UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

DEPARTAMENTO DE ENGENHARIA DE SISTEMAS

UMA CONTRIBUIÇÃO PARA O ESTUDO DO PROBLEMA DE *FLOWSHOP* COM BUFFER ILIMITADO E ZERO PARA MINIMIZAR A SOMA DOS ATRASOS

DÉBORA PRETTI RONCONI

Orientador:

Prof. Dr. Vinícius Amaral Armentano

Este exemplar corresponde a redação final da tese defendida por DÉBORA PRETTI RONCONI e aprovada pela Comissão Julgada em 03 / 10 / 97.

Vinícius A. Armentano
Orientador

Tese apresentada à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas - UNICAMP, como parte dos requisitos exigidos para a obtenção do título de Doutor na Engenharia Elétrica.

- OUTUBRO 1997 -

R667c

32347/BC

UNICAMP
BIBLIOTECA CENTRAL

UNIDADE BC
N.º CHAMADA: TUNICAMP
R667c
V. Ex.
TOMBO BC/32347
PROC. 98/197
C D X
PREÇO R$ 11,00
DATA 29/12/97
Nº CPD

CM-00103572-8

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

R667c

Ronconi, Débora Pretti
Uma contribuição para o estudo do problema de flowshop com buffer ilimitado e zero para minimizar a soma dos atrasos / Débora Pretti Ronconi.--Campinas, SP: [s.n.], 1997.

Orientador: Vinícius Amaral Armentano
Tese (doutorado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Planejamento da produção. 2. Heurística. 3. Otimização combinatória. I. Armentano, Vinícius Amaral. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. IV. Título.

Agradecimentos

Agradeço a todos que colaboraram para a realização deste trabalho e em especial:

- a Deus pelo seu companheirismo;

- ao Vinícius, pela orientação cuidadosa e motivadora;

- ao Ernesto pelo amor e pela paciência de me ouvir falar de *Tabu & Branch-and-Bound*;

- à Fran, Cintia e Denise pelas valorosas conversas sobre a tese;

- à Regina, Pablo, Lúcia, Renata, Vitória, Hamilton, Jussara, Gonzaga, Gelson, Cássio, Ricardo, Luciana, Valéria, pelos momentos extremamente agradáveis que passei e que seguramente não vou esquecer;

- à Márcia pela sua paciência;

- ao Walcir pelo apoio computacional e pelo telefone;

- à Luciana pelo apoio computacional e pelas bolachas;

- aos funcionários da BAE, em especial à Albertina, Rose, Raquel, e Vera pela eficiência e simpatia;

- a CAPES pelo apoio financeiro.

Resumo

Este trabalho trata do problema de programar **n** tarefas em **m** máquinas no ambiente *flowshop* permutacional com buffer entre máquinas ilimitado e zero. O critério de otimalidade utilizado é a minimização da soma dos atrasos das tarefas. Primeiramente são sugeridas formulações de programação inteira mista para os problemas. Além disso realiza-se uma análise do comportamento de todas as soluções para diferentes cenários de datas de entrega para problemas de dimensão reduzida. A seguir são propostas heurísticas que utilizam Busca Tabu como método de exploração do espaço de soluções. Estratégias de diversificação, intensificação e de redução da vizinhança são apresentadas e avaliadas. O método *Branch-and-Bound* é utilizado para obter soluções ótimas, e um limitante inferior para o problema de *flowshop* com buffer zero é proposto. Testes computacionais são apresentados e a heurística é comparada com o algoritmo NEH e com o método *Branch-and-Bound.*

Abstract

This work addresses the problem of scheduling n tasks on m machines in the permutation flowshop with both unlimited and zero buffers with the objective of minimizing total tardiness. First, we suggest mixed integer programming formulations for the problems. Furthermore, the behavior of all solutions for small problems is analyzed for different due date scenarios. Then Tabu Search-based heuristics are proposed as a method to explore the solution space. Diversification, intensification, and neighborhood restriction strategies are presented and evaluated. The Branch-and-Bound method is applied to obtain optimal solutions, and a lower bound for the flowshop problem with zero buffer is proposed. Computational tests are presented and compared with the NEH algorithm and with the Branch-and-Bound method.

Sumário

Introdução 1

Capítulo 1. Problema *Flowshop* 3

1.1. INTRODUÇÃO 3

1.2. PROBLEMA DE *FLOWSHOP* COM BUFFER ILIMITADO 4

1.2.1. ANÁLISE DE COMPORTAMENTO DAS SOLUÇÕES 9

1.3. PROBLEMA DE *FLOWSHOP* COM BUFFER ZERO 13

1.3.1. ANÁLISE DE COMPORTAMENTO DAS SOLUÇÕES 16

1.4. REVISÃO BIBLIOGRÁFICA 18

Capítulo 2. Heurísticas Propostas - Versões Básicas 24

2.1. INTRODUÇÃO 24

2.2. BUSCA TABU 24

2.3. DESCRIÇÃO DA VERSÃO BÁSICA DAS HEURÍSTICAS 26

2.3.1. SOLUÇÃO INICIAL 26

2.3.2. COMPONENTES DA BUSCA TABU PARA *FLOWSHOP* COM BUFFER ILIMITADO 31

2.3.3. COMPONENTES DA BUSCA TABU PARA *FLOWSHOP* COM BUFFER ZERO 39

Capítulo 3. Heurísticas Propostas - Estratégias Adicionais 43

3.1. INTRODUÇÃO 43

3.2. PROBLEMA DE *FLOWSHOP* COM BUFFER ILIMITADO 45

3.2.1. ESTRATÉGIAS DE DIVERSIFICAÇÃO 45

3.2.2. ESTRATÉGIAS DE INTENSIFICAÇÃO 51

3.2.3. ESTRATÉGIAS DE DIVERSIFICAÇÃO E INTENSIFICAÇÃO 54

3.3. PROBLEMA DE *FLOWSHOP* COM BUFFER ZERO 57
3.3.1. ESTRATÉGIAS DE DIVERSIFICAÇÃO 57
3.3.2. ESTRATÉGIAS DE INTENSIFICAÇÃO 58
3.3.3. ESTRATÉGIAS DE DIVERSIFICAÇÃO E INTENSIFICAÇÃO 59
3.4. REDUÇÃO DA VIZINHANÇA 61
3.5. ANÁLISE GERAL DOS RESULTADOS 68
Capítulo 4. Métodos Ótimos 70
4.1. INTRODUÇÃO 70
4.2. MÉTODO *BRANCH-AND-BOUND* PARA *FLOWSHOP* COM BUFFER ILIMITADO 70
4.3. MÉTODO *BRANCH-AND-BOUND* PARA *FLOWSHOP* COM BUFFER ZERO 80
Capítulo 5. Conclusões 93
Bibliografia 96

Introdução

O objetivo desta tese é estudar um problema de programação de tarefas, conhecido na língua inglesa como *job scheduling*, e aplicar a este métodos de resolução. O problema genérico de programação consiste na alocação de recursos para tarefas, respeitando as restrições impostas e otimizando um ou mais critérios. Os recursos e tarefas podem ter várias formas. Os recursos podem ser máquinas numa fábrica, costureiras de uma confecção, pistas de um aeroporto, unidades de processamento de um computador, etc.. As tarefas podem ser operações num processo de produção, aterrissagens e decolagens de aviões, execuções de programas, etc.. As tarefas podem ter níveis de prioridades, disponibilidades e datas de entrega distintas. Os critérios de otimalidade mais comumente utilizados são: minimização do tempo de término da última tarefa (*makespan*), minimização do número de tarefas atrasadas no sistema, minimização do fluxo ponderado das tarefas no sistema, minimização dos atrasos das tarefas em relação às datas de entrega (*tardiness*), e minimização do desvio em relação às datas de entrega (*lateness*).

De forma geral, o problema de programação de tarefas é muito complexo devido a sua natureza combinatorial, e dentre os problemas de otimização combinatória pode ser classificado como um dos mais difíceis. Este trabalho aborda o ambiente *flowshop* com capacidade de armazenamento ou buffer entre máquinas ilimitado e zero. Este ambiente consiste de **n** tarefas que devem ser processadas por **m** máquinas ordenadas. A quantidade de tarefas e máquinas, o tempo de processamento das operações, e a data de entrega das tarefas são dados conhecidos e fixos, como necessário para problemas determinísticos. O critério de otimalidade selecionado é a minimização da soma dos atrasos das tarefas. O nível de dificuldade deste problema pode ser estimado através de um caso particular cuja resolução é comprovadamente complexa: o problema com uma máquina é NP-*hard* para o critério de minimização da soma dos atrasos das tarefas (Du & Leung 1990).

Critérios de otimalidade envolvendo datas de entrega são de grande importância nos sistemas de manufatura pois podem existir uma série de custos quando uma tarefa é entregue com atraso. Dentre estes podem ser citados: penalidades contratuais, perda de credibilidade resultando em uma boa probabilidade de perder o cliente para algumas, ou para todas as possíveis futuras tarefas, e danos na reputação da empresa que poderão afastar outros clientes. A dimensão destes custos depende da

importância da encomenda, do cliente, e da dimensão do atraso da entrega. A dificuldade de lidar com a soma dos atrasos, e também com a maioria das outras medidas de desempenho relacionadas com atraso, se baseia no fato que este não é uma função linear do *makespan* (Baker 1974). A pesquisa sobre o critério da soma dos atrasos para o ambiente *flowshop* é limitada conforme observado por Sen & Gupta(1984), e mais recentemente por Koulomas(1994). Embora poucos trabalhos abordem o problema envolvendo atraso, este critério é muito importante em muitos sistemas reais. Por exemplo, manufatura de papel, processamento e refinamento de metais, e fabricação de pães podem ser caracterizados por um processo de fluxo (Raman 1995). Além destas aplicações, Dudek *et al.*(1992) e Reklaitis(1982) indicam as indústrias químicas como uma das mais relevantes aplicações do *flowshop*. Dado que os equipamentos específicos para os processos químicos são caros, existe uma forte motivação para otimizar este ambiente de produção.

A proposta deste trabalho é estudar o problema de *flowshop* com buffer ilimitado e zero e propor métodos de resoluções para os mesmos. No capítulo 1 apresenta-se uma descrição destes ambientes onde são sugeridas duas formulações de programação inteira mista, além de uma análise do comportamento de suas soluções para diferentes cenários de datas de entrega. No capítulo 2 apresenta-se a estrutura básica das heurísticas propostas que utilizam Busca Tabu como método de exploração do espaço de soluções. O capítulo 3 contém os resultados computacionais das implementações das estratégias de diversificação, intensificação e redução da vizinhança. No capítulo 4 são apresentados métodos ótimos baseados na técnica *Branch-and-Bound*. Além disso, sugere-se um limitante inferior para a soma dos atrasos para o problema de *flowshop* com buffer zero. O capítulo 5 apresenta as conclusões deste trabalho.

Capítulo 1

Problema de *Flowshop*

1.1. INTRODUÇÃO

O ambiente abordado neste trabalho se caracteriza pelo processamento de tarefas em linha, ou seja **n** tarefas que são processadas por **m** máquinas na mesma ordem. Este ambiente é conhecido como *flowshop*, onde a quantidade de tarefas e máquinas, o tempo de processamento das operações, e a data de entrega das tarefas são dados conhecidos e fixos. Além disso, todas as tarefas estão disponíveis para processamento no instante zero; os tempos de preparação das máquinas são independentes da seqüência e estão incluídos nos tempos de processamento; as máquinas estão continuamente disponíveis, e as operações não podem ser interrompidas. Estas propriedades definem que este ambiente pode ser visualizado como um problema determinístico e estático. O problema consiste em determinar a seqüência de processamento das tarefas nas máquinas, e o início do processamento de cada tarefa em cada máquina de forma a minimizar a soma dos atrasos. A Figura 1.1 ilustra este ambiente.

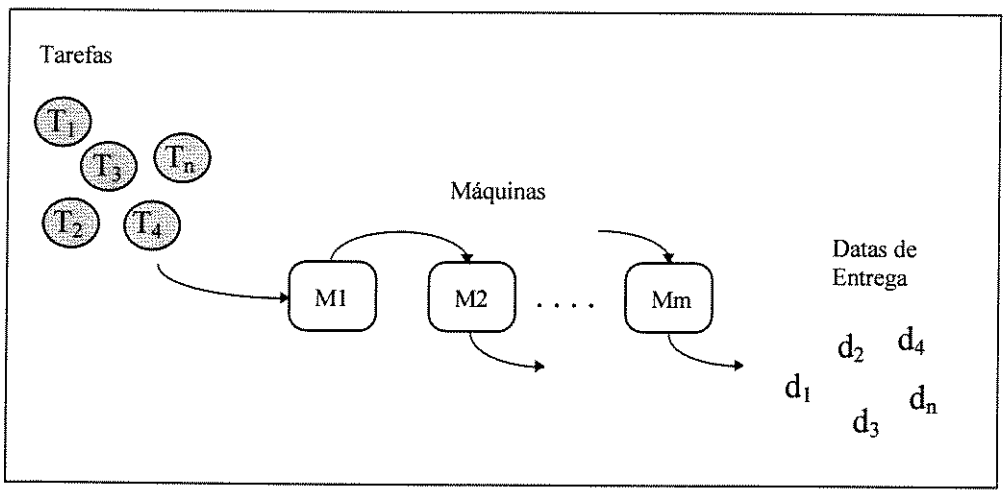


Figura 1.1. Representação gráfica do ambiente *flowshop* com datas de entrega.

Descreve-se a seguir características básicas dos dois ambientes *flowshop* selecionados para estudo neste trabalho: *flowshop* com buffer ilimitado e *flowshop* com buffer zero.

1.2. PROBLEMA DE *FLOWSHOP* COM BUFFER ILIMITADO

Este problema de *flowshop*, além de representar um processo de fluxo de **n** tarefas em **m** máquinas, se caracteriza por não impor restrições sobre a quantidade de tarefas que possa existir na fila de qualquer máquina. Analisando-se o problema, facilmente se verifica que existem $(n!)^m$ diferentes alternativas para seqüenciar **n** tarefas em **m** máquinas com buffer ilimitado. Porém, na maioria das pesquisas considera-se somente um subconjunto destas alternativas, partindo do princípio que a seqüência das tarefas nas máquinas é a mesma. Neste caso, o número de alternativas é reduzido para (n!) e os programas correspondentes são chamados de programas de permutação. Kim(1993b) apresenta comparações entre programas de permutação e não-permutação obtidos com regras de despacho ou regras de prioridades (regras que ordenam as tarefas a serem processadas) para o problema de *flowshop* com buffer ilimitado. O desempenho das regras que consideram os programas de não-permutação se mostra levemente superior ao desempenho das regras que consideram somente os programas de permutação, porém quando adicionou-se uma busca local baseada em trocas de pares de tarefas adjacentes na solução das regras que só consideram permutação, obteve-se um ganho considerável sobre as regras de despacho que consideram programas de não-permutação. Pode ser observado que a maioria dos trabalhos trata do *flowshop* permutacional dado que é mais fácil projetar heurísticas construtivas mais elaboradas, e heurísticas de busca tais como Busca Tabu, *Simulated Annealing*, e Algoritmos Genéticos para este problema. Neste trabalho consideram-se somente os programas de permutação.

Considere uma seqüência de tarefas $t_1, t_2, ..., t_i, ..., t_n$ onde t_i representa a tarefa na i-ésima posição da seqüência, p_{ij} o tempo de processamento da tarefa t_i na máquina j, e C_{ij} o instante de término de t_i em cada máquina j. O valor de C_{ij} pode ser calculado recursivamente da seguinte forma:

$$C_{1j} = \sum_{k=1}^{j} p_{1k} \qquad j = 1,...,m$$

$$C_{i1} = \sum_{z=1}^{i} p_{z1} \qquad i = 1,...,n.$$

$$C_{ij} = \max(C_{i,j-1}, C_{i-1,j}) + p_{ij} \qquad i = 2,...,n \quad j = 2,...,m$$

Calculados os instantes de término das tarefas na máquina **m,** obtém-se a soma dos atrasos individuais (T_i) baseados nas datas de entrega de cada tarefa (d_i).

$$\sum_{i=1}^{n} T_i = \sum_{i=1}^{n} \max(C_{im} - d_i, 0)$$

Estas fórmulas apesar de simples, podem se tornar extremamente custosas em termos de tempo computacional quando se considera a avaliação de um grande número de soluções. A complexidade do cálculo da soma dos atrasos é O(n.m).

Apresenta-se a seguir um modelo de programação inteira mista para o problema. O modelo é uma adaptação da formulação apresentada por Wagner(1959) para o problema de *flowshop* com o objetivo de minimizar o *makespan*. Para formular este problema considere os seguintes parâmetros:

p_{kj}: tempo de processamento da tarefa k na máquina j;

d_k: data de entrega da tarefa k;

e as seguintes variáveis:

X_{ki}: assume valor 1 caso a tarefa k esteja na posição i da seqüência, caso contrário, assume valor 0;

I_{ij}: tempo ocioso na máquina j entre o processamento das tarefas na posição i e i+1;

W_{ij}: tempo de espera da tarefa na posição i entre a máquina j e j+1;

C_{ij}: instante de término da tarefa na posição i na máquina j;

T_i: atraso da tarefa na posição i.

Formulação:

minimizar $$\sum_{i=1}^{n} T_i \qquad (1)$$

sujeito a $$T_i \geq 0 \qquad i = 1,..., n \qquad (2)$$

$$T_i \geq C_{im} - \sum_{k=1}^{n} X_{ki}.d_k \qquad i = 1,..., n \qquad (3)$$

$$C_{1m} = \sum_{j=1}^{m} \sum_{k=1}^{n} X_{k1} \cdot p_{kj} \qquad (4)$$

$$C_{im} = C_{i-1,m} + I_{i-1,m} + \sum_{k=1}^{n} X_{ki} \cdot p_{km} \qquad i = 2,..., n \qquad (5)$$

$$\sum_{k=1}^{n} X_{ki} = 1 \qquad i = 1,..., n \qquad (6)$$

$$\sum_{i=1}^{n} X_{ki} = 1 \qquad k= 1,..., n \qquad (7)$$

$$I_{ij} + \sum_{k=1}^{n} X_{k,i+1} \cdot p_{kj} + W_{i+1,j} - W_{ij} - \sum_{k=1}^{n} X_{ki} \cdot p_{k,j+1} - I_{i,j+1} = 0 \qquad (8)$$

$$i = 1,..., n-1; j = 1,..., m-1$$

$$I_{ij} \geq 0 \qquad i = 1,..., n-1; j = 2,..., m \qquad (9)$$

$$W_{ij} \geq 0 \qquad i = 2,..., n; j = 1,..., m-1 \qquad (10)$$

$$X_{ki} \in \{0,1\} \qquad k = 1,..., n; i = 1,..., n \qquad (11)$$

$$I_{i1} = 0 \qquad i = 1,..., n-1 \qquad (12)$$

$$W_{1j} = 0 \qquad j = 1,..., m-1 \qquad (13)$$

A função objetivo (1) representa a soma dos atrasos. Os conjuntos de restrições (2) e (3) correspondem à linearização de $T_i = \max (C_{im} - d_i , 0)$. Os conjuntos (4) e (5) representam o cálculo dos instantes de término das tarefas na máquina **m** para cada posição **i**. As restrições (6) e (7) especificam, respectivamente, que cada tarefa pode ocupar uma única posição, e que cada posição pode conter uma única tarefa. O conjunto de restrições (8) relaciona as variáveis de decisão com as restrições físicas do problema. Estas restrições estabelecem as relações necessárias para manter a consistência entre o tempo ocioso das máquinas e o tempo de espera das tarefas. Estas representam a diferença entre o tempo $\mathbf{t_1}$ em que a tarefa na posição **i** termina seu processamento na máquina **j,** e o tempo $\mathbf{t_2}$ em que a tarefa na posição **i+1** começa na máquina **j+1,** como ilustrado na Figura 1.2.

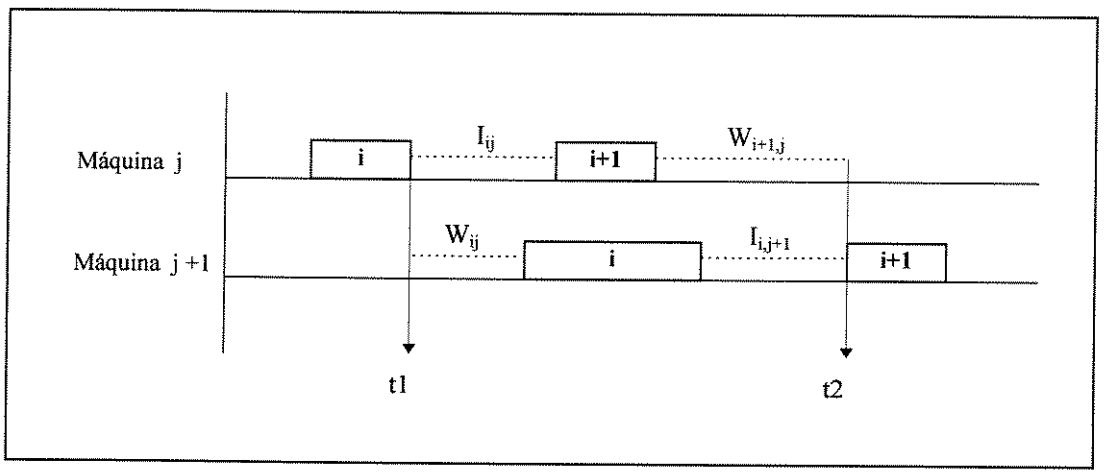


Figura 1.2. Representação gráfica do oitavo conjunto de restrições.

Embora a Figura 1.2 retrate uma possibilidade viável, espera-se que a relação descrita por $W_{i+1,j} \cdot I_{i,j+1} = 0$ seja satisfeita na solução ótima, conforme comentado por Pinedo(1995). No entanto

esta condição não é obrigatória para se obter o ótimo deste problema. Conforme Baker(1974), é suficiente considerar somente os programas semiativos para encontrar uma solução ótima para a soma dos atrasos. Um programa é semiativo se nenhuma operação pode ser terminada mais cedo sem alterar a seqüência de processamento em qualquer máquina. Esta afirmação reduz o conjunto de programas possíveis de serem considerados dada uma certa seqüência de tarefas, porém não impede que um programa não semiativo possua o mesmo valor ótimo que um programa semiativo, como ilustrado no exemplo da Figura 1.3 apresentada a seguir.

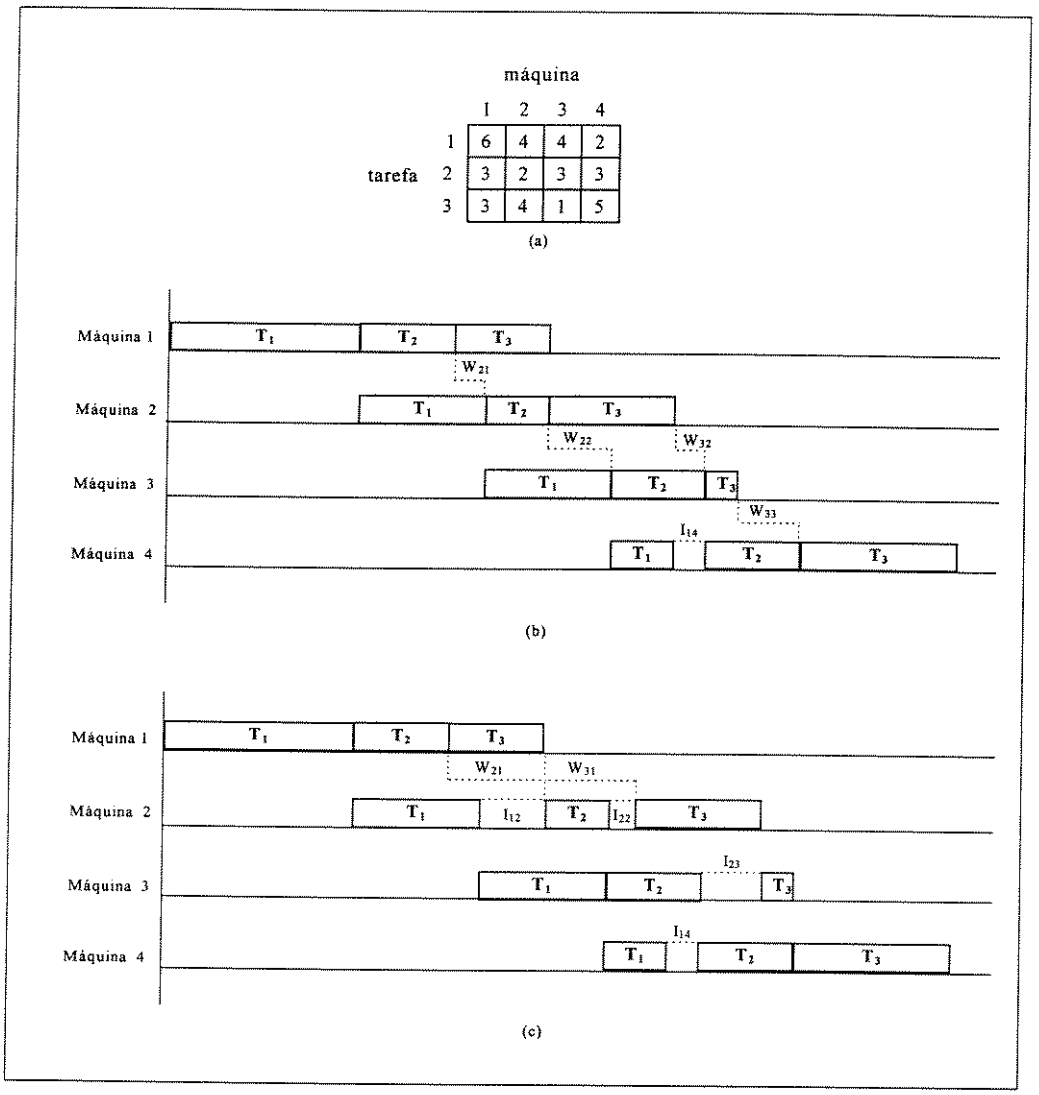


Figura 1.3. Exemplo dos efeitos da inserção de tempos adicionais em um programa semiativo. (a) Tempos de processamento. (b) Programa semiativo. (c) Programa com inserção de tempos ociosos e de espera.

Da Figura 1.3(c) pode ser observado que W_{31} e I_{22} são não nulos, e que obviamente a relação $W_{31}.I_{22} = 0$ não é satisfeita, ao contrário do que acontece na Figura 1.3(b) onde as duas variáveis são nulas. No entanto, nota-se que os instantes de término de todas as tarefas no programa (b) e no

programa (c) são os mesmos e que as tarefas terão os mesmos atrasos, e conseqüentemente os dois programas terão o mesmo valor de soma dos atrasos.

As variáveis que representam tempo ocioso e tempo de espera são contínuas e não negativas. Somente as variáveis X_{ki} são binárias. Os dois últimos conjuntos de restrições representam que na primeira máquina não há tempo ocioso e que a primeira tarefa não possui tempo de espera em nenhuma máquina. O número de restrições desta formulação é de **3n + nm - m + 1**, o número de variáveis binárias é $\mathbf{n^2}$, e o número de variáveis contínuas não negativas é **2nm - m**.

Raman(1995) apresenta uma outra formulação adaptada da formulação clássica para o critério *makespan* para *jobshop* (Baker 1974) que abrange o problema *flowshop* permutacional e não-permutacional. Apresenta-se a seguir esta formulação com algumas pequenas simplificações que limitam sua abrangência ao *flowshop* permutacional. Parâmetros básicos:

p_{ij} : tempo de processamento da tarefa i na máquina j;

d_i : data de entrega da tarefa i;

M: valor positivo muito grande.

Variáveis:

C_{ij} : instante de término da tarefa i na máquina j;

T_i : atraso da tarefa i;

X_{iz} : assume valor 1 caso a tarefa i seja processada primeiro que a tarefa z, caso contrário, assume valor 0;

Formulação:

minimizar $\sum_{i=1}^{n} T_i$ (1)

sujeito a

$T_i \geq 0$ $i = 1,..., n$ (2)

$T_i \geq C_{im} - d_i$ $i = 1,..., n$ (3)

$C_{i1} \geq p_{i1}$ $i = 1,..., n$ (4)

$C_{ij} - C_{i,j-1} \geq p_{ij}$ $i = 1,..., n;\ j = 2,..., m$ (5)

$C_{ij} - C_{zj} \geq p_{ij} - M.X_{iz}$ $i = 1,..., n;\ z = i+1,..., n;\ j = 1,..., m;$ (6)

$C_{zj} - C_{ij} \geq p_{zj} - M.(1 - X_{iz})$ $i = 1,..., n;\ z = i+1,..., n;\ j = 1,..., m;$ (7)

$C_{ij} \geq 0$ $i = 1,..., n;\ j = 1,..., m$ (8)

$$X_{iz} \in \{0,1\} \qquad i = 1,..., n;\ z = i+1,..., n; \qquad (9)$$

A função objetivo (1) representa a soma dos atrasos. Os conjuntos de restrições (2) e (3) correspondem à linearização de $T_i = \max (C_{im} - d_i , 0)$. Os conjuntos (4) e (5) indicam que uma operação não pode ser concluída antes que a tarefa precedente a esta tenha sido terminada. As restrições disjuntivas (6) e (7) garantem que não mais de uma operação é processada em cada máquina em qualquer instante de tempo, e que alguma ordem deve existir entre diferentes tarefas na mesma máquina. O conjunto de restrições (8) e (9) estabelecem, respectivamente, que as variáveis C_{ij} são contínuas e não negativas, e que as variáveis X_{iz} são binárias. O número de variáveis contínuas não negativas é $\mathbf{n + nm}$, e o número de restrições desta formulação é de $\mathbf{n^2m + n}$. O número de variáveis binárias X_{iz} é $\mathbf{(n^2 - n)/2}$, dado que X_{zi} não precisa ser definida se X_{iz} esta' na formulação. Observa-se que o número de variáveis binárias é menor que o da formulação proposta, porém o número de restrições é muito maior e de ordem quadrática em relação ao número de tarefas.

As duas formulações expostas anteriormente foram avaliadas através da resolução de alguns exemplos de dimensão reduzida utilizando-se o pacote de otimização CPLEX. Analisando-se o resultado desta pequena avaliação exposta na Tabela 1.1, observa-se que a formulação apresentada neste trabalho possui melhor desempenho, em termos de tempo computacional, que a formulação de Raman. Isto se deve ao fato de que a formulação proposta explora características do *flowshop* permutacional através das variáveis W_{ij} e I_{ij} que são conectadas pela restrição (8) desta formulação.

Tabela 1.1. Tempo em segundos para as formulações avaliadas.

Dimensão n x m	Formulação Proposta	Formulação Raman
3 x 3	0,08	0,07
6 x 5	0,37	3,80
8 x 5	11,15	310,53
10 x 7	59,42	45027,80

1.2.1. Análise de Comportamento das Soluções

Com o intuito de explorar as características do problema, realizou-se uma análise do seu comportamento em problemas com 9 tarefas e 10 máquinas. Aplicou-se enumeração completa a um conjunto de 100 matrizes de tempos de processamento com quatro diferentes cenários de datas de entrega, num total de 400 problemas. Selecionou-se esta dimensão por ser viável, em termos de tempo computacional, de se resolver através da enumeração completa. O tempo de processamento é uniformemente distribuído entre 1 e 99. Cada cenário representa uma diferente geração das datas de

entrega, que são uniformemente distribuídas entre P(1- T -R/2) e P(1 - T + R/2) (Potts & Van Wassenhove 1982), onde T e R são respectivamente o fator de atraso das tarefas, e a faixa de dispersão da data de entrega, e P é um limitante inferior do *makespan*, definido por:

$$P = \max\left\{ \max_{1\le j\le m}\left\{ \sum_{i=1}^{n} p_{ij} + \min_i \sum_{l=1}^{j-1} p_{il} + \min_i \sum_{l=j+1}^{m} p_{il} \right\}, \max_i \sum_{j=1}^{m} p_{ij} \right\}$$

onde p_{ij} é o tempo de processamento da tarefa i na máquina j. Este limitante é apresentado por Taillard(1993) baseado no trabalho de Ignall & Schrage(1965). Os cenários são obtidos variando-se T e R, conforme explicitado a seguir:

- **cenário 1:** baixo fator de atraso (T=0,2) e pequena faixa de datas de entrega (R=0,6);
- **cenário 2:** baixo fator de atraso (T=0,2) e larga faixa de datas de entrega (R=1,2) ;
- **cenário 3:** alto fator de atraso (T=0,4) e pequena faixa de datas de entrega (R=0,6);
- **cenário 4:** alto fator de atraso (T=0,4) e larga faixa de datas de entrega (R=1,2).

A Figura 1.4 ilustra os quatro cenários gerados.

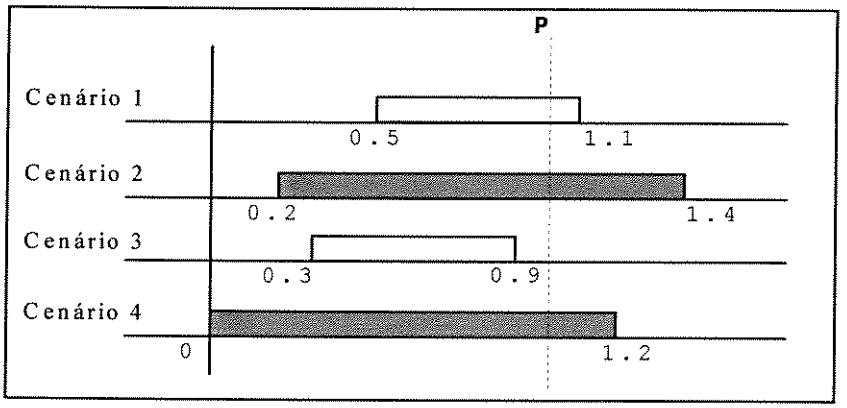


Figura 1.4. Representação dos cenários gerados.

A Figura 1.5 ilustra um dos resultados obtidos através desta análise. Para cada cenário tem-se a relação entre a porcentagem do total de soluções e o seu correspondente valor percentual acima do valor ótimo. Cada curva representa a média do comportamento de 100 problemas. Um estudo similar a este foi apresentado por Taillard(1990) para a minimização do *makespan*.

Da análise desta figura, nota-se que os cenários com fatores de atraso iguais possuem um comportamento muito similar em termos de quantidade de soluções em relação aos percentuais acima do valor ótimo. O aumento da faixa de dispersão das datas de entrega (R) ocasionou um pequeno deslocamento das curvas para esquerda, e um achatamento no ponto máximo. O deslocamento acarretou um maior número de soluções nas primeiras porcentagens acima do valor ótimo, ou seja, o aumento de R possui como conseqüência um acréscimo no número de soluções de melhor qualidade. Ressalta-se que a influência do fator de dispersão (R) é maior quando o fator de atraso (T) é menor.

Nota-se também que para os cenários 3 e 4 , que possuem fator de atraso maior, a maioria das soluções está a um valor mais próximo do valor ótimo que os cenários 1 e 2.

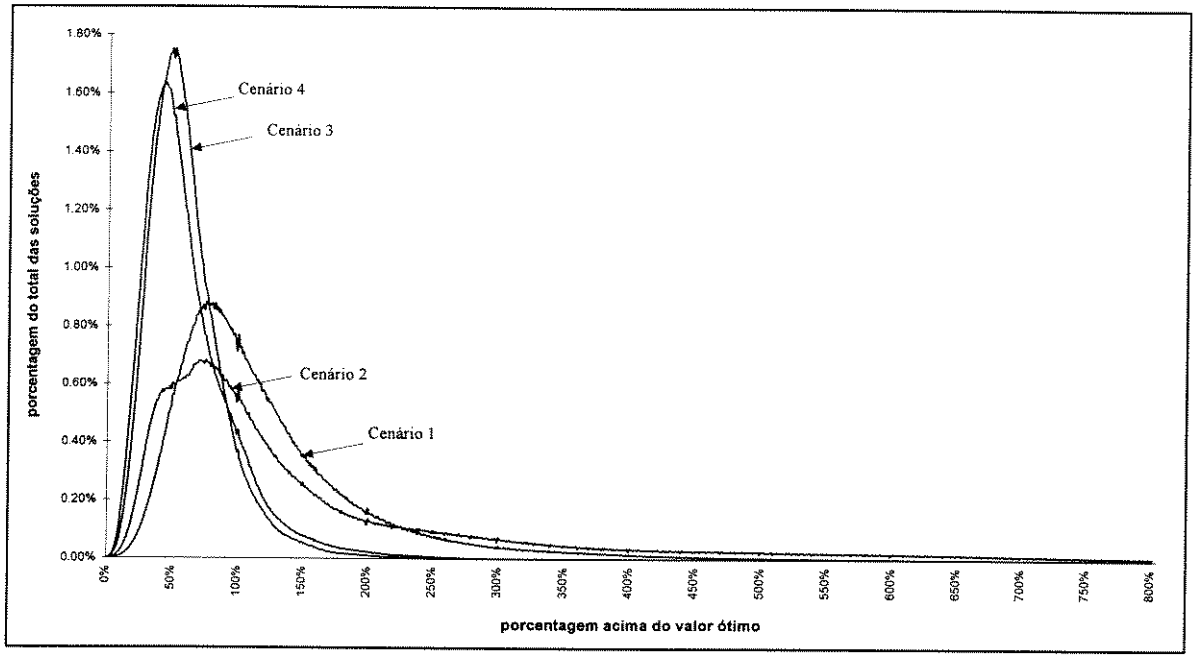


Figura 1.5. Representação gráfica dos cenários.

Analisou-se também os valores médios da porcentagem acima do valor ótimo para cada curva apresentados na Tabela 1.2. Nota-se que os valores médios dos cenários 2 e 4 são, respectivamente, maiores que os valores médios dos cenários 1 e 3, embora os primeiros cenários contenham um maior número de soluções de melhor qualidade. Este fato se justifica devido ao achatamento nos pontos máximos que causa uma menor centralização das porcentagens acima do ótimo, e conseqüentemente um aumento também no número de soluções em porcentagens muito acima do ótimo.

Tabela 1.2. Valores médios para cada cenário.

Cenário	T	R	Valor Médio %	Desvio Padrão %
1	0,2	0,6	119,30	75,84
2	0,2	1,2	146,61	143,75
3	0,4	0,6	60,33	29,06
4	0,4	1,2	61,84	35,52

Com o intuito de aprimorar a nossa análise diferenciada por cenários, elaborou-se o gráfico apresentado na Figura 1.6 que representa de forma acumulativa a porcentagem de soluções por valor percentual acima do ótimo. Através desta figura analisou-se o valor máximo acima do valor ótimo correspondente à 80% das soluções. Para o cenário 3 e 4 este percentual corresponde a aproximadamente 83%, enquanto para o cenário 1 corresponde a 169% e para o cenário 2 a 262%. Ou seja, os cenários 3 e 4 contém a grande maioria (80%) de suas soluções a 83% do ótimo, enquanto os outros dois cenários possuem valores muito mais distantes do ótimo. Se fosse aplicado uma

enumeração que avaliasse 20% do total de soluções mais 1, poder-se-ia garantir que para os cenários 3 e 4, esse método encontraria uma solução no máximo a 83% acima do ótimo. Se o mesmo fosse aplicado aos cenários 1 ou 2, somente poder-se-ia garantir uma solução respectivamente, a 169 e 262% acima do ótimo. Esta análise sugere uma indicativa acerca da complexidade de cada um dos cenários.

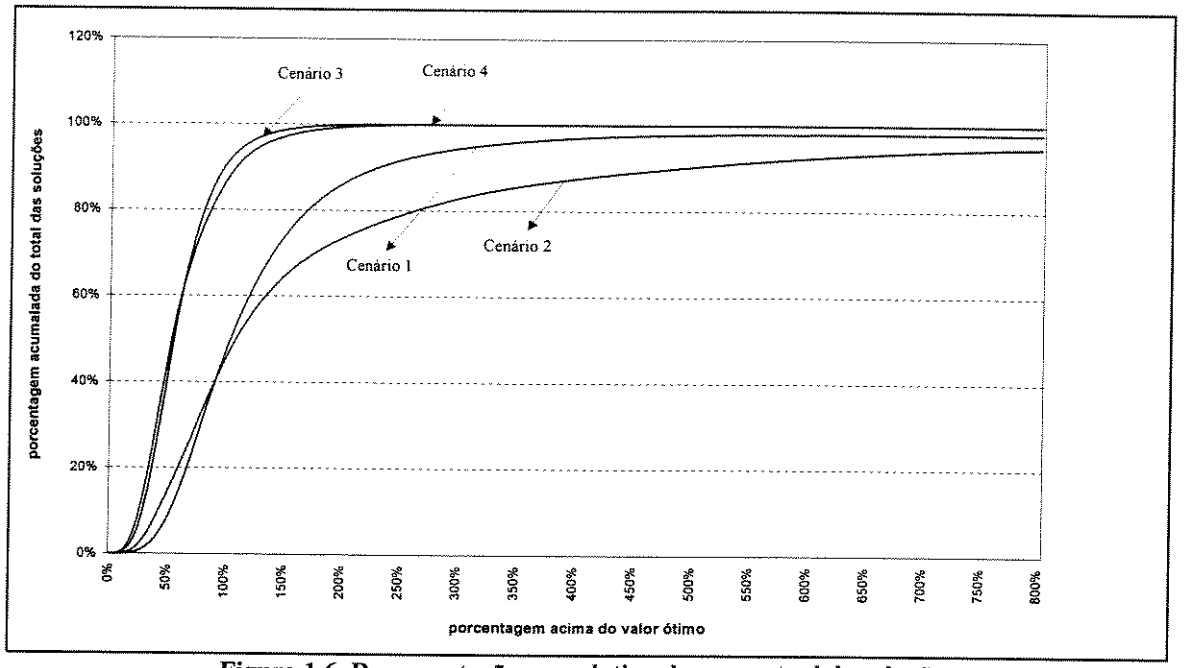


Figura 1.6. Representação cumulativa do percentual de soluções em relação a porcentagem acima do ótimo.

Para se detalhar melhor a região mais próxima do ótimo, elaborou-se o gráfico apresentado na Figura 1.7.

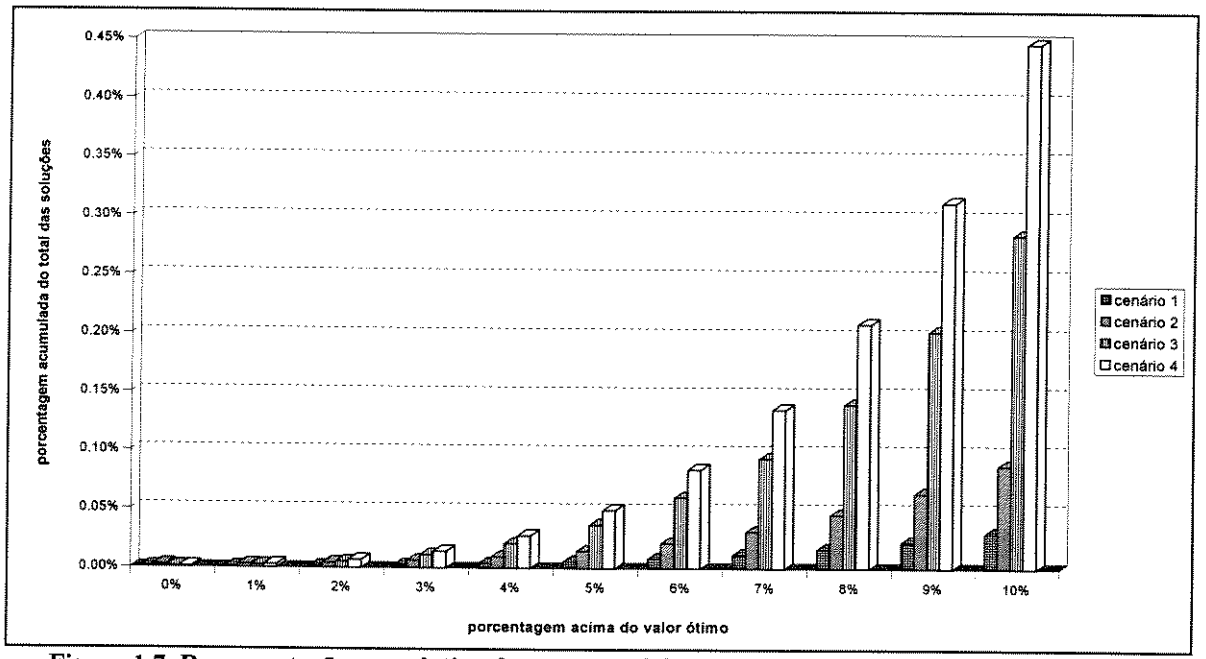


Figura 1.7. Representação cumulativa do percentual de soluções até 10% acima do valor ótimo.

Este gráfico representa a mesma relação exposta na Figura 1.6 porém, somente até 10% acima do ótimo. Esta faixa pode ser considerada como possuidora de soluções de muito boa qualidade. Observa-se claramente que nesta faixa a relação entre os cenários se mantém, ou seja, o cenário 4 é o que possui maior porcentagem de soluções a 1,2,3...,10% do ótimo, seguido pelos cenários 3, 2 e 1. Este fato reforça a idéia de ser mais fácil encontrar aleatoriamente boas soluções para os cenários 3 e 4 do que para os cenários 1 e 2. Porém, cabe destacar que a porcentagem acumulada das soluções que está a 10% do ótimo, no melhor caso, é de 0,45%. Isto indica que a probabilidade de se encontrar uma boa solução através de uma escolha aleatória é praticamente nula.

1.3. Problema de *Flowshop* com Buffer zero

Este problema consiste de **n** tarefas que devem ser processadas na mesma ordem por **m** máquinas distintas sem espaço para armazenamento de tarefas entre estas. Quando o processamento de uma tarefa é terminado numa máquina, esta última só estará liberada quando a máquina seguinte estiver livre. Caso a máquina seguinte não esteja liberada, tem-se uma situação conhecida como bloqueio. Neste trabalho considera-se somente o problema de capacidade de armazenamento zero, porque qualquer *flowshop* com capacidade de armazenamento finita não-zero pode ser modelado como um problema de *flowshop* com capacidade de armazenamento ou buffer zero. Isto ocorre porque pode-se simular a capacidade de se armazenar uma tarefa por uma máquina fictícia com tempos de processamento nulos (McCormick *et al.* 1989). O problema de programação de tarefas com capacidade limitada de buffer é um ambiente comumente encontrado na produção industrial. Com o desenvolvimento da manufatura *just-in-time* e dos sistemas kanban que mantêm um estoque limitado e fixo de tarefas em processamento, este problema assume grande significância, e necessita ser intensamente estudado (Leisten 1990, Berkley 1993, Hall & Sriskandarajah 1996, Sharadapriyadarshini & Rajendran 1996). Esta mesma recomendação é feita por Reklaitis(1982), dada a adequação deste problema ao ambiente químico. Dado que os equipamentos para processos químicos são caros, existe uma forte motivação para otimizar este ambiente de produção.

Apresentam-se a seguir as fórmulas recursivas (Pinedo 1995) para uma seqüência de tarefas $t_1, t_2, \ldots, t_i, \ldots, t_n$ onde t_i representa a tarefa na i-ésima posição da seqüência. Seja D_{ij} o instante no qual a tarefa t_i libera a máquina j. Claramente $D_{ij} \geq C_{ij}$, onde C_{ij} é o instante de término da tarefa t_i na máquina j. A igualdade ocorre quando a máquina seguinte não está bloqueada. O instante no qual a tarefa t_i inicia o seu processamento na primeira máquina é representado por D_{i0}.

$$D_{10} = 0$$

$$D_{1j} = \sum_{k=1}^{j} p_{1k} \qquad j = 1,...,m$$

$$D_{i0} = D_{i-1,1} \qquad i = 2,.....,n$$

$$D_{ij} = \max(D_{i,j-1} + p_{ij}\ ,\ D_{i-1,\,j+1}) \qquad i = 2,.....,n \qquad j = 1,...,m\text{-}1$$

$$D_{im} = D_{i,m-1} + p_{im} \qquad i = 1,.....,n$$

Ressalta-se que na recursão acima, calcula-se primeiro os instantes de liberação da primeira tarefa em todas as máquinas, a seguir da segunda tarefa e assim sucessivamente até a tarefa **n**. Das fórmulas descritas acima obtém-se o instante em que cada tarefa libera a máquina **m** os quais são iguais aos instantes de término de cada tarefa nesta máquina, isto é $D_{im} = C_{im}$, logo pode ser calculada a soma dos atrasos individuais (T_i).

$$\sum_{i=1}^{n} T_i = \sum_{i=1}^{n} \max(D_{im} - d_i\ ,0)$$

Descreve-se a seguir uma formulação inteira mista para este problema que contém os seguintes parâmetros:

p_{kj} : tempo de processamento da tarefa k na máquina j;

d_k : data de entrega da tarefa k;

e as seguintes variáveis:

X_{ki} : assume valor 1 caso a tarefa k esteja na posição i da seqüência, caso contrário, assume valor 0;

I_{ij} : tempo ocioso na máquina j entre o processamento das tarefas na posição i e i+1;

B_{ij} : tempo de bloqueio da máquina j ao término do processamento da tarefa na posição i;

D_{ij} : instante de liberação da máquina j após o processamento da tarefa na posição i;

T_i : atraso da tarefa na posição i.

Formulação:

$$\text{minimizar} \quad \sum_{i=1}^{n} T_i \qquad (1)$$

$$\text{sujeito a} \quad T_i \geq 0 \qquad i = 1,..., n \qquad (2)$$

$$T_i \geq D_{im} - \sum_{k=1}^{n} X_{ki} \cdot d_k \qquad i = 1,..., n \qquad (3)$$

$$D_{1m} = \sum_{j=1}^{m} \sum_{k=1}^{n} X_{k1} \cdot p_{kj} \qquad (4)$$

$$D_{im} = D_{i-1,m} + I_{i-1,m} + \sum_{k=1}^{n} X_{ki} \cdot p_{km} \qquad i = 2,..., n \qquad (5)$$

$$\sum_{k=1}^{n} X_{ki} = 1 \qquad i = 1,..., n \qquad (6)$$

$$\sum_{i=1}^{n} X_{ki} = 1 \qquad k = 1,..., n \qquad (7)$$

$$I_{ij} + \sum_{k=1}^{n} X_{k,i+1} \cdot p_{kj} + B_{i+1,j} - B_{i,j+1} - \sum_{k=1}^{n} X_{ki} \cdot p_{k,j+1} - I_{i,j+1} = 0 \qquad (8)$$

$$i = 1,..., n-1; j = 1,..., m-1$$

$$I_{ij} \geq 0 \qquad i = 1,..., n-1; \; j = 2,..., m \qquad (9)$$

$$B_{ij} \geq 0 \qquad i = 2,..., n; \; j = 1,..., m-1 \qquad (10)$$

$$X_{ki} \in \{0,1\} \qquad k = 1,..., n; \; i = 1,..., n \qquad (11)$$

$$I_{i1} = 0 \qquad i = 1,..., n-1 \qquad (12)$$

$$B_{1j} = 0 \qquad j = 1,..., m-1 \qquad (13)$$

$$B_{im} = 0 \qquad i = 1,..., n \qquad (14)$$

As restrições (2) a (7) são idênticas a do modelo com buffer infinito. O conjunto de restrições (8) estabelece as relações necessárias para manter a consistência entre o tempo ocioso das máquinas e o tempo de bloqueio das máquinas. Estas representam a diferença entre o instante t_1 em que a tarefa na posição **i** libera a máquina **j,** e o instante t_2 em que a tarefa na posição **i+1** inicia seu processamento na máquina **j+1,** como ilustrado na Figura 1.8. Espera-se que a relação descrita por $B_{i+1,j} \cdot I_{i,j+1} = 0$ seja satisfeita porém, como ilustrado na seção 1.2, esta condição não é obrigatória em uma solução ótima.

Os três últimos conjuntos de restrições representam que na primeira máquina não há tempo ocioso, que a primeira tarefa nunca encontra bloqueio, e que na última máquina não ocorrerá bloqueio. O número de restrições desta formulação é de **3n + nm - m + 1**, o número de variáveis binárias é $\mathbf{n^2}$, e o número de variáveis contínuas não negativas é **2nm - m + n**.

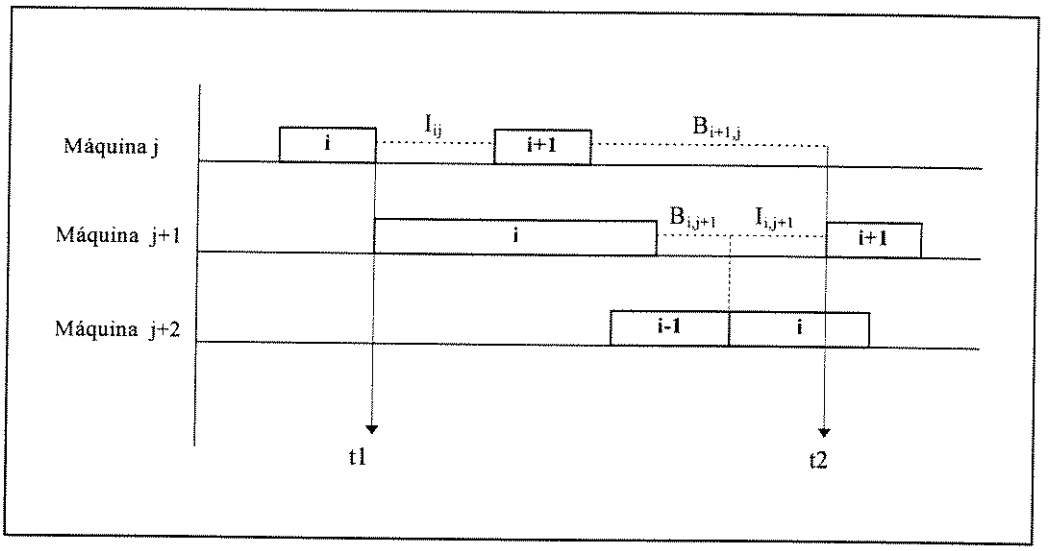


Figura 1.8. Representação gráfica do oitavo conjunto de restrições.

1.3.1. ANÁLISE DE COMPORTAMENTO DAS SOLUÇÕES

O comportamento das soluções deste problema foi analisado através de um estudo que utiliza os 400 problemas com 9 tarefas e 10 máquinas gerados na seção 1.2.1. A Figura 1.9 contém as curvas que representam, para cada cenário, a relação entre a porcentagem do total de soluções e o seu correspondente valor percentual acima do valor ótimo.

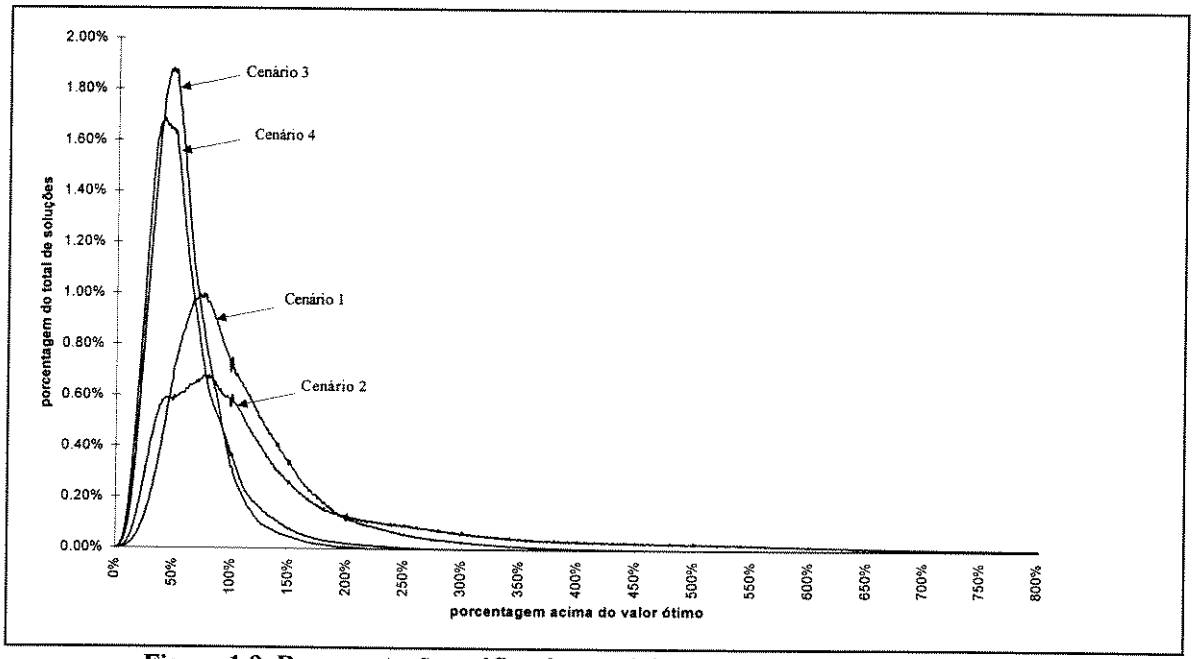


Figura 1.9. Representação gráfica dos cenários para *flowshop* com buffer zero.

Nota-se que o comportamento deste problema, sob o prisma desta análise, é muito semelhante ao comportamento observado para o problema de *flowshop* com buffer ilimitado. Para confirmar esta semelhança calculou-se na Tabela 1.3 os valores médios da porcentagem acima do valor ótimo para cada curva.

Tabela 1.3. Valores médios para cada cenário.

Cenário	T	R	Valor Médio %	Desvio Padrão %
1	0,2	0,6	105,85	64,40
2	0,2	1,2	139,43	133,53
3	0,4	0,6	57,95	27,49
4	0,4	1,2	60,21	34,63

Observa-se que os valores médios são menores que os valores obtidos para o problema de *flowshop* com buffer ilimitado, o que era previsível dado que os pontos de máximo das curvas do problema de *flowshop* com buffer zero são levemente superiores, e que estes pontos se localizam em porcentagem acima do ótimo não muito altas. Por exemplo, o ponto de máximo do cenário 3 no buffer zero possui 1,88% do total das soluções, e estas situam-se a 47% acima do valor ótimo, enquanto que para buffer ilimitado o ponto de máximo possui 1,75% das soluções que estão a 48% do ótimo. Porém cabe destacar que esta diferença é muito sutil, não modificando as principais conclusões obtidas para o primeiro problema analisado, ou seja:

- os cenários 1 e 2 possuem soluções menos centralizadas em relação ao maior valor percentual de soluções;
- os cenários 3 e 4 possuem mais soluções próximas do valor ótimo;
- o aumento do fator de dispersão (R) acarreta um aumento no número de soluções mais próximas do ótimo;
- o número de soluções a menos de 10% do valor ótimo é muito pequeno.

Logo, conclui-se que para a resolução do problema de *flowshop* com buffer ilimitado e zero deve-se utilizar um método que possua uma boa estratégia seletiva de soluções de qualidade.

Uma comparação dos valores das soluções ótimas para os 400 problemas gerados revelou que o atraso para o *flowshop* com buffer zero é em média 7,98% maior que o atraso para o *flowshop* com buffer ilimitado. Este fato é coerente com a possibilidade de bloqueio existente no problema com buffer zero.

1.4. REVISÃO BIBLIOGRÁFICA

- ***Flowshop* com Buffer Ilimitado**

O problema de encontrar um programa para o ambiente *flowshop* de permutação utilizando o critério de minimização do *makespan* foi abordado por diversos autores através de métodos ótimos (Johnson 1954, Ignall & Schrage 1965, Lomnicki 1965, McMahon & Burton 1967, Carlier & Rebai 1996), e métodos heurísticos (Nawaz *et al.* 1983, Widmer & Hertz 1989, Osman & Potts 1989, Taillard 1990, Ogbu & Smith 1990, Werner 1993, Reeves 1993,1995, Moccellin 1995, Nowicki & Smutnicki 1996a). A motivação para o desenvolvimento de heurísticas se deve à complexidade computacional deste problema, que para mais de duas máquinas é classificado como *NP-hard* (Garey *et al.* 1976). Este problema com duas máquinas é resolvido em tempo polinomial pelo algoritmo de Johnson(1954).

A metaheurística Busca Tabu tem sido utilizada com sucesso no problema de *flowshop* com o critério de minimização do *makespan*. A primeira proposta com este enfoque foi apresentada por Widmer & Hertz(1989) que apresentam um método que consiste basicamente de dois passos: obtenção da solução inicial através de uma analogia com o problema do caixeiro viajante, e a melhoria desta solução com a utilização de Busca Tabu. O primeiro passo desta heurística foi melhorado por Moccellin(1995) através da proposta de uma nova medida de distância entre tarefas baseada no cálculo do limitante superior do tempo ocioso da última máquina entre o processamento de duas tarefas adjacentes. Um trabalho interessante, foi realizado por Taillard(1990), destacando-se o extensivo estudo para a definição dos componentes da busca. O autor compara o resultado do seu algoritmo com o obtido através de uma implementação computacional eficiente do algoritmo NEH proposto por Nawaz *et al.*(1983), e conclui que a Busca Tabu obtém melhores resultados. Reeves(1993) destaca em seu trabalho o balanceamento que deve existir entre a diversificação da busca no espaço de soluções e a exploração mais detalhada em certas regiões. Este balanceamento é atingido através da divisão aleatória da vizinhança em subpartes, e nesta vizinhança menor são permitidas mais iterações. O desempenho deste algoritmo superou o proposto por Taillard em problemas sugeridos pelo próprio Taillard(1993) com $20 \leq n \leq 500$ e $5 \leq m \leq 20$.

Mais recentemente, Nowicki & Smutnicki(1996a) propuseram uma heurística baseada em Busca Tabu que utiliza a idéia de blocos de tarefas em *flowshop* proposta por Grabowski *et al.*(1983). Um bloco é definido a partir do caminho crítico, que é o caminho mais longo em um grafo direcionado que representa o problema de *flowshop*. O caminho crítico determina o instante de término de todas as

tarefas (*makespan*). Este grafo é construído com base nas restrições impostas pelas relações de precedência entre as tarefas nas máquinas, e entre as operações nas tarefas. Um bloco de tarefas pode ser definido como um conjunto contendo k tarefas, k>1, que são processadas consecutivamente em cada máquina dentro do caminho crítico. Nowicki & Smutnicki exploram propriedades específicas destes blocos para restringir a busca no espaço de soluções. Os autores comparam sua heurística com a proposta por Taillard, e obtém resultados melhores num tempo menor sobre o conjunto de problemas sugeridos por Taillard(1993). Embora não sejam apresentadas comparações com o trabalho apresentado por Reeves, acredita-se que este algoritmo por explorar mais detalhadamente as características específicas do problema, possua um melhor desempenho. Outras metaheurísticas foram aplicadas a este problema com critério de minimizar o *makespan* como *Simulated Annealing* (Osman & Potts 1989, Ogbu & Smith 1990) e Algoritmos Genéticos (Reeves 1995), porém o algoritmo proposto por Nowicki & Smutnicki apresenta os melhores resultados.

A literatura envolvendo o critério de minimização da soma dos atrasos não é muito vasta. O nível de dificuldade do problema de *flowshop* com este critério pode ser estimado através de dois casos básicos: minimizar a soma dos atrasos em uma máquina foi estabelecido por Du *et al.*(1990) como NP-*hard*, e minimizar a soma dos atrasos no ambiente *flowshop* com duas máquinas foi estabelecido por Lenstra *et al.*(1977) como NP-*hard*, e por Koulomas(1994) como NP-*hard in the strong sense*. Boas revisões bibliográficas para este ambiente podem ser encontradas em Koulamas(1994) e Kim (1993b). Descrevem-se a seguir os principais autores que trataram deste problema envolvendo atraso.

Buscando encontrar soluções ótimas Towsend(1977), Sen *et al.*(1989), e Kim(1993a,1995) desenvolveram algoritmos baseados na técnica de *Branch-and-Bound*. Townsend(1977) desenvolveu um algoritmo *Branch-and-Bound* baseado nos resultados de Smith(1956) para o problema de uma máquina. Smith demonstrou que as tarefas devem ser ordenadas em ordem não-crescente de suas datas de entrega para obter o programa ótimo para o critério do atraso máximo. Towsend ampliou este resultado para **m** máquinas, desenvolvendo um limitante inferior para este problema. Sen *et al.* aplicaram a técnica *Branch-and-Bound* ao problema de *flowshop* com duas máquinas com o objetivo de minimizar o atraso médio. Foi proposto um teorema baseado na troca de pares adjacentes, que é utilizado para avaliar a validade de uma possível troca para a redução do atraso total. Além disso, elaborou-se um limitante inferior para o problema. Kim(1993a) aborda o mesmo problema que Sen *et al.* e propõe um *Branch-and-Bound* onde cada nó da árvore corresponde a um subproblema que é definido por um conjunto de tarefas que são posicionadas no fim da seqüência. Denomina-se este conjunto de seqüência parcial. Dois limitantes independentes são calculados para este nó, um para a seqüência parcial e outro para o conjunto

das tarefas que não estão incluídas nesta seqüência. A soma destes é utilizada como limitante inferior para a soma dos atrasos. Kim também utiliza o mesmo teorema desenvolvido por Sen *et al.*, além de uma regra de dominância. Esta regra se baseia no fato de que as tarefas que possuem data de entrega superior a um limitante superior do *makespan*, certamente estarão no fim da seqüência ótima. Testes computacionais em problemas gerados aleatoriamente com número de tarefas variando de 10 a 15, mostraram que este algoritmo supera o algoritmo de Sen *et al.*. Kim(1995) estendeu os limitantes inferiores desenvolvidos para este problema para a resolução do problema de *flowshop* com **m** máquinas. Foram realizados testes em problemas com $10 \leq n \leq 20$ e $5 \leq m \leq 25$. Aparentemente não foram testadas todas as possíveis combinações de quantidades de tarefas e máquinas devido ao esforço computacional.

Gelders & Sambadam(1978) e Ow(1985) apresentam heurísticas baseadas em cálculos de prioridades entre tarefas com o objetivo de minimizar a soma ponderada dos atrasos. No primeiro trabalho são propostas quatro heurísticas, onde a variação entre estas está na maneira de avaliar as prioridades. As maiores prioridades são atribuídas às tarefas mais custosas de serem seqüênciadas mais tarde. Ow desenvolveu um procedimento para um caso especial, conhecido como *flowshop* proporcional, onde uma constante de proporcionalidade $\mathbf{c_j}$ é associada à cada máquina. Cada tarefa com tempo de referência **p** possui o seguinte conjunto de tempos de processamento nas máquinas: **p**, $\mathbf{p.c_2}$, $\mathbf{p.c_3}$, $\mathbf{p.c_4}$ ... $\mathbf{p.c_m}$. O método trata as conseqüências na máquina com a maior soma dos tempos de processamento, denominada máquina gargalo. A esta aplicam uma regra de prioridades, conhecida como regra do tempo ocioso, que visa ordenar as tarefas de forma a minimizar a soma dos atrasos ponderados nesta máquina. Raman(1995) avalia várias heurísticas para o problema de *flowshop* considerando permutação e não-permutação, dentre estas uma adaptação da heurística de Ow. Nesta adaptação proposta por Raman cada máquina do *flowshop* é considerada como sendo a máquina gargalo, e um programa para todo o ambiente é obtido através da regra do tempo ocioso. Dentre todos os **m** programas gerados desta forma, seleciona-se o que possui menor soma dos atrasos. Este método obteve os melhores resultados dentre as heurísticas que consideram programas de permutação.

Kim(1993b) compara diversas adaptações de heurísticas normalmente utilizadas para o critério *makespan*, para minimizar a soma dos atrasos, inclusive o algoritmo NEH. Inicialmente, este método ordena todas as tarefas de acordo com algum critério preestabelecido, de forma a criar uma lista de prioridades entre estas. As duas tarefas com a maior prioridade são selecionadas da lista, e as duas seqüências possíveis para estas são geradas. Estas são avaliadas de acordo com o critério de otimalidade e a melhor seqüência parcial é escolhida. As posições relativas destas duas tarefas não se modificam durante todo algoritmo. A seguir, a tarefa com a terceira maior prioridade é selecionada e três seqüências

são geradas ao se posicionar esta tarefa no início, no meio, e no fim da seqüência gerada no passo anterior. A melhor seqüência parcial encontrada irá fixar as posições relativas destas três tarefas no resto do algoritmo. Este processo é repetido até que todas as tarefas sejam fixadas e a seqüência final encontrada. Kim recomenda a utilização do algoritmo NEH associado à regra EDD (ordem não-crescente de datas de entrega), dado que este forneceu bons resultados num tempo viável. Neste mesmo trabalho avalia-se a adaptação da Busca Tabu proposta por Widmer & Hertz(1989). Esta adaptação apresenta em média os melhores resultados, porém com um tempo computacional alto e com resultados inferiores a outras heurísticas, inclusive ao algoritmo NEH em 25% dos problemas testados. Estas comparações são realizadas em um conjunto de problemas gerados aleatoriamente com $15 \leq n \leq 50$ e $5 \leq m \leq 25$.

Outra aplicação de Busca Tabu é apresentada por Adenso_Díaz(1992) que trata o problema do *flowshop* proporcional para minimizar a soma dos atrasos. Este autor utiliza como solução inicial a heurística proposta por Ow, e tem como maior contribuição a redução de duas vizinhanças geradas por movimentos normalmente utilizados: o de inserção e o de troca de tarefas. Mais recentemente, este mesmo autor propôs uma melhoria da solução inicial através da aplicação da metaheurística *Simulated Annealing* (Adenso_Díaz(1996)). Esta nova proposta diminui o número necessário de iterações para que a Busca Tabu alcance o mesmo nível de qualidade de soluções obtida com a solução inicial dada pela regra proposta por Ow. Estes testes foram realizados em problemas com $10 \leq n \leq 50$ e $5 \leq m \leq 25$.

Uma proposta que considera o problema de *flowshop* não-permutacional para a minimização da soma dos atrasos é apresentada por Bornstein(1989). O algoritmo é constituído de quatro etapas. Primeiramente, determinam-se aquelas tarefas que poderão ser as últimas da sequência. A seguir ordena-se as tarefas restantes transformando o problema de m máquinas em m problemas de uma máquina. A terceira etapa realiza a ordenação das tarefas na primeira máquina. Na última etapa ordenam-se as tarefas nas máquinas restantes. Bornstein compara seu algoritmo com o desenvolvido por Gupta(1971) que utiliza uma enumeração associada com regras de dominância para obter uma solução ótima considerando-se somente os programas de permutação. A comparação entre os dois algoritmos é realizada em problemas pequenos com a dimensão variando quase que exclusivamente entre 7, 8 e 9 tarefas, e o número de máquinas entre 5 e 10. Em geral, o algoritmo proposto por Bornstein apresenta resultados inferiores aos obtidos por Gupta, porém, como era esperado, apresenta tempos computacionais muito menores, o que viabiliza a resolução de problemas de maior dimensão tais como 20 tarefas e 5 máquinas.

Uma alternativa possível para solucionar este problema é a sua abordagem como um Problema

de Otimização Restrito, conhecido na língua inglesa como *Constrained Optimization Problem* (COP). Uma aplicação desta técnica ao problema de programação de tarefas no ambiente *jobshop* encontra-se em Sadeh (1991). Este autor utiliza técnicas que constroem um programa de tarefas através de um processo de busca dirigido pela detecção das decisões mais críticas e pelos valores mais promissores para estas decisões.

- ***Flowshop* com Buffer Zero**

Em contraste com o problema de *flowshop* com buffer ilimitado, o *flowshop* com buffer zero entre as máquinas foi pouco explorado na literatura. Na maioria dos trabalhos aborda-se este problema com o objetivo de minimizar o *makespan* e em ambientes especiais. Uma boa revisão bibliográfica pode ser encontrada em Hall & Sriskandarajah(1996), onde além de se tratar do problema de *flowshop* com buffer zero, revisa-se também a literatura do *flowshop* sem espera entre as máquinas (*no-wait*). Estes autores estudam a complexidade deste problema para o critério de minimização do *makespan*, e mostram que o problema de *flowshop* com buffer zero para três máquinas é NP-*hard in the strong sense*. Para duas máquinas, Reddi & Ramamoorthy(1972) mostraram que este problema pode ser reduzido a um caso especial do problema do caixeiro viajante, resolvido otimamente em tempo polinomial por Gilmore & Gomory(1964). O problema de *flowshop* com buffer limitado considerando a minimização da soma dos atrasos aparentemente não foi tratado na literatura. Por este motivo descreve-se a seguir os principais autores que trabalharam neste ambiente para minimizar o *makespan*.

Dutta & Cunningham(1975) utilizam programação dinâmica para obter soluções ótimas para o problema de buffer finito com duas máquinas. Este algoritmo requer um armazenamento excessivo para problemas com mais de 18 tarefas. Baseado neste trabalho, Reddi(1976) desenvolveu um método ótimo alternativo que combina programação dinâmica com o algoritmo de Johnson(1954). Este método possui melhor desempenho quanto menor for a expressão n - número de buffers - 2. Papadimitriou & Kanellakis(1980) mostraram que o problema de *flowshop* com duas máquinas e buffer finito não-zero é NP-*hard*, e descrevem uma heurística para o caso onde o buffer possui capacidade de armazenamento para uma tarefa. A heurística inicialmente encontra uma seqüência ótima para buffer *no-wait* através do algoritmo de Gilmore & Gomory(1964), e então utiliza esta seqüência para realizar a programação das tarefas com o espaço no buffer limitado a uma tarefa. O valor da solução fornecida por esta heurística situa-se no pior caso a 50% do valor da solução ótima. Berkley(1993) apresenta um trabalho que trata do problema de *flowshop* com buffer finito não-zero. Este autor descreve as características necessárias dos tempos de processamento e da capacidade do

buffer para que se realize a melhor seleção entre a regra SPT (*shortest processing time*) e FCFS (*first come-first served*).

Levner(1969) e Suhami & Mah(1981) apresentam algoritmos *Branch-and-Bound* que tratam especificamente do *flowshop* com buffer zero. Levner, inicialmente, demonstra que obter o *makespan* de uma permutação é equivalente a encontrar o caminho crítico em um grafo direcionado. Este grafo é construído de forma similar ao grafo associado ao buffer ilimitado, além de considerar as restrições impostas pela ausência de buffer. Baseado no grafo, o autor sugere dois limitantes inferiores e recomenda que em cada estágio do algoritmo se eleja o limitante de maior valor. Resultados computacionais não são apresentados, embora o autor garanta encontrar a solução ótima em problemas de até 15 tarefas. Suhami & Mah propõem um *Branch-and-Bound* que utiliza limitantes baseados em estimativas de média e variância de *makespans* derivados de seqüências parciais já fixadas. Para problemas com três máquinas, seis e dez tarefas, o algoritmo proposto usualmente encontra soluções dentro de 1% acima do valor de uma solução ótima. Uma outra proposta que também trata do *flowshop* com buffer zero é apresentada por McCormick *et al.*(1989). Este autor propôs uma heurística conhecida por *profile fitting* (PF) onde procura-se seqüenciar primeiro as tarefas que acarretam uma menor soma dos tempos de ociosidade e de bloqueio nas máquinas.

Uma abordagem mais ampla é apresentada por Leisten(1990) que propôs duas heurísticas que tratam o problema de *flowshop* de permutação e de não-permutação com buffer finito, ilimitado, e *no-wait*. O principal objetivo das heurísticas propostas é maximizar o uso dos buffers e minimizar o bloqueio das máquinas. Leisten compara estas heurísticas com outras conhecidas da literatura para *flowshop* com capacidade de armazenamento ilimitada e *no-wait* e conclui que a heurística proposta por Nawaz *et al.*(1983) possui um desempenho superior. Esta comparação é efetuada em problemas com duas ou três máquinas, e com quantidade de tarefas variando de 10 a 50.

Novas propostas para o problema com buffer finito podem ser encontradas em Sharadapriyadarshini & Rajendran(1996). Os autores propõem novas fórmulas recursivas que consideram tempos de transporte, tempos de preparação das máquinas, e espaços diferentes no buffer para armazenar tarefas.

A proposta deste trabalho é sugerir métodos de resolução para o problema de *flowshop* com buffer ilimitado e zero com o critério de minimização da soma dos atrasos. Inicialmente, são propostas heurísticas baseadas na metaheurística Busca Tabu. Escolheu-se esta metaheurística por seu excelente desempenho no problema de *flowshop* com buffer ilimitado e critério de minimização do *makespan*. Além das heurísticas, serão apresentados métodos ótimos que utilizam a técnica *Branch-and-Bound*, e um limitante inferior para a soma dos atrasos para o problema de *flowshop* com buffer zero.

Capítulo 2
Heurísticas Propostas
Versões Básicas

2.1. Introdução

O objetivo deste capítulo é apresentar as heurísticas propostas para os problemas de *flowshop* com buffer zero e ilimitado com objetivo de minimizar a soma dos atrasos. Na seção a seguir são sucintamente descritas as principais características da técnica Busca Tabu. A seção 2.3 contém a descrição dos componentes das heurísticas nas suas versões básicas. Os resultados da aplicação da Busca Tabu básica ao problema de *flowshop* com buffer ilimitado são apresentados no trabalho de Ronconi & Armentano(1995).

2.2. Busca Tabu

A maior motivação para o desenvolvimento de métodos heurísticos em problemas de otimização combinatória é a grande dificuldade encontrada em resolver estes problemas. Heurísticas que exploram características dos problemas podem gerar soluções de boa qualidade em tempo rápido. Isto é fundamental para decisões de prazo muito curto como aquelas associadas com a programação da produção. Uma heurística popular com largas aplicações é a busca local ou busca em vizinhança. A idéia é muito simples: inicia-se de uma solução qualquer, e busca-se em uma certa vizinhança a melhor solução possível. A partir desta, o processo de busca é reiniciado e prossegue até que nenhuma melhoria seja obtida. A solução final é chamada ótimo local, dado que esta é a melhor solução dentre toda uma vizinhança. A desvantagem evidente deste método é que este ótimo local na maioria das vezes não é o ótimo global. Buscando evitar que a heurística estacione nestes ótimos locais surgem as

metaheurísticas que trabalham como guias da busca local direcionando seus movimentos para a exploração de diferentes regiões do espaço de soluções.

Busca Tabu é uma metaheurística proposta por Glover(1989, 1990) que tem apresentado excelentes resultados na resolução de problemas de programação de tarefas, como por exemplo no ambiente com uma máquina (Laguna, *et al.* 1991), máquinas paralelas (Laguna & Velarde 1991, Barnes & Laguna 1993, Hübscher & Glover 1994, França *et al.* 1995, Yamashita 1996), *jobshop* (Dell´ Amico & Trubian 1993, Taillard 1994, Barnes & Chambers 1995, Nowicki & Smutnicki 1996b), e *flowshop* (Widmer & Hertz 1989, Taillard 1990, Reeves 1993, Moccellin 1995, Nowicki & Smutnicki 1996a). Os conceitos básicos e avançados desta técnica, e algumas das aplicações desenvolvidas nos últimos anos podem ser encontrados nos trabalhos de Glover(1989, 1990, 1995, 1996), de Werra & Hertz(1989), Glover & Laguna(1993), e Laguna(1995).

Esta metaheurística inicia seu procedimento da mesma forma que uma busca local, prosseguindo iterativamente de uma solução para outra, através de uma operação denominada movimento, até que um determinado critério de parada seja satisfeito. Porém, diferente da busca local que só aceita soluções que melhorem a solução atual, a Busca Tabu pode aceitar soluções que não sejam melhores que a solução atual, escapando desta forma de ótimos locais. Busca Tabu é baseada na premissa que a resolução de um problema, para ser classificada como inteligente, deve incorporar uma memória adaptativa e boas estratégias de busca, ou seja, com a utilização das informações da história da busca deve ser capaz de realizar escolhas estratégicas ao longo de sua trajetória. A memória adaptativa pode ser de curto ou de longo prazo. O princípio básico da utilização da memória de curto prazo é o de restringir a busca através da proibição de certos movimentos dirigindo-a para regiões não exploradas. O uso mais comum desta memória é para armazenar informações associadas às características das soluções recentemente visitadas. Seja um problema de otimização do tipo:

$$\begin{aligned} &\textit{Minimizar} && f(x) \\ &\textit{sujeito a} && x \in X \end{aligned}$$

onde f(x) é qualquer função da variável discreta **x**, e **X** é o conjunto de soluções factíveis. Cada $\mathbf{x} \in \mathbf{X}$ possui uma **vizinhança** associada $\mathbf{V(x)} \subset \mathbf{X}$, e cada solução $\mathbf{x`} \in \mathbf{V(x)}$ é alcançada de x através de uma operação chamada **movimento.** Normalmente em Busca Tabu, as vizinhanças utilizadas são simétricas, isto é, x´ é um vizinho de x se e somente se x é vizinho de x´. A cada movimento realizado, armazena-se na memória de curto prazo uma **regra de proibição** relativa a um **atributo** da solução gerada. As regras de proibição evitam que alguns movimentos sejam realizados. Armazenam-

se estas regras por um certo número de iterações conhecido como **duração tabu** que pode ser um valor fixo ou dinâmico. Um movimento classificado como tabu pode ser executado caso seja considerado atrativo de acordo com o **critério de aspiração**. Existem vários possíveis critérios de aspiração, e o mais utilizado é a melhoria da função objetivo, ou seja, se o movimento tabu resultar em uma solução com valor superior ao valor da melhor solução encontrada até o momento. O **critério de parada** deste método pode ter várias formas, tais como um certo número de iterações sem melhoria da busca, um número fixo de iterações, ou um determinado tempo de execução do algoritmo. Este critério é estabelecido de acordo com o problema e com o objetivo de sua aplicação.

2.3. Descrição da Versão Básica das Heurísticas

As heurísticas propostas são compostas por uma busca em vizinhança orientada por Busca Tabu. Denominou-se como versão básica das heurísticas a utilização de Busca Tabu somente com a memória de curto prazo. Para tal se torna necessária a definição da solução inicial a ser utilizada e dos componentes básicos da busca. Dentre estes destacam-se: estratégia de geração de vizinhança, escolha da próxima solução vizinha, atributos, regras de proibição, duração tabu, critério de aspiração, e critério de parada. Descreve-se a seguir os testes realizados na seleção destes elementos para as heurísticas.

Os programas computacionais foram escritos na linguagem C e os testes realizados numa estação de trabalho SUN SPARCstation classic, com 50Mhz de *clock*, 48Mbytes de RAM, e sistema operacional SUN OS 4.1.3. Os dados foram gerados conforme sugerido em Taillard(1993), de tal forma a torná-los independentes do computador utilizado. Para cada uma das dimensões do problema (**n x m**) gerou-se diferentes matrizes de tempos de processamento, e para cada uma destas foram selecionados quatro distintos cenários. Denominou-se classe cada combinação de dimensão e cenário. O tempo de processamento é inteiro e uniformemente distribuído entre 1 e 99. As datas de entrega e os cenários foram gerados como na seção 1.2.1.

2.3.1. Solução Inicial

Avaliou-se para solução inicial adaptações de regras de despacho e do algoritmo NEH, considerado o método mais eficiente para o critério do atraso médio para buffer ilimitado(Kim 1993b),

e para o critério do *makespan* para buffer zero (Leisten 1990). Apresenta-se a seguir uma descrição do algoritmo NEH adaptado (NEHad) para o critério de minimização da soma dos atrasos das tarefa:

Passo 1: Para cada tarefa i atribua uma prioridade R_i;

Passo 2: Construa uma lista de tarefas em ordem decrescente de prioridades;

Passo 3: Selecione as duas primeiras tarefas da lista do passo 2, e encontre a melhor seqüência possível para esta duas tarefas de acordo com o critério de minimização da soma dos atrasos. Faça i=3;

Passo 4: Selecione a tarefa i da lista, e insira i em todas as posições possíveis da seqüência atual. Escolha a melhor seqüência dentre as i geradas.

Passo 5: Se i=n, pare, senão i= i+1 e volte para o passo 4.

O número de enumerações deste algoritmo é **n(n+1)/2 -1**, das quais **n** são completas e o resto são enumerações parciais. Conseqüentemente, existirão **n(n+1)/2 -1** avaliações a serem realizadas. Para diminuir o tempo computacional desta etapa, utilizou-se no passo 4 uma ordem específica de posicionamento da tarefa **i** na seqüência original. Inicialmente posiciona-se a tarefa **i** no final da seqüência já existente, pois desta forma somente o cálculo do atraso desta tarefa precisa ser realizado, dado que o instante de término das outras tarefas não se modificou. A localização seguinte de **i** é a posição **i-1**, onde somente devem ser recalculados os atrasos das tarefas da posição **i** e **i-1**. Este processo se repete até que tarefa **i** atinja a primeira posição da seqüência, onde todos os atrasos são recalculados. Desta forma, o cálculo da soma dos atrasos de cada uma das **n(n+1)/2 -1** seqüências é reduzido.

O algoritmo NEH original proposto por Nawaz *et al.*(1983) ordena, inicialmente, as tarefas em ordem não-crescente da soma de seus tempos de processamento nas máquinas. Esta escolha é coerente com o critério de minimização do *makespan*. Dado que o algoritmo NEH adaptado para a minimização do critério da soma dos atrasos (NEHad) também necessita de uma regra que ordene inicialmente as tarefas, optou-se pela utilização das mesmas regras de despachos que são avaliadas como solução inicial para Busca Tabu. Estas regras foram selecionadas por serem as mais simples ou conhecidas por trabalharem melhor que outras (Kim 1993b). Cada regra se baseia numa medida diferente que é calculada, se necessário, a cada instante de tempo **t** no qual as tarefas são consideradas para seleção. Ordenam-se as tarefas em ordem não-decrescente destas medidas. As seguintes regras foram selecionadas:

- **EDD** : d_i,
- **SLACK**: $d_i - C_i(\sigma)$,

- **MDD**: $\max(d_i , C_i(\sigma))$,

onde,

$\mathbf{d_i}$: data de entrega da tarefa **i**;

σ : seqüência de tarefas já estabelecida no instante $\mathbf{t} = \sum_{i \in \sigma} p_{il}$;

$\mathbf{C_i(\sigma)}$: instante de finalização da tarefa **i**, $i \notin \sigma$, se esta for seqüênciada logo após a seqüência σ.

Além destas, sugere-se aqui a utilização de um limitante inferior do atraso de cada tarefa (LI) como regra para ordenação das tarefas. Este limitante agrega mais informações que a regra EDD, mantendo a simplicidade do cálculo.

- **LI**: $d_i - \sum_j p_{ij}$

onde, p_{ij} é o tempo de processamento da tarefa **i** na máquina **j**. Observe que as regras de despacho LI e EDD são estáticas, pois determinam toda a seqüência no instante zero, enquanto que as regras MDD e SLACK são dinâmicas dado que a seleção das tarefas é realizada ao longo do tempo.

- **Problema de Flowshop com buffer ilimitado**

Nesta etapa de testes analisa-se o desempenho de regras de despacho isoladamente e em conjunto com o algoritmo NEHad nos problemas cujas dimensões são apresentadas na Tabela 2.1. Para cada dimensão foram geradas dez diferentes matrizes de tempos de processamento. Além disso, variou-se o cenário como descrito na seção 1.2.1., ou seja, foram testados 10 x 10 x 4 = 400 problemas.

A Tabela 2.1 mostra o resultado desta primeira fase das implementações. Devido a dificuldade de obter soluções ótimas para estes problemas, utilizou-se como medida de desempenho o índice de desvio relativo (Kim 1993b). Este índice (IDR) é calculado da seguinte forma:

$$IDR = \frac{T_a - T_{menor}}{T_{maior} - T_{menor}}$$

onde $\mathbf{T_a}$ é a solução do método que está sendo avaliado, e $\mathbf{T_{menor}}$ e $\mathbf{T_{maior}}$ são respectivamente a melhor e a pior solução encontradas. A faixa de variação deste índice é de 0 a 1. Este índice é extremamente útil para a avaliação entre diferentes métodos pois posiciona cada método entre a menor e a maior solução dentre todas avaliadas.

Tabela 2.1.Comparação entre regras de despacho e o algoritmo NEHad.

n	m	EDD	LI	MDD	SLACK	NEHad (EDD)	NEHad (LI)	NEHad (MDD)	NEHad (SLACK)
20	5	0,584	0,897	0,383	0,748	0,069	0,044*	0,077	0,077
20	10	0,712	0,937	0,432	0,818	0,051	0,022*	0,080	0,043
20	20	0,805	0,964	0,507	0,896	0,036*	0,037	0,064	0,049
50	5	0,606	0,787	0,402	0,660	0,057	0,054*	0,089	0,064
50	10	0,755	0,862	0,607	0,825	0,062	0,046*	0,082	0,069
50	20	0,861	0,932	0,662	0,894	0,064	0,027*	0,106	0,054
100	5	0,529	0,654	0,391	0,598	0,050	0,025*	0,046	0,042
100	10	0,725	0,788	0,578	0,732	0,053	0,024*	0,070	0,051
100	20	0,797	0,921	0,631	0,859	0,070	0,036*	0,071	0,063
200	10	0,603	0,671	0,490	0,600	0,050	0,033*	0,048	0,037
MÉDIA		0,698	0,841	0,508	0,763	0,056	0,035*	0,073	0,055
DESVIO PADRÃO		0,111	0,112	0,100	0,117	0,001	0,001	0,003	0,002

Os asteriscos na Tabela 2.1 indicam as melhores soluções. Observa-se através destes e da média dos índices que o algoritmo NEHad associado à regra LI obteve os melhores resultados.

Com intuito de confirmar o bom comportamento da combinação do algoritmo NEHad com a regra LI, esta foi aplicada a 960 problemas gerados conforme sugerido por Raman(1995). Esta geração possui dois parâmetros que controlam a geração das datas de entrega: o fator de atraso (**Z**), e a faixa de geração (**R**). As datas são geradas com distribuição uniforme dentro do intervalo $[\bar{d}(1 - R/2), \bar{d}(1 + R/2)]$ onde $\bar{d} = C_{max}(1 - Z)$, e $\mathbf{C_{max}}$ é o *makespan* da seqüência obtida programando as tarefas na ordem de seus índices. Foram considerados quatro valores para **Z**: 0,05, 0.25, 0,50, e 0,75, dois para **R**: 0,50 e 1,50, dois valores para o número de máquinas, 4 e 8, e dois valores para o número de tarefas, 25 e 50. Para cada combinação de **Z**, **R**, **n** e **m** foram gerados 30 problemas com os tempos de processamento gerados no intervalo [1,20] com distribuição uniforme. Os resultados foram comparados com os obtidos pela adaptação realizada por Raman na heurística proposta por Ow(1985), tendo o algoritmo NEHad associado à regra LI superado todos os resultados apresentados por esta adaptação.

- **Problema de Flowshop com buffer zero**

Apresenta-se a seguir na Tabela 2.2 os resultados obtidos pelas regras de despacho e combinação das mesmas com o algoritmo NEHad para o problema de *flowshop* com buffer zero. Esta comparação foi realizada sobre os mesmos 400 problemas descritos para *flowshop* com buffer ilimitado.

Tabela 2.2. Comparação entre regras de despacho e o algoritmo NEHad.

n	m	EDD	LI	MDD	SLACK	NEHad (EDD)	NEHad (LI)	NEHad (MDD)	NEHad (SLACK)
20	5	0,732	0,878	0,506	0,920	0,088	0,050*	0,103	0,077
20	10	0,745	0,929	0,529	0,899	0,068	0,052*	0,100	0,064
20	20	0,805	0,948	0,459	0,897	0,032*	0,037	0,105	0,032*
50	5	0,837	0,888	0,509	0,948	0,078	0,034*	0,125	0,089
50	10	0,832	0,939	0,492	0,932	0,063	0,050*	0,101	0,078
50	20	0,876	0,944	0,634	0,927	0,059	0,036*	0,113	0,044
100	5	0,808	0,900	0,497	0,996	0,035*	0,045	0,091	0,078
100	10	0,863	0,922	0,512	0,973	0,054	0,042*	0,102	0,078
100	20	0,870	0,924	0,583	0,949	0,056*	0,058	0,085	0,059
200	10	0,900	0,917	0,575	0,976	0,039	0,034*	0,078	0,070
Média		0,827	0,919	0,530	0,942	0,057	0,044*	0,100	0,067
Desvio Padrão		0,028	0,005	0,025	0,010	0,003	0,001	0,002	0,003

Nota-se através da média dos índices que o algoritmo NEHad associado a regra LI também obteve os melhores resultados para o *flowshop* com buffer zero. Esta associação é citada neste trabalho como NEHad*. Destaca-se o fato do tempo computacional das regras ser pequeno; para problemas de maior dimensão (200x10) estas requerem um tempo da ordem de 1 segundo enquanto o algoritmo NEHad consome para estes mesmos problemas um tempo da ordem de 100 segundos. O intuito desta análise é encontrar soluções iniciais com um bom equilíbrio entre qualidade e tempo computacional. Logo, como com regras de despacho tem-se soluções de pior qualidade e tempos excelentes, e com o algoritmo NEHad tem-se soluções de melhor qualidade e tempos maiores, conclui-se que não se pode, nesta etapa do trabalho, eliminar nenhuma das alternativas analisadas.

2.3.2. COMPONENTES DA BUSCA TABU PARA *FLOWSHOP* COM BUFFER ILIMITADO

Com o intuito de determinar os diversos componentes da Busca Tabu, foram analisadas e testadas diferentes atribuições para cada um deles que são descritas a seguir:

a. Estratégia de geração de vizinhança:

- uma tarefa **i**, localizada na posição $\pi(i)$, é inserida na posição **y**; esta estratégia gera uma vizinhança de tamanho $(n-1)^2$;
- troca de tarefas: uma tarefa **i**, localizada na posição $\pi(i)$, é trocada com a tarefa **j**, localizada na posição $\pi(j)$; neste caso o tamanho da vizinhança é $\frac{n}{2}(n-1)$.

b. ***Escolha da próxima solução:*** dada uma solução avalia-se cada solução vizinha e seleciona-se:

- a primeira solução que melhora a solução atual (primeiro movimento); ou
- a melhor solução encontrada (melhor movimento).

c. ***Seleção dos atributos do movimento e regras de proibição:*** para cada tipo de movimento foram selecionados atributos e regras específicas. Estes são apresentados a seguir, do mais restrito ao menos restrito. Para o movimento de inserção tem-se:

- **i** - a tarefa **i** não pode ser movida;
- **i** - a tarefa **i** não pode ser escolhida para inserção;
- **i, $\pi(i)$** - a tarefa **i** não pode retornar à posição $\pi(i)$.

Para o movimento de troca, estabeleceu-se como atributo e respectiva regra de proibição:

- **ij** - a tarefa **i** e a tarefa **j** não podem ser movidas.
- **i** - a tarefa **i** não pode ser movida;
- **mi** - a tarefa **i** não pode ser movida para posições **k**, onde $k \leq \pi(i)$;

Estes atributos e regras foram selecionados por serem os mais conhecidos ou por apresentarem melhores resultados em problemas correlacionados como em Laguna *et al.*(1991) e Taillard(1990).

d. ***Duração Tabu:*** a atribuição da duração da aplicação de uma regra de proibição a um movimento é feita de forma dinâmica. Para cada combinação atributo-regra esta duração é gerada dentro de um intervalo com distribuição uniforme. As seguintes formas de atualização são sugeridas:

- Aleatória - a cada iteração gera-se um novo valor para a duração da proibição.

- Ajustada - a cada vinte iterações gera-se um novo valor para a duração da proibição, e a cada iteração é realizado um ajuste neste valor . Este ajuste é feito de acordo com o valor da solução selecionada com relação à solução que gerou o movimento (solução atual). Caso o valor da solução selecionada seja maior que o da solução atual, a duração da proibição associada à solução selecionada é aumentada de uma unidade. Caso o valor da solução selecionada seja menor que o da solução atual, a duração é diminuída de uma unidade. Um procedimento similar é proposto em Dell'Amico & Trubian(1993).

A Tabela 2.3 mostra os limitantes deste intervalo de geração de acordo com a combinação utilizada.

Tabela 2.3. Limitantes de geração da duração tabu.

Movimento	Atributo	Regra	Limitante Inferior	Limitante Superior
Inserção	**i**	**i** não pode ser movido	n/4	♦ n/2
	i	**i** não pode ser escolhido para inserção	n/2	* n
	i, π(**i**)	**i** não pode retornar a posição π(**i**)	4n	♦ 5n
Troca	**ij**	**i** e **j** não podem ser movidos	n/4	* n/2
	i	**i** não pode ser movido	n/2	* n-1
	mi	**i** não pode ser movido para posições **k**, onde **k** $\leq \pi$(**i**)	n/2	♦ 3n/2

Observa-se que quanto mais restrita é a combinação atributo-regra, menores são os limitantes superiores. Os limitantes marcados com asterisco foram selecionados através de um estudo sobre o número de iterações necessárias para que não houvessem mais movimentos disponíveis (não tabu) para serem escolhidos, considerando que se uma regra de proibição é ativada, esta permanece válida ao longo das iterações. Além disso, assume-se que o critério de aspiração não está ativo. Este estudo, obviamente, está diretamente relacionado com o tipo de atributo e regra. Para exemplificar, descreve-se a seguir a determinação do limitante superior de geração para o atributo **i** do movimento de inserção associado à segunda regra de proibição. Suponha que uma dada tarefa foi escolhida para ser inserida. Como a regra de proibição estabelece que a tarefa selecionada não deve mais ser escolhida para inserção, conclui-se que tem-se agora **n-1** tarefas a serem analisadas para o próximo movimento. Considerando que se proibe uma tarefa a cada iteração, a última inserção possível se realizará na iteração **n**. Os limitantes superiores marcados com losangos e os limitantes inferiores foram estimados

através de testes computacionais. Cabe destacar que foram realizados diversos testes preliminares com diferentes intervalos de geração, bem como com vários números fixos para a duração tabu, inclusive com o famoso 7 sugerido por Glover & Laguna(1993). As formas dinâmicas de geração desta duração foram selecionadas por apresentarem os melhores resultados.

e. ***Critério de aspiração:*** um movimento pode ser liberado da condição tabu caso possua um valor de solução melhor que a melhor solução encontrada até o momento pela busca.

f. ***Critério de parada:*** tempo de C.P.U.. Estabeleceu-se este critério com o intuito de realizar comparações com diferentes combinações de componentes que utilizam, seguramente, um número diferenciado de iterações. Apresenta-se na Tabela 2.4 abaixo o tempo estabelecido para cada dimensão de problema. Estes tempos foram estabelecidos experimentalmente, onde observou-se que estes eram suficientes para um bom desempenho da heurística.

Tabela 2.4. Tempos utilizados para cada dimensão.

Dimensão (nxm)	Tempo (seg.)
20 x 5	25
20 x 10	50
20 x 20	90
50 x 5	200
50 x 10	500
50 x 20	1000

- **Resultados Computacionais**

Com o intuito de avaliar todas as possíveis combinações das atribuições dos componentes citadas anteriormente, foram geradas 120 diferentes implementações variando-se:

- 4 soluções iniciais fornecidas por regras de despacho, e 1 fornecida pelo algoritmo NEH associado à regra LI (NEHad*);
- 2 estratégias de geração de vizinhanças: troca e inserção;
- 2 formas de escolha da próxima solução: melhor movimento e primeiro movimento;
- 3 atributos do movimento e respectivas regras de proibição para cada vizinhança;
- 2 formas de atualização da duração tabu: aleatória e ajustada.

Estas diferentes implementações foram avaliadas sobre o conjunto de problemas cujas dimensões são apresentadas na Tabela 2.4. Para cada dimensão foram geradas três diferentes matrizes

de tempos de processamento, e para cada uma destas foram selecionados quatro distintos cenários (seção 1.2.1), ou seja, foram testados 6 x 3 x 4 = 72 problemas.

A primeira avaliação realizada considerou as diferentes atribuições para solução inicial. Estas apresentaram, de forma geral, um desempenho equivalente no movimento de inserção. No movimento de troca, destacaram-se as regras de despacho EDD e o algoritmo NEHad*.

Quanto aos atributos e regras de proibição associadas a estes, destacou-se dentre todas as combinações realizadas, o atributo **i** , com proibição da tarefa **i** ser selecionada para inserção, e o atributo **mi** como os melhores atributos, respectivamente, nos movimentos de inserção e troca. Observe que para a vizinhança gerada pela troca o atributo que obteve melhor desempenho é o atributo menos restrito, o que é coerente dado que a vizinhança da troca não é grande comparativamente à da inserção. Por outro lado, o melhor atributo para a vizinhança gerada pela inserção é um atributo razoavelmente restritivo, o que também é coerente dado o tamanho da vizinhança.

Com relação às formas de atualização da duração tabu destacou-se o bom desempenho da lista dinamicamente ajustada em ambas as vizinhanças. Quanto às estratégias de escolha da próxima solução tem-se que o critério que elege o melhor movimento obteve resultados superiores ao que seleciona o primeiro movimento, embora no movimento de troca o primeiro movimento também tenha apresentado bons resultados.

Em relação à estratégia de geração de vizinhanças, observou-se que o movimento de inserção obteve resultados superiores ao movimento de troca em todas as combinações. Este fato ocorre devido à inserção possuir uma vizinhança mais rica em termos de alteração de características da solução atual, o que teve muita influência no seu bom desempenho. Enquanto a troca altera o programa somente nas posições selecionadas para o movimento, a inserção desloca todas as tarefas que se localizem entre a tarefa escolhida para inserção e sua nova localização. São apresentadas a seguir as porcentagens de melhoria, em relação ao algoritmo NEHad*, das melhores combinações de componentes que utilizam o movimento de troca e inserção. Esta porcentagem de melhoria é calculada da seguinte forma:

$$\text{Porcentagem de melhoria} = \frac{T_{NEHad^*} - T_{tabu}}{T_{NEHad^*}} \times 100$$

No cálculo da média desta porcentagem por classe são desconsiderados os problemas nos quais o algoritmo NEHad* ou a Busca Tabu obtiveram a soma dos atrasos zero. Isto porque a melhoria da Busca Tabu é de 100% independente do valor da solução do algoritmo NEHad*. As classes que possuem problemas nos quais a Busca Tabu obteve a soma dos atrasos zero partindo de uma solução

não nula aparecem em negrito. Além disso, caso todos os problemas de uma classe estejam incluídos nestes dois casos, esta será representada por um traço (-), e não constará no cálculo da média por cenário.

Para se ter idéia da dimensão dos atrasos tratados apresenta-se na Tabela 2.5 os valores médios das somas dos atrasos obtidos através do algoritmo NEHad* para cada classe.

Tabela 2.5. Soma dos atrasos obtidos através do algoritmo NEHad*.

Dimensão (n x m)	Cenários				Tempo médio (seg.)
	1	2	3	4	
20x5	499,20	293,80	2267,90	2677,60	0,09
20x10	2021,00	2681,00	5625,60	5263,20	0,14
20x20	6473,80	7629,70	12483,70	13704,00	0,28
50x5	552,60	19,30	6870,60	5858,20	1,19
50x10	3401,20	592,60	15600,40	14800,30	2,16
50x20	9612,90	9255,40	32060,20	34635,40	4,05

A melhor combinação da troca consiste dos seguintes componentes: regra EDD como solução inicial; melhor movimento; atributo **mi**; lista dinamicamente ajustada. Os resultados desta combinação estão expostos na Tabela 2.6.

Tabela 2.6. Melhoria percentual da Busca Tabu com troca em relação ao NEHad*

Dimensão (n x m)	Cenários			
	1	2	3	4
20x5	80,26	22,24	28,01	16,35
20x10	15,55	8,05	5,56	9,90
20x20	2,69	5,28	3,86	2,22
50x5	**88,38**	-	30,27	27,88
50x10	72,49	12,92	12,14	6,65
50x20	36,80	5,47	3,03	4,14
Média	49,36	10,79	13,81	11,19
Desvio Padrão	32,89	6,35	11,24	8,73

Os melhores resultados de inserção possuem os seguinte componentes: solução inicial fornecida pela regra MDD; melhor movimento; atributo **i** com proibição da tarefa **i** ser selecionada para inserção; lista dinamicamente ajustada. Esta implementação obteve as porcentagens de melhoria apresentadas na Tabela 2.7 a seguir.

Tabela 2.7. Melhoria percentual da Busca Tabu com inserção em relação ao NEHad*

Dimensão (n x m)	Cenários			
	1	2	3	4
20x5	83,94	25,52	31,78	19,00
20x10	27,83	9,44	9,99	12,36
20x20	9,23	7,91	6,80	4,76
50x5	**88,38**	-	32,29	34,77
50x10	84,58	21,33	22,65	23,27
50x20	50,15	13,70	8,38	9,72
Média	57,35	15,58	18,98	17,31
Desvio Padrão	30,69	6,81	11,20	9,86

A heurística utilizando a vizinhança de troca obteve uma média total de melhoria de 21,29%, enquanto a utilização da inserção obteve uma média de 27,30%, o que confirma o melhor desempenho da vizinhança de inserção.

Uma reavaliação do fator tempo foi considerada para esta versão básica de Busca Tabu. A heurística foi avaliada com o dobro do tempo para cada problema. Neste teste obteve-se uma média total de melhoria de 27,46%. Comparando-se este resultado com o obtido com o tempo normal de execução (média de melhoria de 27,30%) observa-se que o aumento de tempo não causou grande impacto sobre a melhoria relativa do método. Destaca-se que não houve nenhuma melhoria individual relevante.

Para comprovar o comportamento do método, a melhor combinação utilizando inserção foi avaliada sobre um novo conjunto de 240 problemas conforme apresentado na Tabela 2.8. Para cada uma das seis dimensões gerou-se dez diferentes matrizes de tempos de processamento, e para cada uma destas foram selecionados quatro distintos cenários(seção 1.2.1). Observa-se desta tabela que, em média, a melhoria percentual da Busca Tabu em relação ao algoritmo NEHad* para os cenários 1,3 e 4 se mantém, enquanto para o cenário 2 há um aumento significativo desta média. Tal fato se deve ao comportamento dos métodos comparados nos problemas deste cenário com dimensão 50x5. Nestes problemas a Busca Tabu e o algoritmo NEHad* não obtiveram todas as somas dos atrasos nulas, logo estas puderam ser consideradas na média por cenário.

Tabela 2.8. Melhoria percentual da Busca Tabu em relação ao NEHad* em 240 novos problemas.

Dimensão (n x m)	Cenários				Média por dimensão
	1	2	3	4	
20x5	78,68	10,65	21,75	17,44	32,13
20x10	34,22	12,54	10,12	9,65	16,63
20x20	10,90	8,32	5,50	5,44	7,54
50x5	**93,96**	**79,03**	30,45	22,44	56,47
50x10	**85,58**	37,09	25,32	18,29	42,47
50x20	45,59	17,52	10,30	12,76	21,54
Média	58,16	27,52	17,24	14,34	
Desvio Padrão	30,05	24,91	9,10	5,69	

Através da Tabela 2.8 nota-se que nos cenários 1 e 2, que possuem baixo fator de atraso, a heurística obteve maiores melhorias em relação ao algoritmo NEHad*, enquanto nos cenários 3 e 4 , que possuem alto fator de atraso, a melhoria é menor. Não se pode, no entanto, avaliar o desempenho de Busca Tabu com respeito ao fator de atraso baseado nos resultados desta tabela, pois uma maior melhoria em relação ao NEHad* não implica que a solução fornecida pela Busca Tabu está mais próxima ou mais distante do valor ótimo. Observa-se também através da última coluna da Tabela 2.8 que quanto maior **n**, mantendo-se **m** fixo, maior também é a média de melhoria por dimensão. Com **n** maior, maior o número de soluções possíveis para o problema (**n!**), tornando a solução NEHad*, proveniente de uma enumeração parcial, mais pobre. Tal fato proporciona oportunidade para um maior ganho da Busca Tabu sobre este algoritmo.

Com os componentes já definidos, explicitam-se a seguir os passos básicos da heurística. Para tal se torna necessária a definição de algumas variáveis:

- somatraso_inicial : valor da soma dos atrasos para a solução inicial fornecida pela regra MDD.
- programa_inicial: programa correspondente à solução inicial.
- somatraso_inc : valor da soma dos atrasos da solução incumbente, ou seja, a melhor solução encontrada até o momento pela busca.
- programa_inc: programa correspondente à solução incumbente.
- somatraso_atual: valor da soma dos atrasos da solução da iteração atual da busca.
- programa_atual: programa correspondente à solução atual.
- Proibe: vetor que armazena até que iteração as tarefas não podem ser selecionadas para inserção.

- Obtenção da solução inicial e da correspondente soma dos atrasos através da regra MDD.
- Inicialize um vetor Proibe com n posições com zero.
- somatraso_inc = somatraso_ inicial
- programa_inc = programa_inicial
- programa_atual = programa_inicial
- somatraso_atual = somatraso_inicial
- n_iterações = 0
- *faça enquanto* (tempo_de_execução ≤ tempo_de_parada) ou (somatraso_inc ≠ 0)
    - n_iterações = n_iterações + 1
    - começo = 0
    - n_inserções = 0
    - *faça enquanto* (n_inserções ≤ $(n-1)^2$)
        - n_inserções = n_inserções + 1
        - Insira a tarefa i selecionada na posição y.
        - Calcule a soma dos atrasos relativa a esta inserção e armazene em somatraso.
        - *se* ((somatraso < somatraso_inc) ou (Proibe[i] < n_iterações))
            - *se* (começo = 0)
                - começo=1
                - melhor = somatraso
                - melhor_i = i
                - melhor_pos = y
            - *senão*
                - *se* (somatraso<.melhor)
                - melhor = somatraso
                - melhor_i = i
                - melhor_pos = y
                - *fim_se*
            - *fim_se*
        - *fim_se*
    - *fim_enquanto*
    - Caso nenhum movimento não-tabu tenha sido selecionado como o melhor movimento eleja o movimento tabu que acarrete o menor valor da função objetivo.
    - Modifique a solução atual realizando o melhor movimento encontrado.
    - somatraso_atual = melhor.
    - Atualize a duração tabu de forma dinamicamente ajustada conforme detalhado no item d.
    - Proibe[melhor_i] = n_iterações + duração_tabu
    - *se* (somatraso_atual < somatraso_inc)
        - somatraso_inc = somatraso_atual
        - programa_inc = programa_atual
    - *fim_se*
- *fim_enquanto*

A análise do pior caso para a heurística proposta mostra que a sua complexidade é $\mathbf{O(n^3.m)}$. A obtenção desta complexidade é facilmente explicada dado que o tamanho da vizinhança é da ordem de

$\mathbf{n^2}$, e o cálculo da soma dos atrasos de todas as tarefas é **O(n.m)**. Dentro do laço das iterações todas as outras operações são **O(1)**, e fora deste as operações são de complexidade inferior a $\mathbf{O(n^3.m.)}$.

2.3.3. COMPONENTES DA BUSCA TABU PARA *FLOWSHOP* COM BUFFER ZERO

Nesta etapa dos testes restringiu-se os componentes a serem avaliados para a heurística de acordo com os melhores resultados obtidos para o *flowshop* com buffer ilimitado. Esta restrição é coerente com a análise dos dois problemas realizada nas seções 1.2.1 e 1.3.1, onde se observa um comportamento muito similar da distribuição das soluções dos problemas em relação ao valor ótimo. Somente a estratégia de geração de vizinhança será avaliada com as suas duas possilidades de atribuição: troca e inserção. Isto se deve a grande influência na busca deste componente. Citam-se a seguir as melhores atribuições de componentes testadas para o *flowshop* com buffer ilimitado que também são avaliadas para o *flowshop* com buffer zero:

a. ***Escolha da próxima solução:***

- melhor movimento.

b. ***Seleção do atributo do movimento e regra de proibição:***

- para inserção: atributo **i** , com a proibição da tarefa **i** não ser escolhida para inserção;
- para troca: atributo **mi** , com a proibição da tarefa **i** não pode ser movida para posições **k**, onde $\mathbf{k} \leq \pi(\mathbf{i})$.

c. ***Duração Tabu:*** lista dinamicamente ajustada. Os intervalos de geração com distribuição uniforme dentro dos quais se obtém a duração tabu são **[0,25.n; 0,85.n]** para inserção, e **[0,25.n; 1,0.n]** para troca. Estes intervalos foram determinados através de experimentos computacionais.

d. ***Critério de parada:*** tempo de C.P.U. (Tabela 2.4).

e. ***Critério de aspiração:*** um movimento pode ser liberado da condição tabu, caso possua um valor de solução melhor que a melhor solução encontrada até o momento pela busca.

♦ **Resultados Computacionais**

A Busca Tabu foi implementada com os componentes já descritos para a vizinhança de troca e inserção com objetivo de avaliar o efeito das diferentes soluções iniciais sobre a solução final obtida pelo método. Dado que o número de combinações de atribuições de componentes é menor que o estabelecido o *flowshop* com buffer ilimitado, ampliou-se o número de soluções iniciais avaliadas. Resumidamente foram avaliados:

- 4 soluções iniciais fornecidas por regras de despacho, e 4 fornecidas pelo algoritmo NEHad*;
- 2 estratégias de geração de vizinhança: troca e inserção;
- 1 forma de escolha da próxima solução: melhor movimento;
- 1 atributo do movimento e respectiva regra de proibição para cada vizinhança; e
- 1 forma de atualização da duração tabu: dinamicamente ajustada.

Esta avaliação foi realizada sobre os mesmos 72 problemas utilizados para o problema de *flowshop* com buffer ilimitado. Apresenta-se na Tabela 2.9 a seguir a melhoria percentual da melhor implementação utilizando a vizinhança de troca, em relação ao algoritmo NEHad*. A solução deste algoritmo é utilizada como solução inicial para a Busca Tabu. A diferença percentual é calculada da mesma forma que para buffer ilimitado.

Tabela 2.9. Melhoria percentual de Busca Tabu com troca em relação ao NEHad*.

Dimensão (n x m)	Cenários			
	1	2	3	4
20x5	50,51	27,22	16,80	24,30
20x10	18,49	17,39	10,82	9,78
20x20	5,59	5,22	4,00	2,78
50x5	59,86	**81,12**	29,09	33,69
50x10	41,40	60,18	15,40	10,44
50x20	29,60	15,75	8,01	3,70
Média	34,24	34,48	14,02	14,11
Desvio Padrão	18,54	27,04	7,99	11,23

Nota-se da Tabela 2.9 uma melhoria significativa em todos os cenários. Na Tabela 2.10 apresenta-se o desempenho da melhor implementação utilizando a vizinhança de inserção. Esta implementação também utilizou como solução inicial o algoritmo NEHad*.

Tabela 2.10. Melhoria percentual de Busca Tabu com inserção em relação ao NEHad*

Dimensão (n x m)	Cenários			
	1	2	3	4
20x5	49,95	29,75	18,95	26,16
20x10	20,81	19,60	11,47	10,79
20x20	7,27	6,62	5,57	3,21
50x5	71,03	**89,59**	32,58	40,76
50x10	42,23	59,90	17,73	12,38
50x20	33,18	23,69	10,86	6,36
Média	37,41	38,19	16,19	16,61
Desvio Padrão	20,46	28,10	8,58	12,98

Observa-se que a estratégia de geração de vizinhança que utiliza o movimento de inserção obteve resultados superiores ao movimento de troca em todos os cenários, confirmando os resultados obtidos para *flowshop* com buffer ilimitado. Considerou-se também a possibilidade de ajustes no tempo de execução da heurística e para tal avaliou-se a mesma no dobro do tempo estabelecido anteriormente. Neste teste obteve-se uma média total de melhoria de 28,08%, que comparada a média de melhoria de 27,1% obtida no tempo normal não representa um ganho muito grande. Destaca-se que dobrar o tempo de execução para o conjunto de problemas testes selecionados corresponde a acrescentar aproximadamente 6 horas a mais de tempo de C.P.U.. A complexidade desta heurística é a mesma apresentada para o problema de *flowshop* com buffer ilimitado, ou seja, $\mathbf{O(n^3.m)}$.

Com o intuito de comprovar o comportamento do método, a melhor combinação foi novamente avaliada sobre um conjunto de 240 problemas conforme apresentado na Tabela 2.11. Estes problemas são os mesmos utilizados para o problema de *flowshop* com buffer ilimitado.

Tabela 2.11. Melhoria percentual da Busca Tabu em relação ao NEHad* em 240 novos problemas

Dimensão (n x m)	Cenários				Média por dimensão
	1	2	3	4	
20x5	53,92	**37,40**	19,66	19,04	32,50
20x10	19,53	25,05	11,22	8,76	16,14
20x20	9,59	6,80	5,64	4,81	6,71
50x5	67,56	**91,11**	30,89	27,63	54,30
50x10	42,55	61,79	19,91	16,20	35,11
50x20	30,99	26,17	10,31	7,70	18,79
Média	37,36	41,39	16,27	14,02	
Desvio Padrão	19,76	27,69	8,29	7,83	

Da tabela anterior nota-se que em média a melhoria percentual de Busca Tabu em relação ao algoritmo NEHad*, para cada cenário, se mantém. Logo, conclui-se que a heurística tem um desempenho estável. Observa-se através da última coluna que as médias de melhoria aumentam de acordo com o aumento de **n**, mantendo-se **m** fixo. Esta constatação é a mesma realizada para o problema de *flowshop* com buffer ilimitado.

Dos testes apresentados neste capítulo constata-se que as implementações básicas das heurísticas obtiveram bons resultados. Ressalta-se que as heurísticas, respectivamente para *flowshop* com buffer ilimitado e buffer zero, superaram ou obtiveram as mesmas soluções apresentadas pelo algoritmo NEHad* em 99,17% e 100% dos 240 problemas testados. Estes resultados são diferentes dos apresentados por Kim(1993b) para *flowshop* com buffer ilimitado onde a Busca Tabu teve um desempenho inferior a outras heurísticas, incluíndo uma adaptação do algoritmo NEH, em 25% dos problemas testados. Tal resultado se deve ao fato do autor ter comparado esta adaptação do algoritmo NEH com a Busca Tabu proposta por Widmer & Hertz (1989), sendo que o ajuste dos componentes desta proposta foi realizado para outra medida de desempenho, o *makespan*. Na heurística apresentada neste trabalho foi realizado um extenso trabalho de atribuição de componentes específicos para minimizar a soma dos atrasos. Portanto, era previsível que a heurística proposta apresentasse um desempenho superior. Outro fator que pode influenciar tal resultado é o tempo de execução da busca. Porém, tal fator é difícil de ser avaliado, dado que os métodos foram executados em máquinas e configurações diferentes.

No próximo capítulo são descritas as estratégias de diversificação, intensificação, e redução da vizinhança aplicadas às heurísticas propostas com seus respectivos resultados computacionais

Capítulo 3
Heurísticas Propostas
Estratégias Adicionais

3.1. INTRODUÇÃO

Em algumas aplicações, os componentes da memória de curto prazo são suficientes para produzir soluções de alta qualidade. Porém, existem casos onde a Busca Tabu se torna significantemente mais potente ao se incluir a memória de longo prazo e as estratégias associadas a esta. Um tipo de memória de longo prazo usualmente utilizada é aquela baseada em freqüência. Esta geralmente armazena a **freqüência de transição** e a **freqüência de residência**. A freqüência de transição mantém um registro da freqüência com que os atributos se modificam, enquanto a freqüência de residência mantém o registro da quantidade de vezes que cada atributo aparece nas soluções. Através destas freqüências podem ser aplicadas penalizações e incentivos aos movimentos candidatos. Conforme Glover(1995), o uso da memória de longo prazo não requer execuções de larga duração para que seus benefícios se tornem visíveis. Freqüentemente, seu efeito começa a se manifestar em um tempo relativamente curto, e pode permitir que a busca dentro de um determinado tempo atinja soluções superiores às obtidas através da utilização somente da memória de curto prazo.

Dois elementos altamente importantes que utilizam a memória de longo prazo são as estratégias de intensificação e diversificação. As estratégias de intensificação favorecem combinações de movimentos e características de soluções que historicamente tenham sido boas; além disso incentivam o regresso a regiões atrativas para analisá-las de forma mais detalhada. Uma forma popular de identificar regiões atrativas é através da manipulação das **soluções de elite** (soluções de alta qualidade encontradas durante a busca). Uma técnica direta para retornar a estas regiões é o reinício da busca partindo de cada uma destas soluções. Outra estratégia, que também utiliza as informações provenientes das soluções de elite, é a decomposição. Através desta técnica os atributos marcantes das soluções de elite podem ser impostos à estrutura das soluções. Estes atributos podem ser identificados, por exemplo, através da freqüência de residência dos atributos das soluções de elite.

Além destas estratégias, a intensificação pode ser promovida através de modificações no processo de escolha da próxima solução, ou seja, os movimentos que geram soluções que contenham os atributos marcantes nas soluções de elite podem receber incentivos através destas modificações.

As estratégias de diversificação em Busca Tabu estão projetadas para conduzir a busca a novas regiões. Estas estratégias podem ter um papel muito importante para estruturas de vizinhanças em que certas trajetórias de busca podem tornar-se isoladas e distantes de novas alternativas promissoras. Comumente estão baseadas em modificar o processo de escolha através das freqüências de residência ou transição. Os movimentos que geram soluções com atributos pouco freqüentes podem ser incentivados, enquanto movimentos que geram soluções com atributos muito freqüentes podem ser penalizados. Alternativamente, os atributos não freqüentes podem ser introduzidos por um reinício do processo de resolução que incentive a presença destes atributos. Apresenta-se na Figura 3.1 uma estratégia de diversificação proposta por Glover(1995) que conforme o autor tem comprovado seu êxito em problemas de programação de tarefas.

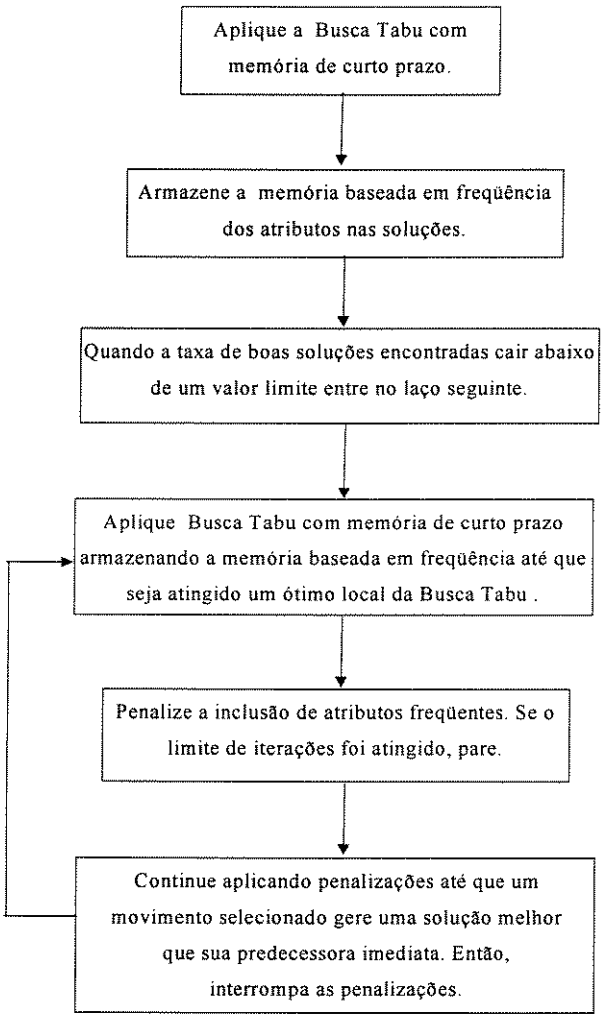


Figura 3.1. Esquema da diversificação proposta por Glover(1995).

UNICAMP BIBLIOTECA CENTRAL

Deve ser destacado que no início do processo de diversificação é comum que ocorra uma maior deterioração dos valores da função objetivo do que com a aplicação da Busca Tabu básica, pois o aparecimento de atributos menos freqüentes inicialmente pode acarretar soluções de qualidade muito inferior.

Uma outra opção para melhorar o desempenho da busca em situações onde a vizinhança é grande ou seus elementos demorados de avaliar, é a utilização da **lista de candidatos**. Esta estratégia objetiva restringir o número de movimentos a serem examinados a cada iteração mantendo a qualidade das soluções encontradas.

Existem outras estratégias de Busca Tabu, mas conforme Glover(1996) muitas são negligenciadas nas pesquisas, inclusive algumas citadas aqui como: lista de candidatos, estratégias de intensificação com o uso das soluções de elite, implementações utilizando diversificação e intensificação, dentre outras.

Neste capítulo são descritas as estratégias propostas de intensificação, diversificação, e de redução da vizinhança. Apresentam-se os resultados computacionais das heurísticas utilizando os componentes definidos para as versões básicas e as melhores estratégias adicionais encontradas. Além disso, descreve-se sucintamente os testes efetuados durante o processo de seleção das estratégias. Uma versão resumida da aplicação destas estratégias ao problema de *flowshop* com buffer ilimitado é apresentada em Armentano & Ronconi(1997).

3.2. PROBLEMA DE *FLOWSHOP* COM BUFFER ILIMITADO

3.2.1. ESTRATÉGIAS DE DIVERSIFICAÇÃO

Para este problema foram avaliadas diversas estratégias de diversificação. São apresentadas a seguir as estratégias que representam as principais linhas testadas. Todas estas passaram por um processo preliminar de ajuste de parâmetros onde foram selecionados os que obtiveram os melhores resultados. Os testes foram realizados no mesmo conjunto de 72 problemas descritos no capítulo 2, ou seja, 6 dimensões, 3 matrizes de processamento para cada dimensão, e 4 cenários distintos para cada matriz. A avaliação destas estratégias é realizada através de sua diferença percentual relativa à melhor adaptação do algoritmo NEH (NEHad*) apresentada no capítulo 2. As implementações apresentadas a seguir foram avaliadas com o dobro do tempo normal porém, de modo similar ao teste realizado para a

versão básica, estas estratégias não apresentaram melhorias relevantes. Portanto, a comparação entre as versões apresentadas é realizada no tempo normal de execução especificado na Tabela 2.4.

Estratégia de penalização do movimento: esta estratégia é baseada na freqüência de ocorrência de cada tarefa em cada posição, que é armazenada na matriz de residência ilustrada na Figura 3.2.

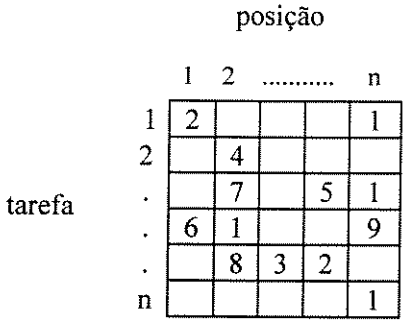


Figura 3.2. Matriz de residência.

A diversificação se realiza através de uma penalização $\mathbf{Pen_{ij}}$ aplicada ao valor do movimento candidato correspondente a inserção da tarefa i na posição j, ou seja:

$$\mathbf{Pen_{ij}} = \textbf{(freqüência da tarefa i na posição j) / (maior freqüência da matriz).}$$

O valor do movimento candidato é alterado para um novo valor penalizado, ou seja:

$$\textbf{valor do movimento penalizado} = \textbf{valor movimento} + \mathbf{d}*\mathbf{Pen_{ij}}.$$

onde **d** é um parâmetro ajustável da diversificação. Grandes valores de **d** indicam um peso maior para a freqüência de ocorrência de uma tarefa na posição selecionada dentro da matriz de residência, e portanto um maior peso para o fator de diversificação. Foram avaliados vários valores para o parâmetro **d**, e os que apresentaram melhores resultados nos testes preliminares foram os que estavam relacionados com o valor da solução inicial. Depois de vários testes avaliando-se diferentes porcentagens desta, adotou-se **d** como 5% da solução inicial.

Esta estratégia é acionada quando depois de **k** iterações sem melhoria da função objetivo, a busca encontra-se num ótimo local similar ao último ótimo local encontrado. Esta verificação de similaridade entre os ótimos é efetuada pois a ocorrência de um ótimo local depois de **k** iterações sem melhoria da função objetivo pode não ser suficiente para caracterizar um bom momento para mudar a busca de região. Este ótimo local, caso seja diferente do anterior, pode merecer uma maior investigação pela Busca Tabu básica. Para estabelecer esta semelhança entre ótimos locais estabeleceu-se uma medida de similaridade (**MS**) entre soluções. No cálculo desta medida uma solução é analisada de forma comparativa a outra solução: caso a tarefa da solução analisada esteja na

mesma posição da outra incrementa-se **MS** de dois, caso a tarefa esteja na posição anterior ou posterior incrementa-se **MS** de um. São consideradas similares as soluções que geram um **MS** maior **1,5.n**. A Figura 3.3 apresenta um exemplo de cálculo desta medida.

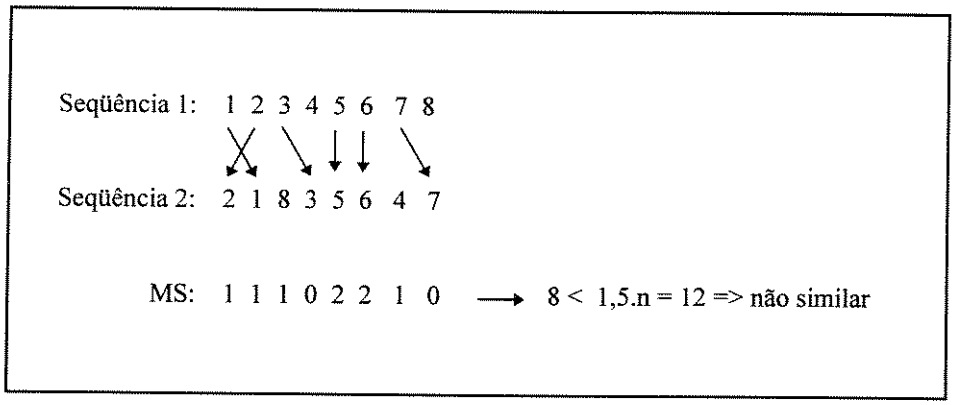


Figura 3.3. Exemplo do cálculo da medida de similaridade.

Ressalta-se que o ótimo local da Busca Tabu pode não ser um ótimo local real pois as restrições impostas por Busca Tabu podem impedir uma descida maior na vizinhança. Aplica-se esta penalização até que a busca encontre uma solução de valor menor que a sua predecessora. Esta estratégia é uma variante da estratégia de diversificação apresentada na Figura 3.1. O parâmetro **k** foi utilizado como regulador da quantidade de diversificações que ocorrem durante a busca. Observou-se que nos cenários 1 e 2 que possuem datas de entrega mais folgadas, necessita-se de um menor número de diversificações (**k=n**) que nos cenários 3 e 4 com datas de entrega mais apertadas(**k=n/2**). A média de melhoria apresentada por esta estratégia é de 27,58%.

- ***Estratégia de penalização da solução*:** nesta estratégia diversifica-se a busca através da penalização das soluções candidatas de acordo com a matriz de residência. Uma penalidade $\mathbf{Pen_{ij}}$ é calculada para cada tarefa **i** que esteja alocada na posição **j** da solução candidata. A penalidade total da solução é a média da penalidades $\mathbf{Pen_{ij}}$, e pode ser expressa por:

$$\textbf{Penalidade total da solução candidata} = \sum_{j=1}^{n} \mathbf{Pen_{ij}} \; / \; \mathbf{n} \; ,$$

onde $\mathbf{Pen_{ij}}$ **= (freqüência da tarefa i na posição j) / (maior freqüência da matriz).**

O valor da solução candidata é alterado para um novo valor penalizado, ou seja:

Novo valor da solução candidata = valor da solução candidata *(1 + Penalidade total).

Esta estratégia é acionada quando encontra-se um ótimo local de Busca Tabu depois de **n** iterações sem melhoria da função objetivo, e é interrompida quando a busca encontra uma solução de

valor menor que a sua predecessora. A média de melhoria apresentada por esta estratégia é de 27,51%.

- ***Estratégia de penalização da função objetivo***: esta estratégia se baseia nos valores da função objetivo correspondentes às soluções candidatas. Para cada nova solução testada avalia-se a percentualmente a diferença entre o seu valor e a média das últimas soluções obtida no momento de acionamento da estratégia. Caso esta porcentagem esteja entre 0 e 200%, esta solução será penalizada por uma taxa decorrente da proximidade do valor da solução candidata do valor da média. Esta taxa é definida por:

taxa de proximidade = 2 - diferença/200,

onde **diferença = 100* (valor da solução candidata - média)/média.**

O cálculo do valor da solução candidata é dado por:

Novo valor da solução candidata = valor da solução candidata *taxa de proximidade.

Esta estratégia de diversificação é acionada quando se caracteriza uma estabilização da heurística, ou seja, os valores da função objetivo não sofrem grandes variações e não estão em declínio. Utilizou-se para detectar esta situação dois parâmetros da probabilidade: a média e o desvio padrão. Calcula-se a média e o desvio padrão baseado nos últimos 9 valores da função objetivo. Caso o desvio padrão desta amostragem for menor que 5% da média e a média atual for maior que a média anterior, caracteriza-se o momento para aplicar a estratégia de diversificação. Em média, esta estratégia obtém 26,95% de melhoria.

- ***Estratégia de reinício através da matriz de residência:*** esta diversificação é realizada através do reinício da busca com uma nova solução gerada de acordo com a matriz de residência, ou seja, para cada posição do seqüênciamento analisa-se todas as possíveis tarefas disponíveis, e seleciona-se a primeira tarefa não seqüênciada que possuir a menor freqüência de ocupação desta posição. Esta estratégia é acionada caso as soluções encontradas possuam o mesmo valor de função objetivo por 10 iterações. A média de melhoria apresentada é de 25,72%.

- ***Estratégia de reinício através da penalização da solução inicial***: esta diversificação é realizada através do reinício da busca com uma nova solução gerada através da regra de despacho MDD (seção 2.3.1) penalizada de acordo com a matriz de residência. A regra de despacho é alterada de acordo com os valores da matriz, ou seja, para cada posição **j** disponível da seqüência calcula-se para cada tarefa **i** não alocada o seguinte valor da regra:

$$\textbf{valor da regra penalizado}_{ij} = \textbf{valor da regra}_{ij} \cdot (1 + \textbf{Pen}_{ij}),$$

onde $\textbf{Pen}_{ij}$ = **(freqüência da tarefa i na posição j) / (maior freqüência da matriz).**

Esta estratégia é acionada quando encontra-se um ótimo local da Busca Tabu depois de **n** iterações sem melhoria da função objetivo. Foi estabelecido um mínimo de 5 iterações entre cada reinício. A média de melhoria alcançada por esta estratégia é de 27,22%.

- ***Estratégia de influência:*** esta estratégia é realizada através da execução de um movimento de influência. Esta diversificação é acionada quando depois de **n** iterações sem melhoria da função objetivo a busca se encontra em um ótimo local da Busca Tabu. Definiu-se como movimento de influência o que possui o menor valor da seguinte expressão:

valor do movimento - distância,

onde **distância = | posição futura da tarefa i - atual posição da tarefa i |.**

A média de melhoria obtida é de 26,83%.

Apresenta-se a seguir os resultados computacionais da estratégia de penalização do movimento por esta possuir o melhor desempenho em relação à média de melhoria dentre as diversas diversificações sugeridas anteriormente. Observe que nem todas as diversificações sugeridas obtém médias superiores à média da versão básica (27,30%). Na Tabela 3.1 tem-se o resultado da aplicação desta estratégia a um conjunto maior de 240 problemas, ou seja, 6 dimensões, 10 matrizes de processamento, e 4 cenários. Apresenta-se para cada classe (combinação de cenário e dimensão) a média da melhoria dos 10 problemas pertencentes a esta. Os números em negrito na tabela indicam as classes nas quais a Busca Tabu encontrou somas dos atrasos zero e o algoritmo NEHad* não.

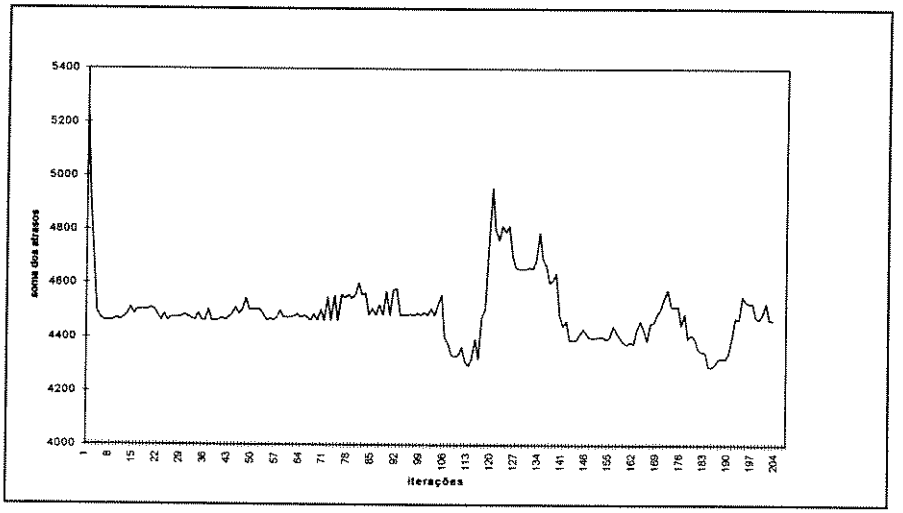
Tabela 3.1. Melhoria percentual de Busca Tabu com diversificação.

Dimensão	Cenários			
(n x m)	1	2	3	4
20x5	78,31	10,65	22,22*	17,60*
20x10	34,46*	13,05*	10,17*	10,00*
20x20	10,98*	8,30	5,32	5,54*
50x5	**93,96**	**79,03**	31,42*	22,27
50x10	**86,14***	37,09	25,32	18,20
50x20	46,58*	18,07*	10,50*	12,52
Média	58,41	27,70	17,49	14,36
Desvio Padrão	29,98	24,82	9,38	5,60

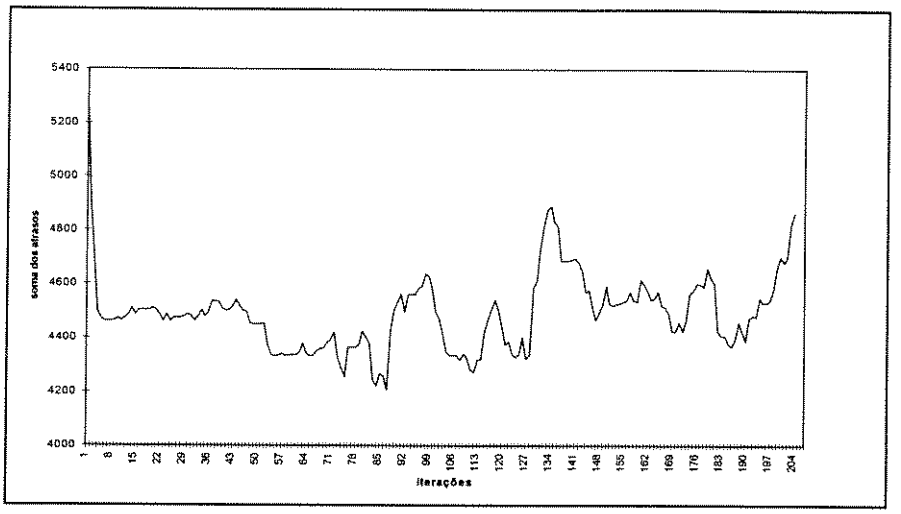
* classe de problemas onde a estratégia avaliada supera a Busca Tabu básica

Nota-se que embora o ganho da diversificação na média em cada cenário seja pequeno, 54,17% das classes dos problemas apresentaram melhorias, e 20,83% das classes mantiveram suas médias. Tal resultado pode ser parcialmente creditado ao ajuste de parâmetros de acordo com os cenários. Este ajuste foi realizado através da Figura 1.5, onde observou-se que os cenários 3 e 4, que possuem como característica comum o alto fator de atraso, apresentam uma maior concentração de soluções em uma determinada porcentagem acima do ótimo. Esta concentração de soluções com valores muito próximos pode dificultar o processo de busca da heurística em sua versão básica, pois esta pode dispender um tempo excessivo para escapar destes patamares de valores. Por este motivo, aplicou-se um maior número de diversificações para estes cenários. De maneira similar, observou-se que os cenários 1 e 2 apresentam menor concentração de soluções, e portanto necessitam de um número menor de diversificações.

Ilustra-se através da Figura 3.4 o comportamento da busca para um problema com 20 tarefas e 5 máquinas na versão básica, e na versão com diversificação.



(a)



(b)

Figura 3.4. Representação gráfica do comportamento da busca (a) na versão básica e (b) com diversificação.

Note que a ação da diversificação é muito suave em relação à deterioração da qualidade da solução, porém suficiente para interromper uma fase de estabilização da busca na sua versão básica, e encontrar uma solução melhor que a solução final da versão básica em menos iterações.

3.2.2. Estratégias de Intensificação

A seguir relacionam-se as principais estratégias de intensificação investigadas. Estas estratégias também foram avaliadas com o dobro do tempo apresentado na Tabela 2.4, porém, dado que não apresentaram resultados relevantes, os testes foram efetuados no tempo original.

- ***Estratégia de decomposição:*** esta estratégia constrói uma nova solução a partir das características das cinco melhores soluções obtidas até o momento do seu acionamento. Para cada posição observa-se qual tarefa aparece com maior freqüência, e esta é alocada para esta posição. As tarefas que aparecem sempre na mesma posição em todas as soluções são proibidas de serem escolhidas para inserção por **n/4** iterações. A memória de curto prazo é apagada. O seu acionamento é permitido quando restar ¼ do tempo de execução e a busca não estiver em descida. Esta descida é caracterizada quando a média das últimas 9 soluções for maior que a média das últimas 8 soluções e da solução atual. A média de melhoria obtida por esta estratégia é 26,45%.

- ***Estratégia de reinício:*** esta estratégia recomeça a busca da melhor solução encontrada. O seu acionamento ocorre quando restar ¼ do tempo total de execução, a busca não esteja em descida, e a melhor solução encontrada seja diferente da solução inicial e da última solução utilizada nesta estratégia

de intensificação. Identifica-se a descida da busca conforme descrito na estratégia anterior. A memória de curto prazo é descartada. Caso nesta busca seja encontrada uma solução melhor que a que originou a intensificação, esta deve ser armazenada e classificada para um novo possível reinício. Com esta estratégia obtém-se uma média de melhoria de 27,35%.

- ***Estratégia de múltiplos reinícios:*** esta estratégia de intensificação armazena em uma matriz as cinco melhores soluções de toda busca e, quando restar ¼ do tempo total de execução e a busca não estiver em descida (ver estratégia de decomposição), recomeça a busca de cada uma das cinco soluções. A memória de curto prazo é descartada, e realiza-se uma busca de no mínimo cinco iterações a partir de cada solução selecionada, começando pela melhor. Caso seja encontrada uma solução melhor que qualquer outra previamente encontrada, esta deve ser armazenada no final da matriz. Esta estratégia obtém 27,31% de melhoria média.

- ***Estratégia de reinícios com memória:*** esta estratégia se baseia no armazenamento de uma matriz contendo as cinco melhores soluções que superaram o valor da incumbente durante a busca, ou alternativamente, as cinco melhores incumbentes da busca. Armazena-se também a memória associada a cada solução além do próximo movimento selecionado. Desta forma espera-se que ao partir de uma destas boas soluções, a busca siga por um caminho diferente do anterior. Quando restar ¼ do tempo total de execução e a busca não estiver em descida (ver estratégia de decomposição), a busca recomeça de cada uma das cinco soluções com sua respectiva memória. Realiza-se uma busca de no mínimo cinco iterações a partir de cada solução selecionada, começando sempre pela última. Caso seja encontrada uma solução melhor que qualquer outra previamente encontrada, esta deve ser armazenada no final da matriz. Esta estratégia de intensificação é proposta por Nowicki & Smutnicki(1996a) com um acionamento diferente baseado no número de iterações sem atualização da incumbente. Em média, esta estratégia obtém 27,37% de melhoria.

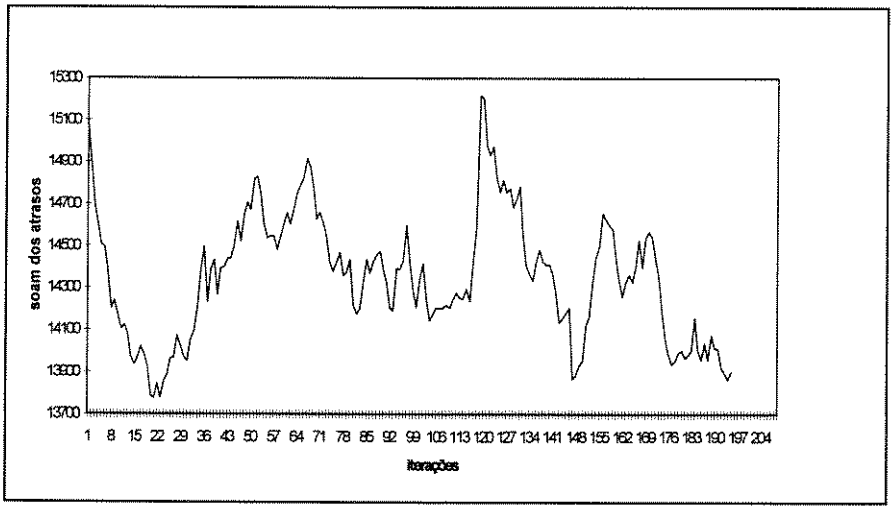
Nos diversos testes realizados destacaram-se como os melhores critérios de acionamento os que só permitem o início da intensificação no último ¼ do tempo total de execução da busca. Tal fato se justifica, dado que acionar a intensificação no final da busca permite um maior tempo para a coleta dos dados que são utilizados para direcioná-la. A estratégia de intensificação baseada em reinícios com memória obtém os melhores resultados em termos de melhoria média. Apresenta-se na Tabela 3.2 os resultados da aplicação da mesma em 240 problemas.

Tabela 3.2. Melhoria percentual de Busca Tabu com intensificação.

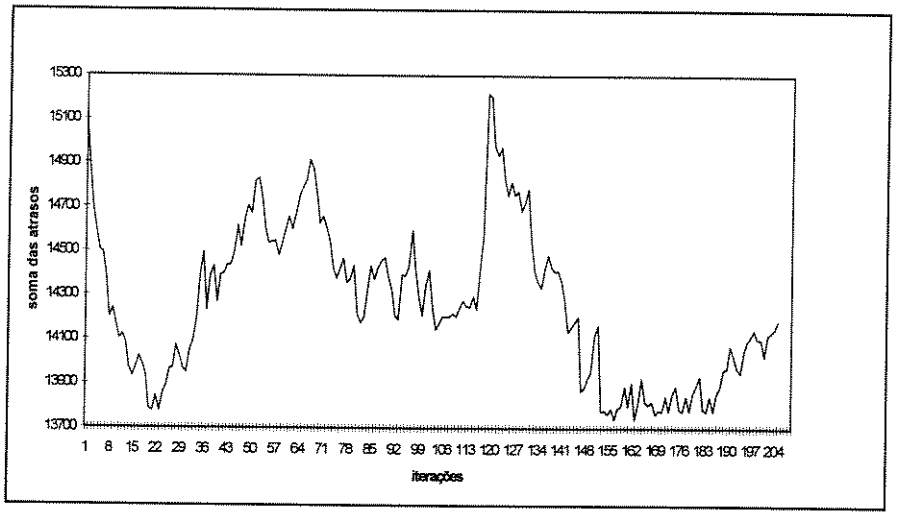
Dimensão (n x m)	Cenários			
	1	2	3	4
20x5	78,34	10,65	21,84*	17,43
20x10	34,22	12,83*	10,10	9,66*
20x20	10,90	8,32	5,53*	5,44
50x5	**93,96**	**79,03**	30,50*	22,26
50x10	**85,56**	36,65	25,88*	18,35*
50x20	46,30*	16,99	10,56*	12,85*
Média	58,21	27,41	17,40	14,33
Desvio Padrão	29,96	24,89	9,17	5,65

Através da Tabela 3.2 nota-se que a intensificação obtém melhorias em 41,67% das classes de problemas, mantém a melhoria em 29,17% das classes, e que somente as médias de melhoria dos cenários 1 e 3 aumentaram em relação à versão básica. Observa-se também que esta estratégia obtém em geral melhorias muito pequenas, embora deva ser destacado o seu desempenho no cenário 3, onde 5 das 6 classes apresentam melhorias.

Para ilustrar o comportamento da intensificação apresenta-se na Figura 3.5 uma aplicação bem sucedida em um problema com 20 tarefas e 20 máquinas do cenário 3. Nota-se que, aproximadamente na iteração 150, a busca se dirige a uma região com valores baixos da soma dos atrasos, e que intensificando-se nesta região atinge-se valores menores que os obtidos até o momento.



(a)



(b)

Figura 3.5. Representação gráfica do comportamento da busca (a) na versão básica e (b) com intensificação.

3.2.3. Estratégias de Diversificação e Intensificação

Como próxima etapa mesclou-se a estratégia de diversificação com a estratégia de intensificação. Várias combinações de estratégias foram avaliadas e a seguir descreve-se a que obteve os melhores resultados no tempo normal de execução da Tabela 2.4.

A estratégia proposta permite a aplicação da diversificação de penalização do movimento na parte inicial da busca, e a aplicação da intensificação de reinícios com memória no tempo final. Nos cenários 2 e 4 não foi aplicada a intensificação, pois esta não se mostrou eficaz. Nestes cenários o uso somente da diversificação, ajustada de acordo com os cenários, foi suficiente para obter resultados superiores aos da versão básica. Nos cenários 1 e 3 aplicou-se a intensificação, respectivamente, nos últimos 10% e 25% do tempo total de execução da busca. Estes parâmetros foram determinados através de experimentos computacionais. A Tabela 3.3 mostra os resultados das porcentagens de melhoria para as várias classes de problemas.

Tabela 3.3. Melhoria percentual de Busca Tabu com diversificação e intensificação.

Dimensão (n x m)	Cenários			
	1	2	3	4
20x5	78,61	10,65	22,45*	17,60*
20x10	34,46*	13,05*	10,59*	10,00*
20x20	11,01*	8,30	5,51*	5,54*
50x5	**93,96**	**79,03**	31,44*	22,27
50x10	**86,26***	37,09	25,88*	18,20
50x20	47,01*	18,07*	10,71*	12,52
Média	58,55	27,70	17,76	14,36
Desvio Padrão	29,99	24,82	9,37	5,60

Obteve-se melhorias em relação à Busca Tabu na versão básica em 62,50% das classes de problemas, e em 16,67% das classes a melhoria se manteve. De forma geral, a estratégia proposta apresenta um bom desempenho em relação às melhorias médias por cenário, as melhores encontradas, destacando-se o seu desempenho no cenário 3, onde todas as classes aumentaram suas melhorias em relação à versão básica.

A Tabela 3. 4 a seguir contém uma síntese dos resultados da aplicação da diversificação e intensificação através da análise do comportamento das classes de problemas.

Tabela 3. 4. Síntese dos resultados obtidos através da aplicação das estratégias adicionais em relação à versão básica.

Estratégia adicional	Porcentagem de classes que aumentaram a melhoria	Porcentagem de classes que mantiveram a melhoria	Porcentagem de classes que diminuíram a melhoria
diversificação	54,17	20,83	25,00
intensificação	41,67	29,17	29,16
diver. + inten.	62,50	16,67	20,83

- **Análise da Aplicação da Diversificação e da Intensificação**

Com o intuito de reforçar as análises anteriores apresenta-se resumidamente na Tabela 3.5 a comparação destas estratégias adicionais com a versão básica através do número de sucessos obtidos nos 240 problemas. Esta técnica de avaliação é a mesma adotada por Widmer & Hertz(1989) e Moccellin(1995). Considera-se sucesso quando a implementação analisada obtém o menor valor da soma dos atrasos. A coluna denominada **Maior %** denota o maior ganho de uma implementação sobre a outra.

Tabela 3.5. Comparações entre a versão básica e as estratégias de diversificação e intensificação para *flowshop* com buffer ilimitado.

Estratégia adicional	Sucessos da estratégia adicional	Melhoria % média da estratégia adicional em relação à versão básica	Maior %	Sucessos da versão básica	Melhoria % média da versão básica em relação à estratégia adicional	Maior %	Empates
diversificação	48	3,16	30,41	27	1,94	11,16	165
intensificação	39	1,10	9,76	15	5,67	45,24	186
diver. + inten.	63	2,76	30,41	26	1,64	11,16	151

Através dos resultados desta tabela confirma-se o desempenho destas estratégias em relação à versão básica. Nota-se que a diversificação e a associação desta com a intensificação obtém um maior número de sucessos que a versão básica. Além disso, nos problemas em estas estratégias superam a versão básica, a porcentagem de melhoria média é superior à porcentagem média nos problemas em que a versão básica supera estas estratégias. O ganho destas estratégias sobre a versão básica chega no maior caso a 30,41%. A intensificação também apresenta um maior número de sucessos que a versão básica, porém a porcentagem média de melhoria nos problemas nos quais esta é superada pela versão básica é alta. Destaca-se que no pior caso a porcentagem de melhoria da versão básica sobre a intensificação chega a 45,24%. Tais observações são coerentes com as análises anteriores relativas às porcentagens de classes cujas melhorias aumentaram. Esta análise mostra que a diversificação associada à intensificação obtém os melhores resultados, seguida pela diversificação, enquanto a intensificação obtém resultados regulares.

A Tabela 3.6 a seguir apresenta uma análise complementar sobre a porcentagem média de melhoria considerando os sucessos e os empates.

Tabela 3.6. Outras comparações entre a versão básica e as estratégias de diversificação e intensificação para *flowshop* com buffer ilimitado.

Estratégia adicional	% de sucessos e empates da estratégia adicional	Melhoria % média da estratégia adicional em relação à versão básica	% de sucessos e empates da versão básica	Melhoria % média da versão básica em relação à estratégia adicional
diversificação	88,75	0,75	80,00	0,27
intensificação	93,75	0,19	83,75	0,42
diver. + inten.	89,17	0,81	73,75	0,24

Nota-se desta tabela que embora a diversificação associada à intensificação possua uma maior média de melhoria numa maior porcentagem de problemas do que a versão básica, esta melhoria é muito pequena. A causa de tal comportamento pode ser atribuida às características do problema de *flowshop* permutacional. Para este problema é possivel fazer uso de uma vizinhança rica gerada pelo movimento de inserção. A versão básica da heurística que utiliza esta vizinhança, e que passou um

extenso processo de atribuição de componentes, está muito bem capacitada para encontrar soluções de excelente qualidade. Tal fato justifica o desempenho pouco expressivo das estratégias adicionais da diversificação e da intensificação em relação a esta versão.

3.3. Problema de *Flowshop* com Buffer Zero

3.3.1. Estratégias de Diversificação

Para este problema restringiu-se os testes às estratégias de diversificação citadas anteriormente para *flowshop* com buffer ilimitado. Dentre estas, somente a estratégia de penalização do movimento superou a versão básica proposta. Por este motivo, ampliaram-se os testes com o intuito de calibrar os seus parâmetros (**k**, **d** e **MS**). Estes testes foram realizados com os 72 problemas descritos no capítulo 2. A atribuição do parâmetro **k** que apresentou os melhores resultados foi **n** para os cenários 1 e 2 que possuem datas de entrega mais folgadas, e **n/2** nos cenários 3 e 4 com datas de entrega mais apertadas. O parâmetro **d** foi estabelecido como 25% da solução inicial. Este ajuste já era previsto dado que as soluções para *flowshop* com buffer zero possuem em média valores superiores às encontradas para o problema de *flowshop* com buffer ilimitado. A melhor atribuição para considerar que duas soluções são similares é **MS** maior 1,5.**n**. Esta estratégia também foi avaliada com o dobro do tempo de execução não apresentando resultados relevantes. Os dados da Tabela 3.7 correspondem à aplicação desta estratégia a 240 problemas.

Tabela 3.7. Melhoria percentual de Busca Tabu para buffer zero com diversificação.

Dimensão (n x m)	Cenários			
	1	2	3	4
20x5	55,22*	**37,62***	20,16*	18,71
20x10	20,78*	24,52	11,13	8,93*
20x20	9,47	6,76	5,39	5,15*
50x5	67,50	**91,11**	30,88	27,43
50x10	42,31	63,63*	19,98*	16,47*
50x20	30,96	25,90	10,24	7,20
Média	37,71	41,59	16,30	13,98
Desvio Padrão	19,77	28,00	8,40	7,73

Através desta tabela, pode ser observado que obteve-se melhoria em 37,5% das classes dos problemas, e que, em média, os cenários 1, 2 e 3 apresentam melhorias em relação à versão básica. Porém, deve ser ressaltado que 58,3% das classes de problemas pioraram suas médias de melhoria, o que torna questionável a utilização desta estratégia.

3.3.2. Estratégias de Intensificação

Avaliou-se para este problema as estratégias de intensificação citadas para *flowshop* com buffer ilimitado. Nenhuma destas superou, em termos de melhoria média, a versão básica proposta. A estratégia de reinícios com memória apresentou o melhor resultado, e por tal motivo ampliou-se os testes para calibragem de seus componentes. Os componentes correspondentes a estes testes são os mesmos utilizados no *flowshop* com buffer ilimitado, com exceção que reduziu-se para um o mínimo de iterações permitidas, partindo de cada uma das soluções selecionadas. Apresenta-se na Tabela 3.8 a seguir a ampliação destes resultados para o conjunto de 240 problemas.

Tabela 3.8. Melhoria percentual de Busca Tabu para buffer zero com intensificação.

Dimensão (n x m)	Cenários			
	1	2	3	4
20x5	53,93*	**37,40**	19,70*	19,04
20x10	19,18	24,82	11,33*	8,87*
20x20	9,49	6,81*	5,70*	4,84*
50x5	67,34	**91,11**	30,97*	27,63
50x10	41,50	61,10	20,55*	15,86
50x20	30,33	26,25*	10,04	7,84*
Média	36,96	41,25	16,38	14,01
Desvio Padrão	19,77	27,62	8,38	7,78

Através desta tabela observa-se que em 45,83% das classes de problemas as melhorias aumentaram, e 16,67% mantiveram as melhorias obtidas com a versão básica. Destaca-se que esta estratégia obtém médias para os cenários 1, 2 e 4 menores que as obtidas com a versão básica.

3.3.3. Estratégias de Diversificação e Intensificação

Neste problema mesclou-se as estratégias de forma similar à utilizada para o *flowshop* com buffer ilimitado, ou seja, inicialmente aplica-se a diversificação de penalização do movimento, e ao fim da busca a intensificação com reinícios com memória. Sobre esta estratégia foram realizados diversos testes variando-se os seus parâmetros básicos, e os melhores resultados foram obtidos através das seguintes combinações para cada cenário:

- **cenário 1:** diversificação no início da busca e nos últimos 10% do tempo de execução aplicou-se intensificação;
- **cenário 2**: somente diversificação;
- **cenário 3 e 4**: somente intensificação nos últimos 25% do tempo de execução.

Os resultados desta mescla de estratégias encontram-se na Tabela 3.9.

Tabela 3.9. Melhoria percentual de Busca Tabu para buffer zero com diversificação e intensificação.

Dimensão (n x m)	Cenários			
	1	2	3	4
20x5	55,50*	**37,62***	19,70*	19,04
20x10	20,78*	24,52	11,33*	8,87*
20x20	9,43	6,76	5,70*	4,84*
50x5	67,17	**91,11**	30,97*	27,63
50x10	42,29	63,63*	20,55*	15,86
50x20	31,09*	25,90	10,04	7,84*
Média	37,71	41,59	16,38	14,01
Desvio Padrão	19,73	28,00	8,38	7,78

A utilização desta estratégia acarretou em um aumento de melhoria em 54,17% das classes de problema, manteve a melhoria em 12,5% das classes, melhorou a média dos cenários 1, 2 e 3 em relação à versão básica, e obtém a melhor média de melhoria sobre os quatro cenários.

A Tabela 3.11 contém uma síntese dos resultados da aplicação da diversificação e intensificação através da análise do comportamento das classes de problemas.

Tabela 3. 10. Síntese dos resultados obtidos através da aplicação das estratégias adicionais em relação à versão básica para o problema de flowshop com buffer zero.

Estratégia adicional	Porcentagem de classes que aumentaram a melhoria	Porcentagem de classes que mantiveram a melhoria	Porcentagem de classes que diminuíram a melhoria
diversificação	37,50	4,20	58,30
intensificação	45,83	16,67	37,50
diver. + inten.	54,17	12,50	33,33

- **Análise da Aplicação da Diversificação e da Intensificação**

Apresenta-se na Tabela 3.11 a comparação das estratégias propostas com a versão básica através do número de sucessos obtidos nos 240 problemas.

Tabela 3.11. Comparações entre a versão básica e as estratégias de diversificação e intensificação para *flowshop* com buffer zero.

Estratégia adicional	Sucessos da estratégia adicional	Melhoria % média em relação à versão básica	Maior %	Sucessos da versão básica	Melhoria % média em relação à estratégia adicional	Maior %	Empates
diversificação	50	3,41	41,02	50	2,00	10,19	140
intensificação	42	1,02	7,00	34	2,92	14,37	164
diver. + inten.	53	2,96	41,02	39	2,24	14,37	148

Os resultados desta tabela complementam a análise anterior sobre o desempenho destas estratégias em relação à versão básica. Nota-se que a estratégia com o melhor desempenho é a associação da diversificação com a intensificação que não só apresenta o maior número de sucessos em relação à versão básica, mas que também possui nestes problemas uma melhor média de melhoria do que a melhoria apresentada pela versão básica quanto supera esta associação. Observa-se também, por esta tabela, que a diversificação possui um desempenho similar à associação da diversificação com a intensificação, porém sabe-se do estudo da melhoria das classes realizado nos itens anteriores que a mescla das estratégias possui um desempenho superior em termos do número de classes que apresentaram melhorias e da média dos quatro cenários. Além disso, a associação da diversificação com a intensificação perde em menos casos para a versão básica. Estas duas estratégias apresentam um ganho no maior caso de 41,02% sobre a versão básica. A intensificação, como para o *flowshop* com buffer ilimitado, apresentou um fraco desempenho.

A Tabela 3.12 apresenta uma outra análise destes dados. Esta análise contém a porcentagem média de melhoria considerando os sucessos e os empates.

Tabela 3.12. Outras comparações entre a versão básica e as estratégias de diversificação e intensificação para *flowshop* com buffer zero.

Estratégia adicional	% de sucessos e empates da estratégia adicional	Melhoria % média da estratégia adicional em relação à versão básica	% de sucessos e empates da versão básica	Melhoria % média da versão básica em relação à estratégia adicional
diversificação	79,17	0,90	79,17	0,53
intensificação	85,83	0,21	82,50	0,50
diver. + inten.	83,75	0,78	77,92	0,47

Nota-se desta tabela que a diversificação associada à intensificação possui uma maior média de melhoria numa maior porcentagem de problemas do que a versão básica, porém, esta melhoria é muito pequena. Este comportamento é similar ao comportamento destas estratégias no problema de flowshop com buffer ilimitado.

3.4. Redução da Vizinhança

Seguramente, uma das etapas mais custosas das heurísticas propostas é a avaliação de todas as soluções geradas pela vizinhança de inserção. Com o intuito de reduzir este esforço, analisou-se o comportamento da busca para encontrar uma possível redução no tamanho da vizinhança que não gerasse perda na qualidade das soluções. Outros autores já sugeriram reduções nesta vizinhança, como Adenso_Díaz(1992) para o problema de *flowshop* proporcional com critério de minimização da soma ponderada dos atrasos, e Nagano & Moccellin(1995) para o *flowshop* com critério *makespan*.

Através de testes computacionais notou-se que na maioria dos problemas, as primeiras inserções eram realizadas em posições mais distantes, e que perto de ótimos locais realizavam-se apenas inserções em pequenas distâncias. Uma observação similar a esta foi feita por Adenso_Díaz(1992). Partindo deste comportamento, decidiu-se gerar um corte cíclico, que permite grandes distâncias no começo, e depois, de acordo com o número de iterações decorridas diminua esta distância. Ao alcançar **n** iterações, o processo de corte se reinicia permitindo novamente atingir soluções distintas. Foram testadas várias relações entre a distância máxima permitida e o número de iterações, e a que apresentou melhores resultados é dada pela expressão a seguir:

distância máxima = - número de iterações + n

Quando a distância máxima permitida for zero seleciona-se a solução tabu com melhor valor da função objetivo. Esta redução da vizinhança pode ser considerada como uma lista de candidatos. A Figura 3.6 ilustra o corte e o comportamento da Busca Tabu básica para um problema com 20 tarefas para *flowshop* com buffer ilimitado. Observa-se que na maioria das iterações o corte permite distâncias superiores à requerida pela Busca Tabu.

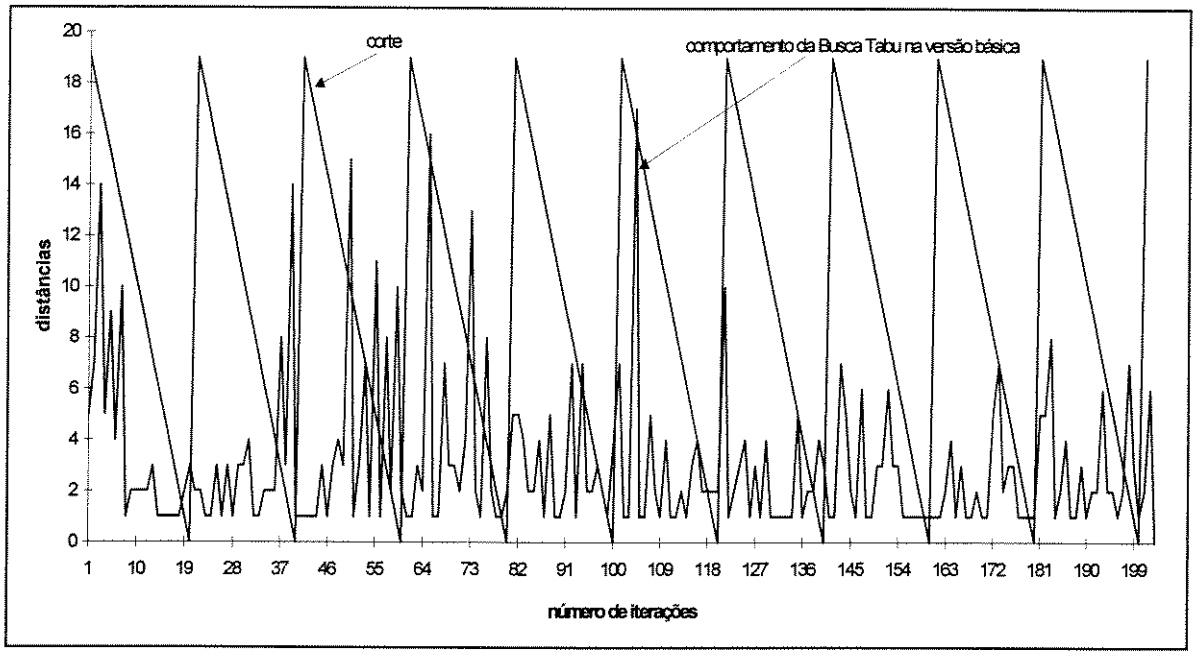


Figura 3.6. Exemplo do corte para um problema com 20 tarefas.

A porcentagem da vizinhança que é reduzida a cada **n** iterações pode ser calculada como descrito a seguir. Considere uma seqüência de tarefas $t_1, t_2, \ldots, t_i, \ldots, t_n$ onde t_i representa a tarefa na i-ésima posição da seqüência, e possui as possibilidades de inserção ilustradas na Figura 3.7. Os valores associados aos arcos indicam a distância entre a posição atual da tarefa e a sua futura posição.

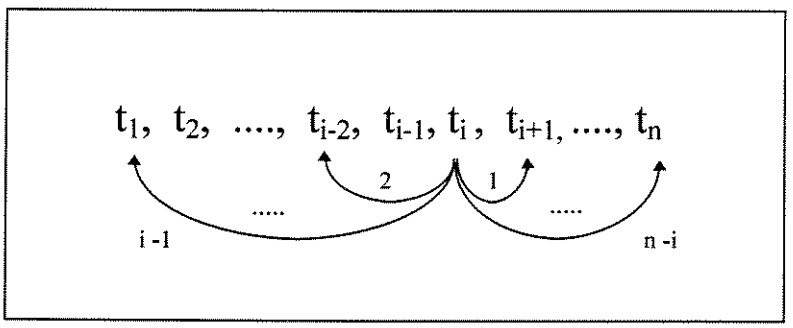


Figura 3.7. Inserções e suas respectivas distâncias.

Considera-se então que cada tarefa t_i possui as seguintes distâncias de inserção dentro da sua vizinhança:

$\mathbf{dist_{i1} = 1,2,..., n-i;}$

$\mathbf{dist_{i2} = 2,..., i-1;}$

onde $dist_{i1}$ e $dist_{i2}$ são, respectivamente, os conjuntos das distâncias para inserção da tarefa t_i nas posições posteriores e anteriores à atual posição. Na primeira iteração, a distância máxima permitida é **n-1** e obviamente nenhuma inserção seria eliminada pelo corte. Na segunda iteração, a distância máxima permitida é **n-2** e o corte eliminaria uma inserção da tarefa $\mathbf{t_1}$ e uma inserção da tarefa $\mathbf{t_n}$, dado que a abrangência das distâncias de suas inserções supera este valor, ou seja:

$\mathbf{dist_{11} = 1,2,..., n-1;}$

$\mathbf{dist_{n2} = 2,..., n-1.}$

Na terceira iteração, a distância máxima permitida é **n-3** e o corte eliminaria as inserções anteriores e todas as inserções cujas distâncias superem este valor, ou seja eliminar-se-ia 6 inserções, conforme explicitado a seguir:

$\mathbf{dist_{11} = 1,2,..., n-2, n-1;}$

$\mathbf{dist_{21} = 1,2,..., n-2;}$

$\mathbf{dist_{n2} = 2,..., n-2, n-1;}$

$\mathbf{dist_{n-1,2} = 2,..., n-2.}$

Na k-ésima iteração, com $\mathbf{k < n}$, a distância máxima permitida é de **n-k**, e o corte eliminaria as inserções anteriores e todas as inserções cujas distâncias superem este valor conforme explicitado a seguir:

$\mathbf{dist_{11} = 1,..., n-k, n-k+1, ..., n-1;}$

.

.

$\mathbf{dist_{k-1,1} = 1,..., n-k+1;}$

$\mathbf{dist_{n2} = 2,..., n-k, n-k+1, ..., n-1;}$

.

.

$\mathbf{dist_{n-k+2,2} = 2,..., n-k+1.}$

Para o conjunto de distâncias do tipo 1 são eliminadas pelo corte **k-1** inserções, do segundo conjunto **k-2**, até o último conjunto desde tipo onde é eliminada uma única inserção. O mesmo pode ser observado no conjunto de distâncias do tipo 2. Logo, tem-se que para uma iteração **k** são eliminadas $\mathbf{2.\sum_{x=1}^{k-1} x}$, expressão esta que pode ser simplificada para **k.(k-1)**. Conseqüentemente, em **n** iterações o total de inserções eliminadas é de:

$$\sum_{k=1}^{n-1} k.(k-1) \quad + \quad (n-1)^2$$

A expressão acima pode ser reduzida a:

$$\frac{n^3 - 4.n + 3}{3}$$

Dividindo-se a expressão anterior pelo total da vizinhança em **n** iterações pode-se calcular a porcentagem da redução da vizinhança obtida com o corte proposto, ou seja:

$$\textbf{Redução \%} = \frac{n^3 - 4.n + 3}{3.n^3 - 6.n^2 + 3.n}.100$$

Com o intuito de se ter noção da dimensão desta redução, avaliou-se esta porcentagem para problemas com 5, 20, 50 e 100 tarefas que resultaram, respectivamente, em 45%, 36,6%, 34,6% e 34%.

- **Aplicação do Corte para *Flowshop* com Buffer Ilimitado**

Apresenta-se na Tabela 3.13 os resultados obtidos por esta redução no conjunto de 240 problemas. Pode ser observado que esta estratégia obteve um aumento na porcentagem de melhoria em 62,50% das classes dos problemas, e manteve a melhoria em 4,16% das classes em relação à versão básica. Diversas combinações desta estratégia com as estratégias de diversificação e intensificação foram avaliadas, porém nenhuma destas apresentou um melhor desempenho que a utilização da redução de vizinhança isoladamente.

Tabela 3.13. Melhoria percentual de Busca Tabu com redução da vizinhança.

Dimensão (n x m)	Cenários			
	1	2	3	4
20x5	77,47	10,78*	21,88*	17,69*
20x10	32,81	12,98*	10,29*	9,71*
20x20	10,78	8,36*	5,46	5,52*
50x5	**96,37***	**79,03**	29,36	24,02*
50x10	**86,11***	37,51*	26,11*	17,84
50x20	46,14*	17,90*	10,00	12,35
Média	58,28	27,76	17,18	14,52
Desvio Padrão	30,68	24,85	9,01	6,06

Pode ser observado que nos cenários 2 e 4 a média de melhoria desta estratégia é superior não só à versão básica, como também à versão com diversificação e intensificação, porém, a mesma é inferior à versão básica no cenário 3. Para aprimorar esta análise, apresenta-se na Tabela 3.14 a comparação desta estratégia com a versão básica e com a versão com diversificação e intensificação através do número de sucessos.

Tabela 3.14. Comparações entre o corte e outras estratégias para *flowshop* com buffer ilimitado.

Versão	Sucessos da versão	Melhoria % média da versão em relação ao corte	Maior %	Sucessos do corte	Melhoria % média do corte em relação à versão	Maior %	Empates
básica	70	4,36	64,29	80	4,21	89,29	90
diver. + inten.	83	4,33	64,29	59	4,42	89,29	98

Através dos resultados desta tabela confirma-se o desempenho do corte em relação à versão básica, ou seja, a estratégia do corte obtém mais sucessos que a versão básica. Quanto a associação da diversificação com a intensificação, esta apresenta um maior número de sucessos que o corte. Cabe destacar que a estratégia do corte chega a obter, no maior caso, uma solução com valor 89,29% melhor que o obtido pela básica e pela diversificação associada à intensificação, e por outro lado, o maior caso onde as duas versões superam o corte alcança um valor 64,29% melhor. Desta análise pode-se concluir que o uso da estratégia do corte pode acarretar resultados muito melhores que as duas versões comparadas, mas que também pode resultar em soluções substancialmente inferiores.

Com o intuito de testar a eficácia da estratégia com relação à manutenção das médias de melhoria com tempos menores, avaliou-se a mesma com um tempo de execução 40% menor. Os resultados são apresentados na Tabela 3.15. Nota-se desta tabela que houve uma perda na qualidade da solução, porém dada a redução obtida no tempo, esta pode ser considerada mínima. Para se ter uma base de comparação, obteve-se também as melhorias apresentadas por Busca Tabu básica em um tempo de execução 40% menor. Os resultados são apresentados na Tabela 3.16.

Tabela 3.15. Melhoria percentual da redução de vizinhança com tempo de execução 40% menor

Dimensão (n x m)	Cenários			
	1	2	3	4
20x5	77,14*	10,65	20,97	17,66*
20x10	32,78	12,37	10,26*	9,66*
20x20	10,06*	8,36*	5,25*	5,41*
50x5	**92,92**	**79,03**	28,84*	22,01*
50x10	86,57*	36,65*	23,62*	17,79*
50x20	44,49*	17,41*	9,73*	11,92*
Média	57,33	27,41	16,45	14,08
Desvio Padrão	30,32	24,90	8,51	5,61

* classe de problemas onde a estratégia avaliada supera a Busca Tabu básica com tempo 40% menor.

Tabela 3.16. Melhoria percentual de Busca Tabu básica com tempo de execução 40% menor

Dimensão (n x m)	Cenários			
	1	2	3	4
20x5	76,73	10,65	21,75	16,99
20x10	34,22	12,48	9,73	9,65
20x20	9,97	8,14	5,05	5,16
50x5	**92,92**	**79,03**	28,15	20,70
50x10	85,60	36,60	23,52	16,45
50x20	43,39	16,18	9,70	11,18
Média	56,87	27,18	16,32	13,36
Desvio Padrão	29,79	25,00	8,52	5,20

Observa-se na Tabela 3.16 que a Busca Tabu básica com uma redução no tempo de 40% apresenta uma maior queda das melhorias em relação à versão básica no tempo normal, do que a versão com corte de vizinhança com uma redução no tempo de 40% em relação à básica. Além disso, ressalta-se que 75% das classes dos problemas desta tabela possuem resultados inferiores à Busca Tabu com redução de vizinhança, o que ressalta a boa qualidade do corte e a validade de sua utilização com o intuito de reduzir o tempo.

- **Aplicação do Corte para *Flowshop* com Buffer Zero**

Na Tabela 3.17 são apresentados os resultados obtidos por esta redução para 240 problemas.

Tabela 3.17. Melhoria percentual de Busca Tabu com redução da vizinhança.

Dimensão (n x m)	Cenários			
	1	2	3	4
20x5	54,79*	**36,90**	19,88*	18,81
20x10	19,81*	24,54	11,23*	9,19*
20x20	9,38	6,71	5,43	4,70
50x5	65,81	**92,09***	31,60*	27,54
50x10	43,19*	64,10*	19,88	15,61
50x20	29,67	26,99*	11,11*	7,59
Média	37,11	41,89	16,52	13,91
Desvio Padrão	19,57	28,28	8,47	7,74

Nota-se que esta estratégia obteve um aumento na porcentagem de melhoria em 45,83% das classes dos problemas, além de melhorar a porcentagem de melhoria dos cenários 2 e 3 em relação à versão básica. Destaca-se porém, que em 54,17% das classes a melhoria diminuiu. Apresenta-se na Tabela 3.18 a comparação desta estratégia com a versão básica e com a versão com diversificação e intensificação através do número de sucessos.

Tabela 3.18. Comparações entre o corte e outras estratégias para *flowshop* com buffer zero.

Versão	Sucessos da versão	Melhoria % média da versão em relação ao corte	Maior %	Sucessos do corte	Melhoria % média do corte em relação à versão	Maior %	Empates
básica	78	2,92	31,60	81	3,47	38,41	81
diver. + inten.	81	3,11	31,60	77	3,04	36,43	82

Através dos resultados desta tabela nota-se que a estratégia do corte obtém resultados semelhantes aos da versão básica e da associação da diversificação com a intensificação, o que torna a seleção entre as estratégias uma tarefa difícil. Uma possibilidade é escolher a estratégia de acordo com a classe a qual pertence o problema a ser resolvido.

Avaliou-se a estratégia do corte com um tempo de execução 40% menor. Os resultados são apresentados na Tabela 3.19. Para se ter uma base de comparação, obteve-se também as melhorias apresentadas por Busca Tabu na versão básica em um tempo de execução 40% menor. Os resultados são apresentados na Tabela 3.20.

Tabela 3.19. Melhoria percentual da redução de vizinhança com tempo de execução 40% menor

Dimensão (n x m)	Cenários			
	1	2	3	4
20x5	53,35	**36,90**	19,58	18,81*
20x10	18,61	24,43	10,90*	9,02*
20x20	9,22*	6,71*	5,43*	4,53
50x5	64,34	**90,82**	30,63*	26,86*
50x10	40,36	63,59*	19,49	15,27
50x20	28,88*	25,56*	10,39*	7,07
Média	35,79	41,34	16,07	13,60
Desvio Padrão	19,12	27,98	8,25	7,66

* classe de problemas onde a estratégia avaliada supera a Busca Tabu básica com tempo 40% menor.

Tabela 3.20. Melhoria percentual de Busca Tabu básica com tempo de execução 40% menor

Dimensão (n x m)	Cenários			
	1	2	3	4
20x5	53,47	**37,06**	19,60	18,67
20x10	19,10	24,82	10,54	8,75
20x20	9,06	6,66	5,40	4,59
50x5	65,15	**90,82**	29,28	26,20
50x10	40,75	60,41	19,84	15,57
50x20	28,77	24,97	9,46	7,10
Média	36,05	40,79	15,69	13,48
Desvio Padrão	19,33	27,60	8,05	7,47

Através destas tabelas nota-se que as médias de melhoria do cenário 2, 3, e 4 são maiores na versão com redução de vizinhança, e que o uso desta estratégia com o tempo 40% menor acarretou melhores ou iguais resultados em relação à versão básica, com o mesmo tempo, em 54,17% das classes. Logo, o uso da estratégia de redução de vizinhança é válido para um tempo 40% menor.

3.5. Análise Geral dos Resultados

Para o problema de *flowshop* com buffer ilimitado a aplicação das estratégias de diversificação e intensificação apresentaram pequenos ganhos em relação à Busca Tabu básica, porém a combinação destas duas estratégias obteve melhorias ou manteve a média em 79,17% das classes dos problemas, além de obter no melhor caso uma melhoria de 30,41% sobre a versão básica. Destaca-se o fato destas

melhorias terem sido obtidas no mesmo tempo de execução que a Busca Tabu na versão básica. Tais dados levam a concluir que a aplicação desta estratégia é válida. Ressalta-se que o uso da estratégia do corte pode proporcionar soluções substancialmente melhores em alguns problemas.

Para o problema de *flowshop* com buffer zero, a associação da diversificação com a intensificação e a estratégia do corte obtiveram um desempenho global regular, mas não conclusivo sobre a superioridade de uma delas.

Dos testes apresentados neste capítulo comprovou-se a afirmação de Glover(1995) que o uso da memória de longo prazo não necessita de longas execuções para que seu benefício se torne visível. Todas as melhores estratégias apresentadas aqui foram avaliadas no dobro do tempo de execução, e nenhuma apresentou resultados significativos com o aumento do tempo.

Cabe destacar que a diversificação associada à intensificação obteve bons resultados em termos do aumento da qualidade das soluções para os dois problemas. A estratégia de redução da vizinhança, utilizada com o intuito de diminuir o tempo necessário para obter soluções de boa qualidade, também apresentou um bom desempenho. Dado que estas estratégias são excludentes, a escolha de uma delas depende do compromisso entre o tempo e a qualidade.

Embora a estratégia de diversificação associada à intensificação tenha obtido resultados positivos, não deve ser desconsiderado que para estabelecer os parâmetros da mesma foram necessários inúmeros testes. Dado que as melhorias médias foram pequenas, questiona-se a aplicação de tais estratégias em problemas com características similares ao do *flowshop* de permutação, ou seja, diversos problemas nos quais se deseja achar a melhor permutação de **n** objetos, e com acesso à vizinhanças ricas como a gerada pelo movimento da inserção. Acredita-se que possivelmente estas estratégias possuam um desempenho mais relevante em problemas com vizinhanças mais restritas como, por exemplo, o *jobshop*.

No próximo capítulo são apresentados métodos ótimos para os problemas de *flowshop* com buffer ilimitado e zero.

Capítulo 4
Métodos Ótimos

4.1. Introdução

Neste capítulo são apresentados métodos ótimos baseados na técnica *Branch-and-Bound* para os problemas estudados. Esta técnica, que pode ser sucintamente definida como uma enumeração implícita, é muito utilizada para resolver problemas de otimização combinatória. No entanto, só problemas pequenos podem ser resolvidos em um tempo computacional viável.

Apresenta-se na seção 4.2 o método proposto por Kim(1995) para o problema *flowshop* com buffer ilimitado. Este método é implementado para obter valores ótimos que sirvam como base de comparação para as heurísticas propostas. Na seção 4.3 apresenta-se a proposta de adaptação deste *Branch-and-Bound* para o problema de *flowshop* com buffer zero. A principal modificação se concentra na proposta de um limitante inferior para o problema em questão. Neste caso, o objetivo é analisar o desempenho de um método ótimo capaz de solucionar este problema, e também avaliar a qualidade das soluções fornecidas pela heurística.

4.2. Método *Branch-and-Bound* para *Flowshop* com buffer ilimitado

A metodologia e os componentes da técnica *Branch-and-Bound* explicitados a seguir são propostos por Kim(1995). O processo de resolução do *Branch-and-Bound* pode ser representado através de uma árvore. Para este problema, cada nó da árvore corresponde a um subproblema que é definido por uma seqüência de tarefas já alocadas, denominada de seqüência parcial (SP), e por um conjunto de tarefas não pertencentes à seqüência parcial (NSP), ou seja, tarefas não alocadas. O método começa com o nó zero na árvore, onde nenhuma tarefa está alocada, e com o valor de uma solução incumbente. Este nó é ramificado de forma a criar **n** nós nos quais cada uma das **n** tarefas

disponíveis é alocada na n-ésima posição da seqüência. A Figura 4.1 ilustra um exemplo de uma árvore parcial para um problema com 5 tarefas, onde os números nos nós representam as seqüências das tarefas alocadas. Observe que quando um nó é ramificado, geram-se novos nós adicionando-se uma tarefa no começo da seqüência parcial associado ao nó ramificado. Através da figura nota-se que os nós provenientes do nó 3 apresentam as possibilidades de alocação para a quarta posição.

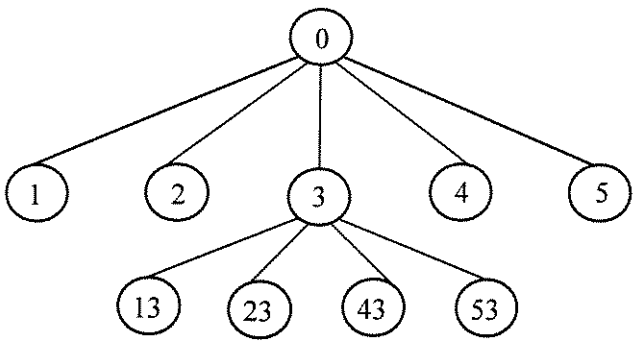


Figura 4.1. Exemplo da árvore parcial gerada pelo *Branch-and-Bound* para um problema com 5 tarefas.

Para cada um dos nós gerados calcula-se um limitante inferior da soma dos atrasos de todas as tarefas. Os nós com limitantes inferiores maiores que o valor da solução incumbente são descartados. Seleciona-se para a próxima ramificação o nó com o maior número de tarefas na seqüência parcial. No caso de empate, o método seleciona o nó com o menor limitante inferior. Esta estratégia de ramificação é conhecida por busca em profundidade. O processo de ramificação e de cálculo dos limitantes continua enquanto a árvore possuir nós com limitantes inferiores menores que o valor da solução incumbente. Caso a busca se encontre em um nó com todas as tarefas fixadas, calcula-se o valor real da soma dos atrasos relativo à esta seqüência. Caso este valor seja inferior ao valor da solução incumbente, esta deve ser atualizada.

O limitante de cada nó consiste de dois limitantes que são calculados de forma independente, um para o conjuntos de tarefas de SP, e outro para as tarefas de NSP. A soma destes dois limitantes gera o limitante inferior do nó em questão. Cabe destacar que nesta metodologia aplica-se ao nó zero uma regra de dominância que elege tarefas que devem ser excluídas e alocadas ao fim de qualquer seqüência. Estas tarefas são desconsideradas pelo método *Branch-and-Bound*

- **Limitante inferior da soma dos atrasos para as tarefas na seqüência parcial (SP)**

As tarefas em SP, de um nó qualquer, estão alocadas no fim da seqüência completa, depois das tarefas em NSP. Portanto, para calcular a soma dos atrasos destas tarefas, é necessário que se conheça os instantes de tempo nos quais as tarefas em NSP liberam as **m** máquinas. Porém, não se sabe a

ordem das tarefas em NSP, e portanto o instante exato de término destas tarefas nas máquinas não pode ser determinado. Um limitante inferior do instante de liberação de cada máquina **k** pelas tarefas en NSP, proposto por Kim(1995), é dado pela expressão (1) abaixo.

$$\max_{1\le j\le k}\left\{\sum_{i\notin SP} p_{ij} + \sum_{z=1}^{k} p_{\min_z} - p_{\min_j}\right\}, \qquad (1)$$

onde,

$$p_{\min_z} = \min_{i\notin SP} p_{iz}\,.$$

Este limitante pode ser representado em um grafo direcionado onde os nós significam o processamento das tarefas nas máquinas, e os arcos representam as relações de precedência entre as tarefas nas máquinas, e entre as operações nas tarefas. Suponha que existam 5 tarefas não pertencentes a SP em um *flowshop* com 4 máquinas. A Figura 4.2 ilustra uma seqüência arbitrária representada por um grafo onde t_i significa a tarefa na i-ésima posição, M_j a j-ésima máquina, e p_{ij} o tempo de processamento de t_i na máquina M_j.

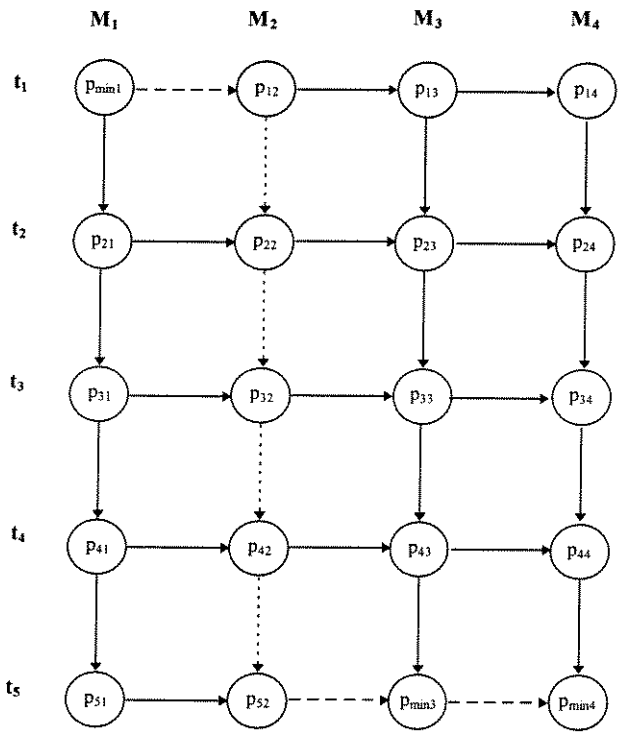


Figura 4.2. Exemplo do cálculo parcial do limitante através de um grafo.

Suponha também que almeja-se calcular o limitante inferior do instante de liberação da última máquina M_4 . Para este exemplo, o limitante é analisado sob o referencial da máquina 2, ou seja,

dentro da expressão (1) faça j =2. A primeira parcela do limitante representa a soma de todos os tempos de processamento das tarefas não pertencentes a SP na máquina M_2; esta parcela é representada pelos arcos pontilhados da Figura 4.2. A segunda e a terceira parcelas são a soma dos processamentos mínimos que M_2 terá que esperar para iniciar o processamento da primeira tarefa, e para que a quinta tarefa processada por M_2 seja liberada por M_4. Estas parcelas são representadas pelos arcos tracejados da figura. Para este exemplo, a expressão (1) deve ser calculada para j = 1, 3 e 4 de maneira análoga ao cálculo sob o referencial da máquina 2, e o seu valor máximo é o limitante inferior do instante de liberação da máquina 4.

Aplicando-se a expressão (1) a um caso genérico, obtém-se os instantes de liberação das **m** máquinas após o processamento das tarefas em NSP. Com tal resultado torna-se viável o cálculo do limitante inferior dos instantes de término das tarefas em SP. Para este cálculo são utilizadas as fórmulas recursivas apresentadas no capítulo 1 partindo dos instantes nos quais as máquinas são liberadas pelas tarefas em NSP. Estas fórmulas são descritas a seguir.

$$C_{i1} = LC_1 + p_{i1} \qquad i = |NSP| + 1;$$

$$C_{ij} = \max(C_{i,j-1}, LC_j) + p_{ij} \qquad i = |NSP| + 1;\ j = 2,...,m;$$

$$C_{i1} = C_{i-1,1} + p_{i1} \qquad i = |NSP| + 2, ...,n;$$

$$C_{ij} = \max(C_{i,j-1}, C_{i-1,j}) + p_{ij} \qquad i = |NSP| + 2, ...,n\ ;\ j = 2,...,m;$$

onde C_{ij} representa o instante de término de t_i em M_j, |NSP| representa o número de tarefas neste conjunto, e LC_j é o limitante inferior do instante de término das tarefas em NSP na máquina j. Calculados os instantes de término das tarefas em SP na máquina **m,** obtém-se a soma dos atrasos individuais baseados nas datas de entrega de cada tarefa.

- **Limitante inferior da soma dos atrasos para as tarefas que não pertencem a seqüência parcial (NSP)**

A expressão (2) representa o limitante inferior para a soma dos atrasos das tarefas de NSP que não estão seqüenciadas.

$$\sum_{i=1}^{n-|SP|} \left[\max_{1 \le j \le m} \left\{ O_j(i) + \sum_{z=1}^{m} p_{min_z} - p_{min_j} \right\} - d_{S_2(i)} \right]^+ \qquad (2)$$

onde S_2 é a seqüência que possui as tarefas de NSP em ordem não-decrescente de datas de entrega, ou

seja, ordenadas de acordo com a regra de despacho EDD, ds_2 (i) é a data de entrega da tarefa na posição i da seqüência S_2, O_j(i) é a soma dos i menores tempos de processamento na máquina j dentre todas as tarefas de NSP, |SP| representa o número de tarefas em SP, e $x^+ = \max (x, 0)$.

A parcela entre colchetes desta expressão representa um limitante inferior do instante de término de uma dada tarefa na posição i de NSP na máquina **m**. Esta expressão é obtida de forma similar ao limitante inferior do instante de liberação de cada máquina pelas tarefas em NSP descrito no item anterior. A Figura 4.3 ilustra uma seqüência arbitrária representada por um grafo supondo que existam 5 tarefas em NSP em um *flowshop* com 4 máquinas. Suponha também que almeja-se calcular o limitante inferior do instante de término da tarefa t_3. Para este exemplo, esta parcela é analisada sob o referencial da máquina 2.

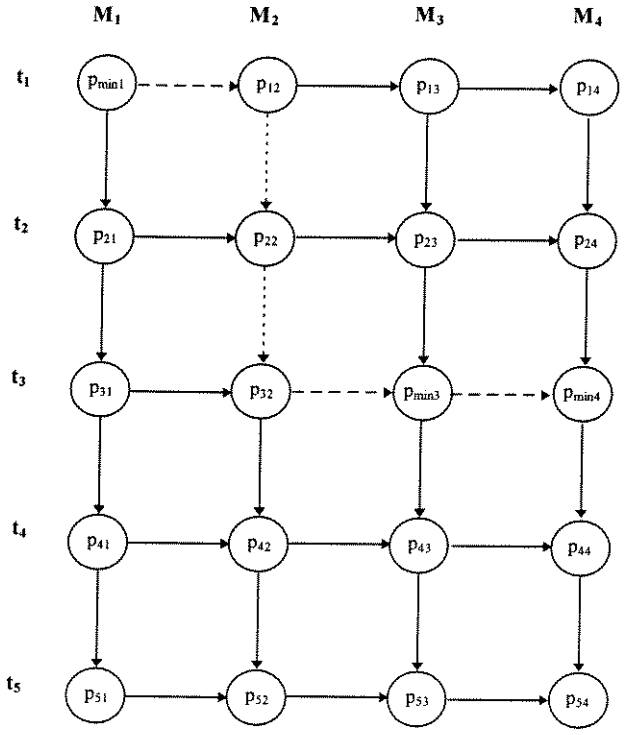


Figura 4.3. Exemplo do cálculo parcial do limitante através de um grafo.

O primeiro componente desta parcela representa a soma dos 3 menores tempos de processamento das tarefas em NSP na máquina M_2, representados por p_{12}, p_{22}, e p_{32} . Esta parcela é representada pelos arcos pontilhados da figura. O segundo e o terceiro componentes são a soma dos processamentos mínimos que M_2 terá que esperar para iniciar o processamento da primeira tarefa, e para que a terceira tarefa processada por M_2 seja liberada por M_4. Estes componentes são representados pelos arcos tracejados da figura. Para este exemplo, esta parcela deve ser calculada para j = 1, 3 e 4 de maneira análoga à máquina 2, e o seu valor máximo é o limitante inferior do instante de término da tarefa t_3.

A validade deste limitante foi demonstrada na seguinte proposição apresentada por Kim(1995).

Proposição: Sejam R_i e d_i dois números não negativos e reais correspondentes às tarefas i para i = 1,2,...,n, S a seqüência das tarefas {1, 2, ..., n}, e S(i) a i-ésima tarefa da seqüência S. Seja S_1 uma seqüência ordenada de acordo com a regra EDD, e defina S_1 (i) de maneira análoga a S(i). Logo tem-se que $ds_i(i) \leq ds_i(i+1)$ para i = 1,2, ..., n-1. Se $R_1 \leq R_2 \leq ..... \leq R_n$, tem-se para qualquer seqüência S que:

$$Z = \sum_{i=1}^{n} \left(R_i - d_{S(i)}\right)^+ \geq \sum_{i=1}^{n} \left(R_i - d_{S_1(i)}\right)^+ .$$

Demonstração: Assuma que Z é minimizado por uma seqüência S na qual existe pelo menos um par de tarefas adjacentes tal que $R_i \leq R_j$ e ds(i) > ds(j). Pode ser demonstrado que a troca da ordem de S(i) e S(j) não aumenta Z. Seja $T = (R_i - ds(i))^+ + (R_j - ds(j))^+$ e $T^* = (R_i - ds(j))^+ + (R_j - ds(i))^+$. Dado que $R_i \leq R_j$ e ds(i) > ds(j), existem seis diferentes casos que relacionam estes quatro números:

- $ds(i) \geq R_j \geq R_i \geq ds(j)$;
- $R_j \geq R_i \geq ds(i) > ds(j)$;
- $R_j \geq ds(i) \geq R_i \geq ds(j)$;
- $ds(i) \geq R_j \geq ds(j) \geq R_i$;
- $R_j \geq ds(i) > ds(j) \geq R_i$;
- $ds(i) > ds(j) \geq R_j \geq R_i$;

Comparado-se os valores de T e T^* para o primeiro caso onde $ds(i) \geq R_j \geq R_i \geq ds(j)$, tem-se que:

$T = (R_i - ds(i))^+ + (R_j - ds(j))^+ = R_j - ds(j)$;

$T^* = (R_i - ds(j))^+ + (R_j - ds(i))^+ = R_i - ds(j)$.

Como $R_j \geq R_i$ tem-se que $T \geq T^*$. Nos outros casos citados anteriormente prova-se de forma análoga que $T \geq T^*$. Logo, a troca de S(i) e S(j) não aumenta Z. Continuando esta troca de tarefas pode-se obter a seqüência S_1 que minimiza Z.

Fazendo $R_i = C_i$, onde C_i é o instante de término da tarefa alocada na i-ésima posição na máquina **m** em uma seqüência arbitraria S em um flowshop de permutação, obviamente $C_1 \leq C_2 \leq ..... \leq C_n$. Então, da proposição acima, sabe-se que o atraso total da seqüência S é maior ou igual a:

$$\sum_{i=1}^{n} \left(C_i - d_{S_1(i)}\right)^+,$$

onde $d_{S1(i)}$ é a data de entrega da i-ésima tarefa e S_1 é uma seqüência ordenada de acordo com a regra EDD.

- **Regra de dominância**

Algumas seqüências podem ser eliminadas no nó zero por serem dominadas por outras seqüências. Suponha que as datas de entrega de algumas tarefas sejam maiores que um limitante superior do *makespan*, então estas tarefas podem ser alocadas no final de qualquer seqüência, dado que o atraso das mesmas será zero. Em outras palavras, as seqüências com estas tarefas no final dominam outras seqüências. A expressão (3) fornece um limitante superior do *makespan* apresentado por Kim(1995).

$$\min\left[\left\{\sum_{i=1}^{n} p_{i(max)} + (m-1)\max_i p_{i(max)}\right\}, \left\{\sum_{j=1}^{m} p_{(max)j} + (n-1)\max_j p_{(max)j}\right\}\right] \quad (3)$$

onde:

$$p_{i(max)} = \max_j p_{ij} \qquad e \qquad p_{(max)j} = \max_i p_{ij}$$

A seguir, o procedimento de verificação pode ser repetido eliminando-se as tarefas já fixadas. Este procedimento continua até que não se encontre nenhuma tarefa sem atraso. Estas tarefas podem ser removidas do problema, e o método *Branch-and-Bound* é aplicado ao problema com as tarefas restantes. Detalhes sobre a comprovação da validade deste limitante pode ser encontrado em Kim(1995).

- **Passos básicos**

Apresenta-se a seguir os passos básicos utilizados na aplicação do método *Branch-and-Bound*. Para tal considera-se a seguinte notação:

- Zinc : valor da solução incumbente;
- LI: limitante inferior da soma dos atrasos para cada nó;
- LA: menor limitante inferior da soma dos atrasos dentre toda árvore;

- tolerância: diferença percentual aceitável entre Zinc e LA. Quando o LA possui valor zero esta tolerância não é considerada.

```
- leitura de Zinc.
- eliminação de tarefas através da regra de dominância.
- inicialização do nó zero.
- colocação do nó zero na lista.
- LA = 0
- profundidade do nó zero = 0
- faça enquanto (lista não estiver vazia) e (tempo < tmax) e (100.(Zinc - LA)/LA≥ tolerância))
    - selecione o nó com maior profundidade e menor LI.
    - se (LI=LA) mudou_LA=1
    - mudou_Zinc=0
    - construa o conjunto preliminar das tarefas não alocadas.
    - faça enquanto existir filhos para ramificar
        - crie o filho através da ramificação do nó-pai.
        - profundidade do filho= profundidade do pai + 1
        - se (profundidade do filho = n -1)
            - calcule a soma dos atrasos da seqüência completa ( T ).
            - se (Zinc > T)
                - Zinc=T
                - mudou_Zinc=1
                - se (100.(Zinc - LA)/LA < tolerância): PARE
            - senão
                - elimine este nó filho.
            - fim_se
        - senão
            - calcule LI.
            - se (100.(Zinc - LI)/LI < tolerância)) /* eliminação de nós */
                - elimine este nó filho.
            - senão
                - coloque o nó na lista.
            - fim_se
        - fim_se
    - fim_enquanto
    - retire o nó pai da lista.
    - se (mudou_LA=1)
        - busque em toda a lista o menor LA.
        - mudou_LA=0
    - fim_se
    - se (mudou_Zinc = 1)
        - teste em toda lista se algum nó foi eliminado.
    - fim_se
- fim_enquanto
```

Uma outra possibilidade para a construção da árvore do *Branch-and-Bound* seria a fixação das tarefas nas posições iniciais. Tal implementação poderia ser realizada utilizando-se um limitante inferior semelhante ao limitante da expressão (2). Quando se fixa primeiramente as tarefas na posições iniciais pode-se calcular de forma exata seus instantes de término e, conseqüentemente, através de suas datas de entrega, calcular também de forma exata seus atrasos. Porém, provavelmente, estas tarefas possuiriam atraso zero ou muito pequeno. As tarefas que potencialmente gerariam atraso, ou seja as últimas, possuirão um cálculo de atraso estimado, baseado nas datas ordenadas de acordo com a regra EDD, e provavelmente contribuirão de forma bem mais fraca do que se seu cálculo pudesse ser realizado com as verdadeiras datas. Por exemplo, a última tarefa não fixada teria uma estimativa de atraso baseada na maior data de entrega dentre as tarefas não alocadas. Por outro lado, ao se fixar as tarefas finais estas possuirão o cálculo da estimativa dos seus atrasos baseada nas datas reais. Este é o motivo para se elaborar uma metodologia que fixe as tarefas, inicialmente, no fim da seqüência. Cabe destacar que com limitantes maiores, a probabilidade de se descartar nós é maior, e conseqüentemente a árvore de busca é menor. Embora este comentário não seja feito por Kim(1995), acredita-se que esta seja a razão pela qual o autor selecionou esta metodologia.

- **Resultados Computacionais**

Esta técnica foi aplicada aos mesmos 72 problemas utilizados no capítulo 2, ou seja uma amostragem de 3 matrizes de tempo de processamento para cada uma das 6 dimensões, e para cada matriz, 4 cenários. Os testes foram realizados numa estação de trabalho SUN modelo SPARCstation 5, com 110MHz de *clock*, 32Mbytes de RAM, e sistema operacional SOLARIS 2.5.1. O tempo de execução do *Branch-and-Bound* foi limitado em 12 horas de C.P.U. para cada problema. A tolerância utilizada foi de 5%.

Utilizou-se como soluções iniciais as melhores soluções fornecidas por todas as variações testadas para Busca Tabu, incluindo todas as estratégias de diversificação, intensificação, redução da vizinhança e combinações destas. Em nenhum problema o *Branch-and-Bound* encontrou solução melhor que a solução heurística inicial. Dentre os 72 problemas testados, foram encontradas 10 soluções ótimas com tolerância de 0,5%; todos estes problemas pertencem ao cenário 1 e 2. Este fato é coerente com a análise de comportamento realizada através da enumeração completa (capítulo 1), onde observou-se que estes cenários possuem um menor número de soluções próximas do ótimo. Partindo do pressuposto que o *Branch-and-Bound* parte de uma boa solução incumbente e que a maioria das soluções possui valores muito acima do ótimo, conclui-se que a maior parte da árvore de busca será

eliminada através da comparação com a solução incumbente, e conseqüentemente aumenta-se a probabilidade de se encontrar o ótimo.

Dada a dificuldade em obter-se soluções ótimas para problemas com as dimensões utilizadas no conjunto original de testes, gerou-se um novo conjunto com dimensões menores para a realização de novos testes. A estes problemas aplicou-se a técnica *Branch-and-Bound* com tolerância de 0,5% e solução inicial fornecida pela Busca Tabu básica. A Tabela 4.1 mostra as novas dimensões geradas e os tempos de C.P.U. para a execução do *Branch-and-Bound*. O esquema de geração foi mantido, ou seja, para cada matriz de tempos de processamento foram gerados 4 cenários, num total de 216 problemas.

Tabela 4.1. Tempos em segundos utilizados para as novas dimensões.

Dimensão (n x m)	Nº. de matrizes	Tempos do *Branch-and-Bound*				Média por dimensão
		Cenário 1	Cenário 2	Cenário 3	Cenário 4	
10 x 5	10	0,66	3,92	6,82	6,06	4,37
10 x 7	10	7,19	11,95	30,53	32,54	20,55
10 x 10	10	24,93	38,39	87,26	92,87	60,86
12 x 5	05	10,21	28,19	89,27	314,90	110,64
12 x 7	05	44,37	275,33	321,04	596,73	309,36
12 x 10	05	182,51	941,75	1425,00	6722,10	2317,84
14 x 5	03	5,31	1710,66	592,01	8170,54	2619,63
14 x 7	03	326,02	5945,04	2178,58	32291,70	10185,33
14 x 10	03	5235,52	77632,27	176212,58	412183,22	167815,89
Média por cenário		648,53	9620,83	20104,79	51156,74	

Analisando-se a média por dimensão apresentada na Tabela 4.1, destaca-se que o acréscimo no número de tarefas possui maior influência no aumento do tempo que o aumento no número de máquinas. Isto se deve ao fato do número de tarefas aumentar diretamente o número de nós da árvore a ser explorada pela técnica *Branch-and-Bound,* além de aumentar o tempo de cálculo de cada nó. O aumento no número de máquinas acarreta um aumento somente no tempo de cálculo de cada nó. Portanto, era previsível que o aumento no número de tarefas ocasionasse um maior aumento no tempo. Analisando-se a linha correspondente às médias por cenários tem-se que o cenário 1 possui a menor média dos tempos, seguido pelos cenários 2,3 e 4. Os valores destas médias reforçam a idéia que os cenários 1 e 2, que apresentam menor fator de atraso, são mais fáceis de serem resolvidos pela técnica *Branch-and-Bound.*

- **Comparações com a heurística**

Nos 216 problemas de menor dimensão descritos anteriormente aplicou-se a Busca Tabu na sua versão básica, com diversificação associada a intensificação, e com a redução de vizinhança com tempo 40% menor. A heurística na versão básica encontrou os valores ótimos em 210 problemas, na versão com redução de vizinhança e tempo 40% menor em 208 problemas e, na versão utilizando diversificação e intensificação em 212 problemas. A associação do algoritmo NEHad com a regra LI (NEHad*) encontrou o valor ótimo em 19 problemas. No pior caso, as heurísticas na versão básica, com redução de vizinhança, e com diversificação e intensificação ficaram respectivamente a 3,9%, 13,53%, e 3,9% do valor ótimo, enquanto o algoritmo NEHad* ficou no pior caso a 440,91% do ótimo. Estes resultados fornecem uma indicativa sobre o comportamento das versões propostas da heurística. Ressalta-se o melhor comportamento da versão que utiliza diversificação associada com a intensificação, e a pequena queda na qualidade das soluções fornecidas pela estratégia de redução da vizinhança. A Tabela 4. 2, apresentada a seguir, contém uma síntese destes resultados.

Tabela 4. 2. Comparações entre os métodos de resolução aplicados a 216 problemas.

Métodos de Resolução	Número de problemas com valor ótimo	Porcentagem de problemas com valor ótimo	Maior porcentagem acima do ótimo
Busca Tabu básica	210	97,2	3,9
Busca Tabu diver. + inten.	212	98,1	3,9
Busca Tabu com corte e tempo 40% menor	208	96,3	13,53
NEHad*	12	8,8	440,91

4.3. MÉTODO *BRANCH-AND-BOUND* PARA *FLOWSHOP* COM BUFFER ZERO

Para este problema utilizou-se a estrutura do método *Branch-and-Bound* descrito na seção anterior, porém com uma proposta de limitante inferior para o instante de término das tarefas em NPS, e conseqüentemente novos limitantes inferiores para a soma dos atrasos para as tarefas em SP e em NSP.

♦ **Limitante inferior da soma dos atrasos para as tarefas que não pertencem à seqüência parcial (NSP)**

Para se calcular este limitante é necessário obter um limitante inferior do instante de término das tarefas em NSP na máquina **m**. Para o cálculo deste limitante para a i-ésima tarefa (t_i) em NSP sugere-se a seguinte expressão:

$$\max_{1 \le j \le m} \left\{ \sum_{k=1}^{m} p\min_{k}^{1} + \sum_{z=2}^{i} \max\left(p\min_{j}^{z}, p\min_{j+1}^{z}\right) \right\} \qquad (4)$$

onde,

$p\min_{k}^{z}$ = z-ézimo menor tempo de processamento na máquina **k** dentre as tarefas $\in$ NPS.

Deste limitante pode ser observado que a primeira parcela é igual para todas as tarefas, o que reduz o seu tempo de cálculo. A Figura 4.4 a seguir, ilustra um grafo representativo do problema com 5 tarefas e 4 máquinas. Os nós (i,j) representam a partida da tarefa i da máquina j, os valores nos arcos horizontais representam o tempo de processamento de cada tarefa em cada máquina, e os arcos inclinados representam o possível bloqueio em uma máquina causado por uma tarefa na máquina posterior à máquina analisada. Observe que estes últimos arcos não possuem pesos. Os nós (i,0) representam o início do processamento da tarefa i na primeira máquina. Esta representação é apresentada por Pinedo(1995). O sentido dos arcos indica a precedência necessária para cada instante de liberação. O limitante inferior pode ser visualizado no grafo da Figura 4.4.

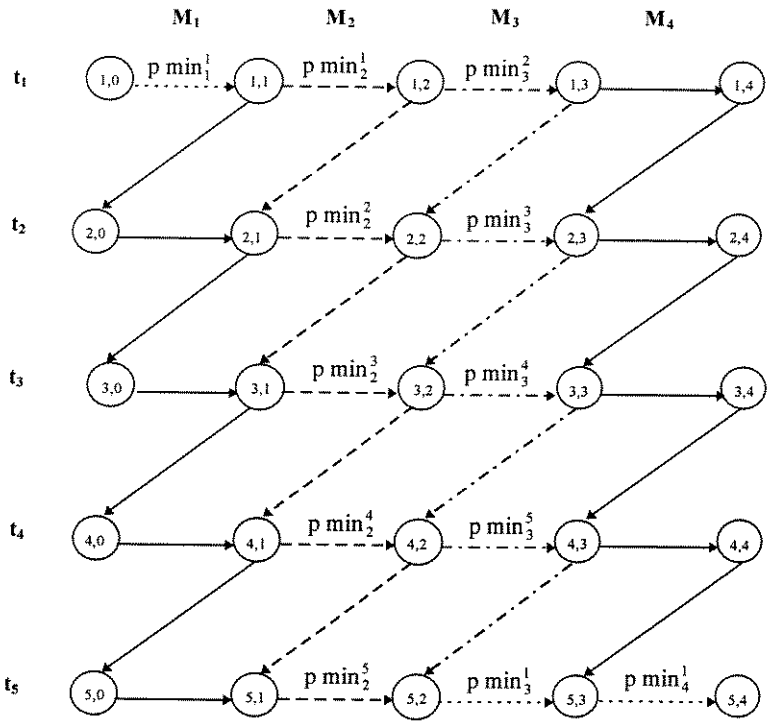


Figura 4.4. Exemplo do cálculo parcial do limitante para buffer zero através de um grafo.

Este limitante é baseado nas máquinas como no *flowshop* com buffer ilimitado. Neste exemplo específico, o mesmo é analisado sob o referencial da máquina M_2, e é relativo ao término na M_4 da tarefa t_5 em NSP. Analisando-se o limitante de acordo com as restrições impostas pelo ambiente e por M_2, sabe-se que:

- o mínimo tempo que M_2 tem que esperar para iniciar a processar qualquer tarefa é o mínimo tempo de processamento de uma tarefa na máquina M_1 (arco pontilhado).
- o mínimo tempo no qual M_2 estará liberada é diretamente associado ao processamento das 5 tarefas que possuam o mínimo tempo de processamento na mesma (arco tracejado).
- além de processar cada uma das 4 últimas tarefas, ao término de cada uma, M_2 terá sempre que esperar que a tarefa com o mínimo tempo de processamento seja processada em M_3, por uma questão de possível bloqueio (arco tracejado e pontilhado).
- depois da partida das 5 tarefas de M_2, tem-se que o tempo mínimo para que t_5 termine é a soma dos tempos mínimos de processamento em M_3 e M_4 (arco pontilhado).

De forma genérica, o limitante inferior para a quinta tarefa pode ser expresso por:

$$\max_{1\le j\le 4}\left\{\sum_{k=1}^{4} p\min_k^1 \;+\; \sum_{z=2}^{5}\max\left(p\min_j^z,\; p\min_{j+1}^z\right)\right\}$$

No caso da máquina M_2, ou seja fazendo j=2, tem-se:

$$\sum_{k=1}^{4} p\min_k^1 + \sum_{z=2}^{5} \max(p\min_2^z, p\min_3^z) ,$$

onde:

$\sum_{k=1}^{4} p\min_k^1$: é representado na Figura 4.4 pelos arcos pontilhados.

$p\min_2^z$ com z = 2,...,5: é representado pelos arcos tracejados.

$p\min_3^z$ com z = 2,...,5: é representado pelos arcos tracejados e pontilhados.

Alguns questionamentos podem ocorrer quanto à distribuição dos menores tempos de processamento nas máquinas analisadas. No exemplo, sabe-se seguramente que 4 dos 5 menores tempos de processamento em M_2 concorrem com 4 dos 5 menores tempos de processamento de M_3. A pergunta é: como formar pares para tal concorrência de forma que a soma seja a menor possível ? A proposta deste limitante para este exemplo é que, dado os 4 tempos de cada máquina, estes sejam associados em ordem não-decrescente dos mesmos. Para comprovar tal proposta, considere a expressão:

$$\max(a_1, b_1) + ... + \max(a_i, b_i) + ...+ \max(a_k, b_k) +..... + \max(a_x, b_x), \quad (5)$$

onde a_w e b_w são números inteiros e positivos com w = {1,...,x}, e $a_i \geq a_k$ e $b_i \leq b_k$.

Seja a expressão a seguir:

$$\max(a_1, b_1) + ... + \max(a_k, b_i) + ...+ \max(a_i, b_k) +..... + \max(a_x, b_x). \quad (6)$$

Considerando as duas expressões apresentadas pode ser realizada uma análise de suas diferenças em todos os possíveis casos:

Caso 1: $a_i \geq b_i$ e $a_k \geq b_k$.

Expressão (5): $\max(a_1, b_1) + ... + a_i + a_k +..... + \max(a_x, b_x)$.

Dado que $a_i \geq a_k \geq b_k \geq b_i$:

Expressão (6) : $\max(a_1, b_1) + ... + a_k + a_i +..... + \max(a_x, b_x)$.

Logo, neste caso (5) = (6).

Caso 2: $a_i \geq b_i$ **e** $b_k \geq a_k$.

Expressão (5): $\max(a_1, b_1) + ... + a_i + b_k + ... + \max(a_x, b_x)$.

Expressão (6) : $\max(a_1, b_1) + ... + \max(a_k, b_i) + ...+ \max(a_i, b_k) + ... + \max(a_x, b_x)$.

Como $b_k \geq a_k$ e $b_k \geq b_i$,

$$\max(a_k, b_i) \leq b_k. \tag{7}$$

Sabe-se também que $a_i \geq a_k$ e $a_i \geq b_i$, logo:

$$\max(a_k, b_i) \leq a_i. \tag{8}$$

Supondo que $\max(a_i, b_k) = a_i$ e da expressão (7) tem-se em (6) que:

$$\max(a_1, b_1) + ... + \max(a_k, b_i) + \max(a_i, b_k) + ... + \max(a_x, b_x) =$$

$$\max(a_1, b_1) + ... + \max(a_k, b_i) + a_i + ... + \max(a_x, b_x) \leq$$

$$\max(a_1, b_1) + ... + b_k + a_i + ... + \max(a_x, b_x).$$

Logo, (5) ≥ (6).

Suponha que $\max(a_i, b_k) = b_k$ e da expressão (8) tem-se em (6) que:

$$\max(a_1, b_1) + ... + \max(a_k, b_i) + \max(a_i, b_k) + ... + \max(a_x, b_x) =$$

$$\max(a_1, b_1) + ... + \max(a_k, b_i) + b_k + ... + \max(a_x, b_x) \leq$$

$$\max(a_1, b_1) + ... + a_i + b_k + ... + \max(a_x, b_x).$$

Logo, (5) ≥ (6).

Caso 3: $a_i \leq b_i$ **e** $a_k \leq b_k$.

Expressão (5): $\max(a_1, b_1) + ... + b_i + b_k + ... + \max(a_x, b_x)$.

Dado que $b_k \geq b_i \geq a_i \geq a_k$.

Expressão (6) : $\max(a_1, b_1) + ... + b_i + b_k + ... + \max(a_x, b_x)$.

Logo, para este caso (5) = (6).

Caso 4: $a_i \leq b_i$ **e** $a_k \geq b_k$.

Dado que $b_k \geq b_i \geq a_i \geq a_k$ é impossível que $a_k \geq b_k$, tornando este caso inviável de ocorrer.

Logo, para todos os casos a expressão (6) é menor ou igual a expressão (5), ou seja, dada uma expressão do tipo somatória dos máximos obtém-se o menor valor para esta associando-se o menor

elemento de a com o menor elemento de b, o segundo menor elemento de a com o segundo menor elemento de b, e assim por diante.

Dado que obteve-se uma forma para gerar um limitante inferior para a soma dos máximos, resta mostrar que dentre os i menores tempos de processamento os i-1 maiores devem entrar na soma dos máximos. Em outras palavras, a proposta para o limitante é que o menor tempo de processamento componha o limitante na parte fixa, ou seja, que contribua diretamente com o seu valor e não faça parte da soma dos máximos. O valor do limitante inferior com o menor tempo possível da máquina referenciada M_j não pertencente à soma dos máximos é o seguinte:

$$\sum_{k=1}^{j-1} p\min_k^1 + \sum_{k=j+1}^{m} p\min_k^1 + p\min_j^1 + \sum_{z=2}^{i} \max(p\min_j^z,\ p\min_{j+1}^z)\ .$$

Suponha agora que seja o segundo menor tempo de $\mathbf{M_j}$ o que não pertence à soma, e o primeiro faça parte da soma dos máximos. Neste caso, tem-se uma nova expressão:

$$\sum_{k=1}^{j-1} p\min_k^1 + \sum_{k=j+1}^{m} p\min_k^1 + p\min_j^2 + \max(p\min_j^1, p\min_{j+1}^2) + \sum_{z=3}^{i} \max(p\min_j^z,\ p\min_{j+1}^z)$$

Para comparar as expressões descritas anteriormente é suficiente estabelecer uma relação entre as expressões (9) e (10) abaixo.

$$p\min_j^1 + \max(p\min_j^2, p\min_{j+1}^2)\ . \tag{9}$$

$$p\min_j^2 + \max(p\min_j^1, p\min_{j+1}^2)\ . \tag{10}$$

Suponha que $p\min_j^2 \geq p\min_{j+1}^2$:

Expressão (9): $p\min_j^1 + p\min_j^2$,

Expressão(10): $p\min_j^2 + \max(p\min_j^1, p\min_{j+1}^2)$.

Como $\max(p\min_j^1, p\min_{j+1}^2) \geq p\min_j^1$ tem-se que (10) ≥ (9).

Suponha que $p\min_j^2 \leq p\min_{j+1}^2$:

Expressão (9): $p\min_j^1 + p\min_{j+1}^2$,

Expressão(10): $p\min_j^2 + \max(p\min_j^1, p\min_{j+1}^2)$.

Como $p\min_{j+1}^{2} \geq p\min_{j}^{2} \geq p\min_{j}^{1}$ tem-se que:

Expressão (10): $p\min_{j}^{2} + p\min_{j+1}^{2}$.

Logo (10) ≥ (9) e conclui-se que o limitante é a expressão onde o menor tempo possível da máquina referenciada M_j não pertença a soma dos máximos, ou seja, o limitante proposto.

Dada a comprovação que a expressão (4) representa um limitante inferior para o instante de término de uma dada tarefa i ∈ NSP, pode-se calcular o limitante inferior da soma dos atrasos destas tarefas conforme sugerido por Kim(1995), ou seja, através da seguinte expressão:

$$\sum_{i=1}^{n-|SP|} \left[\max_{1 \leq j \leq m} \left\{ \sum_{k=1}^{m} p\min_{k}^{1} + \sum_{z=2}^{i} \max\left(p\min_{j}^{z}, p\min_{j+1}^{z}\right) \right\} - d_{S_2(i)} \right]^{+}$$

onde S_2 é a seqüência que possui as tarefas de NSP ordenadas de acordo com a regra de despacho EDD, $d_{S2}(i)$ é a data de entrega da tarefa i de acordo com a seqüência S_2, |SP| representa o número de tarefas em SP, e $x^{+} = \max(x, 0)$.

- **Limitante inferior da soma dos atrasos para as tarefas que pertencem a seqüência parcial (SP)**

Para o cálculo deste limitante necessita-se de uma estimativa do instante de tempo no qual a última tarefa em NSP libera cada máquina. Este limitante para cada máquina **j** pode ser definido por:

$$\max \left\{ \begin{array}{l} \max\limits_{1 \leq w \leq j-1} \left\{ \sum\limits_{k=1}^{j} p\min_{k}^{1} + \sum\limits_{z=2}^{|NSP|} \max\left(p\min_{w}^{z}, p\min_{w+1}^{z}\right) \right\} \\ \sum\limits_{k=1}^{j} p\min_{k}^{1} + \sum\limits_{z=1}^{|NSP|-1} \max\left(p\min_{j}^{z+1}, p\min_{j+1}^{z}\right) \end{array} \right.$$

A primeira parte desta expressão representa este limitante sob o referencial de uma máquina w menor que j. Esta expressão é obtida de forma similar ao limitante inferior do instante de término de cada tarefa em NSP na máquina **m** descrito no item anterior. A Figura 4.6 ilustra o cálculo desta parcela para um exemplo com 4 máquinas e 5 tarefas pertencentes a NSP. Deseja-se calcular o instante de liberação da máquina 2, j=2, sob o referencial da máquina 1, w=1.

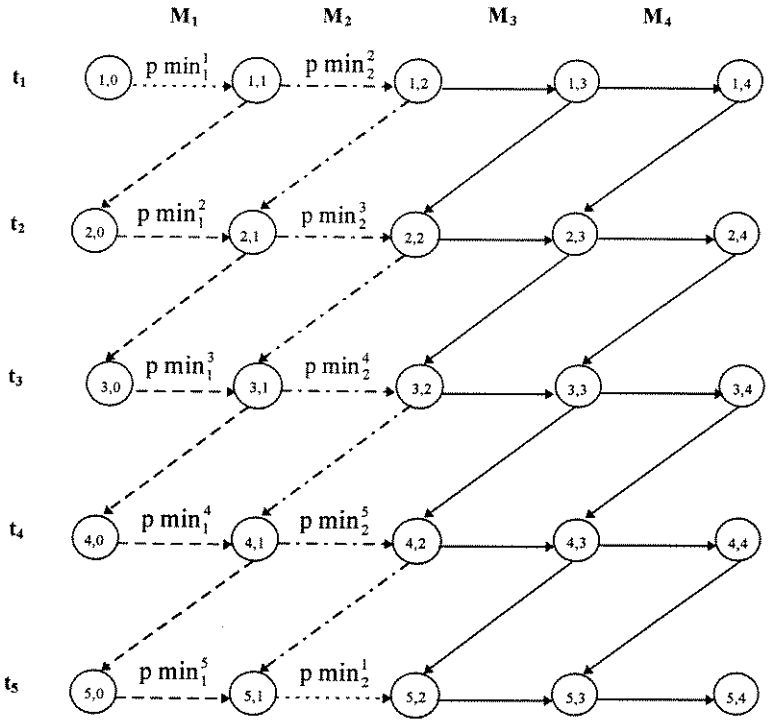


Figura 4.5. Exemplo do cálculo da primeira parcela do limitante inferior do instante de liberação da máquinas pelas tarefas em NSP para buffer zero através de um grafo.

Analisando-se esta parcela do limitante através do grafo, nota-se que:

- os arcos pontilhados indicam a primeira somatória desta parcela;
- os arcos tracejados, e os arcos tracejados e pontilhados representam a somatória dos máximos.

A segunda parcela é uma adaptação da primeira para o cálculo deste limitante sob o referencial da própria máquina j. A Figura 4.6 ilustra o cálculo desta segunda parcela para j=2.

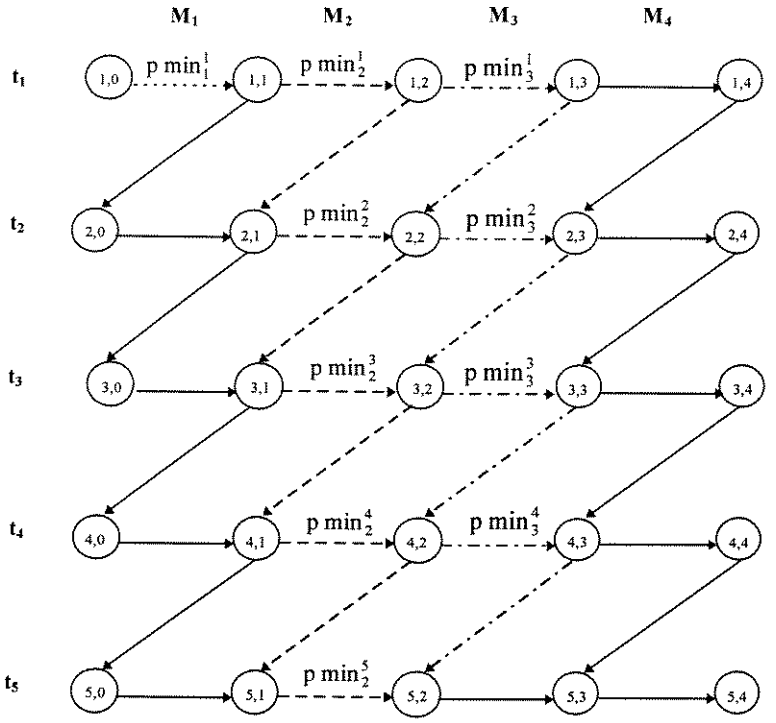


Figura 4.6. Exemplo do cálculo da segunda parcela do limitante inferior do instante de liberação da máquinas pelas tarefas em NSP para buffer zero através de um grafo.

Neste caso, os tempos de processamento que geram a menor soma dos máximos são os 4 maiores tempos de M_2, e os 4 menores tempos de M_3. Para esclarecer o motivo de tal mudança analisa-se a seguir o cálculo desta segunda parcela do limitante através do grafo. Pode-se observar que:

- o mínimo tempo que M_2 tem que esperar para iniciar a processar qualquer tarefa é o mínimo tempo de processamento de uma tarefa na máquina M_1 (arco pontilhado).
- o mínimo tempo no qual M_2 estará disponível é diretamente associado ao processamento das 5 tarefas que possuam o mínimo tempo de processamento na mesma (arco tracejado).
- além de processar cada uma das 4 últimas tarefas, ao término de cada uma, M_2 terá sempre que esperar que a tarefa com o mínimo tempo de processamento seja processada em M_3, por uma questão de possível bloqueio (arco tracejado e pontilhado).
- depois da partida das 5 tarefas de M_2, tem-se o tempo mínimo para que as 5 tarefas em NSP liberam esta máquina.

Observe que o quinto menor tempo de M_3 não necessita entrar no cálculo desta parcela. Isto explica o fato de serem selecionados para a soma dos máximos os 4 menores tempos de M_3.

Dado este limitante, torna-se viável o cálculo do limitante inferior dos instantes de término das tarefas em SP. Para este cálculo são utilizadas as fórmulas recursivas apresentadas no capítulo 1

partindo dos instantes nos quais as máquinas são liberadas pelas tarefas em NSP. Estas fórmulas são descritas a seguir.

$$D_{i0} = LD_1 \qquad i = |NSP| + 1;$$

$$D_{ij} = \max(D_{i,j-1} + p_{ij}, LD_{j+1}) \qquad i = |NSP| + 1;\ j = 1,...,m-1$$

$$D_{i0} = D_{i-1,1} \qquad i = |NSP| + 2, ...,n;$$

$$D_{ij} = \max(D_{i,j-1} + p_{ij}, D_{i-1,j+1}) \qquad i = |NSP| + 2, ...,n;\ j = 2,...,m;$$

$$D_{im} = D_{i,m-1} + p_{im} \qquad i = |NSP| + 1, ...,n;$$

onde D_{ij} representa o instante no qual a tarefa t_i libera a máquina j, |NSP| significa o número de tarefas neste conjunto, e LD_j é o limitante inferior do instante de liberação da máquina j pelas tarefas em NPS. Das fórmulas descritas acima, obtém-se o instante em que as tarefas em SP liberam a máquina **m** os quais são iguais aos instantes de término de cada tarefa nesta máquina, isto é $D_{im}=C_{im}$. Portanto, pode-se calcular a soma dos atrasos individuais.

- **Passos Básicos**

Os passos básicos da aplicação do método *Branch-and-Bound* a este problema são basicamente os mesmos descritos para o problema de *flowshop* com buffer ilimitado. A única etapa excluída foi a eliminação de tarefas através da regra de dominância, pois esta é específica para *flowshop* com buffer ilimitado. Alterou-se o cálculo do limitante inferior da soma dos atrasos conforme a proposta descrita nos itens anteriores. Além disso, deve-se acrescentar uma comparação entre o limitante calculado do nó (nó filho) e o limitante do nó que lhe deu origem (nó pai). O limitante do nó filho deve ser o máximo entre os dois. Tal comparação é realizada pois o limitante de um nó filho não é obrigatoriamente maior que o limitante do nó pai. Isto pode ocorrer pois o limitante final é composto pela soma de dois limitantes calculados de forma independente, o limitante proveniente das tarefas em SP e o limitante das tarefas em NSP. Quando se fixa uma tarefa a mais em SP o limitante destas tarefas aumenta, e conseqüentemente o limitante das tarefas em NSP diminui, porém a soma destes dois limitantes pode diminuir. Apresenta-se na Figura 4.7 a árvore parcial de um exemplo com 5 tarefas e 3 máquinas onde tal situação ocorre. A matriz P contém os tempos de processamento das tarefas nas máquinas, e o vetor d as datas de entrega. O número ao lado dos nós indica o limitante inferior obtido para este nó.

P =

	M_1	M_2	M_3
1	15	6	10
2	7	36	3
3	38	32	90
4	5	8	77
5	50	76	40

d =

1	2	3	4	5
30	78	35	28	19

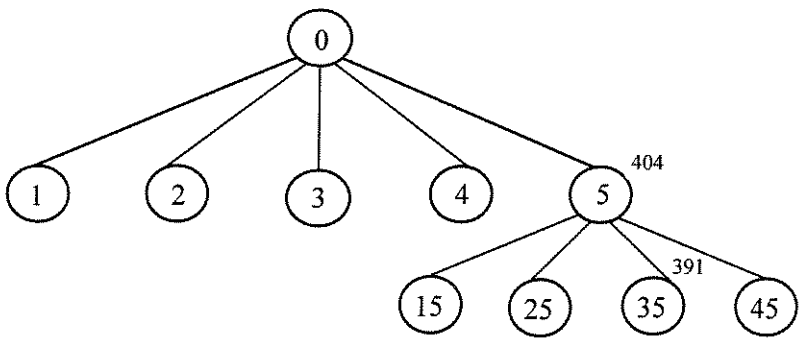


Figura 4.7. Exemplo da árvore parcial gerada pelo *Branch-and-Bound* para um problema com 5 tarefas.

O limitante (404) do nó 5 é obtido da soma do limitante das tarefas em SP (225) com o limitante das tarefas em NSP (179). Note que o nó 35 possui um limitante (391) menor que o obtido para o nó 5. Analisando este limitante de forma relativa ao limitante do nó 5 observou-se que o limitante das tarefas em SP (368) aumentou, enquanto que o limitante das tarefas em NSP (23) diminuiu, e a soma dos dois diminuiu. Cabe ressaltar que em 192 problemas resolvidos por esta metodologia os limitantes inferiores diminuíram em aproximadamente 0,5% dos nós. Quando tal caso ocorre, atribui-se ao nó filho o limitante do nó pai.

O limitante proposto por Kim(1995) para *flowshop* com buffer ilimitado também é composto por dois limitantes que são calculdos de forma independente, um não-crescente e outro não-decrescente com a profundidade. Nos testes computacionais observou-se que a soma é não-decrescente com a profundidade, no entanto não existe demonstração formal para este fato.

- **Resultados Computacionais**

Esta técnica foi aplicada a 192 problemas onde o esquema de geração foi mantida, ou seja, para cada matriz de tempos de processamento gerou-se 4 cenários. Os testes foram realizados numa estação de trabalho SUN modelo SPARCstation 5. A estes problemas aplicou-se o método *Branch-and-Bound* com tolerância de 0,5%, e soluções iniciais fornecidas pela versão básica da Busca Tabu.

Embora os limitantes descritos por Kim sejam específicos para o problema de *flowshop* com buffer ilimitado, os mesmos são aplicáveis ao problema de *flowshop* com buffer zero. Tal fato se justifica devido ao primeiro problema não possuir restrições quanto ao armazenamento de tarefas entre máquinas como o segundo, e portanto não está sujeito ao processo de bloqueio. Dado que esta é a única diferença, um problema **n x m** com os mesmos tempos de processamento e datas de entrega certamente terá no ambiente com buffer ilimitado a soma dos atrasos menor ou igual a esta soma no ambiente com buffer zero. Logo, um limitante inferior para o *flowshop* com buffer ilimitado para a soma dos atrasos também é um limitante inferior para o buffer zero. Para efeitos comparativos, aplicou-se o limitante de Kim ao problema de *flowshop* com buffer zero. A Tabela 4.3 mostra as dimensões utilizadas, o número de matrizes consideradas, e comparações realizadas entre o limitante inferior proposto por Kim(1995) e o proposto neste trabalho.

Tabela 4.3. Comparações entre o limitante de Kim e o proposto.

Dimensão	Nº. de	Tempos médios (seg.)		Número de nós		Ganho
(n x m)	matrizes	Kim	Proposto	Kim	Proposto	%
10 x 5	10	6,62	5,98	73542	68762	1,58
10 x 7	10	29,46	25,15	220077	206217	1,43
10 x 10	10	84,47	72,41	490697	473668	0,62
12 x 5	05	262,56	210,58	2531502	2140631	2,78
12 x 7	05	627,17	497,23	4287969	3856710	1,98
12 x 10	05	3905,15	3205,56	19782434	18640788	1,06
14 x 5	03	6308,66	4516,83	50970704	38323192	2,77

Nesta tabela apresentam-se os tempos médios de execução e o número médio de nós resolvidos correspondentes às diferentes implementações do método *Branch-and-Bound*. Além disso, apresenta-se na última coluna a porcentagem média de ganho relativo do limitante inferior proposto sobre o limitante de Kim. Esta porcentagem foi obtida calculando-se os dois limitantes analisados para cada nó dos 192 problemas testados. Ressalta-se que em todos os nós o valor do limitante proposto foi superior ou igual ao valor do limitante proposto por Kim. Observa-se através da Tabela 4.3 que a utilização do limitante inferior proposto apresenta melhores resultados tanto em termos do tempo de execução quanto em relação ao número de nós resolvidos. Embora o limitante de Kim seja mais fraco pois não é específico para buffer zero, os resultados da Tabela 4.3 não são óbvios pois este limitante pode ser calculado de forma mais rápida que o limitante proposto e mais nós poderiam ser descartados. Porém, como pode ser verificado através das colunas que indicam os tempos dos algoritmos, o uso de um limitante mais forte é válido.

- **Comparações com a heurística**

Nos problemas de menor dimensão descritos anteriormente aplicou-se a Busca Tabu na sua versão básica, com diversificação associada a intensificação, e com a redução de vizinhança com tempo 40% menor. A heurística na versão básica encontrou os valores ótimos em 185 problemas, na versão com redução de vizinhança e tempo 40% menor em 181 problemas e, na versão utilizando diversificação e intensificação em 185 problemas. O algoritmo NEHad* encontrou o valor ótimo em 16 problemas. No pior caso a heurística na versão básica, com redução de vizinhança, e com diversificação e intensificação ficaram respectivamente a 10,42%, 8,05% e 10,42% do valor ótimo, enquanto o algoritmo NEHad* ficou no pior caso a 247,86% do ótimo. A Tabela 4. 4, apresentada a seguir, contém uma síntese destes resultados.

Tabela 4. 4. Comparações entre os métodos de resolução aplicados a 192 problemas.

Métodos de Resolução	Número de problemas com valor ótimo	Porcentagem de problemas com valor ótimo	Maior porcentagem acima do ótimo
Busca Tabu básica	185	96,3	10,42
Busca Tabu diver. + inten.	185	96,3	10,42
Busca Tabu com corte e tempo 40% menor	181	94,3	8,05
NEHad*	16	8,3	247,86

Embora, aparentemente, a mescla de diversificação e intensificação obtenha os mesmos resultados que a versão básica, a mescla está em média a 4,25% do ótimo nos problemas que não estão no ótimo, enquanto que a versão básica está a 6,10%.

Capítulo 5
Conclusões

Este trabalho tratou do problema de programar **n** tarefas em **m** máquinas no ambiente *flowshop* permutacional com buffer entre máquinas ilimitado e zero. O critério de otimalidade utilizado é a minimização da soma dos atrasos das tarefas. Esse critério é de grande importância nos sistemas de manufatura, pois quando uma tarefa é entregue com atraso pode existir uma série de custos.

O nível de dificuldade do problema de *flowshop* com buffer ilimitado com este critério pode ser estimado através de um caso básico: minimizar a soma dos atrasos em uma máquina é NP-*hard*. O problema de *flowshop* com buffer zero considerando a minimização da soma dos atrasos aparentemente não foi tratado na literatura, porém sabe-se que este problema com três máquinas com o objetivo de minimizar o *makespan* é NP-*hard in the strong sense*. Por tais dificuldades surge a motivação para o desenvolvimento de heurísticas para estes problemas. Decidiu-se pela elaboração de heurísticas baseadas na metaheurística Busca Tabu por esta ter sido utilizada com sucesso no problema de *flowshop* com buffer ilimitado com o critério de minimização do *makespan*.

Primeiramente foram apresentadas formulações de programação inteira mista para os problemas. A seguir, realizou-se uma análise do comportamento de todas as soluções para diferentes cenários de datas de entrega para problemas de dimensão reduzida, onde algumas importantes características de cada cenário foram detectadas. Essas características foram fundamentais tanto na elaboração dos métodos heurísticos, quanto na análise dos resultados.

Além disso, foram propostas heurísticas que utilizam Busca Tabu como método de exploração do espaço de soluções. Testes computacionais compararam as heurísticas com a melhor adaptação do algoritmo NEH, o NEHad*. A Tabela 5. 1 apresenta uma síntese desta comparação para os dois problemas tratados através do número de sucessos e empates obtidos por cada método. Considera-se sucesso quando o método analisado apresenta a menor soma dos atrasos.

Tabela 5. 1. Comparações entre a versão básica e o algoritmo NEHad* em 240 problemas

	Métodos de resolução	Número de sucessos	Número de empates
Problema de *flowshop* com buffer ilimitado	**Busca Tabu básica**	225	13
	NEHad*	2	
Problema de *flowshop* com buffer zero	**Busca Tabu básica**	238	2
	NEHad*	0	

Desta tabela constata-se que as implementações básicas das heurísticas, respectivamente para *flowshop* com buffer ilimitado e buffer zero, superam ou obtém as mesmas soluções apresentadas pelo algoritmo NEHad* em 99,17% e 100% dos 240 problemas testados. Deve ser ressaltado que outras adaptações do algoritmo NEH são consideradas extremamente eficientes para o critério do atraso médio para buffer ilimitado, e para o critério do *makespan* para buffer zero.

Com o intuito de aumentar a potencialidade da heurística foram propostas estratégias adicionais de diversificação, intensificação e de redução da vizinhança. Para o problema de *flowshop* com buffer ilimitado a aplicação da combinação das estratégias de diversificação e intensificação obteve melhorias ou manteve a média em 79,17% das classes dos problemas, além de obter no melhor caso uma melhoria de 30,41% sobre a versão básica. Destaca-se o fato destas melhorias terem sido obtidas no mesmo tempo de execução que Busca Tabu na versão básica. A estratégia do corte pode resultar em soluções de melhor qualidade para alguns problemas. Para o problema de *flowshop* com buffer zero, a associação da diversificação com a intensificação e a estratégia do corte obtiveram um desempenho global regular, mas não conclusivo sobre a superioridade de uma delas.

Cabe destacar que a diversificação associada à intensificação obteve bons resultados em termos do aumento da qualidade das soluções para os dois problemas. A estratégia de redução da vizinhança, utilizada com o intuito de diminuir o tempo necessário para obter soluções de boa qualidade, também apresentou um bom desempenho. Dado que estas estratégias são excludentes, a escolha de uma delas depende do compromisso entre o tempo e a qualidade.

Embora a estratégia de diversificação associada à intensificação tenha obtido resultados positivos, não deve ser desconsiderado que para estabelecer os parâmetros da mesma foram necessários inúmeros testes. Dado que as melhorias médias foram pequenas, questiona-se a aplicação de tais estratégias em problemas com características similares ao do flowshop de permutação, ou seja, diversos problemas nos quais se deseja achar a melhor permutação de n objetos, e com acesso à

vizinhanças ricas como a gerada pelo movimento da inserção. Acredita-se que possivelmente estas estratégias possuam um desempenho mais relevante em problemas com vizinhanças mais restritas como, por exemplo, o *jobshop*.

O bom desempenho das heurísticas foi comprovado pela comparação de seus resultados com os obtidos com o método *Branch-and-Bound*. Esta comparação foi realizada em problemas de dimensão reduzida, onde mostrou-se que as heurísticas obtém a grande maioria das soluções com valores ótimos. Tais resultados não garantem que as heurísticas propostas possuam o mesmo desempenho em problemas maiores, porém fornece uma boa indicativa sobre o comportamento das mesmas. Cabe realçar a importância do estudo de métodos ótimos para problemas de natureza combinatorial, pois sem os mesmos seria impossível avaliar propostas de heurísticas.

São descritas a seguir as principais propostas de futuras pesquisas decorrentes deste trabalho.

- Estudo para a seleção de componentes para a aplicação da metaheurística Busca Tabu no ambiente *flowshop* com pesos ponderados para os atrasos das tarefas;
- Estudo de uma nova heurística com Busca Tabu para o ambiente *flowshop* com buffer ilimitado e zero para o critério de minimização da soma dos atrasos utilizando uma vizinhança baseada na representação em grafo do problema;
- Utilização de diferentes regras para escolha do próximo nó a ser avaliado no método *Branch-and-Bound*, como por exemplo, profundidade combinada com o menor limitante inferior.

Bibliografia

Adenso_Díaz, B., 1992, Restricted Neighborhood in the Tabu Search for the Flowshop Problem, *European Journal of Operational Research*, 62, pp. 27-37.

Adenso_Díaz, B., 1996, An SA/TS Mixture Algorithm for the Scheduling Tardiness Problem, *European Journal of Operational Research*, 88, pp. 516-524.

Armentano, V.A., & Ronconi, D.P., 1997, Tabu Search for Total Tardiness Minimization in Flowshop Scheduling Problems, submetido a *Computers and Operations Research.*

Baker, K.R., 1974, Introduction to Sequencing and Scheduling, Addison-Wesley, John Wiley & Sons, New York.

Barnes, J.W., & Laguna,M., 1993, Solving the Multiple-Machine Weighted Flow Time Problem using Tabu Search, *IIE Transactions*, 25.2, pp. 121-128.

Barnes, J.W., & Chambers, J.B., 1995, Solving the Job Shop Scheduling Problem with Tabu Search, *IIE Transactions*, 27, pp. 257-263.

Berkley, B.J., 1993, Effect of buffer capacity and sequencing rules on single-card kanban system performance, *International Journal of Production Research*, 31.12, pp. 2875-2893.

Bornstein, C.T., 1989, An Heuristic Algorithm for the Minimization of Tardiness for the n-Job, m-Machine Scheduling Problem, *AMSE Review*, 12.1, pp. 11-25.

Carlier, J., & Rebai, I., 1996, Two Branch-and-Bound algorithms for Permutation Flowshop Problem, *European Journal of Operational Research*, 90, pp.238-251.

Dell'Amico, M., & Trubian, M., 1993, Applying tabu search to the job-shop scheduling problem, *Annals of Operations Research*, 41, pp. 231-252.

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

Dudek, R.A., Panwalkar, S.S., & Smith, M.L., 1992, The Lessons of Flowshop Scheduling Research, *Operations Research*, 40.1, pp. 7-13.

Dutta, S.K., & Cunningham, A.A., 1975, Sequencing Two-Machine Flow-Shops with Finite Intermediate Storage, *Management Science*, 21.9, pp. 989-996.

França, P.M., Gendreau, M., Laporte, G., Müller, F.M., 1995, The m-Traveling Salesman Problem with MinMax Objetive, *Transportation Science*, 29.3, pp. 267-275.

Garey, M.R., Johnson, D.S., & Sethi, R., 1976, The complexity of Flowshop and Jobshop Scheduling, *Mathematics of Operations Research*, 1, pp. 117-129.

Gelders, L.F., & Sambandam, N., 1978, Four simple heuristics for scheduling a flow-shop, *International Journal of Production Research.*, 16, pp. 221-231.

Gilmore, P.C., & Gomory, R.E., 1964, Sequencing a One State Variable Machine: A Solvable Case of the Traveling Salesman Problem, *Operational Research*, 12, pp. 655-665.

Glover, F.W., 1989, Tabu Search, Part I, *ORSA Journal on Computing*, 1, pp.190-206.

Glover, F.W., 1990, Tabu Search, Part II, *ORSA Journal on Computing*, 2, pp.4-32.

Glover, F.W., 1995, Tabu Search Fundamentals and Uses, Working paper, Graduate School of Business University of Colorado, CO, USA.

Glover, F.W., 1996, Tabu Search and Adaptive Memory Programming - Advances, Applications and Challenges, aparecerá em *Interfaces in Computer Science and Operations Research.*

Glover, F.W. & Laguna, M., 1993, Tabu Search, Modern Heuristic Techniques for Combinatorial Problems, C.Reeves, ed., Blackwell Scientific Publishing, pp.70-150.

Grabowski, J., Skubalska, E., & Smutnicki, C., 1983, On Flow Shop Scheduling with Release and Due Dates to Minimize Maximum Lateness, *Journal of the Operational Research Society*, 34.7, pp. 615-620.

Gupta, J.N.D., 1971, Optimal Flow Shop Scheduling with Due dates and Penalty Costs, *Journal of the Operational Research Society of Japan*, 14, pp. 35-46.

Hall, N.G., & Sriskandarajah, C., 1996, A Survey of Machine Scheduling Problems with Blocking and No-wait in Process, *Operations Research*, 44.3, pp. 510-525.

Hübscher, R., & Glover, F., 1994, Applying Tabu Search with Influential Diversification to Multiprocessor Scheduling, *Computers and Operations Research.*, 21.8, pp. 877-884.

Ignall, E., & Schrage, L.E., 1965, Applications of Branch-and-Bound Technique to Some Flow-shop Scheduling Research, *Operations Research*, 13.3, pp. 400-412.

Johnson, S.M., 1954, Optimal Two- and Three-Stage Production Schedules with Setup Times Included, *Naval Research Logistics Quartely*, 1, pp. 61-67.

Kim, Y-D., 1993a, A New Branch and Bound Algorithm for Minimizing Mean Tardiness in Two-Machine Flowshops, *Computers and Operations Research.*, 20, pp. 391-401.

Kim, Y-D., 1993b, Heuristics for Flowshop Scheduling Problems Minimizing Mean Tardiness, *Journal of the Operational Research Society*, 44.1, pp. 19-28.

Kim, Y-D., 1995, Minimizing Mean Tardiness on Flowshops, *European Journal of Operational Research*, 85.3, pp. 541-555.

Koulamas, C., 1994, The Total Tardiness Problem: Review and Extensions, *Operations Research*, 42.6, pp. 1025-1041.

Laguna, M., 1995, Tabu Search Tutorial, II Escuela de verano Latino-Americana de Investigación Operativa.

Laguna, M., & Velarde J.L.G., 1991, A Search Heuristic for Just-In-Time Scheduling in Parallel Machines, *Journal of Intelligent Manufacturing*, 2, pp. 253-260.

Laguna, M., Barnes,J.W., & Glover, F.W., 1991, Tabu Search Methods for a Single Machine Scheduling Problem, *Journal of Intelligent Manufacturing*, 2, pp. 63-74.

Leisten, R., 1990, Flowshop Sequencing Problems with Limited Buffer Storage, *International Journal of Prodution Research*, 28.11, pp. 2085-2100.

Lenstra, J.K., Rinnooy Kan A.H.G, e Brucker, P., 1977, Complexity of machine scheduling problems, *Annals of Discrete Math.*, 1, pp. 343-362.

Levner, E.M., 1969, Optimal Planning of Parts Machining on a Number of Machines, *Automation and Remote Control*, 12 (Dezembro de 1969), pp. 1972-1978.

Lomnicki, Z.A., 1965, A Branch-and-Bound Algorithm for the Exact Solution of the Three-Machine Scheduling Problem, *Operational Research Quaterly*, 16.1, pp. 89-100.

McCormick, S.T., Pinedo, M.L., Shenker, S., & Wolf, B., 1989, Sequencing in an Assembly Line with Blocking to Minimize Cycle Time, *Operations Research*, 37, pp. 925-936.

McMahon, G.B., & Burton, P.G., 1967, Flow-Shop Scheduling with the Branch and Bound Method, *Operations Research*, 15.3, pp. 473-481.

Moccellin, J.V., 1995, A New Heuristic Method for the Permutation Flow Shop Scheduling Problem, *Journal of Operational Research Society*, 46.7, pp. 883-886.

Nagano M.S., & Moccellin, J.V., 1995, Parametrização de Procedimentos de Busca Tabu para Solução do Problema de Programação de Operações Flow-Shop Permutacional, *Annals of the XV National Congress of Production Engineering*, 3, pp. 1258-1263.

Nawaz, M., Enscore, E.E., & Ham I., 1983, A Heuristic Algorithm for the m-Machine, n-Job Flow-shop Sequencing Problem, *OMEGA*, 11.1, pp. 91-95.

Nowicki E., & Smutnicki C., 1996a, A Fast Tabu Search Algorithm for the Flow Shop Problem, *European Journal of Operational Research*, 91.1, pp. 160-175.

Nowicki E., & Smutnicki C., 1996b, A Fast Taboo Search Algorithm for the Job Shop Problem, *Management Science*, 42.6, pp. 797-813.

Ow, P.S., 1985, Focused scheduling in proportional flowshops, *Management Science*, 31, pp. 852-869.

Ogbu, F.A., & Smith, D.K., 1990, The Application of Simulated Anneling Algoritm to the Solution of the $n/m/C_{max}$ Flowshop Problem, *Computers and Operations Research*, 17, pp. 243-253.

Osman, I.H., & Potts, C.N., 1989, Simulated Annealing for the Permutation for Flow-shop Scheduling, *OMEGA*, 17.6, pp. 551-557.

Papadimitriou, C.H., & Kanellakis, P.C., 1980, Flowshop Scheduling with Limited Temporary Storage, *Journal of the Association for Computing Machinery*, 27.3, pp. 533-549.

Pinedo, M., 1995, Scheduling: theory, algorithms, and systems, Prentice-Hall, New Jersey.

Potts, C.N., & Van Wassenhove, L.N., 1982, A Decomposition Algorithm for the Single Machine Total Tardiness Problem, *Operations Research Letters*, 1.5, pp. 177-181.

Raman, N., 1995, Minimum tardiness scheduling in flow shops: Construction and evaluation of alternative solution approaches, *Journal of Operations Management*, 12, pp. 131-151.

Reddi, S.S., & Ramamoorthy, C.V., 1972, On the Flowshop Sequencing Problem with No Wait in Process, *Operational Research Quarterly*, 23.3, pp. 323-330.

Reddi, S.S., 1976, On the Flowshop Sequencing Problem with No Wait in Process, *Management Science*, 23.2, pp. 216-217.

Reeves, C.R., 1993, Improving the Efficiency of Tabu Search for Machine Sequencing Problems, *Journal of the Operational Research Society*, 44.4, pp.375-382.

Reeves, C.R., 1995, A Genetic Algorithm for Flowshop Sequencing, , *Computers and Operations Research.*, 22, pp. 5-13.

Reklaitis, G.V., 1982, Review of Scheduling of Process Operations, *AIChE Symposium Series*, 78, pp. 119-133.

Ronconi, D.P., & Armentano, V.A., 1995 (publicado em 1997), Busca Tabu para a minimização do tempo total de atraso no problema de flowshop, *Pesquisa Operacional* , 15, pp.23-34.

Sadeh, N., 1991, Tese de Doutorado - Look-Ahead Techniques for Micro-Opportunistic Job Shop Scheduling, Carnigie Mellon University, Pittsburgh, Pennsylvania.

Sen, T., & Gupta, S.K., 1984, A State-of-Art Survey of Static Scheduling Research Involving Due Dates, *OMEGA*, 12.1, pp. 63-76.

Sen, T., Dileepan, P., & Gupta, J.N.D., 1989, The two-machine flowshop scheduling problem with total tardiness, *Computers and Operations Research.*, 16, pp. 333-340.

Sharadapriyadarshini, B., & Rajendran, C., 1996, Formulations and heuristics for scheduling in buffer-constrained flowshop and flowline-based manufacturing cell with different buffer-space requirements for jobs: Part 1, *International Journal of Production Research*, 34.12, pp. 3465-3485.

Smith, W.E., 1956, Various Optimizers for Single-Stage Production, *Naval Research Logistics Quaterly*, 3, pp. 59-66.

Suhami, I., & Mah, R.S.H., 1981, An Implicit Enumeration Scheme for the Flowshop Problem with No Intermediate Storage, *Computers and Chem. Eng.*, 5, pp. 83-91.

Taillard, E.,1990, Some efficient heuristic methods for the flow shop sequencing problem, *European Journal of Operational Research*, 47, pp. 65-74.

Taillard, E.,1993, Benchmarks for basic scheduling problems, *European Journal of Operational Research*, 64, pp. 278-285.

Taillard, E.,1994, Parallel Taboo Search Techniques for the Job Shop Scheduling Problem, *ORSA Journal on Computing*, 6.2, pp. 108-117.

Townsend, W., 1977, Sequencing n jobs on m machines to minimize maximum tardiness: a branch-and-bound solution, *Management Science*, 23, pp. 1016-1019.

Wagner, H.M., 1959, An Integer Programming Model for Machine Scheduling, *Naval Research Logistics Quaterly*, 6, pp. 131-140.

de Werra, D., & Hertz, A., 1989, Tabu Search Techniques: a Tutorial and an Applications to Neural Networks, *OR Spectrum*, 11, pp. 131-141.

Werner, F., 1993, On the Heuristic Solution of the permutation Flow Shop Problem by Path Algorithms, *Computers and Operations Research.*, 20.7, pp. 707-722.

Widmer, M., & Hertz, A., 1989, A New Heuristic Method for the Flow Shop Sequencing Problem, *European Journal of Operational Research*, 41, pp. 186-193.

Yamashita, D.S., 1996, Tese de mestrado - Minimização do Atraso Médio na Programação de Máquinas Paralelas: Uma Aplicação de Busca Tabu, FEEC, UNICAMP, Campinas/SP.