua *due date*. Desta maneira:

$$T_j = \max\{L_j, 0\}.$$

Estas diferentes medidas de desempenho relacionam-se a três tipos de tomada de decisão que, segundo Baker (1974), normalmente prevalecem na programação: i) utilização eficiente dos recursos; ii) resposta rápida à demanda; iii) adaptação à prescritos *deadlines*, que correspondem ao prazo de entrega final de uma tarefa que, se for atingido, anula o processamento já realizado.

Assim, o Máximo *Completion Time*, também designado de *Makespan*, está relacionado à eficiente utilização dos recursos. O *Completion Time* Médio, o Tempo Médio de

Fluxo e o Tempo Médio de Espera, estão relacionados à rápida resposta à demanda. Enquanto que, o *Tardiness* Médio, o Máximo *Tardiness* e o número de tarefas em atraso (N_T) estão relacionados com a adaptação aos *deadlines*.

A medida de desempenho adotado neste trabalho é o *Makespan*, uma vez que, como declaram Bedworth; Bailey (1987), entre os muitos objetivos da programação, o mais obvio é aumentar a utilização dos recursos, que corresponde à redução do tempo ocioso.

O *Makespan*, no ambiente *Flow Shop*, pode ser definido como "... o intervalo de tempo entre o início de execução da primeira tarefa na primeira máquina e o término de execução da última tarefa na última máquina, ou seja, a duração total da programação (...)" (NAGANO, 1999, p. 12)

Para um grupo finito de tarefas, como ressaltam Bedworth; Bailey (1987), a utilização dos recursos é inversamente proporcional ao tempo necessário para realizar todas as tarefas. Este tempo refere-se ao *Makespan*. Assim, em um problema com um grupo finito de tarefas, a utilização dos recursos é melhorada através da programação deste grupo de tarefas, de tal maneira a reduzir o *Makespan*.

1.1.4 Métodos de solução

Muitos métodos têm sido desenvolvidos para resolver os problemas de programação. Esses métodos podem ser basicamente de dois tipos: métodos de solução ótima, e; métodos heurísticos. Os métodos de solução ótima geram uma programação ótima de acordo com a medida de desempenho adotada. Conforme MacCarthy; Liu (1993), o tempo despendido para encontrar a solução ótima é uma função polinomial das variáveis do problema (tarefas e

máquinas). Isto faz com que estes métodos sejam apenas eficientes computacionalmente em problemas de pequeno porte. Logo, como apresenta Fuchigami (2005), muitos métodos heurísticos, que, por sua vez, buscam alcançar uma solução próxima da solução ótima, tem sido propostos.

Neste trabalho, propõem-se dois métodos heurísticos para a solução do problema de programação considerado.

Um método heurístico pode ser entendido, de acordo com Fuchigami (2005), como um processo de solução de problemas que tem por base critérios racionais ou computacionais para escolher um caminho entre os vários possíveis, sem ter que percorrer todas as possibilidades, buscando atingir uma solução viável e não necessariamente ótima, mas próxima da ótima, com tempo de computação aceitável.

Os métodos heurísticos classificam-se de modo geral, conforme Souza; Moccellin (2000), em métodos construtivos e métodos melhorativos.

Nos métodos construtivos a seqüência adotada como solução do problema é obtida: i) diretamente a partir da ordenação das tarefas segundo índices de prioridade; ii) escolhendo-se a melhor seqüência de execução das tarefas a partir de um conjunto de seqüências previamente obtidas, como no método CDS de Campbell; Dudek; Smith (1970); iii) a partir da geração sucessiva de seqüências parciais (subseqüências) das tarefas até a ordenação de uma seqüência completa através de algum critério de inserção de tarefas, como no método NEH de Nawaz; Enscore Jr.; Ham (1983) e no método N&M de Nagano; Moccellin (2002).

Os métodos melhorativos, segundo Fuchigami (2005), a partir de uma solução inicial, através de algum procedimento interativo, buscam encontrar uma programação das tarefas melhor que a atual, quanto à medida de desempenho adotada. Os procedimentos interativos geralmente envolvem trocas de posições das tarefas na seqüência de processamento nas máquinas.

Neste trabalho é proposto um método heurístico construtivo e um método heurístico melhorativo.

1.1.5 Classes de problemas P e NP *hard*

Os problemas de programação podem ser, conforme sua complexidade, segundo Yang; Liao (1999), de duas classes: Polinomial (P); e, Não Polinomial (NP). A classe P representa os problemas que são eficientemente resolvidos e que os algoritmos possuem resolução em tempo polinomial. A classe NP, por sua vez, é essencialmente a classe de problemas cujos algoritmos possuem tempo de resolução em tempo exponencial.

A complexidade de um problema de programação pode ser aumentada, tornando-o um problema não básico, quando restrições adicionais são acrescentadas ao problema tradicional. Portman In Artiba; Elmaghraby (1997) coloca que restrições como tempos de *setup*, tempos de transferência e tempos de remoção separados dos tempos de processamento, existências de *deadlines* e de *release dates*, *breakdowns*[3], entre outras, tornam o problema de programação não básico.

Allahverdi; Soroush (2008) afirmam que, atualmente, a programação da produção com considerações de tempos ou custos de *setup* representam uma regra muito importante para os modernos ambientes de manufatura e serviços. No entanto, eles enfatizam que a vasta maioria da literatura existente sobre programação ignora esse fato. Desta maneira, neste trabalho considera-se a restrição adicional de tempos de *setup* separados dos tempos processamento ao problema básico de programação.

[3] Hipótese de não existir disponibilidade de máquina continuamente.

1.2 PROGRAMAÇÃO DE OPERAÇÕES EM *FLOW SHOP* ENVOLVENDO TEMPOS DE *SETUP* SEPARADOS DOS TEMPOS DE PROCESSAMENTO

Para se entender a programação de operações em *Flow Shop* cujos tempos de *setup* são separados dos tempos de processamento faz-se necessário entender o que vem a ser *setup* e sua classificação.

1.2.1 Definição e conceituação de setup

O *setup* corresponde, segundo Cao; Bedworth (1992) e Allahverdi; Gupta; Aldowaisan (1999), ao trabalho de preparar uma máquina ou um processo para processar uma tarefa. Isto inclui obter ferramentas, limpeza, recolocação de ferramental, posicionamento de acessórios, ajuste de ferramentas e inspeção de materiais.

Assim, conforme Su; Chou (2000), o tempo de *setup* (s_{kj}) é o intervalo de tempo requerido para preparar uma máquina M_k para processar uma tarefa J_j.

1.2.2 Classificação de setup

O *setup* pode ser classificado de acordo com à dependência da seqüência adotada, quanto a ser *setup* das máquinas para tarefas ou para lotes de tarefas e quanto à "antecipabilidade".

Quanto à dependência da seqüência, o *setup* é classificado, por Allahverdi; Gupta; Aldowaisan (1999), Yang; Liao (1999), Cheng; Gupta; Wang (2000) e Allahverdi et al. (2008), em:

- *setup* dependente da seqüência: quando a duração ou custo do *setup* depende tanto da tarefa atual (ou lotes de tarefas) a ser processada quanto da tarefa (ou lotes de tarefas) imediatamente precedente;
- *setup* independente da seqüência: se a duração ou custo do *setup* depende apenas da tarefa atual (ou lotes de tarefas) a ser processada.

As máquinas podem ser preparadas para processar tarefas ou para processar lotes de tarefas, logo, o *setup* é classificado a partir de diferentes terminologias por Allahverdi; Gupta; Aldowaisan (1999), Yang; Liao (1999), Cheng; Gupta; Wang (2000) e Allahverdi et al. (2008).

O *setup* das máquinas para tarefas é designado por Allahverdi; Gupta; Aldowaisan (1999) e por Allahverdi et al. (2008) de *non-batch setup*, por Yang; Liao (1999) e por Cheng; Gupta; Wang (2000) de *job setup*. Este *setup* ocorre quando uma determinada máquina, após completar o processamento de uma tarefa, precisa ser preparada para começar com o processamento de uma próxima tarefa.

A terminologia usada para o *setup* das máquinas para lotes de tarefas por Allahverdi; Gupta; Aldowaisan (1999) e por Allahverdi et al. (2008) é *batch setup*, enquanto que Yang; Liao (1999) designa este tipo de *setup* de *class setup* e Cheng; Gupta; Wang (2000) de *family setup*. O *setup* para lotes de tarefas ocorre quando as tarefas a serem processadas podem ser agrupadas em diferentes classes (lotes) conforme a similaridade de suas operações. Todas as tarefas em uma classe particular precisam do mesmo ou similar *setup* em cada estágio de

produção. Assim, o *setup* requerido para mudar de uma classe para a outra é chamado de *setup* para lotes de tarefas.

Segundo a "antecipabilidade", o *setup* é classificado, segundo Yang; Liao (1999), Cheng; Gupta; Wang (2000) e Allahverdi et al. (2008), em:

- *setup* antecipatório: antes de uma nova tarefa (ou lote de tarefas) estar disponível para ser processada em uma máquina o *setup* requerido por esta tarefa (ou lote de tarefas) na máquina pode ser completamente realizado, como é possível visualizar na Figura 1.3. Geralmente, o *setup* somente pode ser antecipado completamente quando a máquina está ociosa;
- *setup* não-antecipatório: o *setup* requerido por uma nova tarefa (ou lote de tarefas) somente pode ser realizado a partir do momento que a tarefa (ou lote de tarefas) está disponível para ser processada, como ilustra a Figura 1.4.

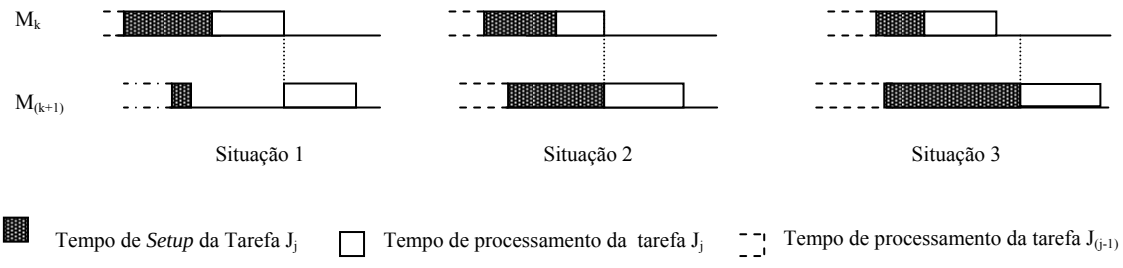

Figura 1.3. Exemplos de programações para uma determinada tarefa J_j com tempos de *setup* antecipatório

A Figura 1.3 representa três situações de programação para uma determinada tarefa J_j, quando o *setup* é antecipatório. Na situação 1 o *setup* da tarefa J_j é finalizado na máquina $M_{(k+1)}$ antes de a tarefa estar disponível para ser processada. Na situação 2 o *setup* é finalizado na máquina $M_{(k+1)}$ no mesmo momento em que a tarefa fica disponível para ser processada. Enquanto que, na situação 3, quando a tarefa J_j fica disponível para ser processada, o *setup* ainda não foi completado.

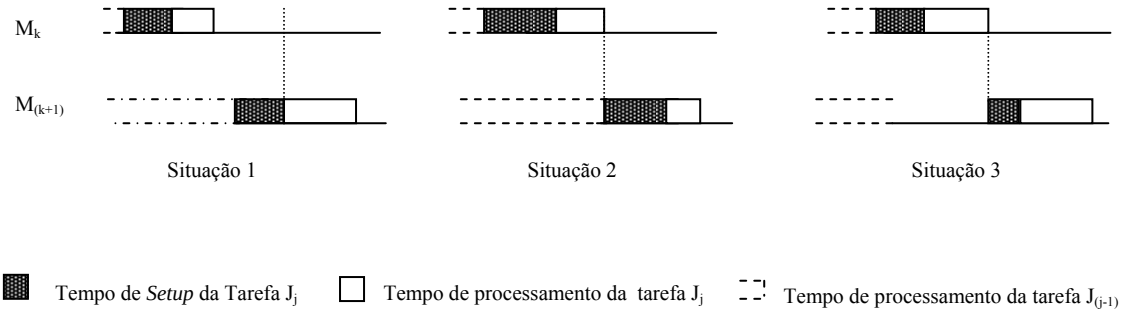

Figura 1.4. Exemplos de programações para uma determinada tarefa J_j com tempos de *setup* não-antecipatório

Três situações de programação para uma determinada tarefa J_j, quando o *setup* é não-antecipatório são representadas na Figura 1.4. Na situação 1 quando a tarefa J_j fica disponível para ser processada a máquina $M_{(k+1)}$ ainda está processamento uma tarefa anterior. Na situação 2 assim que a tarefa J_j fica disponível para ser processada a máquina também fica disponível. Enquanto que, na situação 3, a máquina $M_{(k+1)}$ fica disponível, uma vez que finalizou o processamento de uma tarefa anterior, antes de a tarefa Jj estar disponível para ser processada.

A necessidade de se considerar a "antecipabilidade" do *setup*, segundo Yang; Liao (1999), é realizada apenas no *Flow Shop* e no *Job Shop* ou sobre a suposição de existirem tarefas com *release dates* diferentes de zero.

Revisões de literatura sobre a programação de operações envolvendo considerações de *setup* das máquinas tanto para tarefas quanto para lote de tarefas são realizadas por Allahverdi; Gupta; Aldowaisan (1999), Yang; Liao (1999), Cheng; Gupta; Wang (2000) e Allahverdi et al. (2008).

Allahverdi; Gupta; Aldowaisan (1999) e Yang; Liao (1999) apresentam revisões das pesquisas realizadas em programação de operações nos ambientes de Máquinas Únicas, Máquinas Paralelas, *Flow Shop* e *Job Shop* com considerações de tempo e custo de *setup*.

Além desses ambientes, Allahverdi et al. (2008) considera, também, as pesquisas realizadas no ambiente *Open Shop*.

Enquanto que, Cheng; Gupta; Wang (2000) expõem apenas as pesquisas realizadas em *Flow Shop*.

O caso de *Flow Shop* com máquinas múltiplas com dois estágios de produção, com uma máquina no primeiro estágio e com o número de máquinas no segundo estágio sendo maior ou igual o número de tarefas, estudado por Gupta; Tunc (1994) é reportado por por Allahverdi; Gupta; Aldowaisan (1999), Yang; Liao (1999), Cheng; Gupta; Wang (2000) juntamente com as pesquisas realizadas em *Flow Shop*.

O tempo de *setup* das máquinas para as tarefas, independente da seqüência de tarefas e antecipatório será o tipo de tempo de *setup* considerado neste trabalho.

1.3 CONTEXTO DO TRABALHO

Este trabalho integra a Linha de Pesquisa designada “Pesquisa Operacional Aplicada aos Sistemas de Produção”, pertencente ao Programa de Pós-Graduação do Departamento de Engenharia de Produção da Escola de Engenharia de São Carlos – Universidade de São Paulo/SP.

O principal tema de pesquisa desta Linha é a “Programação de Operações em Sistemas de Operações Intermitentes”, que consiste no estudo de modelos e desenvolvimento de novos métodos de solução.

1.4 OBJETIVO DO TRABALHO

Neste contexto, este trabalho tem por objetivo principal investigar o problema de programação de operações em *Flow Shop* com *m* máquinas[4], com apenas a restrição adicional de tempos de *setup* das máquinas, para as tarefas, separados dos tempos de processamento, independentes da seqüência de execução das tarefas e antecipatórios, buscando propor novos métodos heurísticos para a minimização do *Makespan*.

1.5 ESTRUTURA DO TRABALHO

O texto está estruturado em seis capítulos e apêndices. Neste primeiro capítulo, faz-se uma introdução sobre a programação da produção, classificando-se os problemas de programação, apresentando-se os ambientes de programação, as medidas de desempenho da programação, os métodos de solução e a complexidade de tais problemas. Ainda na Introdução, apresenta-se uma breve justificativa de se considerar, neste trabalho, a restrição adicional de tempos de *setup* separados dos tempos processamento ao problema básico de programação, definindo, conceituando e classificando-se *setup*, em seguida.

No segundo capítulo, apresenta-se a revisão de literatura das pesquisas realizadas em programações de operações em *Flow Shop* envolvendo considerações de tempo de *setup* das máquinas para as tarefas independentes da seqüência de execução das tarefas.

[4] Como no *Flow Shop* (FS) o número de estágios de produção K é igual ao número de máquinas, uma vez que existe em cada estágio de produção apenas uma máquina, pode-se dizer FS com *m* máquinas, ao invés de FS com K estágios de produção.

Os métodos heurísticos melhorativos propostos por Cao; Bedworth (1992) e por Rajendran; Ziegler (1997) tem seus algoritmos detalhados no capítulo três, que também trás o delineamento das experimentações computacionais por eles realizadas.

O quarto capítulo consta da definição do problema, da apresentação de uma propriedade estrutural do problema considerado e da proposição dos novos métodos heurísticos.

A experimentação computacional está detalhada no quinto capítulo.

No sexto capítulo estão as conclusões do trabalho.

Os apêndices trazem os formatos dos arquivos de dados e de saída utilizados na experimentação computacional e o resumo do código fonte do *software* “PF/s.n-b.i.ST/Cmax”

Existem muitos termos técnicos em inglês, na área de Programação da Produção, que não possuem tradução fiel em português e a utilização de um significado aproximado poderia comprometer a sua interpretação. Assim, neste trabalho optou-se por manter alguns termos em inglês, cujos significados estão explicados em notas de rodapé.

Os termos que foram traduzidos, também têm suas definições apresentadas em notas de rodapé.

Em nota de rodapé, também, estão outras informações, consideradas relevantes para o bom entendimento do trabalho.

2 PROGRAMAÇÃO DE OPERAÇÕES EM *FLOW SHOP* (F) COM TEMPOS DE *SETUP* SEPARADOS DOS TEMPOS DE PROCESSAMENTO E INDEPENDENTES DA SEQÜÊNCIA DE EXECUÇÃO DAS TAREFAS

A idéia geral neste capítulo é apresentar a revisão de literatura das pesquisas realizadas em programações de operações em *Flow Shop* envolvendo considerações de tempo de *setup*, das máquinas para as tarefas, independentes da seqüência de execução das tarefas a partir de 1979, ano de publicação do primeiro artigo a tratar dos tempos de *setup* separados dos tempos de processamento.

Os trabalhos encontrados estão separados conforme o problema de programação e a medida de desempenho a que se dedicam, utilizando-se a terminologia adotada por Allahverdi et al. (2008).

2.1 *SETUP* NÃO-ANTECIPATÓRIO E ANTECIPATÓRIO

Os *setups* independentes da seqüência de execução das tarefas podem ser antecipatórios ou não-antecipatórios.

No entanto, como afirmam Cheng; Gupta; Wang (2000), se estes *setups* não puderem ser realizados antecipatoriamente para uma tarefa, o tempo ocioso das máquinas não pode ser usado. Neste caso, segundo Baker (1974), o problema de *setups* independentes da seqüência é automaticamente simplificado considerando-se o *setup* como parte do processamento, ou seja, os tempos de *setup* são considerados como parte dos tempos de processamento, ignorando-se os tempos de *setup*. Assim, de acordo com Cheng; Gupta; Wang (2000), o problema passa a ser formulado e resolvido com um problema de *Flow Shop* tradicional.

Então, a partir disto, sempre que o termo *setup* independente aparecer neste texto estará referindo-se ao *setup* independente da seqüência de execução das tarefas e antecipatório, uma vez que, o problema de programação considerado é o *Flow Shop* (F).

Assim, encontraram-se trabalhos envolvendo considerações de *setup* independente, onde o *setup* é tratado como separado do tempo de processamento, nos *Flow Shop* com duas (F2), três (F3) e *m* máquinas (F*m*).

2.2 *FLOW SHOP* COM DUAS MÁQUINAS (F2)

No *Flow Shop* com duas máquinas (F2) são econtrados trabalhos com medidas de desempenho de *Maximum Lateness* (L_{max}), Tempo de Fluxo (f_j), bicritério de Tempo Total de Fluxo ($\sum f_j$, sendo j = 1,2,..., *n*) e *Makespan* (C_{max}) e *Makespan* (C_{max}).

2.2.1 *Maximum Lateness* (L_{max})

O problema de *Flow Shop* com duas máquinas, com tempo de *setup* independente, cuja medida de desempenho é o *Maximum* Lateness (L_{max}), é tratado por Dileepan; Sen (1991), Allahverdi; Aldowaisan (1998), Allahverdi; Al-Anzi (2002), Allahverdi; Al-Anzi; Gupta (2005) Apud Allahvedi et al. (2008) e por Al-Anzi; Allahverdi (2006), como resume o Quadro 2.1.

AUTOR(S) E REFERÊNCIAS	RESTRIÇÕES ADICIONAIS	COMENTÁRIOS
Dileepan; Sen (1991)	------	- 1 algoritmo B&B: Pequenos problemas. - 2 heurísticas: 1. EDD (melhor); 2. Johnson.
Allahverdi; Aldowaisan (1998)	- Tempos de remoção.	- Extensão de Dileepan; Sen (1991).
Allahvedi; Al-Anzi (2002)	------	- Relações de dominância; - 4 heurísticas.
Allahvedi; Al-Anzi; Gupta (2005)	------	- Mesmo problema de Allahvedi; Al-Anzi (2002); - Algoritmo genético híbrido.
Al-Anzi; Allahvedi (2006)	------	- Proposição e composição de um método para gerar relações de dominância para qualquer problema de programação, a partir das relações estabelecidas por Allahvedi; Al-Anzi (2002).

Quadro 2.1. Resumo das Pesquisas em F2 com medida de L_{max}

Dileepan; Sen (1991) apresentam duas condições para a otimização de tal problema, que são utilizadas para desenvolver, juntamente com um LB, um procedimento B&B, que leva a solução ótima. Dois procedimentos heurísticos, baseados nas condições anteriormente mencionadas, também são desenvolvidos. Tais procedimentos heurísticos são denominados Heurística EDD[5] e Heurística de Johnson, assim chamada devido a similaridade da condição em que é baseada com a condição na qual a regra de Johnson tem por base. Os resultados ótimos obtidos através do procedimento B&B são comparados, durante a investigação computacional, com as soluções obtidas através dos dois procedimentos heurísticos. O procedimento B&B se mostra viável para problemas relativamente pequenos, enquanto que, a Heurística EDD, com melhor desempenho que a Heurística de Johnson, se mostra viável para problemas de maior porte.

Uma extensão do trabalho Dileepan; Sen (1991) é realizada por Allahverdi; Aldowaisan (1998), ao apresentarem relações de dominância que consideram os tempos de

[5] *Earliest Due Date* (do inglês para o português, Primeira Data de Término).

remoção, assim como os de *setup*, separados dos tempos de processamento. O *Completion Time* de uma tarefa é considerado como sendo baseado na disponibilidade da tarefa (*job-based*) e não na disponibilidade da máquina (*machine-based*). Isto significa que, o tempo de remoção não é considerado como parte do *Completion Time* da tarefa, uma vez que se considera que a tarefa está pronta para a próxima operação imediatamente depois de seu processamento estar completo. São demonstradas relações de dominância, que podem ser usadas em procedimentos B&B para encontrar a solução ótima ou em algumas heurísticas para obter uma boa solução, para os casos de *Flow Shop* Clássico[6] e de *Flow Shop Ordered*[7] do Tipo A[8] e do Tipo B[9].

Allahvedi; Al-Anzi (2002) mostram que o problema de programação de objetos de informação multimídia para aplicações WWW pode ser modelado com um *Flow Shop* de duas máquinas e tempos de *setup* independentes da seqüência, com medida de desempenho de *Maximum Lateness* (L_{max}). Assim, eles estabelecem relações de dominância e propõem quatro heurísticas que apresentam performance melhor do que as existentes para resolver o problema.

Allahvedi; Al-Anzi; Gupta (2005) Apud Allahverdi et al. (2008) apresentam um algoritmo genético híbrido e mostram que este apresenta melhor performance do que o proposto por Allahvedi; Al-Anzi (2002).

Al-Anzi; Allahvedi (2006) aplicam e analisam a performance das relações de dominância geradas, anteriormente, por Allahvedi; Al-Anzi (2002), para propor e compor um método para gerar relações de dominância para qualquer problema de programação.

---

[6] Também chamado de *Flow Shop* genérico. São os *Flow Shops* onde não existem relações entre os tempos de processamento nas diversas máquinas.
[7] São definidos como os *Flow Shops* onde existe, de alguma maneira, relações entre os tempos de processamento nas diversas máquinas.
[8] A somatória dos tempos de *setup*, remoção e processamento das tarefas na primeira máquina é maior que tal soma na segunda máquina, no caso de *Flow Shop* com duas máquinas.
[9] O contrário do Tipo A.

2.2.2 Tempo de Fluxo (f_j)

Estudos em *Flow Shop* com duas máquinas e tempos de *setup* independentes são realizados por Bagg*a*; Khurana (1986) Apud Allahverdi; Gupta; Aldowaisan (1999) e Cheng; Gupta; Wang (2000), por Allahverdi (2000), por Al-Anzi; Allahverdi (2001) e por Allahverdi; Aldowaisan (2002), conforme pode ser visualizado no Quadro 2.2.

É importante fazer duas observações: primeiro, quando as *release dates* (l_j) de todas as tarefas são consideradas igual a zero, o Tempo Total de Fluxo ($\sum f_j$) é igual ao *Completion Time* Total ($\sum C_j$), uma vez que, $f_j = C_j - r_j$, então, $f_j = C_j$; segundo, minimizar o Tempo Total de Fluxo, ou seja, a somatória dos Tempos de Fluxos de todas as tarefas ($\sum f_j$, sendo j = 1, 2, ..., *n*), equivale a minimizar o Tempo Médio de Fluxo ($\sum f_j/n$, sendo j = 1, 2,..., *n*).

AUTOR(S) E REFERÊNCIAS	RESTRIÇÕES ADICIONAIS	COMENTÁRIOS
Bagg*a*; Khurana (1986)	------	- *Completion Time* Médio = Tempo Médio de Fluxo ($\sum f_j/n$, sendo 1 = 1, 2, ..., *n*); - Algoritmo B&B.
Allahverdi (2000)	------	- Tempo Médio de Fluxo ($\sum f_j/n$, sendo j = 1, 2, ..., *n*); - 1 algoritmo B&B: ótimo em 2 casos especiais de F; - 3 heurísticas: problema genérico.
Al-Anzi; Allahverdi (2001)	------	- Tempo Total de Fluxo ($\sum f_j$, sendo 1 = 1, 2, ..., *n*); - 9 heurísticas.
Allahverdi; Aldowaisan (2002)	- Tempos de remoção.	- Mesmo problema considerado por Al-Anzi; Allahverdi (2001); - Soluções para casos especiais; - 1 relação de dominância, 1 LB e 1 algoritmo B&B: problema geral; - Diferentes heurísticas.

Quadro 2.2. Resumo das Pesquisas em F2 com medida de f_j

Bagg*a* & Khurana (1986) Apud Allahverdi; Gupta; Aldowaisan (1999) e Cheng; Gupta; Wang (2000) fornecem um algoritmo B&B para o problema de *Flow Shop* com duas máquinas, com tempos de *setup* independentes da seqüência, cuja medida de desempenho é o Tempo de Fluxo (f_j), mas especificamente o Tempo Médio de Fluxo ($\sum f_j/n$, sendo j = 1,2, ..., *n*). Bagga & Khurana (1986), de acordo com Cheng; Gupta; Wang (2000), consideram como medida o *Completion Time* Médio, que é equivalente ao Tempo Médio de Fluxo, quando as tarefas estão liberadas na data zero. Uma relação de dominância e um LB para o problema são desenvolvidos.

Enquanto que, Allahverdi (2000), apresenta o mesmo problema de Bagga & Khurana (1986), estabelecendo duas relações de dominância. Estas relações, juntamente com a desenvolvida por Bagga & Khurana (1986), são usadas em um algoritmo B&B, que leva a solução ótima em dois casos especiais de *Flow Shop – Flow Shop Semi-Ordered* do Tipo A e do Tipo B. Três heurísticas também são propostas para encontrarem soluções aproximadas para grandes problemas.

Al-Anzi; Allahverdi (2001) Apud Allahverdi et al. (2008) mostram que o problema de conectividade de *internet* em um banco de dados cliente-servidor, de três níveis, é equivalente ao problema de programação de operações em *Flow Shop* com duas máquinas, com tempo de *setup* separado do tempo de processamento com critério de minimização do Tempo Total de Fluxo ($\sum f_j$, sendo j = 1,2, ..., *n*). Nesse caso, uma máquina representa o servidor aplicativo e a outra o servidor do banco de dados. Assim, eles propõem nove heurísticas que apresentam melhor performance, ao resolver o problema, do que as propostas por Allahverdi (2000).

O mesmo problema tratado por Al-Anzi; Allahverdi (2001) é considerado por Allahverdi; Aldowaisan (2002) Apud Allahverdi et al. (2008), que consideram também o tempo de remoção como separado do tempo de processamento. Eles obêm soluções ótimas para casos especiais quando os tempos de processamento, *setup* e remoção satisfazem certas

condições. Eles também desenvolvem uma relação de dominância, um LB e um algoritmo B&B para o problema geral, além de diferentes heurísticas para o problema.

2.2.3 Bicritério: Tempo Total de Fluxo ($\sum f_j$) e *Makespan* (C_{max})

O problema de *Flow Shop* estático com duas máquinas, cujos tempos de *setup* são separados dos tempos de processamento e independentes, é estendido para o ambiente dinâmico por Su; Chou (2000), através do bicritério de Tempo Total de Fluxo ($\sum f_j$, sendo j = 1,2,..., *n*) e *Makespan* (C_{max}). O bicritério tem por objetivo, minimizar a soma dos pesos do Tempo Total de Fluxo e do *Makespan*. Um procedimento é proposto para transformar um problema de programação dinâmico, onde ocorrem reprogramações[10], em um problema de programação estática. Um Método de Programação Inteira é formulado, bem como um algoritmo heurístico, uma vez que, como afirmam Su; Chou (2000), um sistema de programação dinâmica requer resposta rápida. Um exemplo ilustrativo é apresentado.

2.2.4 *Makespan* (C_{max})

Os pioneiros em considerarem os tempos de *setup* separados dos tempos de processamento são Yoshida; Hitomi (1979) Apud Allahverdi; Gupta; Aldowaisan (1999), Yang; Liao (1999) e Cheng; Gupta; Wang (2000). Eles desenvolvem uma simples regra de

[10] Reprogramações ocorrem, segundo Su; Chou (2000), quando uma nova tarefa chega ao sistema de programação, uma tarefa é concluída e/ou máquinas quebram.

decisão, que é uma extensão do Algoritmo de Johnson (1954), para o caso onde os tempos de *setup* são separados dos tempos de processamento. Tal regra leva à solução ótima.

O Algoritmo de Yoshida; Hitomi (1979) é estendido por Sule (1982) e por Sule; Huang (1983) Apud Allahverdi; Gupta; Aldowaisan (1999) e Yang; Liao (1999), que consideram, além dos tempos de *setup*, os tempos de remoção (r_{kj})[11], como separados dos tempos de processamento, promovendo um algoritmo ótimo para o problema. Sule (1982) demonstra, segundo Cheng; Gupta; Wang (2000), que uma simples extensão da regra de Johnson (1954) fornece uma solução permutacional ótima para tal problema. Sule & Huang (1983) também consideram o problema de *Flow Shop* com 3 máquinas.

Utilizando os resultados existentes para os problemas de *Flow Shop* com tempos de *lag*[12], Szwarc (1983) generaliza estes resultados para descrever um problema de otimização, onde as *release dates*, os tempos de *setup* e de processamento são separados. Um *Lower Bound* (LB)[13] do *Completion Time* é desenvolvido, resolvendo-se alguns problemas clássicos de *Flow Shop* com duas máquinas e escolhendo a melhor permutação como uma solução aproximada para o modelo.

O problema considerado por Szwarc (1983) é estendido por Nabeshima; Maruyama (1984) Apud Allahverdi; Gupta; Aldowaisan (1999) e Cheng; Gupta; Wang (2000), que consideram, também, os tempos de remoção e os tempos de transferência (t_{kj})[14], como separados dos tempos de processamento.

---

[11] Em inglês *removal* ou *teardown time*. O tempo de remoção de uma tarefa J_j de uma máquina M_k corresponde ao tempo gasto pelas atividades realizadas após o processamento da tarefa na máquina, tais como, desengate de ferramentas, liberação de uma tarefa dos acessórios, retirada de ferramentas e de acessórios, inspeção e aferimento de ferramentas, retorno das ferramentas para a área de depósito e limpeza das máquinas e áreas adjacentes, ou seja, engloba as atividades necessárias para restaurar o estado inicial da máquina.

[12] Do inglês para o português, defasagem.

[13] Do inglês para o português, Limitante Inferior.

[14] Em inglês *transfer time*. Segundo Panwalkar (1991), o tempo de transferência de uma tarefa J_j entre as máquinas M_k e $M_{(k+1)}$ corresponde à soma de três parcelas, o tempo de retirar a tarefa da máquina M_k e coloca-la no 'veículo de transporte', mais o tempo de levar a tarefa até a próxima máquina, tempo este denominado de "tempo de viagem", mais o tempo de retirar a tarefa do 'veículo de transporte' e coloca-la na próxima máquina ou na linha de espera, ou seja, corresponde ao tempo de transferir a tarefa J_j entre as máquinas M_k e $M_{(k+1)}$.

Khurana; Bagga (1984) Apud Allahverdi; Gupta; Aldowaisan (1999) e Cheng; Gupta; Wang (2000) fornecem uma solução ótima para o problema sobre duas condições: primeira, o início e a parada dos *lags* são iguais para cada máquina; segundo, uma vez que o processamento de uma tarefa é maior (ou menor) na máquina M_1 que o tempo de processamento na máquina M_2, então a soma dos tempos de processamento e de *setup* desta tarefa na máquina M_1 é também maior (ou menor) do que esta soma na máquina M_2.

Khurana; Bagga (1985) Apud Allahverdi; Gupta; Aldowaisan (1999) e Cheng; Gupta; Wang (2000) obtêm solução ótima ao resolver o problema de Yoshida; Hitomi (1979) sujeito à restrição de *deadlines* para um grupo de tarefas.

Szwarc (1986) Apud Allahverdi; Gupta; Aldowaisan (1999) e Cheng; Gupta; Wang (2000) usa a fórmula por ele desenvolvida anteriormente (Szwarc, 1983) para encontrar uma solução ótima para o problema sem as duas condições adicionais de tempos de *lag* e *release dates*.

Logendran; Sriskandarjah (1993) tratam o problema de programação de N grupos de tarefas como um problema de programação de *n* tarefas, ou seja, cada grupo de tarefas é tratado como uma tarefa, existindo N grupos de tarefas a serem processadas nas máquinas M_1 e M_2. Dois fatores são incorporados ao problema para torná-lo mais real. Primeiro, uma tarefa em conclusão de seu processamento na máquina M_1 bloqueia a máquina se a máquina M_2 está ocupada processando uma tarefa, ou seja, até que seu *setup* se inicie na segunda máquina. Segundo, os *setups* nas máquinas M_1 e M_2 podem ser realizados antecipatoriamente à chegada de uma tarefa, desde que a máquina esteja livre. Logendran; Sriskandarjah (1993) demonstram que o problema é NP *hard*, através da análise de sua complexidade computacional, bem como analisam o caso de pior performance da heurística.

Allahverdi (1995) mostra que o algoritmo de Yoshida; Hitomi (1979), para problemas determinísticos, minimiza o *Makespan* quando *breakdowns* estocásticos estão presentes. Uma relação de dominância é estabelecida para minimização do *Makespan*.

Levner; Korgan; Maimon (1995) Apud Allahverdi et al. (2008) e Kogan; Levner (1998) se dedicam a um caso especial onde existem células automáticas de manufatura com robôs de transporte controlados por computador, designado por eles de *Robotic Flow Shop*. Eles criam uma extensão do Algoritmo de Johnson (1954) para resolver tal problema.

Para o mesmo caso tratado por Sule; Huang (1983), Allahverdi (1997) prova que o algoritmo de Yoshida; Hitomi (1979), para problemas estocásticos, minimiza o *Makespan* sobre certas condições de *breakdowns* aleatórios. Tais *breakdowns* podem ocorrer em três situações: enquanto as máquinas estão em operação; durante a realização do *setup* da máquina para uma tarefa; quando uma tarefa está sendo removida. As seguintes hipóteses são consideradas: o *setup* na segunda máquina é iniciado assim que a máquina esteja disponível; se um *breakdown* ocorre enquanto a operação de uma tarefa está sendo realizada, o trabalho já realizado não é perdido; a duração esperada do *breakdown* para um dado período é a mesma para ambas as máquinas M_1 e M_2.

Cheng, Wang & Sriskandarjah (1999) Apud Cheng; Gupta; Wang (2000) analisam o problema de *Flow Shop* com duas máquinas e um operador que realiza os *setups* e as operações de desmontagens em ambas as máquinas. Restringindo os movimentos do operador a um ciclo padrão, eles demonstram que problema, tendo como medida de desempenho o *Makespan*, é NP-hard, quando operações de desmontagens são consideradas. Duas heurísticas são propostas, juntamente com a análise da performance de seu pior desempenho.

O mesmo problema de Cheng, Wang & Sriskandarjah (1999) é estudado por Glass; Shafransky; Strusevich (2000), que provam que, sem a restrição adicional de tempos de remoção separados dos tempos de processamento, o problema é NP *hard*.

Um resumo dos trabalhos em *Flow Shop* com duas máquinas e critério de minimização do *Makespan* pode ser encontrado no Quadro 2.3.

AUTOR(S) E REFERÊNCIAS	RESTRIÇÕES ADICIONAIS	COMENTÁRIOS
Yoshida; Hitomi (1979)	------	- Pioneiros em considerar os tempos de *setup* como separado dos tempos de processamento. - Extensão do Algoritmo de Johnson (1954).
Sule (1982)	- Tempos de remoção.	- Extensão de Yoshida & Hitomi (1979).
Sule; Huang (1983)	- Tempos de remoção.	- Algoritmo heurístico.
Szwarc (1983)	- *Release dates*; - Tempos de *lags*.	------
Nabeshima; Maruyama (1984)	- Tempos de *lags*; - Tempos de remoção; - Tempos de transferência.	- Extensão de Szwarc (1983).
Khurana; Bagga (1984)	- Tempos de *lags*;	------
Khurana; Bagga (1985)	- *Deadlines* para grupos de tarefas.	------
Szwarc (1986)	------	- Extensão de Szwarc (1983).
Logendran; Sriskandarjah (1993)	- *Blocking* na primeira máquina.	- Algoritmo heurístico.
Allahverdi (1995)	- Tempos de remoção; - B*reakdowns* estocásticos.	- Problemas determinísticos; - Extensão de Yoshida & Hitomi (1979).
Levner; Korgan; Maimon (1995)	------	- *Robotic Flow Shop*; - Extensão do Algoritmo de Johnson (1954).
Allahverdi (1997)	- Tempos de remoção; - B*reakdowns* aleatórios.	- Mesmo problema de Sule; Huang (1983); - Problemas estocásticos; - Extensão de Yoshida & Hitomi (1979).
Kognan; Levner (1998)	------	- Mesmo problema e método de solução de Levner; Korgan; Maimon (1995)
Cheng; Wang; Sriskandarjah (1999)	- Tempos de remoção.	- Um operador realiza os *setups* e as operações de desmontagens.
Glass; Shafransky; Strusevich (2000)	------	- O mesmo problema de Cheng; Wang; Sriskandarjah (1999).

Quadro 2.3. Resumo das Pesquisas em F2 com medida de C_{max}

2.3 *FLOW SHOP* COM 3 MÁQUINAS (F3)

No *Flow Shop* com três máquinas destacam-se trabalhos cujas medidas de desempenho são o *Total* Completion *Time* ($\sum C_j$), o *Maximum Lateness* (L_{max}) e o Makespan.

2.3.1 *Total Completion Time* ($\sum C_j$)

Allahverdi; Al-Anzi (2006) tratam o problema de programação em um banco de dados distribuídos em três locais como um problema de programação em *Flow Shop* com três máquinas. O tempo de *setup* corresponde ao tempo de enviar o trabalho (operação de uma tarefa) realizado em um dado local (máquina) do banco de dados ao próximo local de processamento. Uma relação de dominância, um LB e um *Upper Bound*[15] (UB) são desenvolvidos. Um algoritmo B&B, que incorpora os *bounds*[16] e as relações supracitadas, é proposto. A efetividade do LB, do UB, das relações de dominância e do algoritmo B&B é investigada em exemplos de problemas gerados aleatoriamente.

[15] Do inglês para o português, Limitante Superior.
[16] Do inglês para o português, limitantes.

2.3.2 *Maximum Lateness* (L_{max})

Al-Anzi; Allahverdi (2007) desenvolvem uma relação de dominância e várias heurísticas para o caso de *Flow Shop* com três máquinas e medida de desempenho de *Maximum Lateness* (L_{max}).

2.3.3 *Makespan* (C_{max})

No *Flow Shop* com três máquinas e tempos de *setup* independentes como o objetivo de minimização do *Makespan* (C_{max}), são encontrados os trabalhos de Sule; Huang (1983) Apud Allahverdi; Gupta; Aldowaisan (1999), Yang; Liao (1999) e Cheng; Gupta; Wang (2000) e de Nabeshima; Maruyama (1984) Apud Allahverdi; Gupta; Aldowaisan (1999) e Cheng; Gupta; Wang (2000), como é possível visualizar no Quadro 2.4.

AUTOR(S) E REFERÊNCIAS	RESTRIÇÕES ADICIONAIS	COMENTÁRIOS
Sule; Huang (1983)	- Tempo de remoção.	- 2 heurísticas.
Nabeshima; Maruyama (1984)	- Tempo de remoção; - Tempos de *lags*.	- Extensão de Szwarc (1983).

Quadro 2.4. Resumo das Pesquisas em F3 com medida de C_{max}

Sule; Huang (1983) Apud Allahverdi; Gupta; Aldowaisan (1999), Yang; Liao (1999) e Cheng; Gupta; Wang (2000) ao considerarem o problema de *Flow Shop* com duas máquinas,

também fornecem dois algoritmos heurísticos para o problema com três máquinas. Os tempos de remoção, assim como os tempos de *setup*, também são considerados separados dos tempos de processamento.

De maneira adicional ao problema de *Flow Shop* com duas máquinas, tratado por Szwarc (1983), Nabeshima; Maruyama (1984) Apud Allahverdi; Gupta; Aldowaisan (1999) e Cheng; Gupta; Wang (2000) consideram o problema de três máquinas, sobre certas condições de tempo de remoção e de tempo de *lag*.

2.4 *FLOW SHOP* COM *m* MÁQUINAS (F*m*)

No caso de *Flow Shop* com *m* máquinas aparecem trabalhos com medidas de desempenho de *Maximum Lateness* (L_{max}), Completion Time (C_j) e *Makespan* (C_{max}).

2.4.1 *Maximum Lateness* (L_{max})

Fondrevelle et al. (2005) Apud Allahverdi et al. (2008) dedicam-se a um estudo do problema de programação em *Flow Shop* permutacional com tempos de *lags* exatos, cujo objetivo é minimizar o *Maximum Lateness* (L_{max}), onde o caso de tempos de *lags* negativos podem ser usados para o problema de *setup* independente da seqüência. Eles apresentam uma relação de dominância, um LB, um UB e um algoritmo B&B.

2.4.2 *Completion Time* (C_j)

Al-Anzi; Allahverdi (2005) apresentam uma heurística de evolução híbrida para a minimização da variância do *Completion Time* (C_j) no caso de programação em *Flow Shop* com *m* máquinas e tempos de *setup* separado do tempo de processamento e independente da seqüência. Tal heurística é por eles analisada computacionalmente

2.4.3 *Makespan* (C_{max})

No *Flow Shop* com *m* máquinas e tempos de *setup* independentes com o objetivo de minimização do *Makespan* (C_{max}) são encontrados vários trabalhos, conforme resume o Quadro 2.5.

AUTOR(S) E REFERÊNCIAS	RESTRIÇÕES ADICIONAIS	COMENTÁRIOS
Gupta (1972)	- Tempos de *lags*.	- Algoritmo de Otimização: viável para pequenos problemas.
Szwarc (1986)	- Tempos de remoção.	- Solução aproximada.
Szwarc; Gupta (1987)	- Tempos de remoção.	- Modelo de *setup* aditivo: o *setup* é soma de dois termos: 1. dependente apenas da tarefa, ou seja, independente da seqüência; 2. dependente da tarefa imediatamente seguinte na programação.
Park; Steudel (1991)	- *Buffer* limitado; - movimentações de materiais peça-por-peça.	- Células de trabalho com fluxos lineares, estruturadas segundo a Tecnologia de Grupo.
Proust; Gupta; Deschamps (1991)	- Tempos de remoção.	- 1 B&B: pequenos problemas; - 4 heurísticas: grandes problemas.
Cao; Bedworth (1992)	- Tempos de transferência diferente de zero.	- Heurística melhorativa.
Han; Dejax (1994)	- Tempos de remoção.	- Extensão de um dos algoritmos de Proust; Gupta; Deschamps (1991); - Cada máquina é considerada um gargalo.
Rajendran; Ziegler (1997)	- Tempos de remoção.	- Heurística melhorativa.
Brucker; Knust; Wang (2005)	- Operador único[17] para operar as *m* máquinas.	- Consideram, também, outras funções objetivos: $\sum C_j$; $\sum T_j$; L_{max}.

Quadro 2.5. Resumo das Pesquisas em F*m* com medida de C_{max}

Gupta (1972) Apud Allahverdi; Gupta; Aldowaisan (1999), Yang; Liao (1999) e Cheng; Gupta; Wang (2000) descreve condições de dominância e um algoritmo de otimização para a minimização do *Makespan* que, de modo geral, é suficiente para permitir tempos de *lags* de início e de parada. No entanto, como afirmam Allahverdi; Gupta; Aldowaisan (1999), o algoritmo proposto requer excessivo esforço computacional, além de não ser adequado para resolver problemas de grande porte.

Szwarc (1986) Apud Allahverdi; Gupta; Aldowaisan (1999) e Cheng; Gupta; Wang (2000) no mesmo artigo que considera o problema de *Flow Shop* com duas máquinas fornece uma solução aproximada para o problema de *m* máquinas.

[17] Em inglês, *single server*.

Szwarc; Gupta (1987) consideram um problema especial de *Flow Shop* com tempos de *setup* dependentes da seqüência, chamado de modelo de *setup* aditivo. Neste modelo cada *setup* é a soma de dois termos: um dependente apenas da tarefa, ou seja, independente da seqüência, e; outro dependente da tarefa imediatamente seguinte na programação. O modelo aditivo é comparado com o problema de *Flow Shop* cujos tempos de *setup* e de remoção são separados dos tempos de processamento.

Park; Steudel (1991) tratam do problema de programação permutacional em células de trabalho, com fluxos em linha, estruturadas segundo a Tecnologia de Grupo. No problema são consideradas as seguintes condições: tempos de *setup* separados dos tempos de processamento; *buffer*[18] limitado entre máquinas; movimentações de materiais peça-por-peça. Uma revisão dos algoritmos de seqüênciamento tratados na literatura possibilita identificar algoritmos apropriados para resolver o problema considerado. Quatro heurísticas existentes e três novas heurísticas são apresentadas e avaliadas em relação a pequenos e grandes problemas do "mundo real".

Os algoritmos B&B para os problemas de *Flow Shop* são estendidos, por Proust; Gupta; Deschamps (1991), para os problemas onde os tempos de *setup* e remoção são separados dos tempos de processamento, através da modificação dos LB para incluir os tempos de *setup* e de remoção. Como o algoritmo B&B não se apresenta muito eficiente em resolver problemas envolvendo um grande número de tarefas ou máquinas, quatro métodos heurísticos são propostos, também a partir dos resultados conhecidos para o problema de *Flow Shop* onde tempos de *setup* e de remoção são ignorados. A efetividade dos métodos propostos é avaliada empiricamente. Os resultados computacionais indicam que os métodos heurísticos propostos são bastante eficientes em minimizar o *Makespan* e podem ser usados

[18] Espaço para estocagem.

para encontrar uma solução inicial para os algoritmos B&B, objetivando aumentar a eficiência destes.

Cao; Bedworth (1992) apresentam o problema de *Flow Shop* com *m* máquinas com uma política de armazenagem entre estágios, tempos de transferência e de *setup* diferentes de zero, onde o tempo de transferência corresponde ao tempo de retirar a tarefa da máquina e colocá-la no recipiente transportador. Um método heurístico melhorativo, composto de três fases é proposto. Na primeira fase, uma seqüência inicial é gerada usando um procedimento similar ao Algoritmo CDS de Campbell; Dudek; Smith (1970). Na segunda fase, através de uma técnica de busca por ordenação, a seqüência atual é recursivamente melhorada. Na fase final, um procedimento de refinamento é usado para tentar encontrar melhores soluções. A performance da heurística proposta é avaliada através de problemas numéricos simulados. Um exemplo ilustrativo é apresentado.

Han; Dejax (1994) Apud Yang; Liao (1999) e Cheng; Gupta; Wang (2000) estendem um dos algoritmos de Proust; Gupta; Deschamps (1991) para gerar melhores programações ao considerarem cada estágio, ou seja, cada máquina, como um gargalo[19].

Rajendran; Ziegler (1997) desenvolvem três métodos heurísticos melhorativos para a programação em *Flow Shop* com *m* máquinas, onde os tempos de *setup*, remoção e processamento são separados. As heurísticas 1 e 2 consistem de duas fases. Na primeira fase uma seqüência é gerada, enquanto que na segunda-fase a seqüência é melhorada, através de uma técnica de inserção. Na heurística 1, na primeira fase aplica-se o procedimento da Heurística CDS de Campbell; Dudek; Smith (1970). A primeira fase da heurística 2 baseia-se nos grupos de tarefas identificados por Johnson (1954): i) grupos de tarefas que apresentam crescente ou constante tempo de processamento indo da máquina M_1 para a máquina M_2; ii) grupos de tarefas que apresentam decrescente tempo de processamento indo da máquina M_1

[19] Ponto do sistema de produção (máquina, transporte, área de estocagem, mão-de-obra, demanda, por exemplo) que, segundo TUBINO (2000), limita o fluxo de itens no sistema.

para a máquina M_2. A fase melhorativa é a mesma para as duas heurísticas. A heurística 3 pega a melhor seqüência, entre as duas seqüências fornecidas pelas heurísticas 1 e 2, e aplica novamente a fase de melhoramento das heurísticas 1 e 2. As heurísticas são avaliadas em comparação com a Heurística de Proust; Gupta; Deschamps (1991), apresentando melhor desempenho do que esta. Eles não afirmam, entre as heurísticas 1 e 2, qual apresentou o melhor desempenho.

Brucker; Knust; Wang (2005) se dedicam ao problema, aqui tratado, considerando que existe um único operador para operar as *m* máquinas do *Flow Shop*. Entre os casos por eles estudados eles consideraram também o caso de todos os tempos de *setup* serem os mesmos. Eles consideram, também, outras funções objetivos, além do *Makespan* (C_{max}), tais como *Total Completion Time* ($\sum C_j$), *Total Tardiness* ($\sum T_j$) e *Maximum Lateness* (L*max*), por exemplo.

2.5 CONSIDERAÇÕES

A partir do exame de literatura realizado, é possível observar que a grande parte dos trabalhos em *Flow Shop* com tempos de *setup* separados dos tempos de processamento e independentes da seqüência considera o ambiente com duas máquinas (F2), conforme é possível observar no Quadro 2.6, que relata a quantidade de trabalhos encontrados por ambiente e medida de desempenho.

	AMBIENTE DE PROGRAMAÇÃO			
MEDIDA DE DESEMPENHO	**F2**	**F3**	**F*m***	**TOTAL**
C_{max}	15	2	9	**26**
L_{max}	5	1	1	**7**
f_j	4	--	--	**4**
C_j	--	1	1	**2**
Bicritério: $\sum f_j$ e C_{max}	1	--	--	**1**
TOTAL	**25**	**4**	**11**	**40**

Quadro 2.6. Quantidade de trabalhos encontrados por ambiente e medida de desempenho

Ao todo foram encontrados 40 trabalhos, destes 25 consideram o *Flow Shop* com duas máquinas (F2), apenas 4 o *Flow Shop* com três máquinas (F3) e 11 consideram o *Flow Shop* com *m* máquinas (F*m*), como mostra o Quadro 2.6.

Dos 25 trabalhos encontrados no F2, 15 consideram como medidas de desempenho o *Makespan* (C_{max}), 5 o *Maximum Lateness* (L_{max}), 4 o Tempo de Fluxo (f_j) e 1 trabalho o bicritério de Tempo Total de Fluxo ($\sum f_j$) e *Makespan* (C_{max}).

No F3 foram encontrados 2 trabalhos tendo o *Makespan* (C_{max}) como medida de desempenho, 1 o *Maximum Lateness* (L_{max}) e 1 o *Total Completion Time* ($\sum C_j$).

Nos poucos trabalhos encontrados no ambiente F*m*, 9 trabalhos têm o *Makespan* (C_{max}) como medida de desempenho, 1 o *Maximum Lateness* (L_{max}) e 1 *Completion Time* (C_j).

É possível perceber, ainda, que dos 40 trabalhos a maioria considera como medida de desempenho o *Makespan* (C_{max}), uma vez que, foram encontrados 26 tendo essa medida, 7 o *Maximum Lateness* (L_{max}), 4 o Tempo de Fluxo (f_j) e 1 trabalho com bicritério de Tempo Total de Fluxo ($\sum f_j$) e *Makespan* (C_{max}).

Trabalhos tendo como medida de desempenho o *Makespan* (C_{max}) e o *Maximum Lateness* (L_{max}) aparecem em ambos os ambientes F2, F3 e F*m*.

Além disso, destaca-se o fato de não terem sido encontrados, no exame de literatura realizado, trabalhos em *Flow Shop* com *m* máquinas (F*m*) com apenas a restrição adicional de tempos de *setup* separados dos tempos de processamento e independentes da seqüência, tendo o *Makespan* (C_{max}) como medida de desempenho, que é o problema de programação tratado neste trabalho. Ambos os trabalhos apresentados, no F*m* tendo o *Makespan* (C_{max}) como medida de desempenho acrescentam alguma restrição adicional ao problema, além da restrição adicional de tempos de *setup* separados dos tempos de processamento, tais como tempos de transferência ou tempos de remoção separados dos tempos de processamento, operador único, por exemplo, como é possível visualizar no Quadro 2.5 (p. 49).

No entanto, os métodos heurísticos melhorativos propostos por Cao; Bedworth (1992) e Rajendran; Ziegler (1997), que, respectivamente, consideram os tempos de transferência e os tempos de remoção como separados dos tempos de processamento (assim como os tempos de *setup*), podem ser adaptados ao problema tratado neste trabalho (onde a única restrição adicional é a de os tempos de *setup* serem separados dos tempos de processamento), fazendo-se os tempos de transferência e de remoção iguais a zero. Desta maneira, tais heurísticas são comparadas aos métodos heurísticos propostos, como o objetivo de avaliar as performances destes, tendo, então, seus algoritmos descritos e o delineamento das experimentações computacionais por eles realizadas comentadas no Capítulo 3.

3 MÉTODOS HEURÍSTICOS EXISTENTES

Como observado anteriormente, os métodos heurísticos propostos por Cao; Bedworth (1992) e Rajendran; Ziegler (1997) podem ser comparados ao método a ser proposto, com o objetivo de avaliar o seu desempenho, ao se considerar os tempos de transferência e de remoção iguais a zero, desta forma os algoritmos dos métodos propostos por eles são aqui descritos e ilustrados. Os delineamentos das experimentações computacionais por eles realizadas são aqui descritos, uma vez que esses métodos são os que servem de base para avaliar a performance do método proposto.

3.1 CAO; BEDWORTH (1992)

O método proposto por Cao; Bedworth (1992), aqui denominado método CB, é um método heurístico melhorativo composto de três fases.

O *Makespan* da solução encontrada pelo método CB será designado de D_{CB}.

3.1.1 Descrição do Algoritmo

O método CB pode ser detalhado conforme o seguinte algoritmo, utilizando-se a seguinte notação, e não a notação original, onde p_{kj}, t_{kj} e s_{kj} são respectivamente os tempos de processamento, transferência e de *setup* na máquina M_k da tarefa J_j, sendo j = 1,2, ..., *n* e k = 1, 2, ..., *m*.

Fase 1 - Geração da seqüência inicial

Passo 1: Forme um sub-problema de duas máquinas com os seguintes tempos de processamento artificiais para a tarefa J_j:

$$p^*_{1j} = p_{1j} + t_{1j} \quad \text{e} \quad p^*_{2j} = p_{mj} + t_{mj}$$

Passo 2: Entre as tarefas a serem programadas, encontre a tarefa que tem o mínimo valor de todos os p^*_{1j} e p^*_{2j}, que são os tempos de processamento da tarefa J_j na primeira e na segunda máquina no sub-problema.

Passo 3: **Se** o tempo de processamento mínimo está na máquina M_1, designe a tarefa J_j para a próxima posição disponível no começo da seqüência. Vá para o passo 5.

Passo 4: **Se** o tempo de processamento mínimo está na máquina M_2, designe a tarefa J_j para a próxima posição disponível no fim da seqüência. Vá para o passo 5.

Passo 5: Remova a tarefa designada da lista de tarefas a serem programadas e retorne ao passo 2 até que todas as tarefas sejam programadas.

[Dois sub-passos devem ser realizados, após todas as tarefas serem programadas, embora não constem no algoritmo de Cao; Bedworth (1992): i) a seqüência inicial deve passar a ser designada melhor seqüência até então obtida (MSEO); ii) deve-se calcular o valor do *Makespan*.]

Fase 2 – Busca para re-ordenação das tarefas

Passo 1: Faça j=1, k=1 e i=1. Re-numere as tarefas de acordo com suas posições na melhor seqüência até então obtida (MSEO), de modo que a tarefa j_* é a j-ésima tarefa na seqüência.

Passo 2: Calcule o tempo de processamento de cada tarefa no i-ésimo subproblema, como:

$$p_{1j^*[i]} = \sum_{k=1}^{i} (p_{kj^*} + t_{kj^*} + s_{k(j+1)^*}) \quad \text{e} \quad p_{2j^*[i]} = \sum_{k=m-i+1}^{m} (p_{kj^*} + t_{kj^*} + s_{k(j+1)^*})$$

Passo 3: Se k=1, entre as tarefas a serem ordenadas, encontre a tarefa que tem o mínimo valor de todos $p_{1j^*[i]}$, e troque esta tarefa com a tarefa j_*. Vá para o passo 5.

Passo 4: Se k=2, entre as tarefas a serem ordenadas, encontre a tarefa que tem o mínimo valor de todos os $p_{2j^*[i]}$, e troque esta tarefa com a tarefa j_*. Vá para o passo 6.

Passo 5: Remova a tarefa que foi trocada com a tarefa j_* da lista de tarefas a serem ordenadas. Calcule o *Makespan* da nova seqüência. **Se** o *Makespan* é menor do que o *Makespan* da MSEO, salve a nova seqüência e seu *Makespan*. Faça j=j+1. Vá para o passo 3. **Se** j=*n*, faça k=k+1 e a mais recente seqüência salva ser a MSEO. Re-numere as tarefas de acordo com suas posições na MSEO, de modo que a tarefa j_* é a j-ésima tarefa na seqüência. Vá para o passo 4.

Passo 6: Remova a tarefa que foi trocada com a tarefa j_* da lista de tarefas a serem ordenadas. Calcule o *Makespan* da nova seqüência. **Se** o *Makespan* é menor que o *Makespan* da MSEO, salve a nova seqüência e seu *Makespan*. Faça j=j-1. Vá para o passo 4. **S**e j=1, faça i=i+1 e a mais recente seqüência salva ser a MSEO. Re-numere as tarefas de acordo com suas posições na MSEO, de modo que a tarefa j_* é a j-ésima tarefa na seqüência. Faça k=1 e vá para o passo 2. **Se**, i=*m*, pare. A mais recente seqüência salva, ou seja, a MSEO, é a seqüência a ser implementada.

[A seqüência obtida como solução da Fase 2 deve passar a ser denominada melhor seqüência até então obtida (MSEO), embora isto não conste no algoritmo de Cao; Bedworth (1992)]

Fase 3: Procedimento de Refinamento

Passo 1: Faça $i_1=n$, $i_2=i_1-1$, e FLAG=0.

Passo 2: Re-numere as tarefas de acordo com suas posições na melhor seqüência até então obtida (MSEO), de modo que a tarefa j_* é a j-ésima tarefa na seqüência. Forme um sub-problema de duas máquinas com os seguintes tempos de processamento artificiais para a tarefa J_{j*}:

$$p^*_{1j^*} = \sum_{k=1}^{m-1} (p_{kj^*} + t_{kj^*} + s_{k\,(j+1)^*}) \quad \text{e} \quad p^*_{2j^*} = \sum_{k=2}^{m} (p_{kj^*} + t_{kj^*} + s_{k\,(j+1)^*})$$

Passo 3: Se $p^*_{1\,i_1^*} \leq p^*_{1\,i_2^*}$, então troque a tarefa i_1^* com a tarefa i_2^*. Faça FLAG igual a 1, e calcule o *Makespan* da nova seqüência. **S**e o *Makespan* é menor que o *Makespan* da MSEO, salve a nova seqüência e seu *Makespan*.

Passo 4: Se $p^*_{2\,i_1^*} \geq p^*_{2\,i_2^*}$, e FLAG=0, então troque a tarefa i_1^* com a tarefa i_2^*. Calcule o *Makespan* da nova seqüência. **Se** o *Makespan* é menor que o *Makespan* da MSEO, salve a nova seqüência e seu *Makespan*.

Passo 5: Troque a tarefa i_1^* com a tarefa i_2^* [20]. Faça $i_2=i_2-1$, e FLAG=0. Vá para o passo 2. **Se** $i_2=0$, faça $i_1=i_1-1$, $i_2=i_1-1$, FLAG=0 e a seqüência mais recentemente salva ser a MSEO. Vá para o passo 2. **Se** $i_1=1$, pare. A seqüência mais recentemente salva, ou seja, a MSEO, é a solução do problema.

[20] Embora não conste no algoritmo, conforme o exemplo ilustrado por Cao; Bedworth (1992), esta troca só deve ser realizada caso ocorra anteriormente, no Passo 3 ou no Passo 4, alguma troca de posição das tarefas. Uma vez que, esta troca, no Passo 5, tem por objetivo voltar à MSEO inicial, que serve de base para a criação de novas seqüências, através de trocas de posições das tarefas.

3.1.2 Delineamento da Experimentação Computacional

Cao; Bedworth (1992) testam, em sua experimentação computacional, 252 problemas, gerados aleatoriamente, divididos em 12 classes de problemas, de acordo com o número de tarefas (n) e o número de máquinas (m). Essas classes são divididas em dois grupos de problemas, por eles designados problemas de tamanho pequeno e problemas de tamanho grande.

O grupo de problemas de tamanho pequeno é constituída de 4 classes de problemas com $n \in \{4, 6\}$ e $m \in \{4, 12\}$. São testados 144 problemas neste grupo.

O grupo de problemas de tamanho grande, por sua vez, é constituído de 9 classes de problemas com $n \in \{10, 20, 30\}$ e $m \in \{4, 8, 12\}$. Neste grupo são testados 108 problemas.

Os tempos de processamento, *setup* e transferência são números inteiros, uniformemente distribuídos, gerados aleatoriamente. Cao; Bedworth (1992) não especificam o intervalo de distribuição dos tempos.

Cao; Bedworth (1992) analisam o método por eles proposto em relação ao tempo computacional médio e a desvio relativo médio do *Makespan*.

É importante salientar que ao realizarem a experimentação computacioanal com o grupo de problemas de tamanho grande, Cao; Bedworth (1992) desconsideram os tempos de *setup* e remoção.

3.2 RAJENDRAN; ZIEGLER (1997)

Três métodos heurísticos melhorativos são propostos por Rajendran; Ziegler (1997), conforme apresentado anteriormente no Capítulo 2.

As heurísticas 1 e 2, aqui denominadas de método RZ_1 e método RZ_2, respectivamente, consistem de duas fases. Na primeira fase gera-se uma seqüência que é, então, melhorada na segunda fase. Os métodos RZ_1 e RZ_2 diferem apenas na primeira fase, sendo a fase melhorativa a mesma, para ambos os métodos.

A heurística 3, aqui denominada de método RZ_3, seleciona a melhor seqüência, entre as duas seqüências fornecidas pelos métodos RZ_1 e RZ_2, e aplica novamente a fase de melhoramento dos métodos RZ_1 e RZ_2. Assim, entre as heurísticas propostas por Rajendran; Ziegler (1997), somente a heurística RZ_3 é comparada com o método proposto.

Os *Makespans* das soluções encontradas pelos métodos RZ_1, RZ_2 e RZ_3 serão designados, respectivamente de D_{RZ1}, D_{RZ2} e D_{RZ3}.

3.2.1 Descrição dos Algoritmos

Os métodos RZ podem ser descritos utilizando-se a notação deste trabalho e não a notação original, onde p_{kj}, r_{kj} e s_{kj} são respectivamente os tempos de processamento, remoção e *setup* na máquina M_k da tarefa J_j, sendo j = 1, 2, ..., *n* e k = 1, 2, ..., *m*.

ALGORITMO DO MÉTODO RZ_1 E ALGORITMO DO MÉTODO RZ_2

MÉTODO RZ_1

Fase 1 – Obtenção da seqüência "semente"

Passo 1: Faça i=1 e $p'_{kj} = p_{kj} + s_{kj} + r_{kj,}$ para cada k = 1,2, ..., *m* e j = 1,2, ..., *n*.

Passo 2: Para cada tarefa J_j calcule:

$$P_{1j} = \sum_{k=1}^{i} p'_{kj} \quad [1] \qquad \text{e} \qquad P_{2j} = \sum_{k=m-i+1}^{m} p'_{kj} \quad [2]$$

Passo 3: Resolva o problema artificial de duas máquinas definido pelas expressões [1] e [2] (considerando P_{1j} e P_{2j} os tempos de processamento da tarefa J_j nas duas máquinas) usando o algoritmo de Johnson (1954). Faça $\sum^i$ ser a seqüência então obtida. Calcule o *Makespan* no problema dado de m máquinas usando os valores originais de p_{kj}, r_{kj} e $s_{kj.}$ Faça o *Makespan* ser designado por Q^i.

Passo 4: Faça i=i+1. Retorne para o passo 2 **se** i<*m*; **senão** prossiga para o passo 5.

Passo 5: Faça i=1 e $p''_{kj} = p_{kj} + r_{kj} - r_{(k-1)\,j}$, para cada k= 1,2, ..., *m* e j = 1,2, ..., *n*.

Passo 6: Para cada tarefa J_j calcule:

$$P_{1j} = \sum_{k=1}^{i} p''_{kj} \quad [3] \qquad \text{e} \qquad P_{2j} = \sum_{k=m-i+1}^{m} p''_{kj} \quad [4]$$

Passo 7: Resolva o problema artificial de duas máquinas definido pelas expressões [3] e [4] (considerando P_{1j} e P_{2j} os tempos de processamento da tarefa J_j nas duas máquinas) usando o algoritmo de Johnson (1954). Faça $\sum^{m-1+i}$ ser a seqüência então obtida. Calcule o *Makespan* no problema dado de m máquinas considerando os valores originais de p_{kj}, r_{kj} e $s_{kj.}$ Faça o *Makespan* ser designado por Q^{m-1+i}.

Passo 8: Faça i=i+1. Retorne para o passo 6 **se** i<*m*; **senão** prossiga para o passo 9.

Passo 9: Determine a seqüência $\sum^s$ tal que $Q^s = \min \{Q^i \mid i = 1, 2, ..., 2m\text{-}2\}$.

Passo 10: A seqüência $\sum^s$ obtida no passo 9 torna-se a seqüência "semente" para ser dada como entrada para a fase de melhoramento.

MÉTODO RZ_2

Fase 1 – Obtenção da seqüência "semente"

Passo 1: Faça $p'_{kj} = p_{kj} + s_{kj} + r_{kj}$, para j = 1, 2, ..., *n* e k = 1, 2, ..., *m*.

Passo 2: Para cada tarefa J_j calcule:

$$\tau_j = (\sum_{k=1}^{m} k \times p'_{kj}) / (\sum_{k=1}^{m} p'_{kj})$$

Passo 3: Para cada tarefa J_j,

Se $\tau_j \geq (m+1)/2$,

então, designe a tarefa J_j para o grupo de tarefas 1;

senão, para o grupo de tarefas 2.

Passo 4: Ordene as tarefas do grupo de tarefas 1 em ordem ascendente de $\sum_{k=1}^{m} p'_{kj}$ e as tarefas do grupo de tarefas 2 em ordem descendente de $\sum_{k=1}^{m} p'_{kj}$.

Passo 5: Anexe o ordenado grupo de tarefas 2 ao ordenado grupo de tarefas 1 para obter a seqüência completa $\sum^1$.

Passo 6: Faça $p''_{kj} = p_{kj} + r_{kj} - r_{(k-1)j}$, para j = 1, 2, ..., *n* e k = 1, 2, ..., *m*.

Passo 7: Repita os passos 2 ao 5 usando p''_{kj} ao invés de $p`_{kj}$. Chame a seqüência completa então obtida de $\sum^2$.

Passo 8: Se o *Makespan* de $\sum^1$ é menor que o de $\sum^2$, então faça$\sum^s = \sum^1$; **senão** $\sum^s = \sum^2$.

Passo 9: A seqüência $\sum^s$ obtida no passo 8 torna-se a seqüência "semente" para ser dada como entrada para a fase de melhoramento.

MÉTODO RZ_1 e **MÉTODO RZ_2**

Fase 2 – Melhoramento (Onde [x] representa a tarefa encontrada na x-ésima posição da seqüência $\sum^s$ e [y] representa a tarefa encontrada na y-ésima posição da seqüência $\sum^b$.)

Passo 1: Faça x=1 e $\sum^b = \sum^s$.

Passo 2: Faça y=1.

Passo 3: Se [x] ≠ [y], então gere uma seqüência $\sum^y$ que difere da seqüência $\sum^b$ apenas por ter movido a tarefa [x] para a y-ésima posição e calcule o *Makespan* Q^y da seqüência $\sum^y$; **senão** faça y'=y e prossiga para o passo 4.

Passo 4: Faça y=y+1. Retorne para o passo 3 **se** y ≤ *n*; **senão** prossiga para o passo 5.

Passo 5: Determine a seqüência $\sum^i$ tal que $Q^i = \min \{Q^y \mid y = 1, 2, ..., n \text{ e } y \neq y'\}$. **Se** $Q^i < Q^b$, faça $\sum^b = \sum^i$.

Passo 6: Faça x=x+1. Retorne para o passo 2 **se** x ≤ *n*; **senão** prossiga para o passo 7.

Passo 7: $\sum^b$ é a seqüência obtida com a heurística. PARE.

Rajendran; Ziegler (1997) não apresentam um algoritmo para o método RZ_3, no entanto, aqui ele é descrito em forma de algoritmo, como segue.

ALGORITMO DO MÉTODO RZ_3

MÉTODO RZ_3

Fase 1 – Obtenção da seqüência semente.

Passo 1: Se o *Makespan* da seqüência final obtida com RZ_1 for menor que o *Makespan* da seqüência final obtida com RZ_2, então a seqüência semente é a seqüência obtida com RZ_1. **Senão**, a seqüência semente é a seqüência obtida com RZ_2.

Fase 2 – Melhoramento

(Mesmos passos da fase de melhoramento dos métodos RZ_1 e RZ_2)

Após a descrição do algoritmo do método RZ_3 é possível perceber que, entre os métodos RZ, esse sempre leva ao melhor resultado, o que justifica apenas comparar o método RZ_3 com os métodos propostos neste trabalho.

3.2.1 Delineamento da Experimentação Computacional

Um total de 3000 problemas, gerados aleatoriamente, são testados por Rajendran; Ziegler (1997), que os divide em 25 classes de problemas, de acordo com o número de tarefas (n) e o número de máquinas (m). Essas 25 classes de problemas são divididas por eles em dois grupos de problemas, designados problemas de tamanho pequeno e problemas de tamanho grande.

O grupo de problemas de tamanho pequeno constitui-se de 10 classes de problemas com $n \in \{6, 7\}$ e $m \in \{5, 10, 15, 20, 25\}$. Neste grupo são testados 1200 problemas.

O grupo de problemas de tamanho grande é constituído de 15 classes de problemas com $n \in \{20, 40, 60\}$ e $m \in \{5, 10, 15, 20, 25\}$. São testados 1800 problemas neste grupo.

Os tempos de processamento, *setup* e remoção são números inteiros, uniformemente distribuídos, gerados aleatoriamente. Os tempos de processamento são distribuídos no intervalo [1, 99], enquanto que os tempos de *setup* e remoção são distribuídos nos intervalos [1, 49], [1, 99], [51, 149] e [101, 199]. Isto gera 4 relações entre esses intervalos de distribuição: na primeira relação, a média do intervalo de distribuição dos tempos de *setup* e

remoção é metade da média do intervalo de distribuição dos tempos de processamento; na segunda, a média do intervalo de distribuição dos tempos de *setup* e remoção é igual à média do intervalo de distribuição dos tempos de processamento; na terceira, a média do intervalo de distribuição dos tempos de *setup* e remoção é igual a duas vezes a média do intervalo de distribuição dos tempos de processamento; e, na última, o valor da média do intervalo de distribuição dos tempos de *setup* e remoção é três vezes o valor da média do intervalo de distribuição dos tempos de processamento.

Os resultados foram por eles agrupados em relação ao número de tarefas e analisados em função do número de máquinas e das relações entre os intervalos de distribuição dos tempos de *setup* e remoção e o intervalo de distribuição dos tempos de processamento.

No caso de problemas de tamanho pequeno, Rajendran; Ziegler (1997), analisam os métodos por eles propostos em relação à porcentagem de aumento[21] absoluto (ou porcentagem de erro absoluto) no *makespan* da seqüência gerada por cada heurística em relação ao *makespan* de uma seqüência ótima. Assim, eles apresentam a porcentagem média e máxima do aumento absoluto em todos os problemas analisados.

Por outro lado, no caso de problemas de tamanho grande, Rajendran; Ziegler (1997), analisam os métodos por eles propostos em relação à porcentagem de aumento relativo no *makespan* da seqüência gerada por cada heurística em relação ao melhor *makespan* das soluções encontradas por todas as heurísticas analisadas. A porcentagem média e máxima do aumento relativo em todos os problemas analisados é por eles apresentada.

Rajendran; Ziegler (1997) também avaliam a performance das heurísticas por eles propostas em relação ao tempo computacional.

[21] Denominado neste trabalho de desvio relativo.

Eles analisam 120 problemas em cada uma das 25 classes, sendo 30 problemas considerando cada uma das 4 relações entre e os intervalo de distribuição dos tempos de *setup* e remoção e o intervalo de distribuição dos tempos de processamento.

4 PROPOSIÇÃO DE NOVOS MÉTODOS HEURÍSTICOS

4.1 DEFINIÇÃO DO PROBLEMA

4.1.1 Ambiente de produção

O ambiente de produção tratado neste trabalho é o *Flow Shop* Permutacional com *m* máquinas (PF*m*) e, apenas, a restrição adicional de tempos de *setup*, das máquinas para as tarefas, separados dos tempos de processamento, independentes da seqüência de execução das tarefas e antecipatórios.

4.1.2 Hipóteses do problema

O problema está sujeito às hipóteses do problema de *Flow Shop* tradicional, cujos tempos de *setup* são considerados como parte dos tempos de processamento, listadas por Baker (1974), Taillard (1993), Ruiz; Maroto (2005) e Gupta; Stafford Jr. (2006), adaptadas ao caso de *Flow Shop* com tempos de *setup* separados dos tempos de processamento, conforme Logendran; Sriskandarajah (1993) e Yang; Liao (1999) e ao caso de não existirem outras

restrições adicionais. Assim, são apresentadas as hipóteses já adaptadas ao problema considerado.

As hipóteses do problema podem, de acordo com Gupta; Stafford Jr. (2006), ser divididas em hipóteses sobre tarefas, sobre máquinas e sobre políticas de operações.

As hipóteses sobre tarefas (G) são as seguintes:

G1. Cada tarefa está disponível para ser processada no início do período de programação;

G2. Cada tarefa pode ter sua própria *due date* que é fixa e não está sujeita a mudanças;

G3. Cada tarefa é independente das outras tarefas;

G4. Cada tarefa consiste de específicas operações, que são realizadas cada uma em uma única máquina.

G5. Todas as tarefas têm operações em todas as máquinas;

G6. Cada tarefa possui uma ordem tecnológica pré-determinada e fixa que é a mesma para todas as outras tarefas;

G7. Cada tarefa (operação) requer um conhecido e finito tempo de processamento em cada máquina. Este tempo de processamento é independente das tarefas precedentes e das sucessoras. Não existem tempos de processamento iguais a zero (isto é resultado da hipótese G5);

G8. Todas as tarefas (operações) requerem *setup* em todas as máquinas;

G9. Cada tarefa (operação) requer um conhecido e finito tempo de *setup* em cada máquina. Este tempo é independente da seqüência de tarefas na respectiva máquina. Não existem tempos de *setup* igual a zero (isto é resultado da hipótese G8);

G10. Cada tarefa não é processada mais que uma vez em nenhuma máquina;

G11. Cada tarefa pode esperar entre máquinas e, portanto o estoque em processo é permitido.

Enquanto que as hipóteses sobre máquinas (R) são as seguintes:

R1. Cada centro de máquinas consiste de apenas uma máquina, assim existe apenas uma máquina de cada tipo;

R2. Cada máquina está inicialmente disponível no início do período de programação, ou seja, nenhuma atividade (processamento, *setup*, por exemplo), referente a uma tarefa, pode ser realizada antes do início do período de programação;

R3. Cada máquina opera independentemente das outras máquinas, sendo então capaz de operar em sua própria taxa de saída máxima;

R4. Cada máquina pode operar no máximo uma tarefa por vez. Isto elimina máquinas que são designadas para processar várias tarefas simultaneamente;

R5. Cada máquina está continuamente disponível para processar tarefas através do período de programação, não sendo previstas interrupções devido a *breakdowns*, manutenção ou outras causas.

Por sua vez, lista-se como hipóteses sobre políticas de operação (H):

H1. Cada tarefa deve ser tratada como se tivesse exatamente *m* operações. Quando algumas operações não existirem, o correspondente tempo de processamento pode ser considerado igual a zero, assim como o tempo de *setup*, no entanto, esse não é o caso aqui (isto é resultado da hipótese G5);

H2. Quando alguma tarefa (operação) não requerer *setup* em alguma máquina o tempo de *setup* pode ser considerado igual a zero, mas este não é o caso neste trabalho (isto é resultado da hipótese G8);

H3. Cada tarefa é processada tão cedo quanto possível. Assim, não existem tarefas esperando intencionalmente ou intenção de permitir que as máquinas tenham tempo ocioso;

H4. Cada tarefa é considerada uma entidade indivisível até mesmo se ela puder ser composta de um número de unidades individuais;

H5. Cada tarefa, uma vez aceita, é processada por completo, ou seja, cancelamentos de tarefas não são previstos;

H6. Cada tarefa (operação), uma vez iniciada em uma máquina, é completada antes de uma outra tarefa poder iniciar seu processamento nesta máquina, isto é, *preemptions*[22], não são permitidas;

H7. Cada tarefa não é processada em mais de uma máquina ao mesmo tempo (isto é resultado das hipóteses G7 e H4);

H8. Cada máquina possui um adequado espaço de espera, permitindo que as tarefas possam esperar antes de seu processamento ser iniciado;

H9. Cada máquina é totalmente alocada para as tarefas durante todo o período de programação, ou seja, as máquinas não são usadas para nenhum outro propósito através do período de programação;

H10. O *setup* pode ser realizado antes de a tarefa estar disponível para processamento, desde que a máquina esteja ociosa (*setup* antecipatório);

H11. O tempo de transferência entre máquinas e o tempo de remoção na máquina são considerados iguais a zero.

[22] Execução parcial de uma operação, com possibilidade de ser completada posteriormente.

4.1.3 Notação

A notação utilizada é assim definida:

n = número de tarefas a serem programadas;

m = número de máquinas do *Flow Shop*;

$\sigma = J_{[1]}, J_{[2]},..., J_{[j]},..., J_{[n]}$ = uma seqüência qualquer, dentre as ($n!$[23]) possíveis seqüências, onde $J_{[j]}$ representa a tarefa que ocupa a j-ésima posição;

$p_{k[j]}$ = tempo de processamento na máquina M_k da tarefa $J_{[j]}$;

$s_{k[j]}$ = tempo de *setup* da máquina M_k para a execução da tarefa $J_{[j]}$;

$X_{[j]k}$ = intervalo de tempo entre o término da preparação da máquina M_k para a execução da $J_{[j]}$ e o inicio do processamento de $J_{[j]}$ nesta máquina, ou seja, é o tempo ocioso na máquina M_k entre o fim do *setup* para a tarefa $J_{[j]}$ e o início do processamento desta tarefa;

$Y_{[j]k}$ = intervalo de tempo entre o término da operação da tarefa $J_{[j]}$ na máquina M_k e o início da operação da mesma tarefa na máquina $M_{(k+1)}$, ou seja, o tempo de espera da tarefa $J_{[j]}$ após o fim de seu processamento na máquina M_k e o início de seu processamento na máquina seguinte;

LB = *Lower Bound* = Limitante Inferior;

UB = *Upper Bound* = Limitante Superior;

D = Duração total da programação = *Makespan*.

Desta maneira, o problema de programação, neste trabalho, consiste em programar um conjunto de n tarefas, definido como $\{J_1, J_2, ..., J_j, ..., J_n\}$, num *Flow Shop* Permutacional com m máquinas $\{M_1, M_2, ..., M_k, ..., M_m\}$, cujos tempos de *setup* são separados dos tempos de processamento, independentes da seqüência de execução das tarefas e antecipatórios com o objetivo de minimização do *Makespan*, como medida de desempenho.

[23] No *Flow Shop* Permutacional, conforme Moccellin (1995), o número de programações possíveis para n tarefas é igual a ($n!$).

4.2 UMA PROPRIEDADE DO PROBLEMA DE PROGRAMAÇÃO DE OPERAÇÕES EM *FLOW SHOP* PERMUTACIONAL COM TEMPOS DE *SETUP* SEPARADOS DOS TEMPOS DE PROCESSAMENTO E ANTECIPATÓRIOS

Aqui apresenta-se uma propriedade do problema de programação de *n* tarefas $\{J_1, J_2, ..., J_j, ..., J_n\}$ em um *Flow Shop* Permutacional com *m* máquinas $\{M_1, M_2, ..., M_k, ..., M_m\}$, com tempos de *setup* das máquinas para as tarefas separados dos tempos de processamento e antecipatórios, podendo ser dependentes ou não da seqüência de execução das tarefas, denominada "Propriedade LBY", que corresponde ao Limitante Inferior da variável $Y_{[j]k}$ (Tempo de Espera). Tal propriedade tem sua origem em investigações realizadas sobre as características estruturais do problema e sua solução.

A solução do problema de programação em *Flow Shop* Permutacional, consiste em estabelecer dentre as (*n*!) seqüências possíveis das tarefas, aquela que otimiza alguma medida de desempenho adotado, como o *Makespan* neste trabalho.

A partir das notações pode-se escrever:

$$D = \sum_{j=1}^{n} \left(p_{1j} + s_{1j} \right) + Y_{[n]1} + p_{2n} + Y_{[n]2} + p_{3n} + ... + Y_{[n]m-1} + p_{mn} \qquad (4.1)$$

A expressão (4.1) pode ser visualizada na Figura 4.1, adaptada de Nagano (1999, p. 21).

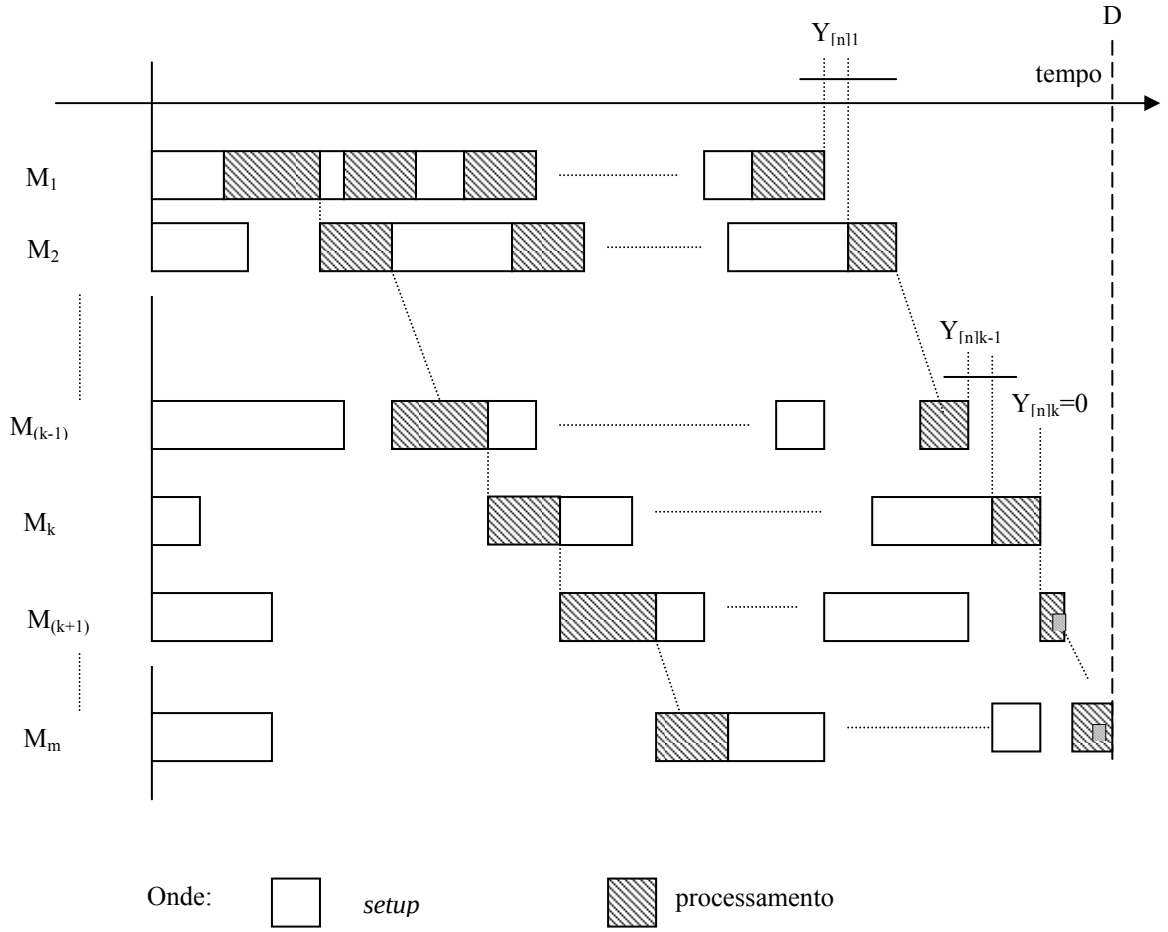

Figura 4.1. Ilustração do *Makespan*

A expressão (4.1) pode assim ser agregada:

$$D = \sum_{j=1}^{n} \left(p_{1j} + s_{1j} \right) + \sum_{k=2}^{m} p_{k[n]} + \sum_{K=1}^{m-1} Y_{[n]k} \qquad (4.2)$$

A expressão (4.2) mostra que a duração total da programação é dada pela soma de três parcelas: a primeira é a soma dos tempos de processamento das *n* tarefas e dos tempos de *setup* para as *n* tarefas na primeira máquina, com um valor constante e, portanto, independente da seqüência σ; a segunda parcela corresponde à soma dos tempos de processamento da tarefa que ocupa a última posição na seqüência σ, a partir da máquina 2 até a máquina *m*, sendo

então dependente da seqüência σ que for considerada; e, a terceira parcela é a soma dos intervalos de tempo entre as operações sucessivas da tarefa $J_{[n]}$, da máquina M_1 até a máquina M_m, também dependente da seqüência σ que for considerada.

Fundamentando-se no estudo efetuado por Moccellin; Nagano (2006), que pode ser considerado uma extensão do estudo anteriormente realizado por Moccellin (1992), duas tarefas adjacentes quaisquer $J_{[j]}$ e $J_{[j+1]}$, aqui denominadas J_u e J_v, podem ser consideradas. Pode-se, também, considerar a atividade de *setup* nessa condição de tarefas sucessivas (independente ou não da seqüência), designada de S_{uv}, ou seja, *setup* para a tarefa J_v após o processamento da tarefa J_u ter sido feito antes. Assim, S_{uv} pode ser considerada como se fosse mais uma tarefa a ser executada entre J_u e J_v. Além disso, após a conclusão, em uma máquina qualquer, da tarefa J_u, essa máquina já pode ser preparada para processar a tarefa J_v, não necessitando aguardar o término da operação da tarefa J_v na máquina anterior. Isto é possível uma vez que está se tratando do *setup* antecipatório. Logo não existe tempo ocioso nas máquinas entre o término de J_u e o início do *setup* S_{uv}.

Desta maneira, considera-se que J_u, S_{uv} e J_v sejam três "tarefas" sucessivas em uma dada seqüência de tarefas. Assim, expressões de viabilidade entre tarefas sucessivas são aplicadas aos pares (J_u, S_{uv}) e (S_{uv}, J_v), levando, segundo Moccellin; Nagano (2006), a:

$$X_{[u\,Suv]k} + s_{k\,uv} + Y_{[u\,Suv]k} = Y_{[Stu\,u]k} + p_{k+1,u} + X_{[u\,Suv]k+1} \qquad \text{(Entre } J_u \text{ e } S_{uv}\text{)} \qquad (4.3)$$

$$X_{[Suv\,v]k} + p_{k\,v} + Y_{[Suv\,v]k} = Y_{[u\,Suv]k} + s_{k+1,uv} + X_{[Suv\,v]k+1} \qquad \text{(Entre } S_{uv} \text{ e } J_v\text{)} \qquad (4.4)$$

Onde, genericamente:

$X_{[xy]k}$ = intervalo de tempo na máquina M_k entre o término de uma atividade (*setup*/processamento) da tarefa J_x e o início de uma outra atividade (processamento da tarefa J_x /*setup* da tarefa J_y), quando a tarefa J_x é precedente direta da tarefa J_y;

$s_{k\,xy}$ = tempo de *setup* da máquina M_k para a execução da tarefa J_y, quando a tarefa J_x é sua precedente direta;

$Y_{[xy]k}$ = intervalo de tempo entre o término da atividade (*setup*/processamento) da tarefa J_y na máquina M_k e o início da atividade (*setup*/processamento) da mesma tarefa na máquina $M_{(k+1)}$, com a tarefa J_x precedendo diretamente a tarefa J_y.

Na expressão (4.3) tem-se que $X_{[u\,Suv]k} = X_{[u\,Suv]k+1} = 0$, o que leva:

$$s_{k\,uv} + Y_{[u\,Suv]k} = Y_{[Stu\,u]k} + p_{(k+1)\,u}$$

Portanto,

$$Y_{[u\,Suv]k} = Y_{[Stu\,u]k} + p_{(k+1)\,u} - s_{k\,uv} \qquad (4.5)$$

Na expressão (4.5) é importante observar que $Y_{[u\,Suv]k}$ não necessariamente mantém a condição de ser positivo ou nulo. Neste caso, a data de término do *setup* S_{uv} na máquina M_k é maior que a data de término desse *setup* na máquina seguinte $M_{(k+1)}$, como é possível visualizar na Figura 4.2. Isto ocorre, pois não existem restrições de precedência entre atividades de *setup* de máquinas sucessivas para o processamento de uma tarefa qualquer, o que não elimina a validade das expressões (4.3) a (4.5), conforme ilustra a Figura 4.2, adaptada de Moccellin; Nagano (2006).

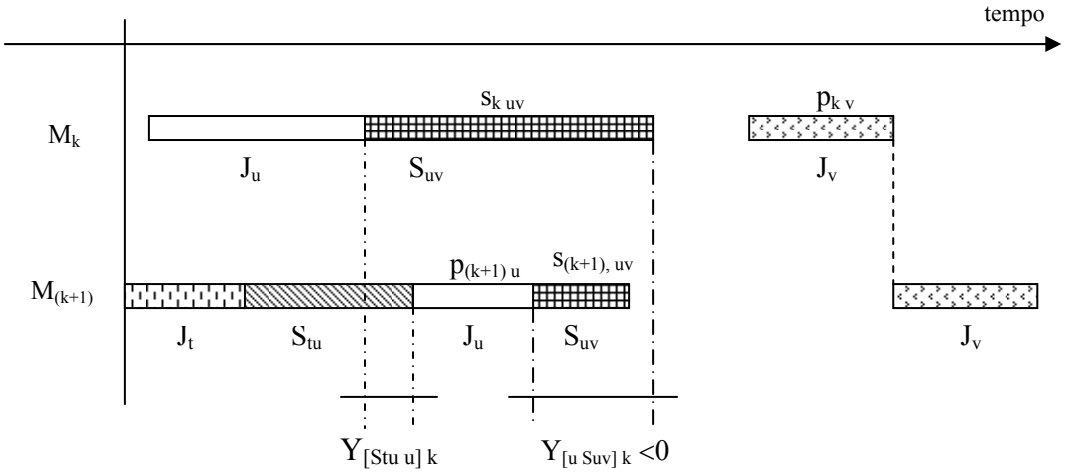

Figura 4.2. Ilustração de uma situação em que $Y_{[u\,Suv]k} < 0$

Substituindo-se $Y_{[u\,Suv]k}$ da expressão (4.5) na expressão (4.4):

$$X_{[Suv\,v]k} + p_{k\,v} + Y_{[Suv\,v]k} = Y_{[Stu\,u]k} + p_{(k+1)u} - s_{k,uv} + s_{k+1,uv} + X_{[Suv\,v]k+1}$$

que leva a:

$$X_{[Suv\,v]k} + (p_{k\,v} + s_{k\,uv}) + Y_{[Suv\,v]k} = Y_{[Stu\,u]k} + (p_{(k+1)u} + s_{(k+1)\,uv}) + X_{[Suv\,v]k+1} \quad (4.6)$$

A expressão (4.6) viabiliza, para duas tarefas sucessivas J_u e J_v e o seu *setup* S_{uv}, as relações entre tempos de máquina parada e tempos de espera de operações sucessivas de uma mesma tarefa, para o problema em que os tempos de *setup* não são incluídos nos tempos de processamento.

Como $X_{[x\,y]k}$ e $Y_{[x\,y]k}$ da expressão (4.6) mantêm a condição de serem sempre positivos ou nulos, é possível completar a expressão (4.6), com as seguintes expressões de viabilidade:

Se $X_{[Suv\ v]k+1} > 0$ então $Y_{[Suv\ v]k} = 0$, e (4.7)

Se $Y_{[Suv\ v]k} > 0$ então $X_{[Suv\ v]k+1} = 0$. (4.8)

As expressões (4.7) e (4.8), ilustradas na Figura 4.3, adaptada de Moccellin; Nagano (2006), demonstram que os termos $X_{[Suv\ v]k+1}$ e $Y_{[Suv\ v]k}$ não podem ser simultaneamente positivos.

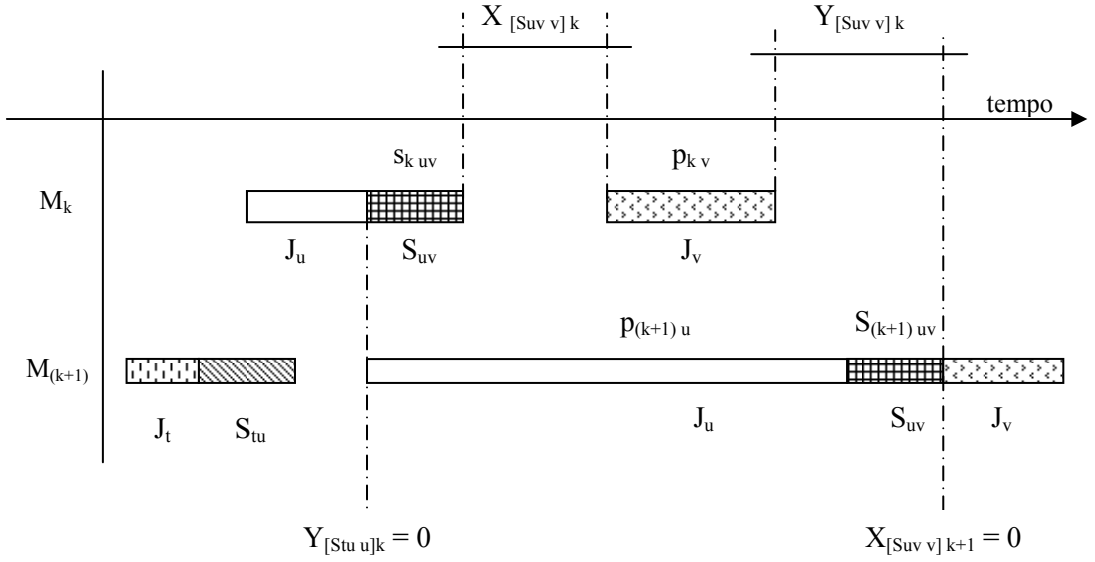

Figura 4.3. Relações de viabilidade entre S_{uv} e J_v

Da expressão (4.6), pode-se, de maneira semelhante ao desenvolvimento efetuado por Nagano (1999), desenvolver uma propriedade LBY.

Uma vez que, por definição $Y_{[stu\ u]k} \geq 0$, pode-se escrever, a partir da expressão (4.6):

$$X_{[Suv\ v]k+1} - Y_{[Suv\ v]k} \leq X_{[Suv\ v]k} + (p_{k\ v} + s_{k\ uv}) - (p_{(k+1)\ u} + s_{(k+1)\ uv}) \quad (4.9)$$

Considerando-se $UBX_{[Suv\ v]k}$ um limitante superior para $X_{[Suv\ v]k}$ (ou seja, $X_{[Suv\ v]k} \leq UBX_{[Suv\ v]k}$), pode-se reescrever a expressão (4.9), da seguinte maneira:

$$X_{[Suv\ v]k+1} - Y_{[Suv\ v]k} \leq UBX_{[Suv\ v]k} + (p_{k\ v} + s_{k\ uv}) - (p_{(k+1)\ u} + s_{(k+1)\ uv}) \quad (4.10)$$

Ou seja:

$$Y_{[Suv\ v]k} \geq (p_{(k+1)\ u} + s_{(k+1)\ uv}) - (p_{k\ v} + s_{k\ uv}) - UBX_{[Suv\ v]k} + X_{[Suv\ v]k+1} \quad (4.11)$$

Uma vez que, por definição, $X_{[Suv\ v]k+1} \geq 0$, tem-se que:

$$Y_{[Suv\ v]k} \geq (p_{(k+1)\ u} + s_{(k+1)\ uv}) - (p_{k\ v} + s_{kuv}) - UBX_{[Suv\ v]k} \quad (4.12)$$

A partir da expressão (4.12) e das expressões de viabilidade (4.7) e (4.8), pode-se estabelecer a Propriedade LBY, que fornece um limitante inferior para $Y_{[Suv\ v]k}$ (ou seja, $Y_{[Suv\ v]k} \geq LBY_{[Suv\ v]k}$), dado por:

$$LBY_{[Suv\ v]k} = Max\ \{0, (p_{(k+1)\ u} + s_{(k+1)\ uv}) - (p_{k\ v} + s_{k\ uv}) - UBX_{[Suv\ v]k}\} \quad (4.13)$$

Sendo k = 1,2, ..., (m-1)

Onde, $UBX_{[Suv\ v]k}$ é um limitante superior do intervalo de tempo entre o término do *setup* e o início do processamento de J_v, na máquina M_k, quando a tarefa J_u é sua precedente direta na seqüência σ, dado, segundo Moccellin; Nagano (2006), por:

$$UBX_{[Suv\ v]k} = Max\ \{0, UBX_{[Suv\ v]k-1} + (p_{(k-1)\ v} + s_{(k-1)\ uv}) - (p_{k\ u} + s_{k\ uv})\} \quad (4.14)$$

Sendo k = 1, 2, ..., *m*, com $UBX_{[Suv\ v]1} = 0$.

Assim, $LBY_{[Suv\ v]k}$ é um limitante inferior do tempo de espera da operação da tarefa J_v entre as máquinas M_k e $M_{(k+1),}$ quando a tarefa J_u é sua precedente direta na seqüência σ e S_{uv} o seu *setup*, como é possível visualizar na Figura 4.4.

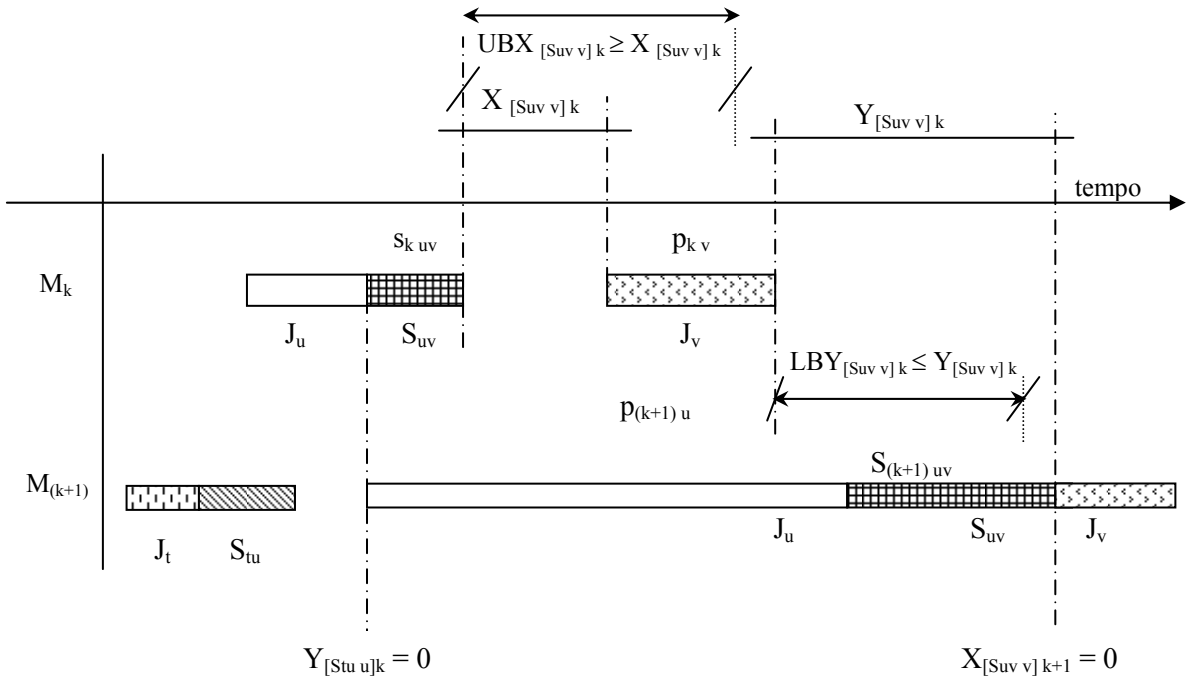

Figura 4.4. Ilustração da Propriedade $LBY_{[Suv\ v]k}$

O valor de $LBY_{[Suv\ v]k}$ pode, então, ser calculado para todos os *n* (*n*-1) arranjos de pares ordenados de tarefas do conjunto de *n* tarefas $\{J_1, J_2, ..., J_j, ..., J_n\}$, tomadas duas a duas.

A somatória dos $LBY_{[Suv\ v]k}$, para k = 1, 2, ..., (*m*-1), leva a:

$$\sum_{k=1}^{m-1} LBY_{[Suv\ v]k} = LBY_{[Suv\ v]1} + LBY_{[Suv\ v]2} + ... + LBY_{[Suv\ v]m-1} \quad (4.15)$$

A expressão (4.15) fornece um limitante inferior da soma dos tempos de espera da tarefa J_v entre as *m* sucessivas máquinas do *Flow Shop*, quando a tarefa J_u for sua precedente direta e S_{uv} o seu *setup*.

4.2.1 Ilustração dos cálculos de $UBX_{[Suv\ v]k}$ e de $LBY_{[Suv\ v]k}$

Os cálculos de $UBX_{[Suv\ v]k}$ e $LBY_{[Suv\ v]k}$ podem ser exemplificados através de um exemplo simples de 2 tarefas $\{J_1,J_2\}$ que devem ser processadas em um Flow Shop com 3 máquinas, cujos tempos de processamento e *setup*, gerados aleatoriamente, são fornecidos na Figura 4.5.

Tarefas	**Máquina 1**		**Máquina 2**		**Máquina 3**	
	p_{kj}	s_{kj}	p_{kj}	s_{kj}	p_{kj}	s_{kj}
1	3	5	4	3	5	2
2	2	4	4	1	1	2

Onde:

p_{kj} = tempo de processamento na máquina M_k da tarefa J_j;
s_{kj} = tempo de *setup* da máquina M_k para a execução da tarefa J_j.

Figura 4.5. Dados do exemplo 1

Sendo:

$UBX_{[Suv\ v]1} = 0$

$UBX_{[Suv\ v]k} = \text{Max}\ \{0, UBX_{[Suv\ v]k-1} + (p_{(k-1)\ v} + s_{(k-1)\ uv}) - (p_{k\ u} + s_{k\ uv})\}$ para k = 1, 2, ..., *m*

$LBY[_{Suv\ v}]_k = \text{Max}\ \{0, (p_{(k+1)\ u} + s_{(k+1)\ uv}) - (p_{k\ v} + s_{k\ uv}) - UBX_{[Suv\ v]k}\}$ para k = 1, 2, ..., (*m* –1)

k = 1

Par J_1,J_2

$UBX_{[S12\ 2]1} = 0$

$LBY[_{S12\ 2}]_1 = \text{Max}\ \{0, (p_{2\ 1} + s_{2\ 12}) - (p_{1\ 2} + s_{1\ 12}) - UBX_{[S12\ 2]1}\}$

$= \text{Max}\ \{0, (4+1) – (2+4) – 0 = \text{Max}\ \{0,-1\} = 0$

Par J_2,J_1

$UBX_{[S21\ 1]1} = 0$

$LBY[_{S21\ 1}]_1 = \text{Max}\ \{0, (p_{2\ 2} + s_{2\ 21}) - (p_{1\ 1} + s_{1\ 21}) - UBX_{[S21\ 1]1}\}$

$= \text{Max}\ \{0, (4+4) – (3+5) – 0 = \text{Max}\ \{0, 0\} = 0$

k = 2

Par J_1,J_2

$UBX_{[S12\ 2]2} = \text{Max}\ \{0, UBX_{[S12\ 2]1} + (p_{1\ 2} + s_{1\ 12}) - (p_{2\ 1} + s_{2\ 12})\}$

$= \text{Max}\ \{0, 0 + (2+4) – (4+1)\} = \text{Max}\ \{0, 1\} = 1$

$LBY[_{S12\ 2}]_2 = \text{Max}\ \{0, (p_{3\ 1} + s_{3\ 12}) - (p_{2\ 2} + s_{2\ 12}) - UBX_{[S12\ 2]2}\}$

$= \text{Max}\ \{0, (5+2) – (4+1) –1 = \text{Max}\ \{0, 1\} = 1$

Par J_2,J_1

$UBX_{[S21\ 1]2} = \text{Max}\ \{0, UBX_{[S21\ 1]1} + (p_{1\ 1} + s_{1\ 21}) - (p_{2\ 2} + s_{2\ 21})\}$

$$= \text{Max } \{0,\ 0 + (3+5) - (4+3\} = \text{Max } \{0,\ 1\} = 1$$

$$LBY[_{S21\ 1}]_2 = \text{Max } \{0,\ (p_{3\ 2} + s_{3\ 21}) - (p_{2\ \ 1} + s_{2\ \ 21}) - UBX_{[S21\ 1]2}\}$$

$$= \text{Max } \{0,\ (1+2) - (4+3) - 1 = \text{Max } \{0,\ -5\} = 0$$

As matrizes $LBY[_{Suv\ v}]_k$, para k = 1, 2, ..., (m-1) e $\sum_{k=1}^{m-1} LBY_{[Suv\ v]k}$ podem ser vistas na Figura 4.6.

$$LBY[_{Suv\ v}]_1 = \begin{bmatrix} - & 0 \\ 0 & - \end{bmatrix} \qquad LBY[_{Suv\ v}]_2 = \begin{bmatrix} - & 1 \\ 0 & - \end{bmatrix} \qquad \sum_{k=1}^{m-1} LBY_{[Suv\ v]k} = \begin{bmatrix} - & 1 \\ 0 & - \end{bmatrix}$$

Figura 4.6. Matrizes da ilustração dos cálculos de $UBX_{[Suv\ v]k}$ e $LBY_{[Suv\ v]k}$

A "Propriedade LBY" pode ser utilizada na concepção de métodos heurísticos e de métodos *Branch-and-Bound*. Assim, neste trabalho, apresenta-se uma primeira aplicação da Propriedade.

4.3 CONCEPÇÃO DOS NOVOS MÉTODOS HEURÍSTICOS

Como uma primeira aplicação da Propriedade LBY, se propõem dois novos métodos heurísticos, um construtivo e um melhorativo, para a programação de operações em *Flow Shop* Permutacional com tempos de *setup* das máquinas, para as tarefas, separados dos tempos de processamento, antecipatórios e independentes da seqüência de execução das tarefas.

Utilizando uma outra Propriedade LBY, semelhante à apresentada neste trabalho, Nagano; Moccellin (2002) propõem um método heurístico construtivo para a programação de

operações em *Flow Shop* Permutacional, com os tempos de *setup* incluídos nos tempos de processamento das tarefas. O método proposto por Nagano; Moccellin (2002), denominado de N&M, difere do método NEH, de Nawaz; Enscore Jr; Ham (1983), apenas nos passos do procedimento de ordenação inicial das tarefas, uma vez que eles percebem ser este o forte do método NEH. O procedimento de inserção de tarefas do método N&M é o mesmo do método NEH.

Assim, salienta-se que os esforços deste trabalho foram direcionados para a determinação de procedimentos de ordenação inicial das tarefas, utilizando a Propriedade LBY, de forma semelhante à realizada por Nagano (1999).

4.3.1 Determinação dos procedimentos de ordenação inicial das tarefas

4.3.1.1 Procedimento de ordenação de tarefas

São apresentados e ilustrados, através de exemplos, por Nagano (1999), oito procedimentos de ordenação de tarefas.

Embora todos os procedimentos apresentados por Nagano (1999) possam ser utilizados em procedimentos de ordenação inicial das tarefas de métodos heurísticos, neste trabalho optou-se por testar, inicialmente, apenas a ordenação *forward* das tarefas, uma vez que, essa ordenação mostra-se, em experimentação computacional realizada por Nagano (1999), mais eficiente do que os outros sete procedimentos.

Na **ordenação *forward* das tarefas** utilizando-se uma dada matriz, busca-se, inicialmente, o maior elemento da matriz, identificando-se, em seguida, as tarefas adjacentes J_u e J_v, que correspondem, respectivamente, aos elementos linha-coluna da matriz. Assim, J_u ocupa a primeira posição na ordenação, enquanto que, J_v a segunda posição. O passo seguinte consiste em encontrar a nova tarefa J'_v, ainda não inserida na ordenação, verificando-se o maior elemento da linha da matriz correspondente a última tarefa já ordenada J_v. Desta maneira, a ordenação parcial seria até o momento J_u, J_v, J'_v. O processo é repetido até que todas as tarefas sejam ordenadas.

É possível perceber, após a descrição da ordenação *forward* das tarefas, que faz-se necessário determinar a matriz a ser utilizada no procedimento de ordenação inicial das tarefas.

4.3.1.2 Matrizes

Se para cada par de tarefas sucessivas J_u e J_v, sendo S_{uv} o *setup* de de J_v, é possível calcular um limitante inferior da somatória dos tempos de espera, então, $[\sum LBY_{Suv\ v}]_{nxn}$ é a matriz correspondente aos n (n-1) arranjos do conjunto de n tarefas tomadas duas a duas, em que a posição de cada elemento linha-coluna representa respectivamente as tarefas J_u e J_v, conforme a Figura 4.7.

$$[\sum LBY_{Suv\,v}]_{nxn} = \begin{bmatrix} - & \sum_{k=1}^{m-1} LBY_{[s12,2]k} & ..... & ..... \sum_{k=1}^{m-1} LBY_{[s1n,n]k} \\ \sum_{k=1}^{m-1} LBY_{[s21,1]k} & - & ..... & ..... \sum_{k=1}^{m-1} LBY_{[s2n,n]k} \\ . & . & & . \\ . & . & & . \\ . & . & & . \\ . & . & & . \\ . & . & & . \\ \sum_{k=1}^{m-1} LBY_{[sn1,1]k} & \sum_{k=1}^{m-1} LBY_{[sn2,2]k} & ..... & .....- \end{bmatrix}$$

Figura 4.7. Matriz $[\sum LBY_{Suv\,v}]_{nxn}$

Pode-se considerar $[\sum P_v]_{nxn}$ como a matriz formada pela soma dos tempos de processamento e $[\sum S_v]_{nxn}$ como a matriz formada pela soma dos tempos de *setup* das tarefas J_v, para v = 1, 2, ..., *n*, em que os (*n*-1) elementos de uma coluna correspondem, respectivamente, à somatória dos tempos de processamento e de *setup* de uma determinada tarefa para as *m* máquinas, como é possível visualizar nas Figuras 4.8 e 4.9.

$$[\sum P_v]_{nxn} = \begin{bmatrix} - & \sum_{k=1}^{m} p_{k2} & ..... & .....\sum_{k=1}^{m} p_{kn} \\ \sum_{k=1}^{m} p_{k1} & - & ..... & .....\sum_{k=1}^{m} p_{kn} \\ . & . & & . \\ . & . & & . \\ . & . & & . \\ . & . & & . \\ . & . & & . \\ \sum_{k=1}^{m} p_{k1} & \sum_{k=1}^{m} p_{k2} & ..... & .....- \end{bmatrix}$$

Figura 4.8. Matriz $[\sum P_v]_{nxn}$

$$[\sum S_v]_{nxn} = \begin{bmatrix} - & \sum_{k=1}^{m} s_{k2} & ..... & .....\sum_{k=1}^{m} s_{kn} \\ \sum_{k=1}^{m} s_{k1} & - & ..... & .....\sum_{k=1}^{m} s_{kn} \\ . & . & & . \\ . & . & & . \\ . & . & & . \\ . & . & & . \\ . & . & & . \\ \sum_{k=1}^{m} s_{k1} & \sum_{k=1}^{m} s_{k2} & ..... & .....- \end{bmatrix}$$

Figura 4.9. Matriz $[\sum S_v]_{nxn}$

Somando-se as matrizes $[\sum P_v]_{nxn}$ e $[\sum S_v]_{nxn}$ pode-se obter uma matriz da somatória dos tempos de processamento e de *setup* das tarefas, denominada matriz $[\sum J_v]_{nxn}$, como ilustra a Figura 4.10.

$$[\sum J_v]_{nxn} = \begin{bmatrix} - & \sum_{k=1}^{m}(p_{k2}+S_{k2}) & ..... & .....\sum_{k=1}^{m}(p_{kn}+S_{kn}) \\ \sum_{k=1}^{m}(p_{k1}+S_{k1}) & - & ..... & .....\sum_{k=1}^{m}(p_{kn}+S_{kn}) \\ . & . & & . \\ . & . & & . \\ . & . & & . \\ . & . & & . \\ . & . & & . \\ \sum_{k=1}^{m}(p_{k1}+S_{k1}) & \sum_{k=1}^{m}(p_{k2}+S_{k2}) & ..... & .....- \end{bmatrix}$$

Figura 4.10. Matriz $[\sum J_v]_{nxn} = [\sum P_v]_{nxn} + [\sum S_v]_{nxn}$

Nas diagonais das matrizes é adotado o valor zero para simplificar as operações computacionais.

A partir das matrizes $[\sum LBY_{Suv\ v}]_{nxn}$ e $[\sum J_v]_{nxn}$, é possível obter as matrizes Ω e ϕ. A matriz Ω é obtida subtraindo-se as matrizes $[\sum J_v]_{nxn}$ e $[\sum LBY_{Suv\ v}]_{nxn}$, logo, $[\Omega]_{nxn} = [\sum J_v]_{nxn} - [\sum LBY_{Suv\ v}]_{nxn}$, como é possível visualizar na Figura 4.11. A matriz ϕ é obtida somando-se as matrizes $[\sum LBY_{Suv\ v}]_{nxn}$ e $[\sum J_v]_{nxn}$, ou seja, $[\phi]_{nxn} = [\sum LBY_{Suv\ v}]_{nxn} + [\sum J_v]_{nxn}$, conforme Figura 4.12.

$$[\Omega]_{nxn} = \begin{bmatrix} - & \sum_{k=1}^{m} J_2 - \sum_{k=1}^{m-1} LBY_{[s12,2]k} & \cdots & \cdots \sum_{k=1}^{m} J_n - \sum_{k=1}^{m-1} LBY_{[s1n,n]k} \\ \sum_{k=1}^{m} J_1 - \sum_{k=1}^{m-1} LBY_{[s21,1]k} & - & \cdots & \cdots \sum_{k=1}^{m} J_n - \sum_{k=1}^{m-1} LBY_{[s2n,n]k} \\ . & . & & . \\ . & . & & . \\ . & . & & . \\ . & . & & . \\ . & . & & . \\ \sum_{k=1}^{m} J_1 - \sum_{k=1}^{m-1} LBY_{[sn1,1]k} & \sum_{k=1}^{m} J_2 - \sum_{k=1}^{m-1} LBY_{[sn2,2]k} & \cdots & \cdots - \end{bmatrix}$$

Figura 4.11. Matriz $[\Omega]_{nxn}$

$$[\phi]_{nxn} = \begin{bmatrix} - & \sum_{k=1}^{m-1} LBY_{[s12,2]k} + \sum_{k=1}^{m} J_2 & \cdots & \cdots \sum_{k=1}^{m-1} LBY_{[s1n,n]k} + \sum_{k=1}^{m} J_n \\ \sum_{k=1}^{m-1} LBY_{[s21,1]k} + \sum_{k=1}^{m} J_1 & - & \cdots & \cdots \sum_{k=1}^{m-1} LBY_{[s2n,n]k} + \sum_{k=1}^{m} J_n \\ . & . & & . \\ . & . & & . \\ . & . & & . \\ . & . & & . \\ . & . & & . \\ \sum_{k=1}^{m-1} LBY_{[sn1,1]k} + \sum_{k=1}^{m} J_1 & \sum_{k=1}^{m-1} LBY_{[sn2,2]k} + \sum_{k=1}^{m} J_2 & \cdots & \cdots - \end{bmatrix}$$

Figura 4.12. Matriz $[\phi]_{nxn}$

Assim, utilizando-se a ordenação *forward* das tarefas e as matrizes $[\sum LBY_{Suv\,v}]_{nxn}$, Ω e ϕ, desenvolveu-se 3 procedimento de ordenação inicial das tarefas e, conseqüentemente, 3

métodos heurísticos construtivos, semelhantes aos métodos NEH e N&M, que diferem destes apenas nos passos de ordenação inicial das tarefas.

4.3.2 Métodos heurísticos construtivos desenvolvidos

Dos 3 métodos heurísticos construtivos desenvolvidos, o primeiro leva em consideração, na hora de aplicar a ordenação *forward* das tarefas, a matriz $[\sum LBY_{Suv\ v}]_{nxn}$, o segundo a matriz Ω e o terceiro ϕ. Aqui eles são designados, respectivamente, de método BM_1, BM_2 e BM_3.

Com o intuito de determinar o melhor procedimento de ordenação inicial das tarefas e de se verificar se a ordenação *forward* das tarefas, também é eficiente para o problema de programação tratado neste trabalho, assim como é para o problema considerado por Nagano (1999), foram testados os métodos BM_1, BM_2 e BM_3, juntamente com os métodos CB e RZ_3.

Assim, analisou-se um total de 540 problemas, que foram divididos em 54 classes de problemas de acordo com o número de tarefas $n \in \{4, 6, 7, 10, 20, 30, 40, 50, 60\}$ e o número de máquinas $m \in \{5, 10, 15, 20, 25, 30\}$. Estes problemas forma divididos em dois grupos. No primeiro grupo, com $n \in \{4, 6, 7\}$ e $m \in \{5, 10, 15, 20, 25, 30\}$ foram analisados 180 problemas, considerados de tamanho pequeno. No segundo grupo, com $n \in \{10, 20, 30, 40, 50, 60\}$ e $m \in \{5, 10, 15, 20, 25, 30\}$, analisou-se 360 problemas, considerados problemas de tamanho grande. Foram analisados 10 problemas para cada classe n x m.

Todos os problemas foram gerados aleatoriamente, sendo ambos os tempos de processamento e *setup* números inteiros distribuídos no intervalo [1, 99].

Os resultados obtidos[24], através da experimentação computacional preliminar realizada[25], mostraram que, entre os métodos BM_1, BM_2 e BM_3, o método BM_2, que aplica a ordenação *forward* na matriz Ω, é mais eficiente em minimizar o *Makespan*. É importante destacar que, ao se analisar os dois grupos de problemas isoladamente, o método BM_2 não foi o que apresentou melhor eficiência em minimizar o *Makespan*, no caso de problemas de tamanho pequeno, ficando um pouco atrás dos BM_1 e BM_3.

Comparando-se o método BM_2 com o método CB, percebeu-se que o método desenvolvido apresentou melhor performance do que esse último.

A comparação entre o método desenvolvido BM_2, que é um método heurístico construtivo, e o método heurístico melhorativo RZ_3, mostrou um forte indicativo de que este último pode levar a melhores resultados, embora o método BM_2 apresente melhor performance em algumas classes de problemas e, algumas vezes, a mesma performance do método RZ_3. Foi possível perceber, ainda, que nos casos em que o método RZ_3 apresentou melhor desempenho do que o método proposto BM_2, o valor do *Makespan* da solução encontrada por este não ficou muito distante do valor encontrado por RZ_3.

Após estas análises preliminares, é possível constatar que a ordenação *forward* das tarefas, também é eficiente para o problema de programação tratado neste trabalho, assim como é para o problema considerado por Nagano (1999).

Assim, propõem-se, neste trabalho, dois novos métodos heurísticos para a solução do problema, denominados métodos BM.

[24] Para a tabulação dos dados e análise dos resultados utilizou-se o Microsoft Excell 5.0/95.
[25] Foi utilizada a linguagem de programação Borland Delphi 6 e o sistema operacional Windowns XP (Versão 2002 SP2). As configurações da máquina, de propriedade da autora, são as seguintes: processador Pentium 4 da Intel com 3.2 GHz de freqüência, 1 GB de memória RAM e disco rígido de 60 GB.

4.3.3 Métodos BM

Um método heurístico construtivo e um melhorativo são propostos neste trabalho. O primeiro, construtivo, é exatamente o método desenvolvido BM_2, designado a a partir de agora BM_c. O segundo, designado BM_m, é um método melhorativo composto de duas fases, a primeira fase corresponde exatamente ao método BM_c, enquanto que, a segunda fase corresponde à fase de melhoramento dos métodos RZ.

Os métodos BM são detalhados conforme algoritmos a seguir.

ALGORITMO DO MÉTODO BM_c

(Procedimento de ordenação inicial das tarefas)

Passo 1:

Passo 1.1: Seja $[\sum J_v]_{nxn}$ a matriz formada pela soma dos tempos de processamento e pela soma dos tempos de *setup* das tarefas J_v, para v = 1, 2, ..., *n*, em que os (*n*-1) elementos de uma coluna correspondem, respectivamente, à somatória dos tempos de processamento e de *setup* de uma determinada tarefa para as *m* máquinas.

Passo 1.2: Para todas as tarefas J_v, sendo v = 1, 2, ..., *n*, calcule $\sum_{k=1}^{m-1} LBY_{[Suv\ v]k}$,

Sendo: $UBX_{[Suv\ v]1} = 0$;

$UBX_{[Suv\ v]k} = Max\ \{0, UBX_{[Suv\ v]k-1} + (p_{(k-1)\ v} + s_{(k-1)\ uv}) - (p_{k\ u} + s_{k\ uv})\}$ para k = 1, 2, ..., *m*; e,

$LBY_{[Suv\ v]k} = Max\ \{0, (p_{(k+1)\ u} + s_{(k+1)\ uv}) - (p_{k\ v} + s_{k\ uv}) - UBX_{[Suv\ v]k}\}$ para k = 1, 2, ..., (*m* –1).

Passo 1.3: Seja $[\sum LBY_{Suv\ v}]_{nxn}$ a matriz $\sum_{k=1}^{m-1} LBY_{[Suv\ v]k}$.

Passo 1.4: Faça $[\sum J_v]_{nxn}$ - $[\sum LBY_{Suv\ v}]_{nxn}$, obtendo a matriz $[\Omega]_{nxn}$, em que a posição de cada elemento linha-coluna representa respectivamente as tarefas J_u e J_v.

Passo 2: (caso existam valores iguais pegue o último encontrado)[26]

Passo 2.1: Encontre o maior elemento da matriz $[\Omega]_{nxn}$;

Passo 2.2: Identifique as tarefas adjacentes J_u e J_v, que correspondem, respectivamente, aos elementos linha-coluna da matriz. Assim, J_u ocupa a primeira posição na ordenação, enquanto que, J_v a segunda posição.

Passo 2.3: Encontre a nova tarefa J'_v, ainda não inserida na ordenação, verificando-se o maior elemento da linha da matriz correspondente a última tarefa já ordenada J_v, até que todas as tarefas sejam ordenadas.

(Procedimento de inserção de tarefas)

Passo 3: Selecione as duas primeiras tarefas da ordenação, seqüenciando-as de maneira a minimizar o *Makespan*, considerando somente estas duas tarefas. Não troque a posição relativa dessas duas tarefas, em relação uma a outra, nos passos seguintes. Marque $j = 3$.

Passo 4: Para $j = 3, 4, ..., n$:

Passo 4.1: Selecione a tarefa que ocupa a j-ésima posição na ordenação obtida no Passo 2;

Passo 4.2: Examine as j possibilidades de acrescentar a tarefa na seqüência até então obtida (MSEO), adotando-se aquela que leva um menor *Makespan* da seqüência parcial.

Passo 5: Se $j = n$, a MSEO é a solução obtida, PARE, se não, faça $j = j + 1$ e volte ao Passo 4.

[26] De acordo com a implementação computacional.

ALGORITMO DO MÉTODO BM_m

Fase 1 – Obtenção da seqüência "semente"

A Fase 1 do Método BM_m corresponde ao métodoBM_c, sendo que a solução obtida é, então, denominada seqüência "semente" $\sum^s$, para ser dada como entrada para a fase de melhoramento.

Fase 2 – Melhoramento (Onde [x] representa a tarefa encontrada na x-ésima posição da seqüência $\sum^s$ e [y] representa a tarefa encontrada na y-ésima posição da seqüência $\sum^b$.)

Passo 1: Faça x=1 e $\sum^b = \sum^s$.

Passo 2: Faça y=1.

Passo 3: Se [x] ≠ [y], então gere uma seqüência $\sum^y$ que difere da seqüência $\sum^b$ apenas por ter movido a tarefa [x] para a y-ésima posição e calcule o *Makespan* Q^y da seqüência $\sum^y$; **senão** faça y'=y e prossiga para o passo 4.

Passo 4: Faça y=y+1. Retorne para o passo 3 **se** y ≤ *n*; **senão** prossiga para o passo 5.

Passo 5: Determine a seqüência $\sum^i$ tal que $Q^i = \min \{Q^y \mid y = 1, 2, ..., n \text{ e } y \neq y'\}$. **Se** $Q^i < Q^b$, faça $\sum^b = \sum^i$.

Passo 6: Faça x=x+1. Retorne para o passo 2 **se** x ≤ *n*; **senão** prossiga para o passo 7.

Passo 7: $\sum^b$ é a seqüência obtida. PARE.

Como é possível observar durante a descrição dos algoritmos dos métodos BM, o método BM_c é composto de dois procedimentos, o procedimento de ordenação inicial das tarefas e o procedimento de inserção de tarefas, o método BM_m é composto de duas fases, na primeira fase uma seqüência é gerada, para ser então melhorada na fase seguinte.

É possível observar, também, que o método BM_c e a fase 1 do método BM_m diferem dos métodos NEH e N&M apenas nos passos do procedimento de ordenação inicial das tarefas, e obviamente, no cálculo do *Makespan*.

5 EXPERIMENTAÇÃO COMPUTACIONAL

A experimentação computacional neste trabalho tem por objetivo avaliar os desempenhos dos métodos propostos BM_c e BM_m em relação aos métodos existentes de Cao; Bedworth (1992) e Rajendran; Ziegler (1997).

5.1 DELINEAMENTO DO EXPERIMENTO

O delineamento do experimento inclui a determinação dos intervalos de distribuição dos tempos de processamento e dos tempos de *setup*, a definição da amostragem, a obtenção dos dados e a determinação do processo de análise. Na determinação dos intervalos de distribuição dos tempos de processamento e dos tempos de *setup* e na definição da amostragem são levados em consideração os experimentos realizados por Cao; Bedworth (1992) e Rajendran; Ziegler (1997).

5.1.1 Relações entre os intervalos de distribuição dos tempos de *setup* e o intervalo de distribuição dos tempos de processamento

Para a realização da experimentação computacional os tempos de processamento foram distribuídos no intervalo [1, 99], e os tempos de *setup* nos intervalos [1, 49], [1, 99], [51, 149] e [101, 199] o que leva, assim como em Rajendran; Ziegler (1997), a 4 relações

entre os intervalos de distribuição dos tempos de *setup* (IDs_{kj}) e o intervalo de distribuição dos tempos de processamento (IDp_{kj}), aqui denominadas Relações IDs_{kj}-IDp_{kj}.

Desta forma, sendo o IDp_{kj} sempre [1, 99], as quatro Relações IDs_{kj}-IDp_{kj} são as seguintes:

Relação *i* - IDs_{kj} [1, 49]: a média do intervalo de distribuição dos tempos de *setup* é igual à metade da média do intervalo de distribuição dos tempos de processamento;

Relação *ii* - IDs_{kj} [1, 99]: a média do intervalo de distribuição dos tempos de *setup* é igual à média do intervalo de distribuição dos tempos de processamento;

Relação *iii* - IDs_{kj} [51, 149]: a média do intervalo de distribuição dos tempos de *setup* é igual ao dobro da média do intervalo de distribuição dos tempos de processamento

Relação *iv* - IDs_{kj} [101, 199]: a média do intervalo de distribuição dos tempos de *setup* e remoção é igual ao triplo da média do intervalo de distribuição dos tempos de processamento.

Os tempos de processamento e de *setup*, números inteiros, foram gerados aleatoriamente.

5.1.2 Definição da amostragem

Na experimentação computacional final analisou-se um total de 3600 problemas, divididos em 30 classes de problemas, de acordo com o número de tarefas (n) e o número de máquinas (m), para $n \in \{4, 6, 7, 20, 40, 60\}$ e $m \in \{5, 10, 15, 20, 25\}$. Para cada classe de problemas, foram testados, objetivando a minimização do erro amostral, 120 problemas, gerados aleatoriamente. Os problemas foram subdivididos em 2 grupos de problemas.

No primeiro grupo, testou-se 1800 problemas, considerados de pequeno porte. Este grupo foi constituído de 15 classes de problemas com $n \in \{4, 6, 7\}$ e $m \in \{5, 10, 15, 20, 25\}$, ou seja, 3 (alternativas do número de tarefas) x 5 (alternativas do número de máquinas) = 15 x 120 (quantidade de problemas analisados por classe) = 1800.

No segundo grupo, constituído, também, de 15 classes de problemas com $n \in \{20, 40, 60\}$ e $m \in \{5, 10, 15, 20, 25\}$, foram testados 1800 problemas, considerados de grande porte.

Os 120 problemas testados, em cada uma classes, foram subdivididos em 4 tipos de 30 de problemas cada, onde os valores dos tempos *setup* e os tempos de processamento foram gerados conforme as 4 Relações IDs_{kj}-IDp_{kj}.

É possível observar que foram gerados 6 classes, com números de tarefas diferentes, e 5 classes distintas de números de máquinas.

5.1.3 Obtenção dos problemas e dos resultados

Considerando-se as 4 Relações IDs_{kj}-IDp_{kj} e os parâmetros definidos na definição da amostragem, foram gerados, aleatoriamente, todos os problemas, utilizando-se de um *software*, especificamente criado para este fim, designado "Gerador de arquivos". A Figura 5.1 apresenta a interface deste *software*.

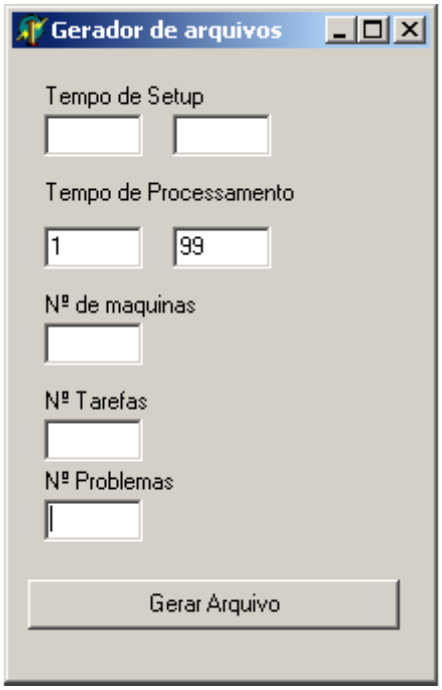

Figura 5.1. Interface do "Gerador de arquivos" de dados

Para resolver os problemas gerados, construiu-se um *software*, intitulado "PF/s.n-b.i.ST/Cmax", cuja interface é mostrada na Figura 5.2. Este é um *software* para resolver o problema de programação em *Flow* Shop Permutacional (PF) com tempos de *setup* separados dos tempos de processamento e independentes da seqüência de execução das tarefas (s.n-b.i.ST, em inglês, *separated, non-batch, and sequence-independent setup times*) com critério de minimização dos *Makespan* (C_{max}).

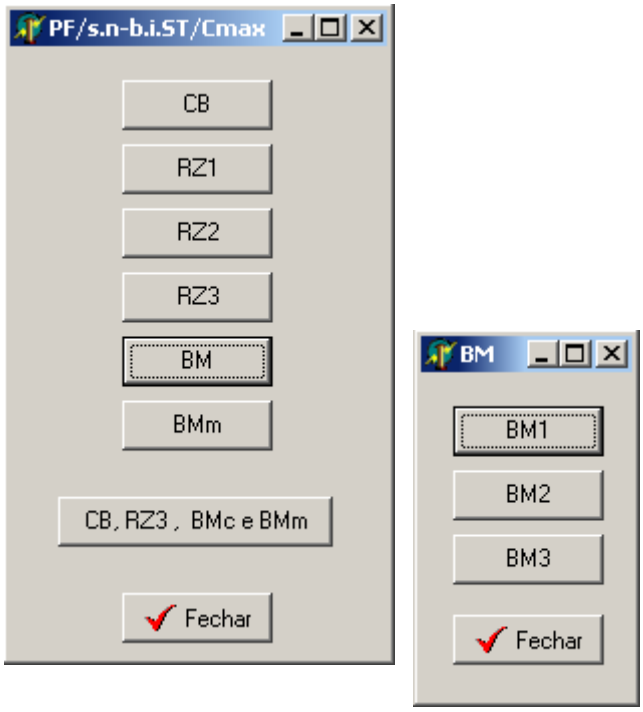

Figura 5.2. Interface do software "PF/s.n-b.i.ST/Cmax"

O *software* "PF/s.n-b.i.ST/Cmax" resolve os problemas de uma determinada classe e Relação IDs_{kj}-IDp_{kj} pelos métodos CB, RZ_1, RZ_2, RZ_3, BM_1, BM_2 (também designado BM_c), BM_3 e BM_m, separadamente, pelos respectivos botões, e os métodos CB, RZ_3, BM_c e BM_m, utilizando um botão conjunto. Tal *software* gera um arquivo de saída em formato *.txt* com o valor do *Makespan* da respectiva programação gerada por cada método, bem como do tempo total de computação para resolver todos os problemas da classe.

Para a construção dos *softwares* "Gerador de arquivos" e "PF/s.n-b.i.ST/Cmax", utilizou-se a linguagem de programação Borland Delphi 6 e o sistema operacional Windowns XP (Versão 2002 SP2). As configurações da máquina[27] são as seguintes: processador Pentium 4 da Intel com 3.2 GHz de freqüência, 1 GB de memória RAM e disco rígido de 60 GB.

[27] De propriedade da autora.

5.1.4 Processo de análise

Os resultados obtidos na experimentação computacional são analisados em relação à porcentagem de sucesso, ao desvio relativo médio (%) e ao tempo médio de computação (ms) dos métodos CB, RZ_3, BM_c e BM_m para cada classe de problemas.

Porcentagem de sucesso

A porcentagem de sucesso é calculada pela quantidade de vezes em que o método forneceu a melhor solução, isoladamente ou não, dividido pelo total de problemas analisados (na classe, no grupo de problemas ou em todos os problemas). Logo, a porcentagem de sucesso de um método *h* ($\%S_h$):

$$\%S_h = \frac{q_h}{T}$$

Onde:

q_h: quantidade de vezes em que o método *h* forneceu a melhor solução, isoladamente ou não

T: total de problemas analisados

Desvio relativo médio (%)

O desvio relativo de um método, também denominado qualidade relativa, mede a variação da solução obtida com esse método em relação à melhor solução obtida entre todos os métodos analisados.

Quando o desvio relativo da solução de um método para um determinado problema é igual a zero, este método forneceu a melhor solução, isoladamente ou não, ou seja, neste trabalho, significa que a solução obtida com o método leva ao menor *Makespan*.

O desvio relativo de um método *h* para um determinado problema *p* (DR_{hp}) pode ser assim calculado:

$$DR_{hp} = \frac{D_{hp} - D_p^*}{D_p^*}$$

Onde:

D_{hp} = *Makespan* obtido por um método *h*, em um determinado problema *p*

D_p^* = melhor *Makespan* obtido, para um determinado problema *p*, por todos métodos analisados

Logo, o desvio relativo médio de um método *h* em uma determinada classe de problema *n*, *m* ($DRM_{h(n,m)}$) é dada pela média aritmética dos desvios relativos dos problemas da classe que tiveram desvio relativo diferente de zero. O $DRM_{h(m,n)}$ pode ser escrito por meio da seguinte expressão:

$$DRM_{h(n,m)} = \frac{DR_{h[1]} + DR_{h[2]} + \ldots\ldots + DR_{h[x]}}{t}$$

Onde:

[*x*] = determinado problema de uma classe em que um determinado método *h* obteve desvio relativo diferente de zero

t = quantidade de problemas em uma determinada classe *n*, *m* em que um método *h* obteve desvio relativo diferente de zero

Tempo médio de computação (ms)

O tempo médio de computação de um determinado método *h* (TMC_h) é calculado pela soma dos tempos de computação em cada problema, dividida pelo número total de problemas resolvidos, ou seja, é a média aritmética dos tempos de computação:

$$TMC_h = \frac{\sum_{p=1}^{P} TC_{hp}}{P}$$

Onde:

TC_{hp} = tempo de computação de um determinado método *h*, em um determinado problema *p*

P = número total de problemas resolvidos

Na experimentação computacional o tempo médio de computação foi medido em milissegundos (ms).

5.2 RESULTADOS OBTIDOS E ANÁLISES

Para a tabulação dos dados e análise dos resultados, utilizou-se o Microsoft Excell 5.0/95.

Os resultados obtidos na experimentação computacional são agrupados em relação ao número de tarefas.

Objetivando a melhor visualização e, conseqüente, análise dos resultados obtidos, estes são apresentados em tabelas e gráficos de linhas e de colunas.

No caso da porcentagem de sucesso os resultados são analisados em relação às classes (*n*, *m*) e em função das Relações IDs_{kj}-IDp_{kj}. Assim, são apresentados gráficos para todos os valores de *n* em função de cada uma das Relações e em função do número de máquinas *m*. No primeiro caso, o objetivo é avaliar a influência do número de máquinas *m* no desempenho de cada método, quando cada Relação é considerada separadamente e quando consideradas em conjunto. Enquanto que, no segundo caso, o objetivo é avaliar a influência das Relações IDs_{kj}-IDp_{kj} em cada uma das classes (*n*, *m*). Também apresenta-se uma análise da porcentagem de sucesso nos problemas de pequeno e grande porte e em todos os problemas.

Quanto ao desvio relativo médio (%) e ao tempo médio de computação (ms), os resultados são analisados em função das classes (n, m), agregando-se as Relações IDs_{kj}-IDp_{kj}.

Para efeitos de simplificação, nos gráficos, a porcentagem de sucesso, o desvio relativo médio (%), o tempo de médio de computação (ms) e o método RZ_3 são designados, respectivamente, %S, DRM (%), TMC (ms) e RZ.

5.2.1 Porcentagem de sucesso

As Tabelas 5.1, 5.3 e 5.5 mostram os resultados da porcentagem de sucesso para $n \in \{4, 6, 7\}$ em função das Relações IDs_{kj}-IDp_{kj} e do número de máquinas (m). Nas Tabelas 5.2, 5.4 e 5.6 os resultados da porcentagem de sucesso para $n \in \{4, 6, 7\}$ são apresentados em função do número de máquinas (m), agregando-se as Relações IDs_{kj}-IDp_{kj}.

Tabela 5.1 - Porcentagem de sucesso para $n = 4$

Relação IDs_{kj}-IDp_{kj}	m	**MÉTODOS**			
		CB	**RZ_3**	**BM_c**	**BM_m**
i	5	73,33	96,67	80,00	90,00
	10	83,33	100,00	80,00	96,67
	15	80,00	100,00	76,67	90,00
	20	93,33	100,00	76,67	93,33
	25	76,67	96,67	76,67	90,00
ii	5	76,67	96,67	60,00	90,00
	10	70,00	96,67	66,67	100,00
	15	76,67	96,67	80,00	93,33
	20	66,67	96,67	50,00	83,33
	25	73,33	93,33	73,33	93,33
iii	5	70,00	93,33	66,67	93,33
	10	70,00	100,00	73,33	90,00
	15	73,33	100,00	76,67	96,67
	20	63,33	96,67	66,67	86,67
	25	83,33	93,33	60,00	83,33
iv	5	53,33	93,33	76,67	96,67
	10	63,33	100,00	80,00	93,33
	15	66,67	93,33	70,00	90,00
	20	73,33	93,33	70,00	93,33
	25	73,33	100,00	70,00	86,67

Os Gráficos 5.1 a 5.4 mostram, para $n = 4$, a comparação da porcentagem de sucesso entre os métodos CB, RZ_3, BM_c e BM_m para as quatro Relações IDs_{kj}-IDp_{kj}.

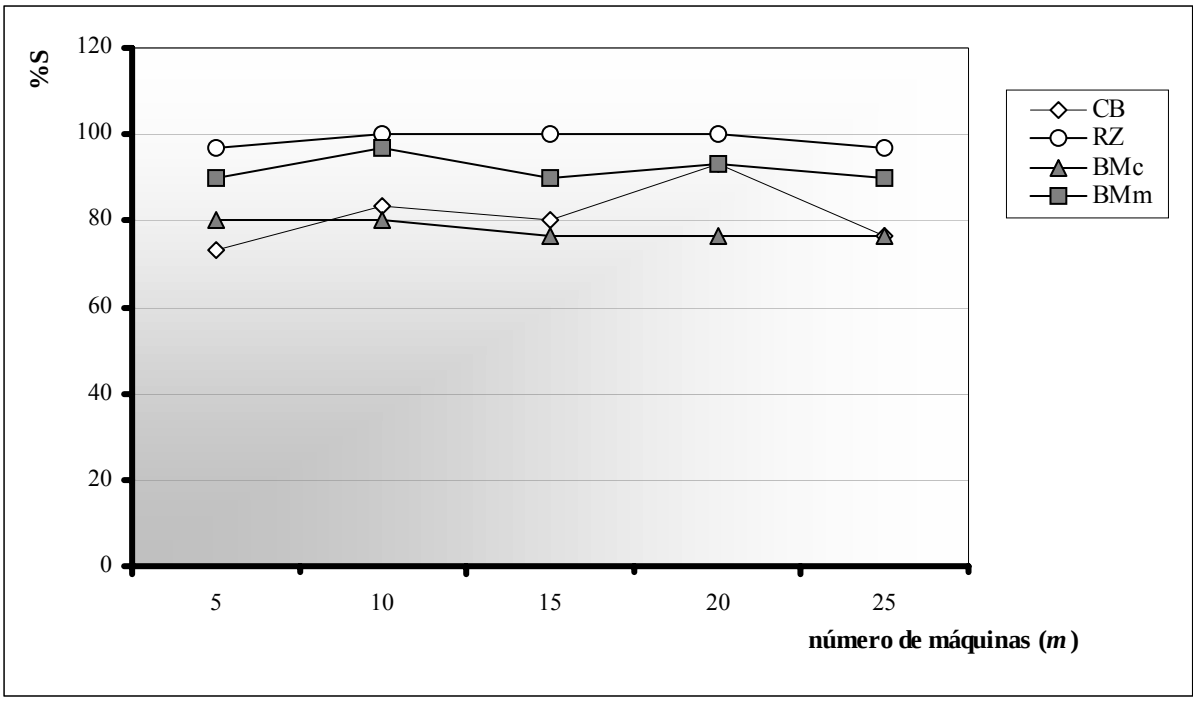

Gráfico 5.1. Porcentagem de sucesso para $n = 4$ – Relação *i*

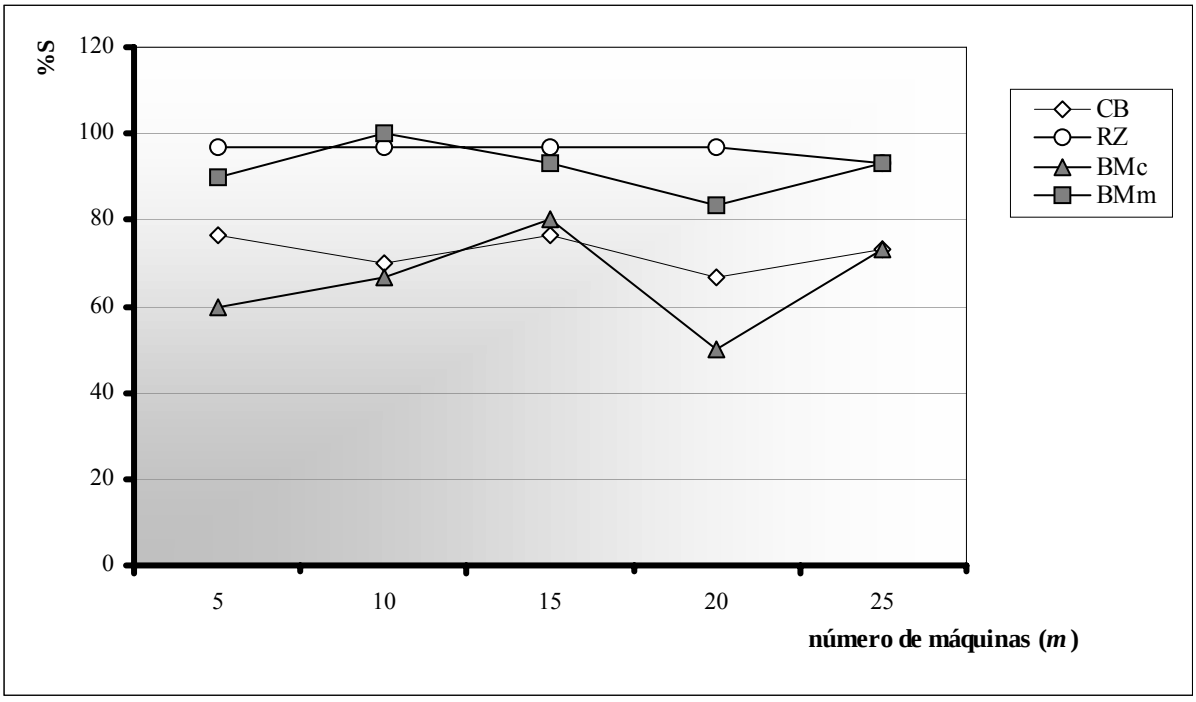

Gráfico 5.2. Porcentagem de sucesso para $n = 4$ – Relação *ii*

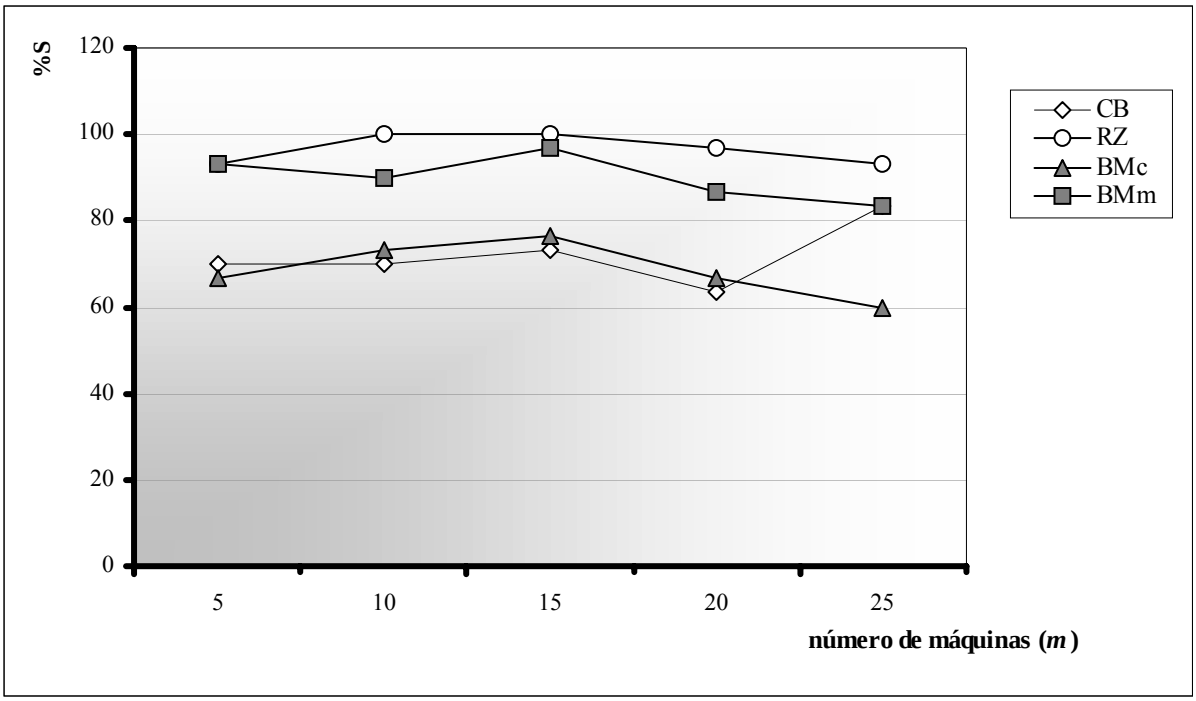

Gráfico 5.3. Porcentagem de sucesso para $n = 4$ – Relação *iii*

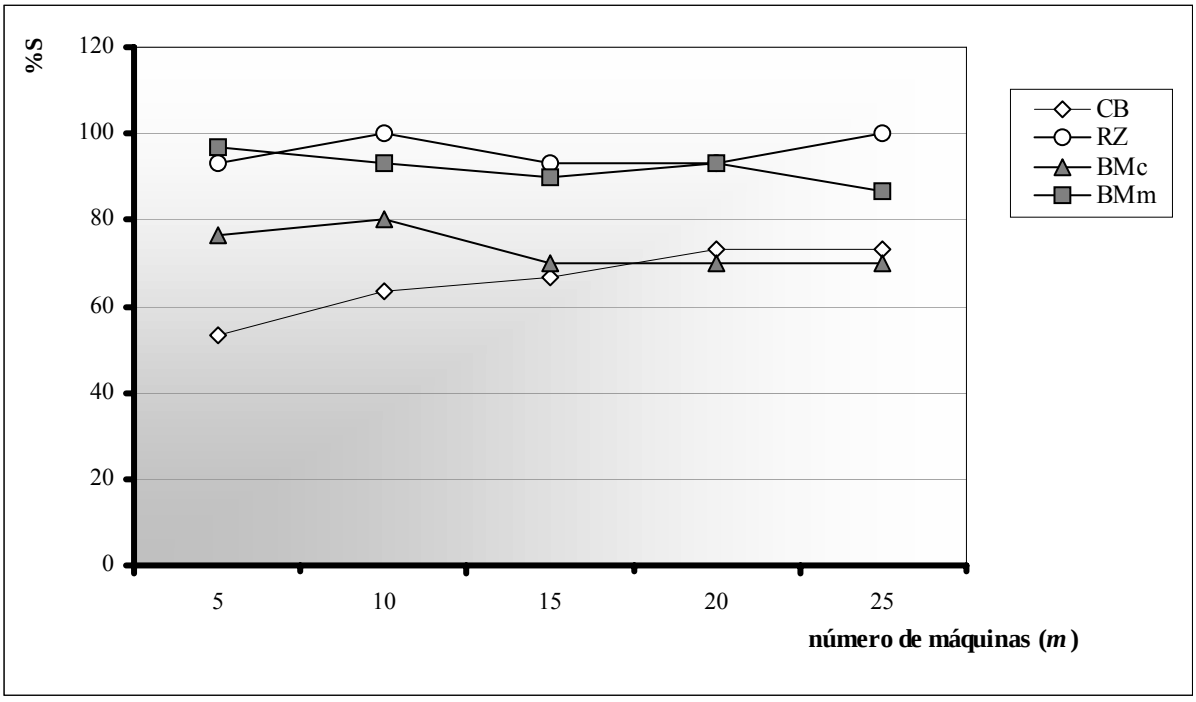

Gráfico 5.4. Porcentagem de sucesso para $n = 4$ – Relação *iv*

Os Gráficos 5.5 a 5.9 mostram, para $n = 4$, a comparação da porcentagem de sucesso entre os métodos CB, RZ_3, BM_c e BM_m para todos os valores de m.

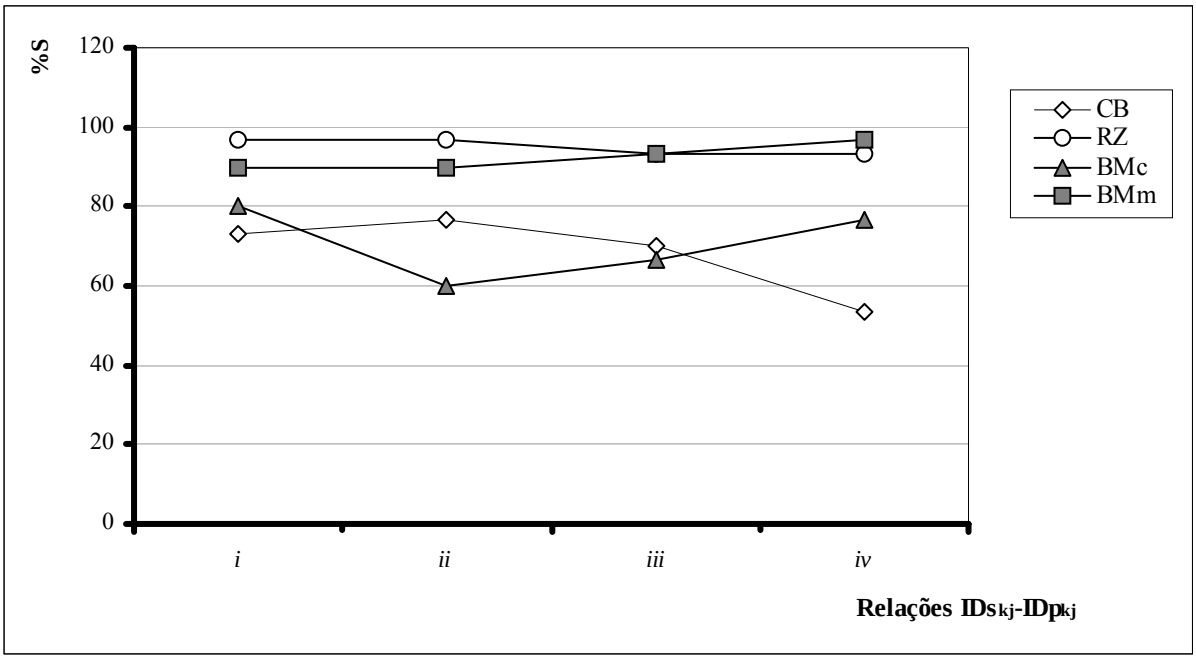

Gráfico 5.5. Porcentagem de sucesso para $n = 4$ e $m = 5$

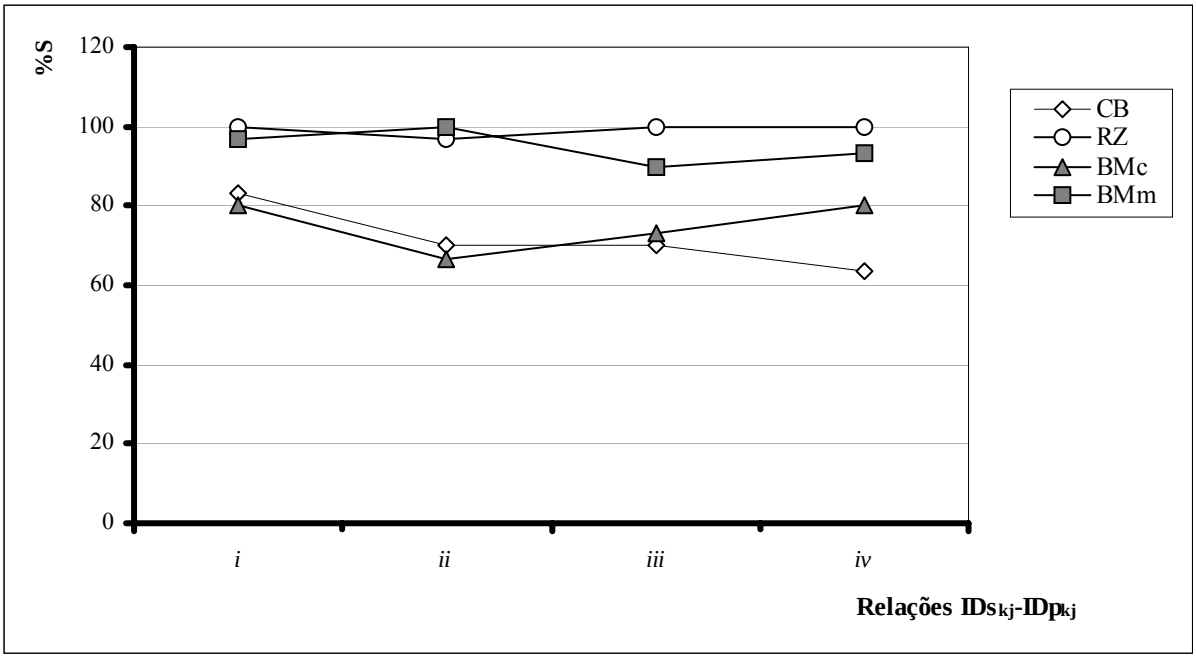

Gráfico 5.6. Porcentagem de sucesso para $n = 4$ e $m = 10$

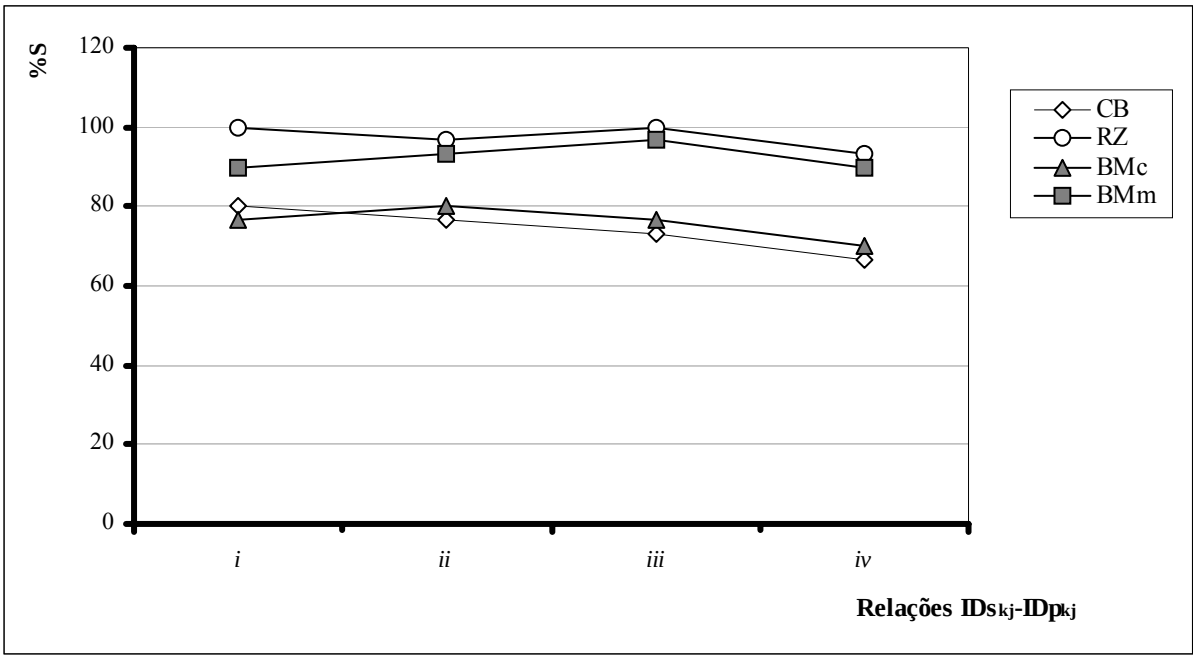

Gráfico 5.7. Porcentagem de sucesso para $n = 4$ e $m = 15$

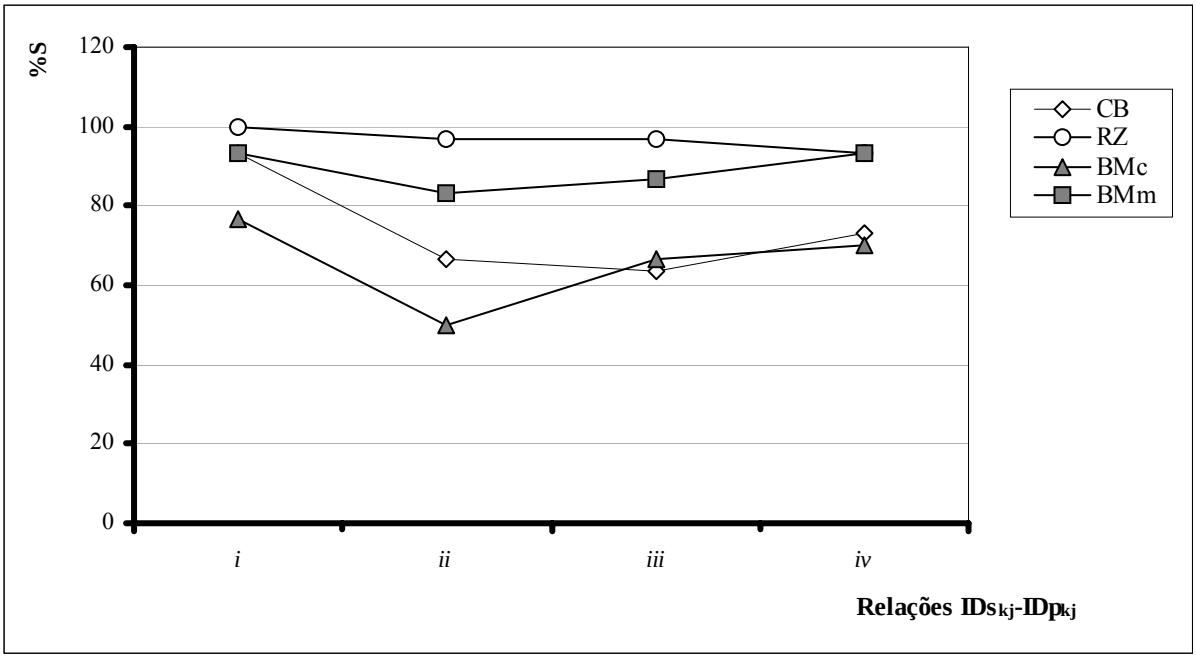

Gráfico 5.8. Porcentagem de sucesso para $n = 4$ e $m = 20$

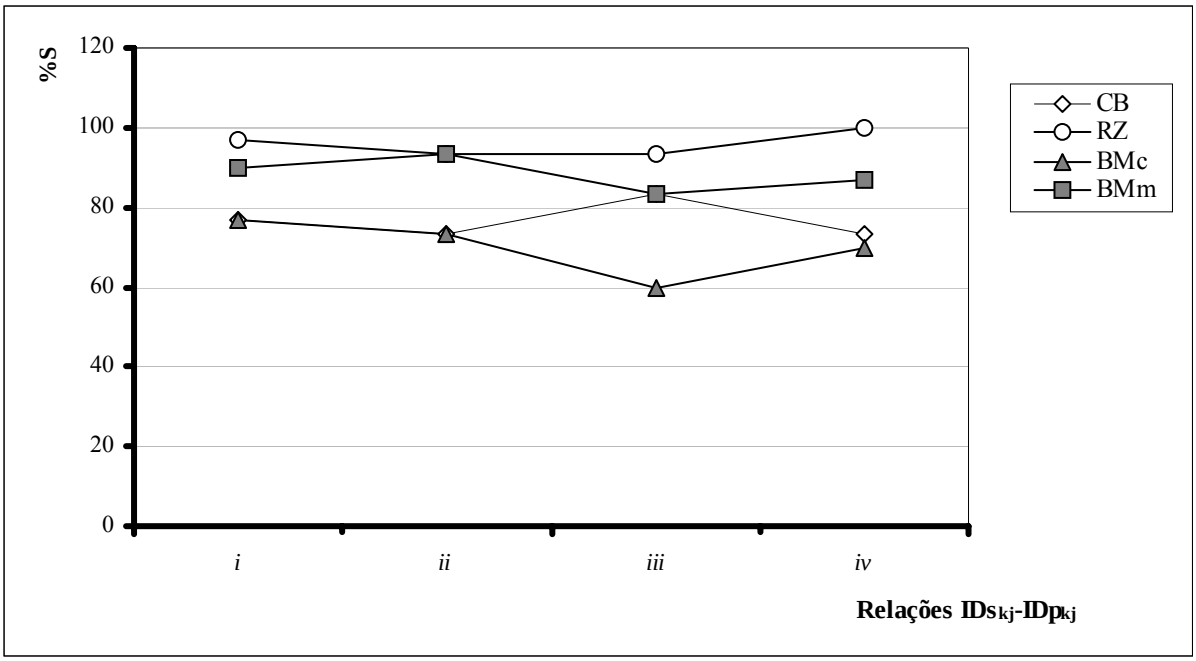

Gráfico 5.9. Porcentagem de sucesso para $n = 4$ e $m = 25$

Analisando-se os Gráficos 5.1 a 5.9 é possível constatar que, no caso de $n = 4$, não houve influência do número de máquinas m e das Relações IDs_{kj}-IDp_{kj} no desempenho dos métodos.

Tabela 5.2 - Porcentagem de sucesso, agregando-se as Relações IDs_{kj}-IDp_{kj} - $n = 4$

	MÉTODOS			
m	**CB**	**RZ_3**	**BM_c**	**BM_m**
5	68,33	95,00	70,83	92,50
10	71,67	99,17	75,00	95,00
15	74,17	97,50	75,83	92,50
20	74,17	96,67	65,83	89,17
25	76,67	95,83	70,00	88,33

O Gráfico 5.10 apresenta, para $n = 4$, a comparação da porcentagem de sucesso entre os métodos CB, RZ_3, BM_c e BM_m, agregando-se as Relações IDs_{kj}-IDp_{kj}.

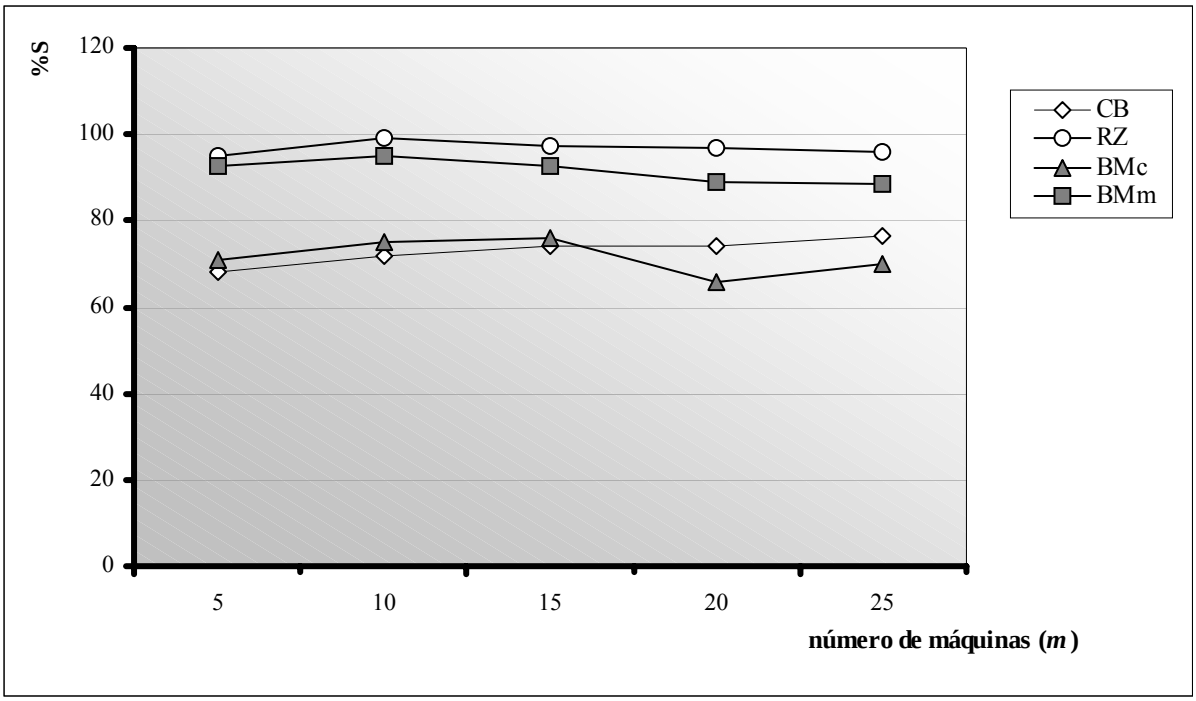

Gráfico 5.10. Porcentagem de sucesso, agregando-se as Relações IDs_{kj}-IDp_{kj} - $n = 4$

Observando o Gráfico 5.10 é possível concluir que, no caso de $n = 4$, a ordem de superioridade no desempenho é a seguinte: RZ_3; BM_m; CB; BM_c. A porcentagem de sucesso média do método BM_m é muito próxima da do método RZ_3 (apenas 5,33 pontos percentuais menor, em média), assim como a porcentagem média de sucesso do método BM_c é muito próxima da do método CB (apenas 1,5 pontos percentuais menor, em média).

Tabela 5.3 - Porcentagem de sucesso para $n = 6$

		MÉTODOS			
Relação IDs_{kj}-IDp_{kj}	*m*	**CB**	**RZ_3**	**BM_c**	**BM_m**
i	5	33,33	86,67	60,00	86,67
	10	63,33	96,67	63,33	80,00
	15	43,33	86,67	53,33	73,33
	20	43,33	93,33	66,67	83,33
	25	50,00	90,00	56,67	80,00
ii	5	43,33	90,00	66,67	83,33
	10	43,33	90,00	70,00	90,00
	15	56,67	93,33	63,33	86,67
	20	46,67	90,00	73,33	83,33
	25	50,00	86,67	46,67	83,33
iii	5	43,33	93,33	63,33	86,67
	10	36,67	83,33	53,33	73,33
	15	50,00	93,33	70,00	86,67
	20	50,00	76,67	63,33	86,67
	25	46,67	96,67	70,00	90,00
iv	5	43,33	90,00	60,00	86,67
	10	40,00	90,00	56,67	83,33
	15	43,33	86,67	46,67	93,33
	20	36,67	76,67	50,00	86,67
	25	36,67	86,67	63,33	80,00

Os Gráficos 5.11 a 5.14 mostram, para $n = 6$, a comparação da porcentagem de sucesso entre os métodos CB, RZ_3, BM_c e BM_m para as quatro Relações IDs_{kj}-IDp_{kj}.

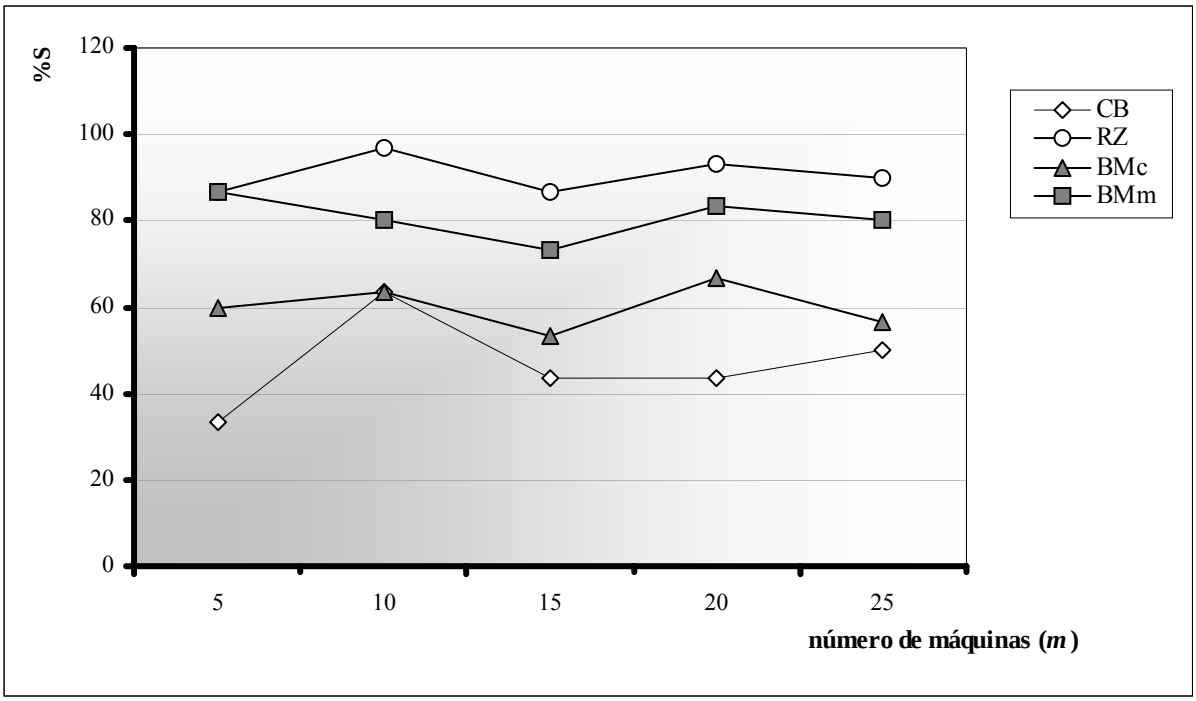

Gráfico 5.11. Porcentagem de sucesso para $n = 6$ – Relação *i*

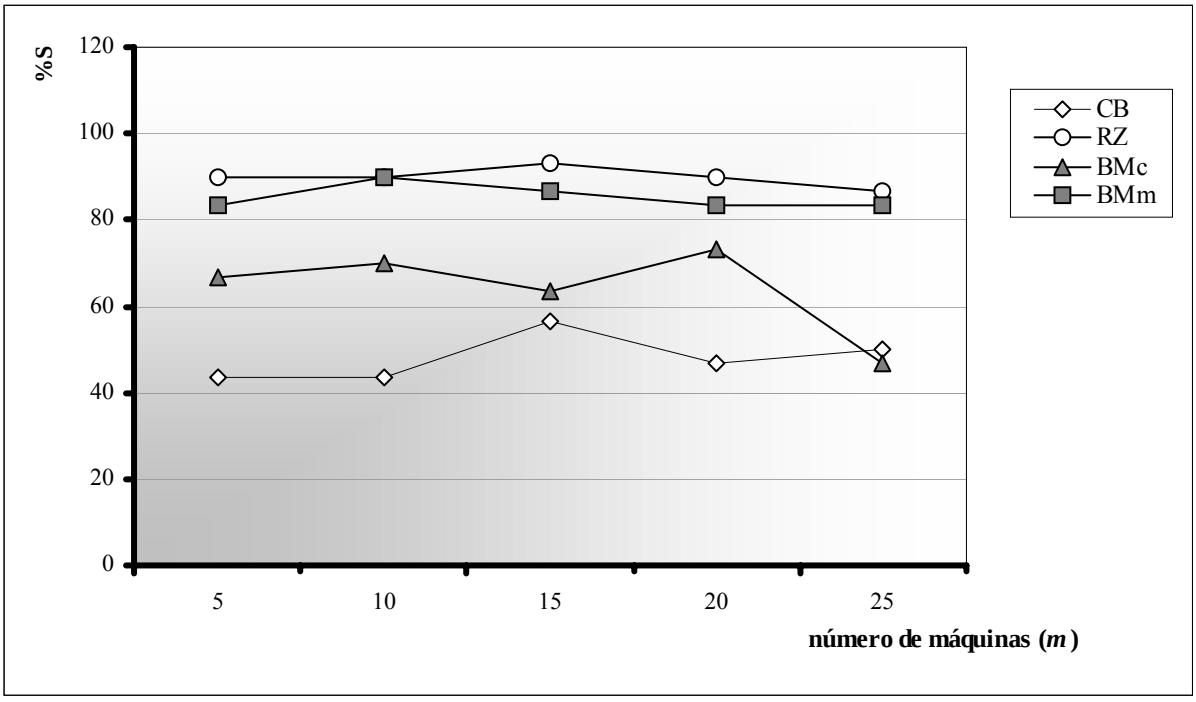

Gráfico 5.12. Porcentagem de sucesso para $n = 6$ – Relação *ii*

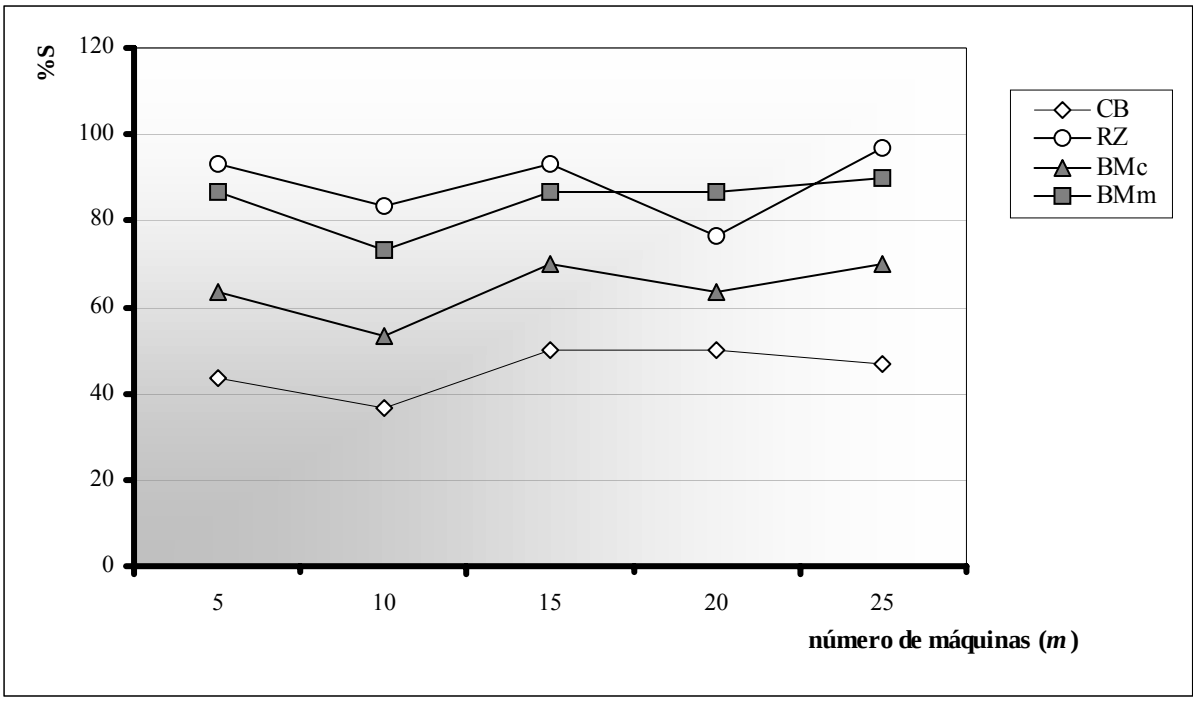

Gráfico 5.13. Porcentagem de sucesso para $n = 6$ – Relação *iii*

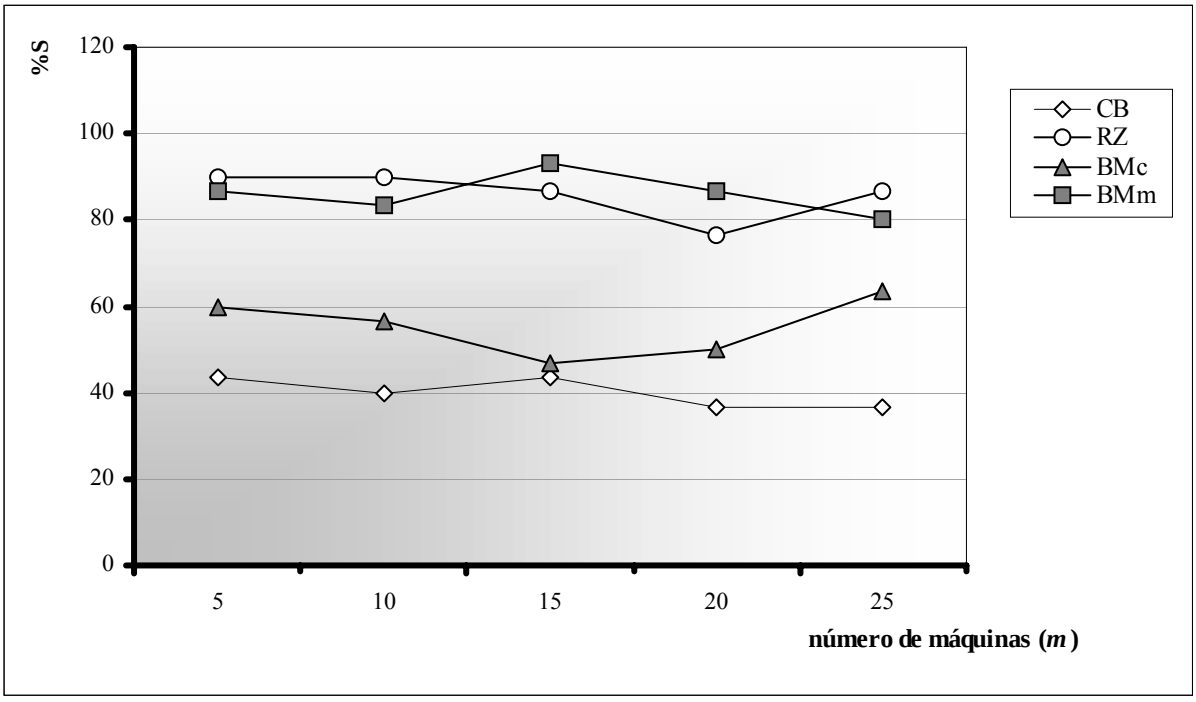

Gráfico 5.14. Porcentagem de sucesso para $n = 6$ – Relação *iv*

Os Gráficos 5.15 a 5.19 mostram, para $n = 6$, a comparação da porcentagem de sucesso entre os métodos CB, RZ_3, BM_c e BM_m para todos os valores de m.

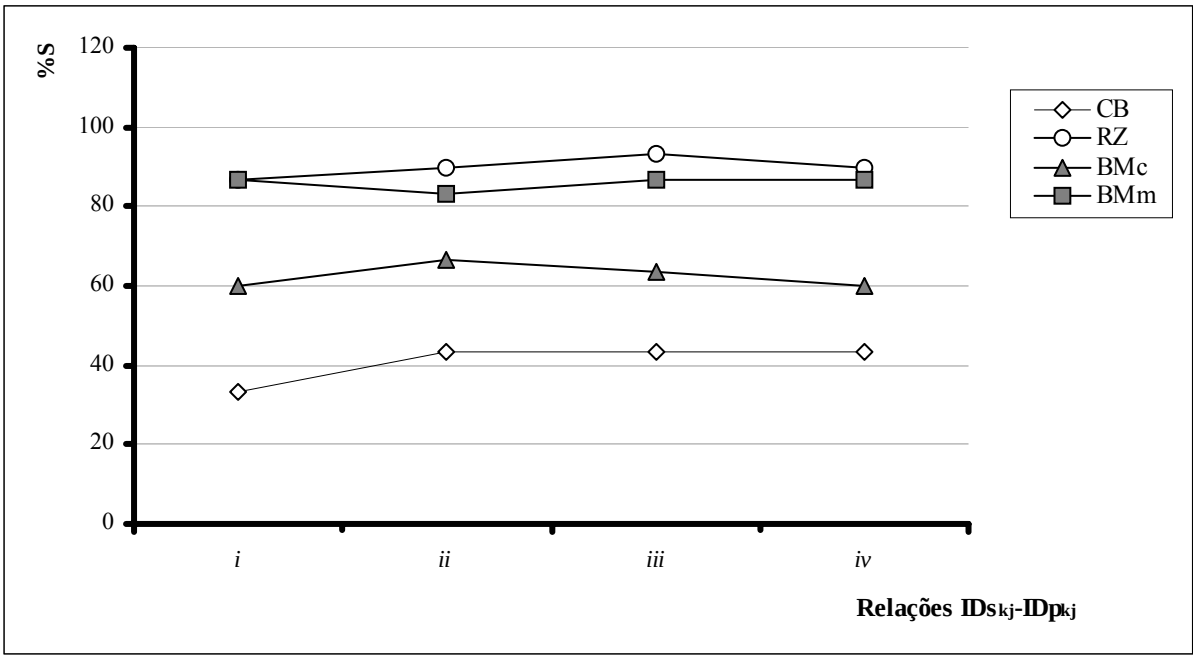

Gráfico 5.15. Porentagem de sucesso para $n = 6$ e $m = 5$

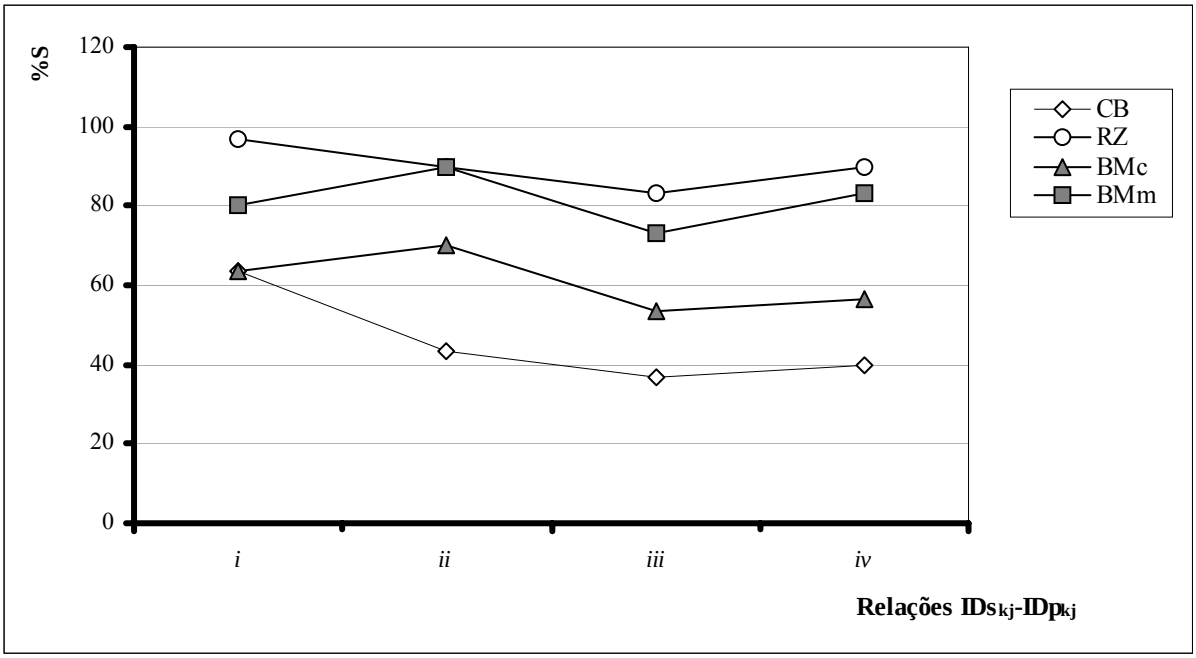

Gráfico 5.16. Porentagem de sucesso para $n = 6$ e $m = 10$

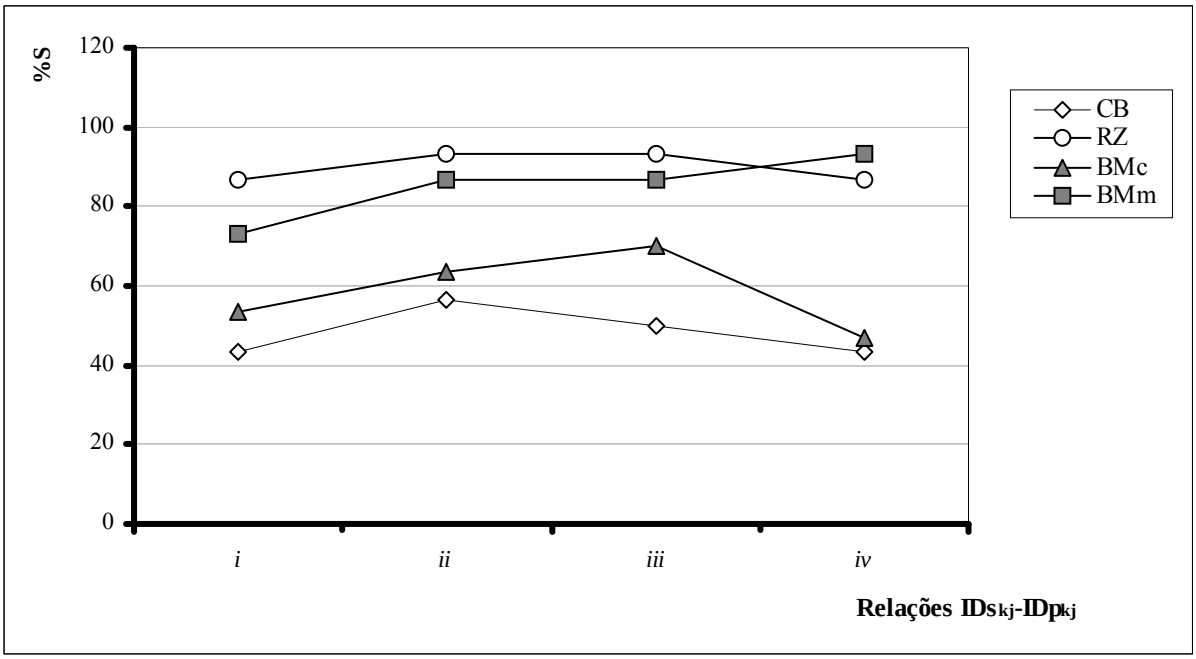

Gráfico 5.17. Porentagem de sucesso para $n = 6$ e $m = 15$

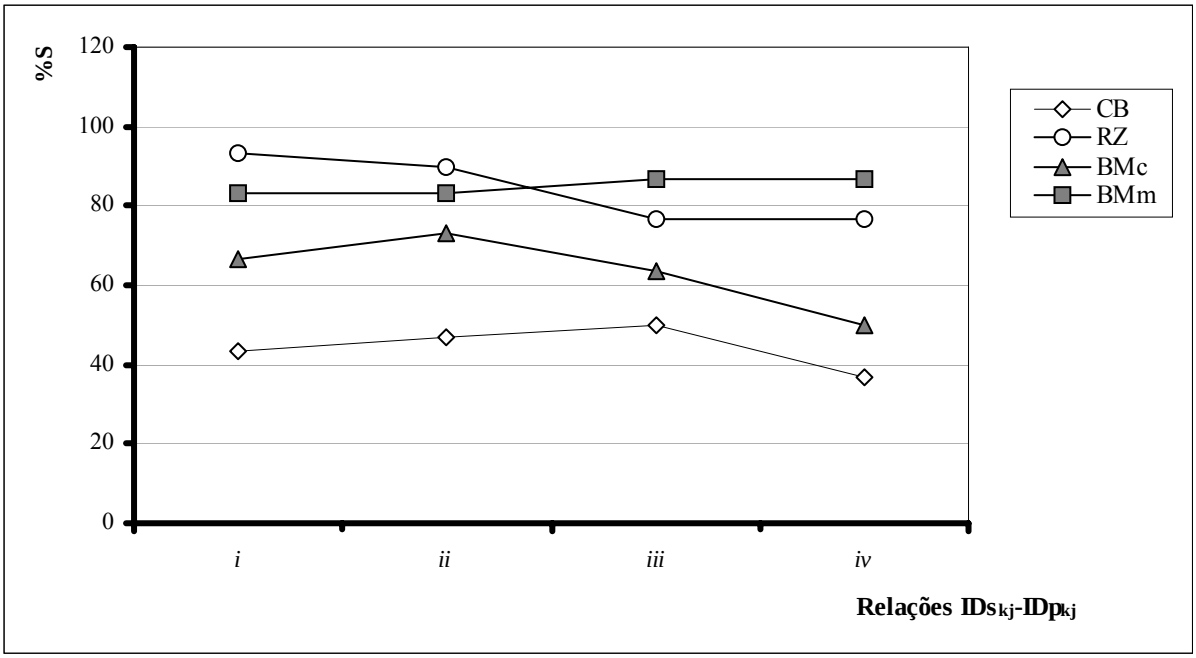

Gráfico 5.18. Porentagem de sucesso para $n = 6$ e $m = 20$

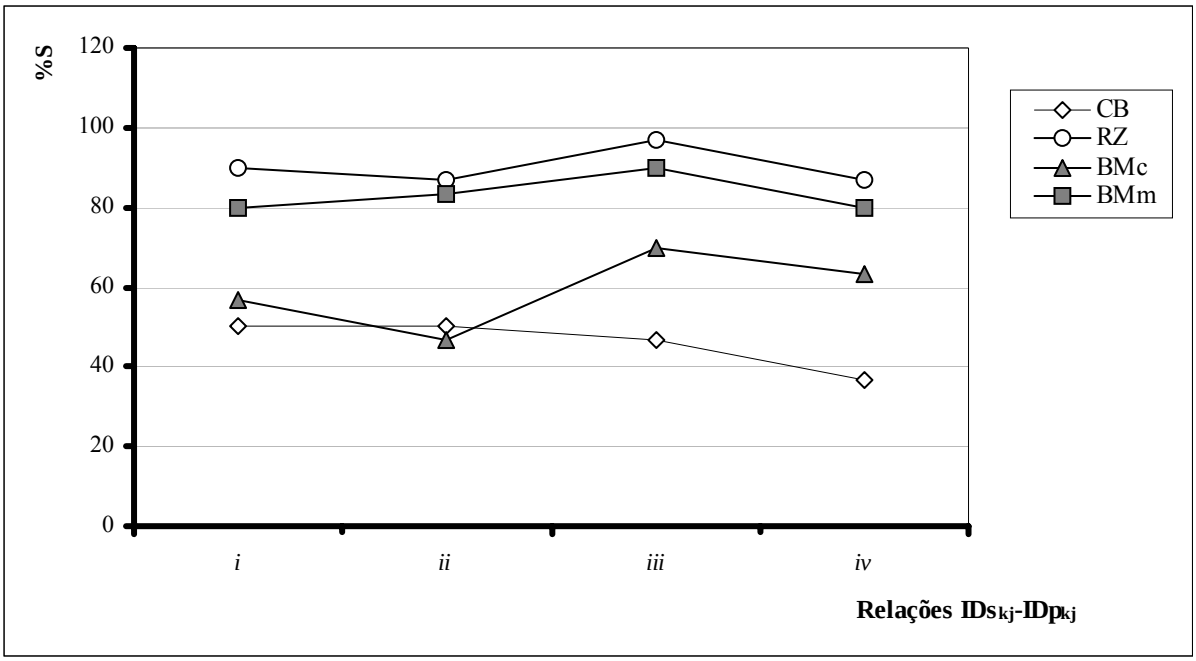

Gráfico 5.19. Porcentagem de sucesso para $n = 6$ e $m = 25$

Tendo como base os Gráficos 5.11 a 5.19 é possível constatar que, no caso de $n = 6$, não ocorreu influência do número de máquinas m e das Relações IDs_{kj}-IDp_{kj} no desempenho de cada método.

Tabela 5.4 - Porcentagem de sucesso, agregando-se as Relações IDs_{kj}-IDp_{kj} - $n = 6$

	MÉTODOS			
m	**CB**	**RZ_3**	**BM_c**	**BM_m**
5	40,83	90,00	62,50	85,83
10	45,83	90,00	60,83	81,67
15	48,33	90,00	58,33	85,00
20	44,17	84,17	63,33	85,00
25	45,83	90,00	59,17	83,33

O Gráfico 5.20 apresenta, para $n = 6$, a comparação da porcentagem de sucesso entre os métodos CB, RZ_3, BM_c e BM_m, agregando-se as Relações IDs_{kj}-IDp_{kj}.

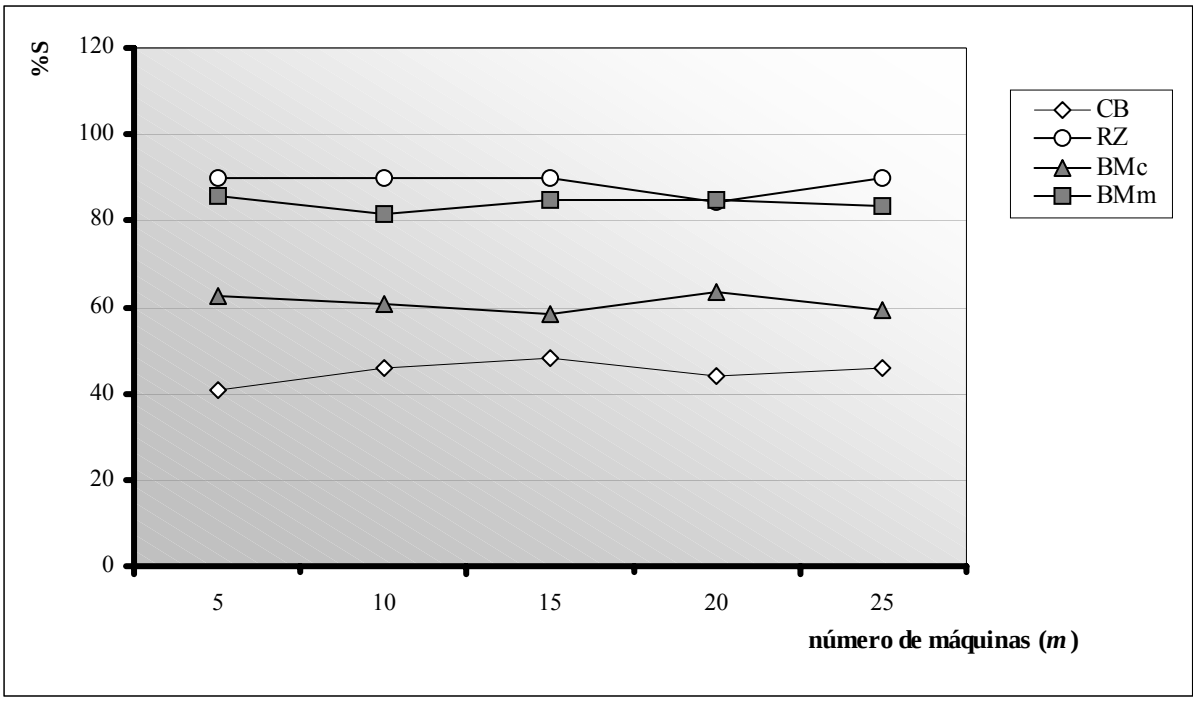

Gráfico 5.20. Porcentagem de sucesso, agregando-se as Relações IDs_{kj}-IDp_{kj} - $n = 6$

Observando o Gráfico 5.20 é possível concluir que, no caso de $n = 6$, a ordem de superioridade no desempenho é a seguinte: RZ_3; BM_m; BM_c; CB. No entanto, a porcentagem de sucesso média do método BM_m é muito próxima da do método RZ_3 (apenas 4,67 pontos percentuais menor, em média)..

Tabela 5.5 - Porcentagem de sucesso para $n = 7$

Relação IDs_{kj}-IDp_{kj}	m	**MÉTODOS** CB	RZ_3	BM_c	BM_m
i	5	33,33	96,67	60,00	73,33
	10	43,33	93,33	53,33	83,33
	15	40,00	90,00	46,67	80,00
	20	36,67	83,33	60,00	83,33
	25	60,00	86,67	60,00	86,67
ii	5	43,33	83,33	56,67	73,33
	10	53,33	93,33	56,67	90,00
	15	40,00	83,33	53,33	90,00
	20	43,33	86,67	63,33	70,00
	25	40,00	90,00	66,67	83,33
iii	5	20,00	96,67	66,67	76,67
	10	46,67	96,67	53,33	96,67
	15	23,33	90,00	50,00	73,33
	20	46,67	90,00	53,33	80,00
	25	33,33	90,00	50,00	93,33
iv	5	36,67	96,67	43,33	76,67
	10	40,00	90,00	60,00	83,33
	15	36,67	93,33	56,67	86,67
	20	43,33	83,33	60,00	90,00
	25	63,33	83,33	73,33	93,33

Os Gráficos 5.21 a 5.24 mostram, para $n = 7$, a comparação da porcentagem de sucesso entre os métodos CB, RZ_3, BM_c e BM_m para as quatro Relações IDs_{kj}-IDp_{kj}.

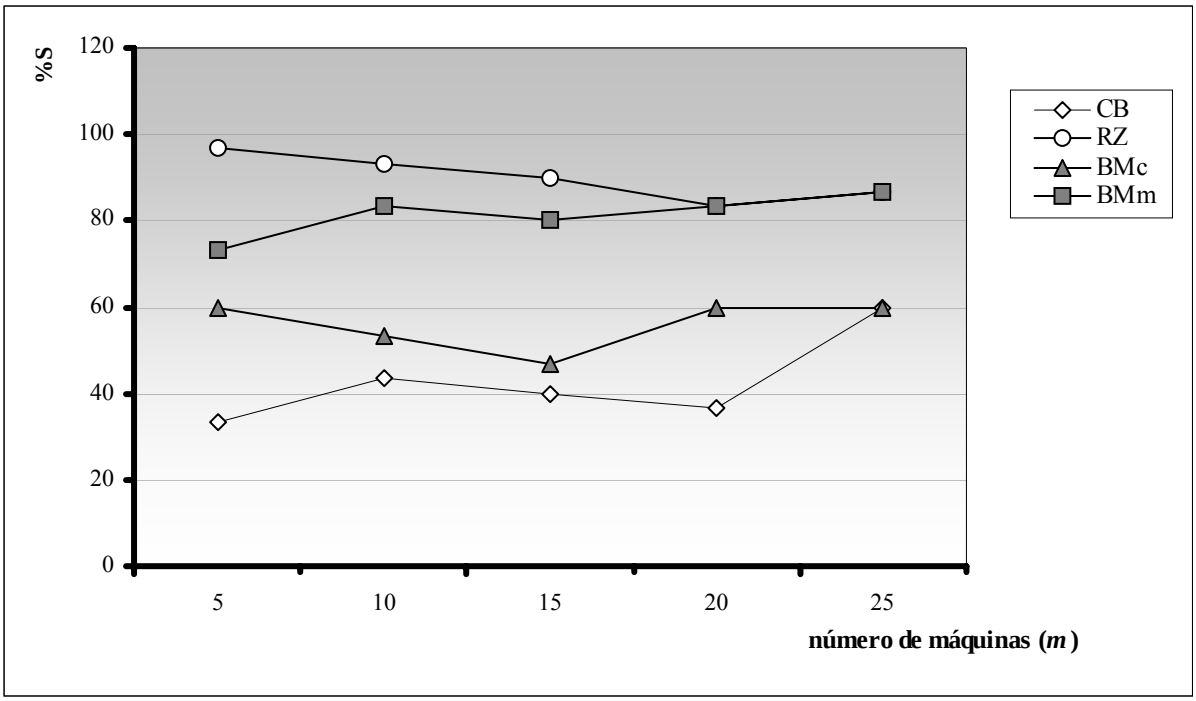

Gráfico 5.21. Porcentagem de sucesso para $n = 7$ – Relação *i*

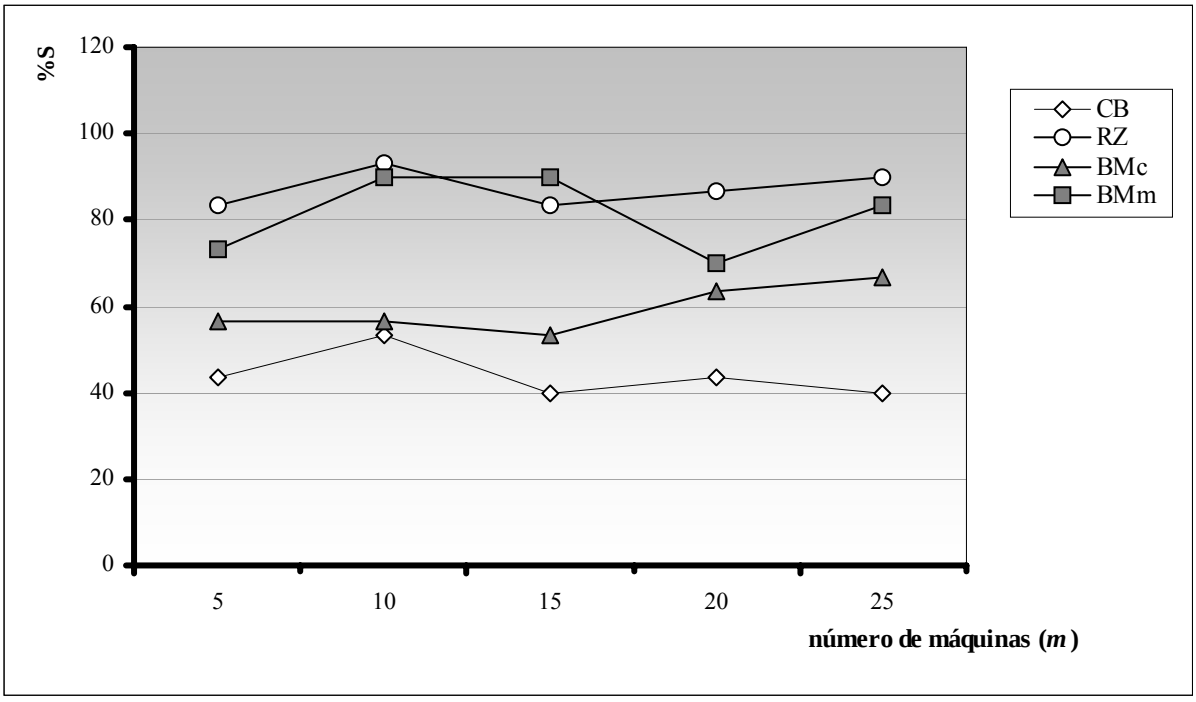

Gráfico 5.22. Porcentagem de sucesso para $n = 7$ – Relação *ii*

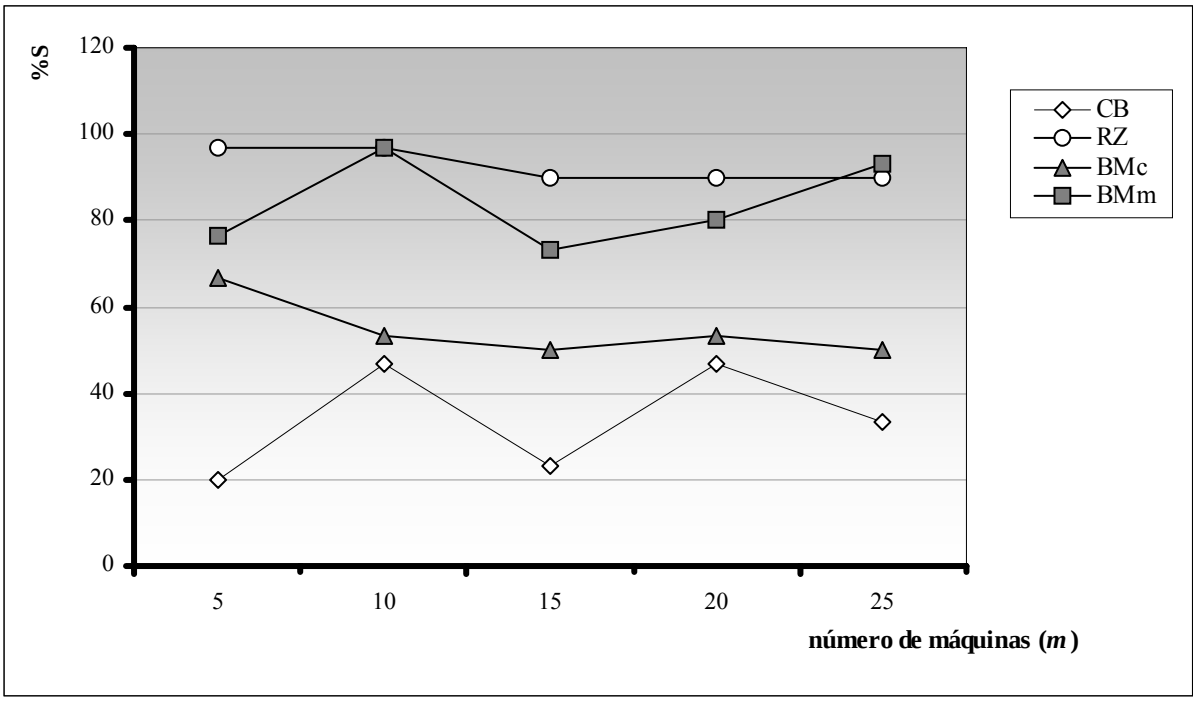

Gráfico 5.23. Porcentagem de sucesso para $n = 7$ – Relação *iii*

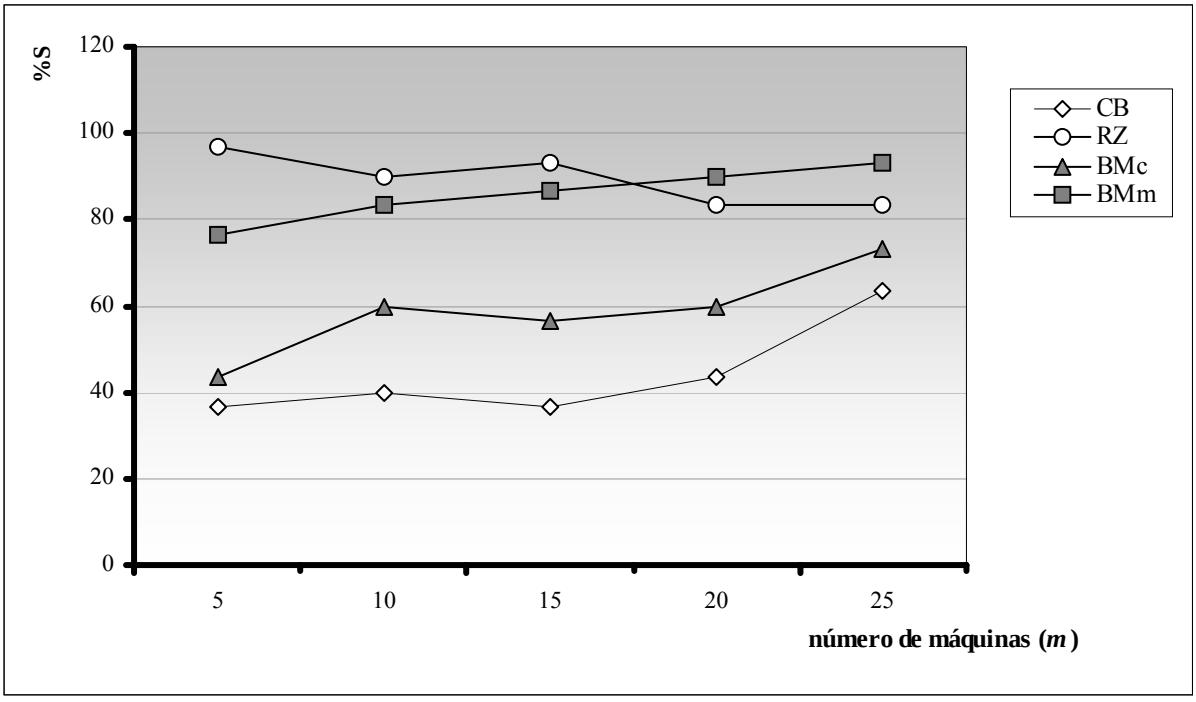

Gráfico 5.24. Porcentagem de sucesso para $n = 7$ – Relação *iv*

Os Gráficos 5.25 a 5.29 mostram, para n = 7, a comparação da porcentagem de sucesso entre os métodos CB, RZ_3, BM_c e BM_m para todos os valores de m.

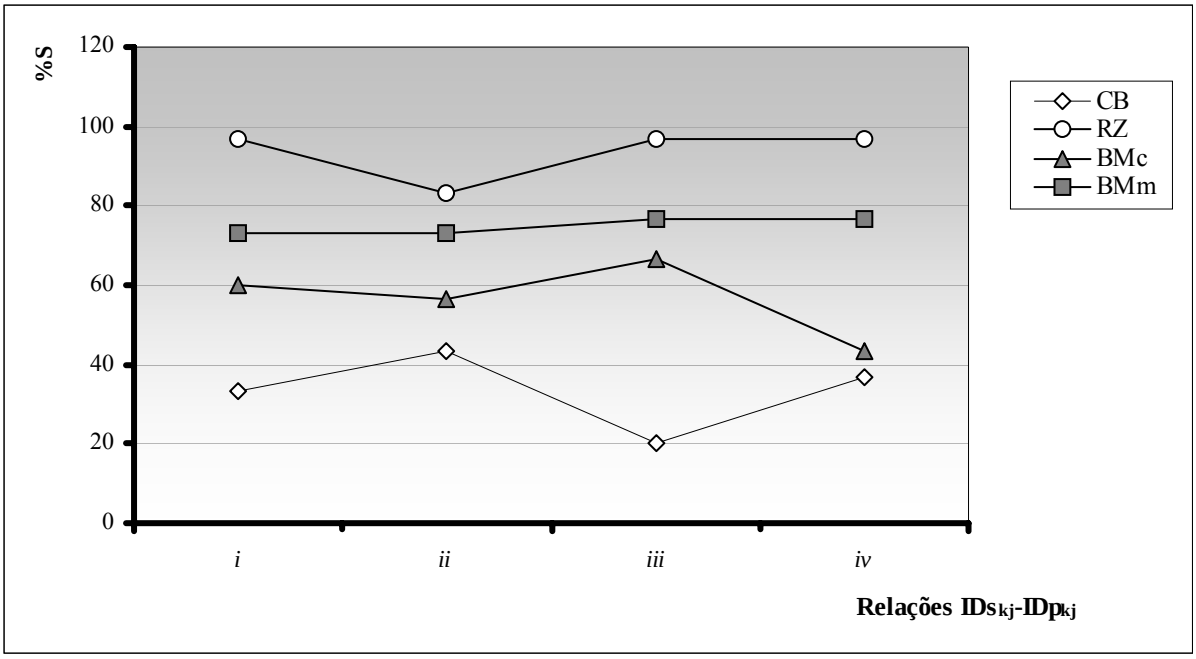

Gráfico 5.25. Porcentagem de sucesso para n = 7 e m = 5

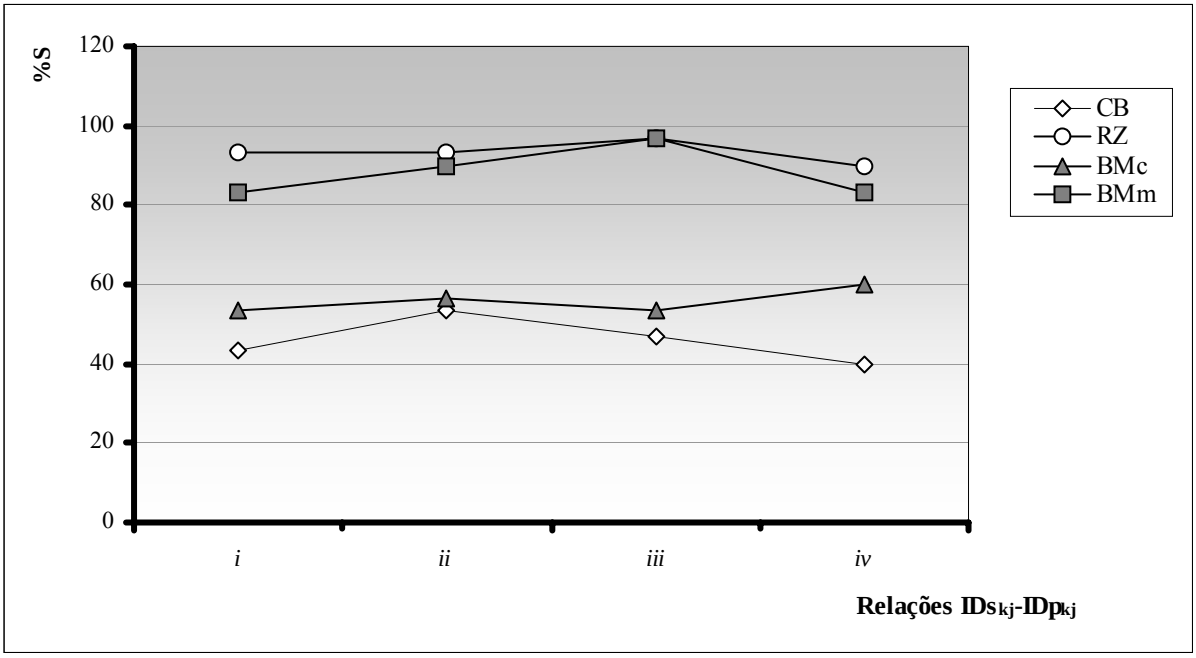

Gráfico 5.26. Porcentagem de sucesso para n = 7 e m = 10

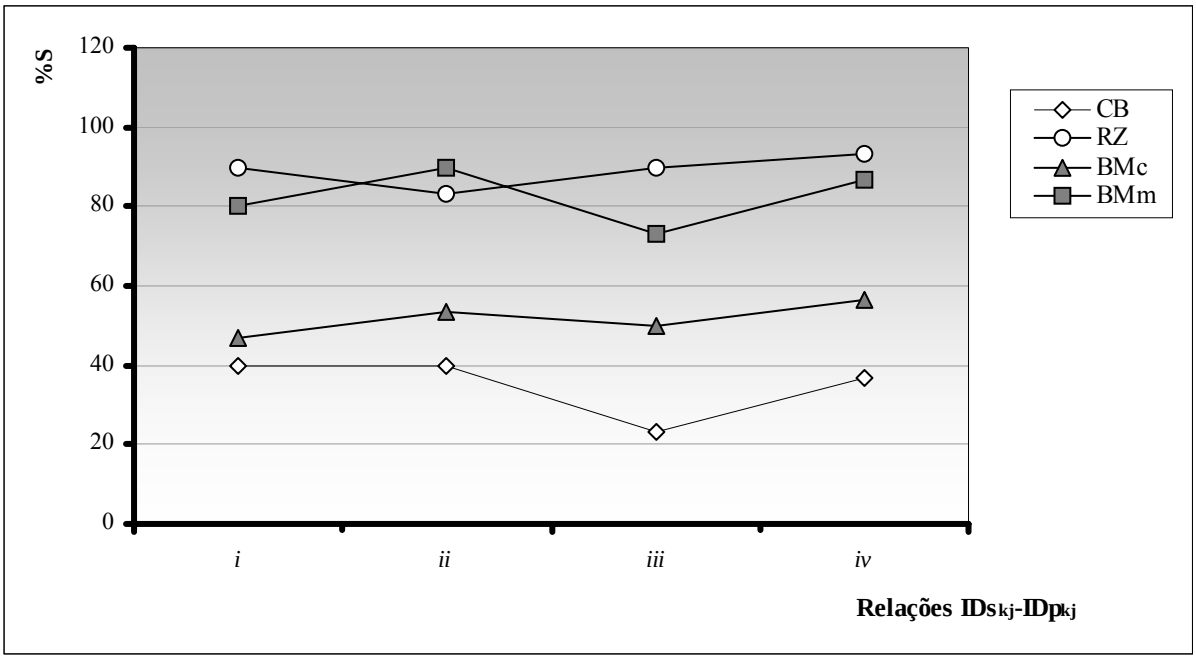

Gráfico 5.27. Porentagem de sucesso para $n = 7$ e $m = 15$

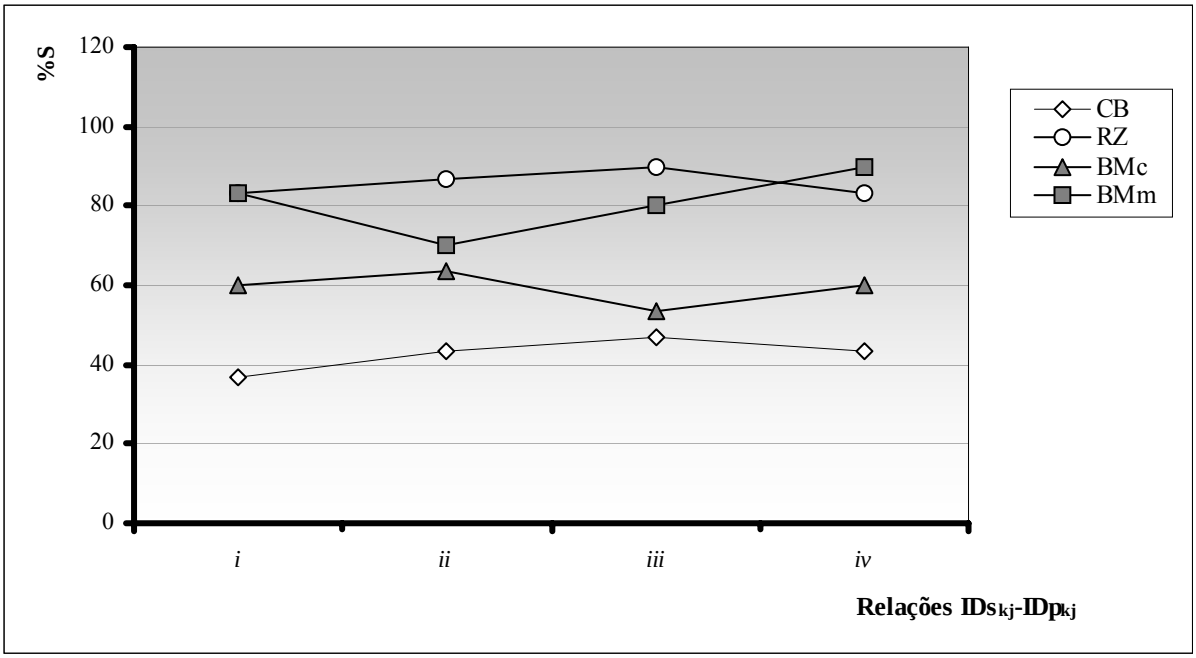

Gráfico 5.28. Porentagem de sucesso para $n = 7$ e $m = 20$

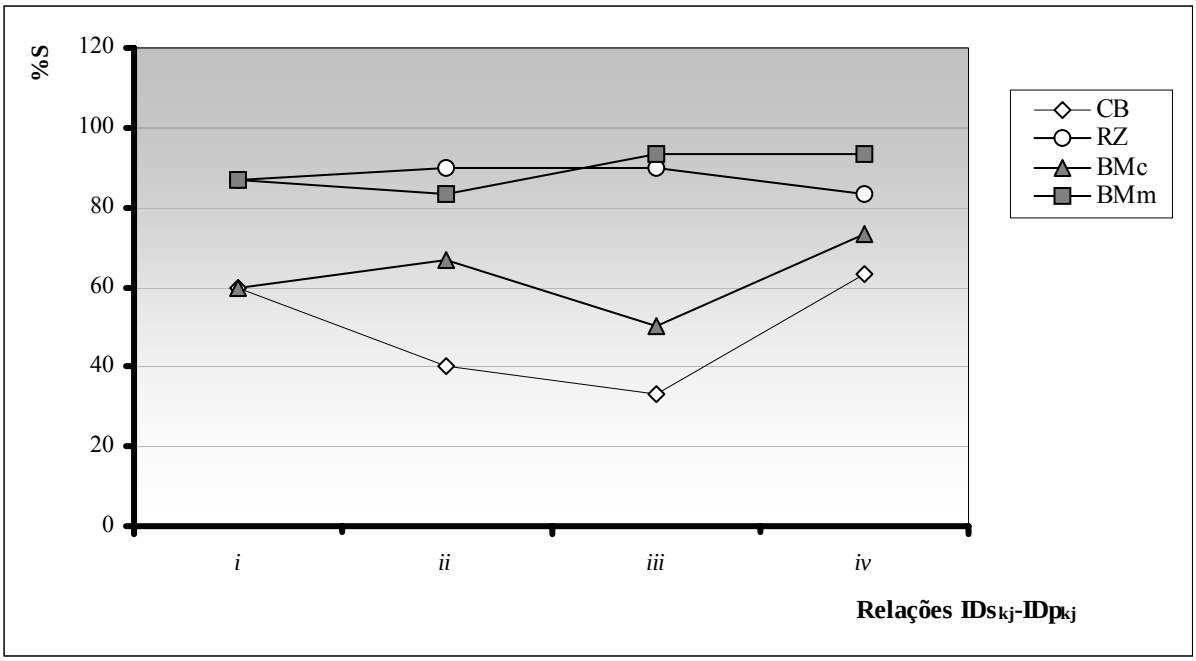

Gráfico 5.29. Porcentagem de sucesso para $n = 7$ e $m = 25$

Analisando-se os Gráficos 5.21 a 5.29 é possível constatar que, no caso de $n = 7$, não houve influência do número de máquinas m e das Relações IDs_{kj}-IDp_{kj} no desempenho de cada método.

Tabela 5.6 - Porcentagem de sucesso, agregando-se as Relações IDs_{kj}-IDp_{kj} - $n = 7$

	MÉTODOS			
m	**CB**	**RZ_3**	**BM_c**	**BM_m**
5	33,33	93,33	56,67	75,00
10	45,00	94,17	55,00	89,17
15	35,00	86,67	52,50	81,67
20	42,50	85,83	59,17	80,83
25	49,17	87,50	62,50	89,17

O Gráfico 5.30 apresenta, para $n = 7$, a comparação da porcentagem de sucesso entre os métodos CB, RZ_3, BM_c e BM_m, agregando-se as Relações IDs_{kj}-IDp_{kj}.

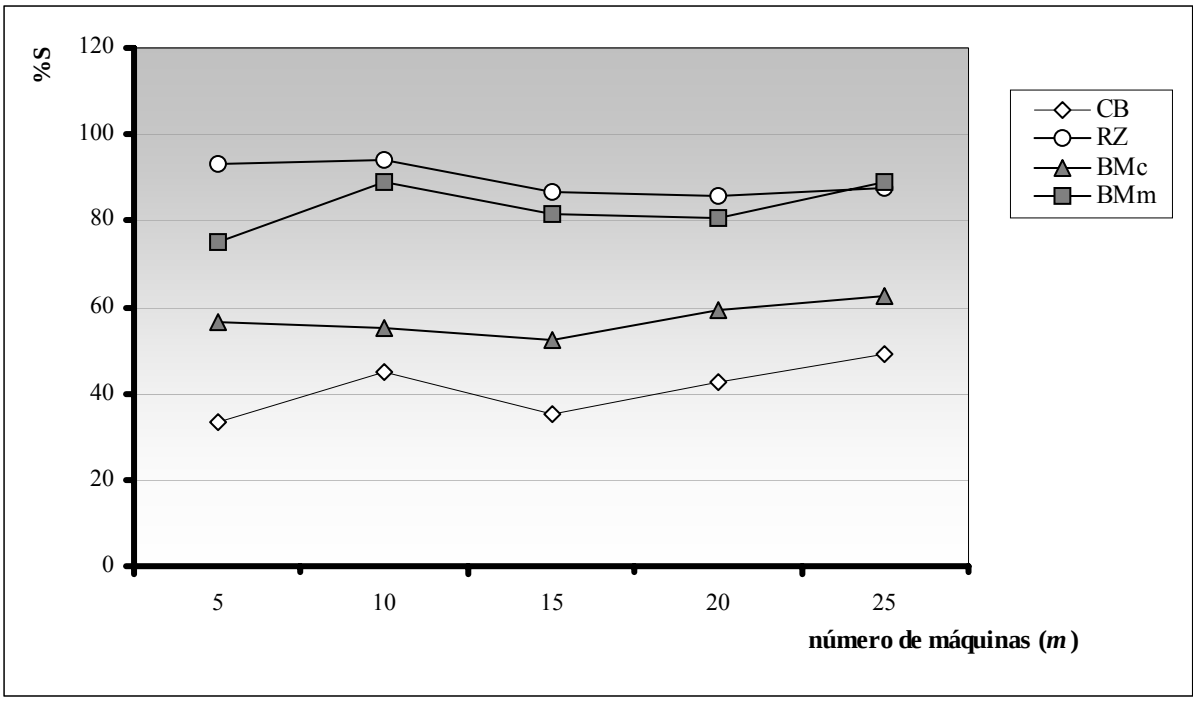

Gráfico 5.30. Porcentagem de sucesso, agregando-se as Relações IDs_{kj}-IDp_{kj} - $n = 7$

É possível concluir, observando o Gráfico 5.30, que, no caso de $n = 7$, a ordem de superioridade no desempenho é a seguinte: RZ_3; BM_m; BM_c; CB. No entanto, a porcentagem de sucesso média do método BM_m é muito próxima da do método RZ_3 (apenas 6,33 pontos percentuais menor, em média). A porcentagem de sucesso média do método BM_c é significativamente superior da do método RZ_3 (16,17 pontos percentuais a mais, em média).

Os Gráficos 5.1 a 5.30 mostram que, no caso de problemas de pequeno porte o método BM_c perde para o método CB apenas com $n = 4$, mas com uma diferença pequena. As análises desses Gráficos indicam ainda que, nesses casos, não houve influência do número de máquinas m e das Relações IDs_{kj}-IDp_{kj} no desempenho dos métodos. Com exceção do método CB que, praticamente em todos os problemas das classes deste grupo, apresentou pior desempenho na Relação *iii*.

Os resultados das porcentagens de sucesso para os problemas de pequeno porte são apresentados na Tabela 5.7.

Tabela 5.7 - Porcentagem de sucesso para os problemas de pequeno porte

	MÉTODOS			
	CB	**RZ_3**	**BM_c**	**BM_m**
Total de problemas	954	1651	1137	1553
%	53,00	91,72	63,17	86,28

O Gráfico 5.31 apresenta uma comparação da porcentagem de sucesso entre os métodos CB, RZ_3, BM_c e BM_m, considerando-se os problemas de pequeno porte.

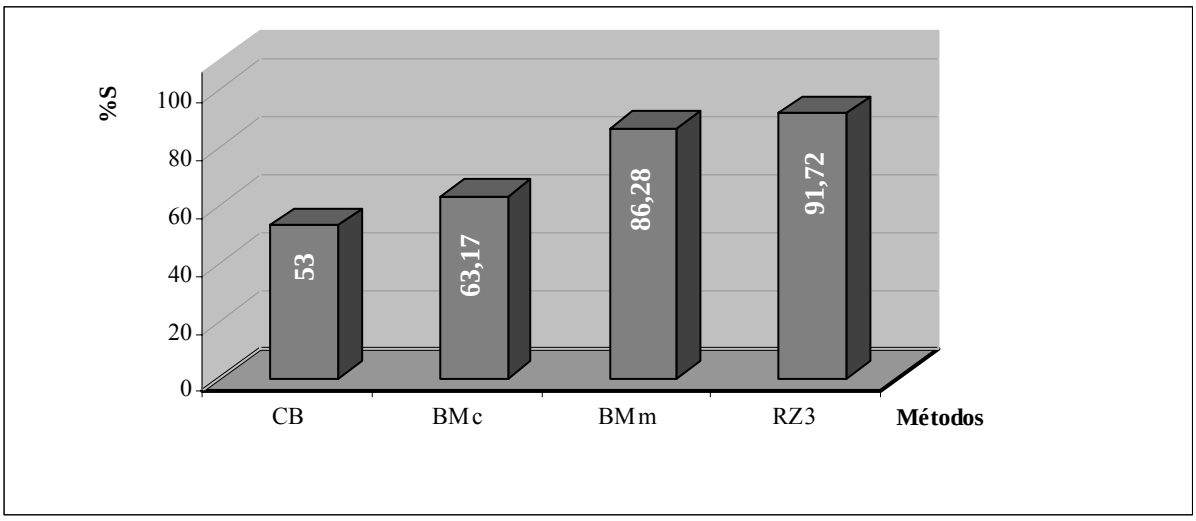

Gráfico 5.31. Porcentagem de sucesso para os problemas de pequeno porte

O Gráfico 5.31 indica que, no caso de problemas de pequeno porte, a ordem de superioridade no desempenho dos métodos é a seguinte: RZ_3; BM_m; BM_c; CB.

As Tabelas 5.8, 5.10 e 5.12 mostram os resultados da porcentagem de sucesso para $n \in \{20, 40, 60\}$ em função das Relações IDs_{kj}-IDp_{kj} e do número de máquinas (m). Nas Tabelas 5.2, 5.4 e 5.6 os resultados da porcentagem de sucesso para $n \in \{20, 40, 60\}$ são apresentados em função do número de máquinas (m), agregando-se as Relações IDs_{kj}-IDp_{kj}.

Tabela 5.8 - Porcentagem de sucesso para $n = 20$

Relação IDs_{kj}-IDp_{kj}	m	**MÉTODOS**			
		CB	**RZ_3**	**BM_c**	**BM_m**
i	5	23,33	60,00	50,00	70,00
	10	26,67	83,33	60,00	90,00
	15	43,33	83,33	53,33	93,33
	20	36,67	90,00	66,67	83,33
	25	33,33	83,33	70,00	86,67
ii	5	26,67	50,00	76,67	93,33
	10	20,00	70,00	40,00	83,33
	15	33,33	73,33	70,00	96,67
	20	23,33	90,00	73,33	86,67
	25	26,67	80,00	66,67	90,00
iii	5	33,33	63,33	66,67	90,00
	10	20,00	70,00	66,67	93,33
	15	30,00	90,00	56,67	90,00
	20	33,33	90,00	70,00	96,67
	25	26,67	83,33	60,00	86,67
iv	5	30,00	53,33	63,33	73,33
	10	30,00	70,00	80,00	83,33
	15	23,33	93,33	43,33	80,00
	20	16,67	86,67	53,33	86,67
	25	23,33	96,67	56,67	73,33

Os Gráficos 5.32 a 5.35 mostram, para $n = 20$, a comparação da porcentagem de sucesso entre os métodos CB, RZ_3, BM_c e BM_m para as quatro Relações IDs_{kj}-IDp_{kj}.

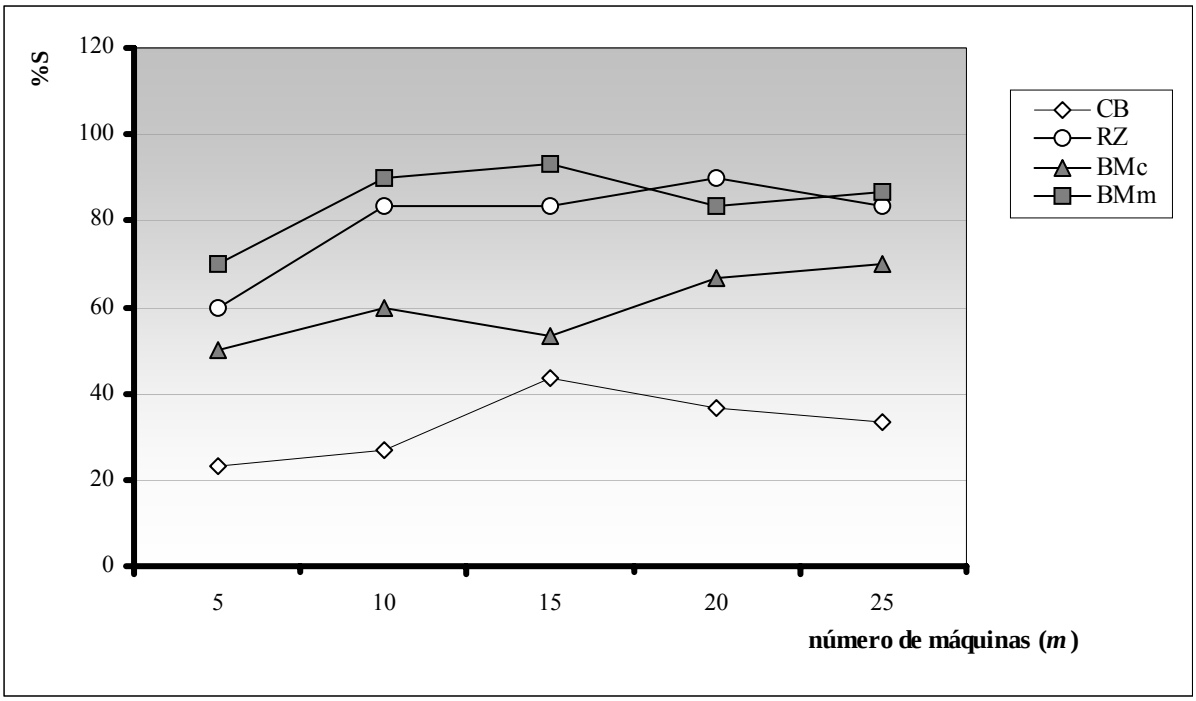

Gráfico 5.32. Porcentagem de sucesso para $n = 20$ – Relação *i*

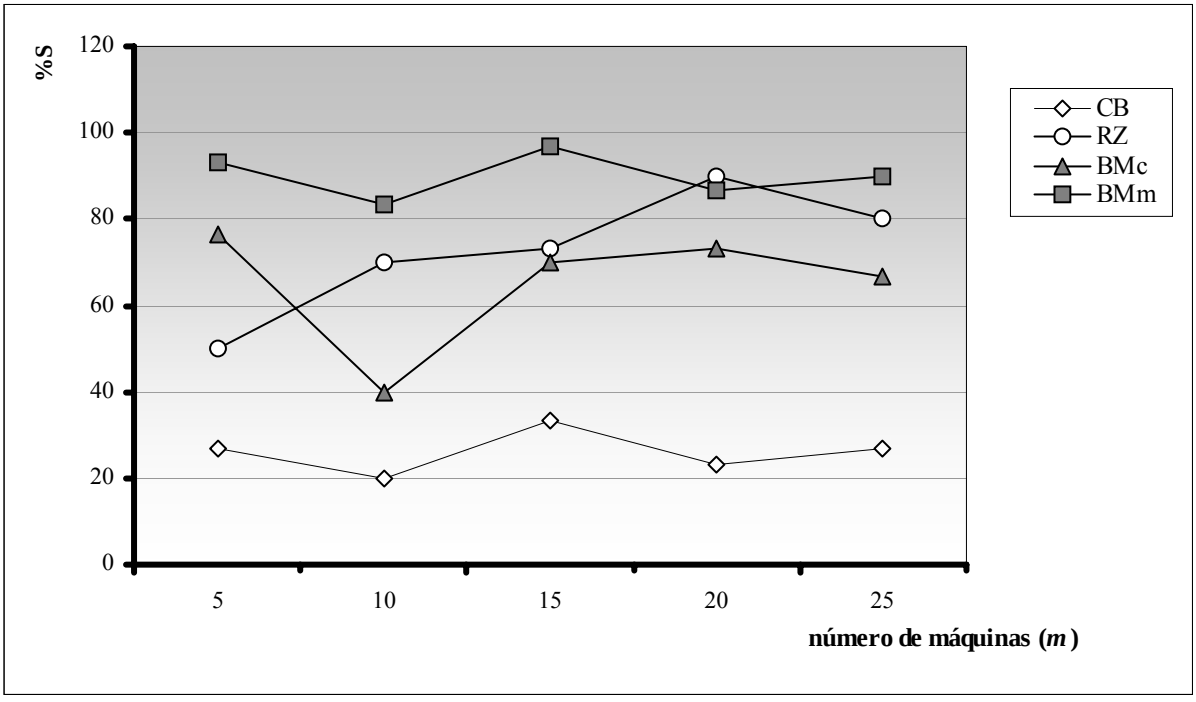

Gráfico 5.33. Porcentagem de sucesso para $n = 20$ – Relação *ii*

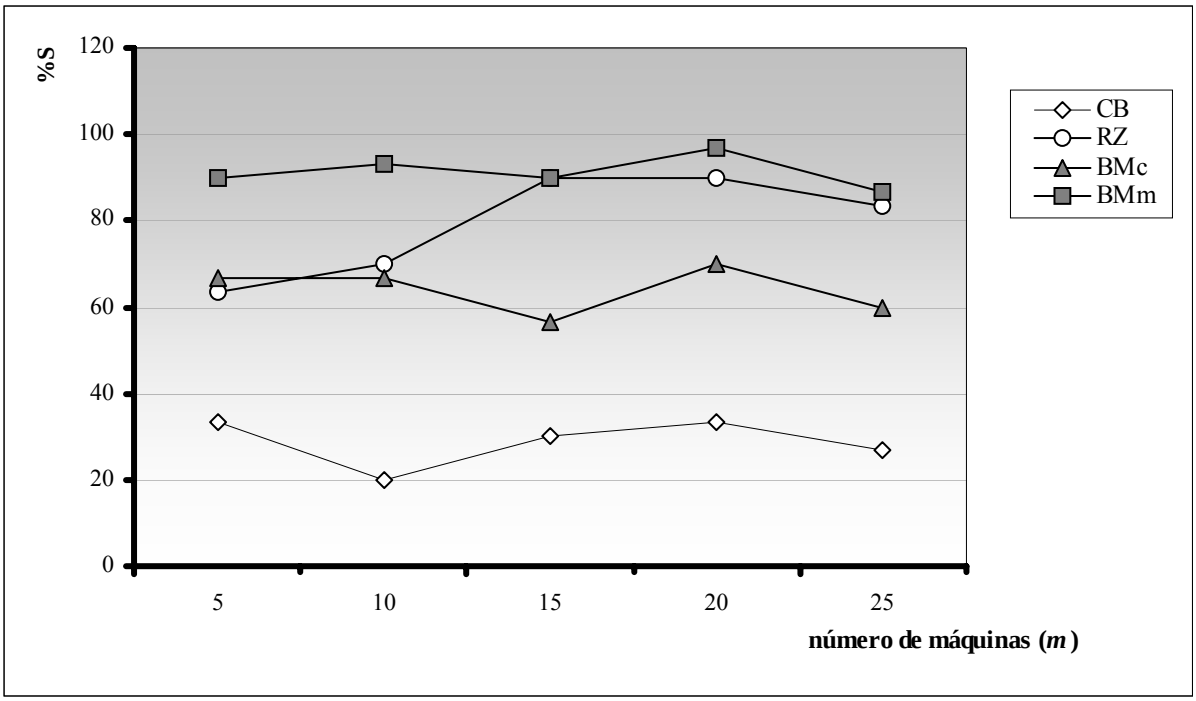

Gráfico 5.34. Porcentagem de sucesso para $n = 20$ – Relação *iii*

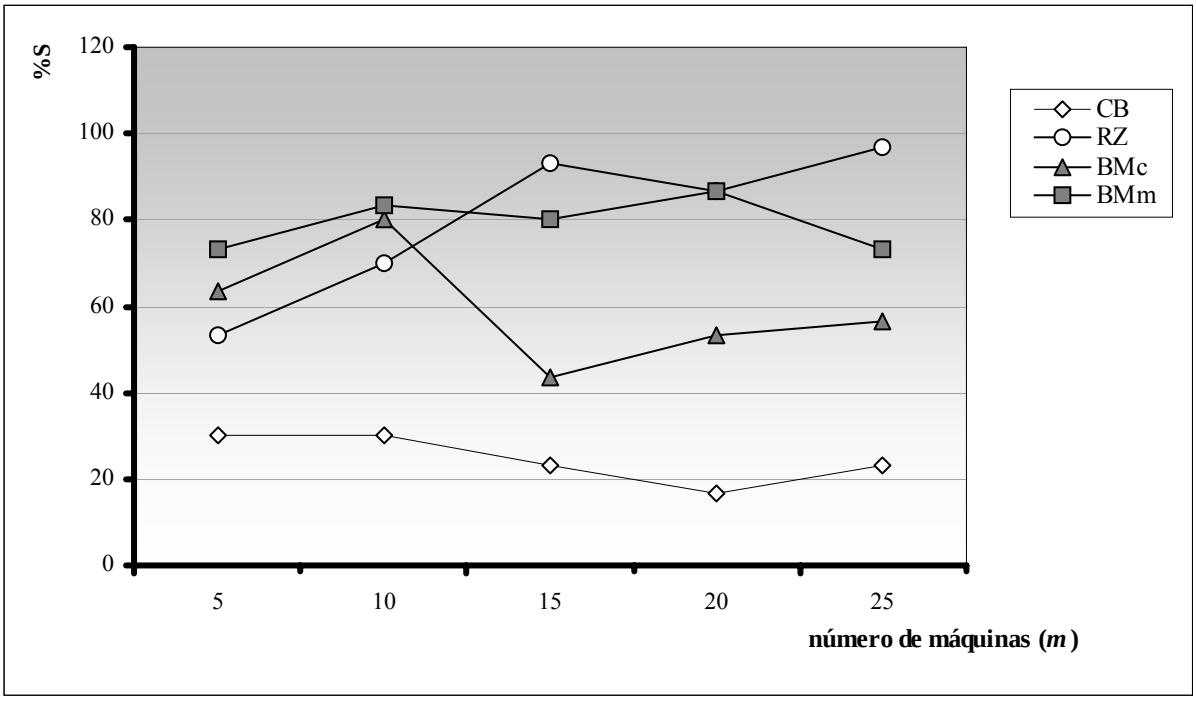

Gráfico 5.35. Porcentagem de sucesso para $n = 20$ – Relação *iv*

Os Gráficos 5.36 a 5.40 mostram, para n = 20, a comparação da porcentagem de sucesso entre os métodos CB, RZ_3, BM_c e BM_m para todos os valores de m.

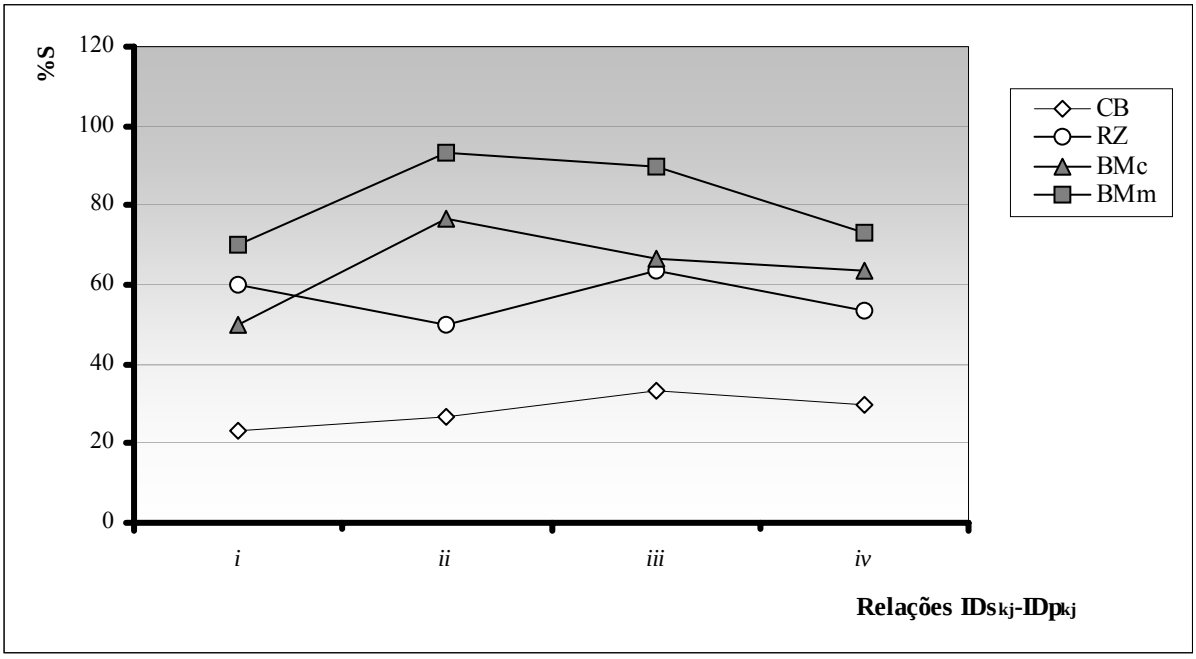

Gráfico 5.36. Porcentagem de sucesso para n = 20 e m = 5

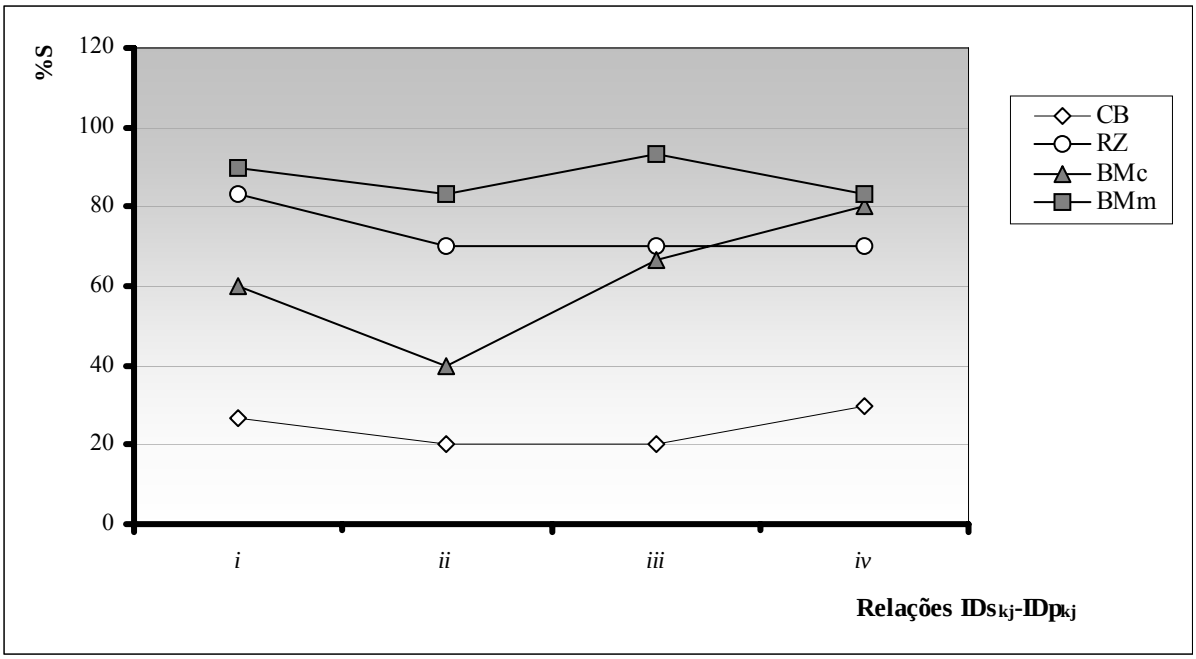

Gráfico 5.37. Porcentagem de sucesso para n = 20 e m = 10

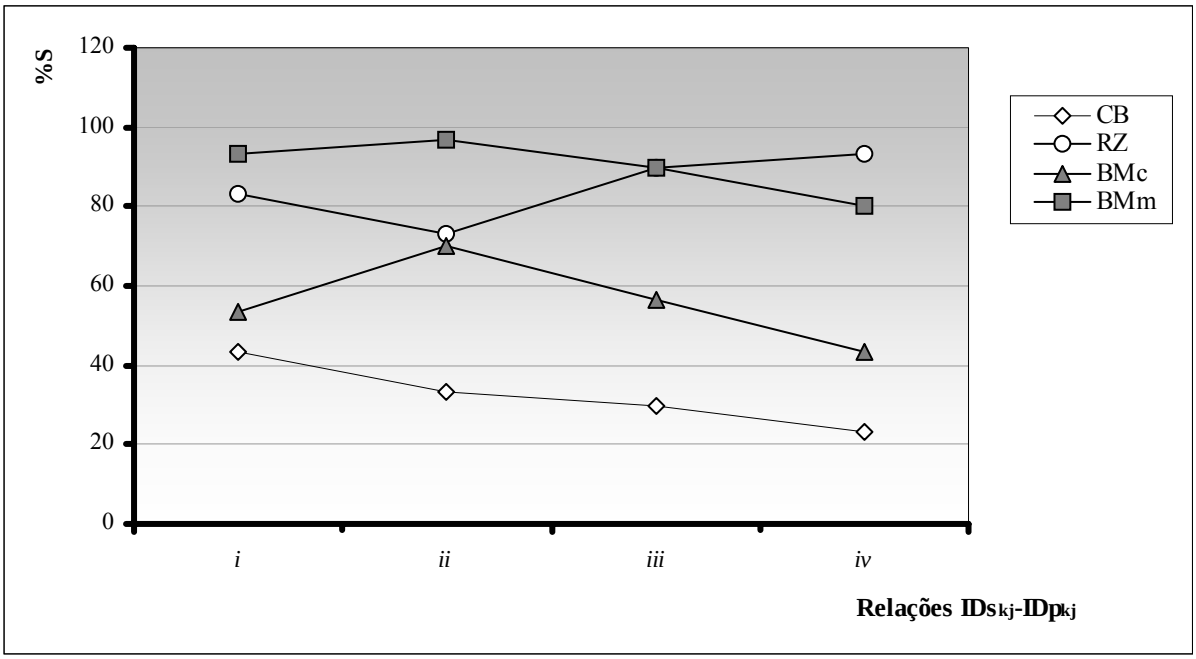

Gráfico 5.38. Porentagem de sucesso para $n = 20$ e $m = 15$

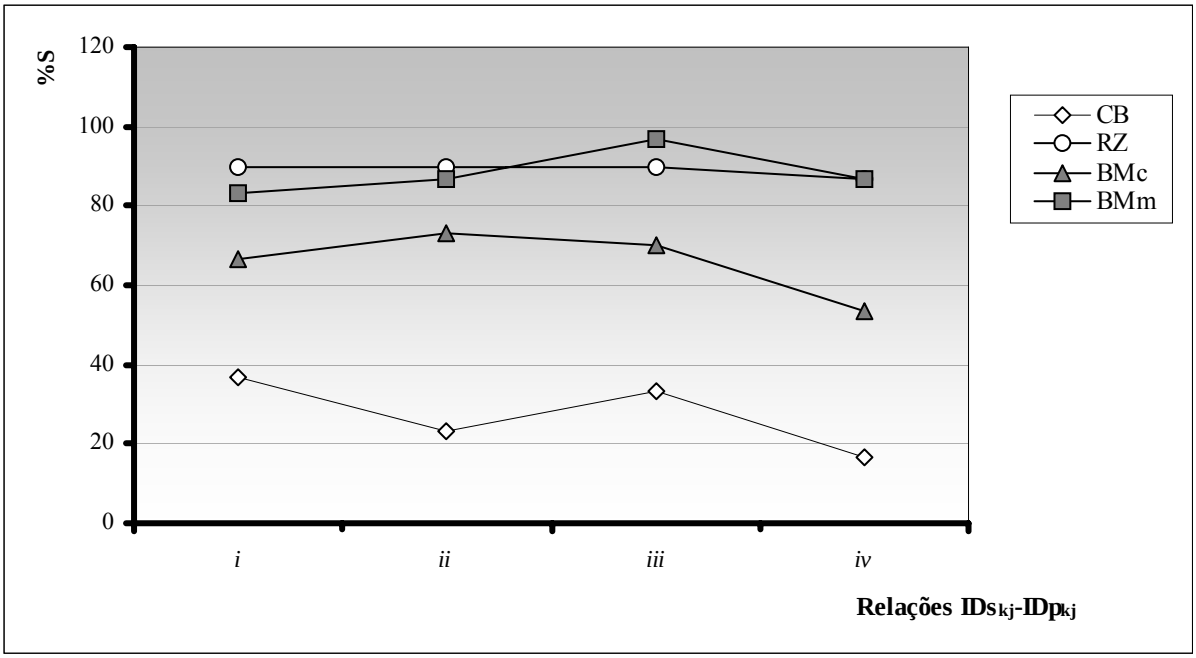

Gráfico 5.39. Porentagem de sucesso para $n = 20$ e $m = 20$

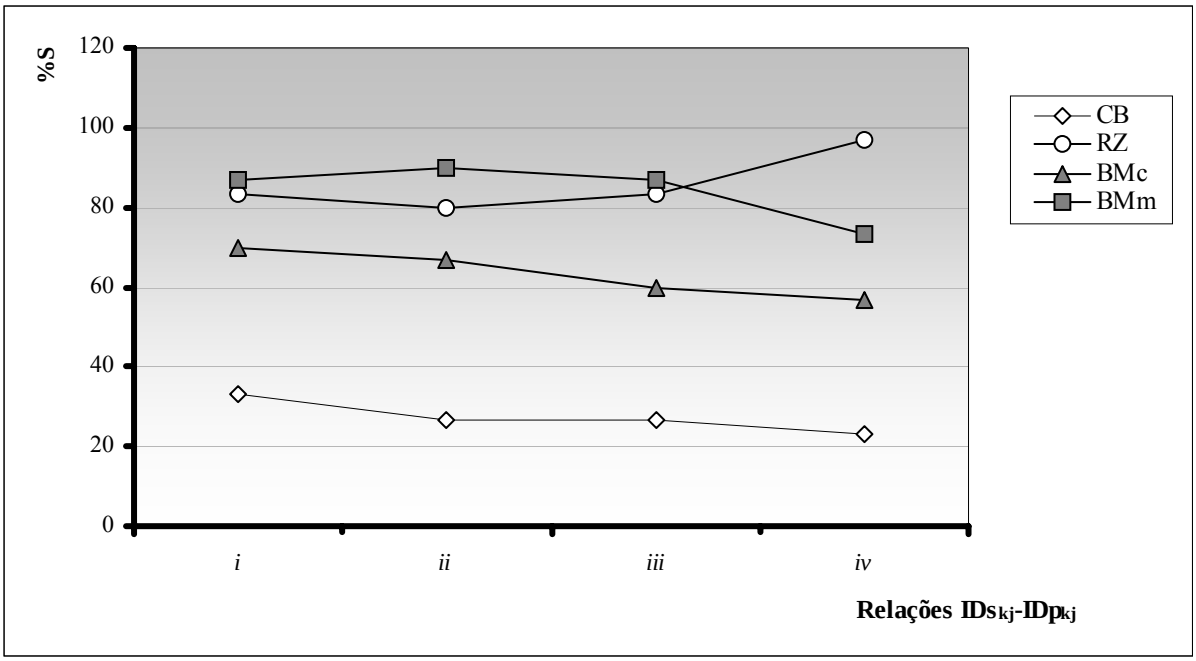

Gráfico 5.40. Porcentagem de sucesso para $n = 20$ e $m = 25$

Analisando-se os Gráficos 5.32 a 5.9 é possível constatar que, no caso de $n = 20$, não ocorreu influência do número de máquinas m e das Relações IDs_{kj}-IDp_{kj} no desempenho de cada método.

Tabela 5.9 - Porcentagem de sucesso, agregando-se as Relações IDs_{kj}-IDp_{kj} - $n = 20$

	MÉTODOS			
m	**CB**	**RZ_3**	**BM_c**	**BM_m**
5	28,33	56,67	64,17	81,67
10	24,17	73,33	61,67	87,50
15	32,50	85,00	55,83	90,00
20	27,50	89,17	65,83	88,33
25	27,50	85,83	63,33	84,17

O Gráfico 5.41 apresenta, para $n = 20$, a comparação da porcentagem de sucesso entre os métodos CB, RZ_3, BM_c e BM_m, agregando-se Relações IDs_{kj}-IDp_{kj}.

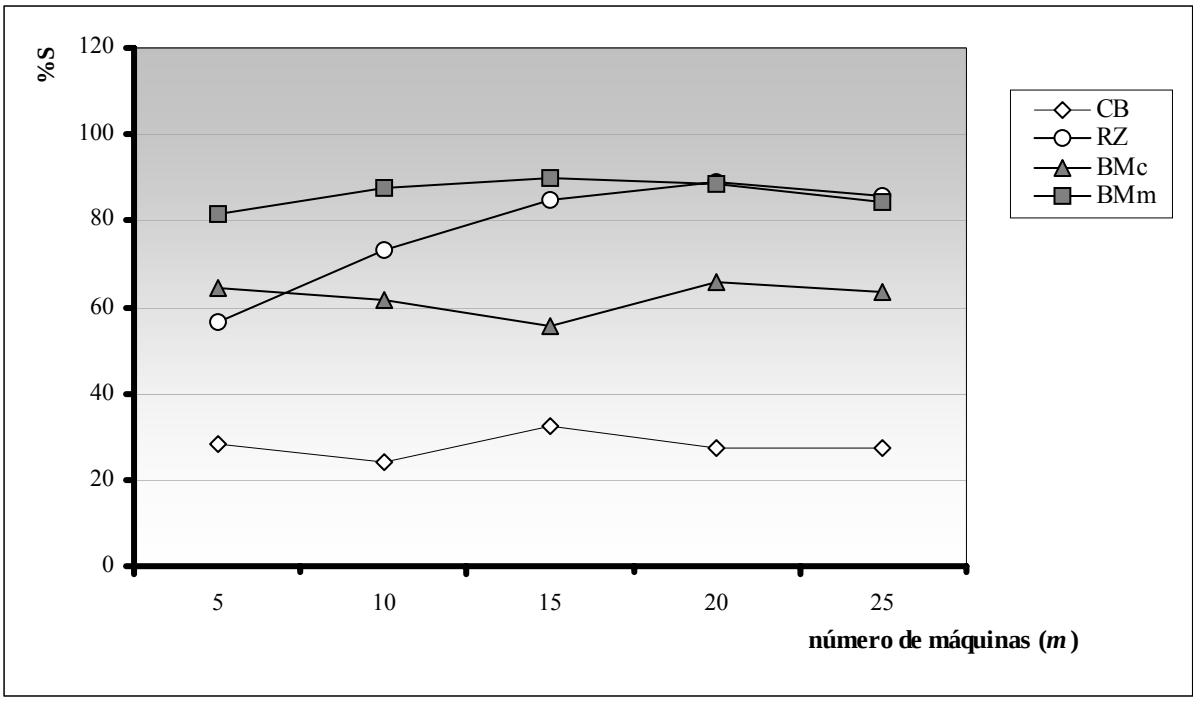

Gráfico 5.41. Porcentagem de sucesso, agregando-se as Relações IDs_{kj}-IDp_{kj} - n = 20

É possível concluir, observando o Gráfico 5.41, que, no caso de n = 20, a ordem de superioridade no desempenho é a seguinte: BM_m; RZ_3; BM_c; CB. A porcentagem de sucesso média do método BM_c é significativamente superior da do método RZ_3 (8,33 pontos percentuais a mais, em média).

Tabela 5.10 - Porcentagem de sucesso para $n = 40$

Relação IDs_{kj}-IDp_{kj}	m	MÉTODOS			
		CB	**RZ_3**	**BM_c**	**BM_m**
i	5	0	33,33	83,33	100,00
	10	0	50,00	80,00	96,67
	15	0	63,33	66,67	86,67
	20	0	66,67	66,67	90,00
	25	0	83,33	70,00	96,67
ii	5	0	46,67	56,67	90,00
	10	0	46,67	66,67	93,33
	15	0	70,00	80,00	90,00
	20	0	66,67	63,33	93,33
	25	0	93,33	76,67	100,00
iii	5	0	66,67	70,00	86,67
	10	0	80,00	80,00	86,67
	15	0	63,33	76,67	93,33
	20	0	90,00	76,67	90,00
	25	0	90,00	63,33	90,00
		0			
iv	5	0	66,67	83,33	90,00
	10	0	46,67	83,33	96,67
	15	0	66,67	73,33	83,33
	20	0	76,67	50,00	86,67
	25	0	86,67	60,00	93,33

Os Gráficos 5.42 a 5.45 mostram, para $n = 40$, a comparação da porcentagem de sucesso entre os métodos RZ_3, BM_c e BM_m para as quatro Relações IDs_{kj}-IDp_{kj}.

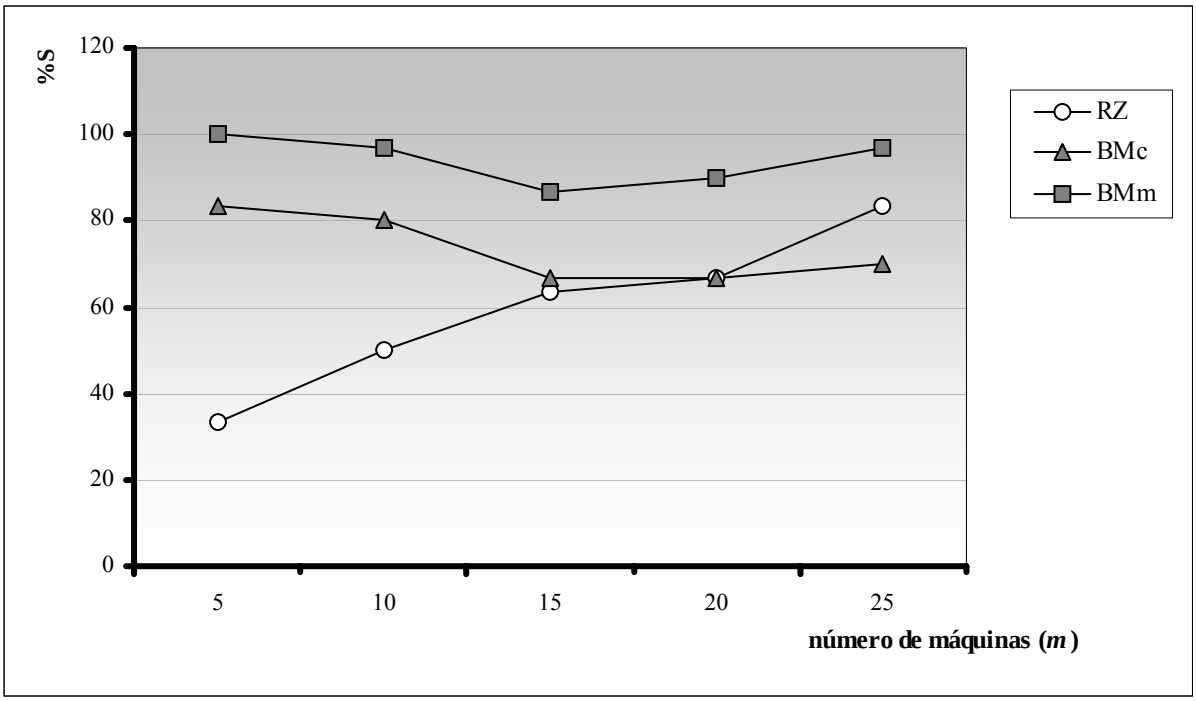

Gráfico 5.42. Porcentagem de sucesso para $n = 40$ – Relação *i*

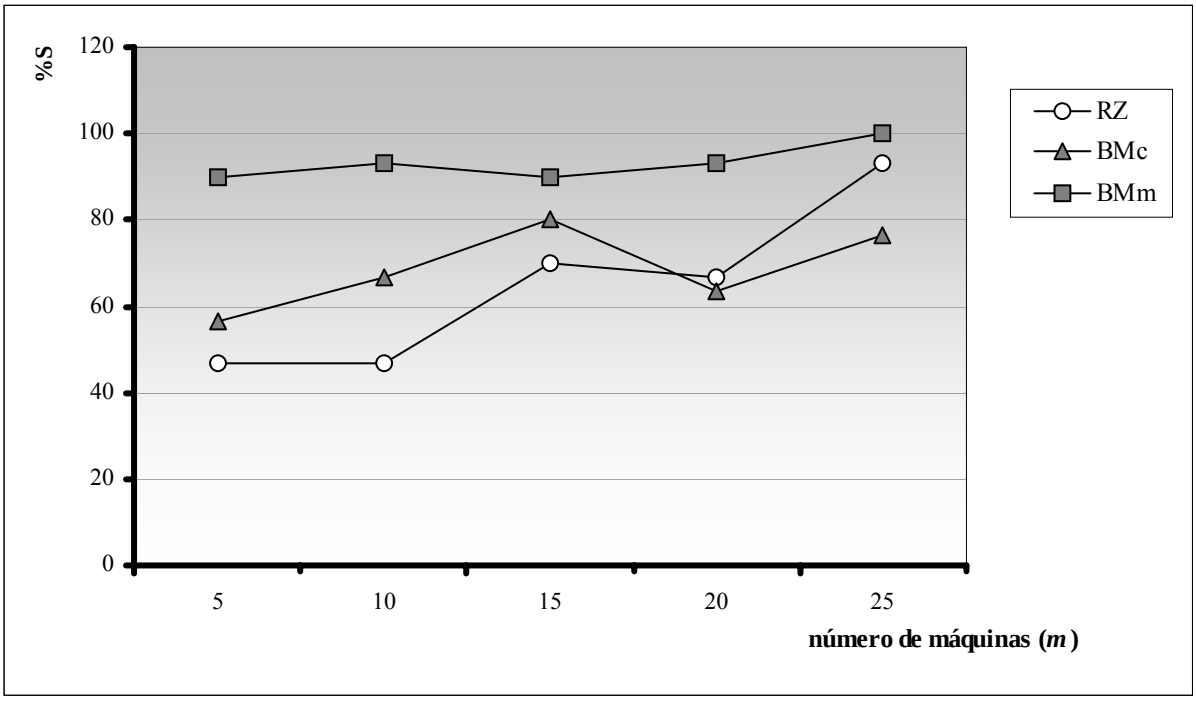

Gráfico 5.43. Porcentagem de sucesso para $n = 40$ – Relação *ii*

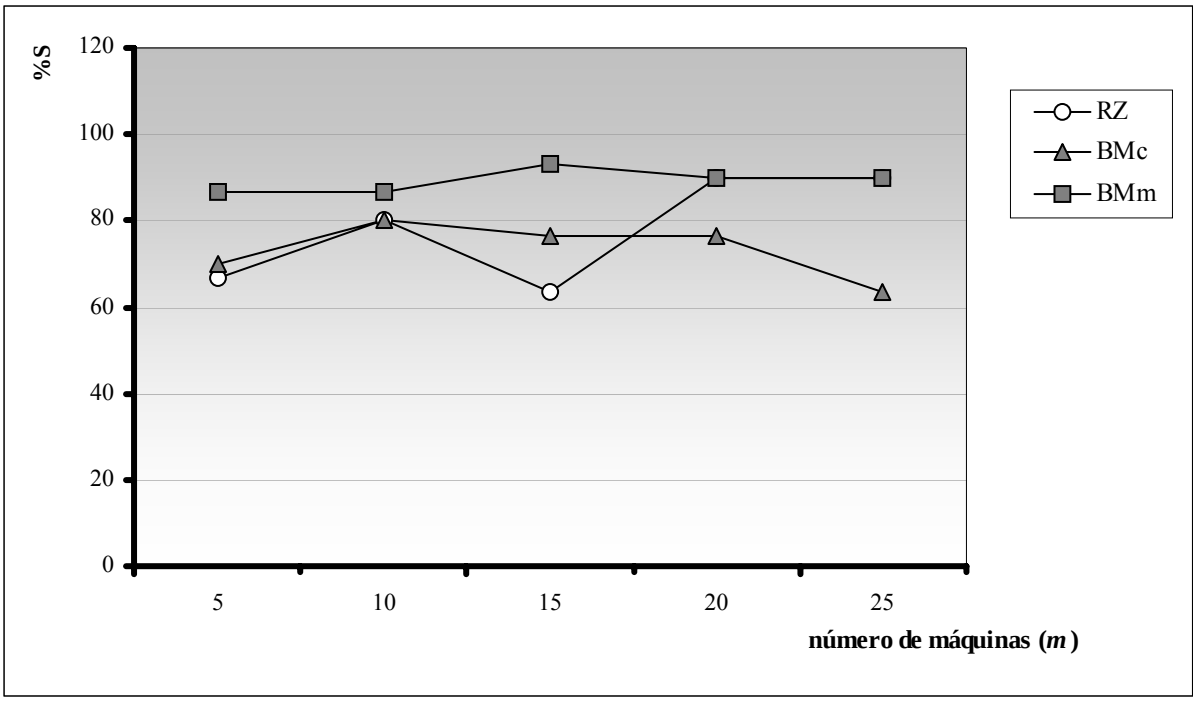

Gráfico 5.44. Porcentagem de sucesso para $n = 40$ – Relação *iii*

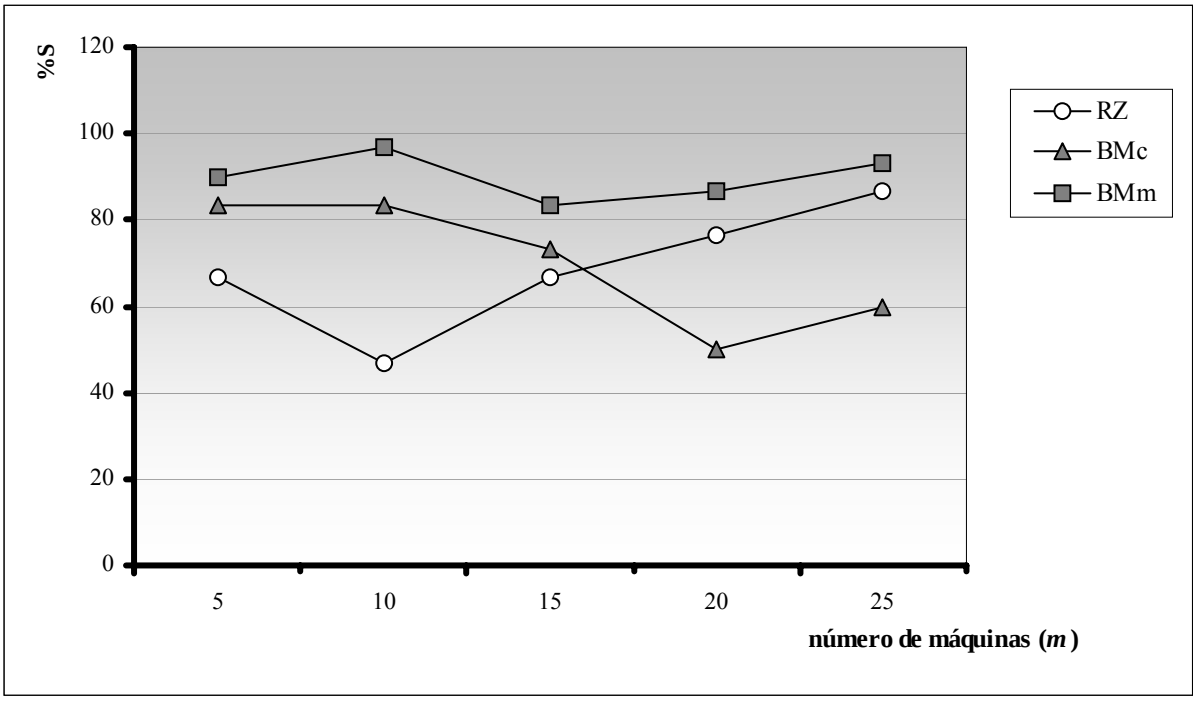

Gráfico 5.45. Porcentagem de sucesso para $n = 40$ – Relação *iv*

Os Gráficos 5.46 a 5.50 mostram, para $n = 20$, a comparação da porcentagem de sucesso entre os métodos RZ_3, BM_c e BM_m para todos os valores de m.

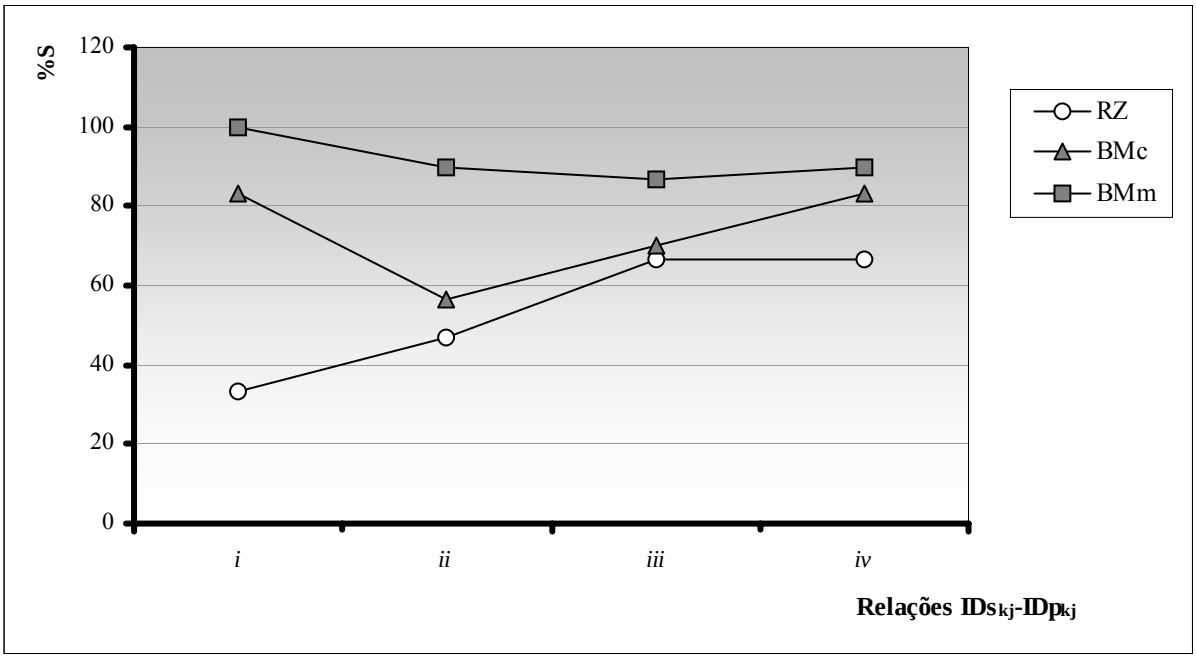

Gráfico 5.46. Porcentagem de sucesso para $n = 40$ e $m = 5$

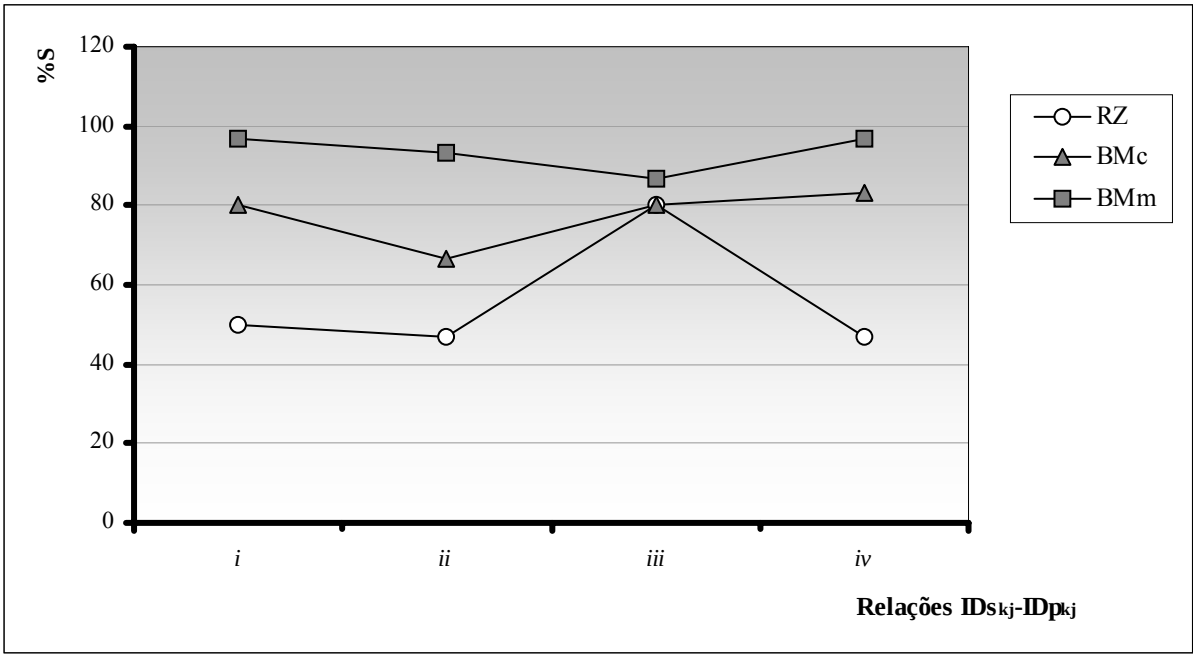

Gráfico 5.47. Porcentagem de sucesso para $n = 40$ e $m = 10$

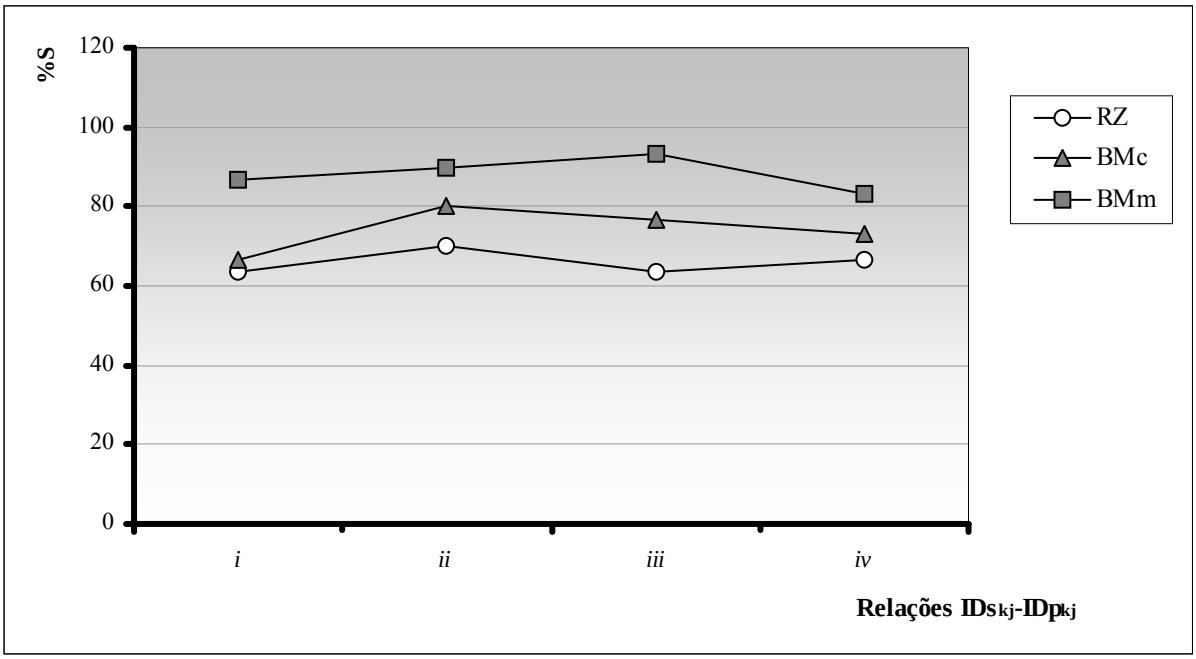

Gráfico 5.48. Porcentagem de sucesso para $n = 40$ e $m = 15$

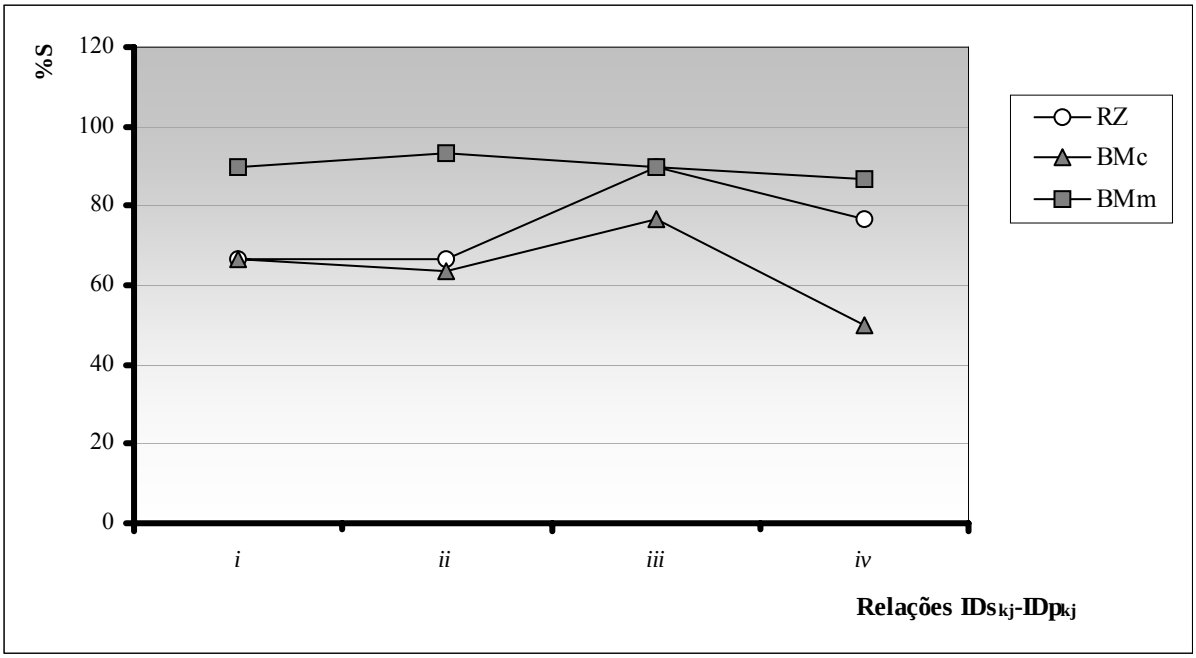

Gráfico 5.49. Porcentagem de sucesso para $n = 40$ e $m = 20$

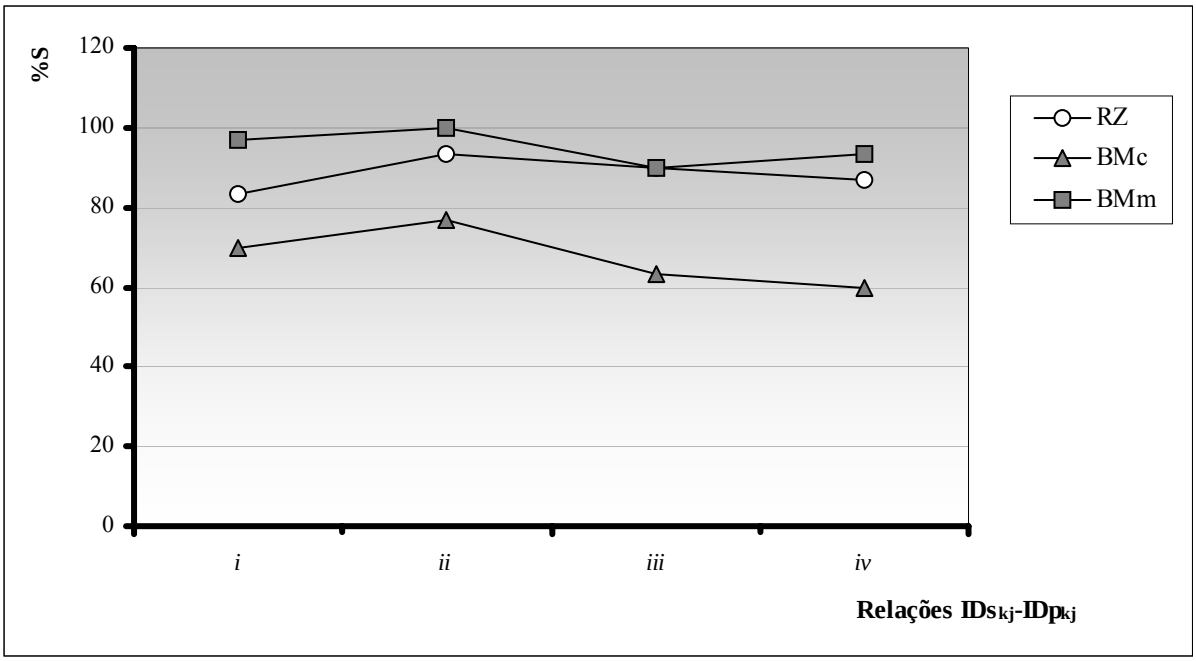

Gráfico 5.50. Porcentagem de sucesso para $n = 40$ e $m = 25$

A análise dos Gráficos 5.42 a 5.50 indica que, no caso de $n = 40$, não ocorreu influência do número de máquinas m e das Relações IDs_{kj}-IDp_{kj} no desempenho de cada método.

Tabela 5.11 - Porcentagem de sucesso, agregando-se as Relações IDs_{kj}-IDp_{kj} - $n = 40$

	MÉTODOS			
m	**CB**	**RZ_3**	**BM_c**	**BM_m**
5	0	53,33	73,33	91,67
10	0	55,83	77,50	93,33
15	0	65,83	74,17	88,33
20	0	75,00	64,17	90,00
25	0	88,33	67,50	95,00

O Gráfico 5.51 apresenta, para $n = 40$, a comparação da porcentagem de sucesso entre os métodos RZ_3, BM_c e BM_m, agregando-se as Relações IDs_{kj}-IDp_{kj}.

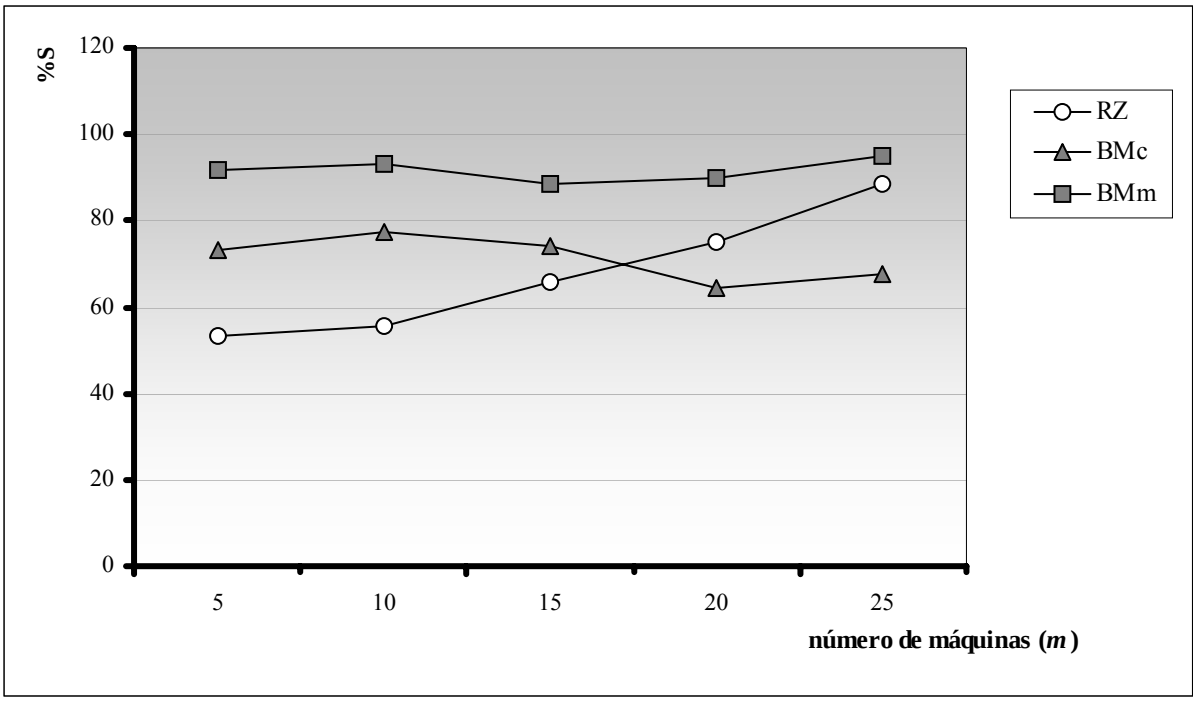

Gráfico 5.51. Porcentagem de sucesso, agregando-se as Relações IDs_{kj}-IDp_{kj} - $n = 40$

É possível concluir, observando o Gráfico 5.51, que, no caso de $n = 40$, a ordem de superioridade no desempenho é a seguinte: BM_m; BM_c; RZ_3 (a porcentagem de sucesso do método BM_m é 24, 00 pontos percentuais, em média, maior que do método RZ_3).

Tabela 5.12 - Porcentagem de sucesso para $n = 60$

Relação IDs_{kj}-IDp_{kj}	m	**MÉTODOS**			
		CB	**RZ_3**	**BM_c**	**BM_m**
i	5	0	56,67	76,67	90,00
	10	0	46,67	86,67	100,00
	15	0	43,33	86,67	96,67
	20	0	66,67	53,33	73,33
	25	0	76,67	80,00	90,00
ii	5	0	50,00	83,33	90,00
	10	0	46,67	80,00	100,00
	15	0	63,33	66,67	80,00
	20	0	63,33	73,33	93,33
	25	0	70,00	53,33	90,00
iii	5	0	50,00	80,00	96,67
	10	0	50,00	63,33	86,67
	15	0	53,33	76,67	96,67
	20	0	60,00	50,00	80,00
	25	0	76,67	73,33	93,33
iv	5	0	56,67	90,00	100,00
	10	0	53,33	86,67	93,33
	15	0	66,67	73,33	86,67
	20	0	53,33	60,00	90,00
	25	0	70,00	63,33	83,33

Os Gráficos 5.52 a 5.55 mostram, para $n = 60$, a comparação da porcentagem de sucesso entre os métodos RZ_3, BM_c e BM_m para as quatro Relações IDs_{kj}-IDp_{kj}.

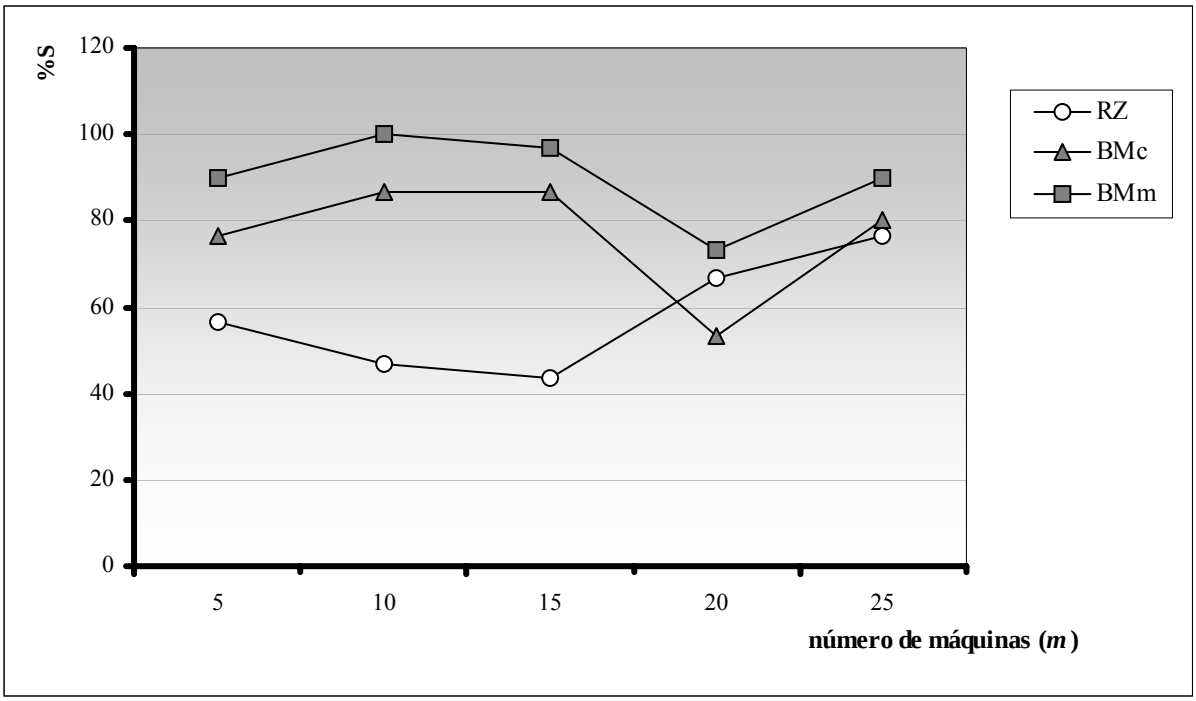

Gráfico 5.22. Porcentagem de sucesso para $n = 60$ – Relação *i*

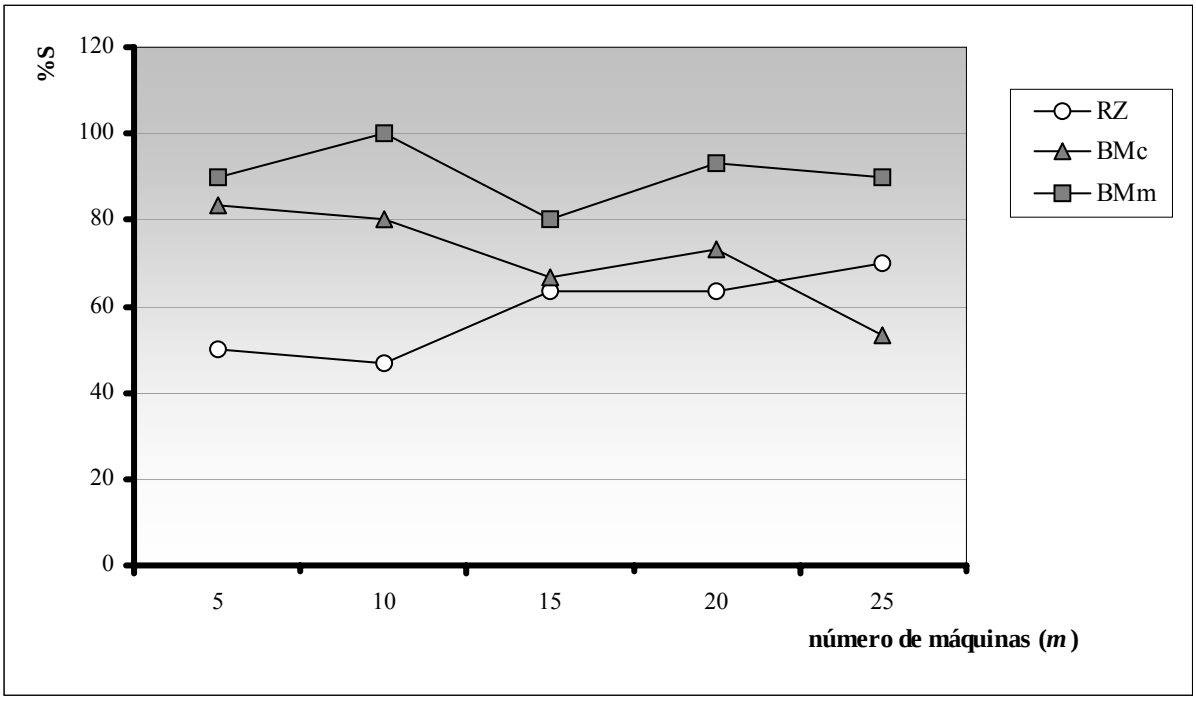

Gráfico 5.53. Porcentagem de sucesso para $n = 60$ – Relação *ii*

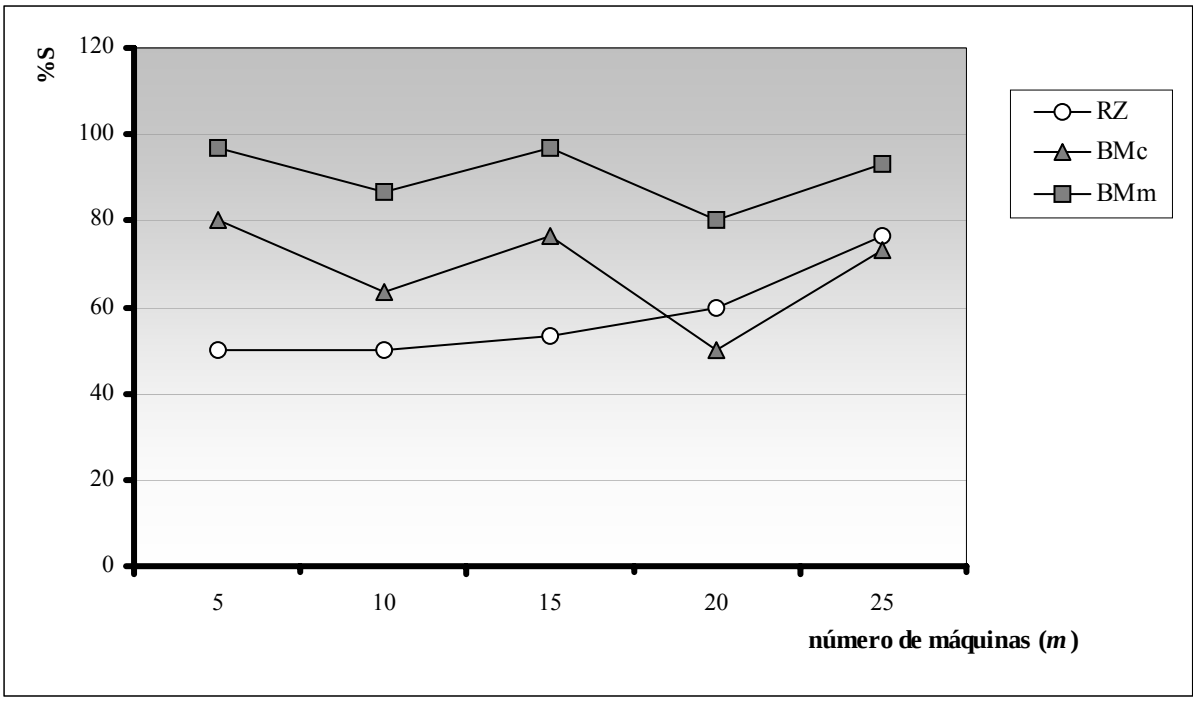

Gráfico 5.54. Porcentagem de sucesso para $n = 60$ – Relação *iii*

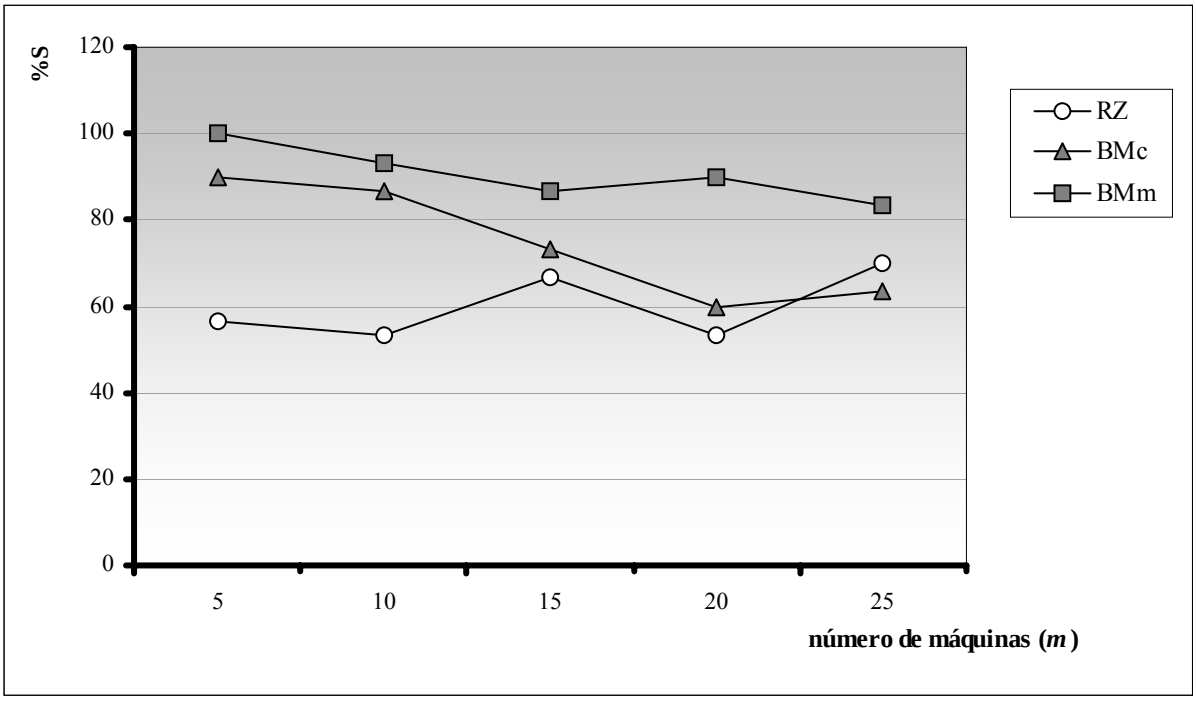

Gráfico 5.55. Porcentagem de sucesso para $n = 60$ – Relação *iv*

Os Gráficos 5.56 a 5.60 mostram, para n = 60, a comparação da porcentagem de sucesso entre os métodos RZ_3, BM_c e BM_m para todos os valores de m.

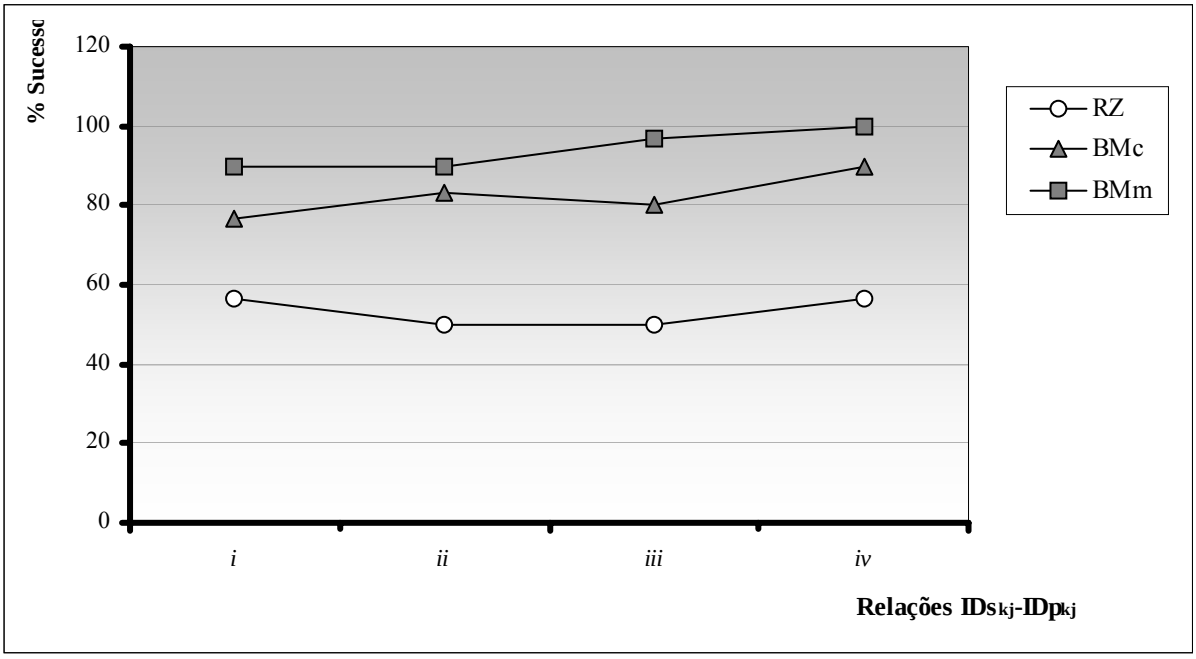

Gráfico 5.56. Porcentagem de sucesso para n = 60 e m = 5

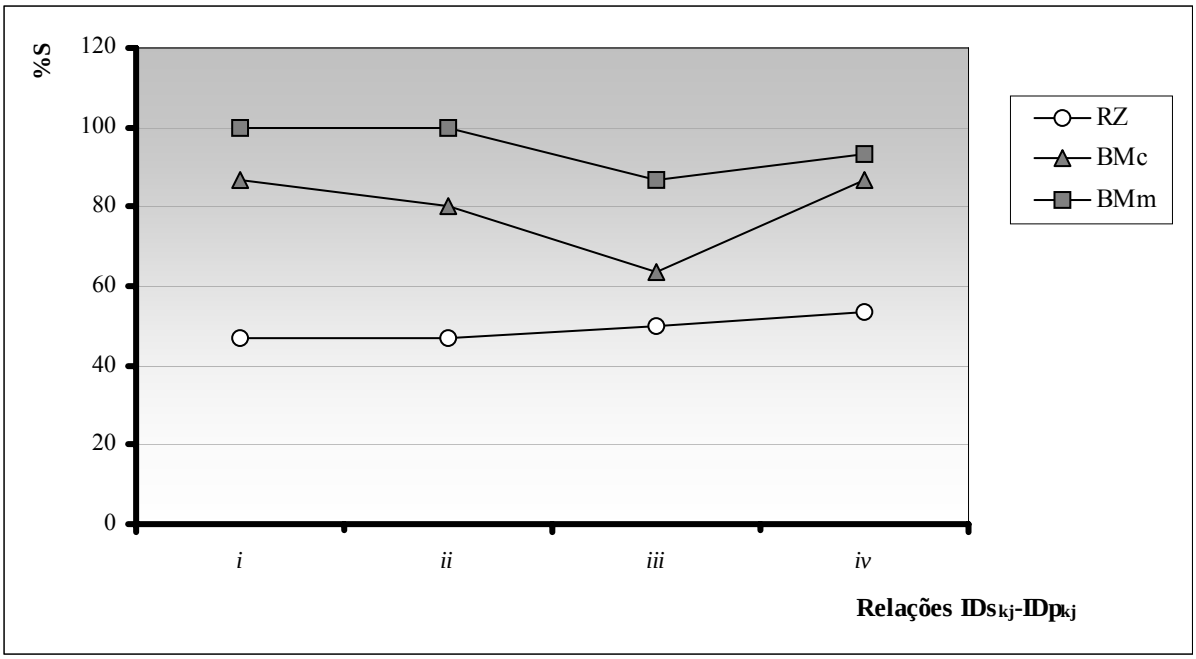

Gráfico 5.57. Porcentagem de sucesso para n = 60 e m = 10

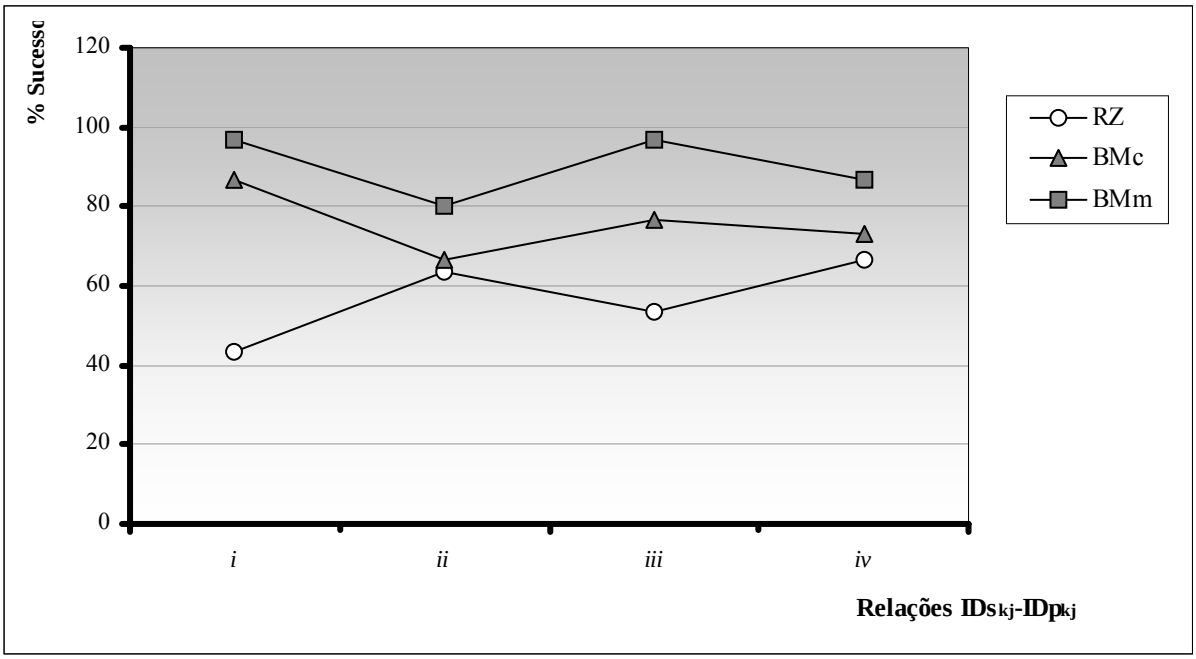

Gráfico 5.58. Porentagem de sucesso para $n = 60$ e $m = 15$

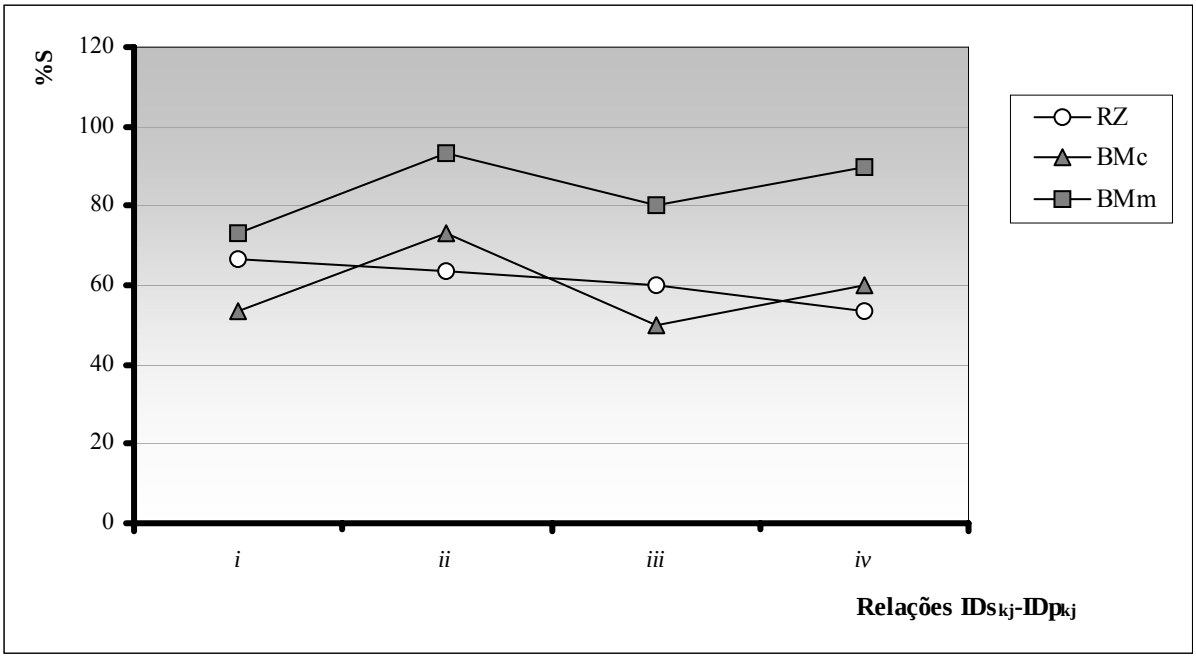

Gráfico 5.59. Porentagem de sucesso para $n = 60$ e $m = 20$

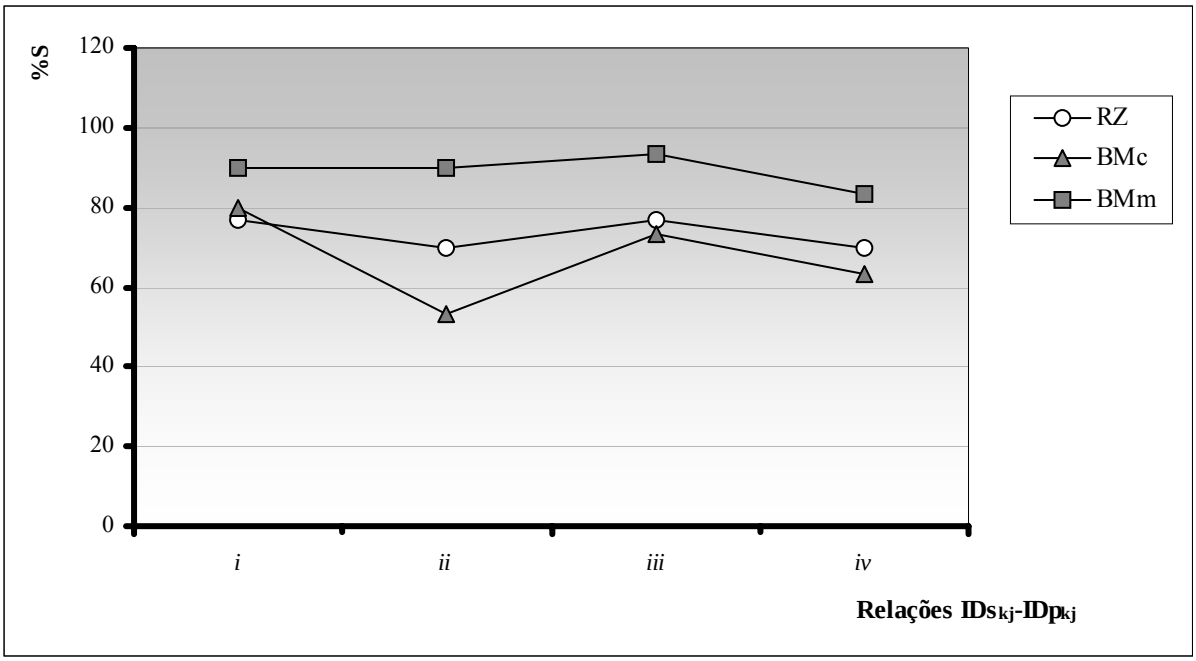

Gráfico 5.60. Porcentagem de sucesso para $n = 60$ e $m = 25$

A análise dos Gráficos 5.52 a 5.60 indica que, no caso de $n = 60$, não ocorreu influência nem do número de máquinas m nem das Relações IDs_{kj}-IDp_{kj} no desempenho de cada método.

Tabela 5.13 - Porcentagem de sucesso, agregando-se as Relações IDs_{kj}-IDp_{kj} - $n = 60$

	MÉTODOS			
m	**CB**	**RZ_3**	**BM_c**	**BM_m**
5	0	53,33	82,50	94,17
10	0	49,17	79,17	95,00
15	0	56,67	75,83	90,00
20	0	60,83	59,17	84,17
25	0	73,33	67,50	89,17

O Gráfico 5.61 apresenta, para $n = 60$, a comparação da porcentagem de sucesso entre os métodos RZ_3, BM_c e BM_m, agregando-se as Relações IDs_{kj}-IDp_{kj}.

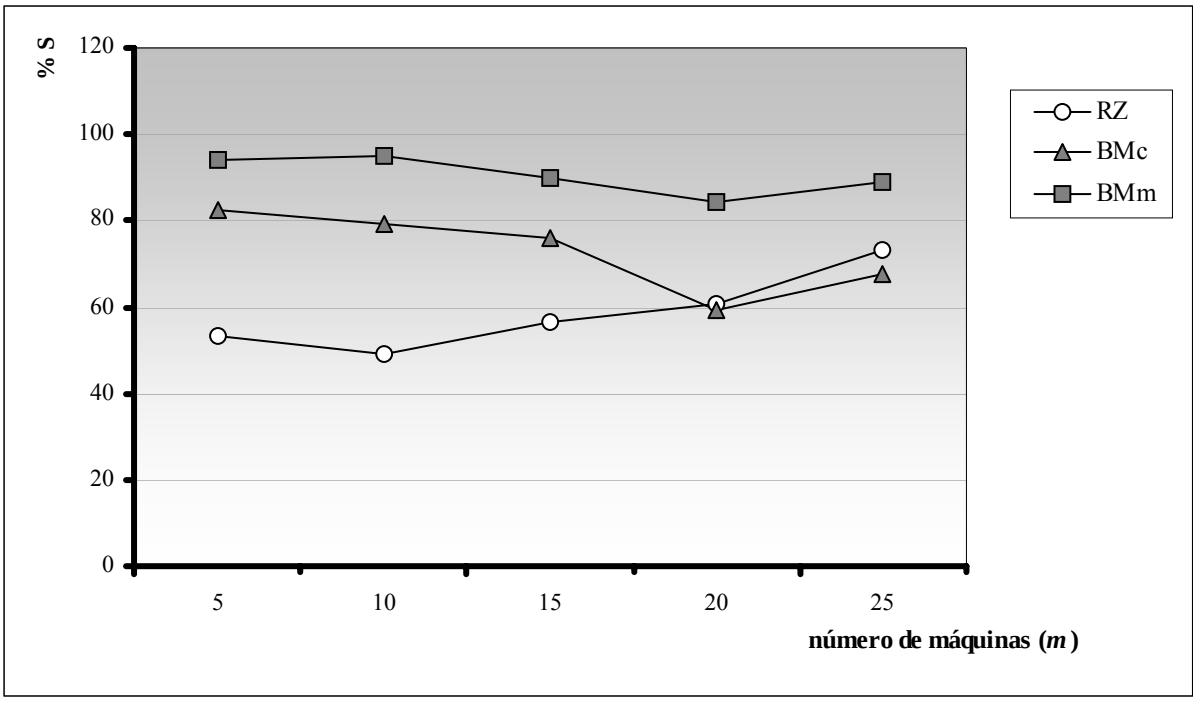

Gráfico 5.61. Porcentagem de sucesso, agregando-se as Relações IDs_{kj}-IDp_{kj} - n = 60

É possível concluir, observando o Gráfico 5.61, que, no caso de n = 60, a ordem de superioridade no desempenho é a seguinte: BM_m; BM_c; RZ_3. Sendo que a porcentagem de sucesso do método BM_m é muito superior a do método RZ_3 (em média, 32 pontos percentuais maior).

Os Gráficos 5.32 a 5.61 mostram que, no caso de problemas de grande porte, o método BM_c perde para o método RZ_3 apenas com n = 20. As análises desses Gráficos indicam ainda que, no caso desse grupo de problemas, não houve influência do número de máquinas m e das Relações IDs_{kj}-IDp_{kj} no desempenho dos métodos.

Os resultados das porcentagens de sucesso para os problemas de grande porte são apresentados na Tabela 5.14.

Tabela 5.14 - Porcentagem de sucesso para os problemas de grande porte

	MÉTODOS			
	CB	**RZ_3**	**BM_c**	**BM_m**
Total de problemas	168	1158	1238	1611
%	9,33	64,33	68,78	89,50

O Gráfico 5.62 apresenta uma comparação da porcentagem de sucesso entre os métodos CB, RZ_3, BM_c e BM_m, considerando-se os problemas de grande porte.

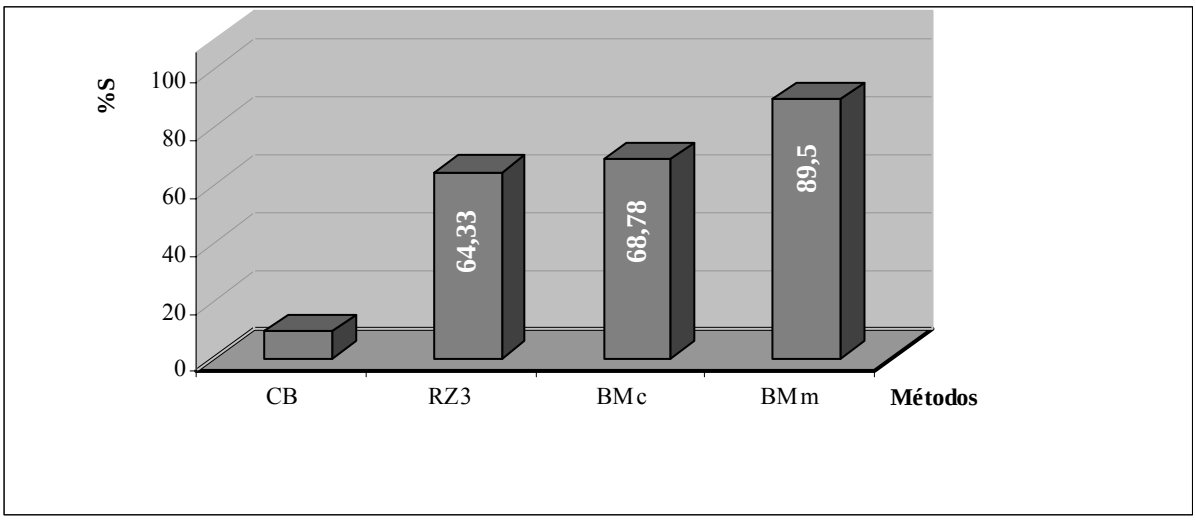

Gráfico 5.62. Porcentagem de sucesso para os problemas de grande porte

O Gráfico 5.62 indica que, no caso de problemas de grande porte, o método construtivo proposto BM_c é muito superior ao método melhorativo CB (mais de 80 pontos percentuais a mais) e ligeiramente bastante superior ao método melhorativo RZ_3 (mais de 25 pontos percentuais a mais). O Gráfico mostra ainda a superioridade do método BM_m em relação a todos os outros métodos.

A Tabela 5.15 apresenta os resultados da porcentagem de sucesso quando todos os problemas são considerados.

Tabela 5.15 - Porcentagem de sucesso para todos os problemas

	MÉTODOS			
	CB	**RZ_3**	**BM_c**	**BM_m**
Total de problemas	1122	2809	2375	3164
%	31,17	78,03	65,97	87,89

O Gráfico 5.63 apresenta uma comparação da porcentagem de sucesso entre os métodos CB, RZ_3, BM_c e BM_m, quando todos os problemas são considerados.

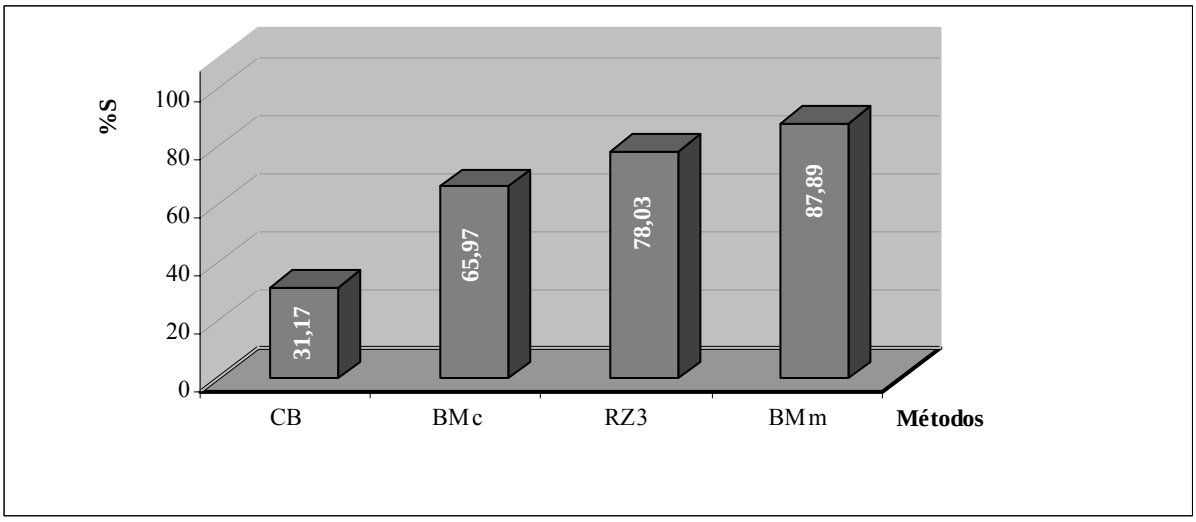

Gráfico 5.63. Porcentagem de sucesso para todos os problemas

A porcentagem global de sucesso, ilustrada no Gráfico 5.63, comprova a superioridade do método proposto BM_m.

Após as análises dos Gráficos referentes à porcentagem de sucesso é possível perceber que, com o aumento do número de tarefas (n), os métodos BM_c e BM_m melhoram muito o seu desempenho, ocorrendo o contrário com os métodos CB e RZ_3.

Com o aumento do número de tarefas (n) a variabilidade da porcentagem de sucesso dos métodos CB e RZ_3 aumenta. Por outro lado, a variabilidade da porcentagem de sucesso dos métodos BM_c e BM_m não são influenciadas pelo aumento do número de tarefas (n), o que mostra a estabilidade desses métodos.

De maneira geral, as curvas de desempenho dos métodos mantiveram o mesmo comportamento com o aumento do número de máquinas *m* e nas Relações IDs_{kj}-IDp_{kj}, o que indica que os desempenhos dos métodos em relação à porcentagem de sucesso não são influenciados por esses.

A Tabela 5.16 apresenta uma comparação geral da performance dos métodos em todas as classes *n*, *m* e em todas as Relações IDs_{kj}-IDp_{kj}, com base na porcentagem de sucesso.

Tabela 5.16 – Resumo do desempenho dos métodos em relação à porcentagem de sucesso

		n					
		Problemas de Pequeno Porte			Problemas de Grande Porte		
Relação IDs_{kj}-IDp_{kj}	*m*	4	6	7	20	40	60
i	5	RZ_3	**BM_m**/ RZ_3	RZ_3	**BM_m**	**BM_m**	**BM_m**
	10	RZ_3	RZ_3	RZ_3	**BM_m**	**BM_m**	**BM_m**
	15	RZ_3	RZ_3	RZ_3	**BM_m**	**BM_m**	**BM_m**
	20	RZ_3	RZ_3	**BM_m**/ RZ_3	RZ_3	**BM_m**	**BM_m**
	25	RZ_3	RZ_3	**BM_m**/ RZ_3	**BM_m**	**BM_m**	**BM_m**
ii	5	RZ_3	RZ_3	RZ_3	**BM_m**	**BM_m**	**BM_m**
	10	**BM_m**	**BM_m**/ RZ_3	RZ_3	**BM_m**	**BM_m**	**BM_m**
	15	RZ_3	RZ_3	**BM_m**	**BM_m**	**BM_m**	**BM_m**
	20	RZ_3	RZ_3	RZ_3	RZ_3	**BM_m**	**BM_m**
	25	**BM_m**/ RZ_3	RZ_3	RZ_3	**BM_m**	**BM_m**	**BM_m**
iii	5	**BM_m**/ RZ_3	RZ_3	RZ_3	**BM_m**	**BM_m**	**BM_m**
	10	RZ_3	RZ_3	**BM_m**/ RZ_3	**BM_m**	**BM_m**	**BM_m**
	15	RZ_3	RZ_3	RZ_3	**BM_m**/ RZ_3	**BM_m**	**BM_m**
	20	RZ_3	**BM_m**	RZ_3	**BM_m**	**BM_m**/ RZ_3	**BM_m**
	25	RZ_3	RZ_3	**BM_m**	**BM_m**	**BM_m**/ RZ_3	**BM_m**
iv	5	**BM_m**	RZ_3	RZ_3	**BM_m**	**BM_m**	**BM_m**
	10	RZ_3	RZ_3	RZ_3	**BM_m**	**BM_m**	**BM_m**
	15	RZ_3	**BM_m**	RZ_3	RZ_3	**BM_m**	**BM_m**
	20	**BM_m**/ RZ_3	**BM_m**	**BM_m**	**BM_m**/ RZ_3	**BM_m**	**BM_m**
	25	RZ_3	RZ_3	**BM_m**	RZ_3	**BM_m**	**BM_m**

Dos 120 casos apresentados na Tabela 5.16 o método BM_m é o melhor em 61 casos, o método RZ_3 em 47, ocorrendo 12 empates. Além disso, a Tabela mostra claramente que, para problemas de grande porte, o método BM_m é o melhor método em todas as Relações IDs_{kj}-IDp_{kj}. A análise da Tabela indica ainda que não existe superioridade de nenhum dos dois métodos no que diz respeito às relações IDs_{kj}-IDp_{kj}.

5.2.2 Desvio relativo médio (%)

As Tabelas 5.17 a 5.22 mostram os resultados do desvio relativo médio para $n \in \{4, 6, 7, 20, 40, 60\}$ em função do número de máquinas (m), agregando-se as Relações IDs_{kj}-IDp_{kj}.

Os Gráficos 5.64 a 5.69 apresentam, para $n \in \{4, 6, 7, 20, 40, 60\}$ a comparação do desvio relativo médio entre os métodos CB, RZ_3, BM_c e BM_m, agregando-se as Relações IDs_{kj}-IDp_{kj}.

Tabela 5.17 – Desvio relativo médio (%), agregando-se as Relações IDs_{kj}-IDp_{kj} - $n = 4$

	MÉTODOS			
m	**CB**	**RZ_3**	**BM_c**	**BM_m**
5	2,57	0,93	3,67	1,90
10	2,40	0,65	3,94	1,99
15	2,66	1,59	3,91	1,63
20	2,21	1,58	3,45	2,56
25	2,23	1,02	2,44	1,39

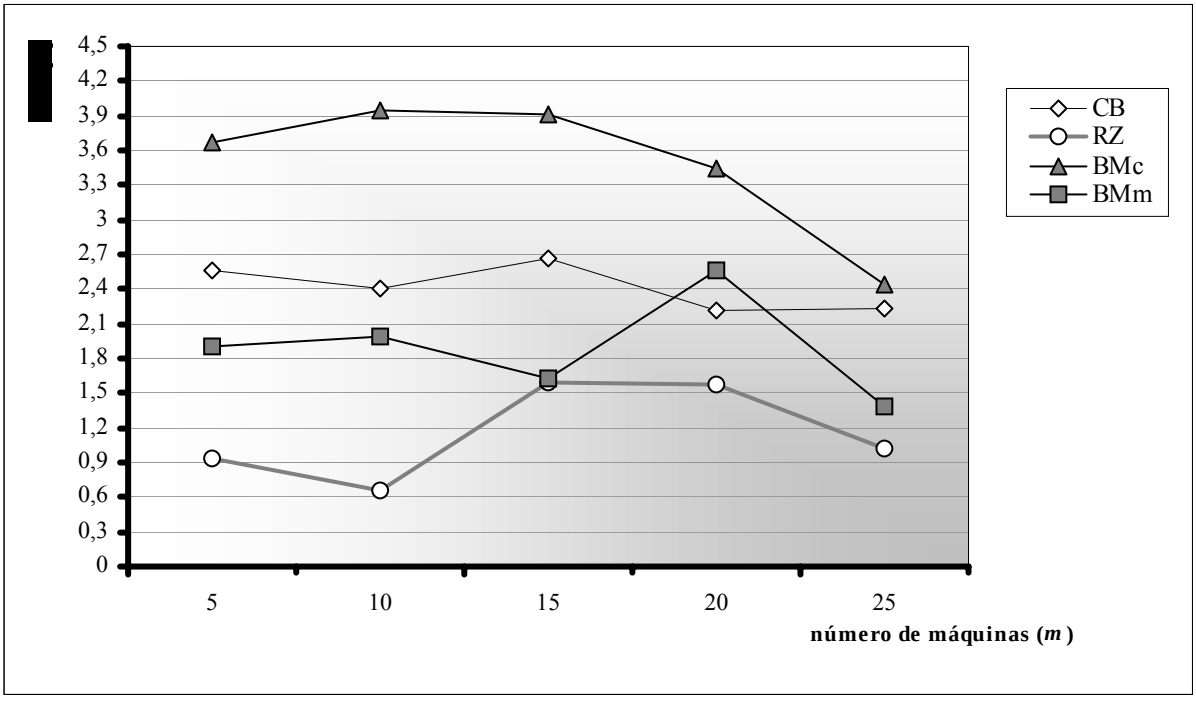

Gráfico 5.64. Desvio relativo médio (%), agregando-se as Relações IDs_{kj}-IDp_{kj} - $n = 4$

Tabela 5.18 – Desvio relativo médio (%), agregando-se as Relações IDs_{kj}-IDp_{kj} - $n = 6$

	MÉTODOS			
m	**CB**	**RZ_3**	**BM_c**	**BM_m**
5	2,36	1,95	1,77	1,33
10	1,94	1,47	2,77	1,69
15	2,50	0,95	1,95	1,51
20	2,15	1,73	2,01	1,37
25	2,14	1,10	3,03	1,92

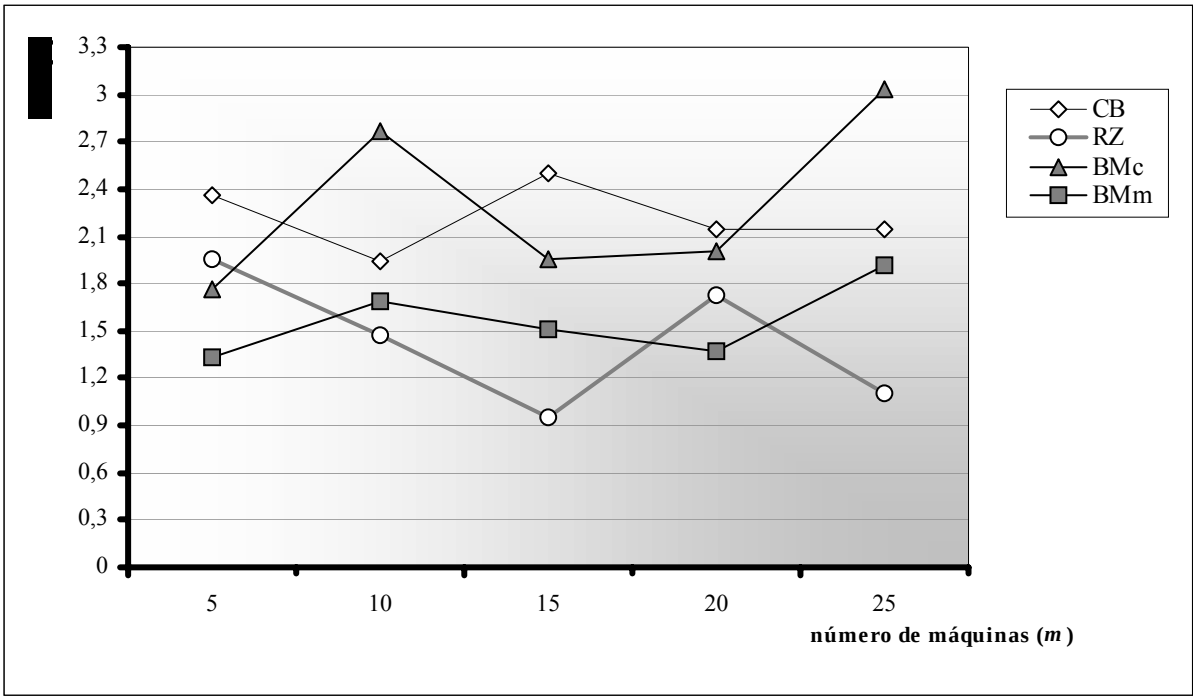

Gráfico 5.65. Desvio relativo médio (%), agregando-se as Relações IDs_{kj}-IDp_{kj} - $n = 6$

Tabela 5.19 – Desvio relativo médio (%), agregando-se as Relações IDs_{kj}-IDp_{kj} - $n = 7$

	MÉTODOS			
m	**CB**	**RZ_3**	**BM_c**	**BM_m**
5	2,60	0,98	2,05	1,17
10	2,01	1,15	1,98	1,13
15	2,07	1,04	2,29	1,51
20	1,98	1,09	2,03	1,48
25	2,01	1,03	2,34	1,25

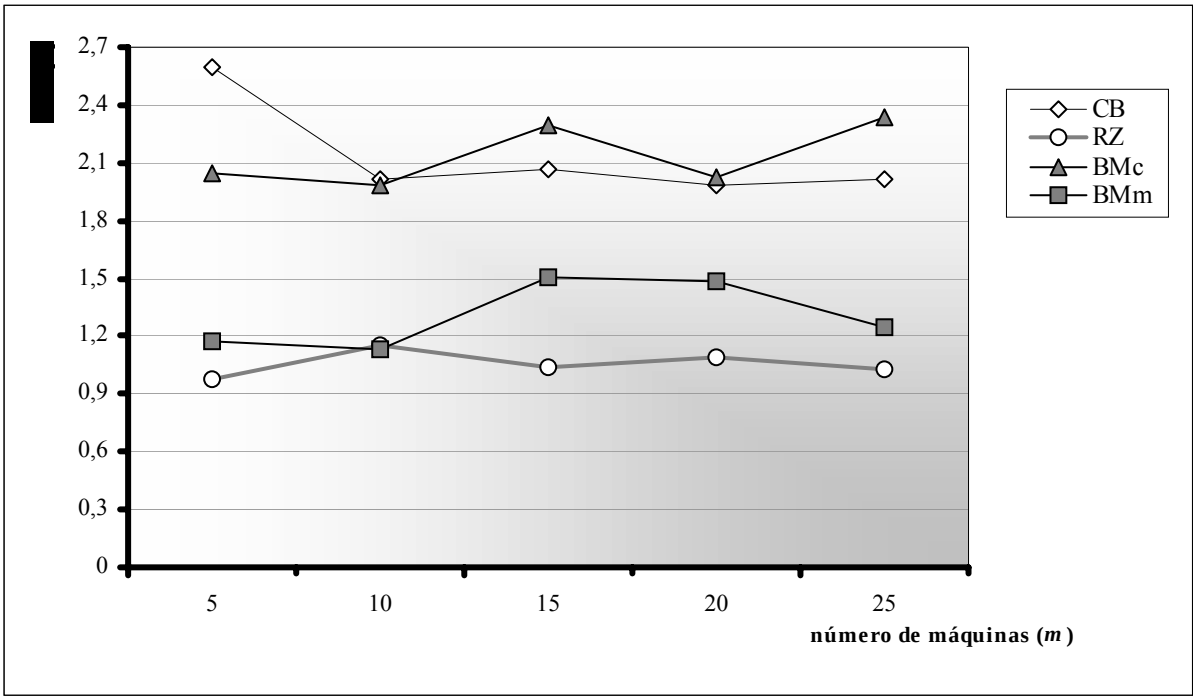

Gráfico 5.66. Desvio relativo médio (%), agregando-se as Relações IDs_{kj}-IDp_{kj} - $n = 7$

Tabela 5.20 – Desvio relativo médio (%), agregando-se as Relações IDs_{kj}-IDp_{kj} - $n = 20$

	MÉTODOS			
m	**CB**	**RZ_3**	**BM_c**	**BM_m**
5	1,38	0,69	1,02	0,66
10	1,22	0,64	1,05	0,54
15	1,07	0,53	0,70	0,56
20	1,18	0,41	0,88	0,57
25	1,18	0,67	0,73	0,67

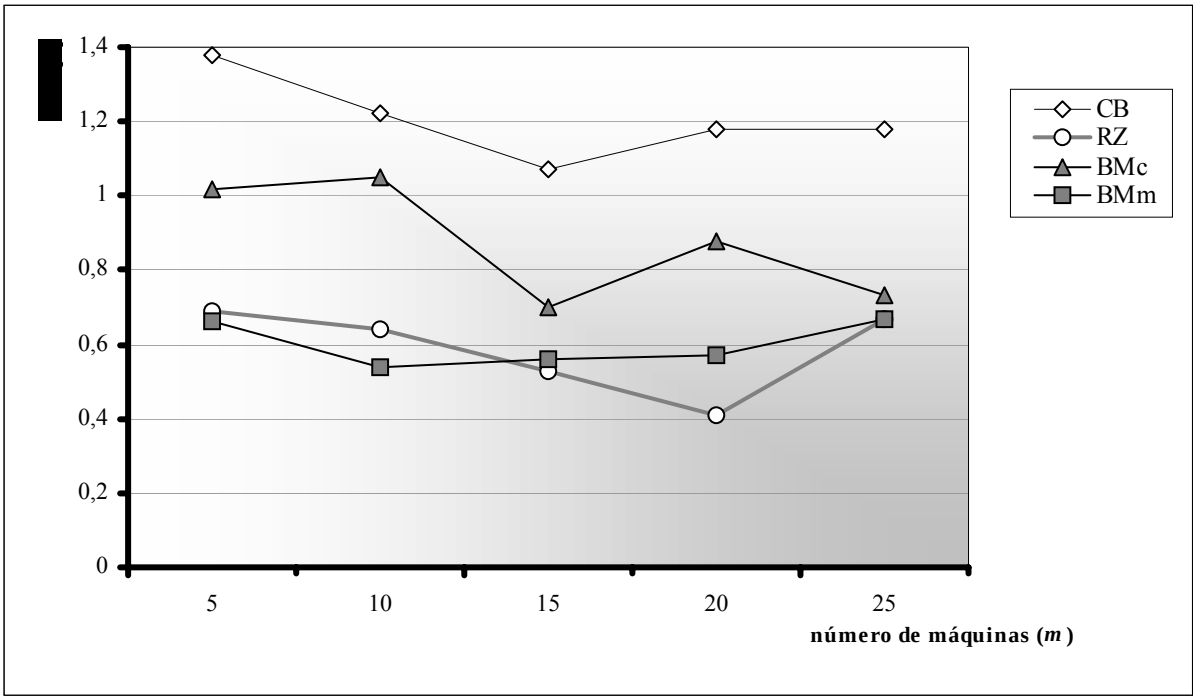

Gráfico 5.67. Desvio relativo médio (%), agregando-se as Relações IDs_{kj}-IDp_{kj} - $n = 20$

Como as porcentagens de sucesso do método CB para $n = 40$ e 60 são iguais a zero este método é desconsiderado na análise do desvio relativo médio (%) para esses valores de n.

Tabela 5.21 – Desvio relativo médio (%), agregando-se as relações IDs_{kj}-IDp_{kj} - $n = 40$

	MÉTODOS			
m	**CB**	**RZ_3**	**BM_c**	**BM_m**
5	--	0,53	0,26	0,12
10	--	0,33	0,35	0,15
15	--	0,26	0,31	0,18
20	--	0,33	0,33	0,23
25	--	0,35	0,34	0,15

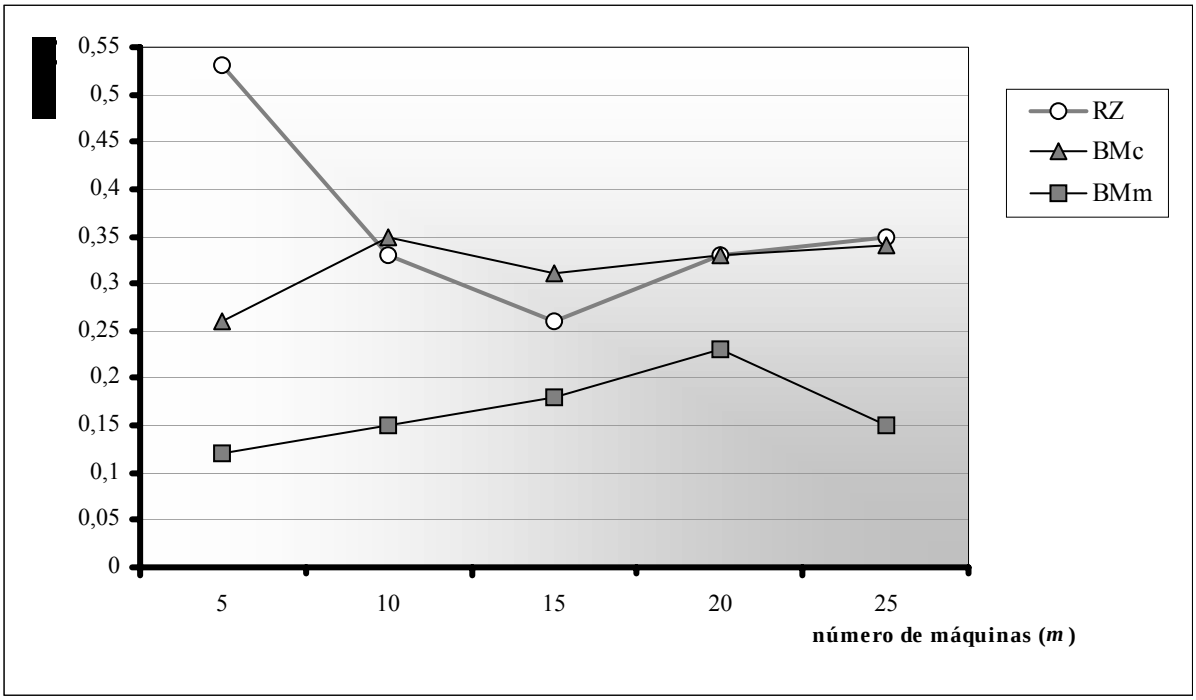

Gráfico 5.68. Desvio relativo médio (%), agregando-se as Relações IDs_{kj}-IDp_{kj} - $n = 40$

Tabela 5.22 – Desvio relativo médio (%), agregando-se as Relações IDs_{kj}-IDp_{kj} - n = 60

	MÉTODOS			
m	**CB**	**RZ_3**	**BM_c**	**BM_m**
5	--	0,25	0,23	0,04
10	--	0,23	0,29	0,05
15	--	0,25	0,23	0,11
20	--	0,20	0,22	0,12
25	--	0,23	0,46	0,36

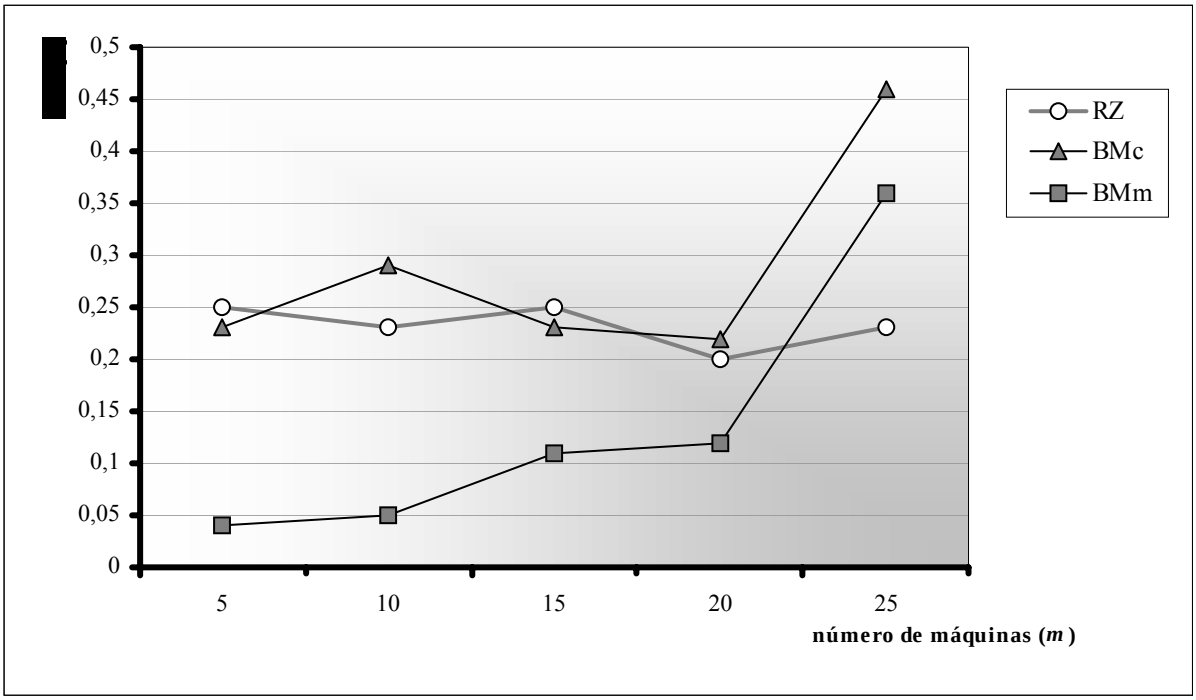

Gráfico 5.69. Desvio relativo médio (%), agregando-se as Relações IDs_{kj}-IDp_{kj} - n = 60

Os Gráficos 5.64 a 5.69 indicam que não houve, em ambos os métodos, influência do número de máquinas m no desvio relativo médio (%). Esses gráficos mostram ainda que o desvio relativo médio (%) não é significante, uma vez que não ultrapassam, no pior caso, 2% em média para todos os problemas.

A Tabela 5.23 apresenta uma comparação geral da performance dos métodos em todas as classes n, m, agregando-se as Relações IDs_{kj}-IDp_{kj}, com base no desvio relativo médio (%).

Tabela 5.23 – Resumo do desempenho dos métodos em termos de DRM (%)

	n					
m	4	6	7	20	40	60
5	RZ_3	**BM_m**	RZ_3	**BM_m**	**BM_m**	**BM_m**
10	RZ_3	RZ_3	**BM_m**	**BM_m**	**BM_m**	**BM_m**
15	RZ_3	RZ_3	RZ_3	RZ_3	**BM_m**	**BM_m**
20	RZ_3	**BM_m**	RZ_3	RZ_3	**BM_m**	**BM_m**
25	RZ_3	**BM_m**	RZ_3	**BM_m**/ RZ_3	**BM_m**	RZ_3

Nas 30 classes *n*, *m* analisadas o método proposto BM_m apresentou o menor desvio relativo em 15 classes, o método RZ_3 em 14, havendo 1 empate.

5.2.3 Tempo médio de computação (ms)

Durante a tabulação dos dados, foi possível observar que, numa mesma classe *n*, *m*, a diferença do tempo médio de computação (ms), entre os problemas resolvidos com base nas 4 Relações IDs_{kj}-IDp_{kj}, não foi significativa, tanto no caso de problemas de pequeno porte quanto nos de grande porte. Assim, optou-se por apresentar os resultados referentes aos tempos médios de computação (ms) agregando-se as relações IDs_{kj}-IDp_{kj}. As tabelas 5.24 e 5.25 apresentam estes resultados.

Os Gráficos 5.70 a 5.75 apresentam, para $n \in \{4, 6, 7, 20, 40, 60\}$ a comparação do tempo médio de computação entre os métodos CB, RZ_3, BM_c e BM_m, agregando-se as Relações IDs_{kj}-IDp_{kj}. Uma vez que, as porcentagens de sucesso do método CB para $n = 40$ e 60 são iguais a zero este método é desconsiderado na análise do tempo médio de computação (ms) para esses valores de *n*.

Tabela 5.24 – Tempo médio de computação (ms), agregando-se as Relações IDs_{kj}-IDp_{kj}, para os problemas de pequeno porte

	MÉTODOS			
	CB	**RZ_3**	**BM_c**	**BM_m**
m		*n* = 4		
5	0,27	0,52	0,13	0,53
10	0,25	1,18	0,13	0,39
15	1,18	1,17	0,25	0,27
20	1,08	1,31	0,39	0,39
25	1,57	1,55	0,52	0,51
		n = 6		
5	0,53	0,84	0,25	0,51
10	1,18	1,57	0,13	0,64
15	1,69	1,57	0,51	0,67
20	1,96	2,09	0,51	0,79
25	2,87	2,60	0,52	1,05
		n = 7		
5	0,53	1,31	0,26	0,52
10	1,18	2,08	0,13	0,92
15	1,73	2,08	0,70	0,87
20	2,48	2,48	0,65	1,03
25	3,52	3,12	0,79	1,31

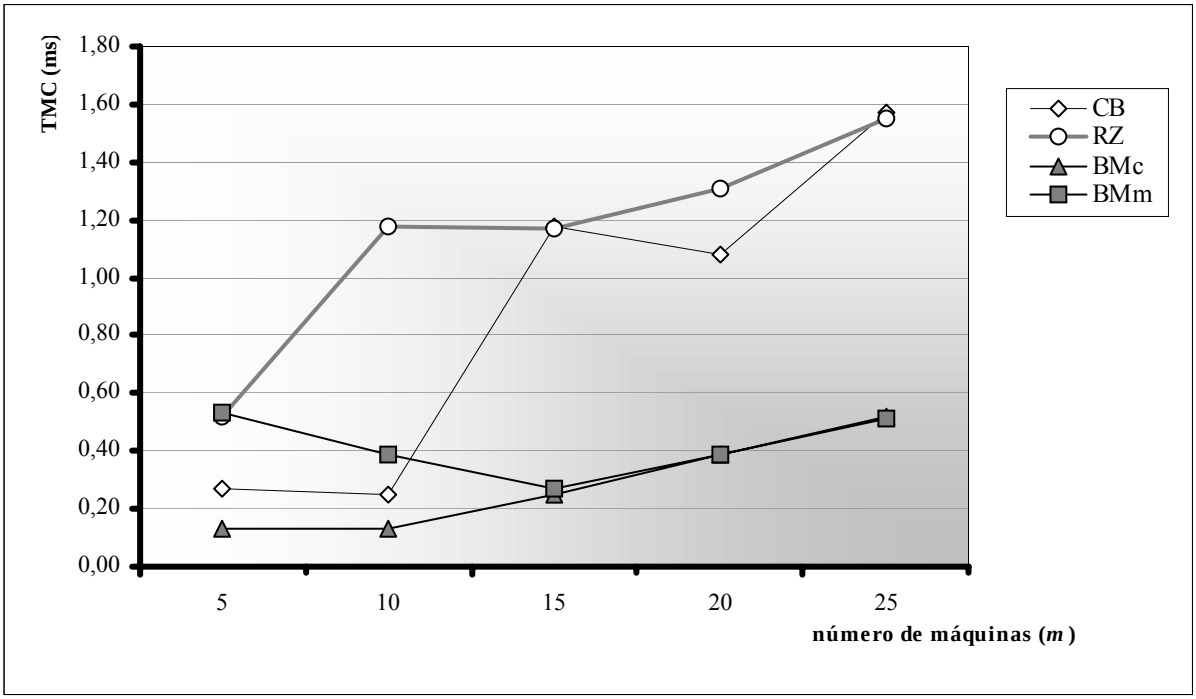

Gráfico 5.70. Tempo médio de computação (ms), agregando-se as Relações IDs_{kj}-IDp_{kj}, para *n* = 4

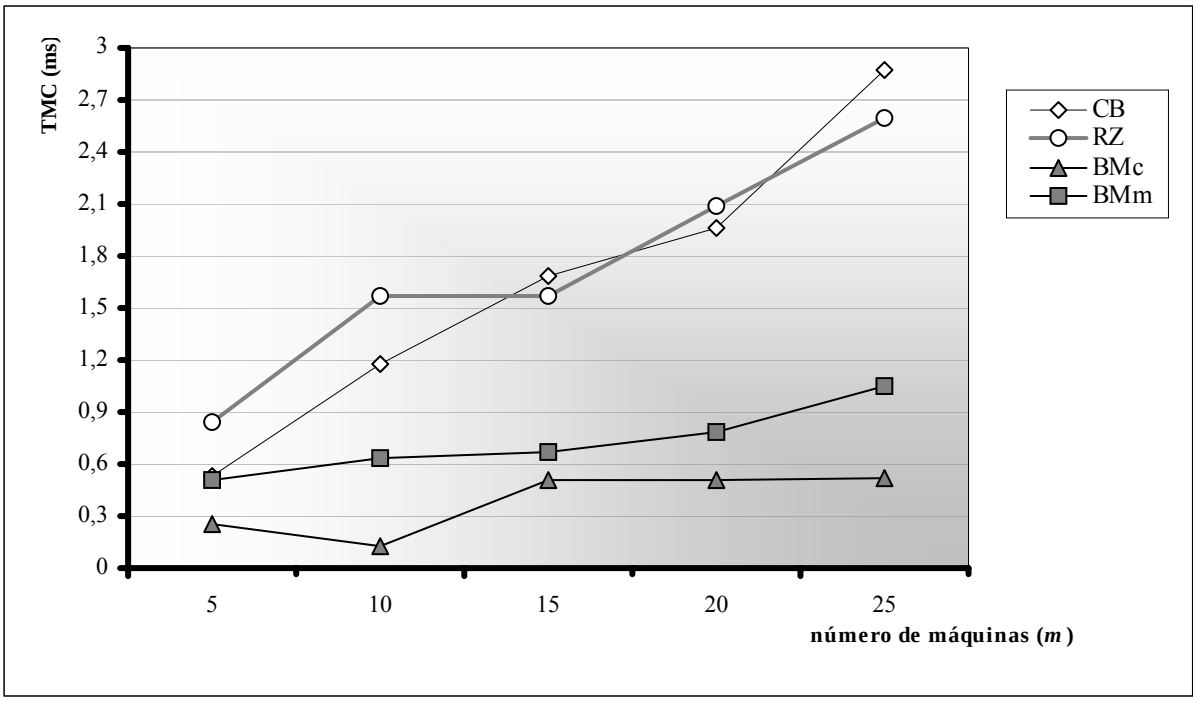

Gráfico 5.71. Tempo médio de computação (ms), agregando-se as Relações IDs_{kj}-IDp_{kj}, para $n = 6$

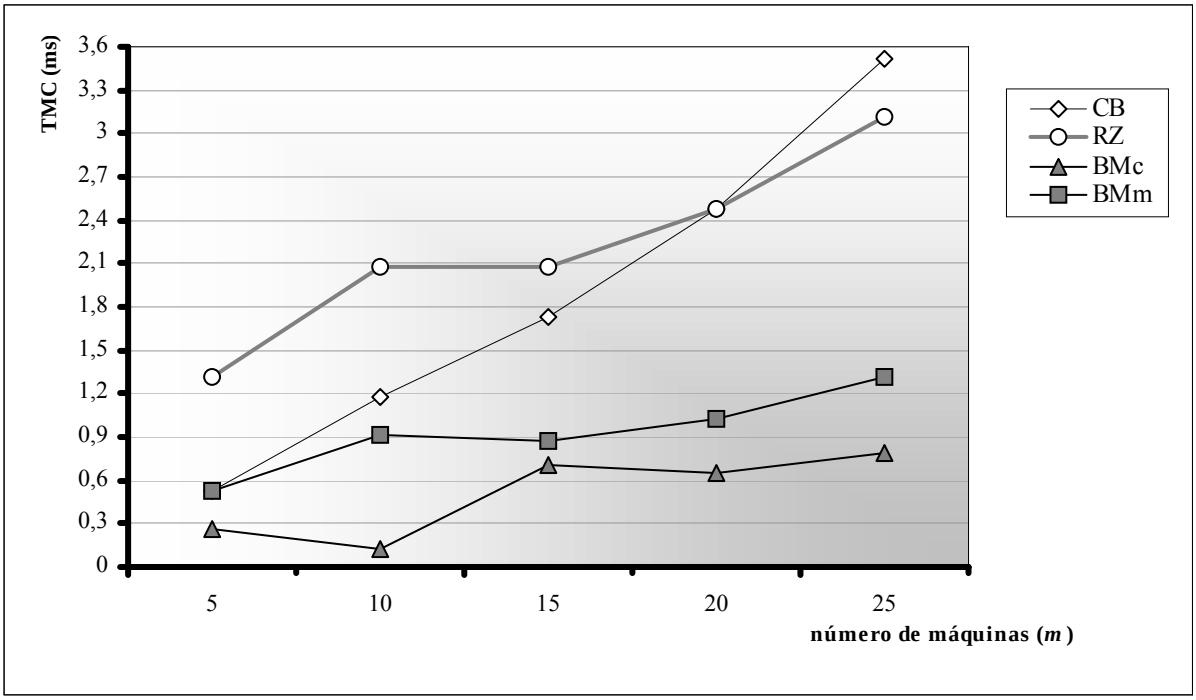

Gráfico 5.72. Tempo médio de computação (ms), agregando-se as Relações IDs_{kj}-IDp_{kj}, para $n = 7$

Tabela 5.25 – Tempo médio de computação (ms), agregando-se as Relações IDs_{kj}-IDp_{kj}, para os problemas de grande porte

	MÉTODOS			
	CB	**RZ_3**	**BM_c**	**BM_m**
		n = 20		
5	3,13	8,60	1,95	3,90
10	7,03	11,99	2,60	6,51
15	10,43	15,88	3,52	8,86
20	16,28	20,43	4,30	9,76
25	22,80	22,53	6,24	11,33
m		*n* = 40		
5	--	39,20	6,63	20,06
10	--	62,76	11,45	31,91
15	--	85,28	15,23	42,97
20	--	106,50	19,68	54,69
25	--	129,57	23,70	66,67
		n = 60		
5	--	107,43	17,31	52,88
10	--	179,03	29,29	87,51
15	--	246,61	40,24	121,74
20	--	313,28	51,43	154,56
25	--	387,12	63,94	192,32

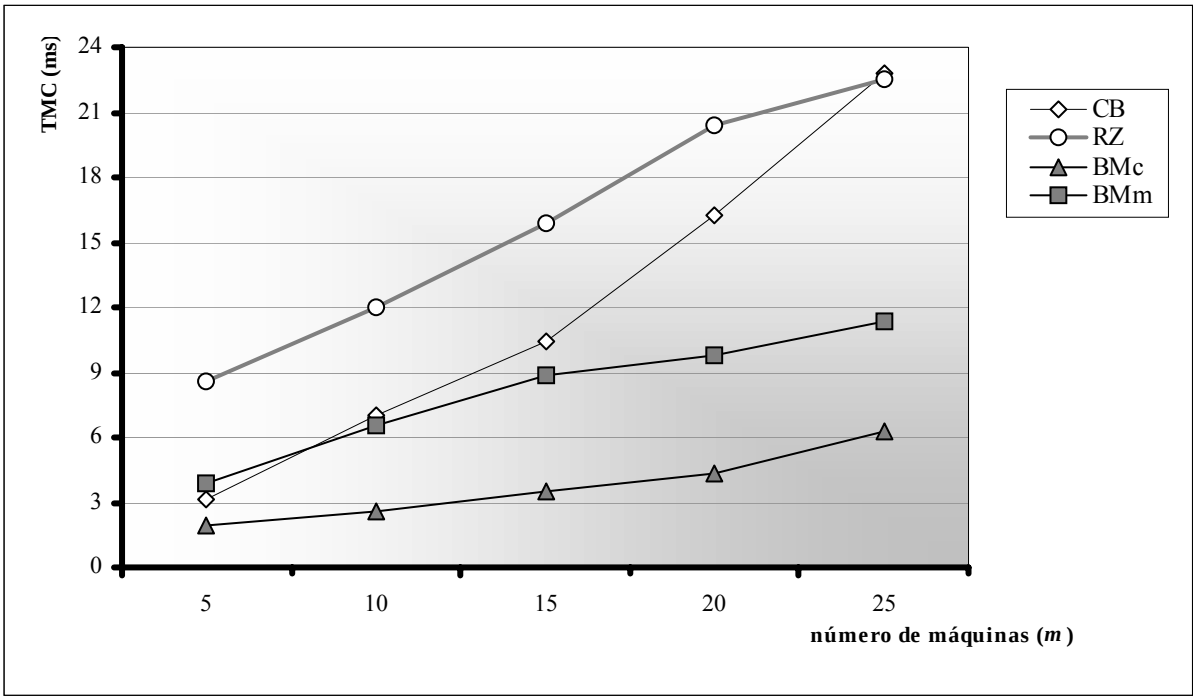

Gráfico 5.73. Tempo médio de computação (ms), agregando-se as Relações IDs_{kj}-IDp_{kj}, para *n* = 20

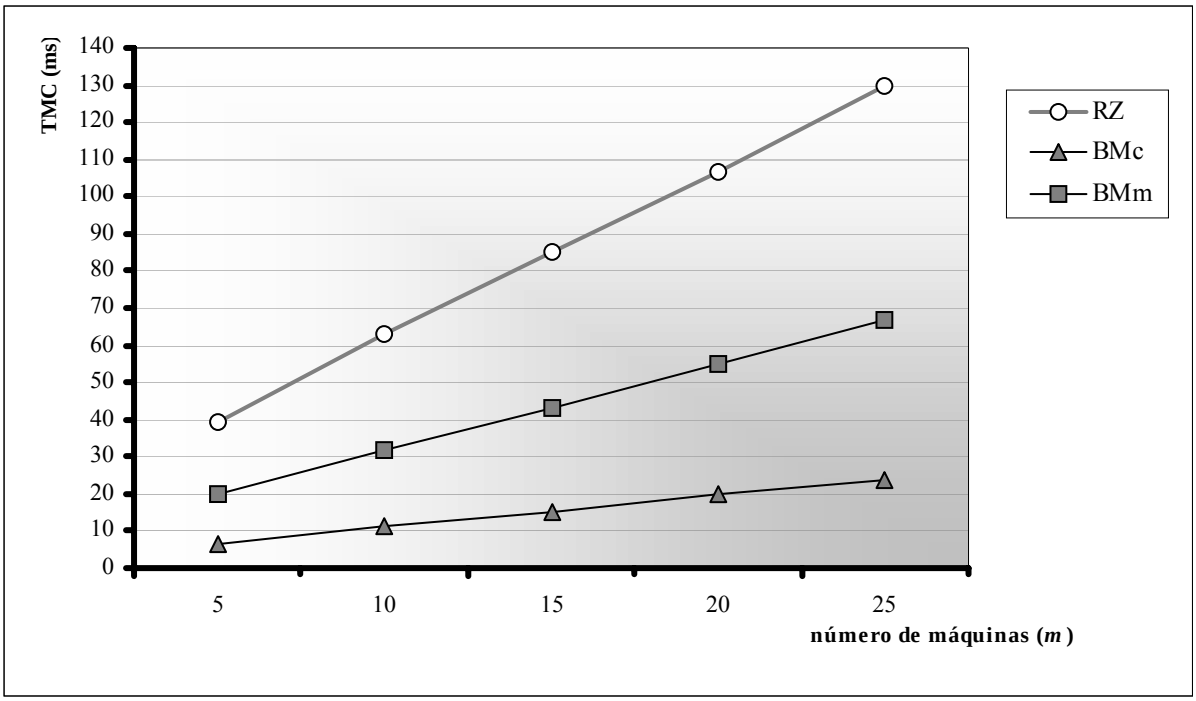

Gráfico 5.74. Tempo médio de computação (ms), agregando-se as Relações IDs_{kj}-IDp_{kj}, para $n = 40$

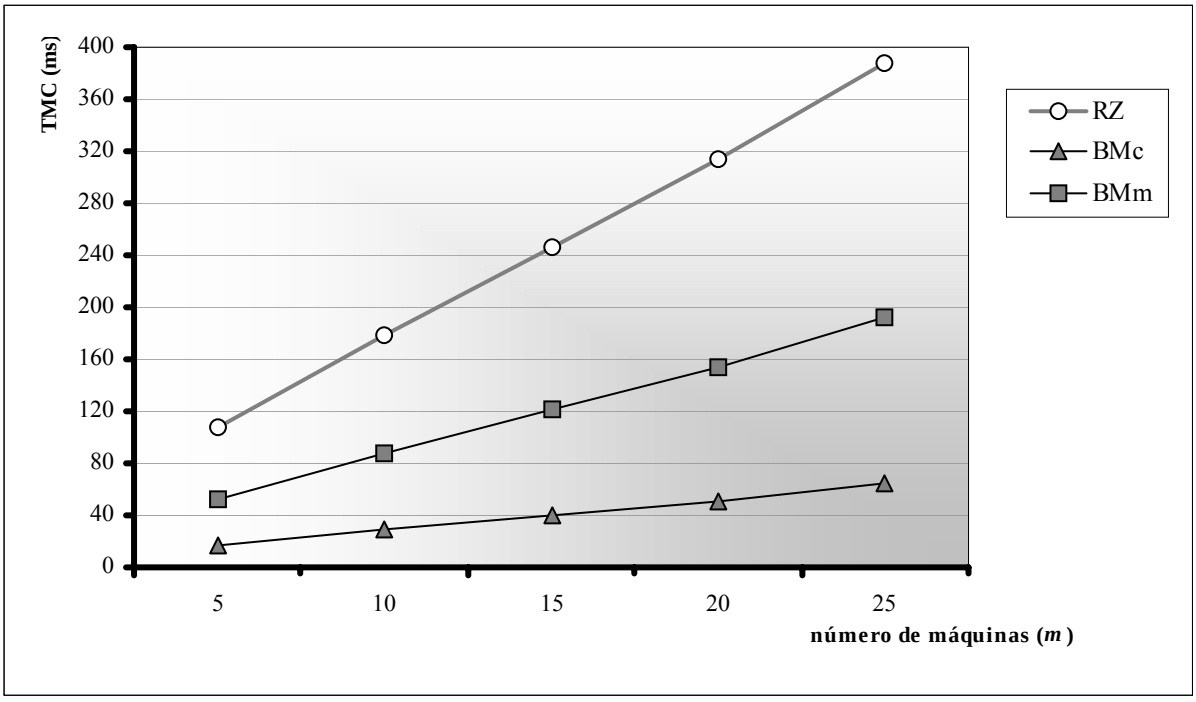

Gráfico 5.75. Tempo médio de computação (ms), agregando-se as Relações IDs_{kj}-IDp_{kj}, para $n = 60$

Nos gráficos 5.70 a 5.75 é possível observar que os tempos médios de computação (ms) de ambos os métodos são influenciados tanto o número de máquinas (m) quanto o número de tarefas (n).

Os valores dos tempos médios de computação do Método RZ_3 são explicados pelo fato de que ele executa os métodos RZ_1 e RZ_2, ambos melhorativos, para selecionar aquele que fornece a melhor solução, usando essa solução como "semente" para aplicar a fase de melhoramento novamente.

Os resultados referentes aos tempos médios de computação enfatizam ainda mais a qualidade dos métodos propostos BM, uma vez que, estes possuem alta porcentagem de sucesso e baixo desvio relativo médio (%).

6 CONCLUSÕES

Neste trabalho existem alguns aspectos principais que podem ser destacados.

O primeiro aspecto refere-se ao fato deste trabalho considerar os tempos de *setup* como separados dos tempos de processamento, uma vez que, embora esta consideração seja muito importante para os modernos ambientes de manufatura e serviço, a maioria da literatura existente sobre programação ignora esse fato, como foi possível constatar a partir do exame de literatura realizado.

Outro aspecto a se destacar, e com certeza, o mais relevante, é a "**Propriedade LBY**", apresentada no Capítulo 4 (Seção 4.2), que constitui uma contribuição ao conhecimento, já existente, sobre o problema de programação em *Flow Shop* Permutacional.

É importante destacar, também, que, neste trabalho apresenta-se uma primeira aplicação bem sucedida da "**Propriedade LBY**", que corresponde aos métodos propostos BM.

Os resultados obtidos na experimentação computacional evidenciam a qualidade do método heurístico construtivo proposto BM_c, uma vez que, considerando-se todos os problemas, este método apresentou uma porcentagem de sucesso de 52,75% a mais que o método melhorativo CB e de apenas 18,28 e 33,23% a menos do que os métodos melhorativos RZ_3 e BM_m, respectivamente, utilizando um tempo de computação muito menor. Considerando-se todos os problemas, o tempo médio de computação do método BM_c corresponde a apenas 17,22, 29,90 e 34,62% dos tempos médios de computação dos métodos RZ_3, CB e BM_m, respectivamente.

Os resultados mostram, também, o melhor desempenho do método BM_m, em relação a ambos os métodos CB e RZ_3, tanto no que diz respeito à porcentagem de sucesso quanto ao

tempo de computação. Considerando-se todos os problemas, o método BM_m apresentou uma porcentagem de sucesso respectivamente, 64,53, 24,94 e 11,22% maior do que as porcentagens de sucesso dos métodos CB, BM_c e RZ_3. Além disso, o tempo médio de computação, em todos os problemas, do método BM_m é em torno de 50 e 40% menor do que os tempos médios de computação dos métodos RZ_3 e CB, respectivamente.

Outro aspecto a se destacar é a maneira completa e didática como as hipóteses do problema são apresentadas neste trabalho, podendo ser utilizadas em trabalhos futuros.

Este trabalho reforça a qualidade do procedimento de inserção de tarefas dos métodos NEH e N&M.

Muitas sugestões sobre o desenvolvimento de futuros trabalhos podem ser feitas.

Sugere-se realizar um estudo, para avaliar a performance dos métodos BM em alcançar a solução ótima.

A realização de um estudo detalhado da influência de ordenações iniciais para métodos melhorativos.

Um estudo que objetive determinar novas formas de ordenação inicial das tarefas, utilizando a "**Propriedade LBY**".

A criação de novos procedimentos de inserção de tarefas.

O desenvolvimento de um método *Branch-and-Bound*, utilizando-se a "**Propriedade LBY**", para obter um eficiente *Lower Bound* do *Makespan*.

A utilização da "**Propriedade LBY**", para conceber um método heurístico para o caso dos tempos de *setup* serem dependentes da seqüência de execução das tarefas, uma vez que, esta considera tanto os *setup* independentes quanto os dependentes da seqüência de execução das tarefas.

Sugere-se também, a realização de adaptações da "**Propriedade LBY**" para os casos em que os tempos de remoção e/ou transferencia são diferentes de zero e não podem ser considerados como parte do tempo de processamento.

A criação de novas propriedades considerando outras medidas de desempenho, que não o *Makespan*, além da criação de propriedades que considerem bicritérios.

REFERÊNCIAS[28]

AL-ANZI, F.; ALLAHVERDI, A. A self-adaptive differential evolution heuristic for two-stage assembly scheduling problem to minimize maximum lateness with setup times. **European Journal of Operational Research**, Amsterdam, v. 182, n. 1, p. 80-94, oct. 2007.

_____. Empirically discovering dominance relations for scheduling problems using an evolutionary algorithm. **International Journal of Production Research**, v. 44, n. 22, p. 4701-4712, 2006.

_____. Using a hybrid evolutionary algorithm to minimize variance in response time for multimedia object requests. **Journal of Mathematical Modelling and Algorithms**, n. 4, p. 435-453, 2005.

ALLAHVERDI, A. Minimizing mean flowtime in a two-machine flowshop with sequence-independent setup times. **Computers & Operations Research**, Oxford, v. 27, n. 2, p. 111-127, 2000.

_____. Scheduling in stochastic flowshop with independent setup, processing and removal times. **Computers & Operations Research**, Oxford, v. 24, n. 10, p. 955-960, 1997.

_____. Two-stage production scheduling with separated set-up times and stochastic breakdowns. **Journal of the Operational Research Society**, Oxford, v. 46, n. 7, p. 896-904, 1995.

_____.; AL-ANZI, F. S. A branch-and-bound algorithm for three-machine flowshop scheduling problem to minimize total completion time with separate setup times. **European Journal of Operational Research**, Amsterdam, v. 169, p. 767-780, 2006.

_____.; AL-ANZI, F. S. Using two-machine flowshop with maximum lateness objective to model multimedia data objects scheduling problem for WWW applications. **Computers and Operations Research**, v. 29, p. 971-994, 2002.

_____.; ALDOWAISAN, T. Job lateness in flowshops with setup and removal times separated. **Journal of the Operational Research Society**, Oxford, v. 49, p. 1001-1006, 1998.

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

[28] De acordo com:
ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR 6023**: informação e documentação: referências: elaboração. Rio de Janeiro, 2002.

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

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

ARTIBA, A.; ELMAGHRABY, S. E. Introduction. In: ARTIBA, A.; ELMAGHRABY, S. E. (1997). **The planning and scheduling of production systems**: methodologies and applications. London: Chapman & Hall.

BAKER, K. R. **Introduction to sequencing and scheduling**. New York: John Wiley & Sons, Inc., 1974.

BEDWORTH, D. D.; BAILEY, J. E. **Integrated Production Control Systems**: management, analysis, design. 2 ed. New York: John Wiley & Sons, Inc., 1987.

BRUCKER, P.; KNUST, S.; WANG, G. Complexity results for flow-shop problems with a single-server. **European Journal of Operational Research**, Amsterdam, v. 165, n. 2, p. 398-407, sep. 2005.

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

CAO, J.; BEDWORTH, D. D. Flow shop scheduling in serial multi-product process with transfer and set-up times. **International Journal of Production Research**, London, v. 30, n. 8, p. 1819-1830, 1992.

CHENG, T. C. E.; GUPTA, J. N. D.; WANG, G. A review of flowshop scheduling research with setup times. **Production and Operations Management**. Munich, v. 9, n. 3, p. 262-282, 2000.

CORRÊA, H. L.; GIANESI, I. G. N.; CAON, M. **Planejamento, Programação e Controle da Produção: MRP II/ERP**: conceitos, uso e implantação. 4 ed. São Paulo: Atlas, 2001.

DILEEPAN, P.; SEN, T. Job lateness in a two-machine flowshop with setup times separated. **Computers & Operations Research**, Oxford, v. 18, n. 6, p. 549-556, 1991.

FOGARTY, D. W.; BLACKSTONE Jr., J. H.; HOFFMANN, T. R. **Production & Inventory Management.** 2 ed. Cincinnati: South-Western Publishing Co., 1991.

FRENCH, S. **Sequencing and Scheduling**: an introduction to the mathematics of the job-shop. New York: John Wiley & Sons, Inc., 1982.

FUCHIGAMI, H. Y. **Métodos heurísticos construtivos para o problema de programação da produção em sistemas *flow shop* híbridos com tempos de preparação das máquinas assimétricos e dependentes da seqüência**. 2005. 134 f. Dissertação (Mestrado em Engenharia de Produção) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos/SP, 2005.

GAITHER, N.; FRAZIER, G. **Administração da Produção e Operações**. 8 ed. Trad. J. C. B. dos Santos. São Paulo: Thomson Learning, 2002. (Original do Inglês).

GLASS, C. A.; SHAFRANSKY, Y. M.; STRUSEVICH, V. A. Scheduling for parallel dedicated machines with a single server. **Naval Research Logistics**, v, 47, n. 4, 304-328, 2000.

GUPTA, J. N. D.; STAFFORD JR., E. F. Flowshop scheduling research after five decades. **European Journal of Operational Research**, Amsterdam, v. 169, p. 699-711, 2006.

_____.; TUNC, A. Scheduling a two-stage hybrid flowshop with separable setup and removal times. **European Journal of Operational Research**, Amsterdam, v. 77, n. 3, p. 415-428, 1994.

HAX, A. C.; CANDEA, D. **Production and inventory management**. New Jersey: Prentice-Hall, 1984.

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

KOGAN, K.; LEVNER, E.; A polynomial algorithm for scheduling small-scale manufacturing cell served by multiple robots. **Computers and Operations Research**, v. 25, p. 53-62, 1998.

LOGENDRAN, R.; SRISKANDARAJAH, C. Two-machine group scheduling problem with blocking and anticipatory setups. **European Journal of Operational Research**, Amsterdam, v. 69, n. 1, p. 467-481, 1993.

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

MOCCELLIN, J. V. A new heurist method for the permutation flow shop scheduling problem. **Journal of the Operational Research Society**, Oxford, v. 46, p. 883-886, 1995.

_____. **Uma contribuição à Programação de Operações em Sistemas de Produção Intermitente '*Flow-Shop*'**. 1992. 126 f. Tese (Livre-Docência) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos/SP, 1992.

_____.; NAGANO, M. S. Uma propriedade estrutural do problema de programação da produção *flow shop* permutacional com tempos de *setup* separados dos tempos de processamento. In: SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL, XXXVIII, 2006, Goiânia. **Anais**... Rio de Janeiro: SOBRAPO, 2006. 1 CD-ROM.

MONKS, J. G. **Operations Management**: Theory and Problems. MacGraw-Hill Book Co., 1984.

NAGANO, M. S. **Um novo método heurístico construtivo de alto desempenho para a programação de operações *flow-shop* permutacional**. 1999. 165 f. Tese (Doutorado em Engenharia de Produção) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos/SP, 1999.

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

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

PANWALKAR, S. S. Scheduling of a two-machine flowshop with travel time between machines. **Journal of the Operational Research Society**, Oxford, v. 42, n. 7, p. 609-613, 1991.

PARK, T.; STEUDEL, H. J. Analysis of heuristics for job sequencing in manufacturing flow line work-cells. **Computer & Industrial Engineering**. Exeter, v. 20, n. 1, p. 129-140, 1991.

PORTMANN, M. C. Scheduling methodology: optimization and compu-search approaches I. In: ARTIBA, A.; ELMAGHRABY, S. E. **The planning and scheduling of production systems**: methodologies and applications. London: Chapman & Hall, 1997.

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

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

RUIZ, R.; MAROTO, C. A comprehensive review and evaluation of permutation flowshop heuristics. **European Journal of Operational Research**, Amsterdam, v. 165, p. 479-494, 2005.

SÁ MOTTA, I. Planejamento e Controle da Produção. In: MACHLINE, C. et al. **Manual de Administração da Produção**. Rio de Janeiro: FGV, 1972. v. 1.

SOUZA, A. B. D.; MOCCELLIN, J. V. Metaheurística híbrida Algoritmo Genético Busca-tabu para a programação de operações *flow shop*. In: SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL, XXII, 2000, Viçosa. **Anais**... Rio de Janeiro: SOBRAPO, 2000. 1 CD-ROM.

SU, Ling-Huey; CHOU, Fuh-Der Heuristic for scheduling in a two-machine bicriteria dynamic flowshop with setup and processing times separated. **Production Planning & Control**. Abington, v. 11, n. 8, p. 806-819, 2000.

SZWARC, W. Flow shop problems with time lags. **Management Science**, Rhode Island, v. 29, Iss. 4, p. 477-480, 1983.

_____.; GUPTA, J. N. D. A flow-Shop problem with sequence-dependent additive setup times. **Naval Research Logistics**. v. 34, p. 619-627, 1987.

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

TUBINO, D. F. **Manual de Planejamento e Controle da Produção**. São Paulo: Atlas, 2000.

YANG, Wen-Hwa; LIAO, Ching-Jong. Survey of scheduling research involving setup times. **International Journal of Systems Science**, Abington, v. 30, n. 2, p. 143-155, 1999.

ZACCARELLI, S. B. **Programação e controle da Produção**. 5 ed. São Paulo: Pioneira, 1979.

APÊNDICES

APÊNDICE 1 – Formatos dos Arquivos de Dados e de Saída

Os formatos dos arquivos de dados gerados para cada classe de problemas *n*, *m* e e Relações IDs_{kj}-IDp_{kj}, bem como dos respectivos, arquivos de saídas, todos de extensão *.txt*, são a seguir apresentados.

Nos arquivos de dados, conforme é possível visualizar na Figura A-1.1 as informações são separadas por ponto, ponto e vírgula e por linhas.

Nas três primeiras linhas tem-se, respectivamente, nesta ordem, o número de máquinas *m*, o número de tarefas *n* e número de problemas gerados.

Após a primeira linha em branco, o arquivo fornece a matriz dos tempos de processamento e de *setup* em todas as *m* máquinas para todas as *n* tarefas, para todos os problemas gerados.

Os dados de uma máquina para outra são separados por uma linha em branco.

Os valores das primeiras colunas correspondem aos tempos de processamento, enquanto que, os valores da segunda, aos tempos de *setup*. Estes tempos são separados por um ponto.

Cada linha representa os valores correspondentes a cada tarefa, separadas por ponto e vírgula, portanto, o número de linhas entre duas linhas em branco é igual ao número de tarefas.

Duas linhas em branco separam os dados de um problema para outro.

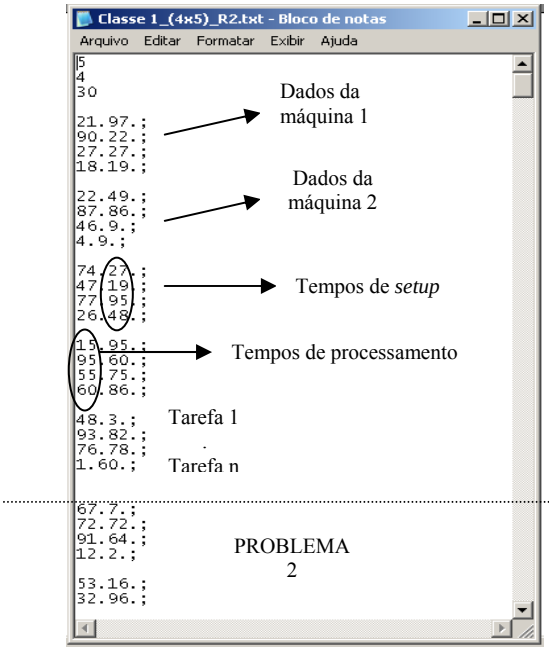

Figura A-1.1. Exemplo de um arquivo de dados dos problemas

A Figura A-1.1 exemplifica o caso em que o número de tarefas é igual a 4, o número de máquinas igual a 5 e o número de problemas gerados é igual a 30.

Um arquivo de saída com os resultados baseados no arquivo de dados apresentado na da Figura A-1.2 é mostrado na Figura A-1.2.

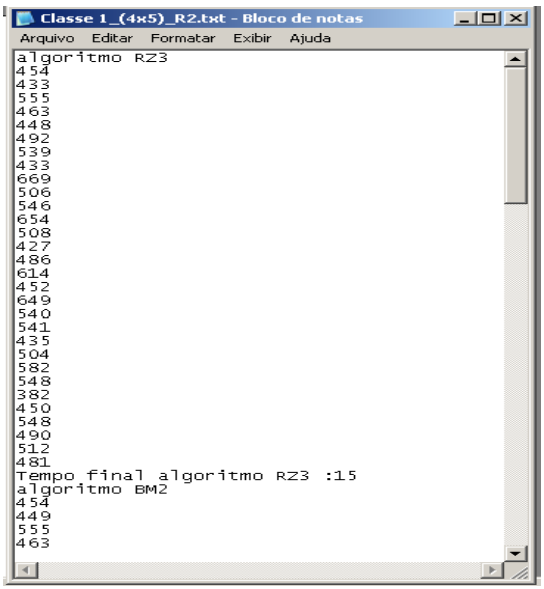

Figura A-1.2. Exemplo de um arquivo de saída

No arquivo de saída os resultados são apresentados em linha, na seguinte seqüência: o nome do método selecionado, através do *software* "PF/s.n-b.i.ST/Cmax", para resolver o problema; os valores dos *Makespans* das soluções encontradas em todos os problemas, um abaixo do outro; tempo de computação (ms) para resolver todos os problemas; logo em seguida, os resultados correspondentes ao outro método selecionado para resolver os problemas.

ANDICE 2 – Resumo do Código Fonte do *Software* “PF/s.n-b.i.ST/Cmax”

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Buttons;

type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    Button5: TButton;
    OpenDialog1: TOpenDialog;
    BitBtn1: TBitBtn;
    Button6: TButton;
    Button7: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button5Click(Sender: TObject);
    procedure Button6Click(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure Button7Click(Sender: TObject);
  private
    { Private declarations }
  public
     procedure carregamatriz();
     function calculo_makespan(): integer;


    { Public declarations }
  end;

var
  Form1: TForm1;
  maq1, maq2, maq3, maq4, maq5, maq6, maq7, maq8, maq9, maq10: array[1..60,1..2]of
integer;
  maq11, maq12, maq13, maq14, maq15, maq16, maq17, maq18, maq19, maq20:
array[1..60,1..2]of integer;
  maq21, maq22, maq23, maq24, maq25: array[1..60,1..2]of integer;
```

```
  mseo, novaseq1, seq_base: array[1..60] of integer;
  numtarefas, numdemaquinas, infinito, argumentos, val_makespan, quantprob : integer;
  cb : string;
  arq, arqtempo : string;
  arquivo, arquivotempo: TextFile;
implementation

uses Unit2, Unit3, Unit4, Unit5, Unit7, Unit6;

{$R *.dfm}

                                                    //CARREGAR MATRIZ
Procedure TForm1.carregamatriz();
var

  str, aux: String;
  i,j,  k, w: integer;
begin

       for w:= 1 to numdemaquinas do
         begin
            for k:=1 to numtarefas do
              begin
                Readln(arquivo, str);
                i:=1;
                j:=1;
                while str[i] <> ';' do
                  begin
                     aux:= '';
                     while str[i] <> '.' do
                       begin
                          aux:= aux + str[i];
                          i:= i+1;
                       end;
                       if w = 1 then
                          maq1[k,j]:= strtoint(aux);
                       if w = 2 then
                          maq2[k,j]:= strtoint(aux);
                       if w = 3 then
                          maq3[k,j]:= strtoint(aux);
                       if w = 4 then
                          maq4[k,j]:= strtoint(aux);
                       if w = 5 then
                          maq5[k,j]:= strtoint(aux);
                       if w = 6 then
                          maq6[k,j]:= strtoint(aux);
                       if w = 7 then
                          maq7[k,j]:= strtoint(aux);
                       if w = 8 then
                          maq8[k,j]:= strtoint(aux);
```

```
                    if w = 9 then
                       maq9[k,j]:= strtoint(aux);
                    if w = 10 then
                       maq10[k,j]:= strtoint(aux);
                    if w = 11 then
                       maq11[k,j]:= strtoint(aux);
                    if w = 12 then
                       maq12[k,j]:= strtoint(aux);
                    if w = 13 then
                       maq13[k,j]:= strtoint(aux);
                    if w = 14 then
                       maq14[k,j]:= strtoint(aux);
                    if w = 15 then
                       maq15[k,j]:= strtoint(aux);
                    if w = 16 then
                       maq16[k,j]:= strtoint(aux);
                    if w = 17 then
                       maq17[k,j]:= strtoint(aux);
                    if w = 18 then
                       maq18[k,j]:= strtoint(aux);
                    if w = 19 then
                       maq19[k,j]:= strtoint(aux);
                    if w = 20 then
                       maq20[k,j]:= strtoint(aux);
                    if w = 21 then
                       maq21[k,j]:= strtoint(aux);
                    if w = 22 then
                       maq22[k,j]:= strtoint(aux);
                    if w = 23 then
                       maq23[k,j]:= strtoint(aux);
                    if w = 24 then
                       maq24[k,j]:= strtoint(aux);
                    if w = 25 then
                       maq25[k,j]:= strtoint(aux);
                    j:=j+1;
                    i:= i+1;
                end;
            end;
            Readln(arquivo, str);
        end;
        Readln(arquivo, str);
end;

                                                    //CÁLCULO DO MAKESPAN
function Tform1.calculo_makespan(): integer;
var
   aux, i, j : integer;
   mat_aux: array[1..30,1..60] of integer;
begin
  aux:= 0;
```

```
for i:=1 to 25 do
  for j:=1 to 60 do
     mat_aux[i,j]:= 0;

for i:=1 to numdemaquinas do
  begin
    aux:= 0;
    for j:=1 to numtarefas do
      begin
        if i = 1 then
          begin
            aux:= aux +  maq1[seq_base[j],1] +  maq1[seq_base[j],2];
            mat_aux[i,j]:= aux;
          end;
        if i = 2 then
          begin
            if j = 1 then
              begin
                aux := maq2[seq_base[j],2];
              end
            else
              aux:= aux + maq2[seq_base[j],2];
            if aux > mat_aux[1,j] then
              begin
                aux:= aux +  maq2[seq_base[j],1];
                mat_aux[i,j]:= aux;
              end
            else
              begin
                aux:=0;
                aux:= mat_aux[1,j] + maq2[seq_base[j],1];
                mat_aux[i,j]:= aux;
              end;
          end;
        if i = 3 then
          begin
            if j = 1 then
              aux := maq3[seq_base[j],2]
            else
              aux:= aux + maq3[seq_base[j],2];
            if aux > mat_aux[2,j] then
              begin
                aux:= aux +  maq3[seq_base[j],1];
                mat_aux[i,j]:= aux;
              end
            else
              begin
                aux:= mat_aux[2,j] + maq3[seq_base[j],1];
                mat_aux[i,j]:= aux;
              end;
```

```
  end;
if i = 4 then
  begin
    if j = 1 then
     begin
       aux := maq4[seq_base[j],2];
     end
    else
      aux:= aux + maq4[seq_base[j],2];
    if aux > mat_aux[1,j] then
      begin
        aux:= aux +  maq4[seq_base[j],1];
        mat_aux[i,j]:= aux;
      end
    else
      begin
       aux:=0;
       aux:= mat_aux[1,j] + maq4[seq_base[j],1];
       mat_aux[i,j]:= aux;
      end;
  end;
if i = 5 then
  begin
    if j = 1 then
      aux := maq5[seq_base[j],2]
    else
      aux:= aux + maq5[seq_base[j],2];
    if aux > mat_aux[2,j] then
      begin
        aux:= aux +  maq5[seq_base[j],1];
        mat_aux[i,j]:= aux;
      end
    else
      begin
       aux:= mat_aux[2,j] + maq5[seq_base[j],1];
       mat_aux[i,j]:= aux;
      end;
  end;
if i = 6 then
  begin
    if j = 1 then
      aux := maq6[seq_base[j],2]
    else
      aux:= aux + maq6[seq_base[j],2];
    if aux > mat_aux[2,j] then
      begin
        aux:= aux +  maq6[seq_base[j],1];
        mat_aux[i,j]:= aux;
      end
    else
```

```
        begin
          aux:= mat_aux[2,j] + maq6[seq_base[j],1];
          mat_aux[i,j]:= aux;
        end;
  end;
if i = 7 then
  begin
    if j = 1 then
      aux := maq7[seq_base[j],2]
    else
      aux:= aux + maq7[seq_base[j],2];
    if aux > mat_aux[2,j] then
      begin
        aux:= aux +  maq7[seq_base[j],1];
        mat_aux[i,j]:= aux;
      end
    else
      begin
        aux:= mat_aux[2,j] + maq7[seq_base[j],1];
        mat_aux[i,j]:= aux;
      end;
  end;
if i = 8 then
  begin
    if j = 1 then
      aux := maq8[seq_base[j],2]
    else
      aux:= aux + maq8[seq_base[j],2];
    if aux > mat_aux[2,j] then
      begin
        aux:= aux +  maq8[seq_base[j],1];
        mat_aux[i,j]:= aux;
      end
    else
      begin
        aux:= mat_aux[2,j] + maq8[seq_base[j],1];
        mat_aux[i,j]:= aux;
      end;
  end;
if i = 9 then
  begin
    if j = 1 then
      aux := maq9[seq_base[j],2]
    else
      aux:= aux + maq9[seq_base[j],2];
    if aux > mat_aux[2,j] then
      begin
        aux:= aux +  maq9[seq_base[j],1];
        mat_aux[i,j]:= aux;
      end
```

```
    else
      begin
        aux:= mat_aux[2,j] + maq9[seq_base[j],1];
        mat_aux[i,j]:= aux;
      end;
  end;
if i = 10 then
  begin
    if j = 1 then
      aux := maq10[seq_base[j],2]
    else
      aux:= aux + maq10[seq_base[j],2];
    if aux > mat_aux[2,j] then
      begin
        aux:= aux +  maq10[seq_base[j],1];
        mat_aux[i,j]:= aux;
      end
    else
      begin
        aux:= mat_aux[2,j] + maq10[seq_base[j],1];
        mat_aux[i,j]:= aux;
      end;
  end;
if i = 11 then
  begin
    if j = 1 then
      aux := maq11[seq_base[j],2]
    else
      aux:= aux + maq11[seq_base[j],2];
    if aux > mat_aux[2,j] then
      begin
        aux:= aux +  maq11[seq_base[j],1];
        mat_aux[i,j]:= aux;
      end
    else
      begin
        aux:= mat_aux[2,j] + maq11[seq_base[j],1];
        mat_aux[i,j]:= aux;
      end;
  end;
if i = 12 then
  begin
    if j = 1 then
      aux := maq12[seq_base[j],2]
    else
      aux:= aux + maq12[seq_base[j],2];
    if aux > mat_aux[2,j] then
      begin
        aux:= aux +  maq12[seq_base[j],1];
        mat_aux[i,j]:= aux;
```

```
        end
      else
        begin
          aux:= mat_aux[2,j] + maq12[seq_base[j],1];
          mat_aux[i,j]:= aux;
        end;
  end;
if i = 13 then
  begin
    if j = 1 then
      aux := maq13[seq_base[j],2]
    else
      aux:= aux + maq13[seq_base[j],2];
    if aux > mat_aux[2,j] then
      begin
        aux:= aux +  maq13[seq_base[j],1];
        mat_aux[i,j]:= aux;
      end
    else
      begin
        aux:= mat_aux[2,j] + maq13[seq_base[j],1];
        mat_aux[i,j]:= aux;
      end;
  end;
if i = 14 then
  begin
    if j = 1 then
      aux := maq14[seq_base[j],2]
    else
      aux:= aux + maq14[seq_base[j],2];
    if aux > mat_aux[2,j] then
      begin
        aux:= aux +  maq14[seq_base[j],1];
        mat_aux[i,j]:= aux;
      end
    else
      begin
        aux:= mat_aux[2,j] + maq14[seq_base[j],1];
        mat_aux[i,j]:= aux;
      end;
  end;
if i = 15 then
  begin
    if j = 1 then
      aux := maq15[seq_base[j],2]
    else
      aux:= aux + maq15[seq_base[j],2];
    if aux > mat_aux[2,j] then
      begin
        aux:= aux +  maq15[seq_base[j],1];
```

```
          mat_aux[i,j]:= aux;
        end
      else
        begin
          aux:= mat_aux[2,j] + maq15[seq_base[j],1];
          mat_aux[i,j]:= aux;
        end;
  end;
if i = 16 then
  begin
    if j = 1 then
      aux := maq16[seq_base[j],2]
    else
      aux:= aux + maq16[seq_base[j],2];
    if aux > mat_aux[2,j] then
      begin
        aux:= aux +  maq16[seq_base[j],1];
        mat_aux[i,j]:= aux;
      end
    else
      begin
        aux:= mat_aux[2,j] + maq16[seq_base[j],1];
        mat_aux[i,j]:= aux;
      end;
  end;
if i = 17 then
  begin
    if j = 1 then
      aux := maq17[seq_base[j],2]
    else
      aux:= aux + maq17[seq_base[j],2];
    if aux > mat_aux[2,j] then
      begin
        aux:= aux +  maq17[seq_base[j],1];
        mat_aux[i,j]:= aux;
      end
    else
      begin
        aux:= mat_aux[2,j] + maq17[seq_base[j],1];
        mat_aux[i,j]:= aux;
      end;
  end;
if i = 18 then
  begin
    if j = 1 then
      aux := maq18[seq_base[j],2]
    else
      aux:= aux + maq18[seq_base[j],2];
    if aux > mat_aux[2,j] then
      begin
```

```
          aux:= aux +  maq18[seq_base[j],1];
          mat_aux[i,j]:= aux;
        end
      else
        begin
          aux:= mat_aux[2,j] + maq18[seq_base[j],1];
          mat_aux[i,j]:= aux;
        end;
  end;
if i = 19 then
  begin
    if j = 1 then
      aux := maq19[seq_base[j],2]
    else
      aux:= aux + maq19[seq_base[j],2];
    if aux > mat_aux[2,j] then
      begin
        aux:= aux +  maq19[seq_base[j],1];
        mat_aux[i,j]:= aux;
      end
    else
      begin
        aux:= mat_aux[2,j] + maq19[seq_base[j],1];
        mat_aux[i,j]:= aux;
      end;
  end;
if i = 20 then
  begin
    if j = 1 then
      aux := maq20[seq_base[j],2]
    else
      aux:= aux + maq20[seq_base[j],2];
    if aux > mat_aux[2,j] then
      begin
        aux:= aux +  maq20[seq_base[j],1];
        mat_aux[i,j]:= aux;
      end
    else
      begin
        aux:= mat_aux[2,j] + maq20[seq_base[j],1];
        mat_aux[i,j]:= aux;
      end;
  end;
if i = 21 then
  begin
    if j = 1 then
      aux := maq21[seq_base[j],2]
    else
      aux:= aux + maq21[seq_base[j],2];
    if aux > mat_aux[2,j] then
```

```
        begin
          aux:= aux +  maq21[seq_base[j],1];
          mat_aux[i,j]:= aux;
        end
      else
        begin
         aux:= mat_aux[2,j] + maq21[seq_base[j],1];
         mat_aux[i,j]:= aux;
        end;
  end;
if i = 22 then
  begin
    if j = 1 then
      aux := maq22[seq_base[j],2]
    else
      aux:= aux + maq22[seq_base[j],2];
    if aux > mat_aux[2,j] then
      begin
        aux:= aux +  maq22[seq_base[j],1];
        mat_aux[i,j]:= aux;
      end
    else
      begin
        aux:= mat_aux[2,j] + maq22[seq_base[j],1];
        mat_aux[i,j]:= aux;
      end;
  end;
if i = 23 then
  begin
    if j = 1 then
      aux := maq23[seq_base[j],2]
    else
      aux:= aux + maq23[seq_base[j],2];
    if aux > mat_aux[2,j] then
      begin
        aux:= aux +  maq23[seq_base[j],1];
        mat_aux[i,j]:= aux;
      end
    else
      begin
        aux:= mat_aux[2,j] + maq23[seq_base[j],1];
        mat_aux[i,j]:= aux;
      end;
  end;
if i = 24 then
  begin
    if j = 1 then
      aux := maq24[seq_base[j],2]
    else
      aux:= aux + maq24[seq_base[j],2];
```

```
            if aux > mat_aux[2,j] then
              begin
                aux:= aux +  maq24[seq_base[j],1];
                mat_aux[i,j]:= aux;
              end
            else
              begin
                aux:= mat_aux[2,j] + maq24[seq_base[j],1];
                mat_aux[i,j]:= aux;
              end;
          end;
        if i = 25 then
          begin
            if j = 1 then
              aux := maq25[seq_base[j],2]
            else
              aux:= aux + maq25[seq_base[j],2];
            if aux > mat_aux[2,j] then
              begin
                aux:= aux +  maq25[seq_base[j],1];
                mat_aux[i,j]:= aux;
              end
            else
              begin
                aux:= mat_aux[2,j] + maq25[seq_base[j],1];
                mat_aux[i,j]:= aux;
              end;
          end;
      end;
   end;
  calculo_makespan := aux;
end;

                                                        //CB
procedure TForm1.Button1Click(Sender: TObject);
var
  s,i,p: integer;
  tempo_inicial, tempo_final, tempo_total : integer;
  tempo_inicial_100, tempo_final_100, tempo_final_total : integer;
  str: string;
begin
 infinito := 999999;
 arq:='';
 arqtempo:= '';
 if OpenDialog1.Execute then
    arq:= OpenDialog1.FileName;
 if arq <> '' then
    begin
      AssignFile(arquivo, arq);
      Reset(arquivo);
```

```
      if OpenDialog1.Execute then
         arqtempo:= OpenDialog1.FileName;
      if arqtempo <> '' then
        begin
          AssignFile(arquivotempo, arqtempo);
          Reset(arquivotempo);
          Append(arquivotempo);
          Readln(arquivo, str);
          numdemaquinas := strtoint(str);
          Readln(arquivo, str);
          numtarefas := strtoint(str);
          Readln(arquivo, str);
          quantprob := strtoint(str);
          Readln(arquivo, str);
          Writeln(Arquivotempo,'algoritmo CB');
          argumentos:= 2;
          tempo_inicial_100:= GetTickCount;
          for s:=1 to quantprob do
            begin
              carregamatriz();
              fase1();
              fase2();
              fase3();
              Writeln(Arquivotempo,val_makespan);
            end;
          tempo_final_100:= GetTickCount;
          tempo_final_total:= tempo_final_100 - tempo_inicial_100;
          Writeln(Arquivotempo,'Tempo final algoritmo CB :'+ inttostr(tempo_final_total));
          Closefile(arquivo);
          Closefile(arquivotempo);
        end;
    end;
end;
                                                    //RZ1
procedure TForm1.Button2Click(Sender: TObject);
var
  s,i,p: integer;
  tempo_inicial, tempo_final, tempo_total : integer;
  tempo_inicial_100, tempo_final_100, tempo_final_total : integer;
  str: string;
begin
 infinito := 999999;
 arq:='';
 arqtempo:= '';
 if form1.OpenDialog1.Execute then
    arq:= form1.OpenDialog1.FileName;
 if arq <> '' then
    begin
      AssignFile(arquivo, arq);
      Reset(arquivo);
```

```
      if form1.OpenDialog1.Execute then
         arqtempo:= form1.OpenDialog1.FileName;
      if arqtempo <> '' then
        begin
          AssignFile(arquivotempo, arqtempo);
          Reset(arquivotempo);
          Append(arquivotempo);
          Readln(arquivo, str);
          numdemaquinas := strtoint(str);
          Readln(arquivo, str);
          numtarefas := strtoint(str);
          Readln(arquivo, str);
          quantprob := strtoint(str);
          Readln(arquivo, str);
          Writeln(Arquivotempo,'algoritmo RZ1');
          argumentos:= 2;
          tempo_inicial_100:= GetTickCount;
          for s:=1 to quantprob do
            begin
              unit1.Form1.carregamatriz();
              Unit2.Form2.fase1RZ1();
              unit2.Form2.fase2RZ();
              Writeln(Arquivotempo,makespan_RZ1);
            end;
          tempo_final_100:= GetTickCount;
          tempo_final_total:= tempo_final_100 -tempo_inicial_100;
          Writeln(Arquivotempo,'Tempo final algoritmo RZ1 :'+ inttostr(tempo_final_total));
          Closefile(arquivo);
          Closefile(arquivotempo);
        end;
    end;
end;
                                                    //RZ2
procedure TForm1.Button3Click(Sender: TObject);
var
  s,i,p: integer;
  tempo_inicial, tempo_final, tempo_total : integer;
  tempo_inicial_100, tempo_final_100, tempo_final_total : integer;
  str: string;
begin
 infinito := 999999;
 arq:='';
 arqtempo:= '';
 if form1.OpenDialog1.Execute then
    arq:= form1.OpenDialog1.FileName;
 if arq <> '' then
    begin
      AssignFile(arquivo, arq);
      Reset(arquivo);           //leitura da maquinas
      if form1.OpenDialog1.Execute then
```

```
          arqtempo:= form1.OpenDialog1.FileName;
        if arqtempo <> '' then
          begin
            AssignFile(arquivotempo, arqtempo);
            Reset(arquivotempo);
            Append(arquivotempo);
            Readln(arquivo, str);
            numdemaquinas := strtoint(str);
            Readln(arquivo, str);
            numtarefas := strtoint(str);
            Readln(arquivo, str);
            quantprob := strtoint(str);
            Readln(arquivo, str);
            Writeln(Arquivotempo,'algoritmo RZ2');
            argumentos:= 2;
            tempo_inicial_100:= GetTickCount;
            for s:=1 to quantprob do
              begin
                unit1.Form1.carregamatriz();
                unit3.Form3.fase1RZ2();
                unit2.Form2.fase2RZ;
                Writeln(Arquivotempo,makespan_RZ2);
              end;
            tempo_final_100:= GetTickCount;
            tempo_final_total:= tempo_final_100 -tempo_inicial_100;
            Writeln(Arquivotempo,'Tempo final algoritmo RZ2 :'+ inttostr(tempo_final_total));
            Closefile(arquivo);
            Closefile(arquivotempo);
          end;
      end;
end;
                                                        //CB, RZ3, BMc e BMm
procedure TForm1.Button5Click(Sender: TObject);
begin
       form5.show;
end;

procedure TForm1.Button6Click(Sender: TObject);
var
   s,i,p: integer;
   tempo_inicial, tempo_final, tempo_total : integer;
   tempo_inicial_100, tempo_final_100, tempo_final_total : integer;
   str: string;
begin
  infinito := 999999;
  arq:='';
  arqtempo:= '';
  if OpenDialog1.Execute then
     arq:= OpenDialog1.FileName;
  if arq <> '' then
```

```
begin
  AssignFile(arquivo, arq);
  Reset(arquivo);
  if OpenDialog1.Execute then
     arqtempo:= OpenDialog1.FileName;
  if arqtempo <> '' then
    begin
      AssignFile(arquivotempo, arqtempo);
      Reset(arquivotempo);
      Append(arquivotempo);

                                                                  //executa o rz3
      reset(arquivo);
      Readln(arquivo, str);
      numdemaquinas := strtoint(str);
      Readln(arquivo, str);
      numtarefas := strtoint(str);
      Readln(arquivo, str);
      quantprob := strtoint(str);
      Readln(arquivo, str);
      Writeln(Arquivotempo,'algoritmo RZ3');
      argumentos:= 2;
      tempo_inicial_100:= GetTickCount;
      for s:=1 to quantprob do
        begin
          unit1.Form1.carregamatriz();
          //executa o rz1
          Unit2.Form2.fase1RZ1();
          unit2.Form2.fase2RZ();
          //executa o rz2
          unit3.Form3.fase1RZ2();
          unit2.Form2.fase2RZ();
          //executa o rz3
          Unit4.Form4.fase1RZ3();
          unit2.Form2.fase2RZ();
          Writeln(Arquivotempo,makespan_RZ3);
        end;
      tempo_final_100:= GetTickCount;
      tempo_final_total:= tempo_final_100 -tempo_inicial_100;
      Writeln(Arquivotempo,'Tempo final algoritmo RZ3 :'+ inttostr(tempo_final_total));
      Closefile(arquivo);
                                                                  //executa o BM2-BMc
      reset(arquivo);
      Readln(arquivo, str);
      numdemaquinas := strtoint(str);
      Readln(arquivo, str);
      numtarefas := strtoint(str);
      Readln(arquivo, str);
      quantprob := strtoint(str);
      Readln(arquivo, str);
```

```
Writeln(Arquivotempo,'algoritmo BM2');
argumentos:= 2;
tempo_inicial_100:= GetTickCount;
for s:=1 to quantprob do
  begin
    unit1.Form1.carregamatriz();
    unit6.Form6.gera_matriz();
    unit7.Form7.ordenacao2;
    unit5.Form5.insercao_tarefas;
    Writeln(Arquivotempo,valor_makespan);
  end;
tempo_final_100:= GetTickCount;
tempo_final_total:= tempo_final_100 - tempo_inicial_100;
Writeln(Arquivotempo,'Tempo final algoritmo BM2 :'+ inttostr(tempo_final_total));
Closefile(arquivo); //Fecha o arquivo texto
                                                    //executa BMm
reset(arquivo);
Readln(arquivo, str);
numdemaquinas := strtoint(str);
Readln(arquivo, str);
numtarefas := strtoint(str);
Readln(arquivo, str);  /
quantprob := strtoint(str);
Readln(arquivo, str);
Writeln(Arquivotempo,'algoritmo Bmm');
argumentos:= 2;
tempo_inicial_100:= GetTickCount;
for s:=1 to quantprob do
  begin
    unit1.Form1.carregamatriz();
    unit6.Form6.gera_matriz();
    unit7.Form7.ordenacao2;
    unit5.Form5.insercao_tarefas;
    unit2.Form2.fase2RZ;
    Writeln(Arquivotempo,makespan_Bmm);
  end;
tempo_final_100:= GetTickCount;
tempo_final_total:= tempo_final_100 - tempo_inicial_100;
Writeln(Arquivotempo,'Tempo final algoritmo Bmm :'+ inttostr(tempo_final_total));
Closefile(arquivo);
                                                    //executa o CB
reset(arquivo);
Readln(arquivo, str);
numdemaquinas := strtoint(str);
Readln(arquivo, str);
numtarefas := strtoint(str);
Readln(arquivo, str);
quantprob := strtoint(str);
Readln(arquivo, str);
Writeln(Arquivotempo,'algoritmo CB');
```

```
            argumentos:= 2;
            tempo_inicial_100:= GetTickCount;
            for s:=1 to quantprob do
              begin
                carregamatriz();
                fase1();
                fase2();
                fase3();
                Writeln(Arquivotempo,val_makespan);
              end;
            tempo_final_100:= GetTickCount;
            tempo_final_total:= tempo_final_100 - tempo_inicial_100;
            Writeln(Arquivotempo,'Tempo final algoritmo CB :'+ inttostr(tempo_final_total));
            Closefile(arquivo);
            Closefile(arquivotempo);
          end;
      end;
end;
                                                        //RZ3
procedure TForm1.BitBtn1Click(Sender: TObject);
begin
 application.Terminate;
end;

procedure TForm1.Button4Click(Sender: TObject);
var
   s,i,p: integer;
   tempo_inicial, tempo_final, tempo_total : integer;
   tempo_inicial_100, tempo_final_100, tempo_final_total : integer;
   str: string;
begin
  infinito := 999999;
  arq:='';
  arqtempo:= '';
  if form1.OpenDialog1.Execute then
     arq:= form1.OpenDialog1.FileName;
  if arq <> '' then
     begin
       AssignFile(arquivo, arq);
       Reset(arquivo);
       if form1.OpenDialog1.Execute then
          arqtempo:= form1.OpenDialog1.FileName;
       if arqtempo <> '' then
         begin
           AssignFile(arquivotempo, arqtempo);
           Reset(arquivotempo);
           Append(arquivotempo);
           Readln(arquivo, str);
           numdemaquinas := strtoint(str);
           Readln(arquivo, str);
```

```
            numtarefas := strtoint(str);
            Readln(arquivo, str);
            quantprob := strtoint(str);
            Readln(arquivo, str);
            Writeln(Arquivotempo,'algoritmo RZ3');
            argumentos:= 2;
            tempo_inicial_100:= GetTickCount;
            for s:=1 to quantprob do
              begin
                unit1.Form1.carregamatriz();
                                                                        //executa o rz1
                Unit2.Form2.fase1RZ1();
                unit2.Form2.fase2RZ();
                                                                         //executa o rz2
                unit3.Form3.fase1RZ2();
                unit2.Form2.fase2RZ;
                                                                         //executa o rz3
                Unit4.Form4.fase1RZ3();
                unit2.Form2.fase2RZ;
                Writeln(Arquivotempo,makespan_RZ3);
              end;
            tempo_final_100:= GetTickCount;
            tempo_final_total:= tempo_final_100 -tempo_inicial_100;
            Writeln(Arquivotempo,'Tempo final algoritmo RZ3 :'+ inttostr(tempo_final_total));
            Closefile(arquivo);
            Closefile(arquivotempo);
          end;
      end;
end;
                                                    //BMm
procedure TForm1.Button7Click(Sender: TObject);
var
  s,i,p: integer;
  tempo_inicial, tempo_final, tempo_total : integer;
  tempo_inicial_100, tempo_final_100, tempo_final_total : integer;
  str: string;
begin

 infinito := 999999;
 arq:='';
 arqtempo:= '';
 if form1.OpenDialog1.Execute then
    arq:= form1.OpenDialog1.FileName;
 if arq <> '' then
    begin
      AssignFile(arquivo, arq);
      Reset(arquivo);
      if form1.OpenDialog1.Execute then
         arqtempo:= form1.OpenDialog1.FileName;
      if arqtempo <> '' then
```

```
        begin
          AssignFile(arquivotempo, arqtempo);
          Reset(arquivotempo);
          Append(arquivotempo);
          Readln(arquivo, str);
          numdemaquinas := strtoint(str);
          Readln(arquivo, str);
          numtarefas := strtoint(str);
          Readln(arquivo, str);
          quantprob := strtoint(str);
          Readln(arquivo, str);
          Writeln(Arquivotempo,'algoritmo Bmm');
          argumentos:= 2;
          tempo_inicial_100:= GetTickCount;
          for s:=1 to quantprob do
            begin
              unit1.Form1.carregamatriz();
              unit6.Form6.gera_matriz();
              unit7.Form7.ordenacao2;
              unit5.Form5.insercao_tarefas;
              unit2.Form2.fase2RZ;
              Writeln(Arquivotempo,makespan_Bmm);
            end;
          tempo_final_100:= GetTickCount;
          tempo_final_total:= tempo_final_100 - tempo_inicial_100;
          Writeln(Arquivotempo,'Tempo final algoritmo Bmm :'+ inttostr(tempo_final_total));
          Closefile(arquivo);
          Closefile(arquivotempo);
        end;
    end;
end;
end.
                                                    //BM
unit Unit8;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs;

type
  TForm8 = class(TForm)
  private
    { Private declarations }
  public
    { Public declarations }
    procedure bm1();
    procedure bm2();
    procedure bm3();
```

```
  end;

var
  Form8: TForm8;

implementation
uses
   unit7, unit1, unit5, unit6;
{$R *.dfm}
                                                        //BM1
procedure Tform8.bm1();

var
   s,i,p: integer;
   tempo_inicial, tempo_final, tempo_total : integer;
   tempo_inicial_100, tempo_final_100, tempo_final_total : integer;
   str: string;
begin

  infinito := 999999;
  arq:='';
  arqtempo:= '';
  if form1.OpenDialog1.Execute then
     arq:= form1.OpenDialog1.FileName;
  if arq <> '' then
     begin
       AssignFile(arquivo, arq);
       Reset(arquivo);
       if form1.OpenDialog1.Execute then
          arqtempo:= form1.OpenDialog1.FileName;
       if arqtempo <> '' then
         begin
            AssignFile(arquivotempo, arqtempo);
            Reset(arquivotempo);
            Append(arquivotempo);
            Readln(arquivo, str);
            numdemaquinas := strtoint(str);
            Readln(arquivo, str);
            numtarefas := strtoint(str);
            Readln(arquivo, str);
            quantprob := strtoint(str);
            Readln(arquivo, str);
            Writeln(Arquivotempo,'algoritmo BM1');
            argumentos:= 2;
            tempo_inicial_100:= GetTickCount;
            for s:=1 to quantprob do
              begin
                 unit1.Form1.carregamatriz();
                 unit6.Form6.gera_matriz();
                 unit7.Form7.ordenacao1;
```

```
                unit5.Form5.insercao_tarefas;
                Writeln(Arquivotempo,valor_makespan);
              end;
            tempo_final_100:= GetTickCount;
            tempo_final_total:= tempo_final_100 - tempo_inicial_100;
            Writeln(Arquivotempo,'Tempo final algoritmo BM1 :'+ inttostr(tempo_final_total));
            Closefile(arquivo);
            Closefile(arquivotempo);
          end;
      end;
end;
                                                    //BM2
procedure Tform8.bm2();
var
   s,i,p: integer;
   tempo_inicial, tempo_final, tempo_total : integer;
   tempo_inicial_100, tempo_final_100, tempo_final_total : integer;
   str: string;
begin

  infinito := 999999;
  arq:='';
  arqtempo:= '';
  if form1.OpenDialog1.Execute then
     arq:= form1.OpenDialog1.FileName;
  if arq <> '' then
     begin
       AssignFile(arquivo, arq);
       Reset(arquivo);
       if form1.OpenDialog1.Execute then
          arqtempo:= form1.OpenDialog1.FileName;
       if arqtempo <> '' then
         begin
           AssignFile(arquivotempo, arqtempo);
           Reset(arquivotempo);
           Append(arquivotempo);
           Readln(arquivo, str);
           numdemaquinas := strtoint(str);
           Readln(arquivo, str);
           numtarefas := strtoint(str);
           Readln(arquivo, str);
           quantprob := strtoint(str);
           Readln(arquivo, str);
           Writeln(Arquivotempo,'algoritmo BM2');
           argumentos:= 2;
           tempo_inicial_100:= GetTickCount;
           for s:=1 to quantprob do
             begin
               unit1.Form1.carregamatriz();
               unit6.Form6.gera_matriz();
```

```
                unit7.Form7.ordenacao2;
                unit5.Form5.insercao_tarefas;
                Writeln(Arquivotempo,valor_makespan);
              end;
            tempo_final_100:= GetTickCount;
            tempo_final_total:= tempo_final_100 - tempo_inicial_100;
            Writeln(Arquivotempo,'Tempo final algoritmo BM2 :'+ inttostr(tempo_final_total));
            Closefile(arquivo);
            Closefile(arquivotempo);
          end;
      end;
end;
                                                        //BM3
procedure tform8.bm3();
var
   s,i,p: integer;
   tempo_inicial, tempo_final, tempo_total : integer;
   tempo_inicial_100, tempo_final_100, tempo_final_total : integer;
   str: string;
begin

  infinito := 999999;
  arq:='';
  arqtempo:= '';
  if form1.OpenDialog1.Execute then
     arq:= form1.OpenDialog1.FileName;
  if arq <> '' then
     begin
       AssignFile(arquivo, arq);
       Reset(arquivo);
       if form1.OpenDialog1.Execute then
          arqtempo:= form1.OpenDialog1.FileName;
       if arqtempo <> '' then
         begin
           AssignFile(arquivotempo, arqtempo);
           Reset(arquivotempo);
           Append(arquivotempo);
           Readln(arquivo, str);
           numdemaquinas := strtoint(str);
           Readln(arquivo, str);
           numtarefas := strtoint(str);
           Readln(arquivo, str);
           quantprob := strtoint(str);
           Readln(arquivo, str);
           Writeln(Arquivotempo,
           argumentos:= 2;
           tempo_inicial_100:= GetTickCount;
           for s:=1 to quantprob do
             begin
               unit1.Form1.carregamatriz();
```

```
                unit6.Form6.gera_matriz();
                unit7.Form7.ordenacao3;
                unit5.Form5.insercao_tarefas;
                Writeln(Arquivotempo,valor_makespan);
              end;
            tempo_final_100:= GetTickCount;
            tempo_final_total:= tempo_final_100 - tempo_inicial_100;
            Writeln(Arquivotempo,'Tempo final algoritmo BM3 :'+ inttostr(tempo_final_total));
            Closefile(arquivo);
            Closefile(arquivotempo);
          end;
      end;
end;
          end.
```