Emerson Carlos Colin

“*BEAM SEARCH* E INSERÇÃO DE OCIOSIDADE NO PROBLEMA DE
PROGRAMAÇÃO DE UMA MÁQUINA EM AMBIENTE DO TIPO JIT”

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

Área de concentração: Engenharia de Produção.

Orientador: Tamio Shimizu.

São Paulo
1997

...

O conhecimento de Deus, única verdadeira sabedoria

Assim fala o SENHOR:

Que o sábio não se glorie de sua sabedoria!

Que o valente não se glorie de sua força!

Que o rico não se glorie de sua riqueza!

Se alguém quiser gloriar-se, glorie-se disto:

de ser bastante arguto para me conhecer, a mim, o

SENHOR, que exerço a solidariedade

o direito e a justiça sobre a terra.

Sim, é isso que me agrada – oráculo do SENHOR.

...

Livro de Jeremias capítulo 9, versículos 22 e 23.

AGRADECIMENTOS

Inicialmente gostaria de agradecer ao prof. Tamio Shimizu pela orientação segura e pela disponibilidade constante, permitindo assim que o trabalho fosse finalizado em um prazo relativamente pequeno.

Além do professor Shimizu, os outros dois membros da banca, prof. Flávio Cipparrone e prof. Oduvaldo Vendrameto, colaboraram imensamente para a melhoria de versões preliminares do trabalho. Se o trabalho não ficou muito didático ou claro, com toda certeza, as versões preliminares estavam menos ainda.

Outras pessoas que colaboraram de maneira indireta, através de discussões importantes para o trabalho são o prof. Miguel C. Santoro e os alunos de pós-graduação do DEP, em especial Carlos E. C. Moreira, Davi N. Nakano, Felipe V. de Araújo, Helvécio L. Reis, João B. Turrioni e Roberto da C. Quinino.

O trabalho teve apoio financeiro fornecido pela CAPES – Fundação Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – por intermédio de uma bolsa de mestrado institucional do Departamento de Engenharia de Produção da Escola Politécnica da USP.

Finalizando, gostaria de mencionar o apoio recebido de meus pais e de minha noiva nos momentos de trabalho em detrimento do convívio familiar e social.

SUMÁRIO

Lista de figuras	vi
Lista de tabelas	vii
Lista de abreviaturas	viii
Lista de símbolos	ix
Resumo	xiii
Abstract	xiv
1 Introdução.....	1
1.1 Exemplo introdutório à programação e seqüenciação.....	1
1.2 Relação entre decisões de programação da produção e decisões de outros níveis.....	4
1.3 Ambiente do tipo JIT (<i>Just-In-Time</i>) e o procedimento de programação da produção	5
1.3.1 Procedimentos de programação da produção	6
1.3.2 Procedimento de programação da produção no ambiente do tipo JIT.....	11
1.4 Procedimentos ótimos X procedimentos heurísticos.....	13
1.5 Organização da dissertação	14
2 Definição, importância e revisão bibliográfica do problema	15
2.1 Funções matemáticas mais utilizadas na dissertação	16
2.2 Definição do problema	18
2.3 Importância do problema.....	20
2.3.1 Importância do problema de uma máquina	20
2.3.2 Importância da inserção de ociosidade e das penalidades distintas para adiantamento e para atraso	22
2.4 Algumas heurísticas conhecidas para a seqüenciação do problema.....	23
2.4.1 Regra de despacho	23
2.4.2 Função de prioridade	26
2.4.3 Troca de pares adjacentes	28
2.4.4 <i>Beam search</i>	29
2.5 Principais resultados do problema com datas múltiplas e com penalidades individualizadas para adiantamento e para atraso	31
3 Beam search	36
3.1 <i>Beam search</i>	36
3.2 <i>Beam search</i> filtrado (BSF).....	39
3.3 Função de prioridade EXP-ET	41
3.4 Exemplo numérico	43
3.5 Revisão bibliográfica.....	46
4 Inserção de ociosidade	52
4.1 Inserção de ociosidade utilizando-se a função de prioridade EXP-ET	53
4.2 Algoritmo de inserção de ociosidade	56
4.3 Implementação do algoritmo de inserção de ociosidade.....	59
4.4 Exemplo numérico	61
5 Estudo computacional.....	63
5.1 Variáveis controladas	63
5.2 Experimento realizado.....	66

5.2.1 Instâncias testadas.....	66
5.2.2 Resolução das instâncias	67
5.3 Interpretação dos resultados	69
5.3.1 Tempos de processamento computacional	69
5.3.2 Comparação com relação aos valores das funções objetivo	71
6 Conclusões.....	74
6.1 Principais resultados obtidos no estudo computacional	74
6.2 Considerações adicionais	74
6.3 Contribuições da dissertação	75
6.4 Algumas extensões possíveis do tema estudado	76
Anexo A: Classificação e redutibilidade de problemas de programação e seqüenciação	78
A1 Ambiente da máquina	78
A2 Características da ordem.....	80
A3 Critério de otimização	81
A4 Exemplos da utilização da notação.....	83
A5 Redutibilidade entre problemas de programação e seqüenciação	84
Anexo B: Complexidade computacional	86
B1 Classificação de problemas algorítmicos.....	88
B2 Problemas da classe P	88
B3 Problemas da classe NP	89
Anexo C: Resultados numéricos	91
Anexo D: Demonstração do teorema.....	101
Referências bibliográficas	103
Apêndice 1: Programas computacionais.....	1
Apêndice 2: Exemplos de instâncias.....	31

LISTA DE FIGURAS

Figura 1: exemplo de fluxo de produção nos procedimentos que puxam e que empurram a produção	9
Figura 2: hierarquia entre algumas funções objetivo	17
Figura 3: primeiro nível da árvore de busca do beam search.....	37
Figura 4: segundo nível da árvore de busca do beam search	37
Figura 5: árvore de busca utilizando o beam search	38
Figura 6: primeiro nível da árvore de busca do beam search filtrado	40
Figura 7: segundo nível da árvore de busca do beam search filtrado	40
Figura 8: árvore de busca utilizando o beam search filtrado	40
Figura 9: significado gráfico da função de prioridade EXP-ET ($k=5$).....	42
Figura 10: significado gráfico dos blocos de um programa.....	57
Figura 11: pseudocódigo para inserção de ociosidade.....	60
Figura 12: variáveis testadas no estudo computacional	64
Figura 13: relação entre tempo de solução, larguras da busca e do filtro no BSF sem inserção de ociosidade	69
Figura 14: tempos de solução do BSF com inserção de ociosidade para $b=1$ e $b=3$	70
Figura 15: tempos de solução do BSF com inserção de ociosidade para $b=5$ e $b=8$	70
Figura 16: grafo de redutibilidade entre problemas de programação e seqüenciação (adaptado de LAWLER <i>et al.</i> , 1993).....	85
Figura 17: ordem de crescimento assintótico de algumas funções com escala logarítmica e escala normal ($1 \leq n \leq 25$)	87
Figura 18: ordem de crescimento assintótico de algumas funções com escala logarítmica e escala normal ($1 \leq n \leq 10$)	87
Figura 19: influência de b e f no desempenho da busca - grupo 1	98
Figura 20: média geral da influência de b e f no desempenho da busca - grupo 1	98
Figura 21: influência de b e f no desempenho da busca - grupo 2	99
Figura 22: média geral da influência de b e f no desempenho da busca - grupo 2	99
Figura 23: influência de b e f no desempenho da busca - grupo 3	100
Figura 24: média geral da influência de b e f no desempenho da busca - grupo 3	100

LISTA DE TABELAS

Tabela 1: carteira de pedidos para se programar o centro de usinagem.....	1
Tabela 2: carga máquina do centro de usinagem	2
Tabela 3: dados do centro de usinagem já adaptados para PS	3
Tabela 4: procedimentos de programação que empurram e que puxam a produção (adaptada de MORTON; PENTICO, 1993, p.540)	7
Tabela 5: exemplos da utilização de regras de despacho na programação do centro de usinagem	25
Tabela 6: exemplo da utilização de uma função de prioridade na programação do centro de usinagem	27
Tabela 7: exemplo da utilização da troca de pares adjacentes na programação do centro de usinagem - semente por EXP-ET e por WLPT.....	28
Tabela 8: seqüências analisadas no primeiro nível do beam search - exemplo do centro de usinagem	30
Tabela 9: seqüências analisadas no beam search - exemplo do centro de usinagem.....	30
Tabela 10: quadro comparativo dos resultados dos problemas com custos individuais de atraso e de adiantamento	35
Tabela 11: dados da resolução numérica do BSF	43
Tabela 12: seqüência EXP-ET, horários de programação e prioridades máximas	44
Tabela 13: resultados obtidos pela EXP-ET no exemplo numérico	44
Tabela 14: resultados obtidos pelo BSF no nível 1	45
Tabela 15: passos intermediários e resultados da solução pelo BSF	45
Tabela 16: quadro comparativo dos resultados da aplicação do beam search a problemas de programação e seqüenciação.....	50
Tabela 17: dados do exemplo para inserção de ociosidade.....	61
Tabela 18: cálculo do exemplo sem inserção de ociosidade.....	62
Tabela 19: cálculo do exemplo com inserção de ociosidade	62
Tabela 20 : programas computacionais utilizados e respectivas funções	68
Tabela 21: resultados do estudo computacional sem inserção de ociosidade - grupo 1	93
Tabela 22: resultados do estudo computacional com inserção de ociosidade - grupo 2	94
Tabela 23: resultados do estudo computacional com inserção de ociosidade - grupo 3	95
Tabela 24: BSF com ociosidade grupo 2÷BSF com ociosidade grupo 3.....	96
Tabela 25: BSF sem ociosidade grupo 1÷BSF com ociosidade grupo 3	97

LISTA DE ABREVIATURAS

BS	<i>Beam Search</i> ;
BSF	<i>Beam Search</i> Filtrado;
CV	Coefficiente de Variação;
desp	Despacho;
EDD	(<i>Earliest Due Date</i>) menor data de entrega;
EXP-ET	(<i>Exponential Earliness Tardiness</i>) adiantamento atraso exponenciais;
FIFO	(<i>First In First Out</i>) o primeiro a entrar é o primeiro a sair;
FMS	(<i>Flexible Manufacturing System</i>) sistema flexível de manufatura;
IDLE	(<i>Idle Time Rule</i>) regra do tempo ocioso;
JIT	<i>Just-in-Time</i> ;
LIFO	(<i>Last In First Out</i>) o último a entrar é o primeiro a sair;
LPT	(<i>Longest Processing Time</i>) maior tempo de processamento;
MST	(<i>Minimum Slack Time</i>) menor tempo de folga dinâmica;
NP	Classe dos problemas não determinísticos polinomiais;
OPT	<i>Optimized Production Technology</i> ;
P	Classe dos problemas determinísticos polinomiais;
prior	Prioridade;
PS	Programação e Seqüenciação;
P&D	Pesquisa e Desenvolvimento;
SPT	(<i>Shortest Processing Time</i>) menor tempo de processamento;
TPM	(<i>Total Productive Maintenance</i>) manutenção produtiva total;
TQC	(<i>Total Quality Control</i>) controle da qualidade total;
WLPT	(<i>Weighted Longest Processing Time</i>) maior tempo de processamento ponderado;
WSPT	(<i>Weighted Shortest Processing Time</i>) menor tempo de processamento ponderado.

LISTA DE SÍMBOLOS

Apresenta-se abaixo a notação geral utilizada no decorrer da dissertação. Além disso, quando um termo estiver aparecendo pela primeira vez no texto, ele será explicado.

$\propto$ =reduz-se a;

$\circ$ =símbolo de vazio;

$\sim U[\cdot]$ =distribuição de probabilidade uniforme;

$|X|$ =cardinalidade do conjunto X ;

$|x|$ =módulo da variável x ;

$\{X\}$ =conjunto X ;

$\langle X \rangle$ =seqüência ou conjunto ordenado X ;

$U \rightarrow V$ =o problema de programação U reduz-se ao problema de programação V ;

α =ambiente da máquina;

β =características da ordem;

γ =critério de otimização;

Δt =intervalo de tempo;

ν =nível da árvore de busca;

π =programa completo;

$\pi_j(\sigma)$ =programa parcial, computado até as primeiras j ordens de uma seqüência σ ,

σ =seqüência completa;

$\sigma(j)$ = j -ésimo índice da seqüência σ . O índice pertence ao conjunto J ;

τ =fator de atraso médio;

ϕ =constante de proporcionalidade de adiantamento;

ψ =constante de proporcionalidade de atraso;

$\phi_j(t)$ =prioridade da ordem J_j no horário t ;

a_j =horário desejado de início da ordem J_j ;

$Adiantado(i)$ =conjunto de ordens pertencentes ao bloco i que possuem a propriedade de se for reduzido e_j , aumenta-se o adiantamento de J_j ;

$Atrasado(i)$ =conjunto de ordens pertencentes ao bloco i que possuem a propriedade de se for reduzido e_j , reduz-se o atraso de J_j ;

b =largura da busca em um *beam search*;

B_i =bloco i de um programa;

C_j =horário de término da ordem J_j ;

$C_{s_q^{v-1}}$ =horário de término do processamento da seqüência parcial fixa

(seqüência fixa, formada pelas ordens até o nível $v-1$) representada pelo nó pai q ;

d_j =data de entrega da ordem J_j ;

$E(\cdot)$ =esperança;

E_j =adiantamento da ordem J_j ;

e_j =horário efetivo de início da ordem J_j ;

F =*flow shop*;

f =largura do filtro em um *beam search* filtrado;

f_j =função de custo real não decrescente da ordem J_j ;

F_j =tempo de fluxo da ordem J_j ;

G_i =trecho i do grafo direcionado acíclico G ;

g_j =função de custo real da ordem J_j ;

$H(\cdot)$ =soma das penalidades de adiantamento;

h_j =penalidade da ordem J_j ;

I =intervalo;

$inst$ =instância;

J =conjunto de ordens;

J =*job shop*;

J_j =ordem j ($j=1,2,\dots,n$);

k =parâmetro de ajuste;

l =número de blocos em um programa;

l_j =folga da ordem J_j ;

L_j =diferença da ordem J_j . Pode ser atraso ou adiantamento;

M_i =máquina i ($i=1,2,\dots,m$);

m_j =número de operações de J_j ;

n =número de ordens;

N = $\{0,1,2,\dots\}$, conjunto dos números naturais;

O	=open job shop;
O_{ij}	=operação i da ordem J_j ;
$O(g)$	=ordem de g ;
P	=pilha;
P	=máquinas paralelas idênticas;
p_{ij}	=tempo de processamento na máquina M_i da ordem J_j ;
p_j	=tempo de processamento da ordem J_j ;
$Primeiro(i)$	=índice da primeira ordem do bloco i ;
$Prop(\cdot)$	=proporção;
q	=nó qualquer;
$\mathbf{Q}$	= $\{m/n: m, n \in \mathbf{Z} \text{ e } n \neq 0\}$, conjunto dos números irracionais;
Q_j	=lucro proporcionado pela ordem J_j ;
r_j	=horário de chegada da ordem J_j ;
R	=amplitude relativa das datas de entrega;
res	=(resource) recurso;
$res1$	=existe um recurso limitado;
R_h	=recurso limitado h ($h=1, 2, \dots, e$)
r_{hj}	=unidades do recurso limitado R_h utilizadas no processamento de J_j ;
R_q^ν	=conjunto das ordens disponíveis para serem processadas no nó q do nível ν da árvore de busca;
s_q	=seqüência fixa;
t	=horário qualquer;
t_j^*	=um determinado horário relacionado com a ordem J_j ;
$T(\cdot)$	=tempo de solução máximo de um algoritmo; tempo no pior caso;
T_j	=atraso da ordem J_j ;
u_j	=penalidade da ordem J_j ;
U_j	=penalidade unitária da ordem J_j ;
$Último(i)$	=índice da última ordem do bloco i ;
$Var(\cdot)$	=variância;
v_j	=penalidade da ordem J_j ;
W	=período de ociosidade utilizado;
W_i	=tempo de ociosidade forçado na máquina M_i ;

W_{ij} =tempo de ociosidade antes da operação O_{ij} ;

w_j =penalidade da ordem J_j ;

W_j =período de ociosidade precedendo a ordem J_j ;

$W(\cdot)$ =soma das penalidades de atraso;

x =variável qualquer;

y =variável qualquer;

$\mathbf{Z}$ = $\{\dots, -2, -1, 0, 1, 2, \dots\}$, conjunto dos números inteiros;

Z_α =fronteira α -eficiente num problema com dois critérios de otimização;

subscritos:

max =valor máximo;

med =valor médio;

min =valor mínimo;

RESUMO

Este trabalho apresenta procedimentos que podem ser utilizados na programação da produção em um ambiente JIT. Esses procedimentos deveriam ser utilizados em sistemas clássicos de programação, onde a utilização do sistema *kanban* é inviável. O caso estudado se baseia em uma única máquina, com datas de entrega múltiplas e com penalidades distintas de adiantamento e de atraso para cada ordem. O objetivo a ser alcançado é a minimização do custo total. Para isso, é utilizado um procedimento de busca denominado *beam search*, para gerar as seqüências, e um algoritmo de inserção de ociosidade, para definir os programas. O algoritmo utilizado é uma generalização do algoritmo de GAREY *et al.* (1988) onde as penalidades são distintas para adiantamento e para atraso. O procedimento e o algoritmo são testados em várias condições sendo comparados com regras de despacho e com a função EXP-ET. Quando a função EXP-ET é utilizada com a possibilidade de inserção de ociosidade, o período de ociosidade ótimo é determinado. Assume-se que a dificuldade de solução do problema é dependente de dois parâmetros clássicos: fator de atraso médio e amplitude relativa das datas de entrega. Testes empíricos comparativos são realizados através de simulação computacional, onde se mede o tempo de solução e o valor alcançado pela função objetivo. Os resultados indicam que o desempenho dos vários procedimentos testados é altamente dependente dos dois parâmetros, mostrando que para a escolha de um procedimento apropriado, deve-se primeiramente conhecer o valor dos parâmetros. São fornecidos os resultados encontrados e os códigos computacionais utilizados no estudo.

ABSTRACT

This work presents some procedures which can be used in the production scheduling problem in a JIT environment. These procedures may be used in classical production scheduling cases where the kanban system utilization is infeasible. The case studied is based on a single machine, with multiple due dates, and distinct earliness and tardiness penalties for each job. The objective function is to minimize total cost. A heuristic search procedure known as beam search is used to construct sequences of jobs, and an idleness insertion algorithm is used to obtain schedules. The used algorithm is a generalization of the GAREY et al. (1988) algorithm, where penalties are distinct for earliness and tardiness. The procedure and algorithm are tested in many conditions involving comparisons with dispatching rules and the EXP-ET function. When EXP-ET function is applied with possibility of idleness insertion, the optimal idleness period is provided. It was assumed that problem hardness is dependent on two classical parameters: average tardiness factor and relative range of due dates. Empirical comparative tests are conducted with computational simulation, where computational solution time and objective function value are evaluated. Results indicate that procedures performance is highly dependent on both parameters, showing that is necessary to know parameters values before choosing an appropriate procedure. The detailed results and computational code used in this study are also provided.

1 INTRODUÇÃO

Este primeiro capítulo apresentará um exemplo introdutório com o intuito de deixar mais claros os conceitos abordados posteriormente no desenvolvimento do texto. Além disso, apresenta como podem ser feitos os relacionamentos entre os diversos níveis de tomada de decisão de uma empresa, onde aparece a necessidade de se utilizar uma filosofia do tipo JIT (*Just-in-Time*), e a razão para se utilizar um método heurístico na programação da produção. Ele é finalizado mostrando como a dissertação está organizada.

1.1 EXEMPLO INTRODUTÓRIO À PROGRAMAÇÃO E SEQÜENCIAÇÃO

Problemas de programação e seqüenciação (PS) surgem em qualquer ambiente onde deseja-se definir horários, locais e atividades a serem realizadas. Para se caracterizar esse ambiente, neste subitem será estudado como, a partir de solicitações feitas pelos clientes, tempos de processamento de produtos e custos internos e externos à empresa, pode-se formular um problema típico de PS.

Uma empresa produtora de rodas automotivas especiais de magnésio possui um único centro de usinagem, que executa todas as operações das rodas fabricadas pela empresa. Há interesse em programar esse centro de usinagem pelo período de 24 horas, de modo que algum critério – definido posteriormente – seja otimizado. Por enquanto pode-se considerar a programação como a determinação dos horários de início e de término de atividades. Para o dia em que se deseja fazer a programação, existem os seguintes pedidos de clientes, conforme tabela abaixo:

Produto	Quantidade	Horário de entrega solicitado	Cliente
Roda A	200	8	Ferrari
Roda B	360	6	Chrysler
Roda C	100	9	Rolls-Royce
Roda D	800	4	Land-Rover
Roda E	100	21	Laborguini

Tabela 1: carteira de pedidos para se programar o centro de usinagem

Sabe-se que a produção horária dos produtos no centro de usinagem é a seguinte: Roda A= 34 peças/hora; Roda B= 120 peças/hora; Roda C= 50 peças/hora; Roda D= 129 peças/hora; Roda E= 59 peças/hora.

Com esses dados, consegue-se definir a quantidade de horas necessárias para o dia de produção. Adicionalmente às horas necessárias, deve-se incluir o tempo de preparação da máquina para cada produto. Os tempos de preparação são fixos e sem relações de precedência, isto é, o tempo de preparação será o mesmo para qualquer combinação de entrada e saída de produtos. Isso acontece pelo fato de que ferramentas e fixadores são individuais para cada produto, não podendo ser aproveitados de uma troca para outra. A Tabela 2 apresenta a carga-máquina em horas do exemplo. A carga-máquina pode ser considerada como o período de ocupação da máquina, gerado pela carteira de pedidos.

Produto	Carga necessária ¹ sem preparação	Tempo de preparação	Carga necessária com preparação
Roda A	5,9	0,1	6
Roda B	3	2	5
Roda C	2	1	3
Roda D	6,2	1,8	8
Roda E	1,7	0,3	2

Tabela 2: carga máquina do centro de usinagem

Além das características já citadas, outra característica importante é o custo referente à produção. Os custos levados em consideração na programação geralmente se referem a custos como os de multa aplicadas pelo cliente, caso uma encomenda não seja entregue na data solicitada.

Pensando sob o ponto de vista do cliente – montadora de veículos no exemplo em questão –, a partir do momento que se tem uma data de entrega esperada, elabora-se a programação das atividades de montagem de veículos considerando que os produtos comprados sejam entregues nas datas prometidas.

Suponha um caso hipotético em que todo o veículo seja montado, só que, por um motivo qualquer, as rodas que deveriam ser entregues para serem utilizadas na montagem ainda não haviam chegado. A montadora perde capital por uma série de motivos, dentre eles: **(1)** o investimento de capital realizado pela montadora na compra

¹ Carga necessária[hora]= Número de produtos [unidades] ÷ Produção horária [unidades/hora].

de componentes e subconjuntos que já foram utilizados no veículo e não podem ser vendidos pelo fato do veículo ainda estar incompleto; **(2)** a perda de confiança por parte do cliente final devido ao não cumprimento dos compromissos na data prometida por parte da montadora; **(3)** os custos referentes a toda movimentação e degradação com relação ao armazenamento provisório de veículos incompletos; **(4)** os custos referentes à utilização de uma área extra – provavelmente não específica para esse fim – de estocagem; etc.²

Além desses custos associados a eventos externos à empresa, pode-se acrescentar os custos associados a eventos internos do sistema produtivo. Esses por sua vez, geralmente estão associados à presença física dos produtos e ao tempo em que os mesmos permanecem no sistema produtivo. Dentre outros fatores, esses custos acontecem devido a: **(1)** investimentos de capital realizado pela empresa na compra das rodas fundidas, antes da usinagem e venda para as montadoras; **(2)** custos referentes à toda a movimentação, degradação e armazenagem de materiais; **(3)** investimentos de capital realizados pela empresa com relação aos recursos utilizados na produção como ferramentas, mão-de-obra e energia; etc.

Para efeitos de PS, todos esses custos de adiantamento e de atraso podem ser modelados por duas penalidades, uma para **adiantamento** e uma para **atraso**, denominadas respectivamente **penalidades de adiantamento** (*earliness penalties*) e **penalidades de atraso** (*tardiness penalties*). Na Tabela 3, estão apresentados valores hipotéticos das penalidades, além de toda a informação relevante ao tipo de problema de PS que será tratado na dissertação.

produto	índice da ordem= j	tempo de processamento= p_j	Data de entrega solicitada= d_j	penalidade de adiantamento= h_j	penalidade de atraso= w_j
Roda A	A	6	8	4	9
Roda B	B	5	6	1	8
Roda C	C	3	9	6	1
Roda D	D	8	4	3	3
Roda E	E	2	21	7	6

Tabela 3: dados do centro de usinagem já adaptados para PS

² Em um caso recente presenciado pelo autor, no contrato de fornecimento de espelhos retrovisores de um fabricante brasileiro de autopeças, uma das plantas da General Motors americana previu uma multa de US\$20.000/minuto, caso os subconjuntos fornecidos ocasionassem a parada da linha de montagem da montadora por qualquer motivo.

Este é um ponto básico para os modelos geralmente encontrados na literatura. A partir daqui, várias soluções são possíveis, dependendo do objetivo que a empresa deseje alcançar e dependendo de outras características adicionais que cada problema pode possuir. Mais adiante serão mencionados problemas diferentes, assim como algumas possíveis alterações neste problema exemplo.

1.2 RELAÇÃO ENTRE DECISÕES DE PROGRAMAÇÃO DA PRODUÇÃO E DECISÕES DE OUTROS NÍVEIS

Muitos estudos têm apontado para a importância da visão integrativa na administração de operações. Um dos primeiros pertence a SKINNER (1969 e 1978) que estava preocupado com o fato de que as decisões em níveis mais baixos hierarquicamente deveriam ser tomadas tendo como base uma estratégia de manufatura definida. Esta por sua vez, deveria ser definida de acordo com a estratégia corporativa da empresa.

A integração dos diversos níveis de decisão em alguns casos pode ser realizada através do **planejamento hierárquico da produção** (HAX; CANDEA, 1984 e BITRAN; TIRUPATI, 1993) que pode ser definido como uma metodologia que tenta fazer uma ligação entre os vários níveis de planejamento possuídos pelas empresas. Ele é embasado em uma classificação proposta por ANTHONY (1965). Na classificação, ANTHONY divide as decisões a serem tomadas pelas empresas com relação aos sistemas de planejamento e controle em três níveis. São eles: o planejamento estratégico, o controle administrativo e o controle operacional. Atualmente os três níveis são classificados como **nível estratégico, nível tático e nível operacional**.

A utilização da classificação permite que problemas de planejamento e controle sejam divididos em subproblemas, fazendo uma ligação entre as decisões de níveis mais altos com decisões de níveis mais baixos (BITRAN; TIRUPATI, 1993, p.525).

Os níveis estratégico e tático estão fora do escopo da presente dissertação e portanto não serão discutidos em profundidade. Serão comentados apenas como um apoio, caso ajudem o bom entendimento de um assunto tratado. O nível operacional inclui as decisões acerca da programação da produção que é um dos principais tópicos em análise.

De acordo com ANTHONY (1965, p.18) o controle operacional – nível operacional de planejamento e controle – é o processo de assegurar que tarefas específicas sejam conduzidas efetivamente e eficientemente. Podem-se citar como características básicas desse nível, a focalização em tarefas específicas e a objetividade dos procedimentos utilizados. Daí a freqüente utilização de ferramentas matemáticas e computacionais para se resolver problemas.

As decisões de nível operacional possuem o maior grau de detalhamento de todos os níveis de decisão. O grau de detalhamento das informações é muito grande. Nesse nível existem definições de que produto, quantidade, horário de fabricação e de entrega, máquinas a serem utilizadas e roteiros a serem seguidos pelos produtos através das máquinas.

Ele é o nível de maior complexidade tanto na elaboração, como na utilização de sistemas de apoio à decisão. Como a precisão e a velocidade da tomada de decisão são importantes nesse nível, surgem grandes dificuldades com a complexidade computacional no tratamento otimizado das informações. Esse tipo de problema acontece de algumas formas nos níveis superiores, entretanto é no nível operacional que surgem as maiores complicações.

1.3 AMBIENTE DO TIPO JIT (*Just-in-Time*) E O PROCEDIMENTO DE PROGRAMAÇÃO DA PRODUÇÃO

Surgidos no Japão, o JIT, o TQC (*Total Quality Control*) e o TPM (*Total Productive Maintenance*) são sistemas de administração industrial relativos ao estoque, a qualidade e a manutenção respectivamente. Todos pertencem à denominada produção enxuta (*lean production*). A produção enxuta veio se desenvolvendo desde o período do pós-guerra, a princípio na Toyota Motor Corporation e posteriormente em sua rede de fornecedores (WOMACK *et al.*, 1992). Algum tempo depois espalhou-se para outras empresas japonesas e, no início da década de 80, outras empresas – americanas principalmente – começaram a se beneficiar com esse sistema de produção (SCHONBERGER, 1986a).

Para a introdução desse sistema na Toyota e em sua rede de fornecedores, foram gastos aproximadamente 20 anos de estudos e adaptações (WOMACK *et al.*,

1992). Os plenos benefícios só são alcançados quando todo o sistema funciona de maneira integrada, desde o projeto de produtos, passando pela distribuição logística da produção, pelo *marketing* e até pela assistência técnica prestada aos clientes.

Principalmente devido ao sucesso comercial/industrial japonês nas décadas de 70, 80 e 90, os elementos do JIT tornaram-se bastante difundidos atualmente. Infelizmente, nem sempre os conceitos são bem definidos e claros. Muita divergência e mistura de conceitos são apresentados na literatura. Autores como CORRÊA; GIANESI (1993, cap.3) citam que o JIT possui características como o papel dos estoques, o tamanho dos lotes de produção e de compra, a forma de encarar os erros, a utilização da capacidade, o papel da mão-de-obra direta e indireta e a organização e limpeza da fábrica. Estudando-se mais profundamente o tema, percebe-se que essas características estão mais relacionadas à produção enxuta pois a organização e a limpeza da fábrica são características do TPM enquanto que a maneira diferente de se enxergar erros é uma característica do TQC. É verdade que esses sistemas são altamente integrados e inter-relacionados, mas de um ponto de vista teórico a separação entre os três sistemas é possível.

Uma das **características mais importante do JIT** – e que neste trabalho denomina-se filosofia JIT – **é evitar que produtos e componentes sejam produzidos ou entregues antes da data correta** (SCHONBERGER, 1986b, GROENEVELT, 1993). Assume-se que características como os erros, a limpeza e a organização da fábrica contribuem apenas indiretamente para esse fim pois fazem parte de um sistema mais macro. Outras características importantes do JIT além da já mencionada são: a diminuição do tempo de preparação; a diminuição de desperdícios como por exemplo, com estoque em processo; o envolvimento de funcionários em questões produtivas e a comunicação horizontal no chão-de-fábrica, isto é, sem a interferência de funcionários de níveis mais elevados hierarquicamente.

1.3.1 PROCEDIMENTOS DE PROGRAMAÇÃO DA PRODUÇÃO

Os dois procedimentos mais conhecidos na atualidade são o **procedimento que empurra (*push*) a produção** e o **procedimento que puxa (*pull*) a produção (*kanban*)**. Para se fazer a programação, o primeiro procedimento utiliza ferramentas

computacionais intensivas, enquanto que o segundo utiliza um controle mais manual ao nível de chão-de-fábrica. Ambos são resumidos na Tabela 4.

Normalmente o JIT utiliza um procedimento de programação que “puxa a produção” denominado *kanban* (MONDEN, 1984; JAPAN MANAGEMENT ASSOCIATION, 1989). Como esse sistema necessita que haja estoque intermediário em todos os produtos e em todos os estágios produtivos, fica claro que o sistema *kanban* foi proposto para ambientes cuja **variabilidade de produtos seja pequena** e cuja **demanda seja alta e estável** (CORRÊA; GIANESI, 1993, p.99-100).

A afirmação acima pode ser melhor entendida através de exemplos. Supondo o caso de uma empresa produtora de metais sanitários que fabrique apenas um tipo “popular” de válvula de descarga. Não haveria dificuldades em se adotar o sistema *kanban* pois a empresa utiliza poucos componentes para a produção do produto, além de o mesmo ter demanda alta e relativamente estável. Agora, suponha que essa mesma empresa possuísse vários modelos de válvulas de descarga, além de vários modelos de torneiras e de registros. Nesse caso, o número de componentes utilizados pela empresa para se fazer os produtos é muito maior e provavelmente a demanda é menor e mais instável. Em consequência seria difícil a utilização do sistema *kanban*, enquanto que a programação convencional é “preparada” para abordar esse tipo de problema.

PROCEDIMENTO QUE EMPURRA A PRODUÇÃO (<i>PUSH PROCEDURE</i>)	PROCEDIMENTO QUE PUXA A PRODUÇÃO (<i>KANBAN</i>) (<i>PULL PROCEDURE</i>)
Criação de uma ordem a partir de um pedido inicial.	Retirar do estoque final e enviar para o cliente o seu pedido.
Rodar um programa piloto e negociar a data de entrega com o cliente.	Avisar o último estágio produtivo que uma determinada quantidade de produtos está faltando no estoque final.
Decisão de quando liberar a ordem para o chão-de-fábrica.	O último estágio fabrica e envia os produtos que estão faltando no estoque final. Para isso ele utiliza seu estoque de entrada.
Cálculo dinâmico das ordens mais prioritárias.	O estoque de entrada do último estágio é relacionado ao estoque de saída do penúltimo estágio e portanto o mesmo deve produzir os itens faltantes.
Prioridades empurram a ordem pelos estágios produtivos.	Esse procedimento continua até o estágio produtivo inicial.
O produto associado à ordem é enviado para o cliente.	

Tabela 4: procedimentos de programação que empurram e que puxam a produção
(adaptada de MORTON; PENTICO, 1993, p.540)

A Figura 1 mostra um esquema gráfico simplificado de como acontece o fluxo de processamento nos sistemas de programação que puxam e que empurram a produção. É um sistema de três máquinas que utiliza um estoque inicial (E1), dois estoques intermediários (E2 e E3) e um estoque final (E4). A elaboração dos produtos deve passar pelas máquinas 1, 2 e 3 respectivamente para poderem ser vendidos. Os estados podem ser compreendidos como as configurações pelas quais o sistema produtivo passou. Os produtos fabricados são representados pelas letras A e B. Os números antes das letras indicam a quantidade de produtos estocados. Descreve-se abaixo como os procedimentos alteram as configurações do sistema para o caso onde um cliente compra 5 unidades do produto A e 5 unidades do produto B.

1) Procedimento que puxa a produção: Para o estado 1, o procedimento que puxa a produção possui estoque dos dois produtos em todos os estágios produtivos. No estado 1, o cliente retira seu pedido do estoque E4. O sistema produtivo passa para o estado 2. Imediatamente a máquina M3 é informada da falta de 5 produtos A e 5 produtos B em seu estoque de saída (E4). Para a fabricação dos produtos, a máquina M3 utiliza os produtos de seu estoque de entrada (E3). Portanto M3 utiliza 5A's e 5B's, faz a operação e consegue suprir o estoque E4. Nesse ponto o sistema passa para o estado 3. M2 percebe a falta de produtos em E3 e utiliza os produtos de E2 para preencher a demanda. O sistema passa para o estado 4. O mesmo procedimento é realizado em M1, com o sistema passando para o estado 5. Para o estado 5, o fornecedor dos produtos – matéria-prima por exemplo – para a empresa faz o abastecimento de E1 e o sistema passa para o estado 6, que é equivalente ao inicial.

2) Procedimento que empurra a produção: O procedimento que empurra a produção no estado 1, não possui estoque em nenhum ponto do sistema produtivo. Nesse estado, o cliente faz a solicitação da quantidade e de que produtos ele precisa. Sabendo-se essas quantidades, E1 é abastecido com as necessidades e o sistema passa para o estado 2. M1 percebe a presença de um estoque de entrada e elabora a operação que lhe cabe. O sistema passa para o estado 3. M2 e M3 fazem o mesmo procedimento de M1 e o sistema passa pelos estados 4 e 5. No estado 5, o cliente pode retirar seu pedido que foi feito quando o sistema estava no estado 1. Após a retirada dos produtos de E4, o sistema passa para o estado 6, que mais uma vez é equivalente ao estado 1.

Figura 1: exemplo de fluxo de produção nos procedimentos que puxam e que empurram a produção

Percebe-se facilmente que tanto os prazos de entrega quanto os estoques são diferentes nos dois exemplos e conseqüentemente, para a escolha de um dos dois sistemas, deve-se ponderar acerca de características presentes no sistema produtivo.

Alguns pontos podem ser esclarecidos com relação a variabilidade de produtos: (1) se a empresa fabricasse 2000 produtos e não apenas 2 como nos exemplos, seria impraticável ter estoques intermediários de todos eles pela fábrica; (2) se o cliente não exigisse a entrega imediata de produtos e pudesse esperar o tempo necessário para sua fabricação por todos estágios produtivos, não haveria necessidade de se ter estoque disponível. Com relação à demanda: (1) se os produtos A e B fossem muito caros seria interessante se diminuir os estoques; (2) se os produtos tivessem uma demanda muito baixa e instável, o procedimento que puxa a produção seria dispendioso pelo estoque envolvido.

Mesmo com esses pontos demonstrando que os sistemas são bastante diferentes, por algum tempo acreditou-se que os procedimentos de puxar a produção eram muito melhores do que os procedimentos de empurrar a produção. A literatura disponível algumas vezes apresenta afirmações que podem induzir ao raciocínio incorreto do leitor. AOKI (1990, p.5) por exemplo, cita que uma empresa que trabalha com o sistema *kanban* produz 20000 variedades de carros. O que não fica evidente em seu trabalho é se o projeto dos carros foi elaborado pensando-se na flexibilidade da faixa de produtos oferecidos pela empresa³. Conforme mencionado anteriormente o JIT pertencendo à produção enxuta é um sistema que engloba toda a empresa e sua cadeia de fornecedores, inclusive a área de projeto de produtos. Uma variedade de faixas de produtos provavelmente seria um fator limitante da aplicação do sistema *kanban*, mesmo no caso dessa empresa típica japonesa.

Pode-se dizer que cada um dos sistemas possui suas vantagens e desvantagens, permitindo que uma mistura adequada entre ambos seja vantajosa. Tentativas práticas têm sido feitas nesse sentido, embora sem muito sucesso (COLIN, 1996 e 1997). Nesses

³ Se para um mesmo modelo, uma empresa fabricante de veículos oferece aos seus clientes, 3 tipos de motores, 3 tipos de rodas, opções de catalisador, de ar condicionado, de toca-fitas, de direção hidráulica e de *air bag*, ela já está oferecendo $3 \times 3 \times 2 \times 2 \times 2 \times 2 = 288$ variedades do modelo. Esse tipo de variedade é chamado de **flexibilidade da faixa** de produtos, ou seja, são variedades com alterações superficiais nos produtos. Uma **variedade de faixas** significaria vários modelos diferentes de veículos, ou seja, alterações profundas nos produtos.

trabalhos, relata-se o caso de uma empresa que tenta utilizar o sistema *kanban* para os produtos de um cliente apenas. O restante dos produtos continuam sendo programados de acordo com o sistema tradicional – procedimentos que empurram a produção.

Mas o que fazer caso a empresa deseje utilizar os benefícios do JIT entretanto possua produtos com demanda baixa e instável, ou grande variabilidade de faixa de produtos? E caso deseje-se programar a produção em um ambiente cujo objetivo é a personalização em massa, isto é, produtos personalizados produzidos em grandes volumes como na produção em massa? E caso deseje-se programar a produção de departamentos de ferramentaria e manutenção, onde geralmente a produção não pode ser mantida em estoques intermediários e de saída? E caso a empresa esteja participando de um empreendimento ágil, onde a produção está grandemente apoiada na idéia do fornecimento rápido de produtos personalizados (COLIN *et al.*, 1997)?

Uma resposta às perguntas anteriormente formuladas seria a utilização de um dos **principais pontos da filosofia JIT** – evitar que produtos e componentes sejam produzidos ou entregues antes da data correta – como parte da **função-objetivo de um sistema que empurra a produção**. Colocando de outra forma, seria a utilização de uma parte da filosofia JIT em um sistema que empurra a produção.

BAKER; SCUDDER (1990, p.22), com um ponto de vista equivalente ao apresentado neste trabalho, citam que o JIT possui uma gama muito mais genérica de princípios do que simplesmente o atendimento de datas de entrega, entretanto modelos de programação com penalidades de adiantamento e de atraso parecem capturar a essência da programação com abordagem JIT.

Colocando de outra forma, pode-se dizer que embora a abordagem JIT tenha sido originada nos sistemas que puxam a produção, ela também pode ser utilizada em sistemas que empurram a produção.

1.3.2 PROCEDIMENTO DE PROGRAMAÇÃO DA PRODUÇÃO NO AMBIENTE DO TIPO JIT

Quando o processamento de um produto é finalizado em horário diferente de sua data de entrega, o mesmo incorre em custos. Caso o horário seja anterior à sua data de entrega, haverá custos referentes ao investimento de capital, ao espaço e movimentação do estoque, à degradação quando o produto é perecível e a muitos outros

custos secundários (GROENEVELT, 1993). Todos esses custos provenientes do adiantamento da execução de uma ordem serão agrupados como um único coeficiente denominado **penalidade de adiantamento (h_j)**.

Caso o horário de término do processamento de um produto seja posterior à sua data de entrega, haverá custos como os de multa aplicados pelo cliente, ou mesmo como o de potencialidade em se perder o cliente. Assim como os de adiantamento, esses custos de atraso serão agrupados em um único coeficiente denominado **penalidade de atraso (w_j)**⁴.

A função-objetivo considerada neste estudo será a soma desses custos provenientes do atraso ou do adiantamento das ordens. Portanto dependendo de certas condições, **pode valer a pena manter uma máquina ociosa**, inserindo-se tempo de ociosidade entre a realização de duas ordens. Essa característica, peculiar ao ambiente do tipo JIT, é válida enquanto os custos provenientes de um suposto atraso não for maior do que os custos provenientes do adiantamento. Vários outros trabalhos que seguem essa mesma linha de análise serão discutidos no capítulo 2.

Utilizando como base a Tabela 4 pode-se dizer que o procedimento de programação se inicia tendo a definição de quais produtos precisam ser produzidos em um determinado horizonte de tempo, juntamente com os dados relativos a cada um desses produtos. Esses dados geralmente são o número da ordem de produção (que na notação adotada é equivalente a J_j), o tempo de processamento (p_j), sua data de entrega desejada (d_j)⁵ e seu nível de importância para a empresa, definido de acordo com as penalidades de adiantamento (h_j) e de atraso (w_j).

Tendo todos esses dados, dispostos como na Tabela 3, deve-se definir qual o horário de início e de término de cada ordem de produção de modo que alguma função objetivo seja melhorada ou otimizada. Como citado anteriormente a função utilizada aqui será a **soma das penalidades de adiantamento e de atraso**. A função-objetivo também pode ser entendida como a soma dos custos envolvidos na atividade produtiva,

⁴ A maneira para se definir esses custos de um modo eficaz é objeto de estudo atualmente. Uma breve explanação sobre esse tema é apresentada em MORTON; PENTICO (1993, p.55).

⁵ No momento do pedido inicial, o próprio vendedor já poderia fornecer uma data que fosse adequada ao sistema produtivo da empresa. Para a definição dessa data, existem vários métodos que podem ser adaptados conforme as características dos sistemas produtivos. Para um tratamento mais extensivo desse assunto, além de uma vasta revisão bibliográfica, vide CHENG; GUPTA, 1989.

pois as penalidades são índices que podem agrupar os dados relativos a custos de uma determinada ordem.

Com o programa pronto, sabe-se quais ordens serão cumpridas conforme as datas de entrega desejadas pelos clientes e quais não. A partir daí faz-se a renegociação das datas de entrega com os clientes, antecipando-se algumas e postergando-se outras de forma que o programa tenha o valor da função objetivo melhorado. A partir das definições do ajuste realizado em conjunto com os clientes – é obvio que algumas datas não poderão ser mudadas –, um programa é definido.

O programa definitivo determina quais as ordens serão atendidas no prazo e quais serão atendidas fora do prazo, de modo que as restrições do problema não sejam infringidas e que a função-objetivo seja otimizada.

1.4 PROCEDIMENTOS ÓTIMOS X PROCEDIMENTOS HEURÍSTICOS

De uma maneira semelhante a outros problemas da Pesquisa Operacional, os problemas de programação e de sequenciação têm sofrido modificações com relação aos métodos de solução. Na década de 50 e início da década de 60, problemas eram resolvidos através de heurísticas deselegantes, embora com grande utilidade na solução de problemas práticos. Os procedimentos empíricos, testando regras de despacho estiveram em seu auge. A década de 60 é marcada por rebuscamentos matemáticos e aplicação de métodos ótimos como a programação dinâmica na resolução de problemas. Os anos 70 e início dos anos 80 são marcados principalmente pelas provas com relação à complexidade computacional dos algoritmos (FISHER; RINNOOY KAN, 1988). No final dos anos 80 e início dos anos 90, percebe-se um marcante retorno dos procedimentos heurísticos na resolução de problemas reais.

Conseqüentemente, para a elaboração da dissertação não houve muitos problemas para se encontrar um procedimento heurístico de bom desempenho e perspectivas de aplicações futuras. O procedimento é denominado *beam search* (BS) e pode ser considerado um **método heurístico de enumeração parcial** (WINSTON, 1992). Pela pesquisa bibliográfica realizada no capítulo 3, sabe-se que além das funções de prioridade, o BS e a troca de pares adjacentes são os únicos procedimentos

heurísticos utilizados até o momento na resolução de problemas de programação equivalente ao estudado nesta dissertação. No próximo capítulo, serão apresentadas de forma simplificada as principais idéias contidas nesses procedimentos heurísticos.

Os **procedimentos ótimos** como a programação dinâmica ou o *branch-and-bound* conseguem resolver apenas instâncias muito pequenas de problemas e portanto só servem como critérios de comparação e avaliação de procedimentos heurísticos. RINNOOY KAN; MAGAZINE (1979) apontam que os **algoritmos otimizantes** devem ser utilizados em formulações que **ignoram algumas complexidades da situação**, enquanto que os **algoritmos aproximativos** são importantes na resolução de problemas com **formulações precisas**.

1.5 ORGANIZAÇÃO DA DISSERTAÇÃO

No capítulo 2 define-se exatamente o problema em questão, como pode ser resolvido, qual sua relevância e as formas como já foi tratado na literatura. Os métodos de *beam search* mais difundidos e uma revisão bibliográfica da aplicação do *beam search* em problemas de programação e seqüenciação são apresentados no capítulo 3. No capítulo 4 apresenta-se uma função de prioridade e um algoritmo que podem ser utilizados para a inserção de ociosidade no problema em análise. O capítulo 5 apresenta o estudo computacional e os resultados que foram obtidos no mesmo. As conclusões e as contribuições, além de sugestões para pesquisas futuras são apresentadas no capítulo 6. O anexo A apresenta a classificação de problemas de programação e seqüenciação utilizadas no texto. O anexo B apresenta um breve resumo acerca da complexidade computacional com relação à programação e à seqüenciação. O anexo C apresenta detalhes dos resultados do estudo computacional, enquanto que o anexo D apresenta a demonstração do teorema utilizado no capítulo 4. Finalmente o apêndice 1, apresenta os programas computacionais em Turbo-Pascal utilizados no experimento e o apêndice 2 apresenta exemplos de instâncias testadas.

2 DEFINIÇÃO, IMPORTÂNCIA E REVISÃO BIBLIOGRÁFICA DO PROBLEMA

A análise de problemas de programação e seqüenciação (PS) da produção é bastante difícil, tanto por sua complexidade quanto por sua extensão. O tema é tratado há muito tempo, possuindo trabalhos importantes já na década de 50.

Desde então, a análise de casos gerais ou particulares, teóricos ou práticos, grandes ou pequenos, tem sido fabulosa. LAWLER *et al.* (1993, p.446) citam que “a teoria da programação e da seqüenciação, mais do que qualquer outra área na pesquisa operacional, é caracterizada por um número virtualmente ilimitado de tipos de problemas.”

Por consequência torna-se fundamental uma classificação coerente dos problemas já estudados ou que necessitem de outros estudos. O anexo A apresenta a classificação presente em LAWLER *et al.* (1993, p.450-4) com algumas variações com a finalidade de torná-la mais abrangente e contendo alguns dos casos aqui estudados.

As definições com relação à complexidade computacional são apresentadas no anexo B. Definições e terminologias utilizadas acerca da PS são baseadas principalmente nos livros de CONWAY *et al.* (1967), BAKER (1974) e em MORTON; PENTICO (1993). As funções matemáticas mais utilizadas na dissertação serão resumidas no próximo subitem.

Neste trabalho haverá uma diferença entre a programação e a seqüenciação. A **seqüenciação** será definida como uma **ordenação dos pedidos** – ordens – de produção. Para um conjunto de ordens $J=\{J_1, \dots, J_n\}$, deve-se encontrar a seqüência $\sigma=\langle \sigma(1), \sigma(2), \dots, \sigma(n) \rangle$ com $\sigma(j) \in \{1, 2, \dots, n\}$ e $\sigma(j) \neq \sigma(i)$ sempre que $j \neq i$. Cada ordem J_j ($j=1, \dots, n$), possui quatro valores associados $\{p_j, d_j, w_j, h_j\}$, onde p_j, d_j, w_j, h_j é o tempo de processamento, a data de entrega, a penalidade de atraso e a penalidade de adiantamento de respectivamente.

A **programação** será definida como a **determinação dos horários de início e de término das ordens** de uma dada seqüência. Para uma dada ordem J_j ($j=1, \dots, n$), existe um intervalo de processamento $[e_j, e_j+p_j]$ onde e_j e p_j significam respectivamente o horário efetivo de início do processamento e o tempo de processamento da ordem J_j . No caso do problema de uma máquina, para duas ordens consecutivas $J_{\sigma(j)}$ e $J_{\sigma(j+1)}$ com

intervalos de processamento definidos por $[e_{\sigma(j)}, e_{\sigma(j)} + p_{\sigma(j)}]$ e $[e_{\sigma(j+1)}, e_{\sigma(j+1)} + p_{\sigma(j+1)}]$, poderá haver intersecção entre os dois intervalos apenas nos pontos extremos dos intervalos. Seja $W_{\sigma(j)}$ o período de ociosidade antes da realização da ordem $J_{\sigma(j)}$. O programa $\pi(\sigma)$ de um conjunto de ordens J , é a determinação do horário de início e de término de cada ordem, ou seja, $\pi(\sigma) = \langle W_{\sigma(1)}, p_{\sigma(1)}, W_{\sigma(2)}, p_{\sigma(2)}, \dots, W_{\sigma(n)}, p_{\sigma(n)} \rangle$. Semanticamente pode-se dizer que para uma dada seqüência, um programa define o período de ociosidade antes do início de cada ordem $W_{\sigma(j)}$ e o período de processamento da mesma $p_{\sigma(j)}$, para $j=1, 2, \dots, n$.

Para casos em que não haja a possibilidade da inserção de ociosidade, isto é, $W_{\sigma(j)}=0 \forall j \in \{1, 2, \dots, n\}$, a programação e a seqüenciação são equivalentes pois cada ordem possui apenas um tempo de processamento a ela atribuído.

2.1 FUNÇÕES MATEMÁTICAS MAIS UTILIZADAS NA DISSERTAÇÃO

Este subitem fará uma revisão sucinta das funções matemáticas mais utilizadas na dissertação com o intuito de facilitar a compreensão nos tópicos adiante. No entanto, casos específicos só serão comentados no anexo A. O anexo A possui explicações em um âmbito mais genérico e mais profundo.

Para um programa π , denomina-se o **horário de término da ordem $J_{\sigma(j)}$** a função:

$$C_{\sigma(j)}(\pi) = \begin{cases} W_{\sigma(j)}(\pi) + p_{\sigma(j)} & \text{para } j = 1 \\ C_{\sigma(j-1)}(\pi) + W_{\sigma(j)}(\pi) + p_{\sigma(j)} & \text{para } j = 2, \dots, n. \end{cases}$$

A maioria das outras funções de programação dependem dessa função. A função **diferença** representa a quantidade de tempo em que o término de uma ordem difere de sua data de entrega. Por uma questão de simplicidade notacional, quando não causar erro conceitual, será dada preferência para a notação que omite a seqüência σ . Exemplos desse caso são apresentados abaixo, onde os valores das funções para $J_{\sigma(j)}$ são equivalentes a valores para J_j . A função diferença é definida como:

$$L_j = C_j - d_j.$$

Costuma-se diferenciar a função diferença quando ela é positiva e quando ela é negativa. Quando ela é negativa, é chamada de **adiantamento** e é definida por:

$$E_j = \max(0, -L_j).$$

Quando L_j é positiva é chamada de **atraso** e pode ser definida por:

$$T_j = \max(0, L_j).$$

Semanticamente pode-se definir o adiantamento e o atraso conforme o conhecimento comum, ou seja, a quantidade de tempo em que uma ordem é terminada antes de sua data de entrega e a quantidade de tempo em que uma ordem é terminada após sua data de entrega, respectivamente.

Então, se uma empresa define no nível de decisões estratégicas, que em suas operações é de fundamental importância que o atraso nas datas de entrega de seus produtos seja pequeno, essa empresa pode definir como objetivo de sua programação a menor quantidade de atrasos possíveis. Nesse caso, o problema poderia ser tratado como $\min \sum T_j$ que é a soma dos atrasos absolutos.

Posteriormente, essa mesma empresa poderia sofisticar seu sistema de programação adicionando uma determinada penalidade w_j para cada ordem J_j de acordo com a “prioridade” de cada uma. Outra forma – e por sinal a mais correta – de se tratar a penalidade é defini-la de acordo com os custos, conforme mencionado anteriormente no capítulo 1. Nesse caso a função objetivo do sistema de programação se transformaria em $\min \sum w_j T_j$.

Figura 2: hierarquia entre algumas funções objetivo

Dependendo do tipo de empresa essa função poderia ser do tipo $\min \sum (E_j + T_j)$, representando que o adiantamento é tão indesejável quanto o atraso. Nesse caso, a função-objetivo considera apenas a quantidade de tempo que as ordens estão adiantando

ou atrasando. Essas funções podem variar muito, gerando as mais diferentes condições para serem pesquisadas. A Figura 2 – excerta da Figura 16 do anexo A – representa uma hierarquia nessas funções. O sentido das setas representa uma maior complexidade e abrangência das funções.

2.2 DEFINIÇÃO DO PROBLEMA⁶

Será considerado o caso de uma **única máquina** que deverá processar uma ordem de cada vez de um conjunto de ordens $J = \{J_1, J_2, \dots, J_n\}$. A ordem $J_{\sigma(j)}$ (onde $\sigma(j)$ representa a j -ésima ordem na seqüência σ) deve ser completada antes que a ordem $J_{\sigma(j+1)}$ seja iniciada para $\sigma(j)=1, 2, \dots, n-1$. As **chegadas** das ordens serão **estáticas**, ou seja, todas estarão disponíveis no instante inicial. O programa deverá ser realizado de maneira a **cumprir as datas de entrega** d_j , sabendo-se que cada ordem demanda um determinado tempo p_j para o seu processamento. Há possibilidade de **inserção de tempo ocioso** W_j antes da elaboração de uma determinada ordem J_j . A **função objetivo** a ser analisada será do tipo $\min \sum_{j=1}^n (h_j E_j + w_j T_j)$, sendo E_j o adiantamento de J_j e T_j o atraso de J_j , com suas respectivas penalidades $h_j > 0$ e $w_j > 0$. De acordo com a notação apresentada no anexo A, esse problema é definido como 1| W_j | $\Sigma(h_j E_j + w_j T_j)$.

Pela característica de se penalizar uma ordem que é feita antes da sua data de entrega, a inserção de ociosidade se torna relevante⁷. Percebe-se pela formulação que o problema se tornou extremamente restrito. Para uma adaptação em um caso real, deve

⁶ Este subitem define exatamente o problema de principal interesse tratado na dissertação. Além desse, outros casos relacionados de alguma forma serão comentados e estudados no decorrer do trabalho.

⁷ Utilizando mais uma vez a Figura 1 para exemplificação, pode-se imaginar o caso onde o custo é de R\$ 100/dia para se manter um lote de 400 produtos A. O tempo de atravessamento pelas 3 máquinas em condições normais é de 1 dia. Um cliente solicitou 400 unidades do produto A para o dia 18, sendo que o dia atual é 3. Se a ordem contendo 400 unidades do produto A for programada para ser feita no dia 4, ela estará disponível no dia 5 e deverá aguardar 13 dias até ser entregue. Como o custo de adiantamento diário é de R\$ 100, o custo de adiantamento total para se produzir o produto no dia 4 será R\$ 1.300. Caso se insira 13 dias de ociosidade e o produto seja programado para o dia 17, dia 18 ele estará disponível, sem custo de adiantamento.

ser levado em consideração que alguns pressupostos foram tomados como verdadeiros. São eles:

- **estoque de produtos:** pela formulação, não está sendo levado em consideração a utilização de estoque para o atendimento da demanda. Em outras palavras, supõe-se que o produto só é feito após o pedido do cliente e, portanto, não há estoque para atendimento imediato. Embora isso seja altamente desejado, dificilmente alguma empresa trabalharia sem estoque de produtos. Pelo exemplo apresentado na Figura 1, a utilização de um estoque de produtos acabados poderia fornecer um atendimento instantâneo ao cliente, ou seja, equivalente ao caso do sistema que puxa produção;
- **estoque de matérias-prima:** assume-se que a matéria-prima está disponível para a realização das ordens de produção. Geralmente existe uma certa dependência entre os prazos de entrega prometidos e a disponibilidade de matérias-prima;
- **mão-de-obra:** também não é levada em consideração no modelo proposto. Assume-se que esteja disponível para a elaboração das ordens, durante todo o período de processamento. As decisões acerca da mão-de-obra devem ser tomadas no nível tático;
- **problemas de qualidade e quebras de máquina:** também são considerados como inexistentes. Nos casos reais, geralmente esses problemas são resolvidos com a utilização de estoques finais e intermediários. Em análises teóricas, essas características geralmente são tratadas probabilisticamente, tornando os modelos de programação muito mais complexos do que os determinísticos.

Aparentemente os pressupostos apontados anteriormente transformam o caso em estudo em um caso irreal. Entretanto, deve-se lembrar que uma atividade produtiva é altamente complexa e sua operação deve ser feita em vários departamentos independentes para só então ser agrupada em um resultado final. Como citado no capítulo 1, as operações devem ser encaminhadas de forma a atender aos objetivos corporativos definidos pela estratégia da empresa⁸. Para o caso da programação da

⁸ Os objetivos corporativos são os principais objetivos de uma empresa. De uma maneira geral eles guiam todos os outros objetivos de âmbito mais restrito. Um objetivo corporativo poderia ser o pagamento para

produção, os objetivos corporativos geralmente se transformam em algum critério relacionado ao atendimento correto de datas e à minimização de custos relacionados a atividades produtivas.

2.3 IMPORTÂNCIA DO PROBLEMA

De uma maneira geral, os procedimentos utilizados em PS se iniciam pelo estudo de uma única máquina. O caso do problema tratado aqui não será uma exceção. Embora seja de relevância indiscutível, poucos estudos foram feitos nesse tema após o trabalho de YANO; KIM (1991).

Uma razão percebida no decorrer da revisão bibliográfica é que o problema de **data única** – onde todas as ordens programadas possuem a mesma data de entrega –, possui 2 propriedades que permitem um desenvolvimento teórico bastante profundo. A primeira propriedade diz que existe um programa ótimo no qual não há inserção de ociosidade a partir da primeira ordem. A segunda diz que o programa ótimo possui um formato em V^9 . Conseqüentemente, quase a totalidade dos pesquisadores têm publicado resultados que levam em consideração apenas uma data de entrega.

No caso de **datas múltiplas**, isto é, cada ordem possui uma data de entrega a ela associada, ainda não se encontrou nenhum resultado do tipo citado anteriormente para datas únicas e as únicas abordagens disponíveis para se tratar instâncias relativamente grandes do problema são as heurísticas. De qualquer forma, o problema de datas múltiplas é uma generalização do problema de data única e por conseqüência será estudado futuramente como uma evolução natural.

os acionistas de dividendos de 15%/ano sobre o capital investido na forma de ações. A obtenção desse objetivo não poderia ser direta, e portanto, outros objetivos que pudessem ser alcançados de maneira mais fácil e em um prazo menor deveriam ser definidos. O objetivo de pagamento de dividendos, traduzido de acordo com o enfoque da programação poderia ser a minimização dos custos diários envolvidos nas atividades produtivas. Percebe-se que a obtenção de objetivos de curto prazo pode levar ao cumprimento de longo prazo - objetivo corporativo.

⁹ Esse tipo de programa, para o caso sem penalidades pode ser definido como LPT até a data de entrega e SPT após a data de entrega. Para o caso com penalidades é um programa cuja seqüência é definida por WLPT antes da data de entrega e por WSPT após a data de entrega.

Publicações que utilizam os procedimentos de **inserção de ociosidade** publicados em 1987, 1988 e 1991 – e discutidos adiante – só começaram a aparecer recentemente. A utilização da inserção de ociosidade no problema definido no item 2.2 ainda não foi abordada até o momento na literatura pesquisada.

2.3.1 IMPORTÂNCIA DO PROBLEMA DE UMA MÁQUINA

Em quase sua totalidade, as empresas industriais possuem sistemas de produção que englobam vários ambientes de programação. Não é difícil encontrar empresas que necessitariam programar máquinas individuais, máquinas paralelas, *flow shops* e *job shops*¹⁰. Uma programação englobando vários ambientes, que são os casos mais gerais e reais, será caracterizada como a programação de um *job shop* que dentre todos, é o nível de maior complexidade (BUFFA, 1971, p.555).

Recentemente, vários métodos com soluções rápidas e próximas da ótima – para alguns critérios de otimização – têm sido apresentados na literatura para a programação de *job shops*. Como exemplos podem-se citar as seguintes heurísticas:

- **Tipo-OPT** (*Optimized Production Technology*) (MORTON; PENTICO, 1993). Nesse tipo de heurística uma fábrica é dividida entre as máquinas anteriores à máquina gargalo¹¹, a máquina gargalo e as máquinas posteriores à máquina gargalo. A programação é feita para a única máquina gargalo e todas as outras máquinas pré e pós-gargalo são programadas com o intuito de otimizar a operação da máquina gargalo;
- **Gargalo móvel** (ADAMS *et al.*, 1988, DAUZERE-PEREZ; LASSERRE, 1993). O método seqüencia as máquinas, uma a uma sucessivamente, identificando a máquina gargalo dentre as máquinas ainda não seqüenciadas. Após uma nova máquina ser seqüenciada, as máquinas que já haviam sido seqüenciadas são re-otimizadas localmente. Os métodos de

¹⁰ As definições de máquinas paralelas, *flow shops* e *job shops* são encontradas no anexo A.

¹¹ A máquina gargalo pode ser definida como aquela que restringe a capacidade de produção de uma empresa. É essa máquina que determina a produção máxima da empresa. Para um *job shop*, dependendo do critério de otimização que se está trabalhando, podem existir vários gargalos, ou seja, gargalos específicos para certas operações em certos produtos. Um procedimento proposto recentemente, denominado **dinâmica do gargalo** (MORTON; PENTICO, 1993) estuda esse tipo de problema, inclusive o modo como podem ser determinadas as máquinas gargalo.

identificação da máquina gargalo e de re-otimização são baseados na repetição de soluções de problemas de máquinas individuais.

Um ponto interessante a ser notado é que as duas heurísticas citadas acima, além de outras, possuem a rapidez da solução fundamentalmente baseadas na solução de problemas de uma máquina, justificando a atualidade e importância do estudo da programação em máquinas individuais.

Os procedimentos propostos na dissertação poderiam ser utilizados na primeira heurística, mas não na segunda. Isso acontece pelo fato de que a primeira heurística é mais simples e aceita uma gama maior de funções objetivo. A segunda é adequada apenas para a minimização do horizonte de processamento ($\min C_{\max}$) e, até o momento, não se conseguiu uma adaptação de um procedimento semelhante para outros casos de funções-objetivo.

Um procedimento válido poderia ser a localização da máquina gargalo em um *job shop*, a utilização dos procedimentos propostos na dissertação para a programação dessa máquina e, após isso, utilizar regras de despacho para se programar as outras máquinas que não sejam gargalo. As regras de despacho utilizadas para as outras máquinas deveriam beneficiar – no sentido de tentar manter – a programação já feita para a máquina gargalo.

2.3.2 IMPORTÂNCIA DA INSERÇÃO DE OCIOSIDADE E DAS PENALIDADES DISTINTAS PARA ADIANTAMENTO E PARA ATRASO

Assim como comentado anteriormente, a inserção de ociosidade é relevante em ambientes onde exista um custo caso o produto seja finalizado em horário anterior à sua data de entrega. Como esse custo é absolutamente eliminado com a inserção de ociosidade não há razão para aceitá-lo. O inconveniente dessa inserção é um possível “arrependimento” futuro no caso de se precisar do tempo que foi gasto como ociosidade.

Outro problema que deve ser resolvido é com relação à ponderação das penalidades de forma que se alcance um **custo baixo**. Caso as penalidades de adiantamento sejam muito grandes com relação às de atraso, pode valer a pena não ter nenhuma ordem realizada antes de sua data e ter algumas atrasadas. No caso oposto, com as penalidades de atraso sendo muito grandes em relação às de adiantamento, pode

não valer a pena ter ociosidade inserida, fazendo com que algumas ordens sejam terminadas antes de sua data prevista.

O caso de **penalidades distintas** para adiantamento e para atraso é facilmente justificável pela **semelhança à realidade**. Como as penalidades deveriam ser especificadas com relação à custos provenientes de adiantamento ou de atraso, é altamente improvável que sejam iguais.

Talvez o que ocorra é uma certa proporcionalidade entre penalidades de adiantamento e de atraso, como utilizado nos estudos de OW; MORTON (1988 e 1989), ou uma proporcionalidade entre tempos de processamento (p_j) e valor das penalidades (h_j e w_j), conforme estudos de RACHAMADUGU (1995). Embora os autores tenham trabalhado com essas hipóteses teoricamente, não se mencionou nem se encontrou nenhuma referência que estudasse especificamente como deveriam ser determinadas.

2.4 ALGUMAS HEURÍSTICAS CONHECIDAS PARA A SEQUÊNCIAÇÃO DO PROBLEMA

Como será visto no capítulo 4, a partir de uma sequência pré-definida, consegue-se determinar a inserção de ociosidade e por conseqüência o programa, sem grandes dificuldades. Portanto, o problema ainda remanescente é a sequênciação. Neste subitem serão apresentados alguns procedimentos heurísticos conhecidos que podem ser utilizados para se determinar a sequência de ordens no problema em questão.

2.4.1 REGRA DE DESPACHO

Um possível método de solução é a utilização de **regras de despacho** para a determinação das seqüências. De acordo com BHASKARAN; PINEDO (1992, p.2182), despacho no contexto da programação da produção é um procedimento que utiliza regras de decisão lógica para selecionar uma ordem de produção dentre as possíveis, para o processamento em uma máquina que terminou de ficar disponível. Essas regras de decisão também são chamadas regras de despacho. De uma maneira mais direta pode-se dizer que é uma regra que determina qual ordem processar, no momento em que uma máquina termina o processamento de outra ordem.

As regras de uma maneira geral são bastante simples sendo que a heurística resume-se em um procedimento de ordenação. Há vinte anos, PANWALKAR;

ISKANDER (1977) já colecionavam 113 regras distintas. As regras em alguns casos e para algumas funções objetivo, são otimizantes. Resultados nesse sentido podem ser encontrados em CONWAY *et al.* (1967), BHASKARAN; PINEDO (1992) e em LAWLER *et al.* (1993).

Considere a seqüência $\sigma = \langle \sigma(1), \sigma(2), \dots, \sigma(n) \rangle$ com $\sigma(j) \in \{1, 2, \dots, n\}$ e $\sigma(j) \neq \sigma(i)$ sempre que $j \neq i$. Algumas das regras mais conhecidas, além das utilizadas na dissertação podem ser definidas como:

- **SPT** (*Shortest Processing Time*): seqüenciar as ordens de acordo com o menor tempo de processamento, isto é, dentre as ordens disponíveis para seqüenciação, processar aquela de menor tempo de processamento. A seqüência é tal que $p_{\sigma(1)} \leq p_{\sigma(2)} \leq \dots \leq p_{\sigma(n)}$;
- **LPT** (*Longest Processing Time*): seqüenciar as ordens de acordo com o maior tempo de processamento. A seqüência é tal que $p_{\sigma(1)} \geq p_{\sigma(2)} \geq \dots \geq p_{\sigma(n)}$;
- **WSPT** (*Weighted Shortest Processing Time*): seqüenciar as ordens de acordo com a maior razão entre a penalidade de atraso e o tempo processamento. A seqüência é tal que $(w_{\sigma(1)}/p_{\sigma(1)}) \geq (w_{\sigma(2)}/p_{\sigma(2)}) \geq \dots \geq (w_{\sigma(n)}/p_{\sigma(n)})$;
- **WLPT** (*Weighted Longest Processing Time*): seqüenciar as ordens de acordo com a menor razão entre a penalidade de adiantamento e o tempo processamento. A seqüência é tal que $(h_{\sigma(1)}/p_{\sigma(1)}) \leq (h_{\sigma(2)}/p_{\sigma(2)}) \leq \dots \leq (h_{\sigma(n)}/p_{\sigma(n)})$;
- **EDD** (*Earliest Due Date*): seqüenciar as ordens de acordo com o menor data de entrega. A seqüência é tal que $d_{\sigma(1)} \leq d_{\sigma(2)} \leq \dots \leq d_{\sigma(n)}$;
- **MST** (*Minimum Slack Time*): seqüenciar as ordens de acordo com o menor tempo de folga, isto é, para cada ordem J_j disponível, calcular $d_j - p_j$ e processar a ordem que produziu o menor valor calculado. A seqüência é tal que $(d_{\sigma(1)} - p_{\sigma(1)}) \leq (d_{\sigma(2)} - p_{\sigma(2)}) \leq \dots \leq (d_{\sigma(n)} - p_{\sigma(n)})$;
- **FIFO** (*First In First Out*): seqüenciar as ordens de acordo com a que está a mais tempo disponível para processamento, isto é, dentre todas as ordens disponíveis, seqüenciar aquela que está a mais tempo na fila da máquina. A

seqüência é tal que $r_{\alpha(1)} \leq r_{\alpha(2)} \leq \dots \leq r_{\alpha(n)}$ sendo r_j o horário em que a ordem J_j ficou disponível para iniciar o processamento;

- **LIFO (Last In First Out)**: seqüenciar as ordens de acordo com a que está a menos tempo disponível para processamento, isto é, dentre todas as ordens disponíveis, seqüenciar aquela que está a menos tempo na fila da máquina.

A seqüência é tal que $r_{\alpha(1)} \geq r_{\alpha(2)} \geq \dots \geq r_{\alpha(n)}$.

Deve-se observar que as regras FIFO e LIFO não podem ser utilizadas na resolução do problema definido no item 2.2 pois são regras que se baseiam em casos dinâmicos, ou seja, as ordens têm horários de chegada diferentes. Como o caso tratado aqui é estático, todas as ordens possuem o mesmo horário de chegada e portanto não há como definir a primeira ou a última ordem que entrou na fila para ser processada.

Após uma seqüência completa ser definida pela regra de despacho, o valor da função-objetivo pode ser determinado.

EXEMPLO NUMÉRICO: Neste exemplo e nos seguintes deste capítulo, serão utilizados os dados referentes ao exemplo da programação do centro de usinagem – item 1.1 – para uma comparação entre valores alcançados pela função objetivo e procedimentos utilizados.

ordem		A	B	C	D	E	$\Sigma(h_j E_j + w_j T_j)$
Data de entrega	d_j	8	6	9	4	21	
Tempo de processamento	p_j	6	5	3	8	2	
Penalidade de adiantamento	h_j	4	1	6	3	7	
Penalidade de atraso	w_j	9	8	1	3	6	
Seqüência EDD	C_j	19	13	22	8	35	264
D,B,A,C,E	L_j	11	7	13	4	14	
Seqüência SPT	C_j	16	10	5	35	2	354
E,C,B,A,D	L_j	8	4	-4	31	-19	
Seqüência WSPT	C_j	13	7	35	21	2	263
E,B,A,D,C	L_j	5	1	26	17	-19	
Seqüência LPT	C_j	14	19	22	8	35	267
D,A,B,C,E	L_j	6	13	13	4	14	
Seqüência WLPT	C_j	19	5	22	13	35	224
B,D,A,C,E	L_j	11	-1	13	9	14	
Seqüência MST	C_j	19	13	22	8	35	264
D,B,A,C,E	L_j	11	7	13	4	14	

Tabela 5: exemplos da utilização de regras de despacho na programação do centro de usinagem

A Tabela 5 resume as seqüências obtidas e alguns passos intermediários da utilização das regras de despacho mencionadas anteriormente, assim como os valores alcançados pela função objetivo.

Pela Tabela 5 percebe-se que a regra de melhor desempenho foi a WLPT. Aparentemente, pelo fato de que a regra WLPT não foi desenvolvida para o tipo de função objetivo utilizada, o resultado é meramente ocasional.

2.4.2 FUNÇÃO DE PRIORIDADE

A distinção entre regra de despacho e função de prioridade acontece pois a última admite a inserção de ociosidade. A literatura até o momento não fez uma distinção muito clara entre ambas. Por exemplo, BHASKARAN; PINEDO (1992) as chamam de regras de despacho compostas e OW; MORTON (1989) de regras de prioridade de despacho, sem diferenciá-las das regras de despacho convencionais. Ambos os trabalhos consideraram seu uso sem a inserção de ociosidade. A não inserção de ociosidade é uma característica peculiar às regras de despacho.

MORTON; PENTICO (1993) as chamam de prioridade ou de funções de prioridade e admitem que as mesmas podem ser utilizadas para se inserir ociosidade. Por admitir a inserção de ociosidade, dificilmente poderiam ter um nome associado aos procedimentos tipo despacho. Portanto, pode-se definir uma **função de prioridade** como um procedimento que utiliza informações das ordens para definir qual deve ser processada subseqüentemente e se a mesma deve aguardar algum tempo antes de iniciar seu processamento.

As funções de prioridade também podem ser utilizadas como regras de despacho, isto é, no modo despacho, desde que não se admita a inserção de tempo ocioso. Sob esse enfoque, a função de prioridade pode ser entendida como uma sofisticação das regras de despacho de modo que se admita a inserção de ociosidade.

Para exemplificar a utilização de uma função de prioridade, será utilizada a função EXP-ET (*Exponential Earliness Tardiness*) proposta por OW; MORTON (1988 e 1989). Nesta função, a prioridade de uma ordem J_j no horário t , $\phi_j(t)$, é definida como sendo:

$$\varphi_j(t) = \begin{cases} \frac{w_j}{p_j} \exp\left(-\frac{(h_j + w_j)}{h_j} \frac{\max(0, d_j - t - p_j)}{p_{med}}\right) & \text{se } d_j - t - p_j \leq \frac{w_j k p_{med}}{h_j + w_j} \\ \frac{h_j}{p_j} \left(\frac{w_j}{h_j} - \frac{(h_j + w_j)}{h_j} \frac{\min(k p_{med}, d_j - t - p_j)}{k p_{med}} \right)^3 & \text{para outros casos,} \end{cases}$$

onde k é um **parâmetro de ajuste** que determina o horário em que a função de prioridade começa crescer e p_{med} é o tempo de processamento médio das ordens que ainda não foram seqüenciadas, isto é, que ainda estão na fila de entrada da máquina.

Para o momento é suficiente entender que assim que uma ordem termina o processamento, em um determinado horário t , deve-se calcular a prioridade $\varphi_j(t)$ para as ordens ainda disponíveis. A ordem com maior prioridade é seqüenciada em seguida. Nesse caso a função de prioridade está sendo utilizada como uma regra de despacho pois não se está admitindo a inserção de ociosidade. Mais adiante, essa função de prioridade será explicada em maior profundidade, inclusive como a mesma deve ser utilizada para se inserir a ociosidade.

EXEMPLO NUMÉRICO (item 1.1): Assume-se $k=5$. Para a utilização da função, define-se as ordens ainda disponíveis para serem seqüenciadas e calcula-se o p_{med} . A Tabela 6 apresenta alguns passos intermediários da utilização da função EXP-ET. A coluna j representa as ordens disponíveis para serem seqüenciadas; p_{med} a média dos tempos de processamento dessas ordens que estão disponíveis; t o horário em que a máquina ficará livre e $\varphi_j(t)$ a prioridade da ordem no horário t .

j	t	$\varphi_j(t)$	j	t	$\varphi_j(t)$	j	t	$\varphi_j(t)$	j	t	$\varphi_j(t)$	j	t	$\varphi_j(t)$
A	0	0,3872												
B	0	0,2454	B	6	1,6000									
C	0	-0,0039	C	6	0,3333	C	11	0,3333	C	19	0,3333	C	21	0,3333
D	0	0,3750	D	6	0,3750	D	11	0,3750						
E	0	-0,8065	E	6	-0,0352	E	11	0,0973	E	19	3,0000			
$p_{med}=4,8$			$p_{med}=4,5$			$p_{med}=4,3$			$p_{med}=2,5$			$p_{med}=3$		

Tabela 6: exemplo da utilização de uma função de prioridade na programação do centro de usinagem

A seqüência definida pela função de prioridade foi A,B,D,E,C, cujo valor da função objetivo é $\Sigma(h_j E_j + w_j T_j) = 119$. Nota-se que embora o cálculo seja um pouco mais complexo do que as regras de despacho, o resultado foi altamente favorável.

2.4.3 TROCA DE PARES ADJACENTES

A troca de pares adjacentes é bastante conhecida na área da programação e da seqüenciação. Isso se deve talvez pelo fato de ter sido utilizada em um dos trabalhos pioneiros da área, pertencente à SMITH (1956).

Basicamente a idéia consiste em definir uma seqüência inicial e a partir daí fazer trocas entre ordens vizinhas na seqüência, de modo que a função-objetivo seja otimizada localmente. Quando uma troca entre ordens vizinhas não permite uma melhoria da função-objetivo, um ótimo local é encontrado. A definição de uma boa seqüência inicial, também chamada de semente, é primordial para o bom desempenho da troca de pares adjacentes. Alguns autores sugerem que a semente pode ser obtida aleatoriamente, mas a geração mais amplamente utilizada é através de uma regra de despacho.

Existem muitas variações dessa heurística como por exemplo permitir que a troca seja feita não só com a ordem vizinha – troca de pares adjacentes –, mas também com uma vizinhança estendida – troca de pares globais –, ou seja, a troca com vizinhos mais distantes. Referências nesse tema são BAKER (1974, p.43-4), FRENCH (1982, p.168-70) e MORTON; PENTICO (1993, p.76-88).

EXEMPLO NUMÉRICO (item 1.1): Neste caso, a troca de pares adjacentes terá $(n-1)=4$ seqüências geradas a partir da semente. Serão utilizadas duas sementes – portanto duas soluções estarão sendo geradas –, uma gerada pela EXP-ET e outra pela WLPT que alcançou o melhor valor da função objetivo dentre as regras de despacho testadas.

seqüência	$\Sigma(h_j E_j + w_j T_j)$	Observação
A,B,D,E,C	119	semente EXP-ET
B,A,D,E,C	99	troca entre A e B
A,D,B,E,C	168	troca entre B e D
A,B,E,D,C	181	troca entre D e E
A,B,D,C,E	190	troca entre E e C
B,D,A,C,E	224	semente WLPT
D,B,A,C,E	264	troca entre B e D
B,A,D,C,E	170	troca entre D e A
B,D,C,A,E	245	troca entre A e C
B,D,A,E,C	153	troca entre C e E

Tabela 7: exemplo da utilização da troca de pares adjacentes na programação do centro de usinagem - semente por EXP-ET e por WLPT

A Tabela 7 apresenta alguns passos intermediários do procedimento aplicado ao problema da programação do centro de usinagem do item 1.1.

Em ambos os casos a troca de pares adjacentes proporcionou uma melhoria com relação à semente.

2.4.4 BEAM SEARCH

Outro possível método de solução é a utilização de um **procedimento de busca** que gere seqüências e analise a função-objetivo, buscando os menores valores encontrados. O procedimento heurístico de busca denominado *beam search* funciona nesse sentido. Existem muitas variações do mesmo, entretanto será utilizada uma que utiliza uma avaliação global – isto é, com seqüências completas –, permitindo um bom desempenho de maneira geral.

O procedimento se inicia com a definição de uma seqüência inicial, que é uma seqüência completa. Obviamente que quanto melhor a seqüência inicial, melhores serão os resultados obtidos pelo método. Posteriormente, no primeiro nível da árvore de busca, todas as ordens são testadas na primeira posição da seqüência com o restante da seqüência sendo definido de acordo com a seqüência inicial sem a ordem que ficou na primeira posição. Todas as n seqüências geradas têm suas funções-objetivo analisadas e apenas b delas são retidas para terem nós arborescidos no segundo nível. O número b é chamado de largura da busca, isto é, o número de seqüências retidas para uma arborescência no nível posterior.

Para o j -ésimo nível, o procedimento pode ser resumido como segue: cada nó pertencente à posição $(j-1)$ das b seqüências retidas no nível $(j-1)$ será arborescido. A arborescência é completa, ou seja, todos as ordens que não pertencem às posições entre 1 e $(j-1)$ da seqüência serão colocados na posição j . À essa seqüência adiciona-se uma seqüência derivada da seqüência inicial para as ordens que não ficaram nas posições entre 1 e $(j-1)$. Tendo a seqüência completa, calcula-se o valor da função objetivo e apenas as que possuírem os b menores valores são retidas para serem arborescidas no nível $(j+1)$.

Um tratamento mais formal e detalhado desse método será fornecido no capítulo 4, incluindo inclusive uma variação do mesmo chamada *beam search* filtrado.

Referências bibliográficas do método são OW; MORTON (1988 e 1989) e MORTON; PENTICO (1993).

EXEMPLO NUMÉRICO (item 1.1): A sequência inicial a ser utilizada será a sequência definida pela função de prioridade EXP-ET, ou seja, a sequência A,B,D,E,C. O primeiro nível formará n sequências com as n possíveis ordens na primeira posição e com o restante da sequência formado pelas ordens da sequência inicial, sem a ordem que foi para a primeira posição. A Tabela 8 apresenta as sequências do primeiro nível com os respectivos valores da função objetivo. Os valores hachurados das funções objetivo são os menores encontrados no nível em análise. Neste exemplo a largura de busca considerada é $b=2$.

Seqüência	$\Sigma(h_i E_i + w_i T_i)$
<u>A</u> ,B,D,E,C	119
<u>B</u> ,A,D,E,C	99
D,A,B,E,C	196
E,A,B,D,C	266
C,A,B,D,E	247

Tabela 8: sequências analisadas no primeiro nível do beam search - exemplo do centro de usinagem

Como $b=2$, deve-se reter as duas sequências cujos valores das funções objetivo sejam menores. No segundo nível, para as duas sequências retidas, deve-se manter a primeira ordem – ordem sublinhada na Tabela 8 – e testar todas as ordens diferentes da primeira na segunda posição da sequência, conforme a Tabela 9.

Nível 2			
Seqüência	$\Sigma(h_i E_i + w_i T_i)$	seqüência	$\Sigma(h_i E_i + w_i T_i)$
<u>B</u> ,A,D,E,C	99	<u>A</u> ,B,D,E,C	119
B,D,A,E,C	153	A,D,B,E,C	168
B,E,A,D,C	221	A,E,B,D,C	232
B,C,A,D,E	199	A,C,B,D,E	210
Nível 3			
Seqüência	$\Sigma(h_i E_i + w_i T_i)$	seqüência	$\Sigma(h_i E_i + w_i T_i)$
<u>B</u> ,A,D,E,C	99	<u>A</u> ,B,D,E,C	119
B,A,E,D,C	161	A,B,E,D,C	181
B,A,C,D,E	171	A,B,C,D,E	191
Nível 4			
Seqüência	$\Sigma(h_i E_i + w_i T_i)$	seqüência	$\Sigma(h_i E_i + w_i T_i)$
<u>B</u> ,A,D,E,C	99	<u>A</u> ,B,D,E,C	119
B,A,D,C,E	170	A,B,D,C,E	190

Tabela 9: sequências analisadas no beam search - exemplo do centro de usinagem

No terceiro e no quarto níveis, o procedimento é análogo. A seqüência sublinhada é a que será mantida fixa no próximo nível da busca. Os valores hachurados são os dois menores valores da função-objetivo encontrados no nível indicado.

A busca é finalizada com a seqüência sendo equivalente a B,A,D,E,C pois forneceu o menor valor da função objetivo no último nível da busca, ou seja, $\Sigma(h_j E_j + w_j T_j) = 99$.

2.5 PRINCIPAIS RESULTADOS DO PROBLEMA COM DATAS MÚLTIPLAS E COM PENALIDADES INDIVIDUALIZADAS PARA ADIANTAMENTO E PARA ATRASO

O problema que trata de custos no adiantamento e no atraso, apesar de ser relativamente recente possui uma vasta bibliografia disponível. Devido a esse fator, o presente tópico será restrito à classe de problemas com datas múltiplas e com penalidades individuais para adiantamento e para atraso.

A primeira publicação que se tem notícia acerca do problema $1|W_j|\Sigma(h_j E_j + w_j T_j)$ – máquina única, com penalidades distintas de adiantamento e de atraso, com datas múltiplas e com possibilidade de inserção de ociosidade – é devida à FRY *et al.* (1987). Embora FRY *et al.* façam todo o desenvolvimento teórico da heurística proposta com penalidades para cada ordem h_j e w_j , os testes computacionais são apresentados com um único valor para todas as ordens, h e w . Não há comentários sobre a razão dessa **simplificação**. Com o procedimento proposto por eles, encontrou-se ótimos globais em diversas instâncias testadas e o comportamento médio em todo estudo foi menor do que 2% acima do ótimo global.

A proposta de FRY *et al.* é uma heurística do tipo **troca de pares adjacentes**, com a possibilidade da **inserção de ociosidade** através da utilização da **programação linear**. Pelas características do problema, o procedimento de programação linear pôde ser resolvido em tempo polinomial, através de um procedimento de um passo.

Os primeiros a provar que o problema $1|W_j|\Sigma(E_j + T_j)$ – máquina única com adiantamento e atraso, com datas múltiplas e com possibilidade de inserção de

ociosidade – é **NP-difícil**¹² foram GAREY *et al.* (1988). Para dois casos particulares, GAREY *et al.* apresentaram um **algoritmo** com tempo $O(n \log n)$ para a **inserção de ociosidade**. O primeiro caso em que um algoritmo polinomial pode ser usado é para uma **seqüência predefinida** e o segundo é para o caso em que todos os **tempos de processamento** são **iguais** ($p_j=p, \forall J_j (j=1,...,n)$). Os autores também apresentaram três propostas de generalização do algoritmo, sendo que uma delas é para o caso onde existam penalidades iguais para cada diferença, isto é, o problema $1|W_j|\Sigma w_j(E_j+T_j)$ – máquina única, com penalidades únicas para cada ordem, com datas múltiplas e com possibilidade de inserção de ociosidade.

ABDUL-RAZAQ; POTTS (1988) analisaram o problema $1|\Sigma(h_jE_j+w_jT_j)$ – máquina única, com penalidades distintas de adiantamento e de atraso e com datas múltiplas –, utilizando a **programação dinâmica** e o **branch-and-bound**. Inicialmente os autores apresentaram **3 maneiras** de se definir **limitantes inferiores** através da utilização de uma relaxação de estado-espço na programação dinâmica¹³. Em seguida, formularam um **procedimento ótimo** utilizando o **branch-and-bound** com seqüenciação para frente, com um limitante superior sendo definido por uma heurística de um passo e com os limitantes inferiores conforme uma das 3 propostas.

Não houve nenhuma indicação forte de que um método era superior a outro em todos os aspectos. ABDUL-RAZAQ; POTTS concluem que a programação dinâmica deve ser utilizada na resolução de problemas de **tamanho pequeno** – até 10 ordens –, enquanto que para problemas de **tamanho médio** – até 25 ordens –, deve-se utilizar o **branch-and-bound** com os limitantes inferiores sendo definidos pela programação dinâmica.

YANO; KIM (1991) analisaram o problema $1|W_j|\Sigma(h_jE_j+w_jT_j)$ – máquina única, com penalidades distintas de adiantamento e de atraso, com datas múltiplas e

¹² As provas com relação à NP-completude ou NP-dificuldade de problemas indicam a improvável existência de algoritmos que resolvam instâncias de tamanho razoável em tempo hábil, justificando a utilização de heurísticas na resolução das mesmas. Assim, para o entendimento da bibliografia disponível sobre o assunto, é importante a compreensão dos principais conceitos e terminologias da **complexidade computacional**. Um breve resumo do tema é apresentado no anexo B.

¹³ NILSSON (1971, p.4) define o que pode ser caracterizado como um estado e como um espaço em métodos de solução de problemas.

com possibilidade de inserção de ociosidade – e forneceram um outro **procedimento de inserção de ociosidade**. Nesse procedimento, os autores utilizaram uma formulação com **fórmulas recursivas**, equivalentes às da programação dinâmica. Além disso, os autores assumem que $w_j \geq h_j \geq 0$, permitindo que a inserção de ociosidade seja feita em tempo $O(n^2 \log n)$. YANO; KIM também discutem alguns casos especiais como o caso proporcional – comentado adiante.

Os autores utilizaram o **branch-and-bound** com seqüenciação de trás para frente e já considerando a inserção de ociosidade no limitante inferior para verificar o desempenho de quatro **regras de despacho** e uma **heurística de troca de pares adjacentes**, com tempo $O(n^2)$. Este último procedimento é parecido com o utilizado por FRY *et al.* (1987) e utiliza como seqüência inicial àquela definida pela regra de despacho com o melhor desempenho dentre as quatro testadas. Nos cinco casos, após a definição da seqüência final, utiliza-se o procedimento para a inserção de ociosidade e elaboração do programa. Os resultados indicaram que em média, a heurística ficou apenas a 0,1% do ótimo¹⁴. Os autores finalizam o trabalho apontando que haveria uma necessidade de se criar melhores limitantes inferiores do que os que foram utilizados no estudo.

As **penalidades que dependem dos tempos de processamento** das ordens – por exemplo $h_j = \phi p_j$ e $w_j = \psi p_j$ sendo ϕ, ψ constantes de proporcionalidade de adiantamento e de atraso respectivamente – são chamadas de proporcionais e foram estudadas por RACHAMADUGU (1995). O problema completo pode ser definido como $1|W_j|\sum p_j(\phi E_j + \psi T_j)$ – máquina única, com penalidades distintas para adiantamento e para atraso e proporcionais ao tempo de processamento, com datas múltiplas e com possibilidade de inserção de ociosidade. O autor identificou um **caso especial** no qual a folga das ordens, $l_j = \max(0, d_j - t - p_j)$ ($j=1, \dots, n$), é uma função não decrescente do tempo de processamento. As datas deveriam ser definidas de modo que essa condição fosse obtida, ou seja, se $p_j = p_i + \Delta t$ – para Δt um intervalo de tempo – então, como $l_j \geq l_i$, $d_j \geq d_i + \Delta t$ ($j=1, \dots, n$; $i=1, \dots, n$ e $j \neq i$). O significado da equação é que em um dado instante t , todas as relações $d_j - p_j$ ($j=1, \dots, n$) entre as ordens será não decrescente. Se essa hipótese

¹⁴ Deve-se levar em consideração que as instâncias com tempo de solução superior a uma hora tiveram o procedimento interrompido antes da confirmação da otimização global.

for verdadeira, a **seqüência ótima** é definida por SPT (menor tempo de processamento) enquanto que o programa é encontrado inserindo-se os tempos de ociosidade com a utilização do procedimento proposto por FRY *et al.* (1987).

A Tabela 10 resume as principais características dos problemas com custos individuais de adiantamento e de atraso apresentadas nos artigos discutidos anteriormente. Os artigos de OW; MORTON serão discutidos no capítulo seguinte por terem um maior relacionamento com o *beam search*.

TIPO DE PROBLEMA	PROPOSTA	RESULTADOS	LIMITAÇÕES DA PROPOSTA	AUTOR
$1 W_j \Sigma(h_jE_j+w_jT_j)$	<ul style="list-style-type: none"> Utilização de um procedimento tipo troca de pares adjacentes para a seqüenciação. O programa é feito através de um procedimento que utiliza a programação linear. 	<ul style="list-style-type: none"> Comportamento médio de 2% acima do ótimo global. 	<ul style="list-style-type: none"> Os testes computacionais foram feitos para $h_j=h$ e $w_j=w, \forall J_j$. 	FRY <i>et al.</i> , 1987.
$1 W_j \Sigma w_j(E_j+T_j)$	<ul style="list-style-type: none"> Fornece um algoritmo de inserção de ociosidade com tempo $O(n \log n)$ para $p_j=p$ ou para uma seqüência predefinida. Prova a NP-dificuldade do problema de otimização $\min \Sigma(E_j+T_j)$. 			GAREY <i>et al.</i> , 1988.
$1 \Sigma(h_jE_j+w_jT_j)$	<ul style="list-style-type: none"> Fornece 3 procedimentos para a definição de limitantes inferiores. Fornece um procedimento para a determinação de limitantes superiores. Fornece um procedimento ótimo utilizando o <i>branch-and-bound</i>. 	<ul style="list-style-type: none"> Problemas de até 10 ordens devem ser resolvidos com a programação dinâmica. Problemas de até 25 ordens devem ser resolvidos com o <i>branch-and-bound</i>. 	<ul style="list-style-type: none"> Apesar de haver indícios de que os dados de entrada exercem grande influência nos resultados, não foram realizados testes extensivos nesse sentido. 	ABDUL-RAZAQ; POTTS, 1988.
$1 \Sigma(h_jE_j+w_jT_j)$ Será comentado no item 3.4.	<ul style="list-style-type: none"> Fornece 4 procedimentos heurísticos de solução do problema. 	<ul style="list-style-type: none"> O melhor procedimento heurístico ficou entre 1% e 16% do limitante inferior nas instâncias com 25 ordens. 	<ul style="list-style-type: none"> O custo de adiantamento foi de aproximadamente 25%, 10% e 5% do custo de atraso em todas as instâncias. 	OW; MORTON, 1988. OW; MORTON, 1989.
$1 W_j \Sigma w_j(E_j+T_j)$	<ul style="list-style-type: none"> Utilização de uma heurística do tipo troca de pares adjacentes para a seqüenciação. Fornece um algoritmo de inserção de ociosidade com tempo $O(n^2 \log n)$ para uma seqüência predefinida e supondo $w_j \geq h_j \geq 0$. Fornece um procedimento para definição de um limitante inferior. 	<ul style="list-style-type: none"> Heurística com comportamento médio 0,1% acima dos ótimos encontrados. 	<ul style="list-style-type: none"> Testes computacionais com $h/p_j=1$ e $w_j/p_j=3$ a 9. Não há um bom detalhamento do estudo computacional nem da aplicação do algoritmo de inserção de ociosidade. Reportam o leitor para KIM; YANO (1986) <i>apud</i> YANO; KIM (1991). 	YANO; KIM, 1991.
$1 W_j \Sigma p_j(\phi E_j+\psi T_j)$	<ul style="list-style-type: none"> Fornece um procedimento com tempo $O(n \log n)$ para o caso em que l_j é não decrescente com o p_j. Menciona a hipótese da inserção de ociosidade. 			RACHAMADUGU, 1995.

Tabela 10: quadro comparativo dos resultados dos problemas com custos individuais de atraso e de adiantamento

3 BEAM SEARCH

A técnica de busca BS – *beam search* – já está inserida em livros básicos de áreas como programação da produção (MORTON; PENTICO, 1993) e inteligência artificial (WINSTON, 1992). O BS pode ser considerado uma mistura de técnicas de busca, com a finalidade de aumentar a rapidez da solução. Combina as técnicas *best-first search* com *breadth-first search*, buscando apenas os **melhores nós** de uma árvore de busca – como no *best-first search* – e atuando **nível a nível** sem retornar a níveis superiores – como no *breadth-first search*. A vantagem produzida na eficiência computacional traz o inconveniente da falta de garantia da localização do ótimo global.

Existem variações como o BSF – *beam search* filtrado – e propostas de variações como o BS com largura variável. Neste capítulo serão abordados apenas o BS convencional e o BSF. Todo o desenvolvimento estará sendo baseado em um procedimento de seqüenciação. Posteriormente será apresentada uma revisão bibliográfica da utilização do BS em problemas de PS.

3.1 BEAM SEARCH

De uma maneira direta e simplificada, pode-se dizer que o BS é uma técnica de busca que utiliza um número de soluções em paralelo em uma árvore de busca. O número de soluções em paralelo é chamado de **largura da busca** e é denotado por b .

Para a apresentação do método, será utilizado um exemplo onde existem 5 ordens para serem seqüenciadas J_j ($j=D,B,A,C,E$). A largura de busca desejada é $b=2$. O procedimento é iniciado encontrando-se uma seqüência de acordo com alguma regra de despacho – uma função de prioridade também poderia ser utilizada – como por exemplo menor data de entrega (EDD).

Utilizando a regra de despacho, a seqüência inicial é definida por A,B,C,D,E. A seqüência inicial será considerada o limitante superior. O primeiro nível da árvore de busca é gerado através de uma enumeração completa das ordens que possam estar na primeira posição da seqüência. Como no caso aqui estudado supõe-se que todas as ordens estejam disponíveis para seqüenciação no instante inicial, as ordens disponíveis

para estar na primeira posição são A,B,C,D e E. A Figura 3 apresenta o primeiro nível da árvore de busca no BS.

Figura 3: primeiro nível da árvore de busca do beam search

Para a avaliação dos nós, utiliza-se o valor das funções objetivo obtidas em cada nó. Mas, como o valor das funções-objetivo só podem ser calculados utilizando-se seqüências completas, as outras ordens, além das que estão na primeira posição – pertencentes ao primeiro nível da árvore de busca – devem ser seqüenciadas. A seqüência dessas outras ordens é obtida retirando-se da seqüência inicial a ordem que foi para a primeira posição.

Por exemplo, a seqüência representada pelo nó C, é aquela que é formada pela ordem C na primeira posição com o restante da seqüência sendo definido pela seqüência inicial sem a ordem C, ou seja, a seqüência C,A,B,D,E. O nó B representa a seqüência B,A,C,D,E e assim sucessivamente para os outros nós.

Tendo as cinco seqüências do primeiro nível completas, pode-se avaliar os valores das funções objetivo. Como nesse caso $b=2$, apenas os nós que representam os dois menores valores das funções-objetivo serão arborescidos no segundo nível. No exemplo em questão, os nós que representam os menores valores das funções objetivo são os nós B e E. Os demais nós representados pelas linhas tracejadas da Figura 4 são eliminados definitivamente da busca.

Figura 4: segundo nível da árvore de busca do beam search

Os nós criados no segundo nível serão apenas os filhos dos nós mantidos no primeiro nível. Para ν um certo nível da árvore de busca – diferente dos níveis 0 e 1 –, o número de nós criados em cada nível é equivalente a $b(n-\nu+1)$. Portanto, no exemplo em análise, cada nó mantido no primeiro nível irá gerar outros quatro nós no segundo nível, perfazendo um total de 8 nós. A Figura 4 mostra como a árvore é gerada no segundo nível. Cada um dos nós será representado pela primeira ordem do primeiro nível, uma enumeração completa das ordens que possam estar no segundo nível e o restante da seqüência completa derivando da seqüência inicial. Por exemplo, o nó EC representa a seqüência ECABD, isto é, a seqüência definida pelo nó pai – nó E –, uma das ordens disponíveis para serem inseridas na segunda posição – qualquer uma, excetuando-se a ordem E – e o restante da seqüência sendo definido pela seqüência inicial sem os elementos que já foram seqüenciados.

As funções-objetivo são avaliadas como no primeiro nível e apenas os nós que obtiverem os dois menores valores do segundo nível são mantidos para arborescência no terceiro nível. Esse procedimento é realizado até o último nível da árvore de busca. A Figura 5 mostra a árvore completa, sendo que as partes tracejadas são os nós que foram eliminados no decorrer do desenvolvimento do procedimento. A seqüência escolhida no nível 5 e que é a solução do problema, é a que permite um menor valor da função objetivo entre as seqüências B,C,D,E,A e E,A,B,D,C.

Figura 5: árvore de busca utilizando o beam search

3.2 BEAM SEARCH FILTRADO (BSF)

O filtro aplicado no BS teve sua origem nos trabalhos apresentados por OW; MORTON (1988 e 1989). O filtro pode ser definido como uma **função de avaliação prévia para o BS**, transformando o BS em um método de busca de dois estágios. Isso faz **o tempo computacional diminuir** na análise de viabilidade de uma alternativa na árvore de busca. Além disso, conforme os estudos de OW; MORTON, pode-se inferir que o BS consegue **melhores resultados** com a utilização de um filtro adequado.

Na PS o filtro pode ser representado por uma função de prioridade da ordem J_j no horário t , $\phi_j(t)$. Dependendo da função objetivo que se esteja trabalhando, essa função de prioridade também pode ser uma regra de despacho. Neste trabalho, será utilizada a função de prioridade denominada EXP-ET presente nos estudos de OW; MORTON e explicada detalhadamente no próximo subitem.

A idéia do filtro é direcionar a busca para os nós que aparentemente são mais promissores. A função de avaliação do filtro faz uma **análise local** das ordens ainda disponíveis para serem seqüenciadas, ou seja, não há necessidade de se gerar seqüências completas como no BS sem o filtro. O filtro elimina as ordens menos promissoras, permitindo que apenas f nós filhos sejam gerados a partir de cada nó pai. O f é denominado **largura do filtro**. Após o filtro ter eliminado as ordens menos promissoras, as seqüências completas são geradas e as funções-objetivo – **análise global** – são calculadas. Posteriormente os b melhores nós são mantidos para arborescência no nível subsequente. O filtro é utilizado novamente para restringir o número de nós expandidos. A busca continua dessa maneira até que a seqüência completa de menor valor da função objetivo seja escolhida no último nível da árvore de busca.

Utilizando o mesmo exemplo do subitem anterior, o BS e o BSF são equivalentes até a determinação da seqüência inicial. A largura do filtro em consideração é $f=3$. Antes da geração do primeiro nível, a função de avaliação prévia – função do filtro – é utilizada para determinar as ordens que devem ficar na primeira posição e portanto os nós que devem ser gerados. No caso em análise, o filtro elimina as ordens A e D da primeira posição e portanto apenas os nós B,C e D – pois $f=3$ – são gerados.

Figura 6: primeiro nível da árvore de busca do beam search filtrado

Figura 7: segundo nível da árvore de busca do beam search filtrado

Figura 8: árvore de busca utilizando o beam search filtrado

A Figura 6 mostra como fica o primeiro nível da árvore de busca, com a utilização do filtro. Apenas os três nós representados têm o valor da função-objetivo analisado. Após a análise da função objetivo, o nó C é eliminado pois possui o maior valor dentre os três nós. Mais uma vez, as ordens disponíveis para serem sequenciadas

têm seu valor analisado pela função do filtro e as ordens mais promissoras – ordens A,C e E para o nó pai B e A,C e D para o nó pai E – podem formar nós filhos.

Esse procedimento é continuado até o último nível da árvore de busca e o resultado obtido é apresentado na Figura 8. Os nós que não aparecem - comparando-se com a Figura 5, p.38 - são os que foram eliminados pelo filtro. Os nós pontilhados são os que foram eliminados pelo valor da função objetivo. Deve-se perceber que o número máximo de nós gerados em cada nível é bf .

Maiores detalhes desse procedimento podem ser obtidos diretamente do código computacional em Turbo-Pascal, disponibilizado no apêndice 1 com o nome BSF.PAS.

3.3 FUNÇÃO DE PRIORIDADE EXP-ET

Como mencionado anteriormente no item 2.4.2, uma função de prioridade é um procedimento que utiliza informações das ordens para definir qual deve ser processada subsequente e se a mesma deve ter ociosidade inserida, antes de iniciar o seu processamento. Especificamente, a função de prioridade EXP-ET ($\varphi_j(t)$) pode ser definida como sendo:

$$\varphi_j(t) = \begin{cases} \frac{w_j}{p_j} \exp\left(-\frac{(h_j + w_j)}{h_j} \frac{\max(0, d_j - t - p_j)}{p_{med}}\right) & \text{se } d_j - t - p_j \leq \frac{w_j k p_{med}}{h_j + w_j} \\ \frac{h_j}{p_j} \left(\frac{w_j}{h_j} - \frac{(h_j + w_j)}{h_j} \frac{\min(k p_{med}, d_j - t - p_j)}{k p_{med}} \right)^3 & \text{para outros casos,} \end{cases}$$

onde k é um **parâmetro de ajuste** que determina o horário em que a função de prioridade começa crescer e p_{med} é o tempo de processamento médio das ordens que ainda não foram seqüenciadas, $p_{med} = |R_q^v|^{-1} \sum_{j \in R_q^v} p_j$, para R_q^v o conjunto de ordens que ainda não foram seqüenciadas no nível v e cujo nó pai é o nó q ($q \in \mathbb{N}$ e $1 \leq q \leq b$).

Um esquema gráfico da função é mostrado na Figura 9. A função tem um valor negativo constante, $-h_j/p_j$ (equivalente ao negativo do valor obtido pela regra de despacho WLPT), até faltar um tempo equivalente ($k p_{med} + p_j$) para d_j . No intervalo $k p_{med}$, o valor da prioridade se altera, passando de negativo para positivo. Quando faltar um tempo exatamente igual a p_j para a data de entrega desejada, a prioridade se transforma em w_j/p_j - equivalente ao valor obtido pela regra de despacho WSPT.

Figura 9: significado gráfico da função de prioridade EXP-ET ($k=5$)

A utilização da função determina que, a ordem que apresentar o maior valor numérico dentre todas as disponíveis deve ser seqüenciada subsequentemente. Então, a função de prioridade possui dois casos especiais:

Caso 1: se todas as ordens estiverem adiantadas por um período maior do que $(kp_{med} + p_j)$, a função de prioridade as seqüencia por WLPT. Como para WLPT a seguinte seqüência é verdadeira $(h_{\sigma(1)}/p_{\sigma(1)}) \leq (h_{\sigma(2)}/p_{\sigma(2)}) \leq \dots \leq (h_{\sigma(n)}/p_{\sigma(n)})$, a multiplicação por (-1) também permite que a seqüência seja verdadeira $(-h_{\sigma(1)}/p_{\sigma(1)}) \geq (-h_{\sigma(2)}/p_{\sigma(2)}) \geq \dots \geq (-h_{\sigma(n)}/p_{\sigma(n)})$. O lema 2 de OW; MORTON (1989) afirma que se a seqüenciação por WLPT resultar em uma seqüência onde não haja nenhuma ordem atrasada, então essa é a seqüência ótima;

Caso 2: se todas as ordens forem se atrasar por um período maior do que 0, a função de prioridade as seqüencia por WSPT. O lema 1 de OW; MORTON (1989) afirma que se a seqüenciação por WSPT resultar em uma seqüência onde não haja nenhuma ordem adiantada, então essa é a seqüência ótima.

Em condições diferentes das duas mencionadas acima, a ordem com o maior valor numérico da função de prioridade é a mais prioritária e portanto a próxima a ser seqüenciada. A função EXP-ET não garante nem ótimos locais embora forneça bons resultados.

A Figura 9 representa como a função de prioridade varia no decorrer do tempo para duas ordens. Deve-se perceber que uma ordem J_j que no horário t era menos prioritária que J_i , pode se tornar mais prioritária do que J_i no horário $t+\Delta t$, onde Δt é um intervalo de tempo.

3.4 EXEMPLO NUMÉRICO

Abaixo será apresentada a Tabela 11 contendo os dados relativos a um conjunto de ordens disponíveis para serem seqüenciadas. Inicialmente deverá ser estabelecida a seqüência inicial para só então se utilizar o BSF.

A **seqüência inicial** será definida de acordo com a função de prioridade EXP-ET no modo despacho, ou seja, sem a inserção de ociosidade.

A função EXP-ET será utilizada após o término de uma ordem ou no instante 0 para o passo inicial. Assume-se que $k=5$. Em cada passo, deve-se reter a ordem cujo valor calculado da função prioridade é máximo. A seqüência com suas respectivas prioridades máximas nos horários t é dada pela Tabela 12.

j	p_i	d_i	h_i	w_i
1	9	23	16	11
2	14	31	9	14
3	11	47	2	14
4	13	55	11	5
5	4	84	17	13
6	1	128	6	7
7	11	145	12	13
8	12	173	9	7
9	15	185	2	19
10	19	197	13	0
11	6	390	2	17
12	6	396	14	17
13	12	464	1	8
14	16	535	8	11

Tabela 11: dados da resolução numérica do BSF

Portanto a seqüência inicial está definida na coluna j . Nesse caso, pode-se derivar a Tabela 13 que representa os adiantamentos ou atrasos de cada ordem de modo que se possa calcular o valor da função objetivo.

j	t	$\phi(t)$
1	0	0,1328
2	9	0,1498
3	23	0,0001
4	34	0,1264
13	47	-0,0833
5	59	0,0799
9	63	-0,1333
11	78	-0,3333
14	84	-0,5000
6	100	-0,0001
7	101	-0,0003
8	112	-0,1919
10	124	-0,4413
12	143	-2,3333

Tabela 12: seqüência EXP-ET, horários de programação e prioridades máximas

O valor alcançado pela **função-objetivo** é equivalente a: $\Sigma(h_j E_j + w_j T_j) = 10637$.

j	p_i	d_i	C_i	L_i	E_i	T_i	$h_j E_j + w_j T_j$
1	9	23	9	-14	14	0	224
2	14	31	23	-8	8	0	72
3	11	47	34	-13	13	0	26
4	13	55	47	-8	8	0	88
13	12	464	59	-405	405	0	405
5	4	84	63	-21	21	0	357
9	15	185	78	-107	107	0	214
11	6	390	84	-306	306	0	612
14	16	535	100	-435	435	0	3480
6	1	128	101	-27	27	0	162
7	11	145	112	-33	33	0	396
8	12	173	124	-49	49	0	441
10	19	197	143	-54	54	0	702
12	6	396	149	-247	247	0	3458

Tabela 13: resultados obtidos pela EXP-ET no exemplo numérico

Essa seqüência pode ser utilizada como **seqüência inicial** no BSF, ou seja, poderá ser utilizada como um limitante superior da busca. O BSF utilizado terá $f=3$ e $b=2$.

Para a solução do exemplo, será utilizada a seguinte notação: v = nível da árvore de busca; q = nó pai do nó em análise (no exemplo haverá hipótese de se ter apenas dois nós pais pois $b=2$); j = nó em análise; $C_{s_q^{v-1}}$ = horário de término do

processamento da sequência parcial fixa (sequência fixa s_q , formada pelas ordens até o nível $\nu-1$) representada pelo nó pai q ; $g(\sigma)$ = valor da função objetivo da sequência completa, formada pela sequência parcial fixa (representada pelo nó pai), a ordem j e a sequência parcial complementar (formada pela sequência inicial, excluindo-se a ordem j e as ordens presentes na sequência parcial fixa). Com base nessa notação, pode-se criar a seguinte tabela:

ν	q	j	$C_{s_q^{\nu-1}}$	$\varphi_j(C_{s_q^{\nu-1}})$	$g(\sigma)$
1	1	1	0	0,1328	10637
1	1	2	0	0,0169	10494
1	1	3	0	0,0000	10477

Tabela 14: resultados obtidos pelo BSF no nível 1

ν	q	j	$C_{s_q^{\nu-1}}$	$\varphi_j(C_{s_q^{\nu-1}})$	$g(\sigma)$	ν	q	j	$C_{s_q^{\nu-1}}$	$\varphi_j(C_{s_q^{\nu-1}})$	$g(\sigma)$
2	1	1	14	1,222	10494	8	1	5	80	3,25	10136
2	1	3	14	0,000	10633	8	1	10	80	-0,6842	10290
2	1	4	14	-0,030	10831	8	1	14	80	-0,50	10376
2	2	1	11	0,7586	10477	8	2	5	80	3,25	10118
2	2	2	11	0,2359	10562	8	2	10	80	-0,6842	10272
2	2	4	11	-0,052	10974	8	2	14	80	-0,5	10358
3	1	3	23	0,0001	10494	9	1	7	84	-0,646	10186
3	1	4	23	-0,0003	10589	9	1	10	84	-0,6842	9991
3	1	13	23	-0,083	10426	9	1	14	84	-0,5	10136
3	2	2	20	1,000	10477	9	2	7	84	-0,646	10168
3	2	4	20	-0,0024	10813	9	2	10	84	-0,6842	9973
3	2	13	20	-0,08	10604	9	2	14	84	-0,5	10118
4	1	3	35	0,5882	10426	10	1	6	103	0,0246	10079
4	1	4	35	0,144	10649	10	1	7	103	-0,036	10080
4	1	9	35	-0,133	10496	10	1	14	103	-0,5	9991
4	2	4	34	0,1264	10477	10	2	6	103	0,0246	10061
4	2	9	34	-0,13	10212	10	2	7	103	-0,036	10062
4	2	13	34	-0,08	10422	10	2	14	103	-0,5	9973
5	1	4	49	0,3846	10212	11	1	6	119	0,6941	9991
5	1	5	49	-0,151	10653	11	1	7	119	0,0183	9976
5	1	13	49	-0,08	10285	11	1	8	119	-0,75	9935
5	2	4	46	0,3846	10422	11	2	6	119	0,6941	9976
5	2	5	46	-0,2727	10663	11	2	7	119	0,0183	9958
5	2	9	46	-0,133	10276	11	2	8	119	-0,075	9917
6	1	5	62	0,1216	10412	12	1	6	131		9935
6	1	11	62	-0,33	10136	12	1	7	131		10024
6	1	13	62	-0,08	10212	12	1	12	131		10173
6	2	4	61	0,3846	10276	12	2	6	131		9917
6	2	5	61	0,1053	10517	12	2	7	131		10006
6	2	11	61	-0,33	10238	12	2	12	131		10155
7	1	5	68	0,4014	10336	13	1	7	132		9935
7	1	13	68	-0,08	10136	13	1	12	132		10117
7	1	14	68	-0,500	10456	13	2	7	132		9917
7	2	5	74	1,0505	10212	13	2	12	132		10099
7	2	11	74	-0,33	10118	14	1	12	143		9935
7	2	14	74	-0,50	10288	14	2	12	143		9917

Tabela 15: passos intermediários e resultados da solução pelo BSF

Portanto as **seqüências retidas** em $v=1$ serão as seqüências $\langle 2,1,3,4,13,5,9,11,14,6,7,8,10,12 \rangle$ e $\langle 3,1,2,4,13,5,9,11,14,6,7,8,10,12 \rangle$. Os valores das funções objetivo das seqüências retidas estão hachurados.

Para os demais níveis, o procedimento é análogo e um resumo dos passos intermediários é apresentado na Tabela 15. Note que para os níveis 12, 13 e 14 não há mais necessidade de se calcular a função de prioridade pois como só existem 3, 2 e 1 ordens respectivamente, todas passariam pelo filtro.

No final do procedimento, o valor alcançado pela função objetivo é equivalente a: $\Sigma(h_j E_j + w_j T_j) = 9917$, ou seja, um **valor inferior** a 10637 definido inicialmente pela seqüência inicial.

3.5 REVISÃO BIBLIOGRÁFICA

Os métodos de busca do tipo BS possuem vastas aplicações tanto no campo da Inteligência Artificial, como no campo da Administração e da Engenharia de Produção. Várias aplicações como o planejamento de *lay-out* (SHIH, 1991), a seleção e projeto de linhas de produtos (NAIR *et al.*, 1995) e a seleção de portfólios de projetos de P&D (COFFIN; TAYLOR III, 1996) são disponíveis. Devido à extensão bibliográfica sobre o assunto, será feita uma focalização nas áreas de programação e de seqüenciação.

O primeiro trabalho a comparar sistematicamente o BS com outros procedimentos heurísticos para resolver problemas de seqüenciação é devido a OW; MORTON (1988 e 1989)¹⁵. Além disso, esse trabalho foi o primeiro a apresentar a proposta para o BSF. O artigo apresenta uma comparação entre **4 procedimentos heurísticos** que combinam regras de despacho e funções de prioridade com BS.

Os problemas analisados foram 1| $\Sigma(h_j E_j + w_j T_j)$ – máquina única, com penalidades distintas de adiantamento e de atraso e com datas múltiplas –, F4| $\Sigma w_j T_j$ – *flow shop* de 4 máquinas, com penalidades de atraso e com datas múltiplas – e F8| $\Sigma w_j T_j$. No trabalho de 1989 só foi estudado o problema de uma máquina.

O melhor dos quatro procedimentos foi o *beam search* filtrado que foi explicado detalhadamente no item 3.2. O desempenho de todos esses procedimentos foi

¹⁵ Um pouco antes, FOX (1983) *apud* OW; MORTON (1988) havia aplicado o BS para resolver problemas de programação relativamente complexos em um sistema chamado ISIS.

medido através do custo obtido pela heurística e pela solução ótima – *branch-and-bound* para problemas pequenos – ou limitante inferior – nos problemas maiores, utilizando uma relaxação envolvendo a divisão da ordem em subordens e estas sendo resolvidas como um problema de programação linear –. No caso da máquina única, para as instâncias de 15 e de 25 ordens, os procedimentos ficaram respectivamente entre 1 e 7% e entre 1 e 16% acima do limitante inferior.

A função de prioridade utilizada no caso dos *flow shops* foi a IDLE (*Idle Time Rule*). De uma maneira simplificada, pode-se dizer que ela sequencia as ordens de acordo com sua folga com relação à data de entrega na máquina gargalo do *flow shop*.

Para cada horário t que o gargalo ficará livre, a função apresenta a ordem mais prioritária dentre todas as que ainda não foram sequenciadas. A **seqüência** elaborada é **permutacional**. Na seqüenciação do tipo permutacional não há inversão na seqüência das ordens no percurso pelas máquinas.

Para as instâncias testadas, o BS com **largura da busca e largura do filtro** de 5, teve um desempenho aproximadamente 4% melhor que a função de prioridade IDLE, quando a mesma é utilizada isoladamente.

A utilização do BS em um FMS (*Flexible Manufacturing System*) é proposta por CHANG *et al.* (1989). Além da seqüenciação de problemas do tipo $J|C_{max}$ – minimização do horário de término da última ordem em um *job shop* –, os autores propõem a possibilidade da **flexibilização de roteiros**, que é uma característica peculiar aos ambientes tipo FMS.

O procedimento proposto foi denominado BS baseado no gargalo e **analisa apenas a máquina gargalo**, de maneira semelhante ao problema de uma máquina. A largura da busca define a quantidade de limitantes superiores mantidos nos vários níveis, sendo que a seqüência remanescente – que ainda não foi sequenciada definitivamente – é determinada pela regra de despacho SPT.

Três experimentos foram realizados visando a comparação entre o BS baseado no gargalo e sete regras de despacho bastante conhecidas. De uma maneira geral, o procedimento proposto foi entre 5% e 50% superior à melhor regra de despacho testada sendo que o tempo de solução piorou.

Outro trabalho que utiliza o BS aplicado a um FMS é devido a SHIH; SEKIGUSHI (1991). O FMS é composto de 3 máquinas e 4 produtos, com

flexibilidade de roteiro e chegadas de ordens dinâmicas. Um aspecto interessante apresentado nesse trabalho é a introdução de um novo parâmetro de controle do método, denominado **profundidade da busca**. Como o sistema é de **tempo real**, a profundidade de uma busca completa define o intervalo de tempo entre decisões.

Cada decisão considera as ordens que estão na fila para serem processadas, mais as ordens que se esperam necessitar de processamento dentro do intervalo de tempo da decisão. A busca é processada com esses dados e a melhor seqüência de acordo com a função objetivo utilizada. Os autores analisaram o BS com relação às funções objetivo diferença absoluta, $\Sigma(E_j+T_j)$ e tempo de fluxo médio $(1/n)\Sigma_{j=1,n}F_j$.

Para o critério $\Sigma(E_j+T_j)$, o BS com $b=3$ foi comparado com duas regras de despacho conhecidas. Os resultados com o BS apresentaram melhor desempenho não somente no desempenho médio, mas também com relação à variância dos valores encontrados. Para o critério $(1/n)\Sigma_{j=1,n}F_j$, o BS com $b=3$ foi comparado com outras duas regras de despacho, obtendo resultados melhores, piores e equivalentes às regras. Isso mostra alguma **incoerência nos experimentos** pois contraria a 2ª proposição de OW; MORTON (1988). A proposição diz que se você utiliza uma regra de despacho para determinar um limitante superior, o BS deve ser no mínimo tão bom quanto esse limitante superior proporcionado pela regra de despacho utilizada.

Para uma determinada seqüência σ , DE *et al.* (1992) propuseram a aplicação do BS na **estimação de fronteiras** eficientes do problema com função objetivo de dois critérios do tipo $1||Z_\alpha(\sigma)$, onde $Z_\alpha(\sigma)=\alpha C_{med}(\sigma)+(1-\alpha)\text{Var}(\sigma)$, para $\{\alpha \in \mathbf{Q} | 0 \leq \alpha \leq 1\}$, $C_{med}(\sigma)=(1/n)\Sigma C_j(\sigma)$ e $\text{Var}(\sigma)=(1/n)\Sigma [C_j(\sigma)-C_{med}(\sigma)]^2$.

Na heurística proposta, o BS tem a função de **gerar seqüências parciais** para a elaboração de limitantes superiores e limitantes inferiores. Consegue-se a avaliação exata através de um procedimento ótimo utilizando a programação dinâmica. Encontrou-se que além de $b=4$, **não houve melhora** de desempenho da heurística.

SLOTNICK; MORTON (1996) analisaram o procedimento de escolha de ordens para serem processadas sabendo-se que existe um excesso das mesmas e que inevitavelmente algumas delas serão atrasadas. O problema possui uma única máquina e a chegada das ordens é estática. O atraso das ordens não processadas possui penalidades do tipo $w_j(C_j-d_j)$. A estratégia utilizada é procurar pelas **ordens que proporcionam um maior lucro líquido**, ou seja, $\max \Sigma U_j [Q_j - w_j(C_j - d_j)^+]$, onde $U_j = (1 \text{ ou } 0)$ – se a ordem é

aceita ou não após a data de entrega original –, $(x)^+ = \max(0, x)$ e Q_j é lucro proporcionado pela ordem J_j .

Apresentam-se 2 propriedades que melhoram a eficiência da busca desse tipo de problema em um procedimento ótimo e em dois procedimentos heurísticos – um BS e um míope¹⁶ – que foram testados. O procedimento ótimo é um *branch-and-bound*. Como o BS seria utilizado para a verificação do desempenho da heurística míope, ele inicialmente foi **comparado com o *branch-and-bound***. Os dados utilizados foram $b=17$ e $n=17$. Encontrou-se a solução ótima pelo BS em todas as instâncias testadas, com economias consideráveis nos tempos de solução.

A **heurística míope** foi testada em 400 instâncias do problema, 100 para cada $n \in \{12, 17, 22, 27\}$. Como comparação, as duas menores ($n=12$ e $n=17$) utilizaram o *branch-and-bound*, enquanto que as duas maiores utilizaram o BS com largura equivalente ao número de ordens do conjunto de ordens, isto é, $b=n$. O procedimento míope teve um desempenho excelente, divergindo em média apenas 1,3% acima das soluções do *branch-and-bound* e do BS.

Um problema um pouco diferente dos anteriormente abordados é o proposto por KOULAMAS (1996). É um problema de duas máquinas idênticas que possuem uma mesma fila de entrada de ordens e que utilizam um mesmo servidor (robô, no caso) para fazer a preparação das máquinas. O servidor pode atender apenas uma máquina de cada vez.

¹⁶ Procedimentos ou heurísticas míopes são aqueles que tentam obter uma boa solução apenas para o passo subsequente, sem se importar com os seguintes, ou seja, uma otimização local. Referências com relação a esse assunto são MORTON; PENTICO (1993, p.160) e CAMPELLO; MACULAN (1994, p.79).

TIPO DE PROBLEMA	PROPOSTA	CARACTERÍSTICAS	LIMITAÇÕES DA PROPOSTA	AUTOR
1 $ \Sigma(h_j E_j + w_j T_j)$ F4 $ \Sigma(w_j T_j)$ F8 $ \Sigma(w_j T_j)$	<ul style="list-style-type: none"> Propõe a utilização de uma função denominada filtro para avaliação preliminar da busca. 	<ul style="list-style-type: none"> Chegou-se a determinar Quais as larguras de busca e de filtro ideais para as instâncias em análise. 	<ul style="list-style-type: none"> Os <i>flow shops</i> testados são permutacionais. 	OW; MORTON, 1988.
1 $ \Sigma(h_j E_j + w_j T_j)$	<ul style="list-style-type: none"> Idem a OW; MORTON, 1988. 	<ul style="list-style-type: none"> Idem a OW; MORTON, 1988. 		OW; MORTON, 1989.
$J C_{\max}$	<ul style="list-style-type: none"> Utilização da flexibilização de roteiros além da seqüenciação convencional. Estudar a eficiência do BS quando comparado com regras de despacho. 	<ul style="list-style-type: none"> A proposta forneceu melhorias entre 5% e 50% da melhor regra de despacho testada, conforme o grau de flexibilização dos roteiros. Os tempos de solução foram proporcionais à b, entre $140b$ e $350b$. 	<ul style="list-style-type: none"> Foram testadas apenas 3 instâncias de dados padrão. As variações aconteceram na flexibilização de roteiros. 	CHANG <i>et al.</i> , 1989.
$J \Sigma(E_j + T_j)$ $J (1/n) \sum_{j=1,n} F_j$	<ul style="list-style-type: none"> Utilização de um parâmetro denominado profundidade da busca, que pode ser utilizado em sistemas de tempo real. 	<ul style="list-style-type: none"> O BS teve desempenho superior (ou muito perto) tanto na média Quanto na variância dos valores das funções objetivo quando comparado às regras de despacho utilizadas. 	<ul style="list-style-type: none"> O estudo é realizado utilizando-se apenas 4 produtos. Um dos experimentos mostra incoerência com relação à 2ª. proposição de OW; MORTON, 1988. 	SHIH; SEKIGUSHI, 1991.
1 $ Z_\alpha(\sigma)$ para $Z_\alpha(\sigma) = \alpha C_{\text{med}}(\sigma) + (1-\alpha)\text{Var}(\sigma)$	<ul style="list-style-type: none"> O BS gera seqüências parciais para a definição de limitantes inferiores e limitantes superiores. 	<ul style="list-style-type: none"> Nas instâncias testadas o procedimento teve um desempenho próximo do ótimo. 		DE <i>et al.</i> , 1992.
1 $ \max \Sigma U_j [Q_j - w_j (C_j - d_j)^+]$	<ul style="list-style-type: none"> O BS é utilizado para a avaliação de um procedimento míope. 	<ul style="list-style-type: none"> Em todos os testes realizados com instâncias de 12 e 17 ordens, o BS encontrou o ótimo global. 	<ul style="list-style-type: none"> Geralmente as decisões que maximizam são feitas nos níveis de decisão superiores. 	SLOTNICK; MORTON, 1996.
P2 $\text{res1} \Sigma(W1_j + W2_j)$	<ul style="list-style-type: none"> Utiliza o BS para a geração de programas parciais, simplificando o cálculo das ociosidades forçadas. 	<ul style="list-style-type: none"> O BS teve um desempenho superior tanto na média como no valor máximo quando comparado com a troca de pares adjacentes. 		KOULAMAS, 1996.

Tabela 16: quadro comparativo dos resultados da aplicação do beam search a problemas de programação e seqüenciação

O autor está interessado em **minimizar o tempo de ociosidade devido à indisponibilidade do servidor** – intersecção no intervalo de preparação das duas máquinas –, isto é, $\min \Sigma(W1_j + W2_j)$ sendo W_i o tempo de ociosidade forçado da máquina M_i . O problema completo pode ser definido como $P2|res1|\Sigma(W1_j + W2_j)$ – problema de duas máquinas paralelas idênticas, com restrição de um recurso, onde deseja-se minimizar o tempo forçado de ociosidade das máquinas.

O BS foi utilizado para a **geração de programas parciais**, sendo comparado com uma heurística de **troca de pares adjacentes**. Concluiu-se que o procedimento utilizando o BS obteve um tempo de solução computacional menor, além de um desempenho superior tanto na média como no valor máximo em todas as instâncias testadas.

A Tabela 16 resume as principais características dos artigos apresentados anteriormente neste subitem.

4 INSERÇÃO DE OCIOSIDADE

Problemas que levam em consideração o adiantamento, **devem necessariamente utilizar a inserção de ociosidade**. Caso não seja assim, uma das principais características desses problemas fica deixada de fora.

Mesmo assim, vários autores não trabalharam com a inserção de ociosidade (OW; MORTON, 1988 e 1989, ABDUL-RAZAQ; POTTS, 1988), provavelmente pelo fato de não se conhecer um algoritmo para tal. Permitindo a inserção de ociosidade como uma variável contínua, um conjunto com n ordens possui infinitos programas viáveis e não mais “apenas” $n!$.

O primeiro trabalho a apresentar um algoritmo para inserção de ociosidade pertence a FRY *et al.* (1987). De uma maneira bastante clara, os autores formularam o problema de inserção de ociosidade no problema de atraso e adiantamento com penalidades individuais, $\Sigma(h_j E_j + w_j T_j)$, como um problema de programação linear. Pela característica do tipo de problema, o mesmo pôde ser resolvido em tempo $O(n^2)$.

De uma maneira provavelmente independente, GAREY *et al.* (1988) também propuseram um algoritmo de inserção de ociosidade ótima no problema de atraso e adiantamento sem penalidades, $\Sigma(E_j + T_j)$, com tempo $O(n \log n)$. Para o caso da utilização de uma penalidade única para adiantamento e para atraso, $\Sigma w_j(E_j + T_j)$, os autores apenas indicam como o algoritmo deveria ser formulado.

Ambos os trabalhos supõem que os algoritmos sejam utilizados em uma **seqüência definida preliminarmente**. Uma diferença entre ambos está no fato de que o algoritmo de FRY *et al.* começa o programa a partir da **última ordem** da seqüência, enquanto que o de GAREY *et al.* começa a partir da **primeira ordem**.

Algum tempo depois, YANO; KIM (1991) propuseram um outro algoritmo de inserção de ociosidade baseado na programação dinâmica. A inserção de ociosidade também é feita após a seqüenciação com função objetivo $\Sigma(h_j E_j + w_j T_j)$ e é suposto que $w_j \geq h_j \geq 0$. O tempo de solução no pior caso é $O(n^2 \log n)$ demonstrando que a utilização desse algoritmo proporciona uma inserção de ociosidade com tempo menos eficiente do que os dois casos comentados anteriormente.

Nos itens subseqüentes deste capítulo serão descritas duas alternativas para se inserir ociosidade. A primeira alternativa é a função de prioridade EXP-ET com

inserção de ociosidade e a segunda é o algoritmo de GAREY *et al.* (1988) com sua respectiva adaptação para o caso da utilização de penalidades individualizadas para o adiantamento e para o atraso, ou seja, o caso $\Sigma(h_j E_j + w_j T_j)$ com tempo $O(n \log n)$.

4.1 INSERÇÃO DE OCIOSIDADE UTILIZANDO-SE A FUNÇÃO DE PRIORIDADE EXP-ET

Além de ser uma regra de despacho de alto desempenho para o problema de atraso e adiantamento, a função EXP-ET pode ser utilizada como uma função de prioridade.

Nos estudos de OW; MORTON (1988, 1989) não se fez referências acerca da possibilidade da inserção de ociosidade. Posteriormente, MORTON; PENTICO (1993, p.147) citam que a função de prioridade pode ser utilizada para se inserir ociosidade. Segundo eles, deve-se utilizar a função normalmente – conforme apresentado nos itens 2.4.2 e 4.3 – e quando todos os valores das prioridades forem negativos, não se programa nenhuma ordem, ou seja, insere-se ociosidade. Todavia, MORTON; PENTICO alertam sobre o **risco** da utilização da inserção de ociosidade dessa forma. Dependendo da instância, a inserção de ociosidade pode ser viável inicialmente, entretanto devido ao agrupamento – congestionamento – de datas de entrega futuras pode ser inviável.

Embora MORTON; PENTICO não citem, para problemas com distribuições de datas de entrega unimodais, esse problema poderia ser resolvido com uma escolha adequada do parâmetro de ajuste k . Quando determinado de forma correta, esse parâmetro de certa forma “prevê” o que irá acontecer no futuro, e, caso haja uma alta probabilidade de congestionamento, as prioridades aumentam antecipadamente.

Não se encontrou na literatura os intervalos de reavaliação caso todas as prioridades fossem negativas, isto é, **caso tenha-se um conjunto de prioridades negativas, não se sabe o instante em que pelo menos uma se tornará positiva**. Nesse instante, já vale a pena se programar a ordem mais prioritária. A proposição 3 adiante apresenta esse horário em que já vale a pena se programar a ordem.

Para a simplificação da prova da proposição, serão descritas algumas características da função EXP-ET. Essa função possui quatro intervalos distintos,

conforme apresentado na Figura 9, p.42. Os dois primeiros intervalos (observando-se da esquerda para a direita, a região onde a função é linear e a região onde a função é estritamente côncava) são os intervalos onde a função toma os valores negativos. Os dois últimos intervalos (a região onde a função é estritamente convexa e a região onde a função é linear) são onde a função assume valores positivos. Pela função EXP-ET isso pode ser confirmado pois, como w_j e p_j ($\forall j \in R_q^v$, onde R_q^v é conjunto de ordens disponíveis para seqüenciação) são positivos, o caso $(d_j - t - p_j) \leq w_j k p_{med} / (h_j + w_j)$ será sempre positivo.

Para os intervalos restantes, isto é, quando $(d_j - t - p_j) > w_j k p_{med} / (h_j + w_j)$, a função de prioridade é definida como

$$\varphi_j(t) = \frac{h_j}{p_j} \left(\frac{w_j}{h_j} - \frac{(h_j + w_j)}{h_j} \frac{\min(k p_{med}, d_j - t - p_j)}{k p_{med}} \right)^3,$$

onde pode-se dividi-la em dois intervalos:

Intervalo 1: $k p_{med} \leq d_j - t - p_j$ e portanto $t \leq d_j - p_j - k p_{med}$. Nesse intervalo, a função toma um valor constante equivalente a:

$$\varphi_j(t) = \frac{h_j}{p_j} \left(\frac{w_j}{h_j} - \frac{(h_j + w_j)}{h_j} \frac{k p_{med}}{k p_{med}} \right)^3 = -\frac{h_j}{p_j}.$$

Intervalo 2: $k p_{med} > d_j - t - p_j$ e portanto $t > d_j - p_j - k p_{med}$. Nesse intervalo, a função é equivalente a:

$$\varphi_j(t) = \frac{h_j}{p_j} \left(\frac{w_j}{h_j} - \frac{(h_j + w_j)}{h_j} \frac{(d_j - t - p_j)}{k p_{med}} \right)^3.$$

Se for provado que no intervalo 2, a função é estritamente crescente, o ponto de máximo de $\varphi_j(t)$ é o máximo t para o qual a função seja válida, t_j^* .

De acordo com JOHNSON; KIOKEMEISTER (1977, p.144), uma função $\varphi_j(t)$ contínua e pertencente a um intervalo I é estritamente crescente em I se $\frac{d}{dt}[\varphi_j(t)] > 0$ para cada t em I , onde t não é um ponto extremo do intervalo I . Então só nos falta verificar que $\frac{d}{dt}[\varphi_j(t)] > 0$.

PROPOSIÇÃO 1. A derivada de $\varphi_j(t)$ no intervalo $(d_j - p_j - kp_{med} < t < d_j - p_j - w_j kp_{med} / (h_j + w_j))$ é estritamente crescente.

DEMONSTRAÇÃO. Para $(d_j - p_j - kp_{med} < t < d_j - p_j - w_j kp_{med} / (h_j + w_j))$,

$$\frac{d}{dt}[\varphi_j(t)] = \frac{3}{h_j p_j} \left(w_j - \frac{(h_j + w_j)(d_j - t - p_j)}{kp_{med}} \right)^2 \left(\frac{h_j + w_j}{h_j kp_{med}} \right),$$

onde para h_j, w_j, p_j, k e p_{med} positivos, fica evidente que $\frac{d}{dt}[\varphi_j(t)] > 0$. ■

PROPOSIÇÃO 2. Para o intervalo $(d_j - p_j - kp_{med} < t < d_j - p_j - w_j kp_{med} / (h_j + w_j))$, $\varphi_j(t)$ não assume valores positivos $\forall j \in R_q^\nu$.

DEMONSTRAÇÃO. $\varphi_j(t) > 0$ se

$$\frac{w_j}{h_j} > \frac{(h_j + w_j)(d_j - t - p_j)}{h_j kp_{med}} \text{ pois } \frac{h_j}{p_j} > 0 \text{ e } \frac{w_j}{h_j} > 0.$$

Isolando-se o t , pode-se dizer que $\varphi_j(t) > 0$ se

$$t > d_j - p_j - \frac{w_j kp_{med}}{h_j + w_j}.$$

Como o ponto de mínimo em que $\varphi_j(t) > 0$ é maior que o ponto máximo do intervalo válido, a proposição está provada. ■

PROPOSIÇÃO 3. Para um conjunto de ordens disponíveis para seqüenciação denotado por R_q^ν , cujas funções de prioridade EXP-ET $\varphi_j(t_1) < 0$ ($\forall j \in R_q^\nu$), o instante t^* onde $\varphi_j(t^*) \geq 0$ para pelo menos um j ($\forall j \in R_q^\nu$), é definido como t^*

$$= t_1 + \min_{j \in R_q^\nu} \left(d_j - p_j - \frac{kp_{med} w_j}{h_j + w_j} \right), \text{ onde } p_{med} = |R_q^\nu|^{-1} \sum_{j \in R_q^\nu} p_j.$$

DEMONSTRAÇÃO. A proposição 1 garante que o t_j^* de $\varphi_j(t)$ é definido por $d_j - p_j - w_j kp_{med} / (h_j + w_j)$ ($\forall j \in R_q^\nu$). A proposição 2 garante que antes de t_j^* os valores de $\varphi_j(t) < 0$. Se, dentre todas as ordens $j \in R_q^\nu$ utilizarmos a que permite o mínimo t_j^* , estaremos obtendo o primeiro instante onde uma prioridade se torna positiva. ■

Portanto, o período de ociosidade W_j antes de uma das ordens $j \in R_q^\nu$ pode ser definido como

$$W_j = t_j^* - C_{j-1},$$

com $C_0=0$. Para um conjunto de ordens com prioridades negativas, o **período de ociosidade efetivamente utilizado** para que pelo menos uma das prioridades se torne positiva, será equivalente a

$$W = \min_{j \in R_q^+} W_j .$$

4.2 ALGORITMO DE INSERÇÃO DE OCIOSIDADE

Neste subitem será feita uma adaptação no algoritmo de inserção de ociosidade de GAREY *et al.* (1988) para a função-objetivo do tipo $\Sigma(h_j E_j + w_j T_j)$. Este subitem e o próximo seguem de forma bastante semelhante o desenvolvimento e a notação utilizada por GAREY *et al.*

Para o desenvolvimento do algoritmo deve-se primeiramente acrescentar algumas características e notações. Daqui em diante, até o final deste capítulo, deve-se sempre ter em mente que o desenvolvimento realizado é feito para uma seqüência já definida. Portanto, J_j representará a j -ésima ordem na seqüência.

O **horário desejado de início** da ordem J_j , será definido como $a_j = d_j - p_j$. O **horário efetivo de início** de uma dada ordem será denotado por e_j . Um **bloco de ordens** B_i ($i=1,2,\dots,l$) é definido como uma seqüência parcial que é programada contigüamente, ou seja, sem inserção de ociosidade entre as ordens da mesma. Matematicamente pode-se dizer que uma dada seqüência de ordens $M = \langle J_{m_1}, J_{m_2}, \dots, J_{m_j} \rangle$ é um bloco de ordens se $e_k + p_k = e_{k+1}$ (para $m_1 \leq k < m_j$), $e_{m_1-1} + p_{m_1-1} < e_{m_1}$ (para $m_1 > 1$) e $e_{m_j} + p_{m_j} < e_{m_{j+1}}$ (para $m_j < n$). Semanticamente pode-se dizer que um bloco de ordens é uma seqüência parcial de ordens, entre as quais não pode existir ociosidade, mas que deve haver ociosidade antes da primeira ordem (se o bloco não for o primeiro) e após a última ordem (se o bloco não for o último).

Assume-se que uma seqüência completa $\pi_n(\sigma)$ possui l blocos $\langle B_1, \dots, B_l \rangle$ ($1 \leq i \leq l$; $l \leq n$). Um esquema gráfico dos conceitos de bloco é apresentado na Figura 10.

Cada bloco B_i é particionado em 2 subconjuntos, denominados *Atrasado*(i) e *Adiantado*(i). Seja uma ordem $J_j \in B_i$; $J_j \in \text{Atrasado}(i)$ se $e_j > a_j$ e $J_j \in \text{Adiantado}(i)$ se $e_j \leq a_j$. Semanticamente pode-se dizer que: se $J_j \in \text{Atrasado}(i)$, reduzindo-se e_j , **reduz-se o**

atraso de J_j ; se $J_j \in \text{Adiantado}(i)$, reduzindo-se e_j , **aumenta-se o adiantamento** de J_j .

Adicionalmente define-se:

$$H(i) = \sum_{\{J_j \in \text{Adiantado}(i)\}} h_j \quad \text{e}$$

$$W(i) = \sum_{\{J_j \in \text{Atrasado}(i)\}} w_j.$$

Para se representar um B_i serão necessários apenas as definições dos **índices extremos**, ou seja, $\text{Primeiro}(i)$ que é o índice do **primeiro elemento** de B_i e $\text{Último}(i)$ que é o índice do **último elemento** de B_i . Pela Figura 10, pode-se perceber que as ordens intermediárias são facilmente identificadas pois são relacionadas com a primeira e com a última.

Figura 10: significado gráfico dos blocos de um programa

O algoritmo tem início simplesmente programando-se o horário efetivo de início de J_1 igual ao horário desejado de início de J_1 , isto é, $e_1 = a_1$. Em geral, para um programa parcial $\pi_j(\sigma)$, programa-se J_{j+1} conforme dois casos:

Caso 1: $e_j + p_j \leq a_{j+1}$. Programa-se J_{j+1} para começar em a_{j+1} . Nesse caso J_{j+1} não possui nem atraso nem adiantamento e $g(\pi_j(\sigma)) = g(\pi_{j+1}(\sigma))$ pois foi inserido um período de ociosidade.

Caso 2: $e_j + p_j > a_{j+1}$. Programa-se J_{j+1} para começar em $e_j + p_j$. Nesse caso J_{j+1} possui atraso. O elemento J_{j+1} é inserido no bloco B_l (onde l é o índice do bloco em análise) e no conjunto $\text{Atrasado}(l)$.

Uma propriedade chave mantida pelo algoritmo e que será explorada mais adiante, é que para cada bloco $B_i \in \pi_j(\sigma)$, ou $W(i) < H(i)$ ou $e_{Primeiro(i)} = 0$ (quando $i=l=1$). Por consequência, após programar-se J_{j+1} , tem-se ou $W(l) < H(l)$, ou $W(l) \geq H(l)$ ou $e_{Primeiro(l)} = 0$.

Se $W(l) < H(l)$ ou $e_{Primeiro(l)} = 0$, **não se toma nenhuma ação** e o programa $\pi_{j+1}(\sigma)$ transforma-se no programa corrente. Por outro lado, se $W(l) = H(l)$ e $e_{Primeiro(l)} \neq 0$, pode-se **movimentar o bloco B_l inteiro**, sem afetar o valor da função objetivo do programa. Caso contrário, se $W(l) > H(l)$ e $e_{Primeiro(l)} \neq 0$, pode-se antecipar o bloco B_l inteiro, diminuindo o valor da função objetivo do programa. Essas movimentações são feitas até uma das 3 hipóteses seguintes acontecer:

1. $e_{Primeiro(l)} = 0$. Ocorrerá somente se $l=1$. Caso ocorra, o bloco B_l não pode ser movimentado para mais cedo pois ele estará começando no horário 0;
2. para algum $J_j \in B_l$, $e_j = a_j$. A ordem é transferida do conjunto $Atrasado(l)$ para $Adiantado(l)$ e $W(l) < H(l)$. Nesse caso, outras movimentações de B_l só irão aumentar o valor da função objetivo. Se a ordem for transferida de $Atrasado(l)$ para $Adiantado(l)$ e $W(l) \geq H(l)$, continua-se a movimentação;
3. $e_{Primeiro(l)} = e_{Último(l-1)} + p_{Último(l-1)}$. O bloco B_l é unido ao bloco B_{l-1} . Se $e_{Primeiro(l-1)} \neq 0$ e $W(l-1) + W(l) \geq H(l-1) + H(l)$, continua-se a movimentação dos blocos unidos $l-1$ e l . Se $e_{Primeiro(l-1)} \neq 0$ e $W(l-1) + W(l) < H(l-1) + H(l)$, deve-se parar a movimentação pois a função objetivo só irá piorar.

O programa resultante é $\pi_{j+1}(\sigma)$. O algoritmo faz **sucessivas aplicações** do procedimento acima formando os programas $\pi_2(\sigma), \pi_3(\sigma), \dots, \pi_n(\sigma)$. A demonstração que esse algoritmo proporciona um programa ótimo para uma sequência predefinida no caso sem penalidades é apresentada em GAREY *et al.* (1988, p.338).

TEOREMA (baseado no teorema 2 de Garey et al. (1988)). Para qualquer j , o programa parcial $\pi_j(\sigma)$ calculado pelo algoritmo tem o mínimo $\Sigma(h_j E_j + w_j T_j)$ dentre todos os programas possíveis para as primeiras j ordens.

DEMONSTRAÇÃO: Ver anexo D. ■

Deve-se perceber que podem existir vários mínimos, quando por exemplo $W(i) = H(i)$. Quando existirem vários mínimos, o algoritmo proposto irá programar o início do bloco o mais cedo possível.

GAREY *et al.* (1988) não explicam de onde provém a propriedade chave proposta e comentada anteriormente, embora seja parte fundamental do algoritmo. Deve-se lembrar que a ordem iniciada no horário desejado pertence ao conjunto *Adiantado* e portanto, na melhor condição possível – condição de $g(\pi)=0$ –, todas as ordens estariam nos conjuntos *Adiantado*. Para o entendimento da propriedade chave, ela deve ser dividida em duas partes como segue:

Parte 1: Quando $e_{Primeiro(i)}=0$. Só acontece quando $i=l=1$. Não há ociosidade antes do primeiro bloco e conseqüentemente não há como diminuir os atrasos. A relação $W(l)>H(l)$ pode ser verdadeira pois não há como movimentar o bloco para mais cedo.

Parte 2: Quando $e_{Primeiro(l)}\neq 0$. Como um bloco l sempre é iniciado com $e_j=a_j$, sendo $Primeiro(l)=j$, a ordem J_j pertence ao conjunto *Adiantado* e $W(l)<H(l)$. Quando se acrescenta a ordem $j+1$ ao bloco B_l , podem ocorrer três casos: **(1)** $w_{j+1}>h_j$, a ordem J_{j+1} irá para o conjunto *Atrasado* e $W(l)>H(l)$; a propriedade chave não é mantida e o bloco deve ser movimentado para mais cedo pois $e_{Primeiro(l)}\neq 0$ e portanto há ociosidade. O bloco é movimentado para mais cedo até que $e_{Primeiro(l)}=0$ ou $W(l)<H(l)$. **(2)** $w_{j+1}<h_j$, a ordem J_{j+1} irá para o conjunto *Atrasado*, mas de maneira contrária ao caso (1), $W(l)<H(l)$ e a propriedade chave é mantida. Uma movimentação para mais cedo fará com que a ordem J_j fique adiantada e como $h_j>w_{j+1}$, a função-objetivo só irá piorar. **(3)** $w_{j+1}=h_j$, a ordem J_{j+1} irá para o conjunto *Atrasado* e $W(l)=H(l)$. As movimentações podem ser feitas pois não irão afetar o valor da função-objetivo. O que se perde quando se adianta a ordem J_j , se ganha com a diminuição do atraso da ordem J_{j+1} .

4.3 IMPLEMENTAÇÃO DO ALGORITMO DE INSERÇÃO DE OCIOSIDADE

Uma **implementação eficiente** do algoritmo do item anterior pode ser baseada na estrutura de dados denominada **pilha**. Sem a utilização das pilhas, o tempo de solução do algoritmo é $O(n^2)$, enquanto que com a utilização das pilhas é $O(n \log n)$.

Para cada bloco de ordens B_i é mantida uma pilha $P(i)$, que determinará o **quanto cada um poderá ser movimentado**. Cada pilha deverá manter dois valores de cada item: o primeiro é o índice das ordens pertencentes a $Atrasado(i)$; o segundo é o

valor equivalente a $e_j - a_j$ para cada índice j (a quantidade máxima que e_j pode ser diminuída, enquanto diminui o atraso de J_j).

```

procedimento Inserer Ociosidade ( $\sigma$ )
início
 $l := 1$ ;  $\text{Primeiro}(1) := 1$ ;  $\text{Último}(1) := 1$ ;
se  $a_1 < 0$  então
    início
         $H(1) := 0$ ;  $W(1) := w_1$ ;  $e_1 := 0$ ;
    fim;
caso contrário
    início
         $H(1) := h_1$ ;  $W(1) := 0$ ;  $e_1 := a_1$ ;
    fim;
 $\text{CriaPilha}(P(1))$ ;
para  $j := 1$  até  $n-1$  faça
    se  $e_j + p_j < a_{j+1}$  então
        início
             $l := l+1$ ;  $\text{Primeiro}(l) := j+1$ ;  $\text{Último}(l) := j+1$ ;  $H(l) := h_{j+1}$ ;  $W(l) := 0$ ;  $e_{j+1} := a_{j+1}$ ;  $\text{CriaPilha}(P(l))$ ;
        fim;
    caso contrário se  $e_j + p_j > a_{j+1}$  então
        início
             $\text{Último}(l) := j+1$ ;  $W(l) := W(l) + w_{j+1}$ ;  $e_{j+1} := e_j + p_j$ ;  $\text{Inserer}(j+1, e_{j+1} - a_{j+1}, P(l))$ ;
        enquanto  $H(l) \leq W(l)$  faça
            início
                 $\text{Diferença1} := \text{EncontraMin}(P(l))$ ;
            se  $l := 1$  então
                 $\text{Diferença2} := e_1$ ;
            caso contrário
                 $\text{Diferença2} := e_{\text{Primeiro}(l)} - e_{\text{Último}(l-1)} - P(\text{Último}(l-1))$ ;
             $\text{Diferença} := \min\{\text{Diferença1}, \text{Diferença2}\}$ ;  $\text{AdicionaParaTodasChaves}(-\text{Diferença}, P(l))$ ;
             $e_{\text{Primeiro}(l)} := e_{\text{Primeiro}(l)} - \text{Diferença}$ ;  $e_{\text{Último}(l)} := e_{\text{Último}(l)} - \text{Diferença}$ ;
        enquanto  $\text{EncontraMin}(P(l)) = 0$  faça
             $j := \text{ApagaMin}(P(l))$ ;  $W(l) := W(l) - w_j$ ;  $H(l) := H(l) + h_j$ ;
        fim de enquanto;
        se  $e_{\text{Primeiro}(l)} = e_{\text{Último}(l-1)} + P(\text{Último}(l-1))$  então
            início
                 $\text{Une}(P(l-1), P(l))$ ;  $\text{Último}(l-1) := \text{Último}(l)$ ;  $H(l-1) := H(l-1) + H(l)$ ;  $W(l-1) := W(l-1) + W(l)$ ;
                 $l := l-1$ ;
            fim;
        fim;
    fim de enquanto
fim;
caso contrário se  $e_j + p_j = a_{j+1}$  então
    início
         $\text{Último}(l) := j+1$ ;  $H(l) := H(l) + h_{j+1}$ ;  $e_{j+1} := e_j + p_j$ ;
    fim;
fim;

```

Figura 11: pseudocódigo para inserção de ociosidade

O algoritmo pode ser elaborado considerando-se seis operações com pilhas, conforme lista-se a seguir:

Operação 1. $\text{CriaPilha}(P)$. Cria uma pilha vazia denominada P ;

Operação 2. $\text{EncontraMin}(P)$. Encontra um item de mínima chave na pilha P e apresenta essa chave sem remover o item da pilha;

Operação 3. $\text{ApagaMin}(P)$. Encontra um item de mínima chave na pilha P . Apresenta seu índice e remove o item da pilha;

Operação 4. $\text{Insere}(j, x, P)$. Insere a o item j com chave x na pilha P ;

Operação 5. $\text{Une}(P_1, P_2)$. Une as pilhas P_1 e P_2 de itens disjuntos em uma nova pilha que transforma-se em P_1 . A pilha P_2 torna-se vazia;

Operação 6. $\text{AdicionaParaTodasChaves}(x, P)$. Adiciona-se x para todas as chaves dos itens pertencentes à pilha P .

O pseudocódigo utilizado para se inserir a ociosidade em uma seqüência de ordens predefinida é apresentado na Figura 11. O código em linguagem Turbo Pascal está disponível no apêndice 1, como um arquivo de inclusão de nome PINSOC13.INC.

Quando o procedimento é finalizado, os valores de $\text{Primeiro}(i)$, $\text{Último}(i)$, $e_{\text{Primeiro}(i)}$ e $e_{\text{Último}(i)}$ estão corretos. Os demais valores constantes nas pilhas podem não estar corretos.

Para se corrigir esse possível erro e elaborar um programa completo, deve-se utilizar a seguinte equação:

$$e_j = \begin{cases} e_{\text{Primeiro}(i)} & \text{se } j = \text{Primeiro}(i) \text{ para algum } i \leq l, \\ e_{j-1} + p_{j-1} & \text{caso contrário.} \end{cases}$$

4.4 EXEMPLO NUMÉRICO

Supondo que o seguinte conjunto de ordens com suas respectivas características está **disponível no instante zero**, já está **ordenado** e necessita que se verifique se a inserção de ociosidade pode promover uma melhoria no valor alcançado pela função objetivo:

j	p_j	d_j	h_j	w_j
1	45	223	50	10
2	82	218	97	58
3	48	285	92	50
4	73	190	35	84
5	36	190	45	2

Tabela 17: dados do exemplo para inserção de ociosidade

Por uma questão de comparação, será calculado inicialmente o valor da função objetivo da seqüência **sem a inserção de ociosidade**. A Tabela 18 ilustra alguns passos intermediários para esse caso.

j	C_j	E_j	T_j	$h_j E_j + w_j T_j$
1	45	178	0	8900
2	127	91	0	8827
3	175	110	0	10120
4	248	0	58	4872
5	284	0	94	188

Tabela 18: cálculo do exemplo sem inserção de ociosidade

O valor da função objetivo em análise sem a inserção de ociosidade é dado por $\Sigma(h_j E_j + w_j T_j) = 32907$. Utilizando-se o procedimento apresentado no subitem anterior, consegue-se chegar à Tabela 19, inclusive com os **períodos de ociosidade** – coluna representada por W_j .

Os valores intermediários dos **horários efetivos de início das ordens** obtidos pelo algoritmo são os seguintes: $e_1=178$; $e_1=178$, $e_2=223$; $e_1=91$, $e_2=136$; $e_1=91$, $e_2=136$, $e_3=237$; $e_1=91$, $e_2=136$, $e_3=237$, $e_4=285$; $e_1=91$, $e_2=136$, $e_3=237$, $e_4=285$, $e_5=358$. Os **blocos** existentes após o algoritmo ser finalizado são os seguintes: $B_1=\{J_1, J_2\}$ e $B_2=\{J_3, J_4, J_5\}$.

j	$a_j = d_j - p_j$	W_j	C_j	E_j	T_j	$h_j E_j + w_j T_j$
1	178	91	136	87	0	4350
2	136	0	218	0	0	0
3	237	19	285	0	0	0
4	117	0	358	0	168	14112
5	154	0	394	0	204	408

Tabela 19: cálculo do exemplo com inserção de ociosidade

Os blocos representam que as ordens devem ser produzidas sem a inserção de ociosidade entre as ordens do mesmo bloco. O valor da função objetivo nesse caso é equivalente a $\Sigma(h_j E_j + w_j T_j) = 18870$, que é comprovadamente o **mínimo valor alcançado** para a seqüência definida inicialmente.

5 ESTUDO COMPUTACIONAL

O principal objetivo do estudo computacional é analisar os **procedimentos de inserção de ociosidade** juntamente com o **BSF**. Adicionalmente algumas regras de despacho serão testadas. Uma comparação tentará identificar casos onde a **inserção de ociosidade poderia ser vantajosa** levando em consideração características intrínsecas dos dados que estão sendo utilizados. Por uma característica própria da proposta, as instâncias testadas deverão ter grandes dimensões. Devido a esse fator, torna-se **inviável** a utilização de um **método ótimo para comparação**.

Uma hipótese que pode ser suposta é que a diferença entre a solução ótima e a solução gerada pelo BSF não se altera conforme o aumento do tamanho da instância. Se essa hipótese for verdadeira¹⁷, pode-se considerar as diferenças obtidas nos estudos de OW; MORTON equivalentes às diferenças obtidas neste estudo.

O estudo computacional foi realizado tendo como base **estudos computacionais anteriores** relativos a problemas de **atraso ponderado**, isto é, cuja função objetivo é do tipo $\sum w_j T_j$, (SRINIVASAN, 1971, WILKERSON; IRWIN, 1971, BAKER; MARTIN, 1974, RINNOOY KAN *et al.*, 1975, FISHER, 1976, POTTS; VAN WASSENHOVE, 1985), assim como estudos relativos a **adiantamento e atraso ponderados**, $\sum (h_j E_j + w_j T_j)$, (ABDUL-RAZAQ; POTTS, 1988, OW; MORTON, 1988 e 1989, FRY *et al.*, 1996).

5.1 VARIÁVEIS CONTROLADAS

Desde o começo da década de 70, foram identificadas duas variáveis que parecem ter uma maior influência nos estudos que levam em consideração o atraso. A primeira delas é o **fator de atraso médio** τ (SRINIVASAN, 1971), que pode ser definida semanticamente como a proporção de ordens que atrasariam caso a sequência fosse obtida aleatoriamente. Matematicamente é definida como

$$\tau = 1 - \frac{d_{med}}{np_{med}} = 1 - \frac{\sum d_j}{n \sum p_j}.$$

¹⁷ Embora não seja confirmada.

Figura 12: variáveis testadas no estudo computacional

A segunda é a **amplitude relativa das datas**, R (WILKERSON; IRWIN, 1971, BAKER; MARTIN, 1974). Essa variável indica o quanto as datas de entrega estão distribuídas no intervalo total de processamento das ordens. Matematicamente

$$R = \frac{d_{\max} - d_{\min}}{np_{\text{med}}} = \frac{d_{\max} - d_{\min}}{\sum p_j}.$$

Para uma maior clareza dos conceitos, a Figura 12 mostra o significado gráfico dos mesmos. Deve-se notar pela figura que as variáveis alteram a média das datas de entrega e o quanto essas datas estão distribuídas com relação ao tempo total de processamento.

Existem divergências quanto aos valores dessas variáveis que mais dificultam a resolução dos problemas. Para o caso do atraso ponderado SRINIVASAN (1971) identificou como sendo 0,6 o valor de τ que mais dificulta a busca; RINNOOY KAN *et al.* (1975) identificaram como sendo 0,8, enquanto que POTTS; VAN WASSENHOVE (1985) identificaram como sendo entre 0,6 e 0,8. Nenhum dos estudos apontou o porquê dessa diferença. Uma possível causa é que SRINIVASAN gerou os tempos de processamento e as datas de acordo com uma normal bivariada, RINNOOY KAN *et al.* e POTTS; VAN WASSENHOVE utilizaram duas distribuições uniformes, com as datas sendo dependentes dos tempos de processamento. RINNOOY KAN *et al.* e FISHER (1976) identificaram $R=0,2$ como sendo o valor de maior dificuldade de solução do parâmetro R nas instâncias testadas. POTTS; VAN WASSENHOVE apontaram que para τ pequeno, um aumento no valor de R torna o problema mais fácil de ser resolvido, enquanto que para τ grande ocorre o contrário.

No caso de adiantamento e atraso ponderados, ABDUL-RAZAQ; POTTS (1988) e OW; MORTON (1989) encontraram que em média, para $\tau=0,2$ parece que a

resolução do problema torna-se mais difícil, entretanto alegam que haveria necessidade de testes mais extensivos para se fazer qualquer afirmação mais garantida. Foi observado que a dificuldade de resolução do problema tende a aumentar com a diminuição do valor de R . FRY *et al.* (1996) que trabalharam com a possibilidade de inserção de ociosidade apontam que conforme τ aumenta, a quantidade de ociosidade inserida tende a diminuir pois existe uma **tendência das ordens serem agrupadas**. Além disso FRY *et al.* alegam que seus resultados indicaram significativo aumento da necessidade computacional quando τ aumenta. Nesse estudo os autores utilizaram uma distribuição uniforme para a geração das datas e não citaram qual a distribuição dos tempos de processamento. Ao invés da utilização do R , os autores utilizaram o coeficiente de variação, CV , cuja expressão pode ser definida como:

$$CV = \frac{\sqrt{n^{-1} \sum (d_i - d_{med})^2}}{d_{med}}.$$

Nos trabalhos analisados, as datas de entrega e os tempos de processamento foram gerados de acordo com **distribuições normais** ou com **distribuições uniformes**. Algumas vezes a geração das datas de entrega é dependente da distribuição real dos tempos de processamento. Segundo BAKER; MARTIN (1974, p.193) o propósito de se estudar dependência entre dados é investigar possíveis diferenças sistemáticas no comportamento dos algoritmos.

Dentre todos os estudos consultados, nenhum apontou qualquer razão para a escolha dos **tipos de distribuição dos dados**. A dependência entre datas de entrega e tempos de processamento pode ser justificada pelo fato de que as gerações de números aleatórios nem sempre são de boa qualidade, conforme apontado por DWYER; WILLIAMS (1996). De uma maneira rigorosa, esses números deveriam ser chamados de **pseudo-aleatórios** pelo fato de que números “aleatórios” gerados pelo computador não são aleatórios.

Outras variáveis testadas como o coeficiente de correlação, entre os tempos de processamento e as datas de entrega (RINNOOY KAN *et al.*, 1975 e OW; MORTON, 1989) e a proporção entre penalidades de adiantamento e de atraso (OW; MORTON, 1989) parecem não influenciar de maneira significativa os resultados.

5.2 EXPERIMENTO REALIZADO

Pode-se dividir o experimento em três partes. A **primeira** – item 6.2.1 – é parte de geração das instâncias; a **segunda** – item 6.2.2 – é a da resolução das instâncias, enquanto que a **terceira** – item 6.3 – é a da interpretação dos resultados obtidos pelo experimento.

Os procedimentos e algoritmos foram programados em Turbo Pascal 6.0 e implementados em um microcomputador com processador PENTIUM-133Mhz, com 8MB de memória RAM. Os programas computacionais estão disponíveis no apêndice 1.

5.2.1 INSTÂNCIAS TESTADAS

As instâncias foram **geradas** a partir do programa ALEATORI.PAS, com as seguintes características:

- Número de ordens: $n=100$;
- Fator de atraso médio: $\tau \in \{0,1;0,2;0,4;0,6;0,8;0,9\}$;
- Amplitude relativa das datas de entrega: $R \in \{0,1;0,2;0,4;0,6;0,8;0,9\}$;
- Tempo de processamento: $p_j \sim U[1,100]$, onde $\sim U[\cdot]$ representa uma distribuição uniforme;
- Data de entrega: $d_j \sim U\left[P(1-\tau) - \frac{PR}{2}, P(1-\tau) + \frac{PR}{2}\right]$, onde $P=nE(p)=\frac{(n^2+n)}{2}$ seria a soma da esperança dos tempos de processamento, de acordo com a média da distribuição;
- Penalidade de adiantamento: $h_j \sim U[1,100]$;
- Penalidade de atraso: $w_j \sim U[1,100]$.

As distribuições são de números naturais. Combinando as instâncias com relação a τ e a R , tem-se **36 tipos de instâncias**. Para cada um desses 36 tipos foram geradas **20 instâncias**.

Como citado anteriormente, existem problemas na geração dos números aleatórios. Por essa razão criou-se um programa denominado VALIDADO.PAS que permite o aceite apenas das instâncias geradas de acordo com um critério de aprovação. O programa VALIDADO.PAS calcula a proporção da **diferença entre as variáveis reais e nominais**, ou seja:

$$\text{Prop}(\tau) = \frac{|\tau - \tau_{real}|}{\tau} e$$

$$\text{Prop}(R) = \frac{|R - R_{real}|}{R}.$$

As definições de τ_{real} e R_{real} para uma determinada instância $inst$, são as seguintes:

$$\tau_{real}(inst) = 1 - \frac{\sum d_j(inst)}{n(inst) \sum p_j(inst)} e$$

$$R_{real}(inst) = \frac{d_{\max}(inst) - d_{\min}(inst)}{\sum p_j(inst)}.$$

Com a utilização do critério de aceitação não houve necessidade de se gerar as datas de entrega dependentes dos tempos de processamento. Deve-se observar que os **casos tratados na literatura** e apresentados anteriormente **podem ter problemas na geração dos tempos de processamento**. Não há verificação se os mesmos foram gerados conforme o esperado. No caso proposto aqui, ambos, as datas de entrega e os tempos de processamento, têm seus valores avaliados após a geração.

Para uma dada proporção, $\text{Prop}(\cdot)$, o programa **aceita ou rejeita a instância em questão**. Neste trabalho, utilizou-se uma proporção de 15%, tanto em τ como em R , para a aceitação das instâncias.

Para os pares ordenados $(\tau, R) \in \{(0,8;0,8), (0,8;0,9), (0,9;0,4), (0,9;0,6), (0,9;0,8), (0,9;0,9)\}$, permitiu-se a geração de **datas de entrega negativas**. Os demais pares ordenados foram gerados de acordo com datas positivas. A razão para esse fato fica clara observando-se a Figura 12. Se o τ e o R são grandes, a parte esquerda de R – de $d_{\min}$ até d_{med} – possui uma região que fica antes de 0. O apêndice 2 apresenta duas instâncias completas que foram utilizadas no experimento computacional.

5.2.2 RESOLUÇÃO DAS INSTÂNCIAS

A resolução das instâncias foi separada em 3 grandes grupos. O primeiro grupo – **grupo 1** – é o das resoluções no modo despacho, isto é, sem inserção de ociosidade. O **grupo 2** é no modo programação, isto é, com possibilidade de inserção de ociosidade, com a seqüência inicial sendo definida por EXP-ET no modo despacho enquanto que o **grupo 3** também é no modo programação só que com a seqüência inicial sendo definida por EXP-ET no modo prioridade.

No **modo despacho**, programa-se as ordens de acordo com a seqüência, **sem a inserção de ociosidade**. Esse modo é muito mais facilmente implementável além de ser **amplamente utilizado** em indústrias.

No **modo programação**, define-se uma seqüência, insere-se a ociosidade nessa seqüência definida preliminarmente para só então se elaborar o programa com os respectivos **horários de início e de término** de cada ordem.

Uma **exceção** aos grupos 2 e 3 é a utilização da função de prioridade EXP-ET. Neste caso, calcula-se a prioridade de cada ordem disponível para ser programada. A ordem com maior prioridade é a que será programada subsequente. Se todas as prioridades forem negativas, deve-se aguardar – inserindo ociosidade – até que pelo menos uma delas se torne positiva.

A Tabela 20 apresenta os **programas computacionais utilizados** no experimento com suas respectivas **funções**. Detalhes mais específicos sobre cada programa podem ser obtidos no apêndice 1, através de comentários contidos nos mesmos.

PROGRAMA COMPUTACIONAL	FUNÇÃO DO PROGRAMA
BSF.PAS	Seqüência um conjunto de ordens de acordo com o <i>beam search</i> filtrado. Utiliza a função de prioridade EXP-ET no modo despacho como seqüência inicial.
BSFOCI.PAS	Idem ao BSF.PAS. Adiciona-se a inserção de ociosidade após a definição das seqüências completas que foram retidas. A função EXP-ET é utilizada como regra de despacho.
BSFOCISE.PAS	Idem ao BSFOCI.PAS. A seqüência inicial é definida pela função de prioridade EXP-ET no modo programação.
EDDTEMPO.PAS	Seqüência um conjunto de ordens de acordo com a menor data de entrega e calcula o valor da função objetivo.
EXPET.PAS	Seqüência um conjunto de ordens de acordo com a função de prioridade EXP-ET no modo despacho. Calcula o valor da função objetivo. É equivalente ao BSF com $b=1$ e $f=1$.
EXPETOCL.PAS	Seqüência um conjunto de ordens de acordo com a função de prioridade EXP-ET no modo programação, ou seja, admite-se a inserção de ociosidade.
WSPT.PAS	Seqüência um conjunto de ordens de acordo com a menor razão entre penalidade de atraso e tempo de processamento e calcula o valor da função objetivo.

Tabela 20 : programas computacionais utilizados e respectivas funções

5.3 INTERPRETAÇÃO DOS RESULTADOS

Pelo fato de vários testes e combinações de procedimentos terem sido feitos, pôde-se chegar a várias conclusões sob diversos pontos de vista. Esta seção será dividida de forma a facilitar o entendimento e a análise dos resultados.

5.3.1 TEMPOS DE PROCESSAMENTO COMPUTACIONAL

Os resultados apresentados aqui serão os **tempos de processamento computacional médios** das 20 instâncias resolvidas com relação a um determinado par ordenado (τ, R) .

Os tempos com relação às regras de despacho EDD e WSPT variaram muito pouco com relação a (τ, R) . Além disso, são muito pequenos, sendo que os maiores tempos ficaram em torno de 16 cs (centésimos de segundo). A função de prioridade EXP-ET também teve um tempo muito pequeno, com o máximo tempo medido em torno de 22 cs.

A Figura 13 apresenta os tempos de processamento computacional nos experimentos do grupo 1. Nesse caso não houve variação dos tempos com relação aos parâmetros (τ, R) . Pode-se notar que concorda com a 1ª proposição de OW; MORTON (1988) que a complexidade de tempo do BSF nesse tipo de problema é $O(n^2)$.

Figura 13: relação entre tempo de solução, larguras da busca e do filtro no BSF sem inserção de ociosidade

Figura 14: tempos de solução do BSF com inserção de ociosidade para $b=1$ e $b=3$

Figura 15: tempos de solução do BSF com inserção de ociosidade para $b=5$ e $b=8$

No caso dos procedimentos com inserção de ociosidade, o BSF depende dos parâmetros (τ, R) . Como a sequência inicial tem pequena participação no tempo total de solução, **não houve diferenças significativas com relação ao tipo de sequência inicial**, se por EXP-ET (prioridade) ou EXP-ET (despacho). A Figura 14 apresenta os

tempos de processamento computacional para $b=1$ e para $b=3$. A Figura 15 apresenta para $b=5$ e para $b=8$.

Pelos gráficos pode-se observar que enquanto o produto $b.f$ é pequeno com relação à escala de tempos, o tempo de solução é relativamente invariante com (τ, R) . De uma maneira geral, o tempo diminui com o aumento de τ e de R . Portanto, os **menores tempos** são obtidos com os **maiores valores do par ordenado (τ, R)** . Isso acontece pois nesse caso, a inserção de ociosidade é muito pequena – quando existe – e portanto o algoritmo de inserção de ociosidade é pouco utilizado.

5.3.2 COMPARAÇÃO COM RELAÇÃO AOS VALORES DAS FUNÇÕES-OBJETIVO

Os resultados numéricos alcançados pelas funções objetivo com os vários métodos utilizados são apresentados no anexo C. Abaixo são resumidos os resultados encontrados com relação aos diversos procedimentos testados.

REGRA DE DESPACHO: As regras de despacho foram utilizadas no grupo 1 de experimentos. O comportamento médio acima do mínimo encontrado em um determinado par ordenado (τ, R) foi 249,7% para WSPT e 145,8% para EDD. Para EXP-ET o desempenho foi muito superior do que as duas anteriores, ficando apenas 1,8% acima do mínimo encontrado.

Pode-se notar pela Tabela 21 – p.93 – que o desempenho de WSPT aumenta muito para τ alto, ou seja, onde quase todas as ordens já estão atrasadas. Para o caso em que todas as ordens estivessem atrasadas, WSPT forneceria a sequência ótima.

A regra EXP-ET teve um desempenho relativamente bom em todas os pares ordenados (τ, R) . Os maiores desvios (9% e 12%) aconteceram para $\tau=0,4$ e $R=0,8$ e $0,9$.

FUNÇÃO DE PRIORIDADE EXP-ET: A função de prioridade EXP-ET teve um desempenho bom para determinados pares ordenados (τ, R) . Observando-se a Tabela 25 – p.97 – verifica-se que o receio apontado por MORTON; PENTICO (1993, p.147) é até certo ponto justificável. Principalmente instâncias com $R=0,1$ para qualquer τ , tiveram resultados piores que a regra no modo despacho, isto é, valores menores do que 1.

Os casos testados que mais valeram a pena a utilização de ociosidade foram para τ pequeno e R grande. Essa conclusão é intuitivamente correta quando se observa a Figura 12.

Se todas as combinações testadas de (τ, R) forem levadas em consideração, parece que a utilização da regra EXP-ET no modo despacho é mais vantajosa. De qualquer forma, uma análise prévia do conjunto de ordens a serem programadas pode levar a consideráveis melhorias nos valores alcançados pelas funções-objetivo.

SEQÜÊNCIA INICIAL NO BEAM SEARCH FILTRADO: Talvez esse seja o fator de maior influência nos procedimentos testados. Pela Tabela 21 – p.93 – verifica-se a marcante diferença entre EXP-ET e EDD como seqüências iniciais. Mesmo se b e f forem grandes, uma seqüência inicial definida por EDD faz a busca ter um desempenho bastante inferior a uma que tenha a seqüência inicial definida por EXP-ET. Resultados semelhantes já haviam sido encontrados na resolução do exemplo do centro de usinagem no capítulo 2.

Outra indicação dessa importância é a Tabela 24 – p.96 –, onde foram comparados dois procedimentos iguais com as seqüências iniciais diferentes. Houve casos em que a diferença no valor das funções objetivo alcançou 1000%.

LARGURA DA BUSCA E DO FILTRO: Um fato curioso alcançado nos experimentos para o grupo 1 é que a largura do filtro “ideal” para 25 ordens alcançada por OW; MORTON (1988 e 1989) se não equivalente, é muito próxima da alcançada aqui. Outro ponto que pode ser apontado é com relação aos parâmetros testados. Nos estudos de OW; MORTON o leitor fica inclinado a entender que a utilização do filtro é boa para qualquer combinação de (τ, R) . Pela Tabela 21 – p.93 – pode-se inferir que há casos em que a utilização do filtro ou mesmo do BSF melhora a busca em apenas frações de um ponto percentual.

A Figura 20 – p.98 – aponta que na média, o f ideal para o grupo 1 está entre 2 e 4. Sem levar em consideração o tempo computacional, no caso de b , quanto maior seu valor melhor.

No grupo 2, o filtro, na média geral tendeu a piorar o desempenho da busca, conforme indicado na Figura 22 – p.99. Resultado semelhante ainda não foi comentado na literatura e talvez indique que a função utilizada no filtro é inadequada para o procedimento. Ao invés dela direcionar a busca para os caminhos mais viáveis na árvore

de busca, ela está apontando caminhos que são piores. Com relação a b , o procedimento teve seu desempenho melhorado com seu aumento.

Para os procedimentos do grupo 3, o filtro teve um comportamento diferente dos dois casos comentados anteriormente. Em média – vide Figura 24, p.100 –, o procedimento teve melhores resultados com o aumento de b e de f . Para casos específicos como $(0,2;0,2)$ o aumento no tamanho do filtro piorou o resultado da função objetivo. Mesmo nesse caso, o aumento de b melhorou o valor da função objetivo.

Como resultado final e sugestão pode-se dizer que o filtro utilizado com cautela em casos bem estudados, pode melhorar significativamente o BS com relação ao valor alcançado pela função-objetivo.

INSERÇÃO DE OCIOSIDADE: Para casos específicos, a inserção de ociosidade torna-se absolutamente vital. Pela Tabela 25 – p.97 – pode-se notar que para τ pequeno e R grande, a inserção de ociosidade melhora bastante o valor da função objetivo. Por outro lado casos como $(0,4;0,6)$ tiveram resultados bem piores do que sem o procedimento de inserção de ociosidade. Portanto mais uma vez os resultados indicam que uma boa escolha do tipo de procedimento, e dos parâmetros utilizados no procedimento como por exemplo b e f têm importância significativa para o bom desempenho do procedimento utilizado.

6 CONCLUSÕES

Assim como comentado no capítulo anterior, várias conclusões podem ser obtidas com relação ao estudo computacional. Além disso, os subitens subseqüentes apresentarão outros comentários, além dos obtidos pelo estudo computacional; as contribuições da dissertação e algumas possíveis extensões dos temas estudados.

6.1 PRINCIPAIS RESULTADOS OBTIDOS NO ESTUDO COMPUTACIONAL

Os principais resultados obtidos foram com relação ao desempenho dos procedimentos testados. Ficou muito claro que é de significativa importância a **escolha adequada de uma seqüência – ou programa – inicial**.

Além disso, é muito difícil fazer generalizações com relação a um suposto procedimento que seja melhor. As variáveis (τ, R) escolhidas se mostraram bastante adequadas para se analisar o tipo de problema aqui abordado pois mostrou forte influência no desempenho dos procedimentos.

Dependendo da combinação de (τ, R) deve-se escolher o melhor procedimento disponível, incluindo-se inclusive as regras de despacho pela sua simplicidade, rapidez e resultados alcançados.

6.2 CONSIDERAÇÕES ADICIONAIS

Se forem levados em consideração casos reais, a importância da inserção de ociosidade provavelmente seria diminuída. Isso aconteceria devido às empresas sempre terem uma carteira de pedidos que se aproxima de sua capacidade produtiva.

Mesmo assim, a inserção de ociosidade como no caso estudado aqui deve ser encarada como uma ferramenta adicional de programação. Seu principal intuito é diminuir os custos referentes ao adiantamento na utilização de recursos.

Para situações reais, uma inserção de ociosidade freqüente indica que alguma providência deve ser tomada nos outros níveis de decisão. Por exemplo, a eliminação de turnos de trabalho, o remanejamento de pessoal ou a diminuição do número de horas-

extras realizadas podem ser opções mais lucrativas sob o ponto de vista da empresa. Lembrando mais uma vez, a inserção de ociosidade na programação da produção é feita com o intuito de diminuição de custos, mas sem levar em consideração a hipótese de se tomar decisões pertencentes a outros níveis.

Pela complexidade do procedimento, da forma que foi apresentado, parece que o *beam search* só se adequaria num ambiente real de programação a casos de uma máquina. Dessa forma, ele seria utilizado como descrito no item 2.3.2, fazendo a programação da máquina gargalo. As outras máquinas deveriam ser programadas com procedimentos mais simples como regras de despacho.

Pelos resultados alcançados neste trabalho, há fortes evidências que as funções de prioridade alcançam resultados muito bons e sob o ponto de vista da utilização, são apenas um pouco mais sofisticadas do que as regras de despacho. Mesmo no campo teórico, por serem relativamente recentes e por alcançarem resultados excelentes em alguns casos, as funções de prioridade parecem ter um futuro promissor.

6.3 CONTRIBUIÇÕES DA DISSERTAÇÃO

No decorrer da dissertação, houve a necessidade de se utilizar alguns conceitos ainda não tratados diretamente na literatura disponível. São relacionados abaixo, por ordem de importância sob o ponto de vista do autor, os principais pontos que estão presentes na dissertação e ausentes na literatura consultada:

- a) Adaptação do algoritmo de inserção de ociosidade de GAREY *et al.* (1988) para o caso de penalidades distintas de adiantamento e de atraso;
- b) Determinação do intervalo ótimo de reavaliação da função de prioridade EXP-ET;
- c) Teste de uma função de prioridade para se inserir ociosidade. MORTON; PENTICO (1993) haviam citado essa hipótese mas sem testá-la;
- d) Distinção clara entre funções de prioridade e regras de despacho;
- e) Utilização de um procedimento de inserção de ociosidade para o caso de penalidades distintas de adiantamento e de atraso. Embora FRY *et al.* (1987) tivessem o algoritmo para penalidades individuais, utilizaram penalidades únicas para adiantamento e para atraso;

- f) Expansão da classificação presente em LAWLER *et al.* (1993). Acrescentaram-se características de inserção de ociosidade entre ordens e algumas funções objetivo que não são não decrescentes com o horário de término da ordem;
- g) Expansão do experimento realizado por OW; MORTON (1988 e 1989). Ampliou-se a gama de (τ, R) testados, além do número de ordens ser significativamente maior. Anteriormente testes haviam sido feitos para casos de até 25 ordens e aqui para casos de 100 ordens;
- h) Uma revisão bibliográfica atualizada da aplicação do BS em problemas gerais de programação e seqüenciação.

6.4 ALGUMAS EXTENSÕES POSSÍVEIS DO TEMA ESTUDADO

Os tópicos abaixo apresentam algumas sugestões para pesquisas futuras que tenham algum relacionamento com o tema abordado nesta dissertação:

- a) Expansão da classificação presente em LAWLER *et al.* (1993) para funções-objetivo que levem em consideração múltiplos critérios, além de medidas de variância. Pela bibliografia consultada, ambas as áreas parecem estar em franco desenvolvimento;
- b) Definição do parâmetro k de acordo com os parâmetros (τ, R) . Testes “paralelos” que não foram apresentados nesta dissertação indicaram que o parâmetro k exerce forte influência nos valores das funções-objetivo para a função EXP-ET;
- c) Determinação de uma função de prioridade equivalente à EXP-ET como um polinômio de grau 5 ou inferior. Nesse caso a prioridade de uma ordem cresceria ou decresceria continuamente com o tempo e sem intervalos constantes como na EXP-ET convencional. Isto poderia levar a uma maior facilidade de previsão de valores além de facilitar uma análise estatística ou probabilística. Nesse caso haveria uma hipótese de se estudar um f variável de acordo com os valores obtidos pela função do filtro – função de avaliação prévia.

- d) Desenvolvimento de um procedimento “mais poderoso” com relação aos resultados alcançados pelo *beam search* com inserção de ociosidade. Em alguns casos apresentados aqui, a inserção de ociosidade fez os resultados da busca piorarem. Tentar de alguma forma elaborar um limitante superior que utilize ambos, os procedimentos com e sem inserção de ociosidade de modo que o procedimento alcance o melhor dos dois. Por exemplo, YANO; KIM (1991) utilizam o procedimento de inserção de ociosidade após a seqüência final ser definida. Por consequência, não há como o valor da função objetivo no caso com inserção de ociosidade ser pior do que no caso sem ociosidade.

ANEXO A: CLASSIFICAÇÃO E REDUTIBILIDADE DE PROBLEMAS DE PROGRAMAÇÃO E SEQÜENCIAÇÃO

Há muito tempo existem tentativas bem sucedidas para se fazer a classificação dos diversos tipos de problemas de PS da produção como por exemplo em CONWAY *et al.* (1967). Uma síntese recente e bastante difundida na literatura científica é a classificação apresentada por LAWLER *et al.* (1993, p.450-53), proposta inicialmente por GRAHAM *et al.* (1979), na qual o presente anexo será baseado. Nessa classificação, os autores dividem os problemas em 3 campos $\alpha|\beta|\gamma$, denominados respectivamente **ambiente da máquina, características da ordem e critério de otimização**.

A 1 AMBIENTE DA MÁQUINA

O ambiente da máquina engloba as várias características que um problema possui com relação a suas máquinas. Leva em consideração o número delas, a rapidez que cada uma tem para processar uma determinada ordem de produção e como acontece o fluxo das ordens entre as máquinas.

O ambiente da máquina dos problemas de PS é definido por $\alpha=\alpha_1\alpha_2$. Seja $^{\circ}$ o símbolo de vazio, O_{ij} a operação i – realizada na máquina M_i – da ordem J_j e p_{ij} o tempo de processamento de O_{ij} .

Se $\alpha_1 \in \{^{\circ}, P\}$, a ordem J_j terá uma operação única ($J_j=O_{1j}$) na máquina ou no grupo de máquinas paralelas. Esse grupo de máquinas paralelas pode ser um setor inteiro que possui várias máquinas iguais, ou máquinas diferentes mas que podem fazer as mesmas operações onde deve-se decidir em qual delas a ordem deve ser processada. Os caixas de uma agência bancária podem ser considerados como máquinas – ou processadores no caso – paralelas.

$\alpha_1=^{\circ}$: uma máquina; $p_{1j}=p_j$;

$\alpha_1=P$: máquinas paralelas idênticas; $p_{ij}=p_j$ para todas M_i . Todas as máquinas têm a mesma rapidez para processar as ordens;

Se $\alpha_1 \in \{F, J\}$, existe uma seqüência obrigatória na realização das operações. Seja m_j o número de operações de J_j ; para $J_j = \{O_{1j}, O_{2j}, \dots, O_{m_jj}\}$, pode existir um ou

mais $J_k = \{O_{1k}, O_{2k}, \dots, O_{m_k k}\}$ sendo $|J_j| \neq |J_k|$ onde $|X|$ significa a cardinalidade do conjunto X .

$\alpha_1 = F$: *flow shop*; existe apenas um sentido de fluxo das ordens nas máquinas, ou seja, a matriz de transferência entre máquinas¹⁸ é uma matriz triangular superior (HAX; CANDEA, 1984, p.267-9). Para o caso de um *flow shop* não haveria valores abaixo dos “x” da matriz da nota de rodapé 18. Alguns exemplos são as células de produção e as linhas de montagem;

$\alpha_1 = J$: *job shop*; existem roteiros que devem ser seguidos. Não há necessidade que o sentido de processamento das ordens nas máquinas seja único, ou seja, a matriz de transferência entre máquinas é uma matriz quadrada de ordem $m+1$, onde m é o número de máquinas do *job shop* (HAX; CANDEA, 1984, p.267-9). Nesse caso, a matriz de transferência seria exatamente igual a apresentada na nota de rodapé 18.

Se $\alpha_1 \in \{O\}$, não existe uma seqüência obrigatória na realização das operações.

$\alpha_1 = O$: *open job shop*; é semelhante ao *job shop*, com a diferença que não há ordem de processamento obrigatória nas operações. Idealmente uma matriz de transferência do tipo da apresentada na nota de rodapé 18 possuiria todos os valores iguais, pois não haveria preferência na transferência entre as máquinas.

$\alpha_2 \in \{^\circ, c\}$. Representa o número de máquinas possuídas pelo ambiente da máquina.

$\alpha_2 = ^\circ$: m é uma variável que depende do problema em questão, ou seja, o problema é genérico e se adequa a qualquer valor de m .

¹⁸ De acordo com HAX; CANDEA (1984), uma matriz de transferência é uma matriz cujos valores especificam a probabilidade de que uma ordem terminada na máquina M_i , vá para outra máquina M_k . Portanto, pode-se especificar uma configuração de máquinas como na matriz abaixo. Por exemplo, (1,2) é a probabilidade de uma ordem produzida na M_1 ir para a M_2 ; (2,0) é a probabilidade de uma ordem que estava na M_2 terminar o processamento.

	para M_1	para M_2	...	para M_m	saída
entrada	(0,1)	(0,2)	(0,...)	(0, m)	x
de M_1	X	(1,2)	(1,...)	(1, m)	(1,0)
de M_2	(2,1)	x	(2,...)	(2, m)	(2,0)
...	(...,1)	(...,2)	x	(..., m)	(...,0)
de M_m	(m ,1)	(m ,2)	(m ,...)	x	(m ,0)

$\alpha_2=c$: $m=c \in \mathbf{N}$. O problema em questão possui um número específico de máquinas.

A 2 CARACTERÍSTICAS DA ORDEM

Ordens de produção são solicitações que o setor responsável pela programação da produção faz para o setor responsável pela produção. O procedimento de elaboração de ordens pode ser equivalente ao descrito no capítulo 1 para o exemplo da programação do centro de usinagem. As ordens podem possuir certas características que influenciam na PS, como por exemplo, necessitar de uma ferramenta especial para a sua elaboração. Algumas dessas características são descritas abaixo neste subitem.

As características da ordem dos problemas de PS podem ser definidas por $\beta = \{\beta_1, \beta_2, \beta_3\}$.

$$\beta_1 \in \{d_j = d, \circ\}$$

$\beta_1 = d_j = d$: todas as ordens possuem a mesma data de entrega. Essa característica atribui dependendo do problema, pelo menos duas condições de otimização que facilitam a resolução do problema de uma máquina. Essa característica poderia representar uma situação onde vários itens do mesmo cliente deveriam ser finalizados, ou mesmo a data onde todos os componentes de uma mesma montagem deveriam estar prontos;

$$\beta_1 = \circ$$
: as ordens possuem datas de entrega individuais, onde $d_j \in \mathbf{N}$.

$\beta_2 \in \{W_j, W_{ij}, \circ\}$, sendo W_j o período de ociosidade antes da realização da ordem J_j e W_{ij} o período de ociosidade antes da realização da operação O_{ij} da ordem J_j . Essa característica é extensivamente explorada nesta dissertação.

$\beta_2 = W_j$: antes da realização da ordem J_j , pode haver um período de ociosidade W_j . Um programa com essa característica possui infinitas combinações caso o tempo seja uma variável contínua;

$\beta_2 = W_{ij}$: antes da realização da operação O_{ij} da ordem J_j , pode haver um período de ociosidade W_{ij} . Um programa com essa característica possui infinitas combinações caso o tempo seja uma variável contínua. Poderá acontecer apenas se $\alpha_1 \in \{F, J, O\}$;

$$\beta_2 = \circ$$
: não é permitido tempo ocioso entre a realização de duas ordens.

$$\beta_3 \in \{res, res1, \circ\}$$
, onde *res* representa recurso.

$\beta_3=res$: assume-se que haja a presença de recursos limitados R_h ($h=1,...,e$) para a realização das ordens. Para a elaboração de uma ordem J_j , necessita-se a utilização de r_{hj} unidades de R_h durante todo o período de produção. O recurso limitado pode ser mão-de-obra ou ferramentas especiais por exemplo. Sabendo-se que existe uma determinada mão-de-obra disponível no chão-de-fábrica, deve-se programar atividades de modo que não sejam infringidas as quantidades disponíveis;

$\beta_3=res1$: assume-se que haja apenas um recurso limitado. No capítulo 3 apresenta-se o exemplo da utilização de um robô que é considerado um recurso limitado;

$\beta_3=^{\circ}$: não se especifica nenhuma restrição com relação aos recursos para a elaboração das ordens.

A 3 CRITÉRIO DE OTIMIZAÇÃO

Por uma questão de simplificação notacional, daqui em diante, um somatório do tipo $\sum_{j=1}^n x_j$ será representado por $\sum x_j$. O critério de otimização definido como uma **medida de eficiência agregada** pode ser dado por $\gamma \in \{f_{max}(\pi), \Sigma f_j(\pi), \Sigma g_j(\pi, d), \Sigma g_j(\pi)\}$. Antes de se explicar cada um desses critérios, serão apresentadas as principais **medidas de eficiência unitária**, relativa a uma dada ordem.

Os critérios de eficiência agregada são funções do programa π que por sua vez define certas funções de medida do horário de realização das ordens como as que seguem. Pode-se denominar o **horário de término da ordem** $J_{\sigma(j)}$ ($j=1,2,...,n$) a função:

$$C_{\sigma(j)}(\pi) = \begin{cases} r_{\sigma(j)} + \sum_{i=1}^m (W_{i\sigma(j)}(\pi) + p_{i\sigma(j)}) & \text{para } j = 1 \\ C_{\sigma(j-1)}(\pi) + \sum_{i=1}^m (W_{i\sigma(j)}(\pi) + p_{i\sigma(j)}) & \text{para } j = 2, \dots, n \end{cases}$$

$$= \begin{cases} r_{\sigma(j)} + W_{\sigma(j)}(\pi) + p_{\sigma(j)} & \text{para } j = 1 \\ C_{\sigma(j-1)}(\pi) + W_{\sigma(j)}(\pi) + p_{\sigma(j)} & \text{para } j = 2, \dots, n. \end{cases}$$

A maioria das outras funções de programação dependem dessa função. Mais uma vez, por questões de simplicidade notacional, quando não causar erro conceitual, será dada preferência para a notação que omite a seqüência σ . Exemplos desse caso são apresentados abaixo, onde os valores das funções para $J_{\sigma(j)}$ são equivalentes a valores

para J_j . A variável das funções do programa π também serão omitidas. Por exemplo, $C_j(\pi)$ e $L_j(\pi, d)$ serão representadas por C_j e $L_j(d)$.

A função **diferença** é dada por:

$$L_j = C_j - d_j.$$

Costuma-se dividir a função diferença quando ela é positiva e quando ela é negativa. Quando ela é negativa, é chamada de **adiantamento** e é definida por:

$$E_j = \max(0, -L_j).$$

Quando L_j é positiva é chamada de **atraso** e pode ser definida por:

$$T_j = \max(0, L_j).$$

A função **tempo de fluxo** F_j , é o tempo em que a ordem demora até ficar pronta, e é dada por:

$$F_j = C_j - r_j = \sum_{i=1}^m (p_{ij} + W_{ij}) = p_j + W_j.$$

A função **penalidade unitária** U_j é utilizada como uma penalidade quando algum fato indesejado ocorrer como por exemplo uma ordem atrasar. Nesse caso, seu valor não varia conforme a magnitude do atraso e é definida como:

$$U_j = \begin{cases} 0 & \text{para } C_j \leq d_j \\ 1 & \text{para outros casos.} \end{cases}$$

Os critérios de otimização mais tratados são denominados **critérios regulares**, f_j . Esses critérios podem ser definidos como sendo **funções não decrescentes com o aumento de C_j** (CONWAY *et al.*, 1967, p.12). Nos critérios regulares os problemas de programação podem ser reduzidos a problemas de seqüenciação pois um programa com ociosidade sempre será pior do que um programa sem ociosidade. Eles envolvem a minimização de $f_{\max}(\sigma)$ ou de $\sum f_j(\sigma)$. Portanto, se $\gamma \in \{f_{\max}, \sum f_j\}$, ele pode se definido por:

$$\gamma = \min(f_{\max} \in \{C_{\max}, L_{\max}\}) \cup \min(\sum f_j \in \{\sum C_j, \sum T_j, \sum U_j, \sum w_j C_j, \sum w_j T_j, \sum w_j U_j\}),$$

com

$$\min(f_{\max}) \cap \min(\sum f_j) = \emptyset, \text{ onde } w_j \text{ é uma penalidade da ordem } J_j.$$

Para uma melhor entendimento e clareza dos tipos de problemas que foram e que serão abordados, sugere-se a observação da Figura 16, p.85.

Nos **critérios não-regulares**, g_j , a **função-objetivo não precisa ser não decrescente com o aumento de C_j** . Isso representa um grande inconveniente, pois uma série de resultados generalizáveis ao caso regular não são válidos. Sejam h_j, w_j, v_j e u_j penalidades de J_j . Se $\gamma \in \{\Sigma g_j(\pi, d), \Sigma g_j(\pi)\}$, ele pode ser definido como:

$$\gamma = \min \left(\sum g_j \in \left\{ \begin{array}{l} \sum (E_j + T_j), \sum w_j (E_j + T_j), \sum (h_j E_j + w_j T_j), \\ \sum (h_j E_j + w_j T_j + v_j d_j), \sum (h_j E_j + w_j T_j + v_j C_j), \\ \sum (h_j E_j + w_j T_j + v_j C_j + u_j d_j) \end{array} \right\} \right)$$

O problema com duas variáveis é considerado mais fácil do que o problema com uma variável. Isso acontece devido à três propriedades de otimização presentes no problema irrestrito de data única, conforme apresentado em BAKER; SCUDDER (1990). Esses autores citam que os problemas com datas de entrega distintas d_j parecem ser intrinsecamente diferentes dos problemas com data única d .

O problema de data única é dividido em 2 casos principais: (1) o problema restrito ou de data fixa, onde $d < \Sigma p_j$ e (2) o problema irrestrito ou de data variável, onde não há restrições quanto ao valor de d .

A 4 EXEMPLOS DA UTILIZAÇÃO DA NOTACÃO

Para se ilustrar a utilização da notação apresentada neste anexo, seguem abaixo alguns exemplos:

$F4 | \Sigma w_j T_j$: problema de um *flow shop* com 4 máquinas, onde deseja-se minimizar a função objetivo definida como a soma do atraso ponderado de todas as ordens.

$1 | W_j | \Sigma (h_j E_j + w_j T_j)$: problema de máquina única, onde deseja-se minimizar a função objetivo definida como a soma de todas as diferenças ponderadas das ordens, sejam elas adiantamentos ou atrasos. Admite-se a inserção de ociosidade entre o término de uma ordem e início de outra.

$1 | d_j = d | \Sigma (h_j E_j(\pi, d) + w_j T_j(\pi, d))$: problema de máquina única, onde deseja-se minimizar a função objetivo definida como a soma de todas as diferenças ponderadas das ordens, sejam elas adiantamentos ou atrasos. Existe apenas uma data de entrega para todas as ordens e a mesma é definida de forma a otimizar a função-objetivo.

$J|C_{max}$: problema de um *job shop* genérico, isto é, sem restrições quanto ao número de máquinas, onde deseja-se minimizar a função-objetivo definida como o horário de término da última ordem realizada.

A 5 REDUTIBILIDADE ENTRE PROBLEMAS DE PROGRAMAÇÃO E SEQÜENCIAÇÃO

Informalmente pode-se dizer que a redutibilidade é uma característica na qual um problema U com menor nível de dificuldade de solução pode ser reduzido a outro problema V de maior nível de dificuldade de solução, ou seja, U é um caso especial de V . Outra maneira de se pensar nessa característica, é como um procedimento de generalização, onde existem problemas genéricos aos quais tenta-se fazer que outros problemas se tornem casos especiais desses problemas genéricos. Formalmente, um problema U é redutível – polinomialmente – a V (denota-se $U \propto V$) se existe um procedimento com tempo polinomial para a construção de uma instância $v \in V$ para qualquer instância $u \in U$, tal que resolvendo-se v resolve-se u (LENSTRA; RINNOOY KAN, 1979 e MONMA, 1980).

Uma proposta de se utilizar conceitos semelhantes para problemas de PS foi apresentada inicialmente por GRAHAM *et al.* (1979) e utilizada por MONMA (1980), LAGEWEG *et al.* (1982) e LAWLER *et al.* (1993) entre outros.

Esses autores propuseram a utilização de grafos para a obtenção de relações entre problemas. Cada problema pertencente às classes apresentadas nos subitens precedentes é representado por uma 5-tupla $(u_0, u_1, \dots, u_4)$ onde cada u_i é um nó dos grafos G_i mostrados na Figura 16. Para dois problemas $U=(u_0, u_1, \dots, u_4)$ e $V=(v_0, v_1, \dots, v_4)$, escreve-se $U \rightarrow V$ se $u_i = v_i$ ou se G_i contém uma rota direcionada de u_i para v_i , para $i=0,1,\dots,4$. A notação presente no grafo é explicada detalhadamente neste anexo.

Em geral (LAGEWEG *et al.* 1982, p.817) $U \rightarrow V$ implica que $U \propto V$. Os grafos portanto definem reduções elementares entre os problemas de PS. Daí pode-se concluir que (GRAHAM *et al.* 1979, LAGEWEG *et al.* 1982):

Se $U \rightarrow V$, e se U é NP-difícil, então V também será NP-difícil.

Se $U \rightarrow V$, e V pode ser resolvido em tempo polinomial, então U também poderá ser resolvido em tempo polinomial.

As provas formais com relação à redutibilidade são bastante difíceis em alguns casos, mas de uma maneira geral são amplamente disponíveis na literatura. Informalmente pode-se enxergar alguns casos sem muitas dificuldades. Um *flow shop* por exemplo, é redutível a um *job shop* pois permite apenas um sentido de processamento de ordens, enquanto que o último permite qualquer sentido de processamento de ordens. O problema com atraso simples pode ser reduzido ao problema com atraso ponderado caso os pesos do último sejam todos unitários, isto é, $\Sigma T_j \rightarrow \Sigma w_j T_j$ se $w_j = 1 \forall J_j$.

Figura 16: grafo de redutibilidade entre problemas de programação e sequênciação (adaptado de LAWLER *et al.*, 1993)

ANEXO B: COMPLEXIDADE COMPUTACIONAL

De uma maneira geral, as soluções de problemas de PS podem ser classificadas ou comparadas com relação à complexidade computacional de suas soluções. A complexidade pode ser uma medida de **tempo de solução**, da **memória necessária** ou de qualquer outra **unidade que analise a utilização de um determinado algoritmo** (CAMPELLO; MACULAN, 1994, p.29).

Na dissertação, a medida mais abordada é a **complexidade em tempo do algoritmo** – também denominada tempo de solução no caso pessimista ou rapidez do algoritmo (TERADA, 1990, p.5) ou ainda o tempo de solução máximo (GAREY; JOHNSON, 1979, p.6) – T como uma função do tamanho da entrada n . O n também pode ser chamado de número de dados de entrada que a instância possui e é definido como o número de *bits* – dígitos binários – necessários para codificá-la.

Em computadores comuns, cada **instância** de um dado problema, possui um determinado conjunto de valores de entrada $I=(i_1, i_2, \dots, i_n)$ que deverão ser transformados em uma notação binária. Portanto, para um determinado valor $i_k \in \mathbf{Z}$, sua representação binária conterá um número de *bits* equivalente a $\lceil \log_2(i_k+1) \rceil \forall i_k \in \mathbf{Z}$, onde $\lceil x \rceil = \min \{y \in \mathbf{Z} \mid y \geq x\}$ significa o menor inteiro não menor que x ¹⁹.

Na complexidade computacional, o que é importante é a **ordem assintótica de crescimento** de uma determinada função e não o valor exato da função. Valores de constantes ou elementos de ordens inferiores são suprimidos. Matematicamente pode-se dizer que (LAWLER *et al.*, 1993, p.447-50):

$$T(n) = O(g(n)) \text{ com } (n \rightarrow \infty)$$

Se $\exists a, n_0 > 0$ tal que $T(n) \leq a \cdot g(n)$ para todos $n \geq n_0$.

Utilizando-se o limite da função $T(n)$, pode-se dizer que $T(n)$ é dominada por $g(n)$ e que (BARTLE, 1983, p.125):

$$\lim_{n \rightarrow \infty} T(n)/g(n) = 0.$$

Informalmente pode-se dizer que para instâncias muito grandes, T certamente não cresce com uma taxa maior do que g . Poderia crescer na mesma taxa ou poderia

¹⁹ O número zero é uma exceção nesse caso e o número de *bits* necessários para codificá-lo é 1.

crescer mais devagar. Deve-se notar que um polinômio do tipo $a_m n^m + a_{m-1} n^{m-1} + \dots + a_0$ é de $O(n^m)$.

Abaixo são apresentados alguns gráficos (Figura 17 e Figura 18) que mostram as **diferenças nas ordens de crescimento** de algumas funções. Deve-se notar que para n pequeno, uma função de ordem assintoticamente mais alta não possui um valor necessariamente maior.

Figura 17: ordem de crescimento assintótico de algumas funções com escala logarítmica e escala normal ($1 \leq n \leq 25$)

Figura 18: ordem de crescimento assintótico de algumas funções com escala logarítmica e escala normal ($1 \leq n \leq 10$)

Para um procedimento computacional qualquer que resolva um problema, pode-se determinar a eficiência do mesmo a partir do número de passos utilizados para se chegar à solução, sabendo-se a quantidade de dados de entrada. Analisando procedimentos ou algoritmos computacionais com relação à ordem de crescimento do

tempo necessário para a solução, pode-se facilmente determinar o melhor dentre os disponíveis.

No campo da complexidade computacional os problemas são divididos em duas classes denominadas **problemas da classe P** – cujos algoritmos conhecidos, utilizados em sua solução são de tempo polinomial – e **problemas da classe NP** – cujos algoritmos conhecidos, utilizados em sua solução são de tempo não-polinomial. Para o entendimento dessas classes, há necessidade de se classificar os **problemas algorítmicos**.

B1 CLASSIFICAÇÃO DE PROBLEMAS ALGORÍTMICOS

Problemas algorítmicos podem ser classificados em três categorias denominadas **problemas de decisão**, **problemas de localização** e **problemas de otimização**, conforme CAMPELLO; MACULAN (1994, p.57-8).

Os problemas de decisão são aqueles cujo objetivo da solução é responder perguntas com respostas do tipo sim ou não. Nos problemas de localização, as respostas devem possuir determinadas estruturas que satisfaçam aos requisitos especificados pela pergunta. Para o caso dos problemas de PS, é uma resposta viável – atendendo às restrições – mas não necessariamente ótima. Já problemas de otimização buscam respostas satisfazendo requisitos especificados pela pergunta e além disso sejam ótimos para um critério pré-definido.

Com isso pode-se notar que existe uma certa **hierarquia** nessa classificação. Para se resolver um problema de otimização, deve-se resolver pelo menos um problema de localização e para se resolver um problema de localização, deve-se resolver pelo menos um problema de decisão.

Como geralmente o problema de decisão é o mais fácil de ser resolvido, a intratabilidade do problema pode ser definida nesse nível. A idéia é que se um problema possui uma determinada dificuldade de solução no nível de decisão, ele também terá, na melhor hipótese, a mesma dificuldade de resolução nos níveis de localização e de otimização.

B2 PROBLEMAS DA CLASSE P

São problemas cujos algoritmos conhecidos fornecem **soluções que podem ser obtidas através de uma função polinomial de n** , ou seja: $T(n)=O(n^k)$, onde k é uma constante (LAWLER *et al.*, 1993, p.448). Eles são considerados de fácil solução e os algoritmos mais eficientes pertencem a essa classe.

Se um algoritmo resolve um problema com notação do tipo unária²⁰ em tempo polinomial e com notação diferente de unária em tempo superpolinomial – tempo superior ao polinomial –, ele é denominado de **tempo pseudopolinomial**. Geralmente sua ordem de grandeza será do tipo $O(n\sum i_j)$ com i_j ($j=1,2,\dots,n$) sendo os valores da entrada e n o número total de entradas.

B3 PROBLEMAS DA CLASSE NP

São problemas cujos algoritmos conhecidos fornecem **soluções que podem ser obtidas de maneira enumerativa**, quer seja ela **implícita ou não**. De maneira geral o número de combinações possíveis é assustadoramente grande fazendo com que os algoritmos enumerativos não consigam resolver em tempo hábil, problemas com um número grande de entradas. São denominados algoritmos de tempo exponencial ou mais corretamente tempo superpolinomial (CAMPELLO; MACULAN, 1994, p.50-1). Os algoritmos do tipo *branch-and-bound* e o algoritmo simplex pertencem a essa classe (GAREY; JOHNSON, p.9).

Problemas NP-completos: (LAWLER *et al.*, 1993, p.449) São problemas que possuem uma forte evidência da não existência de um algoritmo cuja solução seja função polinomial do número de iterações. São considerados os mais difíceis da classe NP e, se algum deles for resolvido em tempo polinomial, então todos os problemas NP também poderão ser resolvidos em tempo polinomial. Se um problema NP-completo pode ser resolvido em tempo pseudopolinomial, ele é denominado **NP-completo no sentido ordinário**. Se o algoritmo não puder ser resolvido em tempo pseudopolinomial, ele é denominado **fortemente NP-completo**.

²⁰ A notação unária representa números através de segmentos de números "1". Por exemplo, $5=\langle 11111 \rangle_u$ em notação unária, ao passo que em notação binária $5=\langle 101 \rangle_b$.

Para se provar que um problema U é NP-completo, geralmente prova-se que V é NP-completo e o mesmo precisa apenas da prova que $V \in \text{NP}$ e que $U \propto V$ (a redutibilidade foi tratada no anexo A).

Problemas NP-difíceis: (LAWLER *et al.*, 1993, p.449) Os problemas de otimização, cujos problemas de decisão correspondentes são NP-completos são denominados NP-difíceis. Outra forma de caracterizar um problema NP-difícil é a seguinte: para um problema de decisão NP denominado V , diz-se que ele é NP-difícil se for redutível à U para todo $V \in \text{NP}$. Matematicamente, U é NP-difícil se $V \propto U \forall V \in \text{NP}$.

Quando se sabe que um problema de otimização é NP-difícil, tem-se a certeza que nem sempre a solução ótima será encontrada rapidamente. Por consequência aparece a necessidade da utilização de heurísticas para se tratar com esses problemas.

As denominações de problemas **NP-difíceis no sentido ordinário** e **fortemente NP-difíceis**, seguem as mesmas definições dos problemas NP-completos.

ANEXO C: RESULTADOS NUMÉRICOS

As tabelas 21, 22 e 23 apresentam comparações entre os valores das funções-objetivo e o menor valor de uma determinada linha, ou seja, de um determinado par ordenado (τ, R) . As colunas determinam os procedimentos de solução utilizados. Para o grupo BSF – na primeira linha –, EDD, EXP-ET (despacho) e EXP-ET (prioridade) representam as seqüências iniciais. Para o grupo BSF, os dois números presentes na parte superior das colunas representam respectivamente a largura da busca (b) e a largura do filtro (f).

Seja $g_{col}(\sigma | \tau, R)$ o valor da função objetivo da coluna col para um determinado par ordenado (τ, R) . Os valores presentes nas três primeiras tabelas podem ser definidos como

$$\frac{g_{col}(\sigma | \tau, R) - \min_{col \in proced} g_{col}(\sigma | \tau, R)}{\min_{col \in proced} g_{col}(\sigma | \tau, R)} \times 100,$$

onde *proced* é o conjunto de procedimentos testados presentes em uma determinada tabela. Na fórmula, a notação foi utilizada para a seqüência σ – Tabela 21 –, entretanto para o programa π – Tabela 22 e Tabela 23 –, a definição é análoga.

Semanticamente pode-se defini-los como o valor percentual acima do mínimo encontrado em uma determinada linha.

A Tabela 24 representa a seguinte razão:

$$\frac{g_{col}(\pi_{\text{EXP-ET}(\text{desp})} | \tau, R)}{g_{col}(\pi_{\text{EXP-ET}(\text{prior})} | \tau, R)},$$

onde $\pi_{\text{EXP-ET}(\text{prior})}$ representa o programa π com o programa inicial sendo definido por EXP-ET (prioridade) e $\pi_{\text{EXP-ET}(\text{desp})}$ representa o programa π com a sequencia inicial sendo definida por EXP-ET (despacho).

A Tabela 25 representa

$$\frac{g_{col}(\sigma_{\text{EXP-ET}(\text{desp})} | \tau, R)}{g_{col}(\pi_{\text{EXP-ET}(\text{prior})} | \tau, R)}.$$

As figuras de número 19 até 24 representam graficamente alguns resultados presentes nas tabelas do anexo C. A parte superior das mesmas apresenta o par

ordenado (τ, R) para o qual os procedimentos foram testados. O eixo da ordenada contém os valores presentes nas tabelas e definidos conforme especificado acima.

PARÂM. NOM.		EDD	WSPT	EXP-ET desp.	BSF																					
atraso médio	amplít. distrib.				EDD	EXP-ET (despacho)																				
						3,7	1,1	1,3	1,5	1,8	1,12	3,1	3,3	3,5	3,8	3,12	5,1	5,3	5,5	5,8	5,12	8,1	8,3	8,5	8,8	8,12
0,1	0,1	119,3	180,3	0,179	129,7	0,179	0,010	0,001	0,029	0,078	0,179	0,009	0,000	0,029	0,078	0,179	0,009	0,000	0,028	0,078	0,179	0,009	0,000	0,028	0,078	0,179
	0,2	122,3	180,1	0,442	127,0	0,442	0,001	0,031	0,087	0,153	0,442	0,001	0,030	0,087	0,153	0,442	0,000	0,029	0,086	0,152	0,442	0,000	0,029	0,086	0,152	0,442
	0,4	110,0	176,5	0,732	96,8	0,732	0,005	0,152	0,226	0,310	0,732	0,002	0,145	0,223	0,307	0,732	0,000	0,134	0,216	0,303	0,732	0,001	0,136	0,220	0,304	0,732
	0,6	112,5	179,4	0,891	88,0	0,891	0,012	0,098	0,299	0,409	0,891	0,012	0,092	0,287	0,399	0,891	0,000	0,090	0,280	0,398	0,891	0,003	0,089	0,291	0,394	0,891
	0,8	96,1	163,2	0,944	69,7	0,944	0,015	0,178	0,299	0,360	0,944	0,004	0,171	0,296	0,359	0,944	0,000	0,170	0,296	0,354	0,944	0,010	0,139	0,286	0,351	0,944
0,9	96,9	162,2	1,259	67,2	1,259	0,026	0,233	0,379	0,541	1,259	0,001	0,196	0,345	0,540	1,259	0,000	0,152	0,309	0,531	1,259	0,002	0,159	0,323	0,503	1,259	
0,2	0,1	134,1	206,1	0,188	148,4	0,188	0,001	0,003	0,017	0,050	0,188	0,000	0,001	0,017	0,049	0,188	0,000	0,001	0,017	0,048	0,188	0,000	0,001	0,017	0,049	0,188
	0,2	143,3	252,1	0,721	181,7	0,721	0,157	0,001	0,212	0,277	0,721	0,155	0,001	0,208	0,277	0,721	0,154	0,000	0,208	0,276	0,721	0,154	0,000	0,211	0,276	0,721
	0,4	146,2	263,2	1,177	187,2	1,177	0,020	0,155	0,258	0,396	1,177	0,020	0,154	0,256	0,395	1,177	0,001	0,098	0,241	0,394	1,177	0,000	0,112	0,239	0,392	1,177
	0,6	168,1	297,9	2,299	180,5	2,299	0,029	0,392	0,692	0,857	2,299	0,012	0,334	0,629	0,854	2,299	0,012	0,332	0,611	0,843	2,299	0,000	0,320	0,625	0,815	2,299
	0,8	140,2	275,0	2,532	134,6	2,532	0,153	0,493	0,815	1,113	2,532	0,051	0,459	0,784	1,080	2,532	0,050	0,402	0,727	1,043	2,532	0,000	0,419	0,725	1,060	2,532
0,9	116,6	238,5	2,326	106,9	2,326	0,054	0,315	0,804	0,971	2,326	0,039	0,304	0,782	0,942	2,326	0,027	0,242	0,752	0,933	2,326	0,000	0,232	0,759	0,924	2,326	
0,4	0,1	148,1	203,1	0,441	142,2	0,441	0,131	0,004	0,144	0,163	0,441	0,131	0,000	0,144	0,162	0,441	0,131	0,000	0,144	0,161	0,441	0,131	0,000	0,144	0,161	0,441
	0,2	156,5	249,0	0,703	175,2	0,703	0,019	0,003	0,049	0,116	0,703	0,018	0,000	0,048	0,115	0,703	0,018	0,000	0,048	0,115	0,703	0,018	0,007	0,047	0,115	0,703
	0,4	177,5	395,5	1,342	284,5	1,342	0,133	0,025	0,046	0,140	1,342	0,126	0,021	0,004	0,084	1,342	0,116	0,011	0,000	0,043	1,342	0,111	0,000	0,004	0,048	1,342
	0,6	221,7	670,1	3,818	480,1	3,818	0,470	0,068	0,428	0,741	3,818	0,470	0,007	0,406	0,710	3,818	0,377	0,000	0,338	0,710	3,818	0,334	0,000	0,157	0,622	3,818
	0,8	257,3	1002,4	9,092	681,9	9,092	0,196	0,910	1,417	2,405	9,092	0,104	0,464	1,025	1,968	9,092	0,022	0,285	0,895	1,825	9,092	0,000	0,343	0,791	1,572	9,092
0,9	215,5	1001,6	12,229	631,9	12,229	0,481	1,842	2,984	3,981	12,229	0,204	1,487	2,377	3,469	12,229	0,068	1,348	2,337	3,160	12,219	0,000	0,981	2,108	3,005	12,219	
0,6	0,1	130,9	95,1	0,930	66,1	0,930	0,311	0,071	0,356	0,376	0,930	0,263	0,017	0,310	0,368	0,930	0,263	0,014	0,305	0,364	0,930	0,263	0,000	0,299	0,352	0,930
	0,2	128,0	109,3	0,592	76,4	0,592	0,107	0,015	0,027	0,064	0,592	0,091	0,013	0,022	0,055	0,592	0,083	0,000	0,019	0,041	0,592	0,086	0,008	0,016	0,049	0,592
	0,4	144,6	188,5	1,187	136,3	1,187	0,128	0,049	0,029	0,066	1,187	0,118	0,045	0,017	0,048	1,187	0,109	0,036	0,014	0,044	1,187	0,108	0,012	0,000	0,030	1,187
	0,6	188,4	426,4	3,659	301,9	3,659	1,250	1,212	1,289	1,364	3,659	1,305	1,020	1,118	0,937	1,883	1,188	0,976	0,995	0,846	0,000	1,037	0,984	0,984	0,891	1,883
	0,8	260,1	728,4	4,203	516,3	4,203	1,234	1,609	1,327	1,346	4,072	0,925	1,161	0,851	0,919	2,617	0,624	0,864	0,682	0,524	0,561	0,561	0,530	0,215	0,000	0,561
0,9	278,6	789,0	3,366	571,9	3,366	1,533	1,415	1,274	0,854	3,319	1,145	0,966	0,782	0,335	3,017	0,944	0,860	0,575	0,195	2,981	0,894	0,787	0,346	0,000	2,981	
0,8	0,1	95,3	15,0	1,121	16,2	1,121	0,690	0,806	0,637	0,902	1,121	0,689	0,796	0,629	0,893	1,121	0,684	0,788	0,605	0,884	0,000	0,674	0,789	0,596	0,874	1,121
	0,2	127,2	33,7	2,156	37,3	2,156	1,231	1,549	1,600	1,663	2,156	1,213	1,569	1,575	1,628	2,156	1,205	1,553	1,557	1,602	0,000	1,123	1,538	1,548	1,586	2,156
	0,4	168,0	73,8	0,545	70,9	0,545	0,200	0,209	0,204	0,153	0,440	0,127	0,117	0,101	0,043	0,197	0,079	0,084	0,061	0,020	0,115	0,049	0,068	0,049	0,000	0,197
	0,6	172,1	104,5	0,690	87,3	0,690	0,256	0,280	0,247	0,224	0,660	0,170	0,176	0,147	0,112	0,422	0,123	0,097	0,027	0,054	0,375	0,095	0,063	0,000	0,040	0,660
	0,8	126,2	93,9	0,361	68,2	0,361	0,061	0,086	0,088	0,089	0,361	0,018	0,050	0,058	0,059	0,360	0,007	0,043	0,044	0,034	0,360	0,000	0,029	0,025	0,015	0,360
0,9	113,0	80,8	0,279	58,2	0,279	0,037	0,082	0,083	0,082	0,278	0,018	0,060	0,058	0,056	0,278	0,005	0,041	0,042	0,041	0,278	0,000	0,032	0,030	0,031	0,278	
0,9	0,1	106,3	7,3	1,643	13,1	1,643	1,120	1,248	1,003	1,164	0,737	1,321	1,089	0,983	1,124	0,356	1,083	1,081	0,962	1,100	0,000	1,019	1,075	0,917	1,088	0,356
	0,2	117,7	14,4	0,251	21,7	0,251	0,064	0,064	0,064	0,081	0,231	0,025	0,026	0,014	0,029	0,185	0,001	0,011	0,014	0,013	0,114	0,000	0,001	0,007	0,006	0,231
	0,4	116,1	22,0	0,203	29,5	0,203	0,026	0,048	0,063	0,056	0,203	0,012	0,026	0,035	0,032	0,202	0,005	0,017	0,028	0,017	0,202	0,000	0,009	0,018	0,006	0,202
	0,6	108,5	33,3	0,197	37,1	0,197	0,027	0,049	0,054	0,053	0,197	0,012	0,029	0,036	0,039	0,197	0,007	0,022	0,029	0,032	0,197	0,000	0,025	0,028	0,027	0,197
	0,8	97,3	39,1	0,156	36,3	0,156	0,027	0,034	0,044	0,044	0,156	0,019	0,024	0,032	0,035	0,156	0,015	0,018	0,012	0,015	0,156	0,000	0,006	0,009	0,010	0,156
0,9	86,8	37,5	0,130	32,5	0,130	0,017	0,023	0,039	0,041	0,129	0,010	0,014	0,031	0,031	0,129	0,001	0,001	0,015	0,013	0,129	0,000	0,000	0,012	0,009	0,129	
médias		145,8	249,7	1,750	169,2	1,750	0,284	0,382	0,489	0,602	1,715	0,246	0,307	0,410	0,519	1,560	0,206	0,272	0,376	0,478	1,375	0,186	0,248	0,338	0,440	1,375

Tabela 21: resultados do estudo computacional sem inserção de ociosidade - grupo 1

PARÂM. NOM.		EXP-ET prior.	BSF com ociosidade																			
atraso médio	amplit. distrib.		EXP-ET (despacho)																			
			1,1	1,3	1,5	1,8	1,12	3,1	3,3	3,5	3,8	3,12	5,1	5,3	5,5	5,8	5,12	8,1	8,3	8,5	8,8	8,12
0,1	0,1	8,791	0,230	0,040	0,008	0,072	0,118	0,230	0,033	0,020	0,072	0,118	0,230	0,031	0,034	0,041	0,088	0,230	0,000	0,018	0,040	0,082
	0,2	0,000	7,094	6,232	5,990	6,052	6,133	6,771	6,201	5,862	5,921	6,161	6,451	6,183	6,031	5,782	6,032	6,228	6,057	5,796	5,702	5,982
	0,4	0,000	57,09	57,74	58,34	58,38	57,82	55,48	56,77	57,86	57,47	57,12	54,90	56,83	57,52	57,59	57,43	54,24	56,68	57,39	57,48	57,59
	0,6	0,000	159,8	162,6	167,0	166,4	166,1	152,9	161,1	161,8	163,2	161,0	152,3	159,6	163,8	161,6	162,8	151,9	158,3	162,4	161,1	160,3
	0,8	0,000	477,1	490,7	495,9	488,0	493,3	463,7	479,6	483,5	488,3	491,9	462,8	470,2	483,8	485,3	482,5	459,2	471,3	485,0	492,5	487,5
	0,9	0,000	639,8	640,6	654,5	660,5	652,1	609,4	614,0	646,0	622,8	642,0	595,0	603,5	638,0	619,4	632,5	592,1	593,8	622,5	626,6	627,5
0,2	0,1	43,085	0,188	0,001	0,003	0,017	0,050	0,188	0,000	0,001	0,017	0,049	0,188	0,000	0,001	0,017	0,048	0,188	0,000	0,001	0,017	0,049
	0,2	37,051	0,723	0,159	0,001	0,212	0,277	0,723	0,157	0,001	0,211	0,265	0,723	0,156	0,000	0,210	0,263	0,723	0,156	0,002	0,210	0,260
	0,4	0,000	13,33	11,36	11,49	11,52	11,46	12,92	11,65	11,08	11,23	11,24	12,46	11,35	10,81	10,85	11,29	12,45	11,05	11,16	11,21	11,23
	0,6	0,000	71,8	68,3	69,9	69,4	69,9	71,1	68,6	69,5	70,3	69,4	70,1	68,3	68,4	68,4	69,4	69,9	68,4	68,7	68,0	69,2
	0,8	0,000	193,6	207,6	213,0	212,6	214,9	188,4	200,6	208,1	206,3	211,8	186,2	196,0	205,2	204,9	208,2	184,4	197,7	204,3	203,3	205,8
	0,9	0,000	375,3	388,0	390,7	389,3	399,9	357,3	360,3	382,1	392,7	390,3	350,4	357,6	376,3	370,4	380,4	348,5	350,5	365,9	367,1	371,5
0,4	0,1	99,556	0,441	0,131	0,004	0,144	0,163	0,441	0,131	0,000	0,144	0,162	0,441	0,131	0,000	0,144	0,161	0,441	0,131	0,000	0,144	0,161
	0,2	104,695	0,703	0,019	0,003	0,049	0,116	0,703	0,018	0,000	0,048	0,115	0,703	0,018	0,000	0,048	0,115	0,703	0,018	0,007	0,047	0,115
	0,4	60,821	1,342	0,133	0,025	0,046	0,140	1,342	0,126	0,021	0,004	0,084	1,342	0,116	0,011	0,000	0,043	1,342	0,111	0,000	0,004	0,048
	0,6	52,064	3,785	0,490	0,068	0,440	0,753	3,785	0,490	0,007	0,418	0,722	3,785	0,397	0,000	0,349	0,722	3,785	0,353	0,000	0,168	0,633
	0,8	6,897	8,779	0,783	1,050	1,257	2,278	8,614	0,420	0,564	0,953	1,121	8,596	0,181	0,240	0,665	0,961	7,960	0,000	0,172	0,386	0,609
	0,9	0,000	37,52	23,21	25,41	26,19	25,71	34,08	22,43	24,36	23,09	24,35	32,34	20,59	23,55	22,81	23,49	31,47	20,32	21,91	22,77	21,86
0,6	0,1	88,370	0,930	0,311	0,071	0,356	0,376	0,930	0,263	0,017	0,310	0,368	0,930	0,263	0,014	0,305	0,364	0,930	0,263	0,000	0,299	0,352
	0,2	72,465	0,592	0,107	0,015	0,027	0,064	0,592	0,091	0,013	0,022	0,055	0,592	0,083	0,000	0,019	0,041	0,592	0,086	0,008	0,016	0,049
	0,4	52,683	1,187	0,128	0,049	0,029	0,066	1,187	0,118	0,045	0,017	0,048	1,187	0,109	0,036	0,014	0,044	1,187	0,108	0,012	0,000	0,030
	0,6	33,088	3,659	1,250	1,212	1,289	1,364	3,659	1,305	1,020	1,118	0,937	1,883	1,188	0,976	0,995	0,846	0,000	1,037	0,984	0,984	0,891
	0,8	6,792	4,203	1,232	1,607	1,326	1,346	4,072	0,923	1,160	0,849	0,919	2,617	0,622	0,862	0,681	0,524	0,561	0,559	0,528	0,214	0,000
	0,9	3,371	3,353	1,558	1,390	1,259	0,832	3,306	1,171	0,941	0,769	0,316	3,004	0,970	0,835	0,568	0,182	2,968	0,919	0,762	0,357	0,000
0,8	0,1	23,694	1,121	0,690	0,806	0,637	0,902	1,121	0,689	0,796	0,629	0,893	1,121	0,684	0,788	0,605	0,884	0,000	0,674	0,789	0,596	0,874
	0,2	16,168	2,156	1,231	1,549	1,600	1,663	2,156	1,213	1,569	1,575	1,628	1,028	1,205	1,553	1,557	1,602	0,000	1,123	1,538	1,548	1,586
	0,4	0,545	0,545	0,200	0,209	0,204	0,153	0,440	0,127	0,117	0,101	0,043	0,197	0,079	0,084	0,061	0,020	0,115	0,049	0,068	0,049	0,000
	0,6	0,690	0,690	0,256	0,280	0,247	0,224	0,660	0,170	0,176	0,147	0,112	0,422	0,123	0,097	0,027	0,054	0,375	0,095	0,063	0,000	0,040
	0,8	0,361	0,361	0,061	0,086	0,088	0,089	0,361	0,018	0,050	0,058	0,059	0,360	0,007	0,043	0,044	0,034	0,360	0,000	0,029	0,025	0,015
	0,9	0,279	0,279	0,037	0,082	0,083	0,082	0,278	0,018	0,060	0,058	0,056	0,278	0,005	0,041	0,042	0,041	0,278	0,000	0,032	0,030	0,031
0,9	0,1	2,346	1,643	1,120	1,248	1,003	1,164	0,737	1,321	1,089	0,983	1,124	0,356	1,083	1,081	0,962	1,100	0,000	1,019	1,075	0,917	1,088
	0,2	0,251	0,251	0,064	0,064	0,064	0,081	0,231	0,025	0,026	0,014	0,029	0,185	0,001	0,011	0,014	0,013	0,114	0,000	0,001	0,007	0,006
	0,4	0,203	0,203	0,026	0,048	0,063	0,056	0,203	0,012	0,026	0,035	0,032	0,202	0,005	0,017	0,028	0,017	0,202	0,000	0,009	0,018	0,006
	0,6	0,197	0,197	0,027	0,049	0,054	0,053	0,197	0,012	0,029	0,036	0,039	0,197	0,007	0,022	0,029	0,032	0,197	0,000	0,025	0,028	0,027
	0,8	0,156	0,156	0,027	0,034	0,044	0,044	0,156	0,019	0,024	0,032	0,035	0,156	0,015	0,018	0,012	0,015	0,156	0,000	0,006	0,009	0,010
	0,9	0,130	0,130	0,017	0,023	0,039	0,041	0,129	0,010	0,014	0,031	0,031	0,129	0,001	0,001	0,015	0,013	0,129	0,000	0,000	0,012	0,009
médias		19,854	57,507	57,401	58,394	58,304	58,607	55,234	55,282	57,165	56,941	57,627	54,273	54,378	56,672	55,959	56,728	53,723	53,913	55,868	56,164	56,263

Tabela 22: resultados do estudo computacional com inserção de ociosidade - grupo 2

PARÂM. NOM.		EXP-ET	BSF com ociosidade																			
atraso médio	amplit. distrib.		EXP-ET (prioridade)																			
			prioridade	1,1	1,3	1,5	1,8	1,12	3,1	3,3	3,5	3,8	3,12	5,1	5,3	5,5	5,8	5,12	8,1	8,3	8,5	8,8
0,1	0,1	34,266	23,368	24,169	22,384	17,022	12,041	22,761	23,844	20,905	9,528	1,031	22,724	21,856	20,425	9,080	0,000	22,385	21,370	18,586	8,055	0,902
	0,2	39,761	43,149	32,182	25,805	17,738	12,016	35,135	23,134	18,919	10,002	0,095	33,077	21,696	16,441	4,006	0,793	32,035	20,159	11,481	2,213	0,000
	0,4	41,121	48,644	44,347	37,874	29,466	15,911	37,661	31,879	28,458	23,286	9,175	35,564	33,845	24,040	17,663	0,000	31,670	39,252	28,315	8,359	3,302
	0,6	21,487	102,105	97,840	68,414	49,510	47,474	74,957	87,626	84,824	25,619	19,771	61,264	77,115	58,686	53,542	9,028	52,433	102,149	44,733	48,694	0,000
	0,8	40,129	209,861	152,077	76,805	16,998	11,164	165,666	95,012	56,088	12,254	9,237	136,762	73,076	29,219	7,067	2,064	127,836	68,444	62,622	2,464	0,000
	0,9	42,159	218,105	82,340	18,989	9,994	9,833	160,456	68,746	8,602	4,549	4,015	152,170	54,947	6,090	2,624	2,198	123,887	28,621	11,486	0,000	0,329
0,2	0,1	42,816	0,000	3,953	5,905	6,721	8,207	0,000	3,460	5,345	6,676	8,697	0,000	3,370	5,616	7,313	7,893	0,000	2,972	5,791	6,645	8,053
	0,2	46,799	7,946	10,704	13,982	13,226	13,518	7,158	10,088	10,090	8,417	2,973	7,124	8,676	11,858	7,496	0,000	6,937	8,720	9,754	8,648	2,092
	0,4	38,883	61,756	48,332	29,613	32,244	8,204	55,525	40,026	32,691	19,974	10,980	47,757	39,252	20,536	20,789	8,864	41,890	42,396	35,007	18,248	0,000
	0,6	11,182	88,777	58,503	72,537	31,894	3,079	52,197	63,921	47,167	26,513	4,701	44,257	56,132	36,703	19,847	12,241	37,376	51,911	49,950	28,594	0,000
	0,8	40,970	207,220	116,452	90,059	60,240	22,358	147,959	130,833	80,841	30,546	2,876	122,883	120,584	76,746	36,501	0,960	93,607	90,717	64,756	26,232	0,000
	0,9	47,432	214,975	66,641	40,655	18,026	11,617	137,725	21,896	10,050	5,804	4,200	101,738	58,745	5,012	1,909	1,265	97,345	58,733	3,316	1,595	0,000
0,4	0,1	98,688	0,000	2,691	3,242	3,615	3,892	0,000	1,675	3,048	3,464	3,508	0,000	1,222	2,764	3,127	3,071	0,000	1,391	2,743	2,858	3,258
	0,2	103,266	0,000	5,044	8,433	9,430	10,115	0,000	4,996	7,464	9,280	9,614	0,000	4,902	6,900	8,877	9,259	0,000	4,207	6,867	8,623	9,055
	0,4	55,016	4,960	7,471	13,794	18,223	23,160	1,702	9,168	12,477	17,686	18,831	0,156	7,191	11,617	14,285	19,094	0,000	7,071	12,974	14,772	17,549
	0,6	22,549	14,731	27,054	22,726	17,680	21,162	5,210	36,265	38,078	22,341	5,495	3,255	34,871	23,994	32,119	6,563	0,000	29,296	30,559	21,562	8,227
	0,8	35,511	151,785	139,712	92,690	48,333	22,118	106,950	106,256	87,718	21,897	14,044	96,906	103,787	77,473	11,545	14,202	88,860	95,691	51,377	30,373	0,000
	0,9	45,937	167,435	67,092	26,770	11,166	10,348	108,903	40,379	15,836	4,499	3,850	85,539	31,117	19,483	1,642	1,307	71,411	19,328	5,514	0,353	0,000
0,6	0,1	109,007	11,987	7,076	5,374	3,075	1,111	11,987	6,873	5,222	2,904	0,226	11,987	6,837	5,186	2,848	0,094	11,987	6,708	4,985	2,419	0,000
	0,2	85,435	8,151	5,454	4,297	2,088	0,939	8,151	5,303	3,782	1,818	0,441	8,151	5,230	3,831	1,575	0,036	8,151	5,125	3,970	1,534	0,000
	0,4	50,891	0,000	3,670	4,965	4,743	3,397	0,000	3,072	4,845	4,063	2,918	0,000	2,606	4,460	3,622	2,735	0,000	2,980	3,782	3,747	2,535
	0,6	32,996	3,630	7,298	7,679	8,195	6,603	3,434	6,282	6,308	6,897	5,047	1,877	6,226	5,889	6,315	5,070	0,000	5,314	5,827	5,803	4,491
	0,8	6,494	3,912	1,551	1,913	1,718	1,335	3,782	1,234	1,475	0,983	0,782	2,331	1,071	1,176	0,434	0,606	0,280	0,000	0,574	0,180	0,076
	0,9	3,371	3,353	1,558	1,390	1,259	0,832	3,306	1,171	0,941	0,769	0,316	3,004	0,970	0,835	0,568	0,182	2,968	0,919	0,762	0,357	0,000
0,8	0,1	33,216	8,906	3,643	1,848	0,594	0,407	8,906	3,692	1,892	0,525	0,106	8,906	3,589	1,814	0,546	0,035	7,699	3,354	1,867	0,544	0,000
	0,2	20,995	6,402	2,448	1,412	1,091	0,450	6,402	1,804	1,296	0,961	0,193	5,227	1,723	1,152	0,777	0,061	4,156	1,312	0,831	0,699	0,000
	0,4	0,545	0,545	0,200	0,209	0,204	0,153	0,440	0,127	0,117	0,101	0,043	0,197	0,079	0,084	0,061	0,020	0,115	0,049	0,068	0,049	0,000
	0,6	0,690	0,690	0,256	0,280	0,247	0,224	0,660	0,170	0,176	0,147	0,112	0,422	0,123	0,097	0,027	0,054	0,375	0,095	0,063	0,000	0,040
	0,8	0,361	0,361	0,061	0,086	0,088	0,089	0,361	0,018	0,050	0,058	0,059	0,360	0,007	0,043	0,044	0,034	0,360	0,000	0,029	0,025	0,015
	0,9	0,279	0,279	0,037	0,082	0,083	0,082	0,278	0,018	0,060	0,058	0,056	0,278	0,005	0,041	0,042	0,041	0,278	0,000	0,032	0,030	0,031
0,9	0,1	2,346	1,643	0,923	0,942	0,679	0,828	0,737	1,087	0,761	0,581	0,757	0,356	0,716	0,744	0,555	0,718	0,000	0,624	0,693	0,544	0,678
	0,2	0,251	0,251	0,064	0,064	0,064	0,081	0,231	0,025	0,026	0,014	0,029	0,185	0,001	0,011	0,014	0,013	0,114	0,000	0,001	0,007	0,006
	0,4	0,203	0,203	0,026	0,048	0,063	0,056	0,203	0,012	0,026	0,035	0,032	0,202	0,005	0,017	0,028	0,017	0,202	0,000	0,009	0,018	0,006
	0,6	0,197	0,197	0,027	0,049	0,054	0,053	0,197	0,012	0,029	0,036	0,039	0,197	0,007	0,022	0,029	0,032	0,197	0,000	0,025	0,028	0,027
	0,8	0,156	0,156	0,027	0,034	0,044	0,044	0,156	0,019	0,024	0,032	0,035	0,156	0,015	0,018	0,012	0,015	0,156	0,000	0,006	0,009	0,010
	0,9	0,130	0,130	0,017	0,023	0,039	0,041	0,129	0,010	0,014	0,031	0,031	0,129	0,001	0,001	0,015	0,013	0,129	0,000	0,000	0,012	0,009
médias		32,098	44,878	28,359	19,483	12,107	7,859	32,481	23,059	16,546	7,843	4,009	27,643	21,711	13,306	7,693	3,016	24,023	19,970	13,315	7,064	1,686

Tabela 23: resultados do estudo computacional com inserção de ociosidade - grupo 3

PARÂMETROS NOM.		EXP-ET	BSF																			
Fator Atr.	Ampl. Distr.	prioridade	1,1	1,3	1,5	1,8	1,12	3,1	3,3	3,5	3,8	3,12	5,1	5,3	5,5	5,8	5,12	8,1	8,3	8,5	8,8	8,12
0,1	0,1	1,000	1,003	0,994	1,009	1,055	1,103	1,008	0,997	1,021	1,128	1,223	1,008	1,013	1,025	1,132	1,235	1,011	1,017	1,041	1,143	1,224
	0,2	1,000	1,046	1,123	1,177	1,259	1,324	1,104	1,205	1,244	1,346	1,482	1,118	1,219	1,273	1,421	1,470	1,124	1,234	1,326	1,445	1,481
	0,4	1,000	1,491	1,542	1,621	1,726	1,921	1,594	1,678	1,734	1,803	2,031	1,613	1,654	1,792	1,890	2,222	1,653	1,588	1,731	2,051	2,153
	0,6	1,000	1,562	1,613	1,926	2,165	2,192	1,756	1,690	1,721	2,545	2,647	1,901	1,781	2,020	2,070	2,928	2,008	1,552	2,203	2,133	3,163
	0,8	1,000	2,610	3,284	4,723	7,043	7,479	2,973	4,165	5,239	7,344	7,592	3,331	4,616	6,331	7,660	7,997	3,439	4,752	5,041	8,104	8,232
0,9	1,000	3,306	5,774	9,014	9,829	9,734	3,872	6,015	9,764	9,828	10,141	3,918	6,454	9,890	9,966	10,190	4,394	7,668	9,212	10,329	10,308	
0,2	0,1	1,000	1,000	0,960	0,942	0,935	0,923	1,000	0,965	0,947	0,936	0,919	1,000	0,966	0,945	0,930	0,926	1,000	0,969	0,943	0,936	0,924
	0,2	1,000	0,999	0,969	0,940	0,948	0,946	1,007	0,975	0,973	0,990	1,043	1,007	0,987	0,958	0,999	1,074	1,009	0,987	0,976	0,988	1,052
	0,4	1,000	0,973	1,043	1,195	1,171	1,431	1,008	1,107	1,163	1,288	1,392	1,057	1,111	1,277	1,275	1,420	1,101	1,083	1,144	1,306	1,545
	0,6	1,000	1,012	1,181	1,095	1,428	1,832	1,250	1,143	1,281	1,496	1,799	1,311	1,199	1,370	1,562	1,678	1,375	1,233	1,251	1,453	1,881
	0,8	1,000	1,347	2,004	2,321	2,750	3,629	1,639	1,836	2,402	3,307	4,273	1,810	1,892	2,434	3,149	4,303	2,071	2,200	2,604	3,388	4,311
0,9	1,000	2,225	4,317	5,143	6,112	6,603	2,836	5,568	6,459	6,866	6,937	3,291	4,250	6,687	6,806	6,994	3,351	4,185	6,649	6,778	6,952	
0,4	0,1	1,000	1,000	0,971	0,964	0,962	0,960	1,000	0,981	0,966	0,964	0,963	1,000	0,985	0,969	0,967	0,968	1,000	0,983	0,969	0,969	0,966
	0,2	1,000	1,000	0,946	0,916	0,908	0,903	1,000	0,946	0,924	0,909	0,907	1,000	0,947	0,929	0,912	0,910	1,000	0,953	0,929	0,915	0,912
	0,4	1,000	0,931	0,898	0,847	0,816	0,784	0,960	0,884	0,857	0,819	0,812	0,975	0,900	0,864	0,843	0,810	0,977	0,901	0,853	0,840	0,820
	0,6	1,000	0,729	0,637	0,657	0,688	0,670	0,795	0,594	0,584	0,661	0,769	0,810	0,600	0,650	0,612	0,762	0,836	0,625	0,617	0,664	0,749
	0,8	1,000	0,548	0,533	0,665	0,865	1,062	0,665	0,617	0,679	1,050	1,124	0,699	0,623	0,716	1,144	1,121	0,725	0,648	0,839	0,976	1,275
0,9	1,000	0,750	1,076	1,444	1,657	1,663	0,937	1,273	1,567	1,719	1,747	1,041	1,342	1,509	1,763	1,779	1,119	1,472	1,686	1,785	1,778	
0,6	0,1	1,000	1,000	1,039	1,054	1,080	1,101	1,000	1,041	1,055	1,082	1,111	1,000	1,041	1,055	1,082	1,113	1,000	1,043	1,057	1,087	1,113
	0,2	1,000	1,000	1,021	1,031	1,053	1,066	1,000	1,022	1,036	1,056	1,071	1,000	1,023	1,036	1,059	1,075	1,000	1,024	1,034	1,059	1,076
	0,4	1,000	1,000	0,955	0,942	0,944	0,956	1,000	0,960	0,943	0,950	0,961	1,000	0,964	0,946	0,954	0,962	1,000	0,961	0,952	0,953	0,964
	0,6	1,000	1,000	0,943	0,939	0,936	0,950	1,001	0,953	0,950	0,945	0,960	0,999	0,952	0,953	0,949	0,959	0,999	0,959	0,954	0,954	0,965
	0,8	1,000	1,000	0,994	0,994	0,993	0,997	1,000	0,994	0,994	0,996	0,999	1,000	0,993	0,994	1,000	0,996	1,000	1,003	0,997	0,998	0,996
0,9	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	
0,8	0,1	1,000	1,000	1,046	1,066	1,077	1,082	1,000	1,046	1,065	1,078	1,085	1,000	1,047	1,066	1,078	1,086	1,000	1,049	1,066	1,078	1,086
	0,2	1,000	1,000	1,029	1,043	1,047	1,054	1,000	1,036	1,044	1,048	1,056	1,000	1,036	1,046	1,050	1,058	1,000	1,040	1,049	1,050	1,058
	0,4	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000
	0,6	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000
	0,8	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000
0,9	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	
0,9	0,1	1,000	1,000	1,002	1,003	1,003	1,003	1,000	1,002	1,003	1,004	1,004	1,000	1,004	1,003	1,004	1,004	1,000	1,004	1,004	1,004	1,004
	0,2	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000
	0,4	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000
	0,6	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000
	0,8	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000
0,9	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	

Tabela 24: BSF com ociosidade grupo 2÷BSF com ociosidade grupo 3

PARÂMETROS NOM.		EXP-ET(desp,)/ EXP-ET(prior.)	BSF																			
Fator Atr.	Ampl. Distr.		1,1	1,3	1,5	1,8	1,12	3,1	3,3	3,5	3,8	3,12	5,1	5,3	5,5	5,8	5,12	8,1	8,3	8,5	8,8	8,12
0,1	0,1	0,921	1,003	0,995	1,009	1,055	1,103	1,008	0,997	1,021	1,128	1,223	1,008	1,013	1,025	1,132	1,236	1,011	1,017	1,041	1,143	1,225
	0,2	1,080	1,054	1,137	1,195	1,277	1,343	1,117	1,220	1,264	1,367	1,503	1,134	1,235	1,291	1,446	1,493	1,143	1,250	1,348	1,471	1,505
	0,4	1,772	1,682	1,720	1,803	1,922	2,148	1,816	1,882	1,935	2,018	2,281	1,845	1,855	2,004	2,114	2,490	1,899	1,783	1,937	2,296	2,410
	0,6	3,311	1,990	2,015	2,369	2,674	2,714	2,299	2,125	2,159	3,183	3,342	2,494	2,251	2,514	2,604	3,671	2,639	1,972	2,757	2,689	4,002
	0,8	8,563	3,873	4,717	6,735	10,191	10,732	4,517	6,096	7,629	10,621	10,921	5,068	6,868	9,215	11,136	11,688	5,267	7,058	7,320	11,635	11,929
	0,9	11,875	5,307	9,146	14,044	15,215	15,262	6,482	9,880	15,382	16,002	16,115	6,695	10,760	15,739	16,296	16,400	7,540	12,962	14,978	16,726	16,701
0,2	0,1	0,700	1,000	0,960	0,942	0,935	0,923	1,000	0,965	0,947	0,936	0,919	1,000	0,966	0,945	0,930	0,926	1,000	0,969	0,943	0,936	0,924
	0,2	0,735	0,999	0,969	0,940	0,948	0,946	1,007	0,975	0,973	0,990	1,043	1,007	0,987	0,958	0,999	1,074	1,009	0,987	0,976	0,988	1,052
	0,4	1,147	0,985	1,062	1,217	1,194	1,461	1,024	1,125	1,188	1,316	1,424	1,078	1,131	1,307	1,307	1,452	1,123	1,106	1,167	1,335	1,581
	0,6	1,960	1,155	1,345	1,240	1,627	2,085	1,432	1,300	1,453	1,695	2,052	1,511	1,365	1,564	1,789	1,914	1,587	1,402	1,425	1,667	2,148
	0,8	4,242	1,946	2,698	3,084	3,669	4,819	2,411	2,528	3,240	4,502	5,730	2,683	2,645	3,313	4,303	5,837	3,088	3,058	3,554	4,653	5,894
	0,9	7,739	3,622	6,695	7,952	9,523	10,086	4,799	9,151	10,163	10,621	10,801	5,655	7,026	10,643	11,023	11,113	5,781	7,024	10,817	11,058	11,253
0,4	0,1	0,503	1,000	0,971	0,964	0,962	0,960	1,000	0,981	0,966	0,964	0,963	1,000	0,985	0,969	0,967	0,968	1,000	0,983	0,969	0,969	0,966
	0,2	0,492	1,000	0,946	0,916	0,908	0,903	1,000	0,946	0,924	0,909	0,907	1,000	0,947	0,929	0,912	0,910	1,000	0,953	0,929	0,915	0,912
	0,4	0,630	0,931	0,898	0,847	0,816	0,784	0,960	0,884	0,857	0,819	0,812	0,975	0,900	0,864	0,843	0,810	0,977	0,901	0,853	0,840	0,820
	0,6	0,683	0,729	0,637	0,657	0,688	0,670	0,795	0,594	0,584	0,662	0,770	0,811	0,600	0,650	0,612	0,762	0,837	0,626	0,617	0,664	0,750
	0,8	1,082	0,582	0,562	0,704	0,919	1,127	0,708	0,652	0,719	1,114	1,202	0,745	0,660	0,759	1,216	1,198	0,776	0,687	0,891	1,039	1,365
	0,9	1,853	1,011	1,449	1,935	2,232	2,270	1,294	1,720	2,111	2,360	2,400	1,457	1,839	2,044	2,426	2,453	1,577	2,019	2,306	2,451	2,482
0,6	0,1	0,536	1,000	1,039	1,054	1,080	1,101	1,000	1,041	1,055	1,082	1,111	1,000	1,041	1,055	1,082	1,113	1,000	1,043	1,057	1,087	1,113
	0,2	0,583	1,000	1,021	1,031	1,053	1,066	1,000	1,022	1,036	1,056	1,071	1,000	1,023	1,036	1,059	1,075	1,000	1,024	1,034	1,059	1,076
	0,4	0,663	1,000	0,955	0,942	0,944	0,956	1,000	0,960	0,943	0,950	0,961	1,000	0,964	0,946	0,954	0,962	1,000	0,961	0,952	0,953	0,964
	0,6	0,779	1,000	0,943	0,939	0,936	0,950	1,001	0,953	0,950	0,945	0,960	0,999	0,952	0,953	0,949	0,959	0,999	0,959	0,954	0,954	0,965
	0,8	0,976	1,000	0,994	0,994	0,993	0,997	1,000	0,994	0,994	0,996	0,999	1,000	0,993	0,994	1,000	0,996	1,000	1,003	0,997	0,998	0,996
	0,9	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000
0,8	0,1	0,818	1,000	1,046	1,066	1,077	1,082	1,000	1,046	1,065	1,078	1,085	1,000	1,047	1,066	1,078	1,086	1,000	1,049	1,066	1,078	1,086
	0,2	0,879	1,000	1,029	1,043	1,047	1,054	1,000	1,036	1,044	1,048	1,056	1,000	1,036	1,046	1,050	1,058	1,000	1,040	1,049	1,050	1,058
	0,4	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000
	0,6	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000
	0,8	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000
	0,9	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000
0,9	0,1	0,993	1,000	1,002	1,003	1,003	1,003	1,000	1,002	1,003	1,004	1,004	1,000	1,004	1,003	1,004	1,004	1,000	1,004	1,004	1,004	1,004
	0,2	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000
	0,4	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000
	0,6	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000
	0,8	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000
	0,9	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000

Tabela 25: BSF sem ociosidade grupo 1+BSF com ociosidade grupo 3

Figura 19: influência de b e f no desempenho da busca - grupo 1

Figura 20: média geral da influência de b e f no desempenho da busca - grupo 1

Figura 21: influência de b e f no desempenho da busca - grupo 2

Figura 22: média geral da influência de b e f no desempenho da busca - grupo 2

Figura 23: influência de b e f no desempenho da busca - grupo 3

Figura 24: média geral da influência de b e f no desempenho da busca - grupo 3

ANEXO D: DEMONSTRAÇÃO DO TEOREMA

Para a demonstração do teorema, por uma questão de simplicidade notacional será omitida a seqüência na qual o programa está sendo construído, ou seja, o programa parcial $\pi_j(\sigma)$ será representado por π_j . A j -ésima ordem em uma seqüência será representada por J_j e não mais por $J_{\sigma(j)}$. A representação do valor da função objetivo de um programa parcial até as primeiras j ordens será $g(\pi_j) = \sum_{i=1}^j (h_i E_i + w_i T_i)$ e de um programa completo será $g(\pi) = \sum_{i=1}^n (h_i E_i + w_i T_i)$. Os valores das penalidades h_j e w_j serão sempre positivos.

TEOREMA (baseado no teorema 2 de Garey et al. (1988)). Para qualquer j , o programa parcial π_j calculado pelo algoritmo tem o mínimo $g(\pi_j)$ dentre todos os programas possíveis para as primeiras j ordens.

DEMONSTRAÇÃO. Como hipótese de indução, assume-se que em j o teorema é verdadeiro. Para o programa vazio π_0 , o teorema é verdadeiro. Portanto só falta provar que para $j+1$, o teorema também é verdadeiro. Seja π_{j+1}^* algum programa das primeiras $j+1$ ordens e seja e_i e e_i^* os horários efetivos de início da ordem J_i em π_j e π_{j+1}^* , respectivamente. Deve-se considerar dois casos:

Caso sem atraso. Se $e_j + p_j \leq a_{j+1}$, então para π_{j+1} não haverá aumento na função-objetivo ($g(\pi_{j+1}) = g(\pi_j)$) pois a ordem J_{j+1} é programada para iniciar em a_{j+1} . Também pode-se perceber que $E_{j+1} = T_{j+1} = 0$. Mas $g(\pi_{j+1}^*) \geq g(\pi_j)$ pois π_{j+1}^* sem a ordem J_{j+1} , é um programa para as primeiras j ordens, tendo a função objetivo com valor de no mínimo $g(\pi_j)$ pela hipótese de indução, e com $E_{j+1}^* \geq 0$ ou $T_{j+1}^* \geq 0$ (e na verdade não só para π_{j+1}^* mas para qualquer outro programa). Daí conclui-se que $g(\pi_{j+1}^*) \geq g(\pi_{j+1})$ e portanto o teorema é verdadeiro.

Caso com atraso. Se $e_j + p_j > a_{j+1}$, então haverá atraso e portanto aumento no valor da função objetivo. Seja π_{j+1} o programa formado por π_j acrescido da ordem J_{j+1} iniciada em $e_j + p_j$. Então $g(\pi_{j+1}) = g(\pi_j) + w_{j+1}(e_j + p_j - a_{j+1}) = g(\pi_j) + w_{j+1}T_{j+1}$. Para a comparação entre $g(\pi_{j+1})$ e $g(\pi_{j+1}^*)$ deve-se considerar dois casos:

Caso 1 com atraso: Se $e_{j+1}^* \geq e_j + p_j$, então $g(\pi_{j+1}^*) \geq g(\pi_j) + w_j(e_{j+1}^* - a_{j+1}) \geq g(\pi_j) + w_j(e_j + p_j - a_{j+1})$ e portanto $g(\pi_{j+1}^*) \geq g(\pi_{j+1})$ pois $g(\pi_{j+1}) = g(\pi_j) + w_j(e_j + p_j - a_{j+1}) = g(\pi_j) + w_{j+1}T_{j+1}$, fazendo com que o teorema seja verdadeiro.

Caso 2 com atraso: Para o último caso, isto é, $e_{j+1}^* < e_j + p_j$, deve-se utilizar as estruturas de blocos, conforme explicado no item 4.2. Seja B_l o último bloco em π_{j+1} , que é equivalente ao último bloco de π_j mais a ordem J_{j+1} pois não há adiantamento e, por consequência, não há criação de um bloco novo. Pelo desenvolvimento do algoritmo sabe-se que a relação $W(l) < H(l)$ deve ser mantida para os valores com relação à π_{j+1} . Desde que cada ordem em B_l não é programada mais tarde em π_{j+1} do que em π_{j+1}^* , o valor da função objetivo no bloco l , $\sum_{i \in B(l)} (h_i E_i + w_i T_i)$, em π_{j+1} não será maior do que em π_{j+1}^* . Como pela hipótese de indução, até a ordem $\text{Primeiro}(l)-1$, as funções objetivo dos programas eram equivalentes, isto é, $g(\pi_{\text{Primeiro}(l)-1}^*) = g(\pi_{\text{Primeiro}(l)-1})$, o teorema continua sendo verdadeiro. ■

REFERÊNCIAS BIBLIOGRÁFICAS

- ABDUL-RAZAQ, T. S.; POTTS, C. N. Dynamic programming state-space relaxation for single machine scheduling. **Journal of Operational Research Society**, v.39, n.2, p.141-52, 1988.
- ADAMS, Joseph; BALAS, Egon; ZAWACK, Daniel. The shifting bottleneck procedure for job shop scheduling. **Management Science**, v.34, n.3, p.391-401, 1988.
- ANTHONY, Robert N. **Planning and control systems: a framework for analysis**. Princeton, Harvard University, Graduate School of Business Administration, 1965.
- AOKI, Masahiko. Toward an economic model of the Japanese firm. **Journal of Economic Literature**, v.28, p.1-27, 1990.
- BAKER, Kenneth R. **Introduction to sequencing and scheduling**. New York, Wiley, 1974.
- _____; MARTIN, James B. An experimental comparison of solution algorithms for the single-machine tardiness problem. **Naval Research Logistics Quarterly**, v.21, n.1, p.187-99, 1974.
- _____; SCUDDER, Gary D. Sequencing with earliness and tardiness penalties: a review. **Operations Research**, v.38, n.1, p.22-36, 1990.
- BARTLE, Robert G. **Elementos de análise real**. Tradução de Alfredo A. de Farias. Rio de Janeiro, Campus, 1983.
- BHASKARAN, Kumar; PINEDO, Michael. Dispatching. In: SALVENDY, Gavriel (ed.) **Handbook of industrial engineering**. 2. ed. New York, Wiley, 1992. p.2182-98.
- BITRAN, Gabriel R.; TIRUPATI, Devanath. Hierarchical production planning. In: GRAVES, S. C.; RINNOOY KAN, A. H. G.; ZIPKIN, P. H. (eds.) **Logistics of production and inventory**. Amsterdam, North-Holland, 1993. p.523-70 (Handbooks in Operations Research and Management Science, v. 4)
- BUFFA, Elwood S. **Basic production management**. New York, Wiley, 1971.
- CAMPELLO, Ruy E.; MACULAN, Nelson. **Algoritmos e heurísticas: desenvolvimento e avaliação de performance**. Niterói, EDUFF, 1994.
- CHANG, Yih-Long; MATSUO, Hirofumi; SULLIVAN, Robert S. A bottleneck-based beam search for job shop scheduling in a flexible manufacturing system. **International Journal of Production Research**, v.27, n.11, p.1949-61, 1989.
- CHENG, T. C. E.; GUPTA, M. C. Survey of scheduling research involving due dates determination decisions. **European Journal of Operational Research**, v.38, p.156-66, 1989.
- COFFIN, M. A.; TAYLOR III, B. W. R&D project selection and scheduling with filtered beam search approach. **IIE Transactions**, v.26, p.167-76, 1996.
- COLIN, Emerson C. Estudo da implementação do sistema kanban em uma empresa brasileira de autopeças: dificuldades e caminhos. In: ENCONTRO NACIONAL DE ENGENHARIA DE PRODUÇÃO, 16., Piracicaba, 1996. **Anais**. Piracicaba, ABEPRO, UNIMEP, 1996. /CD-Rom/
- _____. Bem aplicado, o sistema kanban elimina o atraso nos prazos de entrega. **Máquinas e Metais**, v. 32, n.375, p.260-9, 1997.
- _____; QUININO, Roberto da C.; SHIMIZU, Tamio. Manufatura ágil: um novo conceito na produção industrial. In: REUNIÃO ANUAL DA SOCIEDADE

- BRASILEIRA DE PESQUISADORES NIKKEIS, 5., Águas de Lindóia, 1997.
Anais. Águas de Lindóia, SBPN, 1997. p.170-5.
- CONWAY, Richard W.; MAXWELL, William L.; MILLER, Louis W. **Theory of scheduling**. Reading, Massachusetts, Addison-Wesley, 1967.
- CORRÊA, Henrique L.; GIANESI, Irineu G. **Just-in-time, MRPII e OPT: um enfoque estratégico**. São Paulo, Atlas, 1993.
- DAUZERE-PERES, S.; LASSERRE, J.-B. A modified shifting bottleneck procedure for job-shop scheduling. **International Journal of Production Research**, v.31, n.4, p.923-932, 1993.
- DE, Prabuddha; GHOSH, Jay B.; WELLS, Charles E. Heuristic estimation of the efficient frontier for a bi-criteria scheduling problem. **Decision Sciences**, v.23, n.3, p.596-609, 1992.
- DWYER, Jerry; WILLIAMS, K. B. Testing random number generators. **C/C++ Users Journal**, v.14, n.6, p.39-48, 1996.
- FISHER, M. L. A dual algorithm for the one-machine scheduling problem. **Mathematical Programming**, v.11, p.229-51, 1976.
- _____; RINNOOY KAN, Alexander H. G. The design, analysis and implementation of heuristics. **Management Science**, v.34, n.3, p.263-265, 1988.
- FRENCH, Simon. **Sequencing and scheduling: an introduction to the mathematics of the job-shop**. New York, Wiley, 1982.
- FRY, Timothy D.; ARMSTRONG, Ronald D.; BLACKSTONE, John H. Minimizing weighted absolute deviation in single machine scheduling. **IIE Transactions**, v.19, n.4, p.445-50, 1987.
- _____; _____. DARBY-DOWMAN, Kenneth; PHILIPPOU, Patrick R. A branch and bound procedure to minimize mean absolute lateness on a single processor. **Computers and Operations Research**, v.23, n.2, p.171-82, 1996.
- GAREY, Michael R.; TARJAN, Robert E.; WILFONG, Gordon T. One-processor scheduling with symmetric earliness and tardiness penalties. **Mathematics of Operations Research**, v.13, n.2, p.330-348, 1988.
- _____; JOHNSON, David S. **Computers and intractability: a guide to the theory of NP-Completeness**. New York, Freeman & Company, 1979.
- GRAHAM, R. L.; *et al.* Optimization and approximation in deterministic sequencing and scheduling: a survey. **Annals of Discrete Mathematics**, v.5, p.287-326, 1979.
- GROENEVELT, Harry. The just-in-time system. In: GRAVES, S. C.; RINNOOY KAN, A. H. G.; ZIPKIN, P. H. (eds.) **Logistics of production and inventory**. Amsterdam, North-Holland, 1993. p.629-70 (Handbooks in Operations Research and Management Science, v. 4)
- HAX, Arnaldo C.; CANDEA, Dan. **Production and inventory management**. Englewood Cliffs, Prentice-Hall, 1984.
- JAPAN MANAGEMENT ASSOCIATION. **Kanban just-in-time at Toyota: management begins at the workplace**. Translated by David J. Lu. Portland, Productivity, 1989.
- JOHNSON, R. E.; KIOKEMEISTER, F. L. **Calculo con geometria analitica**. Traducido por Andres S. Bouclier. México, Compañía Editorial Continental, 1977.
- KOULAMAS, Christos P. Scheduling two parallel semiautomatic machines to minimize machine interference. **Computers and Operations Research**, v.23, n.10, p.945-56, 1996.
- LAGEWEG, B. J.; *et al.* Computer-aided complexity classification of combinatorial problems. **Communications of the ACM**, v.25, n.11, p.817-22, 1982.

- LAWLER, Eugene L.; *et al.* Sequencing and scheduling: algorithms and complexity. In: GRAVES, S. C.; RINNOOY KAN, A. H. G.; ZIPKIN, P. H. (eds.) **Logistics of production and inventory**. Amsterdam, North-Holland, 1993. p.445-522 (Handbooks in Operations Research and Management Science, v. 4)
- LENSTRA, J. K.; RINNOOY KAN, A. H. G. Computational complexity of discrete optimization problems. **Annals of Discrete Mathematics**, v.4, p.121-40, 1979.
- MONDEN, Yasuhiro. **Produção sem estoques**: uma abordagem prática do sistema de produção da Toyota. São Paulo, IMAM, 1984.
- MONMA, C. L. Sequencing to minimize the maximum job cost. **Operations Research**, v.28, n.4, p.942-51, 1980.
- MORTON, Thomas E.; PENTICO, David W. **Heuristic scheduling systems**: with applications to production systems and project management. New York, Wiley, 1993.
- NAIR, S. K.; THAKUR, L. S.; WEN, K. -W. Near optimal solutions for product line design and selection: beam search heuristics. **Management Science**, v.41, n.5, p.767-85, 1995.
- NILSSON, Nils J. **Problem-solving methods in artificial intelligence**. New York, McGraw-Hill, 1971.
- O'BRIEN, Stephen K. **Turbo pascal 6**: completo e total. Tradução e revisão técnica ARX publicações Ltda. São Paulo, Makron, McGraw-Hill, 1993.
- OW, Peng S.; MORTON, Thomas E. Filtered beam search in scheduling. **International Journal of Production Research**, v.26, n.1, p.35-62, 1988.
- _____; _____. The single machine early/tardy problem. **Management Science**, v.35, n.2, p.177-91, 1989.
- PANWALKAR, S. S.; ISKANDER, Wafik. A survey of scheduling rules. **Operations Research**, v.25, n.1, 1977.
- POTTS, C. N.; Van WASSENHOVE, L. N. A branch and bound algorithm for the total weighted tardiness problem. **Operations Research**, v.33, n.2, p.363-77, 1985.
- RACHAMADUGU, Ram. Scheduling jobs with proportionate early/tardy penalties. **IIE Transactions**, v.27, p.679-82, 1995.
- RINNOOY KAN, A. H. G.; LAGEWEG, B. J.; LENSTRA, J. K. Minimizing total costs in one-machine scheduling. **Operations Research**, v.23, n.5, p.908-27, 1975.
- _____; MAGAZINE, M. J. Report of the session on scheduling. **Annals of Discrete Mathematics**, v.5, p.423-6, 1979.
- SCHONBERGER, Richard J. Just-In-Time production systems: replacing complexity with simplicity in manufacturing management. In: JACOBS, F. Robert; MABERT, Vicent A. (eds) **Production planning, scheduling, and inventory control**: concepts, techniques, and systems. 3. ed. Atlanta, Institute of Industrial Engineers, 1986a, p.101-7.
- _____. Some observations on the advantages and implementation issues of just-in-time production systems. In: JACOBS, F. Robert; MABERT, Vicent A. (eds) **Production planning, scheduling, and inventory control**: concepts, techniques, and systems. 3. ed. Atlanta, Institute of Industrial Engineers, 1986b, p.108-18.
- SHIH, Heloisa M.; SEKIGUSHI, Takashi. FMS scheduling with routing flexibility: a beam search and petri net based scheduling system. In: INTERNATIONAL CONFERENCE ON INDUSTRIAL & ENGINEERING APPLICATIONS OF ARTIFICIAL INTELLIGENCE AND EXPERT SYSTEMS, 4., s.l., 1991. **Proceedings**. S.n.t.

- SHIH, Li C. **AI search techniques and applications to production management problems**. Tokyo, 1991. 123p. Doctoral Thesis - Department of Industrial Engineering and Management, Tokyo Institute of Technology.
- SHIMIZU, Tamio. **Introdução à ciência da computação**. 2. ed. São Paulo, Atlas, 1988.
- SKINNER, Wickham. **Manufacturing in the corporate strategy**. New York, John Wiley, 1978.
- _____. Manufacturing-missing link in corporate strategy. **Harvard Business Review**, may-june, 1969.
- SLOTNICK, Susan A.; MORTON, Thomas E. Selecting jobs for a heavily loaded shop with lateness penalties. **Computers and Operations Research**, v.23, n.2, p.131-40, 1996.
- SMITH, Wayne E. Various optimizers for single-stage production. **Naval Research Logistics Quarterly**, v.3, p.59-66, 1956.
- SRINIVASAN, V. A hybrid algorithm for the one machine sequencing problem to minimize total tardiness. **Naval Research Logistics Quarterly**, v.18, n.3, p.317-27, 1971.
- TERADA, Routo. **Introdução à complexidade de algoritmos paralelos**. São Paulo, IME-USP, 1990.
- WILKERSON, L. J.; IRWIN, J. D. An improved method for scheduling independent tasks. **AIIE Transactions**, v.3, n.3, p.239-45, 1971.
- WINSTON, Patrick H. **Artificial Intelligence**. 3. ed. Reading, Massachusetts, Addison-Wesley, 1992.
- WOMACK, James P.; JONES, Daniel T.; ROOS, Daniel. **A máquina que mudou o mundo**. 4. ed. Rio de Janeiro, Campus, 1992.
- WOOD, Steve. **Turbo pascal**: guia do usuário. Tradução Maria C. S. R. Ratto. São Paulo, McGraw-Hill, 1987.
- YANO, Candace A.; KIM, Yeong-Dae. Algorithms for a class of single-machine weighted tardiness and earliness problems. **European Journal of Operational Research**, v.52, p.167-78, 1991.

APÊNDICE 1: PROGRAMAS COMPUTACIONAIS

Este apêndice apresenta os programas computacionais em Turbo Pascal 6.0 utilizados nos experimentos propostos na dissertação. Os arquivos “.INC” são arquivos de inclusão e são utilizados como procedimentos de outros arquivos. Os arquivos “.PAS” são os arquivos utilizados para a geração dos executáveis “.EXE”.

Todos os programas e procedimentos estão ordenados alfabeticamente. A função dos programas foi apresentada no capítulo 5 e outros comentários são apresentados no corpo dos procedimentos e programas. Os caracteres em negrito destacam inícios, termos e nomes de procedimentos que são utilizados em outros programas ou em outros procedimentos. Os textos entre chaves “{...}” - excluindo os casos “{\$...}”, que fazem parte do código - são instruções e comentários que facilitam o entendimento do programa. As referências utilizadas para a elaboração dos códigos deste apêndice são WOOD (1987), SHIMIZU (1988) e O'BRIEN (1993).

ARQUIVO FAMINPI.INC

```
function ApagaMin(var X:ConjOrdenado):integer;
{funcao que encontra um item de minima chave na matriz e apresenta-o
 (o indice), removendo o par de classificacao da matriz}
var
    I           :integer;
    MinChave:real;
    Valor       :integer;
    PontoRetirada:integer;
begin
    MinChave:=EncontraMin(X);
    for I:=1 to X.NumAtivos do
        if (X.MatrizPrior[I].CritOrd = MinChave) then
            begin
                Valor:=X.MatrizPrior[I].Indice;
                PontoRetirada:=I;
            end;
    if (X.NumAtivos > 1) then
        begin
            for I:=PontoRetirada to X.NumAtivos-1 do
                X.MatrizPrior[I]:=X.MatrizPrior[I+1];
            X.NumAtivos:=X.NumAtivos-1;
        end
    else
        X.NumAtivos:=0;
    ApagaMin:=Valor;
end;
```

ARQUIVO FEMINPI.INC

```
function EncontraMin(X:ConjOrdenado):real;
{funcao que encontra um item de minimo criterio de ordenacao (chave) da
 pilha X e apresenta essa chave sem remover o item da pilha}
var
    I           :integer;
    ValorMinimo:real;
begin
    if (X.NumAtivos <= 0) then
        begin
            writeln('aconteceu um erro em algum bloco de ordens!');
            writeln(X.NumAtivos);
            halt;
        end;
end;
```

```

if (X.NumAtivos = 1) then
  EncontraMin:=X.MatrizPrior[1].CritOrd
else
  begin
    for I:=1 to X.NumAtivos-1 do
      ValorMinimo:=MenorReal(X.MatrizPrior[I].CritOrd,X.MatrizPrior[I+1].CritOrd);
    EncontraMin:=ValorMinimo;
  end;
end;

```

ARQUIVO FHORARIO.INC

```

function Horario:longint;
{funcao que le o horario em que a mesma e solicitada (em centesimos de
segundo correntes)}
var
  Hora,Min,Seg,CentSeg :word;
begin
  gettime(Hora,Min,Seg,CentSeg);
  Horario:=CentSeg+100*Seg+6000*Min;
end;

```

ARQUIVO FMAREAL.INC

```

function MaiorReal(X,Y:real):real;
{funcao que calcula o maior valor entre dois valores reais}
begin
  if (X > Y) then
    MaiorReal:=X
  else
    MaiorReal:=Y;
end;

```

ARQUIVO FMAXINT.INC

```

function MaximoInteiro(X,Y:integer):integer;
{funcao que calcula o maximo de dois valores inteiros}
begin
  if (X > Y) then
    MaximoInteiro:=X
  else
    MaximoInteiro:=Y;
end;

```

ARQUIVO FMERREAL.INC

```

function MenorReal(X,Y:real):real;
{funcao que calcula o menor valor entre dois valores reais}
begin
  if (X < Y) then
    MenorReal:=X
  else
    MenorReal:=Y;
end;

```

ARQUIVO FMININT.INC

```

function MinimoInteiro(X,Y:integer):integer;
{funcao que calcula o minimo de dois valores inteiros}
begin
  if (X < Y) then
    MinimoInteiro:=X
  else
    MinimoInteiro:=Y;
end;

```

ARQUIVO FOCIEET.INC

```

function OciosidadeExpET(X:ConjOrdenado;Z:real):real;
{funcao que insere ociosidade de acordo com a funcao de prioridade EXP-
ET.}
{X=conj. ordenado que estamos utilizando como sequencia
Z=tempo de processo medio das ordens disponiveis para serem programadas}
var
  I :integer; {indice do loop}
  MOciExpET:array[1..NumOrdensMaximo] of real; {matriz de ociosidade}

```

```

    OciosidadeMinima:real;
begin
    OciosidadeMinima:=maxint;
    for I:=1 to X.NumAtivos do
        begin
            MOciExpET[I]:=int(Dados[X.MatrizPrior[I].Indice].Data-
                Dados[X.MatrizPrior[I].Indice].TProces-
                (K*Z*Dados[X.MatrizPrior[I].Indice].PenAtr)/
                (Dados[X.MatrizPrior[I].Indice].PenAtr+
                Dados[X.MatrizPrior[I].Indice].PenAdi))+1;
            OciosidadeMinima:=MenorReal(OciosidadeMinima,MOciExpET[I]);
        end;
    OciosidadeExpET:=OciosidadeMinima;
end;

```

ARQUIVO FPREXPET.INC

```

function PrioridadeExpET(X,Y:real;Z:integer):real;
{funcao que calcula a prioridade ExpET. X=folga, Y=Pmed, Z=ordem}
var
    Simplific :real;
begin
    if (X <= (Dados[Z].PenAtr*K*Y)/(Dados[Z].PenAtr+Dados[Z].PenAdi)) then
        begin
            X:=MaiorReal(0.0,X);
            PrioridadeExpET:=Dados[Z].PenAtr/Dados[Z].TProces*
                exp(-(Dados[Z].PenAdi+Dados[Z].PenAtr)/Dados[Z].PenAdi)*(X/Y));
        end
    else
        begin
            if (K*Y < X) then
                X:=MenorReal(X,K*Y);
            Simplific:=(Dados[Z].PenAtr/Dados[Z].PenAdi)-
                ((Dados[Z].PenAdi+Dados[Z].PenAtr)*X/(Dados[Z].PenAdi*K*Y));
            PrioridadeExpET:=(Dados[Z].PenAdi/Dados[Z].TProces)*
                Simplific*Simplific*Simplific;
        end;
    end;
end;

```

ARQUIVO FREPLIC.INC

```

function Replicate(C:char; Vezes:integer):string;
{funcao que acrescenta um caractere varias vezes seguidas}
var
    Auxiliar :string;
    VezesReal :integer;
begin
    VezesReal:=MinimoInteiro(Vezes,255);
    if (VezesReal <= 0) then
        Replicate:=''
    else
        begin
            Auxiliar:='';
            while (VezesReal > 0) do
                begin
                    Auxiliar:=C+Auxiliar;
                    VezesReal:=VezesReal-1;
                end;
            Replicate:=Auxiliar;
        end;
    end;
end;

```

ARQUIVO FSTRZERO.INC

```

function StrZero(N,Comprimento:integer):string;
var
    Auxiliar:string;
begin
    Str(N:0,Auxiliar);
    if (length(auxiliar) < Comprimento) then
        begin
            Auxiliar:=Replicate('0',Comprimento-Length(Auxiliar))+Auxiliar;
        end;
    StrZero:=Auxiliar;
end;

```

ARQUIVO FTPROMED.INC

```

function TProcesMedio(Conjunto:ConjOrdenado):real;
{funcao que calcula o tempo de processo medio das ordens pertencentes a
um conjunto ordenado}
var
  I      :integer; {indice do loop}
  SomaTProces :array[1..NumOrdensMaximo] of real;
           {soma dos tempos de processamento}
begin
  SomaTProces[1]:=Dados[Conjunto.MatrizPrior[1].Indice].TProces;
  for I:=2 to Conjunto.NumAtivos do
    SomaTProces[I]:=SomaTProces[I-1]+
      Dados[Conjunto.MatrizPrior[I].Indice].TProces;
  TProcesMedio:=(1/Conjunto.NumAtivos)*SomaTProces[Conjunto.NumAtivos];
end;

```

ARQUIVO PAPTCOPI.INC

```

procedure AdicionaParaTodosCritOrd(X:real;var Y:ConjOrdenado);
{procedimento que adiciona X para todas as chaves da pilha Y}
var
  I      :integer;
begin
  for I:=1 to Y.NumAtivos do
    Y.MatrizPrior[I].CritOrd:=Y.MatrizPrior[I].CritOrd+X;
end;

```

ARQUIVO PCENPP.INC

```

procedure ColocaElemNaPrimPos(Elemento      :ParClassificacao;
                               var Conjunto:ConjOrdenado);
{procedimento que coloca um determinado elemento de uma matriz do tipo par
de classificacao na primeira posicao da matriz e reordenando-a de forma
que os outros elementos permaneçam com a mesma sequencia}
var
  I      :integer;
  PosRetirada:integer;
begin
  PosRetirada:=0;
  for I:=1 to Conjunto.NumAtivos do
    if (Conjunto.MatrizPrior[I].Indice = Elemento.Indice) then
      PosRetirada:=I;
  if (PosRetirada = 0) then {caso o elem. a ser retirado nao seja encontrado}
  begin
    writeln('Nao foi encontrado o elemento a ser retirado do conjunto');
    writeln('ordenado. Aconteceu algum erro no programa. Verificar antes');
    writeln('de continuar!');
    writeln;
    halt;
  end
  else {caso o elemento a ser retirado seja encontrado}
  begin
    for I:=PosRetirada downto 2 do
      Conjunto.MatrizPrior[I]:=Conjunto.MatrizPrior[I-1];
    Conjunto.MatrizPrior[1]:=Elemento;
  end;
end;

```

ARQUIVO PCFOBD.CO.INC

```

procedure CalcFObjDeConjOrd(X:ConjOrdenado; Y:integer);
{procedimento que calcula o valor da fun#Eo objetivo de acordo com uma se-
quencia definida pelo conjunto ordenado. X=conjunto ordenado que define
a sequencia dos dados de entrada. Y=numero de elementos da sequencia par-
cial fixa}
var
  C,L      :array[1..NumOrdensMaximo] of real;
           {C=horario de termino; L=diferen#a}
  E,T      :array[1..NumOrdensMaximo] of real;
           {E=adiantamento; T=atraso}
  HE,WT    :array[1..NumOrdensMaximo] of real;
           {HE=peso adiantamento vezes adiantamento; HT=peso atraso vezes atraso}
  FuncObj  :array[1..NumOrdensMaximo] of real;
           {funcao objetivo em cada ordem da sequencia}
  I      :integer; {indice do loop}
{$M 40960,0,630784}
begin

```

```

E[1]:=0;
T[1]:=0;
C[1]:=Dados[X.MatrizPrior[1].Indice].TProces+
    Dados[X.MatrizPrior[1].Indice].Chegada;
L[1]:=C[1]-Dados[X.MatrizPrior[1].Indice].Data;
if (L[1] < 0) then
    E[1]:=-L[1]
else
    T[1]:=L[1];
HE[1]:=Dados[X.MatrizPrior[1].Indice].PenAdi*E[1];
WT[1]:=Dados[X.MatrizPrior[1].Indice].PenAtr*T[1];
FuncObj[1]:=HE[1]+WT[1];
for I:=2 to NumOrdens do
begin
    E[I]:=0;
    T[I]:=0;
    C[I]:=C[I-1]+Dados[X.MatrizPrior[I].Indice].TProces;
    L[I]:=C[I]-Dados[X.MatrizPrior[I].Indice].Data;
    if (L[I] < 0) then
        E[I]:=-L[I]
    else
        T[I]:=L[I];
    HE[I]:=Dados[X.MatrizPrior[I].Indice].PenAdi*E[I];
    WT[I]:=Dados[X.MatrizPrior[I].Indice].PenAtr*T[I];
    FuncObj[I]:=FuncObj[I-1]+HE[I]+WT[I];
end;
writeln;
ValFuncObjetivo:=FuncObj[NumOrdens];
HorTer:=C[Y];
end;

```

ARQUIVO PCFOBJ.INC

```

procedure CalcFObj (MatrizDados:TAEntrada;NumElemSeqParFixa:integer);
{procedimento que calcula o valor da fun#Eo objetivo}
var
    C,L          :array[1..NumOrdensMaximo] of real;
    {C=horario de termino; L=diferen#a}
    E,T          :array[1..NumOrdensMaximo] of real;
    {E=adiantamento; T=atraso}
    HE,WT        :array[1..NumOrdensMaximo] of real;
    {HE=peso adiantamento vezes adiantamento; HT=peso atraso vezes atraso}
    FuncObj       :array[1..NumOrdensMaximo] of real;
    {funcao objetivo em cada ordem da sequencia}
    I             :integer; {indice do loop}
{$M 40960,0,630784}
begin
    E[1]:=0;
    T[1]:=0;
    C[1]:=MatrizDados[1].TProces+MatrizDados[1].Chegada;
    L[1]:=C[1]-MatrizDados[1].Data;
    if (L[1] < 0) then
        E[1]:=-L[1]
    else
        T[1]:=L[1];
    HE[1]:=MatrizDados[1].PenAdi*E[1];
    WT[1]:=MatrizDados[1].PenAtr*T[1];
    FuncObj[1]:=HE[1]+WT[1];
    for I:=2 to NumOrdens do
        begin
            E[I]:=0;
            T[I]:=0;
            C[I]:=C[I-1]+MatrizDados[I].TProces;
            L[I]:=C[I]-MatrizDados[I].Data;
            if (L[I] < 0) then
                E[I]:=-L[I]
            else
                T[I]:=L[I];
            HE[I]:=MatrizDados[I].PenAdi*E[I];
            WT[I]:=MatrizDados[I].PenAtr*T[I];
            FuncObj[I]:=FuncObj[I-1]+HE[I]+WT[I];
        end;
    writeln;
    ValFuncObjetivo:=FuncObj[NumOrdens];
    HorTer:=C[NumElemSeqParFixa];
end;

```

ARQUIVO PCFOBOCI.INC

```

procedure CalcFObjOci(X,Y:ConjOrdenado;Z:integer);
{procedimento que calcula o valor da fun#Eo objetivo de um programa, le-
vando em consideracao a insercao de ociosidade}
{X=conjunto que define a sequencia, Y=conj. que define as ociosidades,
Z=numero de elementos da sequencia paracial fixa}
var
  C,L      :array[1..NumOrdensMaximo] of real; {C=horario de termino;
  L=diferen#a}
  E,T      :array[1..NumOrdensMaximo] of real; {E=adiantamento; T=atraso}
  HE,WT    :array[1..NumOrdensMaximo] of real; {HE=peso adiantamento vezes
  adiantamento; HT=peso atraso vezes atraso}
  FuncObj  :array[1..NumOrdensMaximo] of real; {funcao objetivo}
  I        :integer; {indice do loop}
{$M 40960,0,630784}
begin
  E[1]:=0;
  T[1]:=0;
  C[1]:=Dados[X.MatrizPrior[1].Indice].TProces+Y.MatrizPrior[1].CritOrd;
  L[1]:=C[1]-Dados[X.MatrizPrior[1].Indice].Data;
  if (L[1] < 0) then
    E[1]:=-L[1]
  else
    T[1]:=L[1];
  HE[1]:=Dados[X.MatrizPrior[1].Indice].PenAdi*E[1];
  WT[1]:=Dados[X.MatrizPrior[1].Indice].PenAtr*T[1];
  FuncObj[1]:=HE[1]+WT[1];
  for I:=2 to X.NumAtivos do
    begin
      E[I]:=0;
      T[I]:=0;
      C[I]:=C[I-1]+Dados[X.MatrizPrior[I].Indice].TProces+
      Y.MatrizPrior[I].CritOrd;
      L[I]:=C[I]-Dados[X.MatrizPrior[I].Indice].Data;
      if (L[I] < 0) then
        E[I]:=-L[I]
      else
        T[I]:=L[I];
      HE[I]:=Dados[X.MatrizPrior[I].Indice].PenAdi*E[I];
      WT[I]:=Dados[X.MatrizPrior[I].Indice].PenAtr*T[I];
      FuncObj[I]:=FuncObj[I-1]+HE[I]+WT[I];
    end;
  writeln;
  ValFuncObjetivo:=FuncObj[X.NumAtivos];
  HorTer:=C[Z];
end;

```

ARQUIVO PIEFSPF.INC

```

procedure InsElemFimSeqParFixa(X:ConjOrdenado; Y:ParClassificacao;
  var Z:ConjOrdenado);
{procedimento que insere um determinado elemento apos a sequencia parcial
fixa, retirando-o da sequencia parcial complementar}
{X=conjunto que vai servir como sequencia parcialfixa e como sequencia par-
cial complementar; Y=elemento que se deseja colocar apos a sequencia par-
cial fixa; Z=conjunto em que a nova sequencia completa e apresentado}
var
  I      :integer;
begin
  Z:=X;
  RetiraDeConjOrd(Y,Z);
  for I:=Z.NumAtivos downto X.PosCorr+1 do
    Z.MatrizPrior[I+1]:=Z.MatrizPrior[I];
  Z.MatrizPrior[X.PosCorr+1]:=Y;
  Z.NumAtivos:=Z.NumAtivos+1;
end;

```

ARQUIVO PINFCO.INC

```

procedure InsereNoFimConjOrd(Elemento :ParClassificacao;
  var Conjunto :ConjOrdenado);
{procedimento que insere um elemento no final de um conjunto ordenado}
begin
  if (conjunto.NumAtivos = 0) then {se o conjunto for vazio}
    begin
      Conjunto.MatrizPrior[1]:=Elemento;
    end;

```

```

    Conjunto.NumAtivos:=Conjunto.NumAtivos+1;
end
else {se o conjunto ordenado nao estiver vazio}
begin
    Conjunto.NumAtivos:=Conjunto.NumAtivos+1;
    Conjunto.MatrizPrior[Conjunto.NumAtivos]:=Elemento;
end;
end;
end;

```

ARQUIVO PINSOCI3.INC

```

procedure InsereOciosidade(X: ConjOrdenado);
{procedimento que insere ociosidade otima entre as ordens cuja sequencia
 e especificada na entrada do procedimento, ou seja, dada uma sequencia
 pre-definida, insere-se a ociosidade otima}
{X=sequencia completa na qual deseja-se inserir a ociosidade}
var
    HorDeseIni      :array [1..NumOrdensMaximo] of real; {hor desejado de inicio}
    HorEfetIni      :array [1..NumOrdensMaximo] of real; {hor efetivo de inicio}
    I,J             :integer; {indice do loop}
    Primeiro        :array [1..NumMaxPilha] of integer; {indice do primei-
ro elemento de um determinado bloco}
    SomaPenAdi      :array [1..NumMaxPilha] of real; {soma das penali-
dades de adiantamento de um determinado bloco}
    SomaPenAtr      :array [1..NumMaxPilha] of real; {soma das penalidades
de atraso de um determinado bloco}
    Ultimo          :array [1..NumMaxPilha] of integer; {indice do ultimo
elemento de um determinado bloco}
    Diferenca1      :real;
    Diferenca2      :real;
    Diferenca       :real;
    Jota            :integer;
begin
    for I:=1 to NumOrdens do
        HorDeseIni[I]:=Dados[X.MatrizPrior[I].Indice].Data-
        Dados[X.MatrizPrior[I].Indice].TProces;
    NumBlocos:=1;
    P[1].NumAtivos:=0; {CriarPilha(P[1]);}
    Primeiro[1]:=1;
    Ultimo[1]:=1;
    if (HorDeseIni[1] < 0) then
        begin
            SomaPenAdi[1]:=0;
            SomaPenAtr[1]:=Dados[X.MatrizPrior[1].Indice].PenAtr;
            HorEfetIni[1]:=0;
        end
    else
        begin
            SomaPenAdi[1]:=Dados[X.MatrizPrior[1].Indice].PenAdi;
            SomaPenAtr[1]:=0;
            HorEfetIni[1]:=HorDeseIni[1];
        end;
    for I:=1 to X.NumAtivos-1 do
        if (HorEfetIni[I]+Dados[X.MatrizPrior[I].Indice].TProces <
        HorDeseIni[I+1]) then
            begin
                NumBlocos:=NumBlocos+1;
                P[NumBlocos].NumAtivos:=0; {CriarPilha(P[NumBlocos]);}
                Primeiro[NumBlocos]:=I+1;
                Ultimo[NumBlocos]:=I+1;
                SomaPenAdi[NumBlocos]:=Dados[X.MatrizPrior[I+1].Indice].PenAdi;
                SomaPenAtr[NumBlocos]:=0;
                HorEfetIni[I+1]:=HorDeseIni[I+1];
            end
        else if (HorEfetIni[I]+Dados[X.MatrizPrior[I].Indice].TProces =
        HorDeseIni[I+1]) then
            begin
                Ultimo[NumBlocos]:=I+1;
                SomaPenAdi[NumBlocos]:=SomaPenAdi[NumBlocos]+
                Dados[X.MatrizPrior[I+1].Indice].PenAdi;
                HorEfetIni[I+1]:=HorEfetIni[I]+Dados[X.MatrizPrior[I].Indice].TProces;
            end
        else if (HorEfetIni[I]+Dados[X.MatrizPrior[I].Indice].TProces >
        HorDeseIni[I+1]) then
            begin
                Ultimo[NumBlocos]:=I+1;
                SomaPenAtr[NumBlocos]:=SomaPenAtr[NumBlocos]+

```

```

    Dados[X.MatrizPrior[I+1].Indice].PenAtr;
    HorEfetIni[I+1]:=HorEfetIni[I]+Dados[X.MatrizPrior[I].Indice].TProces;
    InseRe(I+1,HorEfetIni[I+1]-HorDeseIni[I+1],P[NumBlocos]);
    while (SomaPenAdi[NumBlocos] <= SomaPenAtr[NumBlocos]) and
      (HorEfetIni[Primeiro[NumBlocos]] > 0) do
    begin
      Diferenca1:=EncontraMin(P[NumBlocos]);
      if (NumBlocos = 1) then
        Diferenca2:=HorEfetIni[1]
      else
        Diferenca2:=HorEfetIni[Primeiro[NumBlocos]]-
          HorEfetIni[Ultimo[NumBlocos-1]]-
          Dados[X.MatrizPrior[Ultimo[NumBlocos-1]].Indice].TProces;
      Diferenca:=MenorReal(Diferenca1,Diferenca2);
      AdicionaParaTodosCritOrd(-Diferenca,P[NumBlocos]);
      HorEfetIni[Primeiro[NumBlocos]]:=HorEfetIni[Primeiro[NumBlocos]]-
        Diferenca;
      HorEfetIni[Ultimo[NumBlocos]]:=HorEfetIni[Ultimo[NumBlocos]]-
        Diferenca;
      if (P[NumBlocos].NumAtivos > 0) then
      begin
        for J:=1 to P[NumBlocos].NumAtivos do
          if (P[NumBlocos].MatrizPrior[J].CritOrd*5 = 0) then
          begin
            SomaPenAtr[NumBlocos]:=SomaPenAtr[NumBlocos]-
              Dados[X.MatrizPrior[J].Indice].PenAtr;
            SomaPenAdi[NumBlocos]:=SomaPenAdi[NumBlocos]+
              Dados[X.MatrizPrior[J].Indice].PenAdi;
          end;
          for J:=1 to P[NumBlocos].NumAtivos do
            if (P[NumBlocos].MatrizPrior[J].CritOrd*5 = 0) then
            for Jota:=J to P[NumBlocos].NumAtivos-1 do
            begin
              P[NumBlocos].MatrizPrior[Jota]:=P[NumBlocos].MatrizPrior[Jota+1];
              P[NumBlocos].NumAtivos:=P[NumBlocos].NumAtivos-1;
            end;
          end;
        if (HorEfetIni[Primeiro[NumBlocos]] = HorEfetIni[Ultimo[NumBlocos-1]]+
          Dados[X.MatrizPrior[Ultimo[NumBlocos-1]].Indice].TProces) and
          (NumBlocos > 1) then
        begin
          Une(P[NumBlocos-1],P[NumBlocos]);
          Ultimo[NumBlocos-1]:=Ultimo[NumBlocos];
          SomaPenAdi[NumBlocos-1]:=SomaPenAdi[NumBlocos-1]+
            SomaPenAdi[NumBlocos];
          SomaPenAtr[NumBlocos-1]:=SomaPenAtr[NumBlocos-1]+
            SomaPenAtr[NumBlocos];
          NumBlocos:=NumBlocos-1;
        end;
      end;
    end;
  end;
  for I:=1 to NumOrdens do
    ConjOci.MatrizPrior[I].CritOrd:=0;
    ConjOci.MatrizPrior[1].CritOrd:=HorEfetIni[Primeiro[1]];
    for I:=2 to NumBlocos do
      ConjOci.MatrizPrior[Primeiro[I]].CritOrd:=HorEfetIni[Primeiro[I]]-
        HorEfetIni[Ultimo[I-1]]-
        Dados[X.MatrizPrior[Ultimo[I-1]].Indice].TProces;
    end;
end;

```

ARQUIVO PINSPI.INC

```

procedure InseRe(X:integer;Y:real;var Z:ConjOrdenado);
  {procedimento que insere o item X com chave Y na pilha Z}
  begin
    Z.NumAtivos:=Z.NumAtivos+1;
    Z.MatrizPrior[Z.NumAtivos].Indice:=X;
    Z.MatrizPrior[Z.NumAtivos].CritOrd:=Y;
  end;

```

ARQUIVO PLEINST.INC

```

procedure LeInstancia;
  {procedimento que le a instancia a ser ordenada}
  var
    NumOrdensLido :integer; {numero de ordens lidas efetivamente no
      arquivo que esta sendo lido}

```



```

FaAtrNominal      :real; {fator de atraso nominal}
AmDisDaNomi       :real; {amplitude da distribuicao de data de entrega}
I                 :integer; {indice do loop}
Instancia         :text;
begin
{abrir arquivo de instancias para leitura apenas, verificando se esta
 tudo certo}
assign(Instancia, NomeInstancia);
reset(Instancia);
readln(Instancia, NomeInstancia);
readln(Instancia); {le uma linha de informacao do arquivo}
readln(Instancia, NumOrdens, FaAtrNominal, AmDisDaNomi);
readln(Instancia); {le uma linha de informacao do arquivo}
if (NumOrdens > NumOrdensMaximo) then
begin
writeln('Aten#Eo! O numero de ordens apontado no cabecalho da instancia');
writeln('e maior do que o numero maximo permitido. O programa foi in-');
writeln('terrompido. Verificar o erro antes de continuar!');
writeln;
halt;
end;
{ler os dados da instancia}
NumOrdensLido:=0;
for I:=1 to NumOrdens do
{$I-} {desliga a verificacao de erros}
begin
readln(Instancia,
  Dados[I].Ordem, Dados[I].Chegada, Dados[I].TProces,
  Dados[I].Data, Dados[I].PenAdi, Dados[I].PenAtr);
if (IOResult <> 0) then {caso haja algum erro de leitura}
begin
writeln('Aconteceu algum erro na leitura do arquivo ', NomeInstancia);
writeln('Verificar antes de continuar!');
writeln;
exit;
end
else {caso nao haja erro de leitura}
begin
NumOrdensLido:=NumOrdensLido+1;
end;
end;
{$I+} {religa a verificacao de erros}
if (NumOrdensLido < NumOrdens) then
begin
writeln('O numero de ordens lido no arquivo , menor do que o especifi-');
writeln('cado em seu cabecalho. O numero de ordens sera atualizado con-');
writeln('forme o numero de ordens lido! NumOrdens<--', NumOrdensLido);
NumOrdens:=NumOrdensLido;
end;
close(Instancia);
end; {le instancia}

```

ARQUIVO POCDCO.INC

```

procedure OrdenaConjDeConjOrd(var Conjunto :ConjDeConjOrd);
{ordena o conjunto de conjuntos ordenados, de acordo com menor funcao
 objetivo no inicio da matriz}
var
  I, J          :integer;
begin
if (Conjunto.NumConjAtivos > 1) then
  for I:=1 to Conjunto.NumConjAtivos-1 do
    for J:=I+1 to Conjunto.NumConjAtivos do
      if (Conjunto.FuncObjetivo[I] > Conjunto.FuncObjetivo[J]) then
        begin
          SwapConjunto(Conjunto.MatrizConj[I], Conjunto.MatrizConj[J]);
          SwapReal(Conjunto.FuncObjetivo[I], Conjunto.FuncObjetivo[J]);
        end;
    end;
end;

```

ARQUIVO POEETOCI.INC

```

procedure OrdenaExpET0ci;
{determinacao da sequencia de acordo com a funcao de prioridade E/T}
var
  C              :array[1..NumOrdensMaximo] of real; {C=hor. de termino}
  ConjOrdensDisp :ConjOrdenado; {conjunto das ordens disponiveis para

```

```

    serem sequenciadas}
Folga          :array[1..NumOrdensMaximo] of real;
I,J            :integer; {indice de loop}
Pmed           :real; {tempo de processo medio das ordens que estao
no conjunto de ordens disponiveis para serem sequenciadas}
PrioridadeMaxima :ParClassificacao;
T              :real; {tempo ou horario}
TerminoPrioMax  :real; {horario de termino da ordem de prioridade maxima}
begin
InicializaConjOrdVazio(SeqParFixa);
InicializaConjOrdVazio(ConjOci);
for I:=1 to NumOrdens do
  ConjOci.MatrizPrior[I].CritOrd:=0;
InicializaConjOrdCheio(ConjOrdensDisp);
T:=0;
ConjOci.MatrizPrior[1].CritOrd:=T;
ConjOci.NumAtivos:=1;
Pmed:=TProcesMedio(ConjOrdensDisp);
PrioridadeMaxima.CritOrd:=-maxint;
for I:=1 to ConjOrdensDisp.NumAtivos do
  begin
    C[I]:=T+Dados[ConjOrdensDisp.MatrizPrior[I].Indice].TProces;
    Folga[I]:=Dados[ConjOrdensDisp.MatrizPrior[I].Indice].Data-T-
      Dados[ConjOrdensDisp.MatrizPrior[I].Indice].TProces;
    ConjOrdensDisp.MatrizPrior[I].CritOrd:=
      PrioridadeExpET(Folga[I],Pmed,ConjOrdensDisp.MatrizPrior[I].Indice);
    if (PrioridadeMaxima.CritOrd < ConjOrdensDisp.MatrizPrior[I].CritOrd) then
      begin
        PrioridadeMaxima.Indice:=ConjOrdensDisp.MatrizPrior[I].Indice;
        PrioridadeMaxima.CritOrd:=ConjOrdensDisp.MatrizPrior[I].CritOrd;
        TerminoPrioMax:=C[I];
      end;
    end;
  if (PrioridadeMaxima.CritOrd < 0) then
    begin
      T:=OciosidadeExpET(ConjOrdensDisp,Pmed);
      ConjOci.MatrizPrior[1].CritOrd:=T;
      for I:=1 to ConjOrdensDisp.NumAtivos do
        begin
          C[I]:=T+Dados[ConjOrdensDisp.MatrizPrior[I].Indice].TProces;
          Folga[I]:=Dados[ConjOrdensDisp.MatrizPrior[I].Indice].Data-T-
            Dados[ConjOrdensDisp.MatrizPrior[I].Indice].TProces;
          ConjOrdensDisp.MatrizPrior[I].CritOrd:=
            PrioridadeExpET(Folga[I],Pmed,ConjOrdensDisp.MatrizPrior[I].Indice);
          if (PrioridadeMaxima.CritOrd < ConjOrdensDisp.MatrizPrior[I].CritOrd) then
            begin
              PrioridadeMaxima.Indice:=ConjOrdensDisp.MatrizPrior[I].Indice;
              PrioridadeMaxima.CritOrd:=ConjOrdensDisp.MatrizPrior[I].CritOrd;
              TerminoPrioMax:=C[I];
            end;
          end;
        end;
      T:=TerminoPrioMax;
      InsereNoFimConjOrd(PrioridadeMaxima,SeqParFixa);
      RetiraDeConjOrd(PrioridadeMaxima,ConjOrdensDisp);
      for J:=2 to NumOrdens do
        begin
          Pmed:=TProcesMedio(ConjOrdensDisp);
          PrioridadeMaxima.CritOrd:=-maxint;
          for I:=1 to ConjOrdensDisp.NumAtivos do
            begin
              C[I]:=T+Dados[ConjOrdensDisp.MatrizPrior[I].Indice].TProces;
              Folga[I]:=Dados[ConjOrdensDisp.MatrizPrior[I].Indice].Data-T-
                Dados[ConjOrdensDisp.MatrizPrior[I].Indice].TProces;
              ConjOrdensDisp.MatrizPrior[I].CritOrd:=
                PrioridadeExpET(Folga[I],Pmed,ConjOrdensDisp.MatrizPrior[I].Indice);
              if (PrioridadeMaxima.CritOrd < ConjOrdensDisp.MatrizPrior[I].CritOrd) then
                begin
                  PrioridadeMaxima.Indice:=ConjOrdensDisp.MatrizPrior[I].Indice;
                  PrioridadeMaxima.CritOrd:=ConjOrdensDisp.MatrizPrior[I].CritOrd;
                  TerminoPrioMax:=C[I];
                end;
              end;
            end;
          if (PrioridadeMaxima.CritOrd < 0) then
            begin
              PrioridadeMaxima.CritOrd:=-maxint;
              ConjOci.MatrizPrior[J].CritOrd:=
                OciosidadeExpET(ConjOrdensDisp,Pmed)-T;

```

```

T:=OciosidadeExpET(ConjOrdensDisp,Pmed);
for I:=1 to ConjOrdensDisp.NumAtivos do
begin
C[I]:=T+Dados[ConjOrdensDisp.MatrizPrior[I].Indice].TProces;
Folga[I]:=Dados[ConjOrdensDisp.MatrizPrior[I].Indice].Data-T-
Dados[ConjOrdensDisp.MatrizPrior[I].Indice].TProces;
ConjOrdensDisp.MatrizPrior[I].CritOrd:=
PrioridadeExpET(Folga[I],Pmed,ConjOrdensDisp.MatrizPrior[I].Indice);
if (PrioridadeMaxima.CritOrd < ConjOrdensDisp.MatrizPrior[I].CritOrd) then
begin
PrioridadeMaxima.Indice:=ConjOrdensDisp.MatrizPrior[I].Indice;
PrioridadeMaxima.CritOrd:=ConjOrdensDisp.MatrizPrior[I].CritOrd;
TerminoPrioMax:=C[I];
end;
end;
T:=TerminoPrioMax;
ConjOci.NumAtivos:=J;
InsereNoFimConjOrd(PrioridadeMaxima,SeqParFixa);
RetiraDeConjOrd(PrioridadeMaxima,ConjOrdensDisp);
end;
end;

```

ARQUIVO POEXPET.INC

procedure OrdenaExpET;

```

{determinacao da sequencia de acordo com a funcao de prioridade E/T}
var
C
:array[1..NumOrdensMaximo] of real; {C=hor. de termino}
ConjOrdensDisp :ConjOrdenado; {conjunto das ordens disponiveis para
serem sequenciadas}
Folga
:array[1..NumOrdensMaximo] of real;
I,J
:integer; {indice de loop}
Pmed
:real; {tempo de processo medio das ordens que estao
no conjunto de ordens disponiveis para serem sequenciadas}
PrioridadeMaxima :ParClassificacao;
T
:real; {tempo ou horario}
TerminoPrioMax
:real; {horario de termino da ordem de prioridade maxima}
begin
InicializaConjOrdVazio(SeqParFixa);
InicializaConjOrdCheio(ConjOrdensDisp);
T:=0;
Pmed:=TProcesMedio(ConjOrdensDisp);
PrioridadeMaxima.CritOrd:=-maxint;
for I:=1 to ConjOrdensDisp.NumAtivos do
begin
C[I]:=T+Dados[ConjOrdensDisp.MatrizPrior[I].Indice].Chegada+
Dados[ConjOrdensDisp.MatrizPrior[I].Indice].TProces;
Folga[I]:=Dados[ConjOrdensDisp.MatrizPrior[I].Indice].Data-T-
Dados[ConjOrdensDisp.MatrizPrior[I].Indice].TProces;
{calcula das prioridades do conjunto de ordens disponiveis}
ConjOrdensDisp.MatrizPrior[I].CritOrd:=
PrioridadeExpET(Folga[I],Pmed,ConjOrdensDisp.MatrizPrior[I].Indice);
if (PrioridadeMaxima.CritOrd < ConjOrdensDisp.MatrizPrior[I].CritOrd) then
begin
PrioridadeMaxima.Indice:=ConjOrdensDisp.MatrizPrior[I].Indice;
PrioridadeMaxima.CritOrd:=ConjOrdensDisp.MatrizPrior[I].CritOrd;
TerminoPrioMax:=C[I];
end;
end;
T:=TerminoPrioMax;
{inserir a prioridade maxima na sequencia parcial fixa e retira-la do
conjunto de ordens disponiveis para sequenciacao}
InsereNoFimConjOrd(PrioridadeMaxima,SeqParFixa);
RetiraDeConjOrd(PrioridadeMaxima,ConjOrdensDisp);
for J:=2 to NumOrdens do
begin
Pmed:=TProcesMedio(ConjOrdensDisp);
PrioridadeMaxima.CritOrd:=-maxint;
for I:=1 to ConjOrdensDisp.NumAtivos do
begin
C[I]:=T+Dados[ConjOrdensDisp.MatrizPrior[I].Indice].Chegada+
Dados[ConjOrdensDisp.MatrizPrior[I].Indice].TProces;
Folga[I]:=Dados[ConjOrdensDisp.MatrizPrior[I].Indice].Data-T-
Dados[ConjOrdensDisp.MatrizPrior[I].Indice].TProces;
{calcula das prioridades do conjunto de ordens disponiveis}
ConjOrdensDisp.MatrizPrior[I].CritOrd:=

```

```

        PrioridadeExpET(Folga[I],Pmed,ConjOrdensDisp.MatrizPrior[I].Indice);
    if (PrioridadeMaxima.CritOrd < ConjOrdensDisp.MatrizPrior[I].CritOrd) then
    begin
        PrioridadeMaxima.Indice:=ConjOrdensDisp.MatrizPrior[I].Indice;
        PrioridadeMaxima.CritOrd:=ConjOrdensDisp.MatrizPrior[I].CritOrd;
        TerminaPrioMax:=C[I];
    end;
end;
T:=TerminoPrioMax;
{inserir a prioridade maxima na sequencia parcial fixa e retira-la do
conjunto de ordens disponiveis para sequenciacao}
InserirNoFimConjOrd(PrioridadeMaxima,SeqParFixa);
RetiraDeConjOrd(PrioridadeMaxima,ConjOrdensDisp);
end;
end; {OrdenaExpET}

```

ARQUIVO PORDEDD.INC

```

procedure OrdenaEdd(var Dados:TAEntrada);
{procedimento que sequencia de acordo com EDD(menor data de entrega)}
var
    I,J :integer; {indices para loops}
begin
    for I:=1 to NumOrdens-1 do
        for J:=I+1 to NumOrdens do
            if Dados[I].Data>Dados[J].Data then
                SwapLinha(Dados[I],Dados[J]);
        end;
    end;

```

ARQUIVO PORDPPRI.INC

```

procedure OrdenaPorPrior(var Conjunto :ConjOrdenado);
{procedimento que sequencia um conjunto de acordo com a prioridade}
var
    I,J :integer; {indices para loops}
begin
    for I:=1 to Conjunto.NumAtivos-1 do
        for J:=I+1 to Conjunto.NumAtivos do
            if (Conjunto.MatrizPrior[I].CritOrd <
                Conjunto.MatrizPrior[J].CritOrd) then
                SwapParClassif(Conjunto.MatrizPrior[I],Conjunto.MatrizPrior[J]);
        end;
    end;

```

ARQUIVO PRDCO.INC

```

procedure RetiraDeConjOrd(Elemento :ParClassificacao;
var Conjunto :ConjOrdenado);
{procedimento que retira um elemento de um conjunto ordenado}
var
    I :integer;
    PosRetirada :integer;
begin
    PosRetirada:=0;
    for I:=1 to Conjunto.NumAtivos do
        begin
            if (Conjunto.MatrizPrior[I].Indice = Elemento.Indice) then
                begin
                    PosRetirada:=I;
                end;
            end;
    end;
    if (PosRetirada = 0) then {caso o elem. a ser retirado nao seja encontrado}
        begin
            writeln('Nao foi encontrado o elemento a ser retirado do conjunto');
            writeln('ordenado. Aconteceu algum erro no programa. Verificar antes');
            writeln('de continuar!');
            writeln;
            halt;
        end
    else {caso o elemento a ser retirado seja encontrado}
        begin
            for I:=PosRetirada to Conjunto.NumAtivos-1 do
                Conjunto.MatrizPrior[I]:=Conjunto.MatrizPrior[I+1];
            Conjunto.NumAtivos:=Conjunto.NumAtivos-1;
        end;
    end;
end;

```

ARQUIVO PSCEC.INC

```

procedure SubstConjEmConj(Conjunto:ConjOrdenado;
                           var ConjDeConj:ConjDeConjOrd);
  var
    I          :integer;
    PosInsercao:integer;
  begin
    for I:=ConjDeConj.NumConjAtivos downto 1 do
      if (Conjunto.NumAtivos > ConjDeConj.MatrizConj[I].NumAtivos) then
        PosInsercao:=I;
    ConjDeConj.MatrizConj[PosInsercao]:=Conjunto;
  end;

```

ARQUIVO PSWPCONJ.INC

```

procedure SwapConjunto(var X,Y: ConjOrdenado);
  var
    Lixo      :ConjOrdenado;
  begin
    Lixo:=X;
    X:=Y;
    Y:=Lixo;
  end;

```

ARQUIVO PSWPLINH.INC

```

procedure SwapLinha(var X,Y:TEntrada);
  {procedimento que inverte uma linha}
  var
    Lixo      :TEntrada; {variavel para troca de posicoes}
  begin
    Lixo:=X;
    X:=Y;
    Y:=Lixo;
  end;

```

ARQUIVO PSWPPC.INC

```

procedure SwapParClassif(var X,Y:ParClassificacao);
  {procedimento que inverte uma linha de um par de classificacao}
  var
    Lixo      :ParClassificacao; {variavel para troca de posi#ões}
  begin
    Lixo:=X;
    X:=Y;
    Y:=Lixo;
  end;

```

ARQUIVO PSWPREAL.INC

```

procedure SwapReal(var X,Y:real); {procedimento que inverte dois reais}
  var
    Lixo      :real; {variavel para troca de posi#ões}
  begin
    Lixo:=X;
    X:=Y;
    Y:=Lixo;
  end;

```

ARQUIVO PUCO.INC

```

procedure UneConjOrd(Conjunto1, Conjunto2 :ConjOrdenado;
                     var ConjuntoUniao :ConjOrdenado);
  {procedimento que une dois conjuntos ordenados, com o 1 na frente do 2}
  var
    I          :integer;
  begin
    ConjuntoUniao:=Conjunto1;
    ConjuntoUniao.NumAtivos:=Conjunto1.NumAtivos+Conjunto2.NumAtivos;
    for I:=1 to Conjunto2.NumAtivos do
      ConjuntoUniao.MatrizPrior[I+Conjunto1.NumAtivos]:=
        Conjunto2.MatrizPrior[I];
  end;

```

ARQUIVO PUNEPI.INC

```

procedure Une(var X,Y:ConjOrdenado);
{procedimento que une as pilhas X e Y de itens disjuntos em uma nova pilha
 que transforma-se em X. A pilha Y torna-se vazia}
var
  I      :integer;
begin
  for I:=1 to Y.NumAtivos do
    X.MatrizPrior[I+X.NumAtivos]:=Y.MatrizPrior[I];
  X.NumAtivos:=X.NumAtivos+Y.NumAtivos;
  Y.NumAtivos:=0;
end;

```

ARQUIVO TCONJUNT.INC

```

{arquivo do tipo de conjuntos que serao utilizados nos problemas de
 programacao e sequenciacao}
ParClassificacao = record
  Indice  :integer; {indice associado a matriz de da-
 dos iniciais}
  CritOrd :real; {criterio de ordenacao utilizado}
end;

ConjOrdenado = record {tipo de um conjunto ordenado}
  NumAtivos  :integer; {numero de ordens ativas no
 conjunto, isto e, a cardinalidade do conjunto}
  PosCorr    :integer; {posicao corrente que esta em
 analise}
  TempoCorr  :real; {horario corrente}
  MatrizPrior :array[1..NumOrdensMaximo] of
    ParClassificacao; {matriz que possui a prioridade
 e sua posicao associada}
end;

ConjDeConjOrd = record {tipo de um conjunto de conjuntos ordenados}
  NumConjAtivos :integer;
  MatrizConj    :array[1..LargMaxDoFiltro]
    of ConjOrdenado;
  FuncObjetivo  :array[1..LargMaxDoFiltro] of real;
end;

```

ARQUIVO TENTRADA.INC

```

{arquivo do tipo de entrada, contendo as variaveis de instancias
 de problemas de programacao e sequenciacao}
type
  TEntrada = record
    Ordem :integer; {tipo da entrada}
    Chegada:integer; {ordem de producao}
    TProces:integer; {horario de chegada}
    Data :integer; {tempo de processamento}
    PenAdi :integer; {data de entrega solicitada}
    PenAtr :integer; {penalidade de adiamento}
    {penalidade de atraso}
  end;

  TAEntrada=array[1..NumOrdensMaximo] of TEntrada; {tipo do arquivo de
 entrada}

```

PROGRAMA ALEATORI.PAS

```

program Aleatori;
{programa que gera instancias de problemas de programacao contendo
 numeros aleatorios}
const
  NumOrdens      =100;    {numero de ordens}
  Espaco         =#32;    {separador de colunas}
  ChegadaMax     =0;      {horario maximo de chegada}
  PenAtrMax      =100;    {penalidade de atraso maxima}
  PenAdiMax      =100;    {penalidade de adiamento maxima}
  TempoMax       =100;    {tempo de processamento maximo}
  NomeUltimo     ='C:\PASCAL\INSTANCI\ULTIMO.TXT'; {nome do ultimo arquivo
 gerado e do ultimo aceito}
  FaAtrNominal   =0.9;    {fator de atraso nominal}
  AmplDistrData  =0.9;    {amplitude da distribuicao de data de entrega}

var
  I              :integer;    {indice do loop}
  NumAleatorio   :real;       {numero aleatorio}
  Instancia      :text;
  Ultimo         :text;
  NomeInstancia  :string[64];
  UltimoAceito   :integer;

```

```

    UltimoGerado      :integer;
    DataMax           :integer;      {data de entrega maxima}
{$I C:\PASCAL\DISSERTA\FMININT.INC}
{$I C:\PASCAL\DISSERTA\FREPLIC.INC}
{$I C:\PASCAL\DISSERTA\FSTRZERO.INC}
{comeco do programa principal}
begin
{le UltimoAceito e UltimoGerado}
assign(Ultimo, NomeUltimo); {associa o nome Ultimo com o arquivo Ultimo}
reset(Ultimo);
readln(Ultimo, UltimoAceito, UltimoGerado);
close(Ultimo);
{cria nome do proximo arquivo de instancia a ser gerado}
UltimoGerado:=UltimoAceito+1;
NomeInstancia:='C:\PASCAL\INSTANCI\INST'+StrZero(UltimoGerado,4)+''.TXT';
{abrir arquivo de instancias}
assign(Instancia, NomeInstancia);
rewrite(Instancia);
{criacao do arquivo de instancias}
randomize;
writeln(Instancia, NomeInstancia);
writeln(Instancia, '(Num. de ordens) (fator de atraso nominal) (amplitude)');
writeln(Instancia, NumOrdens, FaAtrNominal:5:2, AmplDistrData:5:2);
writeln(Instancia, 'Ordem r    p    d    h    w');
for I:=1 to NumOrdens do
begin
    write(Instancia, I:3);
    write(Instancia, Espaco);
    write(Instancia, ChegadaMax:3);
    write(Instancia, Espaco);
    NumAleatorio:=random(TempoMax-1)+1; {indica que o tempo proc min , 1}
    write(Instancia, NumAleatorio:3:0);
    write(Instancia, Espaco);
    NumAleatorio:=random(round(NumOrdens*((TempoMax+1)/2)*AmplDistrData))+
        NumOrdens*((TempoMax+1)/2)*(1-FaAtrNominal)-
        NumOrdens*((TempoMax+1)/2)*AmplDistrData/2;
    write(Instancia, NumAleatorio:4:0);
    write(Instancia, Espaco);
    NumAleatorio:=random(PenAdiMax-1)+1;
    write(Instancia, NumAleatorio:3:0);
    write(Instancia, Espaco);
    NumAleatorio:=random(PenAtrMax-1)+1;
    writeln(Instancia, NumAleatorio:3:0);
end;
close(Instancia);
{reabre arquivo Ultimo para escrita}
assign(Ultimo, NomeUltimo);
rewrite(Ultimo);
writeln(Ultimo, UltimoAceito, Espaco, UltimoGerado);
close(Ultimo);
end.

```

PROGRAMA BSF.PAS

```

{$N+} {desliga a diretiva padrao de execucao de ponto flutuante. Utiliza a
        diretiva do processador aritmetico. WOOD, 1987, p.207}
program BSFiltro;
{programa que sequencia um conjunto de ordens de acordo com o Beam Search
    Filtrado. A funcao prioridade utilizada no filtro e a Exp-E/T (atraso/
    adiantamento). Calcula-se o valor da funcao objetivo. Faz esse procedi-
    mento para o conjunto de instancias a serem testadas. Nao permite a inser-
    cao de ociosidade entre as ordens}
uses dos; {utilizado no calculo do tempo (no gettime)}
const
    Espaco           =#32; {separador de colunas}
    K                =5;
    LargDaBusca      =3;
    LargDoFiltro     =12;
    LargMaxDoFiltro  =25;
    {largura maxima do filtro.E utilizado no type de conjunto de conj ord}
    NumOrdensMaximo  =120;
    NomeSaida        = 'C:\PASCAL\DISSERTA\BSFRESUL.TXT'; {TES1.TXT};
    SequenciaDeSaida='C:\PASCAL\DISSERTA\BSFSEQU.TXT'; {TES2.TXT};
{$I C:\PASCAL\DISSERTA\TENTRADA.INC} {acresc. o arq. de type da entrada}
{$I C:\PASCAL\DISSERTA\TCONJUNT.INC} {acresc. o arq. de type de conjuntos}
var {define variaveis globais}
    ConjSeqRetida    :ConjDeConjOrd;

```

```

Dados          :TAEntrada; {matriz para se fazer os calculos sem mexer
na original}
DadosSeqInicial:TAEntrada;
ValFuncObjetivo:real; {funcao objetivo final, com a sequencia completa}
HorFin         :longint;   {horario final}
HorIni         :longint;   {horario inicial}
HorTer         :real;      {tempo ou horario de termino}
ILOOP,JLOOP    :integer;   {indice do loop}
NomeInstancia  :string[64];
NumOrdens      :integer;   {num. de ordens lido no arquivo que sera aberto}
NumAnalizados  :integer;
Saida          :text;
SeqEmAnalise   :ConjOrdenado;
SeqParFixa     :ConjOrdenado;
SeqSaida       :text; {sequencia de saida, de acordo com ExpET}
Tempo          :longint;   {tempo de solucao}

{inclusao de funcoes}
{$I C:\PASCAL\DISSERTA\FHORARIO.INC} {funcao horario-le o horario corrente}
{$I C:\PASCAL\DISSERTA\FMAREAL.INC}  {funcao maior real}
{$I C:\PASCAL\DISSERTA\FMEREAL.INC}  {funcao menor real}
{$I C:\PASCAL\DISSERTA\FMININT.INC}  {funcao minimo inteiro}
{$I C:\PASCAL\DISSERTA\FMAXINT.INC}  {funcao maximo inteiro}
{$I C:\PASCAL\DISSERTA\FPREXPET.INC} {funcao prioridade exp ET}
{$I C:\PASCAL\DISSERTA\FREPLIC.INC}  {funcao replicate}
{$I C:\PASCAL\DISSERTA\FSTRZERO.INC} {funcao string zero}
{$I C:\PASCAL\DISSERTA\FTPROMED.INC} {funcao tempo de processo medio}

{inclusao de procedimentos}
{$I C:\PASCAL\DISSERTA\PSWPCONJ.INC} {proced. swap dois conjuntos}
{$I C:\PASCAL\DISSERTA\PSWPLINH.INC} {proced. swap linha da instancia}
{$I C:\PASCAL\DISSERTA\PSWPREAL.INC} {proced. swap dois numeros reais}
{$I C:\PASCAL\DISSERTA\PCENPP.INC}   {proced. coloca elem. na primeira pos.}
{$I C:\PASCAL\DISSERTA\PCFOBDICO.INC} {proced. Calc. funcao obj. de conj. ordenado}
{$I C:\PASCAL\DISSERTA\PICECO.INC}   {proced. insere conj. em conj. ordenado}
{$I C:\PASCAL\DISSERTA\PICOCHEI.INC} {proced. inicializa conj. ordenado cheio}
{$I C:\PASCAL\DISSERTA\PICOVAZI.INC} {proced. inicializa conj. ordenado vazio}
{$I C:\PASCAL\DISSERTA\PIECO.INC}    {proced. insere em conj. ordenado}
{$I C:\PASCAL\DISSERTA\PINFICO.INC}  {proced. insere no fim de conj. ordenado}
{$I C:\PASCAL\DISSERTA\PLEINST.INC}  {proced. le instancia}
{$I C:\PASCAL\DISSERTA\POCDICO.INC}   {proced. ordena um conj. de conj. ordenado}
{$I C:\PASCAL\DISSERTA\PRDCO.INC}     {proced. retira de conj. ordenado}
{$I C:\PASCAL\DISSERTA\POEXPET.INC}   {proced. que ordena dados por exp ET}
{$I C:\PASCAL\DISSERTA\PSCEC.INC}     {proced. substitui conj. em conj. de conj.}
{$I C:\PASCAL\DISSERTA\PUCO.INC}      {proced. une dois conj. ordenados}
{$I C:\PASCAL\DISSERTA\PIEFSPF.INC}   {proced. insere elemento no final de seq. parcial}
fixa}

procedure BeamSearchFiltrado;
{procedimento que vai gerando as sequencias de acordo com o beam search
filtrado}
var
  I,J,H          :integer; {indices de loop}
  C              :array[1..NumOrdensMaximo] of real;
  ConjOrdensDisp :ConjOrdenado;
  ConjOrdensRetFiltro:ConjOrdenado;
  ConjSeqFiltro  :ConjDeConjOrd;
  Folga          :array[1..NumOrdensMaximo] of real;
  Nivel          :integer; {nivel da arvore de busca}
  ElemDoFiltroAnalis:integer;
  {numero de elementos pertencentes ao filtro analisados}
  Pmed           :real;
  PosInsercao    :integer;
  SeqFiltro      :ConjOrdenado;
  SeqParComplem  :ConjOrdenado;
begin
  ConjSeqRetida.NumConjAtivos:=0;
  HorTer:=0;
  InicializaConjOrdCheio(SeqFiltro);
  InicializaConjOrdVazio(ConjOrdensRetFiltro);
  InicializaConjOrdCheio(ConjOrdensDisp);
  Pmed:=TProcesMedio(ConjOrdensDisp);
  {calcula das prioridades do conjunto de ordens disponiveis}
  for I:=1 to ConjOrdensDisp.NumAtivos do
    begin
      Folga[I]:=Dados[ConjOrdensDisp.MatrizPrior[I].Indice].Data-HorTer-
        Dados[ConjOrdensDisp.MatrizPrior[I].Indice].TProces;
      ConjOrdensDisp.MatrizPrior[I].CritOrd:=
        PrioridadeExpET(Folga[I],Pmed,ConjOrdensDisp.MatrizPrior[I].Indice);
      InsereEmConjOrd(ConjOrdensDisp.MatrizPrior[I],ConjOrdensRetFiltro);
    end;

```



```

{criacao das seq. completas e calc. da funcao obj. das seq. do filtro}
ConjSeqFiltro.NumConjAtivos:=0;
ElemDoFiltroAnalisis:=MaximoInteiro(LargDoFiltro,LargDaBusca);
for I:=1 to ElemDoFiltroAnalisis do
begin
  InicializaConjOrdVazio(ConjSeqFiltro.MatrizConj[I]);
  InicializaConjOrdVazio(SeqFiltro);
  ConjSeqFiltro.NumConjAtivos:=ConjSeqFiltro.NumConjAtivos+1;
  InsereNoFimConjOrd(ConjOrdensRetFiltro.MatrizPrior[I],SeqFiltro);
  InicializaConjOrdCheio(SeqParComplem);
  RetiraDeConjOrd(ConjOrdensRetFiltro.MatrizPrior[I],SeqParComplem);
  UneConjOrd(SeqFiltro,SeqParComplem,ConjSeqFiltro.MatrizConj[I]);
  CalcFObjDeConjOrd(ConjSeqFiltro.MatrizConj[I],1);
  ConjSeqFiltro.FuncObjetivo[I]:=ValFuncObjetivo;
  ConjSeqFiltro.MatrizConj[I].TempoCorr:=HorTer;
  ConjSeqFiltro.MatrizConj[I].PosCorr:=1;
end;
OrdenaConjDeConjOrd(ConjSeqFiltro);
for I:=1 to LargDaBusca do
begin
  ConjSeqRetida.MatrizConj[I]:=ConjSeqFiltro.MatrizConj[I];
  ConjSeqRetida.FuncObjetivo[I]:=ConjSeqFiltro.FuncObjetivo[I];
end;
ConjSeqRetida.NumConjAtivos:=LargDaBusca;
{inicio do loop principal}
for Nivel:=2 to NumOrdens-1 do
begin
  {criacao das seq. completas e calc. da funcao obj. das seq. do filtro}
  ConjSeqFiltro.NumConjAtivos:=0;
  for I:=1 to LargDaBusca do
  begin
    InicializaConjOrdCheio(ConjOrdensDisp);
    {retirar as ordens que ja foram sequenciadas}
    for J:=1 to ConjSeqRetida.MatrizConj[I].PosCorr do
      RetiraDeConjOrd(ConjSeqRetida.MatrizConj[I].MatrizPrior[J],
        ConjOrdensDisp);
    HorTer:=ConjSeqRetida.MatrizConj[I].TempoCorr;
    Pmed:=TProcesMedio(ConjOrdensDisp);
    InicializaConjOrdVazio(ConjOrdensRetFiltro);
    {calculo das prioridades do conjunto de ordens disponiveis}
    for J:=1 to ConjOrdensDisp.NumAtivos do
    begin
      Folga[J]:=Dados[ConjOrdensDisp.MatrizPrior[J].Indice].Data-HorTer-
        Dados[ConjOrdensDisp.MatrizPrior[J].Indice].TProces;
      ConjOrdensDisp.MatrizPrior[J].CritOrd:=
        PrioridadeExpET(Folga[J],Pmed,ConjOrdensDisp.MatrizPrior[J].Indice);
      InsereEmConjOrd(ConjOrdensDisp.MatrizPrior[J],ConjOrdensRetFiltro);
    end;
  ElemDoFiltroAnalisis:=
    MinimoInteiro(LargDoFiltro,ConjOrdensDisp.NumAtivos);
  for J:=1 to ElemDoFiltroAnalisis do
  begin
    InicializaConjOrdVazio(SeqEmAnalise);
    InsElemFimSeqParFixa(ConjSeqRetida.MatrizConj[I],
      ConjOrdensRetFiltro.MatrizPrior[J],SeqEmAnalise);
    SeqEmAnalise.PosCorr:=ConjSeqRetida.MatrizConj[I].PosCorr+1;
    CalcFObjDeConjOrd(SeqEmAnalise,SeqEmAnalise.PosCorr);
    SeqEmAnalise.TempoCorr:=HorTer;
    NumAnalizados:=MaximoInteiro(LargDoFiltro,LargDaBusca);
    NumAnalizados:=MinimoInteiro(NumAnalizados,ConjOrdensDisp.NumAtivos);
    if (ConjSeqFiltro.NumConjAtivos < NumAnalizados) then
      {se o conjunto de seq. do filtro nao estiver cheio}
      begin
        InsereConjEmConj(SeqEmAnalise,ConjSeqFiltro);
        ConjSeqFiltro.FuncObjetivo[ConjSeqFiltro.NumConjAtivos]:=
          ValFuncObjetivo;
        OrdenaConjDeConjOrd(ConjSeqFiltro);
      end
    else
      {se o conjunto de seq. do filtro estiver cheio}
      begin
        PosInsercao:=0;
        for H:=ConjSeqFiltro.NumConjAtivos downto 1 do
          if (ValFuncObjetivo < ConjSeqFiltro.FuncObjetivo[H]) then
            PosInsercao:=H;
        if (PosInsercao >= 1) and
          (PosInsercao < ConjSeqFiltro.NumConjAtivos) then
          begin

```

```

    for H:=NumAnalizados downto PosInsercao+1 do
    begin
        ConjSeqFiltro.MatrizConj[H]:=ConjSeqFiltro.MatrizConj[H-1];
        ConjSeqFiltro.FuncObjetivo[H]:=ConjSeqFiltro.FuncObjetivo[H-1];
    end;
    ConjSeqFiltro.MatrizConj[PosInsercao]:=SeqEmAnalise;
    ConjSeqFiltro.FuncObjetivo[PosInsercao]:=ValFuncObjetivo;
    end;
    if (PosInsercao = ConjSeqFiltro.NumConjAtivos) then
    begin
        ConjSeqFiltro.MatrizConj[PosInsercao]:=SeqEmAnalise;
        ConjSeqFiltro.FuncObjetivo[PosInsercao]:=ValFuncObjetivo;
    end;
    end;
    end;
    for J:=1 to LargDaBusca do
    begin
        ConjSeqRetida.FuncObjetivo[J]:=ConjSeqFiltro.FuncObjetivo[J];
        ConjSeqRetida.MatrizConj[J]:=ConjSeqFiltro.MatrizConj[J];
    end;
    end;
    end;
    {comeco do programa principal}
begin
    for ILOOP:=1 to 720 do
    begin
        writeln(ILOOP);
        NomeInstancia:='C:\PASCAL\INSTANCI\INST'+StrZero(ILOOP,4)+' .TXT';
        HorIni:=Horario;
        LeInstancia;
        OrdenaExpET;
        for JLOOP:=1 to NumOrdens do
            DadosSeqInicial[JLOOP]:=Dados[SeqParFixa.MatrizPrior[JLOOP].Indice];
        for JLOOP:=1 to NumOrdens do
            Dados[JLOOP]:=DadosSeqInicial[JLOOP];
        BeamSearchFiltrado;
        HorFin:=Horario;
        Tempo:=HorFin-HorIni;
        {abrir arquivo de saida dos resultados (tempo e funcao objetivo)}
        assign(Saida, NomeSaida);
        append(Saida);
        writeln(Saida, ILOOP:5, Espaco, HorIni:8, Espaco, HorFin:8, Espaco,
            Tempo:6, Espaco, ConjSeqRetida.FuncObjetivo[1]:18);
        close(Saida);
    end;
    end;
end.

```

PROGRAMA BSFOCI.PAS

```

{$N+} {desliga a diretiva padrao de execucao de ponto flutuante. Utiliza a
        diretiva do processador aritmetico. WOOD, 1987, p.207}
program BSFiltroOci;
    {programa que programa um conjunto de ordens de acordo com o Beam Search
    Filtrado, utilizando o procedimento de insercao de ociosidade antes de
    se analisar a funcao objetivo das sequencias completas. A funcao
    prioridade utilizada no filtro e a Exp-E/T (atraso/adiantamento). Faz
    esse procedimento para o conjunto de instancias a serem testadas.}
    uses dos; {utilizado no calculo do tempo (no gettime)}
    const
        Espaco           =#32; {separador de colunas}
        K                 =5;
        LargDaBusca       =1;
        LargDoFiltro      =1;
        LargMaxDoFiltro   =12;
        {largura maxima do filtro.E utilizado no type de conjunto de conj ord}
        NumOrdensMaximo   =105;
        NumMaxPilha       =55;
        NomeSaida         ='C:\PASCAL\DISSERTA\BSFRESUL.TXT';
        SequenciaDeSaida  ='C:\PASCAL\DISSERTA\BSFSEQU.TXT';
    {$I C:\PASCAL\DISSERTA\TENTRADA.INC} {acresc. o arq. de type da entrada}
    {$I C:\PASCAL\DISSERTA\TCONJUNT.INC} {acresc. o arq. de type de conjuntos}
    {$I C:\PASCAL\DISSERTA\TPILHA.INC}   {acresc. o arq. de type de pilhas}
    var {define variaveis globais}
        ConjOci           :ConjOrdenado; {conjunto das ociosidades}
        ConjSeqRetida     :ConjDeConjOrd;
        Dados              :TAEEntrada; {matriz para se fazer os calculos sem mexer

```

```

na original}
DadosSeqInicial:TAEntrada;
ValFuncObjetivo:real; {funcao objetivo final, com a sequencia completa}
HorFin      :longint;   {horario final}
HorIni      :longint;   {horario inicial}
HorTer      :real;      {tempo ou horario de termino}
ILOOP,JLOOP :integer;   {indice do loop}
NomeInstancia :string[64];
NumOrdens    :integer; {num. de ordens lido no arquivo que sera aberto}
NumAnalizados :integer;
NumBlocos    :integer;
Saida        :text;
SeqEmAnalise :ConjOrdenado;
SeqParFixa   :ConjOrdenado;
SeqSaida     :text; {sequencia de saida, de acordo com ExpET}
Tempo        :longint; {tempo de solucao}
P            :array[1..NumMaxPilha] of ConjOrdenado;
{inclusao de funcoes}
{$I C:\PASCAL\DISSERTA\FHORARIO.INC} {funcao horario-le o horario corrente}
{$I C:\PASCAL\DISSERTA\FMAREAL.INC}  {funcao maior real}
{$I C:\PASCAL\DISSERTA\FMEREAL.INC}  {funcao menor real}
{$I C:\PASCAL\DISSERTA\FMININT.INC}  {funcao minimo inteiro}
{$I C:\PASCAL\DISSERTA\FMAXINT.INC}  {funcao maximo inteiro}
{$I C:\PASCAL\DISSERTA\FPREXPET.INC} {funcao prioridade exp ET}
{$I C:\PASCAL\DISSERTA\FREPLIC.INC}  {funcao replicate}
{$I C:\PASCAL\DISSERTA\FSTRZERO.INC} {funcao string zero}
{$I C:\PASCAL\DISSERTA\FTPROMED.INC} {funcao tempo de processo medio}
{$I C:\PASCAL\DISSERTA\FEMINPI.INC}  {funcao encontra min de uma pilha}
{$I C:\PASCAL\DISSERTA\FAMINPI.INC}  {funcao apaga min de uma pilha}
{inclusao de procedimentos}
{$I C:\PASCAL\DISSERTA\PSWPCONJ.INC} {proced. swap dois conjuntos}
{$I C:\PASCAL\DISSERTA\PSWPLINH.INC} {proced. swap linha da instancia}
{$I C:\PASCAL\DISSERTA\PSWPREAL.INC} {proced. swap dois numeros reais}
{$I C:\PASCAL\DISSERTA\PCENPP.INC}   {proced. coloca elem. na primeira pos.}
{$I C:\PASCAL\DISSERTA\PCFOBOCI.INC} {proced. calcula funcao objetivo}
{$I C:\PASCAL\DISSERTA\PICECO.INC}   {proced. insere conj. em conj. ordenado}
{$I C:\PASCAL\DISSERTA\PICOCHEI.INC} {proced. inicializa conj. ordenado cheio}
{$I C:\PASCAL\DISSERTA\PICOVAZI.INC} {proced. inicializa conj. ordenado vazio}
{$I C:\PASCAL\DISSERTA\PIECO.INC}    {proced. insere em conj. ordenado}
{$I C:\PASCAL\DISSERTA\PINFCO.INC}   {proced. insere no fim de conj. ordenado}
{$I C:\PASCAL\DISSERTA\PLEINST.INC}  {proced. le instancia}
{$I C:\PASCAL\DISSERTA\POCDCO.INC}   {proced. ordena um conj. de conj. ordenado}
{$I C:\PASCAL\DISSERTA\PRDCO.INC}    {proced. retira de conj. ordenado}
{$I C:\PASCAL\DISSERTA\POEXPET.INC}  {proced. que ordena dados por exp ET}
{$I C:\PASCAL\DISSERTA\PSCEC.INC}    {proced. substitui conj. em conj. de conj.}
{$I C:\PASCAL\DISSERTA\PUCO.INC}     {proced. une dois conj. ordenados}
{$I C:\PASCAL\DISSERTA\PIEFSPF.INC}  {proced. insere elemento no final de seq. parcial}
fixa}
{$I C:\PASCAL\DISSERTA\PINSPI.INC}   {proced. insere elemento em pilha}
{$I C:\PASCAL\DISSERTA\PUNEPI.INC}   {proced. une duas pilhas}
{$I C:\PASCAL\DISSERTA\PAPTCOPI.INC} {proced. adiciona p/ todas chaves um valor}
{$I C:\PASCAL\DISSERTA\PINSOCI3.INC} {proced. de insercao de ociosidade}
procedure BeamSearchFiltradoOcios;
{procedimento que vai gerando as sequencias de acordo com o beam search
filtrado e inserindo as ociosidades nas sequencias completas}
var
  I,J,H      :integer; {indices de loop}
  C          :array[1..NumOrdensMaximo] of real;
  ConjOrdensDisp :ConjOrdenado;
  ConjOrdensRetFiltro:ConjOrdenado;
  ConjSeqFiltro  :ConjDeConjOrd;
  Folga         :array[1..NumOrdensMaximo] of real;
  Nivel        :integer; {nivel da arvore de busca}
  ElemDoFiltroAnalis:integer;
  {numero de elementos pertencentes ao filtro analisados}
  Pmed         :real;
  PosInsercao  :integer;
  SeqFiltro    :ConjOrdenado;
  SeqParComplem :ConjOrdenado;
begin
  ConjSeqRetida.NumConjAtivos:=0;
  HorTer:=0;
  InicializaConjOrdCheio(SeqFiltro);
  InicializaConjOrdVazio(ConjOrdensRetFiltro);
  InicializaConjOrdCheio(ConjOrdensDisp);
  Pmed:=TProcesMedio(ConjOrdensDisp);
  {calcula das prioridades do conjunto de ordens disponiveis}
  for I:=1 to ConjOrdensDisp.NumAtivos do

```

```

begin
Folga[I]:=Dados[ConjOrdensDisp.MatrizPrior[I].Indice].Data-HorTer-
  Dados[ConjOrdensDisp.MatrizPrior[I].Indice].TProces;
ConjOrdensDisp.MatrizPrior[I].CritOrd:=
  PrioridadeExpET(Folga[I],Pmed,ConjOrdensDisp.MatrizPrior[I].Indice);
InsereEmConjOrd(ConjOrdensDisp.MatrizPrior[I],ConjOrdensRetFiltro);
end;
{criacao das seq. completas e calc. da funcao obj. das seq. do filtro}
ConjSeqFiltro.NumConjAtivos:=0;
ElemDoFiltroAnalis:=MaximoInteiro(LargDoFiltro,LargDaBusca);
for I:=1 to ElemDoFiltroAnalis do
begin
  InicializaConjOrdVazio(ConjSeqFiltro.MatrizConj[I]);
  InicializaConjOrdVazio(SeqFiltro);
  ConjSeqFiltro.NumConjAtivos:=ConjSeqFiltro.NumConjAtivos+1;
  InsereNoFimConjOrd(ConjOrdensRetFiltro.MatrizPrior[I],SeqFiltro);
  InicializaConjOrdCheio(SeqParComplem);
  RetiraDeConjOrd(ConjOrdensRetFiltro.MatrizPrior[I],SeqParComplem);
  UneConjOrd(SeqFiltro,SeqParComplem,ConjSeqFiltro.MatrizConj[I]);
  InsereOciosidade(ConjSeqFiltro.MatrizConj[I]);
  CalcFObjOci(ConjSeqFiltro.MatrizConj[I],ConjOci,1);
  ConjSeqFiltro.FuncObjetivo[I]:=ValFuncObjetivo;
  ConjSeqFiltro.MatrizConj[I].TempoCorr:=HorTer;
  ConjSeqFiltro.MatrizConj[I].PosCorr:=1;
end;
OrdenaConjDeConjOrd(ConjSeqFiltro);
for I:=1 to LargDaBusca do
begin
  ConjSeqRetida.MatrizConj[I]:=ConjSeqFiltro.MatrizConj[I];
  ConjSeqRetida.FuncObjetivo[I]:=ConjSeqFiltro.FuncObjetivo[I];
end;
ConjSeqRetida.NumConjAtivos:=LargDaBusca;
{inicio do loop principal}
for Nivel:=2 to NumOrdens-1 do
begin
  {criacao das seq. completas e calc. da funcao obj. das seq. do filtro}
  ConjSeqFiltro.NumConjAtivos:=0;
  for I:=1 to LargDaBusca do
  begin
    InicializaConjOrdCheio(ConjOrdensDisp);
    {retirar as ordens que ja foram sequenciadas}
    for J:=1 to ConjSeqRetida.MatrizConj[I].PosCorr do
      RetiraDeConjOrd(ConjSeqRetida.MatrizConj[I].MatrizPrior[J],
        ConjOrdensDisp);
    HorTer:=ConjSeqRetida.MatrizConj[I].TempoCorr;
    Pmed:=TProcesMedio(ConjOrdensDisp);
    InicializaConjOrdVazio(ConjOrdensRetFiltro);
    {calcula das prioridades do conjunto de ordens disponiveis}
    for J:=1 to ConjOrdensDisp.NumAtivos do
    begin
      Folga[J]:=Dados[ConjOrdensDisp.MatrizPrior[J].Indice].Data-HorTer-
        Dados[ConjOrdensDisp.MatrizPrior[J].Indice].TProces;
      ConjOrdensDisp.MatrizPrior[J].CritOrd:=
        PrioridadeExpET(Folga[J],Pmed,ConjOrdensDisp.MatrizPrior[J].Indice);
      InsereEmConjOrd(ConjOrdensDisp.MatrizPrior[J],ConjOrdensRetFiltro);
    end;
  ElemDoFiltroAnalis:=
    MinimoInteiro(LargDoFiltro,ConjOrdensDisp.NumAtivos);
  for J:=1 to ElemDoFiltroAnalis do
  begin
    InicializaConjOrdVazio(SeqEmAnalise);
    InsElemFimSeqParFixa(ConjSeqRetida.MatrizConj[I],
      ConjOrdensRetFiltro.MatrizPrior[J],SeqEmAnalise);
    SeqEmAnalise.PosCorr:=ConjSeqRetida.MatrizConj[I].PosCorr+1;
    InsereOciosidade(SeqEmAnalise);
    CalcFObjOci(SeqEmAnalise,ConjOci,SeqEmAnalise.PosCorr);
    SeqEmAnalise.TempoCorr:=HorTer;
    NumAnalizados:=MaximoInteiro(LargDoFiltro,LargDaBusca);
    NumAnalizados:=MinimoInteiro(NumAnalizados,ConjOrdensDisp.NumAtivos);
    if (ConjSeqFiltro.NumConjAtivos < NumAnalizados) then
      {se o conjunto de seq. do filtro nao estiver cheio}
      begin
        InsereConjEmConj(SeqEmAnalise,ConjSeqFiltro);
        ConjSeqFiltro.FuncObjetivo[ConjSeqFiltro.NumConjAtivos]:=
          ValFuncObjetivo;
        OrdenaConjDeConjOrd(ConjSeqFiltro);
      end
    else

```

```

{se o conjunto de seq. do filtro estiver cheio}
begin
  PosInsercao:=0;
  for H:=ConjSeqFiltro.NumConjAtivos downto 1 do
    if (ValFuncObjetivo < ConjSeqFiltro.FuncObjetivo[H]) then
      PosInsercao:=H;
  if (PosInsercao >= 1) and
    (PosInsercao < ConjSeqFiltro.NumConjAtivos) then
    begin
      for H:=NumAnalizados downto PosInsercao+1 do
        begin
          ConjSeqFiltro.MatrizConj[H]:=ConjSeqFiltro.MatrizConj[H-1];
          ConjSeqFiltro.FuncObjetivo[H]:=ConjSeqFiltro.FuncObjetivo[H-1];
        end;
      ConjSeqFiltro.MatrizConj[PosInsercao]:=SeqEmAnalise;
      ConjSeqFiltro.FuncObjetivo[PosInsercao]:=ValFuncObjetivo;
    end;
  if (PosInsercao = ConjSeqFiltro.NumConjAtivos) then
    begin
      ConjSeqFiltro.MatrizConj[PosInsercao]:=SeqEmAnalise;
      ConjSeqFiltro.FuncObjetivo[PosInsercao]:=ValFuncObjetivo;
    end;
  end;
end;
end;
for J:=1 to LargDaBusca do
  begin
    ConjSeqRetida.FuncObjetivo[J]:=ConjSeqFiltro.FuncObjetivo[J];
    ConjSeqRetida.MatrizConj[J]:=ConjSeqFiltro.MatrizConj[J];
  end;
end;
end;
end;
begin {começo do programa principal}
for ILOOP:=1 to 720 do
  begin
    writeln(ILOOP);
    NomeInstancia:='C:\PASCAL\INSTANCI\I2.txt'; {NST'+StrZero(ILOOP,4)+' .TXT'};
    HorIni:=Horario;
    LeInstancia;
    OrdenaExpET;
    for JLOOP:=1 to NumOrdens do
      DadosSeqInicial[JLOOP]:=Dados[SeqParFixa.MatrizPrior[JLOOP].Indice];
    for JLOOP:=1 to NumOrdens do
      Dados[JLOOP]:=DadosSeqInicial[JLOOP];
    BeamSearchFiltradoOcios;
    HorFin:=Horario;
    Tempo:=HorFin-HorIni;
    {abrir arquivo de saida dos resultados (tempo e funcao objetivo)}
    assign(Saida, NomeSaida);
    append(Saida);
    writeln(Saida, ILOOP:5, Espaco, HorIni:8, Espaco, HorFin:8, Espaco,
      Tempo:6, Espaco, ConjSeqRetida.FuncObjetivo[1]:18);
    close(Saida);
  end;
end.

```

PROGRAMA BSFOCISE.PAS

```

{$N+} {desliga a diretiva padrao de execucao de ponto flutuante. Utiliza a
      diretiva do processador aritmetico. WOOD, 1987, p.207}
program BSFiltroOci;
{programa que programa um conjunto de ordens de acordo com o Beam Search
Filtrado, utilizando o procedimento de insercao de ociosidade antes de
se analisar a funcao objetivo das sequencias completas. A funcao
prioridade utilizada no filtro e a Exp-E/T (atraso/adiantamento). Faz
esse procedimento para o conjunto de instancias a serem testadas. Inicial-
mente utiliza como semente uma sequencia definida por expet-et com ociosi-
dade}
uses dos; {utilizado no calculo do tempo (no gettime)}
const
  Espaco          =#32; {separador de colunas}
  K               =5;
  LargDaBusca     =8;
  LargDoFiltro    =8;
  LargMaxDoFiltro =12;
  {largura maxima do filtro.E utilizado no type de conjunto de conj ord}
  NumOrdensMaximo =105;

```

```

    NumMaxPilha      =55;
    NomeSaida        ='C:\PASCAL\DISSERTA\BSFRESUL.TXT';
    SequenciaDeSaida='C:\PASCAL\DISSERTA\BSFSEQU.TXT';
{$I C:\PASCAL\DISSERTA\TENTRADA.INC} {acresc. o arq. de type da entrada}
{$I C:\PASCAL\DISSERTA\TCONJUNT.INC} {acresc. o arq. de type de conjuntos}
{$I C:\PASCAL\DISSERTA\TPILHA.INC}   {acresc. o arq. de type de pilhas}
var {define variaveis globais}
    ConjOci          :ConjOrdenado; {conjunto das ociosidades}
    ConjSeqRetida    :ConjDeConjOrd;
    Dados             :TAEntrada; {matriz para se fazer os calculos sem mexer
        na original}
    DadosSeqInicial:TAEntrada;
    ValFuncObjetivo:real; {funcao objetivo final, com a sequencia completa}
    HorFin            :longint; {horario final}
    HorIni            :longint; {horario inicial}
    HorTer            :real; {tempo ou horario de termino}
    ILOOP,JLOOP       :integer; {indice do loop}
    NomeInstancia     :string[64];
    NumOrdens          :integer; {num. de ordens lido no arquivo que sera aberto}
    NumAnalisados      :integer;
    NumBlocos          :integer;
    Saida              :text;
    SeqEmAnalise       :ConjOrdenado;
    SeqParFixa         :ConjOrdenado;
    SeqSaida           :text; {sequencia de saida, de acordo com ExpET}
    Tempo              :longint; {tempo de solucao}
    P                  :array[1..NumMaxPilha] of ConjOrdenado;
{inclusao de funcoes}
{$I C:\PASCAL\DISSERTA\FHORARIO.INC} {funcao horario-le o horario corrente}
{$I C:\PASCAL\DISSERTA\FMAREAL.INC}  {funcao maior real}
{$I C:\PASCAL\DISSERTA\FMEREAL.INC}  {funcao menor real}
{$I C:\PASCAL\DISSERTA\FMININT.INC}  {funcao minimo inteiro}
{$I C:\PASCAL\DISSERTA\FMAXINT.INC}  {funcao maximo inteiro}
{$I C:\PASCAL\DISSERTA\FPREXPET.INC} {funcao prioridade exp ET}
{$I C:\PASCAL\DISSERTA\FREPLIC.INC}  {funcao replicate}
{$I C:\PASCAL\DISSERTA\FSTRZERO.INC} {funcao string zero}
{$I C:\PASCAL\DISSERTA\FTPROMED.INC} {funcao tempo de processo medio}
{$I C:\PASCAL\DISSERTA\FEMINPI.INC}  {funcao encontra min de uma pilha}
{$I C:\PASCAL\DISSERTA\FAMINPI.INC}  {funcao apaga min de uma pilha}
{$I C:\PASCAL\DISSERTA\FOCIEET.INC}  {funcao EXPET com ociosidade}
{inclusao de procedimentos}
{$I C:\PASCAL\DISSERTA\PSWPCONJ.INC} {proced. swap dois conjuntos}
{$I C:\PASCAL\DISSERTA\PSWPLINH.INC} {proced. swap linha da instancia}
{$I C:\PASCAL\DISSERTA\PSWPREAL.INC} {proced. swap dois numeros reais}
{$I C:\PASCAL\DISSERTA\PCENPP.INC}   {proced. coloca elem. na primeira pos.}
{$I C:\PASCAL\DISSERTA\PCFOBOCI.INC} {proced. calcula funcao objetivo}
{$I C:\PASCAL\DISSERTA\PICECO.INC}   {proced. insere conj. em conj. ordenado}
{$I C:\PASCAL\DISSERTA\PICOCHEI.INC} {proced. inicializa conj. ordenado cheio}
{$I C:\PASCAL\DISSERTA\PICOVAZI.INC} {proced. inicializa conj. ordenado vazio}
{$I C:\PASCAL\DISSERTA\PIECO.INC}    {proced. insere em conj. ordenado}
{$I C:\PASCAL\DISSERTA\PINFECO.INC}  {proced. insere no fim de conj. ordenado}
{$I C:\PASCAL\DISSERTA\PLEINST.INC}  {proced. le instancia}
{$I C:\PASCAL\DISSERTA\POCDCO.INC}   {proced. ordena um conj. de conj. ordenado}
{$I C:\PASCAL\DISSERTA\PRDCO.INC}    {proced. retira de conj. ordenado}
{$I C:\PASCAL\DISSERTA\PSCEC.INC}    {proced. substitui conj. em conj. de conj.}
{$I C:\PASCAL\DISSERTA\PUCO.INC}     {proced. une dois conj. ordenados}
{$I C:\PASCAL\DISSERTA\PIEFSPP.INC}  {proced. insere elemento no final de seq. parcial}
fixa)
{$I C:\PASCAL\DISSERTA\PINSPI.INC}   {proced. insere elemento em pilha}
{$I C:\PASCAL\DISSERTA\PUNEPI.INC}   {proced. une duas pilhas}
{$I C:\PASCAL\DISSERTA\PAPTCOPI.INC} {proced. adiciona p/ todas chaves um valor}
{$I C:\PASCAL\DISSERTA\PINSOCI3.INC} {proced. de insercao de ociosidade}
{$I C:\PASCAL\DISSERTA\POEETOCI.INC} {proced. que ordena dados por expet com
ociosidade}
procedure BeamSearchFiltradoOcios;
{procedimento que vai gerando as sequencias de acordo com o beam search
filtrado e inserindo as ociosidades nas sequencias completas}
var
    I,J,H              :integer; {indices de loop}
    C                  :array[1..NumOrdensMaximo] of real;
    ConjOrdensDisp     :ConjOrdenado;
    ConjOrdensRetFiltro:ConjOrdenado;
    ConjSeqFiltro       :ConjDeConjOrd;
    Folga              :array[1..NumOrdensMaximo] of real;
    Nivel              :integer; {nivel da arvore de busca}
    ElemDoFiltroAnalis:integer;
    {numero de elementos pertencentes ao filtro analisados}
    Pmed               :real;

```

```

PosInsercao      :integer;
SeqFiltro        :ConjOrdenado;
SeqParComplem    :ConjOrdenado;
begin
ConjSeqRetida.NumConjAtivos:=0;
HorTer:=0;
InicializaConjOrdCheio(SeqFiltro);
InicializaConjOrdVazio(ConjOrdensRetFiltro);
InicializaConjOrdCheio(ConjOrdensDisp);
Pmed:=TProcesMedio(ConjOrdensDisp);
{calculo das prioridades do conjunto de ordens disponiveis}
for I:=1 to ConjOrdensDisp.NumAtivos do
begin
Folga[I]:=Dados[ConjOrdensDisp.MatrizPrior[I].Indice].Data-HorTer-
Dados[ConjOrdensDisp.MatrizPrior[I].Indice].TProces;
ConjOrdensDisp.MatrizPrior[I].CritOrd:=
PrioridadeExpET(Folga[I],Pmed,ConjOrdensDisp.MatrizPrior[I].Indice);
InsereEmConjOrd(ConjOrdensDisp.MatrizPrior[I],ConjOrdensRetFiltro);
end;
{criacao das seq. completas e calc. da funcao obj. das seq. do filtro}
ConjSeqFiltro.NumConjAtivos:=0;
ElemDoFiltroAnalis:=MaximoInteiro(LargDoFiltro,LargDaBusca);
for I:=1 to ElemDoFiltroAnalis do
begin
InicializaConjOrdVazio(ConjSeqFiltro.MatrizConj[I]);
InicializaConjOrdVazio(SeqFiltro);
ConjSeqFiltro.NumConjAtivos:=ConjSeqFiltro.NumConjAtivos+1;
InsereNoFimConjOrd(ConjOrdensRetFiltro.MatrizPrior[I],SeqFiltro);
InicializaConjOrdCheio(SeqParComplem);
RetiraDeConjOrd(ConjOrdensRetFiltro.MatrizPrior[I],SeqParComplem);
UnseConjOrd(SeqFiltro,SeqParComplem,ConjSeqFiltro.MatrizConj[I]);
InsereOciosidade(ConjSeqFiltro.MatrizConj[I]);
CalcFObjOci(ConjSeqFiltro.MatrizConj[I],ConjOci,1);
ConjSeqFiltro.FuncObjetivo[I]:=ValFuncObjetivo;
ConjSeqFiltro.MatrizConj[I].TempoCorr:=HorTer;
ConjSeqFiltro.MatrizConj[I].PosCorr:=1;
end;
OrdenaConjDeConjOrd(ConjSeqFiltro);
for I:=1 to LargDaBusca do
begin
ConjSeqRetida.MatrizConj[I]:=ConjSeqFiltro.MatrizConj[I];
ConjSeqRetida.FuncObjetivo[I]:=ConjSeqFiltro.FuncObjetivo[I];
end;
ConjSeqRetida.NumConjAtivos:=LargDaBusca;
{inicio do loop principal}
for Nivel:=2 to NumOrdens-1 do
begin
{criacao das seq. completas e calc. da funcao obj. das seq. do filtro}
ConjSeqFiltro.NumConjAtivos:=0;
for I:=1 to LargDaBusca do
begin
InicializaConjOrdCheio(ConjOrdensDisp);
{retirar as ordens que ja foram sequenciadas}
for J:=1 to ConjSeqRetida.MatrizConj[I].PosCorr do
RetiraDeConjOrd(ConjSeqRetida.MatrizConj[I].MatrizPrior[J],
ConjOrdensDisp);
HorTer:=ConjSeqRetida.MatrizConj[I].TempoCorr;
Pmed:=TProcesMedio(ConjOrdensDisp);
InicializaConjOrdVazio(ConjOrdensRetFiltro);
{calculo das prioridades do conjunto de ordens disponiveis}
for J:=1 to ConjOrdensDisp.NumAtivos do
begin
Folga[J]:=Dados[ConjOrdensDisp.MatrizPrior[J].Indice].Data-HorTer-
Dados[ConjOrdensDisp.MatrizPrior[J].Indice].TProces;
ConjOrdensDisp.MatrizPrior[J].CritOrd:=
PrioridadeExpET(Folga[J],Pmed,ConjOrdensDisp.MatrizPrior[J].Indice);
InsereEmConjOrd(ConjOrdensDisp.MatrizPrior[J],ConjOrdensRetFiltro);
end;
ElemDoFiltroAnalis:=
MinimoInteiro(LargDoFiltro,ConjOrdensDisp.NumAtivos);
for J:=1 to ElemDoFiltroAnalis do
begin
InicializaConjOrdVazio(SeqEmAnalise);
InsElemFimSeqParFixa(ConjSeqRetida.MatrizConj[I],
ConjOrdensRetFiltro.MatrizPrior[J],SeqEmAnalise);
SeqEmAnalise.PosCorr:=ConjSeqRetida.MatrizConj[I].PosCorr+1;
InsereOciosidade(SeqEmAnalise);
CalcFObjOci(SeqEmAnalise,ConjOci,SeqEmAnalise.PosCorr);

```

```

SeqEmAnalise.TempoCorr:=HorTer;
NumAnalizados:=MaximoInteiro(LargDoFiltro,LargDaBusca);
NumAnalizados:=MinimoInteiro(NumAnalizados,ConjOrdensDisp.NumAtivos);
if (ConjSeqFiltro.NumConjAtivos < NumAnalizados) then
  {se o conjunto de seq. do filtro nao estiver cheio}
  begin
    InsereConjEmConj(SeqEmAnalise,ConjSeqFiltro);
    ConjSeqFiltro.FuncObjetivo[ConjSeqFiltro.NumConjAtivos]:=
      ValFuncObjetivo;
    OrdenaConjDeConjOrd(ConjSeqFiltro);
  end
else
  {se o conjunto de seq. do filtro estiver cheio}
  begin
    PosInsercao:=0;
    for H:=ConjSeqFiltro.NumConjAtivos downto 1 do
      if (ValFuncObjetivo < ConjSeqFiltro.FuncObjetivo[H]) then
        PosInsercao:=H;
    if (PosInsercao >= 1) and
      (PosInsercao < ConjSeqFiltro.NumConjAtivos) then
      begin
        for H:=NumAnalizados downto PosInsercao+1 do
          begin
            ConjSeqFiltro.MatrizConj[H]:=ConjSeqFiltro.MatrizConj[H-1];
            ConjSeqFiltro.FuncObjetivo[H]:=ConjSeqFiltro.FuncObjetivo[H-1];
          end;
        ConjSeqFiltro.MatrizConj[PosInsercao]:=SeqEmAnalise;
        ConjSeqFiltro.FuncObjetivo[PosInsercao]:=ValFuncObjetivo;
      end;
    if (PosInsercao = ConjSeqFiltro.NumConjAtivos) then
      begin
        ConjSeqFiltro.MatrizConj[PosInsercao]:=SeqEmAnalise;
        ConjSeqFiltro.FuncObjetivo[PosInsercao]:=ValFuncObjetivo;
      end;
    end;
  end;
end;
for J:=1 to LargDaBusca do
  begin
    ConjSeqRetida.FuncObjetivo[J]:=ConjSeqFiltro.FuncObjetivo[J];
    ConjSeqRetida.MatrizConj[J]:=ConjSeqFiltro.MatrizConj[J];
  end;
end;
end;
begin {começo do programa principal}
for ILOOP:=1 to 720 do
  begin
    writeln(ILOOP);
    NomeInstancia:='C:\PASCAL\INSTANCI\INST'+StrZero(ILOOP,4)+''.TXT';
    HorIni:=Horario;
    LeInstancia;
    OrdenaExpEToci;
    for JLOOP:=1 to NumOrdens do
      DadosSeqInicial[JLOOP]:=Dados[SeqParFixa.MatrizPrior[JLOOP].Indice];
    for JLOOP:=1 to NumOrdens do
      Dados[JLOOP]:=DadosSeqInicial[JLOOP];
    BeamSearchFiltradoOcios;
    HorFin:=Horario;
    Tempo:=HorFin-HorIni;
    {abrir arquivo de saida dos resultados (tempo e funcao objetivo)}
    assign(Saida, NomeSaida);
    append(Saida);
    writeln(Saida, ILOOP:5, Espaco, HorIni:8, Espaco, HorFin:8, Espaco,
      Tempo:6, Espaco, ConjSeqRetida.FuncObjetivo[1]:18);
    close(Saida);
  end;
end.

```

PROGRAMA EDDTEMPO.PAS

```

program Edd;
{programa que sequencia um conjunto de ordens de acordo com a menor data
de entrega e calcula o valor da função objetivo. Faz esse procedimento
para o conjunto de instancias a serem testadas}
uses dos; {utilizado no calculo do tempo (no gettime)}
const
  Espaco          =#32; {separador de colunas}

```



```

NumOrdensMaximo =700;
NomeSaida      ='C:\PASCAL\DISSERTA\EDDRESUL.TXT';
SequenciaDeSaida='C:\PASCAL\DISSERTA\EDDSEQU.TXT';
{$I C:\PASCAL\DISSERTA\TENTRADA.INC} {acrescenta o arquivo de type}
var {define variaveis globais}
    Dados      :TAEntrada; {matriz para se fazer os calculos sem mexer
        na original}
    FuncObjetivo :real;
    HorFin      :longint; {horario final}
    HorIni      :longint; {horario inicial}
    ILOOP,JLOOP :integer; {indice do loop}
    NomeInstancia:string[64];
    NumOrdens   :integer; {num. de ordens lido no arquivo que sera aberto}
    Saida       :text;
    SeqSaida    :text; {sequencia de saida, de acordo com EDD}
    Tempo      :longint; {tempo de solucao}
{$I C:\PASCAL\DISSERTA\FMININT.INC}
{$I C:\PASCAL\DISSERTA\FREPLIC.INC}
{$I C:\PASCAL\DISSERTA\FSTRZERO.INC}
{$I C:\PASCAL\DISSERTA\FHORARIO.INC}
{$I C:\PASCAL\DISSERTA\PLEINST.INC}
{$I C:\PASCAL\DISSERTA\PSWPLINH.INC}
{$I C:\PASCAL\DISSERTA\PORDEDD.INC}
procedure CalcFObj;
{procedimento que calcula o valor da fun#e#o objetivo}
var
    I      :integer; {indice do loop}
    C,L,E,T,HE,WT:array[1..NumOrdensMaximo] of real; {C=horario de termino;
        L=diferen#a;E=adiantamento;T=atraso}
    FuncObj :array[1..NumOrdensMaximo] of real; {HE=peso adiantamento
        x E;HT=peso atraso x T;FUNCOBJ=funcao objetivo}
{$M 40960,0,655360}
begin
    E[1]:=0;
    T[1]:=0;
    C[1]:=Dados[1].TProces+Dados[1].Chegada;
    L[1]:=C[1]-Dados[1].Data;
    if (L[1] < 0) then
        E[1]:=-L[1]
    else
        T[1]:=L[1];
    HE[1]:=Dados[1].PenAdi*E[1];
    WT[1]:=Dados[1].PenAtr*T[1];
    FuncObj[1]:=HE[1]+WT[1];
    for I:=2 to NumOrdens do
        begin
            E[I]:=0;
            T[I]:=0;
            C[I]:=C[I-1]+Dados[I].TProces;
            L[I]:=C[I]-Dados[I].Data;
            if (L[I] < 0) then
                E[I]:=-L[I]
            else
                T[I]:=L[I];
            HE[I]:=Dados[I].PenAdi*E[I];
            WT[I]:=Dados[I].PenAtr*T[I];
            FuncObj[I]:=FuncObj[I-1]+HE[I]+WT[I];
        end;
    writeln;
    FuncObjetivo:=FuncObj[NumOrdens];
end;
{come#o do programa principal}
begin
for ILOOP:=1 to 720 do
    begin
        NomeInstancia:='C:\PASCAL\INSTANCI\INST'+StrZero(ILOOP,4)+'.TXT';
        HorIni:=Horario;
        LeInstancia;
        OrdenaEdd;
        CalcFObj;
        HorFin:=Horario;
        Tempo:=HorFin-HorIni;
        {abrir arquivo de saida dos resultados (tempo e funcao objetivo)}
        assign(Saida, NomeSaida);
        append(Saida);
        writeln(Saida, ILOOP:5, Espaco, HorIni:8, Espaco, HorFin:8, Espaco,
            Tempo:6, Espaco, FuncObjetivo:18);
        close(Saida);
    end;
end;

```

```

end;
end.

```

PROGRAMA EXPET.PAS

```

{$N+} {desliga a diretiva padrao de execucao de ponto flutuante. Utiliza a
      diretiva do processador aritmetico. WOOD, 1987, p.207}
program ExpET;
{programa que sequencia um conjunto de ordens de acordo com a funcao de
  prioridade E/T (atraso/adiantamento) e calcula o valor da funcao objetivo.
  Faz esse procedimento para o conjunto de instancias a serem testadas}
uses dos; {utilizado no calculo do tempo (no gettime)}
const
  Espaco          =#32; {separador de colunas}
  K               =5;
  NumOrdensMaximo =700;
  NomeSaida       ='C:\PASCAL\DISSERTA\ETRESUL.TXT';
  SequenciaDeSaida='C:\PASCAL\DISSERTA\ETSEQU.TXT';
  {$I C:\PASCAL\DISSERTA\TENTRADA.INC} {acresc. o arq. de type da entrada}
  {$I C:\PASCAL\DISSERTA\TCONJUNT.INC} {acresc. o arq. de type de conjuntos}
var {define variaveis globais}
  Dados          :TAEntrada; {matriz para se fazer os calculos sem mexer
    na original}
  FuncObjetivo   :real;
  HorFin         :longint;   {horario final}
  HorIni         :longint;   {horario inicial}
  ILOOP,JLOOP    :integer;   {indice do loop}
  NomeInstancia  :string[64];
  NumOrdens      :integer; {num. de ordens lido no arquivo que sera aberto}
  SeqParFixa     :ConjOrdenado; {sequencia parcial fixa}
  Saida          :text;
  SeqSaida       :text; {sequencia de saida, de acordo com ExpET}
  Tempo         :longint;   {tempo de solucao}
  {$I C:\PASCAL\DISSERTA\FMEREAL.INC}
  {$I C:\PASCAL\DISSERTA\FMAREAL.INC}
  {$I C:\PASCAL\DISSERTA\FPREXPET.INC}
  {$I C:\PASCAL\DISSERTA\FMININT.INC}
  {$I C:\PASCAL\DISSERTA\FREPLIC.INC}
  {$I C:\PASCAL\DISSERTA\FSTRZERO.INC}
  {$I C:\PASCAL\DISSERTA\FHORARIO.INC}
  {$I C:\PASCAL\DISSERTA\PLEINST.INC}
  {$I C:\PASCAL\DISSERTA\PSWPLINH.INC}
  {$I C:\PASCAL\DISSERTA\PSWPREAL.INC}
  {$I C:\PASCAL\DISSERTA\PICOVAZI.INC}
  {$I C:\PASCAL\DISSERTA\PICOCHEI.INC}
  {$I C:\PASCAL\DISSERTA\PUCO.INC}
  {$I C:\PASCAL\DISSERTA\PINFCO.INC}
  {$I C:\PASCAL\DISSERTA\PRDCO.INC}
  {$I C:\PASCAL\DISSERTA\FTPROMED.INC}
  {$I C:\PASCAL\DISSERTA\POEXPET.INC}
procedure CalcFObj;
{procedimento que calcula o valor da funcao objetivo}
var
  C,L           :array[1..NumOrdensMaximo] of real; {C=horario de termino;
    L=diferença}
  E,T           :array[1..NumOrdensMaximo] of real; {E=adiantamento; T=atraso}
  HE,WT         :array[1..NumOrdensMaximo] of real; {HE=peso adiantamento vezes
    adiantamento; HT=peso atraso vezes atraso}
  FuncObj       :array[1..NumOrdensMaximo] of real; {funcao objetivo}
  I             :integer; {indice do loop}
  {$M 40960,0,630784}
begin
  E[1]:=0;
  T[1]:=0;
  C[1]:=Dados[SeqParFixa.MatrizPrior[1].Indice].TProces+
    Dados[SeqParFixa.MatrizPrior[1].Indice].Chegada;
  L[1]:=C[1]-Dados[SeqParFixa.MatrizPrior[1].Indice].Data;
  if (L[1] < 0) then
    E[1]:=-L[1]
  else
    T[1]:=L[1];
  HE[1]:=Dados[SeqParFixa.MatrizPrior[1].Indice].PenAdi*E[1];
  WT[1]:=Dados[SeqParFixa.MatrizPrior[1].Indice].PenAtr*T[1];
  FuncObj[1]:=HE[1]+WT[1];
  for I:=2 to SeqParFixa.NumAtivos do
    begin
      E[I]:=0;

```

```

T[I]:=0;
C[I]:=C[I-1]+Dados[SeqParFixa.MatrizPrior[I].Indice].TProces;
L[I]:=C[I]-Dados[SeqParFixa.MatrizPrior[I].Indice].Data;
if (L[I] < 0) then
  E[I]:=-L[I]
else
  T[I]:=L[I];
HE[I]:=Dados[SeqParFixa.MatrizPrior[I].Indice].PenAdi*E[I];
WT[I]:=Dados[SeqParFixa.MatrizPrior[I].Indice].PenAtr*T[I];
FuncObj[I]:=FuncObj[I-1]+HE[I]+WT[I];
end;
writeln;
FuncObjetivo:=FuncObj[SeqParFixa.NumAtivos];
end;
{começo do programa principal}
begin
for ILOOP:=1 to 720 do
begin
NomeInstancia:='C:\PASCAL\INSTANCI\INST'+StrZero(ILOOP,4)+' .TXT';
HorIni:=Horario;
LeInstancia;
OrdenaExpET;
CalcFObj;
HorFin:=Horario;
Tempo:=HorFin-HorIni;
{abrir arquivo de saída dos resultados (tempo e função objetivo)}
assign(Saida, NomeSaida);
append(Saida);
writeln(Saida, ILOOP:5, Espaco, HorIni:8, Espaco, HorFin:8, Espaco,
Tempo:6, Espaco, FuncObjetivo:18);
close(Saida);
end;
end.

```

PROGRAMA EXPETOCI.PAS

{\$N+} {desliga a diretiva padrão de execução de ponto flutuante. Utiliza a diretiva do processador aritmético. WOOD, 1987, p.207}

```

program ExpET;
{programa que programa um conjunto de ordens de acordo com a função de
prioridade E/T (atraso/adiantamento), inserindo ociosidade e calcula
o valor da função objetivo. Faz esse procedimento para o conjunto de
instâncias a serem testadas}
uses dos; {utilizado no cálculo do tempo (no gettimeofday)}
const
Espaco           =#32; {separador de colunas}
K                =5;
LargMaxDoFiltro  =25;
NumOrdensMaximo  =120;
NomeSaida        ='C:\PASCAL\DISSERTA\ETRESUL.TXT';
SequenciaDeSaida='C:\PASCAL\DISSERTA\ETSEQU.TXT';
{$I C:\PASCAL\DISSERTA\TENTRADA.INC} {acresc. o arq. de type da entrada}
{$I C:\PASCAL\DISSERTA\TCONJUNT.INC} {acresc. o arq. de type de conjuntos}
var {define variáveis globais}
ConjOci          :ConjOrdenado; {conjunto das ociosidades}
Dados            :TAEntrada; {matriz para se fazer os cálculos sem mexer
na original}
FuncObjetivo     :real;
HorFin           :longint; {horário final}
HorIni           :longint; {horário inicial}
ILOOP, JLOOP     :integer; {índice do loop}
NomeInstancia    :string[64];
NumOrdens        :integer; {num. de ordens lido no arquivo que será aberto}
SeqParFixa       :ConjOrdenado; {sequência parcial fixa}
Saida            :text;
SeqSaida         :text; {sequência de saída, de acordo com ExpET}
Tempo            :longint; {tempo de solução}
{$I C:\PASCAL\DISSERTA\FMEREAL.INC}
{$I C:\PASCAL\DISSERTA\FMAREAL.INC}
{$I C:\PASCAL\DISSERTA\FMININT.INC}
{$I C:\PASCAL\DISSERTA\FREPLIC.INC}
{$I C:\PASCAL\DISSERTA\FSTRZERO.INC}
{$I C:\PASCAL\DISSERTA\FPREXPET.INC}
{$I C:\PASCAL\DISSERTA\FHORARIO.INC}
{$I C:\PASCAL\DISSERTA\FOCIEET.INC}
{$I C:\PASCAL\DISSERTA\PLEINST.INC}
{$I C:\PASCAL\DISSERTA\PSWPLINH.INC}

```

```

{$I C:\PASCAL\DISSERTA\PSWPREAL.INC}
{$I C:\PASCAL\DISSERTA\PICOVAZI.INC}
{$I C:\PASCAL\DISSERTA\PICOCHEI.INC}
{$I C:\PASCAL\DISSERTA\PUCO.INC}
{$I C:\PASCAL\DISSERTA\PINFCO.INC}
{$I C:\PASCAL\DISSERTA\PRDCO.INC}
{$I C:\PASCAL\DISSERTA\FTPROMED.INC}
{$I C:\PASCAL\DISSERTA\POEETOCI.INC}
{$I C:\PASCAL\DISSERTA\PCFOBOCI.INC}
begin {comeco do programa principal}
for ILOOP:=1 to 720 do
begin
NomeInstancia:='C:\PASCAL\INSTANCI\INST'+StrZero(ILOOP,4)+'.TXT';
HorIni:=Horario;
LeInstancia;
OrdenaExpETOCi;
CalcFObjOci(SeqParFixa, ConjOci);
HorFin:=Horario;
Tempo:=HorFin-HorIni;
{abrir arquivo de saida dos resultados (tempo e funcao objetivo)}
assign(Saida, NomeSaida);
append(Saida);
writeln(Saida, ILOOP:5, Espaco, HorIni:8, Espaco, HorFin:8, Espaco,
Tempo:6, Espaco, FuncObjetivo:18);
close(Saida);
end;
end.

```

PROGRAMA VALIDADO.PAS

```

program Validado;
{programa que valida a instfncia gerada pelo arquivo Aleatori contendo
numeros aleatorios. Leva em consideracao os parametros de fator
medio de atraso e amplitude da distribuicao de data}
const
NumOrdensMaximo =1001; {numero de ordens maximo}
Espaco          =#32; {separador de colunas}
NomeUltimo      ='C:\PASCAL\INSTANCI\ULTIMO.TXT'; {nome do fltimo arquivo}
AceiteFaAtr     =0.15; {aceitar um erro de 15% entre o nominal e o real}
AceiteAmDisDa   =0.15; {aceitar um erro de 15% entre o nominal e o real}
{$I C:\PASCAL\DISSERTA\TENTRADA.INC} {acrescenta o arquivo de type}
var
FaAtrNominal    :real; {fator de atraso nominal}
AmDisDaNomi     :real; {amplitude da distribuicao de data de entrega}
NumOrdens       :integer; {num. de ordens a ser lido, conforme o cabe-
calho do programa. O num. de ordens lido efetivamente, transforma-se
no numero de ordens do problema.}
NumOrdensLido   :integer; {numero de ordens lidas efetivamente no arquivo
que esta sendo lido}
I               :integer; {indice do loop}
Instancia       :text;
Ultimo          :text;
NomeInstancia   :string[64];
UltimoAceito    :integer;
UltimoGerado    :integer;
DataMaxReal     :integer; {data de entrega real maxima}
DataMinReal     :integer; {data de entrega real minima}
DataMedia       :array[1..NumOrdensMaximo] of real; {data media acumulada}
FaAtrReal       :real; {fator de atraso real}
AmDisDaReal     :real; {amplitude real da distribuicao de data}
PropFaAtr       :real;
PropAmDisDa     :real;
Dados           :TAEEntrada;
SomaTProces     :array[1..NumOrdensMaximo] of real; {somatorio TProces}
SomaData        :array[1..NumOrdensMaximo] of real; {somatorio data}
{$I C:\PASCAL\DISSERTA\FMININT.INC}
{$I C:\PASCAL\DISSERTA\FREPLIC.INC}
{$I C:\PASCAL\DISSERTA\FSTRZERO.INC}
{$I C:\PASCAL\DISSERTA\FMAXINT.INC}
{comeco do programa principal}
begin
{le arquivo Ultimo.txt, com os valores UltimoAceito e UltimoGerado}
assign(Ultimo, NomeUltimo); {associa o nome Ultimo com o arquivo Ultimo}
reset(Ultimo);
readln(Ultimo, UltimoAceito, UltimoGerado);
close(Ultimo);
{cria nome da ultima instancia que foi gerada}

```

```

UltimoGerado:=UltimoAceito+1;
NomeInstancia:='C:\PASCAL\INSTANCI\INST'+StrZero(UltimoGerado,4)+'.TXT';
{abrir arquivo de instancias para leitura apenas, verificando se esta
  tudo certo}
assign(Instancia,NomeInstancia);
reset(Instancia);
readln(Instancia,NomeInstancia);
readln(Instancia); {le uma linha de informacao do arquivo}
readln(Instancia,NumOrdens,FaAtrNominal,AmDisDaNomi);
readln(Instancia); {le uma linha de informacao do arquivo}
if (NumOrdens > NumOrdensMaximo) then
begin
  writeln('Aten#Eo! O numero de ordens apontado no cabecalho da instancia');
  writeln('e maior do que o numero maximo permitido. O programa foi in-');
  writeln('terrompido. Verificar o erro antes de continuar!');
  writeln;
  halt;
end;
{ler os dados da ultima instancia que foi gerada}
NumOrdensLido:=0;
for I:=1 to NumOrdens do
  {$I-} {desliga a verificacao de erros}
  begin
    readln(Instancia,
      Dados[I].Ordem,Dados[I].Chegada,Dados[I].TProces,
      Dados[I].Data,Dados[I].PenAdi,Dados[I].PenAtr);
    if (IOResult <> 0) then {caso haja algum erro de leitura}
    begin
      writeln('Aconteceu algum erro na leitura do arquivo ',NomeInstancia);
      writeln('Verificar antes de continuar!');
      writeln;
      exit;
    end
    else {caso nao haja erro de leitura}
    begin
      NumOrdensLido:=NumOrdensLido+1;
    end;
  end;
  {$I+} {religa a verificacao de erros}
{calcula os parametros fator de atraso medio e amplitude da distribuicao
de data de entrega}
SomaTProces[1]:=Dados[1].TProces;
SomaData[1]:=Dados[1].Data;
DataMinReal:=Dados[1].Data;
DataMaxReal:=Dados[1].Data;
for I:=2 to NumOrdens do
begin
  SomaTProces[I]:=SomaTProces[I-1]+Dados[I].TProces;
  SomaData[I]:=SomaData[I-1]+Dados[I].Data;
  DataMinReal:=MinimoInteiro(Dados[I].Data,DataMinReal);
  DataMaxReal:=MaximoInteiro(Dados[I].Data,DataMaxReal);
  DataMedia[I]:=(1/I)*SomaData[I];
end;
FaAtrReal:=1-(DataMedia[NumOrdens]/SomaTProces[NumOrdens]);
AmDisDaReal:=(DataMaxReal-DataMinReal)/SomaTProces[NumOrdens];
WRITELN(FAATRREAL,' ',AMDISDAREAL);
{comparacao dos parametros nominais com os reais}
PropAmDisDa:=abs(AmDisDaNomi-AmDisDaReal)/AmDisDaNomi;
PropFaAtr:=abs(FaAtrNominal-FaAtrReal)/FaAtrNominal;
{se as proporcoes estao dentro das aceitaveis, aceitar o arquivo
como sendo o ultimo aceito}
if (PropAmDisDa < AceiteAmDisDa) and (PropFaAtr < AceiteFaAtr) then
begin
  assign(Instancia,NomeInstancia);
  append(Instancia);
  writeln(Instancia,'Fator de atraso real= ',FaAtrReal:5:3);
  writeln(Instancia,'Amplitude real da distr. data= ',AmDisDaReal:5:3);
  UltimoAceito:=UltimoAceito+1;
  close(Instancia);
  {reabre arquivo Ultimo para escrita}
  assign(Ultimo,NomeUltimo);
  rewrite(Ultimo);
  writeln(Ultimo,UltimoAceito,Espaco,UltimoGerado);
  close(Ultimo);
end;
end.

```

PROGRAMA WSPT.PAS

```

program WSpt;
{programa que sequencia um conjunto de ordens de acordo com o menor tempo
 de processamento ponderado e calcula o valor da função objetivo. Faz esse
 procedimento para o conjunto de instancias a serem testadas}
uses dos; {utilizado no calculo do tempo (no gettime)}
const
  Espaco          =#32; {separador de colunas}
  NumOrdensMaximo =500;
  NomeSaida       ='C:\PASCAL\DISSERTA\WSPT\WSPTRRES.TXT';
  SequenciaDeSaida='C:\PASCAL\DISSERTA\SPTSEQ.TXT';
{$I C:\PASCAL\DISSERTA\TENTRADA.INC} {acrescenta o arquivo de type}
var {define variaveis globais}
  Dados          :TAEntrada; {matriz para se fazer os calculos sem mexer
    na original}
  FuncObjetivo   :real;
  HorFin         :longint;   {horario final}
  HorIni         :longint;   {horario inicial}
  ILOOP,JLOOP    :integer;   {indice do loop}
  NomeInstancia  :string[64];
  NumOrdens      :integer; {num. de ordens lido no arquivo que sera aberto}
  Saida          :text;
  SeqSaida       :text; {sequencia de saida, de acordo com SPT}
  Tempo          :longint;   {tempo de solucao}
{$I C:\PASCAL\DISSERTA\FMININT.INC}
{$I C:\PASCAL\DISSERTA\FREPLIC.INC}
{$I C:\PASCAL\DISSERTA\FSTRZERO.INC}
{$I C:\PASCAL\DISSERTA\FHORARIO.INC}
{$I C:\PASCAL\DISSERTA\PLEINST.INC}
{$I C:\PASCAL\DISSERTA\PSWPLINH.INC}
procedure OrdenaWSpt;
{procedim. que sequencia de acordo com SPT(menor tempo de processamento)}
var
  I,J      :integer;   {indices para loops}
begin
  for I:=1 to NumOrdens-1 do
    for J:=I+1 to NumOrdens do
      if ((Dados[I].PenAtr/Dados[I].TProces) <
        (Dados[J].PenAtr/Dados[J].TProces)) then
        SwapLinha(Dados[I],Dados[J]);
    end;
procedure CalcFObj;
{procedimento que calcula o valor da fun#Eo objetivo}
var
  I      :integer; {indice do loop}
  C,L,E,T,HE,WT:array[1..NumOrdensMaximo] of real; {C=horario de termino;
    L=diferen#a;E=adiantamento;T=atraso}
  FuncObj :array[1..NumOrdensMaximo] of real; {HE=peso adiantamento
    x E;HT=peso atraso x T;FUNCOBJ=funcao objetivo}
{$M 40960,0,655360}
begin
  E[1]:=0;
  T[1]:=0;
  C[1]:=Dados[1].TProces+Dados[1].Chegada;
  L[1]:=C[1]-Dados[1].Data;
  if (L[1] < 0) then
    E[1]:=-L[1]
  else
    T[1]:=L[1];
  HE[1]:=Dados[1].PenAdi*E[1];
  WT[1]:=Dados[1].PenAtr*T[1];
  FuncObj[1]:=HE[1]+WT[1];
  for I:=2 to NumOrdens do
    begin
      E[I]:=0;
      T[I]:=0;
      C[I]:=C[I-1]+Dados[I].TProces;
      L[I]:=C[I]-Dados[I].Data;
      if (L[I] < 0) then
        E[I]:=-L[I]
      else
        T[I]:=L[I];
      HE[I]:=Dados[I].PenAdi*E[I];
      WT[I]:=Dados[I].PenAtr*T[I];
      FuncObj[I]:=FuncObj[I-1]+HE[I]+WT[I];
    end;
  writeln;

```

```

    FuncObjetivo:=FuncObj[NumOrdens];
end;
begin {começo do programa principal}
for ILOOP:=1 to 720 do
begin
NomeInstancia:='C:\PASCAL\INSTANCI\INST'+StrZero(ILOOP,4)+'.TXT';
HorIni:=Horario;
LeInstancia;
OrdenaWSpt;
CalcFObj;
HorFin:=Horario;
Tempo:=HorFin-HorIni;
{abrir arquivo de saída dos resultados (tempo e funcao objetivo)}
assign(Saida,NomeSaida);
append(Saida);
writeln(Saida,ILOOP:5,Espaco,HorIni:8,Espaco,HorFin:8,Espaco,
        Tempo:6,Espaco,FuncObjetivo:18);
close(Saida);
end;
end.

```

APÊNDICE 2: EXEMPLOS DE INSTÂNCIAS

Neste apêndice são apresentados dois exemplos de instâncias utilizadas no estudo computacional. Pelo tamanho de cada instância percebe-se a dificuldade de se apresentar outros exemplos. A primeira linha identifica o nome da instância, na quarta linha o número de ordens (100 no caso), e os parâmetros nominais fator de atraso médio e amplitude relativa das datas de entrega. A coluna ordem identifica o número da ordem, e as colunas r,p,d,h,w representam o horário de chegada, o tempo de processamento, a data de entrega, a penalidade de adiantamento e a penalidade de atraso respectivamente. A penúltima e a última linha representam os valores reais – calculados após a geração computacional – dos parâmetros da instância em questão.

instância 001

C:\PASCAL\INSTANCI\INST0001.TXT

(Num. de ordens) (fator de atraso nominal) (amplitude)

100 0.10 0.10

Ordem r p d h w

1	0	2	4480	52	85
2	0	29	4626	38	9
3	0	71	4580	65	31
4	0	11	4505	55	90
5	0	20	4509	77	2
6	0	58	4554	1	63
7	0	37	4774	89	3
8	0	16	4488	71	6
9	0	3	4681	26	56
10	0	54	4658	79	65
11	0	97	4732	93	68
12	0	93	4552	5	3
13	0	71	4759	18	37
14	0	45	4370	20	57
15	0	15	4696	76	90
16	0	21	4770	24	60
17	0	24	4336	5	67
18	0	27	4450	42	83
19	0	33	4601	14	38
20	0	55	4552	32	25
21	0	57	4454	71	60
22	0	60	4736	63	39
23	0	99	4624	82	61
24	0	22	4548	84	91
25	0	68	4355	58	84
26	0	95	4770	98	53
27	0	69	4571	65	45
28	0	42	4408	92	53
29	0	80	4622	67	35
30	0	72	4688	86	33
31	0	26	4607	59	74
32	0	39	4630	45	71
33	0	39	4313	77	82
34	0	47	4493	93	32
35	0	92	4641	2	1
36	0	46	4656	12	41
37	0	42	4452	54	44
38	0	94	4736	26	18
39	0	35	4768	73	96

40	0	32	4607	66	40
41	0	34	4310	71	41
42	0	25	4535	44	29
43	0	89	4545	6	28
44	0	4	4334	87	4
45	0	36	4779	9	71
46	0	92	4679	81	40
47	0	59	4732	12	6
48	0	18	4306	82	85
49	0	47	4507	74	56
50	0	12	4414	42	18
51	0	42	4480	78	58
52	0	97	4787	90	16
53	0	60	4766	20	16
54	0	70	4548	30	93
55	0	29	4368	90	87
56	0	50	4467	1	64
57	0	74	4749	44	26
58	0	41	4342	46	25
59	0	27	4770	42	33
60	0	64	4463	61	8
61	0	4	4484	8	81
62	0	88	4673	16	48
63	0	69	4512	44	38
64	0	35	4325	95	88
65	0	1	4499	53	61
66	0	90	4484	91	45
67	0	24	4348	42	7
68	0	77	4457	13	76
69	0	53	4304	10	85
70	0	47	4611	62	16
71	0	8	4609	26	65
72	0	27	4651	18	32
73	0	13	4471	4	84
74	0	88	4575	81	27
75	0	3	4353	45	33
76	0	65	4366	31	70
77	0	94	4497	82	53
78	0	94	4719	59	78
79	0	62	4346	16	79
80	0	24	4594	90	24
81	0	94	4605	16	25
82	0	8	4694	91	75
83	0	34	4308	40	87


```

84 0 99 4471 72 2
85 0 9 4785 13 20
86 0 32 4582 68 76
87 0 55 4488 14 9
88 0 70 4558 6 56
89 0 29 4408 35 19
90 0 66 4327 74 65
91 0 83 4592 39 44
92 0 98 4624 70 42
93 0 71 4404 4 25
94 0 99 4586 23 7

```

```

95 0 98 4579 39 86
96 0 57 4736 27 79
97 0 85 4690 20 18
98 0 33 4478 48 20
99 0 16 4423 74 94
100 0 30 4628 20 19
Fator de atraso real= 0.097
Amplitude real da distr. data= 0.096

```

instância 393

C:\PASCAL\INSTANCI\INST0393.TXT
 (Num. de ordens) (fator de atraso
 nominal) (amplitude)
 100 0.60 0.20

```

Ordem r p d h w
1 0 52 2424 10 25
2 0 62 2028 6 81
3 0 19 1589 13 18
4 0 70 1715 9 21
5 0 22 2362 17 15
6 0 43 1699 67 42
7 0 31 1924 92 93
8 0 28 1907 60 14
9 0 71 2353 48 97
10 0 84 2359 18 21
11 0 91 2422 42 40
12 0 33 1608 55 57
13 0 93 2142 18 14
14 0 35 1536 14 20
15 0 82 1851 85 60
16 0 28 1537 63 89
17 0 58 1809 33 56
18 0 77 1729 41 91
19 0 47 1859 76 63
20 0 82 1859 11 55
21 0 70 2145 51 77
22 0 7 2269 80 51
23 0 45 2431 41 71
24 0 52 2352 43 21
25 0 36 2367 43 5
26 0 77 1850 68 92
27 0 23 1830 60 44
28 0 38 2458 4 86
29 0 27 2185 57 50
30 0 77 2505 55 99
31 0 62 1814 21 73
32 0 49 1796 71 93
33 0 34 1860 84 20
34 0 82 1585 76 52
35 0 60 1975 79 13
36 0 72 1894 46 43
37 0 45 2486 17 38
38 0 23 2483 10 48
39 0 81 1923 25 81
40 0 22 1647 1 7
41 0 93 2228 8 47
42 0 48 2449 36 13
43 0 20 2013 46 74
44 0 18 1753 40 39
45 0 25 2329 39 26
46 0 80 1852 2 7
47 0 62 2170 83 82
48 0 34 2407 71 85
49 0 61 1553 84 4
50 0 90 2099 90 44
51 0 59 1836 9 98
52 0 70 1563 65 8
53 0 65 2075 69 67

```

```

54 0 45 1665 71 2
55 0 75 2039 65 34
56 0 9 1616 10 78
57 0 98 2139 33 78
58 0 83 2190 76 35
59 0 55 1525 52 32
60 0 79 1929 87 11
61 0 83 1598 89 85
62 0 72 2379 45 84
63 0 93 2201 71 73
64 0 26 1598 24 54
65 0 70 1763 96 35
66 0 65 2090 7 96
67 0 90 2348 96 93
68 0 61 2069 58 25
69 0 99 1850 13 82
70 0 76 2163 31 28
71 0 11 2506 36 96
72 0 31 1941 7 3
73 0 56 1970 60 9
74 0 28 2245 86 63
75 0 55 1833 92 69
76 0 84 1607 14 51
77 0 44 1831 89 25
78 0 29 2287 21 27
79 0 20 2309 31 86
80 0 24 1859 34 35
81 0 41 1913 40 99
82 0 24 2165 3 89
83 0 53 1839 30 24
84 0 78 1697 73 16
85 0 62 1846 3 57
86 0 65 1731 96 93
87 0 29 2494 59 84
88 0 64 1865 68 85
89 0 23 1591 84 92
90 0 67 2216 22 53
91 0 94 1938 31 88
92 0 37 1878 41 61
93 0 3 2499 68 90
94 0 77 1794 18 70
95 0 52 1997 62 10
96 0 70 1982 57 13
97 0 6 1533 77 32
98 0 32 1709 4 52
99 0 99 1549 45 69
100 0 7 1983 5 56
Fator de atraso real= 0.629
Amplitude real da distr. data= 0.183

```

ERRATA DA DISSERTAÇÃO DE MESTRADO: *BEAM SEARCH* E INSERÇÃO DE OCIOSIDADE NO PROBLEMA DE PROGRAMAÇÃO DE UMA MÁQUINA EM AMBIENTE DO TIPO JIT

Emerson Carlos Colin, 14/10/97.

página	onde se lê	leia-se
ii	seu sabedoria	sua sabedoria
iv	troca de peras	troca de pares
xi	processadas no nó q do nível ν da árvore de busca	processadas no nível ν da árvore de busca cujo nó pai é o nó q
17	atraso e o adiantamento	adiantamento e o atraso
33	utilizaram	utilizaram
36	inserira	inserida
36	uma árvores	uma árvore
37	o valor	os valores
39	conideração	consideração
39	nós B,C e D	nós B,C e E
48	processadas possuem	processadas possui
57 (figura)	$Primeiro(i)+2$	$PPrimeiro(i)+2$
58	anexo 4	Anexo D
59	bloco t	bloco l
74	retiradas	obtidas
101 e 102 (cabeçalho)	Referências Bibliográficas	Anexo D: Demonstração do teorema