UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE ENGENHARIA ELÉTRICA

DEPARTAMENTO DE ENGENHARIA DE SISTEMAS

ALGORITMOS HEURÍSTICOS E EXATOS PARA RESOLUÇÃO DO PROBLEMA DE SEQUENCIAMENTO EM PROCESSADORES PARALELOS

Este exemplar corresponde à redação final da tese defendida por Felipe Martins Müller e aprovada pela Comissão Julgadora em 22 10 1993.

Orientador

Felipe Martins Müller

Orientador : Paulo Morelato França

Co-Orientador: Michel Gendreau

Tese apresentada à Faculdade de Engenharia Elétrica da Universidade Estadual de Campinas - UNICAMP como parte dos requisitos para obtenção do título de Doutor em Engenharia Elétrica.

OUTUBRO - 1993

UNICAMP
BIBLIOTECA CENTRAL

Para Pedro e Heloisa
Simplesmente Fundamentais

AGRADECIMENTOS

Ao Paulo França pela orientação correta e pacienciosa e, principalmente, pela oportunidade de conhecê-lo como pessoa. Obrigado AMIGO.

Ao Vinícius Armentano, que aceitou o desafio de participar ativamente desta tese com suas sugestões sempre bem vindas e oportunas.

Ao Gilbert Laporte pela motivação e orientação quando da minha estada no CRT.

A Neuzinha pela acolhida, amizade e luz.

Ao CRT - Centre de Recherche sur Les Transports - Montreal - Canadá pela oportunidade de desenvolver parte no meu trabalho usando sua estrutura e principalmente ao meus orientadores do hemisfério norte: Michel Gendreau e Gilbert Laporte. Não esquecendo Daniel Charboneaux, Serge Goyette e Helène Mercure pelo indispensável apoio computacional e a todas as pessoas, funcionários e amigos que por lá ficaram.

Aos amigos de longa data, de fé e outros que surgiram de forma furacônica e de alguma maneira contribuiram para o que se concretiza neste trabalho. Arrebenta GURIZADA MEDONHA: Ricardo Troian, Celso, Zebra, Marie, Humberto, Ieda, Susi, Clélia, Saulo, Luciana, Ricardo Luders, Marcel, Alexandre, Piriquito, Adinei, Pinda, Andréia, Marcelo, Cíntia, Zake, Regina, Vitória, Stefano Benati (Mago Mona), Marie Josée e vários outros que sabem que têm seu lugar.

Aos amigos e funcionários da UNICAMP, principalmente à Márcia pela disposição e amizade e a Elza Aoki pelo paciente trabalho de impressão deste texto.

Ao NPD-UFSM que proporcionou o tempo necessário para realização deste trabalho, principalmente ao Aldiocir, atual diretor e constante incentivador.

A um amor nascente: GISELA.

Ao FOUFOUNES ELECTRIQUES: o *Bar Underground* que toda cidade merecia ter.

Ao Heavy Metal, meu tranquilizante, Rock & Roll, MPB e as Gaudérias que tiveram, todas, seu momento nesta caminhada.

APRESENTAÇÃO

Esta tese trata da resolução do Problema de Sequenciamento (*scheduling*) que consiste em alocar n tarefas a m processadores paralelos idênticos, com o objetivo de minimizar o tempo total de execução das tarefas. São considerados dois casos: no primeiro as tarefas são independentes e os tempos de processamento são inteiros e positivos; no segundo caso os tempos de processamento são dependentes da seqüência em que as tarefas são processadas.

Para o primeiro caso foi desenvolvido um método heurístico de três fases: alocação inicial, fase de balanceamento e duplas trocas. O algoritmo 3-FASES é testado contra três outros métodos da literatura, escolhidos pelo bom desempenho conhecido. Além disso, é feita uma comparação com um método exato a fim de poder avaliar a qualidade das soluções encontradas pelas heurísticas.

Foram feitos testes com problemas gerados aleatoriamente com diversas distribuições de probabilidade. O método se mostrou superior (com somente uma exceção em 1175 problemas testados) aos outros métodos considerados. Para os 999 problemas para os quais se conhece a solução ótima a heurística 3-FASES encontrou também a solução ótima em aproximadamente 70% casos.

Para o problema do segundo caso, também foi desenvolvida uma heurística de três fases: alocação inicial, fase tabu e fase de pós-otimização. Este algoritmo usa técnicas de busca tabu e um procedimento de inserção, originalmente desenvolvido para o Problema do Caixeiro Viajante, denominado GENIUS adaptado para o problema de sequenciamento. Um procedimento de vizinho mais próximo também foi adaptado para o problema de sequenciamento e usado para comparações contra o método proposto. Um método exato também é apresentado para o problema e usado para comparações das heurísticas.

Foram feitos testes em problemas gerados aleatóriamente de uma forma denominada estruturada e não estruturada.

A apresentação da tese está organizada da seguinte forma.

No Capítulo 1 os problemas tratados nesta tese são apresentados e classificados dentro do universo dos problemas de sequenciamento. São feitas algumas considerações

sobre complexidade e é feita uma revisão da literatura que trata dos problemas aqui abordados.

No Capítulo 2 é proposto o algoritmo heurístico 3-FASES e apresentado um algoritmo ótimo para a resolução do problema de sequenciamento. A descrição do algoritmo heurístico é acompanhada de explicações e exemplos que auxiliam sua compreensão.

O Capítulo 3 trata dos métodos propostos para solução do problema de sequenciamento com tempos de processamento dependentes da seqüência. Novamente um algoritmo heurístico e um ótimo são apresentados e detalhados. Também se faz uso de exemplos para melhorar o entendimento dos mesmos.

No Capítulo 4 são apresentados os resultados computacionais obtidos pelos métodos propostos e as comparações feitas em relação a outros métodos da literatura.

A tese é finalizada com a apresentação das conclusões, de um apêndice sobre os fundamentos da Busca Tabu e das referências bibliográficas.

ABSTRACT

This thesis deals with the problem of scheduling n jobs on m identical parallel machines with the objective of minimizing the total execution time (makespan). Two cases are considered: in the first one the jobs are independent and the processing times are positive integers; in the second case we have sequence dependent times.

For the first case we propose a 3-PHASE heuristic: initial assignment, job reassignment and job interchange. The 3-PHASE algorithm is compared with three other heuristics chosen from the literature by its good known average performance. The new heuristic is also compared with an exact method in order to evaluate the quality of the solutions obtained by the heuristic.

Extensive computational tests were performed for randomly generated problems and they exhibited that the 3-PHASE heuristic yields average solution values at least as good as (with only one exception) those obtained with any of the three alternative heuristics used for comparison. The 3-PHASE algorithm found the optimal solution in around 70% of the problems for wich the optimal solution were known.

In the second case we also propose a three phase heuristic: initial assignment, tabu phase and post-optimization phase. This algorithm makes use of tabu search techniques and general insertion procedure called GENIUS, originally designed for the Traveling Salesman Problem and properly adapted for the scheduling problem. A nearest neighbour procedures was also adapted for the scheduling problem and was used in comparisons with the proposed method. An exact method was developed for the problem in question.

Tests were performed in randomly generated problems in a structured fashion and in a non-structured fashion. Results for both cases are presented and commented.

SUMÁRIO

Capítulo 1 - INTRODUÇÃO E REVISÃO BIBLIOGRÁFICA

1.1. Problemas de Sequenciamento: Classificação 1
1.2. Considerações sobre Complexidade 4
1.3. O Problema *P*| |*Cmax* 9
1.4. O Problema *P*|*tpds*|*Cmax* 14
1.4.1. Caso $m = 1$ 15
1.4.2. Caso $m > 1$ 22

Capítulo 2 - ALGORITMO PARA SOLUÇÃO DO PROBLEMA DE SEQUENCIAMENTO EM PROCESSADORES PARALELOS - *P*| |*Cmax*

2.1. Introdução 26
2.2. Modelo Matemático do Problema *P*| |*Cmax* 27
2.3. Algoritmo Heurístico 3-FASES 28
2.4. Prova de Parada 41
2.5. Limitante de Pior Caso 44
2.5. Esquema Ótimo para Resolução do Problema *P*| |*Cmax* 45

Capítulo 3 - ALGORITMO PARA SOLUÇÃO DO PROBLEMA DE SEQUENCIAMENTO EM PROCESSADORES PARALELOS COM TEMPOS DE PROCESSAMENTO DEPENDENTES DA SEQÜÊNCIA - *P*|*tpds*|*Cmax*

3.1. Introdução 48
3.2. O Problema *P*|*tpds*|*Cmax* 49
3.3. Algoritmo Heurístico para Solução do Problema *P*|*tpds*|*Cmax* 50
3.3.1. Procedimento de Inserção: GENI 51
3.3.2. Procedimento de Retirada: US 59
3.3.3. Descrição do Algoritmo e Exemplo Ilustrativo 67
3.4. Algoritmo Ótimo para Solução do Problema *P*|*tpds*|*Cmax* 74

Capítulo 4 - RESULTADOS COMPUTACIONAIS

4.1. Introdução 77
4.2. Resultados Computacionais do Problema *P*| |*Cmax* 78
4.2.1. Geração dos Conjuntos de Teste 79
4.2.2. Escolha do Parâmetro *r* 80
4.3. Resultados Computacionais do Problema *P*|*tpds*|*Cmax* 89

CONCLUSÃO 103
APÊNDICE - Fundamentos da Busca Tabu 107
CONJECTURA DO PESQUISADOR 115
REFERÊNCIAS BIBLIOGRÁFICAS 116

CAPÍTULO 1

Introdução e Revisão Bibliográfica

1.1. PROBLEMAS DE SEQUENCIAMENTO: CLASSIFICAÇÃO

Esta tese trata de problemas de sequenciamento (*scheduling*). Inicialmente vamos apresentar uma classificação dos problemas de sequenciamento que permite enquadrar os problemas tratados dentro da teoria de sequenciamento.

Suponha que m processadores M_i ($i = 1,...,m$) devem executar n tarefas J_j ($j = 1,...,n$). Um sequenciamento (*schedule*) é a alocação destas tarefas aos processadores. Um sequenciamento é *factível* se as condições impostas pela natureza do problema (características próprias das tarefas e dos processadores) são completamente satisfeitas. Um sequenciamento é *ótimo* se ele minimiza um dado critério de otimalidade. O conjunto de características dos processadores, das tarefas e o critério de otimalidade definem um tipo de problema que pode ser representado por uma classificação de três campos $\alpha \mid \beta \mid \gamma$, introduzida por LAWLER et al. (1989).

O primeiro campo α, representa as características dos processadores e é subdividido em dois campos α_1 e α_2 . Seja ° representando o símbolo vazio.

Se $\alpha_1 \in \{°, P, Q, R\}$, cada tarefa J_j consiste de uma operação simples que pode ser realizada em qualquer processador M_i ; o tempo de processamento de J_j em M_i será representado por p_{ij} . Estes quatro valores de α_1 podem ser descritos como:

- $\alpha_1 = °$: processador único; $p_{1j} = p_j$;
- $\alpha_1 = P$: processadores paralelos idênticos; $p_{ij} = p_j$ para todo M_i ;
- $\alpha_1 = Q$: processadores paralelos uniformes; $p_{ij} = p_j / s_i$ para uma dada velocidade de processamento s_i de M_i ;

- α_1 = R : processadores paralelos não-relacionados; $p_{ij} = p_j / s(i,j)$ para um conjunto de velocidades de processamento $s(i,j)$ dependentes da tarefa j de M_i .

Se α_1 = O, tem-se um *Open Shop*, no qual J_j consiste de um conjunto de operações $(O_{1j}, \ldots, O_{mj})$. O_{ij} deve ser processado em M_i durante p_{ij} unidades de tempo, mas a ordem em que estas operações são executadas é irrelevante. Se $\alpha_1 \in \{F, J\}$, uma ordenação é imposta no conjunto de operações correspondente a cada tarefa. Se α_1 = F, tem-se um *Flow Shop*, no qual cada J_j consiste numa cadeia $(O_{1j}, \ldots, O_{mj})$. O_{ij} deve ser processado em M_i durante p_{ij} unidades de tempo. Se α_1 = J, tem-se um *Job Shop*, no qual cada J_j consiste numa cadeia $(O_{1j}, \ldots, O_{mj})$. O_{ij} deve ser processado em um dado processador u_{ij} durante p_{ij} unidades de tempo, com $u_{ij} \neq u_{i+1,j}$, para $i = 1, \ldots, m_j - 1$.

Se α_2 é um inteiro positivo, então m é uma constante igual a α_2 ; isto é especificado como parte do tipo de problema. Se α_2 = ° , então m é variável e seu valor depende do problema em questão. Obviamente, α_1 = ° se e somente se $\alpha_2 = 1$.

O segundo campo $\beta \subset \{\beta_1, \ldots, \beta_4\}$ indica um número de características das tarefas que são definidas como:

1. $\beta_1 \in \{pmtm, °\}$
   β_1 = pmtm : Preempção (divisão da tarefa) é permitida; o processamento de cada operação pode ser interrompido e retomado tempos depois;
   β_1 = ° : Preempção não é permitida.
2. $\beta_2 \in \{prec, tree, °\}$
   β_2 = prec : uma relação de precedência $\rightarrow$ é especificada entre as tarefas. Isto é derivado de um grafo direto acíclico G com um conjunto de vértices $\{1, \ldots, n\}$. Se G contém um caminho direto de j para k, escreve-se $J_j \rightarrow J_k$ e é necessário que J_j seja finalizado antes de J_k se iniciar;
   β_2 = tree : G é uma árvore enraizada, com no máximo um arco saindo de cada vértice ou no máximo um arco chegando em cada vértice;
   β_2 = ° : nenhuma relação de precedência é especificada.
3. $\beta_3 \in \{r_j, °\}$
   $\beta_3 = r_j$: tempos de disponibilidade são especificados e podem diferir para cada tarefa;
   β_3 = ° : todo $r_j = 0$.

4. $\beta_4 \in \{p_j = 1, p_{ij} = 1, °\}$
   β_4 = $p_j = 1$: cada tarefa tem um tempo de processamento unitário. Isto ocorre somente se $\alpha_1 \in \{P, Q, °\}$;

β_4 = $p_{ij} = 1$: cada operação tem um tempo de processamento unitário. Isto ocorre somente se $\alpha_1 \in \{F, J, °\}$;

β_4 = ° : todo p_j ou p_{ij} são inteiros não- negativos quaisquer.

O terceiro campo $\gamma \in \{f_{max}, \Sigma f_j\}$ é relativo ao critério de otimalidade. Dado um sequenciamento, pode-se calcular para cada J_j:

- o tempo de finalização : C_j;
- a demora : $L_j = C_j - d_j$, (onde d_j é o tempo limite de finalização de uma tarefa J_j);
- o atraso : $T_j = \max\{0, C_j - d_j\}$;
- a penalidade unitária : $U_j = 0$ se $C_j \leq d_j$ e $U_j = 1$ caso contrário.

O critério de otimalidade geralmente escolhido envolve a minimização de $f_{max} \in \{C_{max}, L_{max}\}$, onde $f_{max} = \max_{1 \leq j \leq n} f_j(C_j)$, com $f_j(C_j) = C_j, L_j$, respectivamente, ou de $\Sigma f_j \in \{\Sigma C_j, \Sigma T_j, \Sigma U_j, \Sigma w_j C_j, \Sigma w_j T_j, \Sigma w_j U_j\}$, onde $\sum f_j = \sum_{j=1}^{n} f_j(C_j)$ com $f_j(C_j) = C_j, T_j, U_j, w_j C_j, w_j T_j, w_j U_j$ respectivamente e w_j é um peso que indica a importância relativa da tarefa J_j e $f_j(t)$ é uma função real não decrescente que mede o custo que ocorre se J_j é finalizado no tempo t.

Deve-se notar que $\Sigma w_j C_j$ e $\Sigma w_j L_j$ diferem por uma constante $\Sigma w_j d_j$ e, portanto, são equivalentes. Além disto, qualquer sequenciamento minimizando L_{max} também minimiza T_{max} e U_{max}, mas não vice-versa.

De acordo com esta classificação os problemas estudados nesta tese são o $P \mid\mid C_{max}$ e o $P \mid tpds \mid C_{max}$. O $P \mid\mid C_{max}$ é definido como alocar n tarefas a m processadores idênticos de uma forma não preemptiva, objetivando minimizar o tempo de finalização máximo das tarefas (*makespan*). Já o $P \mid tpds \mid C_{max}$ necessita da definição de t_{ij} e de um novo campo β ainda não definido. O $P \mid tpds \mid C_{max}$ é o $P \mid\mid C_{max}$ com tempos de processamento dependentes da seqüência, isto é, define-se um $\beta_5 \in \{tpds, °\}$, como sendo:

β_5 = tpds : tempos de processamento dependentes da seqüência, isto é, o tempo de processamento de uma tarefa J_j é composto da soma de duas parcelas: um tempo de execução inteiro e positivo p_j e o tempo de preparação inteiro e positivo para a tarefa J_j se ela for executada imediatamente após a tarefa J_i, chamado s_{ij}. Portanto, o tempo de processamento pode ser expresso como: $t_{ij} = p_j + s_{ij}$;

β_5 = ° : não existe nenhuma relação de dependência entre as tarefas.

O objetivo do $P \mid tpds \mid C_{max}$, continua sendo minimizar o tempo de finalização máximo das tarefas. Os problemas de sequenciamento com tempos de processamento dependentes da seqüência estão intimamente ligados ao problema do caixeiro viajante e suas aplicações. Uma revisão bibliográfica do $P \mid \mid C_{max}$ e do $P \mid tpds \mid C_{max}$, bem como algumas considerações sobre complexidade, serão o tema deste capítulo.

1.2. CONSIDERAÇÕES SOBRE COMPLEXIDADE

Na prática, constata-se que alguns problemas são mais fáceis de resolver que outros. Para alguns problemas de sequenciamento existem algoritmos capazes de resolver problemas com milhares de tarefas, enquanto que para outros problemas, os melhores algoritmos existentes encontram dificuldades para resolver exemplos com somente algumas poucas tarefas.

A teoria da complexidade fornece um arcabouço matemático no qual os problemas computacionais podem ser classificados como de *fácil* ou de *difícil* resolução. Uma revisão dos principais pontos desta teoria será feita baseada nos trabalhos de KARP (1975), GAREY e JOHNSON (1978), LENSTRA e RINNOOY KAN (1979), SHMOYS e TARDOS (1989) e STOCKMEYER (1990) e nos livros de GAREY e JOHNSON (1979) e PAPADIMITRIOU e STEIGLITZ (1982), e estará fundamentada na análise de pior caso do tempo computacional ou na qualidade da solução encontrada.

Um problema computacional pode ser visto como uma função f que mapeia cada entrada x, em um certo domínio, para uma saída $f(x)$. Apesar de haver várias maneiras de representar o domínio de entrada para um dado problema, isto será sem importância no momento. O interesse aqui é estudar o tempo necessário para calcular $f(x)$ como uma função do tamanho da codificação da entrada x, representado por $|x|$. Diante do que nos propusemos nesta discussão será suficiente pensar em termos de qualquer linguagem padrão de programação, considerando um algoritmo que calcula $f(x)$ de uma entrada x. A eficiência deste algoritmo será medida por um limitante superior $T(n)$ do número de passos que o algoritmo gasta com qualquer entrada x com $|x|=n$.

Na prática é desnecessário determinar a função $T(n)$ completamente. Tudo o que se precisa é uma indicação de como é o seu comportamento à medida que o tamanho do problema aumenta. Assim pode-se dizer que $T(n)$ é $O(g(n))$ (leia-se T e g são de mesma

ordem), sempre que: $T(n) / O(g(n)) \rightarrow c$, à medida que $n \rightarrow \infty$, onde c é uma constante qualquer.

Assim $T(n)$ é $O(g(n))$ se sua divisão tende a uma constante à medida que n cresce. Em outras palavras o comportamento de $T(n)$ e $g(n)$ se tornam cada vez mais semelhantes até ficarem exatamente iguais. A partir desta definição pode-se derivar as seguintes propriedades:

Se $T(n)$ é $O(n^k)$ e $g(n)$ é $O(n^l)$, então:

(*i*) $T(n)\, g(n)$ é $O(n^{k+l})$;

(*ii*) $T(n) / g(n)$ é $O(n^{k-l})$;

(*iii*) $T(g(n))$ é $O(n^{kl})$.

Além disso, o polinômio $(a_k n^k + a_{k-1} n^{k-1} + \ldots + a_0)$ é $O(n^k)$. Dizemos que o algoritmo tem complexidade de tempo polinomial (isto é, o problema pode ser considerado de *fácil* resolução) se $T(n)$ é $O(p(n))$ para algum polinômio $p(n)$. Caso contrário o algoritmo tem complexidade de tempo exponencial.

Até aqui vimos a diferença entre dois tipos de algoritmos: aqueles com complexidade de tempo polinomial e aqueles com complexidade de tempo exponencial. Isto nos permite definir duas classes: ***P*** e ***NP***. Note que, estas são classes de problemas e não de algoritmos.

A classe ***P*** consiste de todos os problemas para os quais são conhecidos algoritmos com tempo polinomial. Essencialmente, a classe ***NP*** é o conjunto de problemas para os quais foram encontrados algoritmos de tempo exponencial. Fica claro que ***P*** está contido em ***NP***, pois, se temos um algoritmo de tempo polinomial para um problema, sempre há meios de torná-lo ineficiente até atingir um tempo exponencial. Ocasionalmente, tem-se o caso de um problema originalmente em ***NP*** (porém, fora de ***P***) ser movido para ***P*** , caso seja descoberto um algoritmo de tempo polinomial para ele.

Porém, observa-se na prática que para inúmeros problemas ainda não foram propostos algoritmos polinomiais, a despeito do grande esforço de pesquisa dedicado a busca de algoritmos eficientes. Assim, surge uma das grandes conjecturas da matemática moderna: a existência de problemas em ***NP*** que nunca poderão ser movidos para ***P***. Em outras palavras, muitos matemáticos (COOK - 1971, KARP - 1972 e LEVIN - 1975) acreditam que algoritmos com tempo polinomial nunca serão encontrados para algumas classes de problemas, simplesmente porque eles realmente são extremamente difíceis. A maioria dos problemas de sequenciamento pertencem a esta categoria (FRENCH - 1986).

Antes de discutir a conjectura $P \neq NP$, é preciso formalizar um pouco mais a definição dada a estas classes. Primeiro, imaginemos que o tipo de problemas em P e NP são *problemas de otimização*, por exemplo: encontre um sequenciamento ótimo para o problema $J3 \mid p_{ij}=1 \mid Cmax$ (minimizar o tempo máximo de finalização de um Job Shop, em três processadores, com tempos de processamento unitários). Porém, de acordo com a definição dada pela ciência da computação para as classes P e NP , os problemas de otimização não pertencem a nenhuma delas.

Portanto, devemos concentrar a atenção nos problemas conhecidos como *problemas de decisão*, que respondem questões sobre existência, aceitando somente respostas SIM ou NÃO. Por exemplo: para um dado problema $J3 \mid p_{ij}=1 \mid Cmax$, existe um sequenciamento com $Cmax \leq 7$? Entretanto, restringir as classes P e NP aos problemas de decisão não impede a análise dos problemas de otimização, pois problemas de otimização têm uma solução em tempo polinomial se e somente se o problema de decisão associado a ele também tem. Considere o exemplo: se pudermos encontrar o sequenciamneto ótimo para o problema $J3 \mid p_{ij}=1 \mid Cmax$, poderemos checar se o tempo máximo de finalização é menor que 7 e, então, responder ao problema de decisão associado. Se o tempo necessário para encontrar o sequenciamento ótimo é polinomial, responder ao problema de decisão não aumentará de grande quantidade este tempo. Portanto o problema de decisão associado pertence a P . Por outro lado, se o problema de decisão associado pudesse ser respondido em tempo polinomial, o problema de otimização que o gerou também poderá. Assim, um problema de otimização e o problema de decisão a ele associado têm o mesmo grau de dificuldade, ou seja, se existe um algoritmo de tempo polinomial para responder ao problema de decisão associado, o mesmo deve ser verdadeiro para o problema de otimização.

Definiu-se como NP a classe de problemas para os quais existem algoritmos de tempo exponencial. Entretanto, uma melhor definição pode ser dada usando-se o conceito de problema de decisão. Um problema de sequenciamento de decisão está em NP se é possível, em tempo polinomial, achar um sequenciamento factível candidato e verificar se este sequenciamento produz uma resposta SIM para o problema de decisão associado. Esta definição implica que algoritmos de tempo exponencial existem para todos os problemas em NP . Por exemplo, para um dado problema de tamanho n, cada processo de encontrar um sequenciamento factível candidato e checar sua validade em relação ao problema de decisão associado leva menos tempo que $p(n)$, para algum polinômio $p(n)$. Pode-se dizer que existirão, no máximo, $n!$ sequenciamentos possíveis. Assim, se considerarmos um sequenciamento factível candidato por vez, levaríamos um tempo de

$O(n!p(n))$ para fazer uma verificação exaustiva das alternativas de sequenciamento. Resumindo, a enumeração completa do problema toma um tempo exponencial.

Tendo definido ***P*** e ***NP*** , voltemos à conjectura que $\boldsymbol{P \neq NP}$ e suas implicações aos problemas de sequenciamento. Um conceito indispensável nesta discussão será a idéia de *reduzir* um problema a outro. FRENCH (1986) mostra que certos problemas de sequenciamento podem ser transformados em um problema de programação inteira. Seguindo a terminologia adotada, diz-se que o problema de sequenciamento foi reduzido a um problema de programação inteira. É claro que este é um exemplo envolvendo problemas de otimização, porém, a idéia de redutibilidade é idêntica quando se trata de problemas de decisão.

Suponha que se possa reduzir um problema Π_1 em outro problema Π_2 em tempo polinomial, então diz-se que Π_1 é polinomialmente redutível ao problema Π_2 , e escreve-se: $\Pi_1 \propto \Pi_2$. Isto significa que, dado Π_1 de tamanho n, pode-se construir Π_2 em $p(n)$ operações, para um dado polinômio $p(n)$. Além disto, os dois problemas são equivalentes no sentido de que o primeiro tem uma resposta SIM se e somente se o segundo também tem. Nota-se também que o tamanho de Π_2 deve ser menor que $p(n)$ (GAREY e JOHNSON - 1979).

Suponha que $\Pi_1 \propto \Pi_2$ e que Π_2 está em ***P***. Então, pode-se construir um algoritmo de tempo polinomial para responder Π_1 , demonstrando que ele também pertence a ***P***. Dado Π_1 de tamanho n podemos reduzi-lo, em um tempo limitado por um polinômio $p(n)$, a Π_2 com tamanho máximo igual a $p(n)$. Como Π_2 pertence a ***P***, qualquer Π_2 de tamanho η pode ser resolvido em um tempo polinomial $q(\eta)$. Assim, pode-se resolver o problema em questão em $q(p(n))$. Adicionando o tempo gasto na redução, pode-se resolver o problema original Π_1 em $p(n) + q(p(n))$, que continua sendo um tempo polinomial. De outro modo, se $\Pi_1 \propto \Pi_2$ e se, neste caso, Π_1 não está em ***P***, consequentemente, Π_2 também não estará. Se $\Pi_1 \propto \Pi_2$, então problemas do tipo Π_2 devem ser, no mínimo, tão difíceis de resolver quanto os problemas do tipo Π_1 . Esta classificação dos problemas quanto a sua dificuldade, será vital para o entendimento do que segue.

Considerando-se Π_1 , Π_2 e Π_3 problemas quaisquer, então: $(\Pi_1 \propto \Pi_2$ e $\Pi_2 \propto \Pi_3) \Rightarrow \Pi_1 \propto \Pi_3$, o que define a propriedade *transitiva* de $\propto$.

Diz-se que um problema Π em ***NP*** é ***NP*-completo**, se todos os outros problemas em ***NP*** são polinomialmente redutíveis a Π , isto é, $\Pi' \propto \Pi$ para todo Π' em ***NP*** . Assim, os problemas ***NP***-completos formam uma subclasse de ***NP*** . Além disso, esta subclasse

consiste dos problemas mais difíceis de ***NP*** . Portanto, se encontrarmos um algoritmo de tempo polinomial para algum problema ***NP***-completo, então poderemos responder todos os problemas em ***NP*** em tempo polinomial. Então, se tal algoritmo for encontrado, $P = NP$ e a conjectura de que $P \neq NP$ é equivalente à conjectura de que nenhum problema ***NP*** - completo pode ser respondido em tempo polinomial.

Até aqui não se falou nada sobre a codificação das entradas, porém existe uma diferença que irá ter um papel importante nesta exposição. A maneira natural de codificar inteiros é usar a notação binária (por exemplo, 5 = <101>). Entretanto, pode-se também considerar a notação unária (por exemplo, 5 = <11111>). Existe uma diferença exponencial entre os tamanhos dos dois códigos. No *problema de partição*, a entrada consiste de n números a_1 , ..., a_n e a questão a ser respondida é se existe um subconjunto $S \subset \{1 , \ldots , n\}$ tal que $\sum_{j \in S} a_j = \sum_j a_j / 2$. Este problema é ***NP***-completo sobre uma codificação binária. Por outro lado, ele pode ser resolvido por programação dinâmica em um tempo $O\left(n \sum_j a_j\right)$, que é polinomial sobre uma codificação unária. Este método é então chamado de algoritmo de tempo pseudo-polinomial. Existem também problemas envolvendo números que são ***NP***-completos mesmo quando os números estão sob uma codificação unária. No *problema de tri-partição*, a entrada consiste de $3n$ inteiros a_1 , ..., a_{3n} e a questão a ser respondida é se existe uma partição de $\{1 , \ldots , 3n\}$em n conjuntos de três elementos S_1 , ..., S_n , tais que $\sum_{j \in S_i} a_j = \sum_j a_j / n$ para $i = 1 , \ldots , n$. Este problema permanece ***NP***-completo sob uma codificação unária e é chamado de ***NP***-completo num sentido forte.

Um problema de otimização ***NP***-difícil é aquele cujo problema de decisão associado é ***NP***-completo. Um problema de otimização ***NP***-difícil sugere que não é sempre possível encontrar a solução ótima de uma maneira rápida. Entretanto, ainda é possível o uso de *algoritmos aproximados* (algoritmos heurísticos) para encontrar soluções que, provavelmente, são próximas da ótima. Para um problema de minimização f, um algoritmo ρ-aproximado ($\rho > 1$) encontra, garantidamente, uma solução com valor máximo de $\rho f(x)$ para cada entrada x. Para alguns problemas ***NP***-difíceis há um *esquema aproximado polinomial*, que é uma família de algoritmos $\{A_\varepsilon\}$ tal que, para cada $\varepsilon > 0$, A_ε é um algoritmo $(1+\varepsilon)$-aproximado com tempo polinomial. O tempo computacional de A_ε pode

depender não somente do tamanho da entrada, mas do valor de ε. Se ele é limitado por um polinômio em |x| e 1/ε, então a família é chamada de *esquema de aproximação totalmente polinomial*.

1.3. O PROBLEMA $P\,|\,|\,Cmax$

O problema $P\,|\,|\,Cmax$ consiste em alocar n tarefas independentes, J_1 , ... , J_n a m processadores paralelos idênticos M_1 , ... , M_m , de uma forma não-preemptiva. Supondo $n \geq m \geq 2$ e que cada tarefa J_j tem um tempo de execução inteiro e positivo p_j , então o objetivo do problema é minimizar o tempo de finalização máximo das tarefas (*makespan*) definido como: $Cmax = \max_{i=1,\ldots,m} \{C_i\}$, onde C_i é o tempo de execução de todas as tarefas alocadas ao processador M_i .

Uma possível formulação matemática do $P|\,|Cmax$ é:

$$x_{ij} = \begin{cases} 1, \text{ se a tarefa } j \text{ é alocada ao processador } i \\ 0, \text{ caso contrário} \end{cases}$$

$$Min\ \sigma$$

$$tal\ que \sum_j p_j x_{ij} \leq \sigma, \quad \forall i$$

$$\sum_i x_{ij} = 1, \quad \forall j$$

$$x_{ij} \in \{0,1\}, \quad \forall i, j$$

O $P|\,|Cmax$ é um dos problemas mais estudados da teoria de sequenciamento. GAREY e JOHNSON (1979) provaram que este problema é *NP*-difícil num sentido forte quando o número de processadores é arbitrário. O problema pode ser resolvido em um tempo pseudo-polinomial quando o número de processadores é fixo, sendo assim *NP*-difícil somente no sentido ordinário. Apesar disso é conhecido um grande número de algoritmos heurísticos para o $P|\,|Cmax$.

Estes algoritmos podem ser classificados em *heurísticas construtivas* e *heurísticas de melhoramento*. As heurísticas construtivas, geralmente, fazem uma pré-ordenação nas tarefas e alocam-nas aos processadores uma a uma de acordo com alguma regra de alocação. As heurísticas de melhoramento partem de uma solução inicial qualquer

(podendo até mesmo ser uma obtida por uma heurística construtiva) e então procuram melhorar esta solução inicial através da troca de uma ou mais tarefas, até que um critério de parada seja satisfeito. A maioria destes algoritmos pertence à primeira categoria e, geralmente, tem conhecido seu limitante de desempenho de pior caso. O limitante de desempenho de pior caso pode ser definido como: dado um procedimento heurístico H para um problema P, seja H(I) e OPT(I) o valor produzido por H e o valor da solução ótima, respectivamente, para um dado exemplo I de P. O limitante de desempenho de pior caso de H é o máximo valor R(H) tal que: $H(I) / OPT(I) \leq R(H), \forall I$.

Dentre os algoritmos heurísticos construtivos, pode-se dizer que o primeiro foi publicado por GRAHAM (1966) e foi denominado LIST SCHEDULING. As tarefas são organizadas em uma lista, sem qualquer ordenação prévia, sendo então alocadas, uma a uma, ao processador menos carregado. Este algoritmo tem um valor de desempenho de pior caso igual a $2\text{-}1/m$. Nota-se que LIST SCHEDULING garante uma solução com tempo máximo de finalização menor do que duas vezes o ótimo. Uma vez que sempre existe uma dada ordenação das tarefas que produz a solução ótima dentro de um esquema de LIST SCHEDULING, é natural que se considere alguns refinamentos neste algoritmo.

Em 1969, o mesmo GRAHAM, sugeriu uma ordenação das tarefas em ordem decrescente de tempos de processamento e então aplicava o LIST SCHEDULING. Este esquema recebeu a denominação de LPT (*Longest Processing Time first*) e melhorou o valor de desempenho de pior caso para $4/3 - 1/3m$. LPT tem uma complexidade de tempo igual a $O(nlogn)$, dominada pela ordenação inicial das tarefas. Testes numéricos mostram que LPT tem um bom desempenho, especialmente quando a dimensão de n aumenta (FINN e HOROWITZ - 1979, FRANÇA e MÜLLER - 1992 e FRANÇA et al. - 1993a).

Outra heurística construtiva clássica é a apresentada por COFFMAN, GAREY e JOHNSON (1978) chamada MULTIFIT. Está baseada em um princípio diferente dos procedimentos de LIST SCHEDULING; ele resolve uma série de Problemas de Empacotamento (*Bin Packing Problems*) que têm estreita relação com o $P|\ |Cmax$.

O modelo básico para o Problema de Empacotamento (PE) é o seguinte: dada uma capacidade C positiva e inteira, de cada caixa (bin) e um conjunto de itens $L = (q_1, q_2, \ldots, q_n)$, cada item q_i tendo um tamanho inteiro e positivo $s(q_i)$ satisfazendo $0 \leq s(q_i) \leq C$, o objetivo é encontrar o menor inteiro m tal que exista uma partição $L = B_1 \cup B_2 \cup \ldots \cup B_m$ satisfazendo $\sum_{q_i \in B_j} s(q_i) \leq C,\ 1 \leq j \leq m$. Geralmente, imagina-se cada conjunto B_i como

sendo uma caixa de capacidade C e tenta-se minimizar o número de caixas necessárias para empacotar L.

O $P|\ |Cmax$ pode ser visto como um Problema de Empacotamento (*Bin Packing Problem*). Para uma dada capacidade C e um conjunto de tarefas (itens) $J=(J_1, J_2, \ldots, J_n)$ cada uma tendo um tempo de processamento (tamanho) inteiro e positivo p_j ,com $0 \leq p_j \leq C$. O objetivo é encontrar o mínimo valor de m tal que exista uma partição de J em subconjuntos $J_1', J_2', \ldots, J_m'$ satisfazendo $\sum_{J_i \in J_k'} t_i \leq C$, para $k = 1, \ldots, m$.

O procedimento MULTIFIT resolve heuristicamente uma série de Problemas de Empacotamento que dirão se devemos aumentar ou diminuir a capacidade dos processadores dependendo se o número de processadores necessários para alocar as n tarefas é maior ou menor que os m disponíveis. (COFFMAN, GAREY e JOHNSON - 1984). É mostrado também que se são feitas k tentativas de alocação das tarefas, MULTIFIT tem um tempo computacional de $O(nlogn + knlogm)$, que é independente de m e tem um valor de desempenho de pior caso de $11/9 + 2^{-k}$, também independente de m.

Este limitante de pior caso ainda foi melhorado através de refinamentos no procedimento de busca dicotômica incluso em MULTIFIT. Em 1984, FRIESEN chegou ao valor de $1,2 + 2^{-k}$, posteriormente melhorado ainda mais por FRIESEN e LANGSTON (1986), chegando ao valor de $72/61 + 2^{-k}$.

HOCHBAUM e SHMOYS (1987) usaram uma variação de MULTIFIT para fornecer um esquema polinomial aproximado para o $P|\ |Cmax$, que faz uma substituição do algoritmo aproximado (tradicional) na busca dicotômica por um *algoritmo dual aproximado*. Usar um algoritmo ρ-dual aproximado dentro da busca dicotômica, por k iterações, irá produzir um algoritmo $(\rho + 2^{-k})$-aproximado para o $P|\ |Cmax$. Foram apresentados algoritmos para $\rho = 1/5$ e $\rho = 1/6$, que apresentaram tempos computacionais de $O(n\ (k + logn))$ e $O(n\ (km^4 + logn))$, respectivamente.

Em 1988, LEE e MASEY apresentaram uma heurística construtiva combinando as características de rapidez do algoritmo LPT e o bom desempenho de pior caso de MULTIFIT, resultando em um algoritmo de tempo computacional $O(nlogn + knlogm)$, que é o mesmo de MULTIFIT. O algoritmo desenvolvido mostrou um bom desempenho, superando LPT e MULTIFIT, porém notou-se nos resultados uma semelhança muito

grande no desempenho das heurísticas LPT e MULTIFIT no que diz respeito à qualidade da solução.

KARMARKAR e KARP (1988) desenvolveram um algoritmo baseado numa operação denominada *set differencing* que, como a maioria das heurísticas construtivas, tem seu melhor desempenho para valores grandes de n. Este algoritmo fica melhor ilustrado no caso em que $m = 2$. A operação de *set differencing* consiste em selecionar um par de tarefas com tempos computacionais p_i e p_j, do conjunto de tarefas $J = (J_1, J_2, \ldots, J_n)$ e restringir a solução a alocações onde J_i e J_j apareçam em processadores diferentes. O novo e menor problema é equivalente a particionar $J' = J - \{J_i, J_j\} \cup \{|p_i - p_j|\}$, até que a cardinalidade do conjunto gerado para a nova partição seja igual a 1.

Dentre as heurísticas de melhoramento que procuram através de trocas e realocações de tarefas, melhorar uma solução inicial factível obtida de uma maneira qualquer, pode-se citar o algoritmo denominado 0/1-INTERCHANGE apresentado por FINN e HOROWITZ em 1979. O algoritmo 0/1-INTERCHANGE inicia alocando n tarefas aos m processadores em uma ordem aleatória. Então faz uma ordenação dos processadores de modo que $C_1 \geq C_2 \geq \ldots \geq C_m$. Seja $d = C_1 - C_m$ a diferença entre os tempos de finalização do processador mais carregado e do menos carregado. Se existir uma tarefa no processador M_1, cujo tempo de processamento p_j é menor que d, então esta tarefa é retirada de M_1 e realocada em M_m. O algoritmo continua com uma nova ordenação dos processadores de acordo com suas novas cargas, recalcula d e busca por uma tarefa no novo processador M_1 cujo tempo de processamento seja menor que d e realiza a troca. O algoritmo continua até que nenhuma tarefa em M_1 satisfaça a condição $p_j < d$. Este algoritmo roda em um tempo computacional de $O(nlogm)$ e o valor de seu limitante de desempenho de pior caso é 2. Porém, os autores afirmam que este limitante cai para um valor menor que o encontrado por COFFMAN, GAREY e JOHNSON (1978) para o algoritmo MULTIFIT, se o número de tarefas alocadas ao processador mais carregado for superior a 6, para um valor qualquer de m.

Em 1982, LANGSTON trabalhou em refinamentos no procedimento de alocação inicial do algoritmo 0/1-INTERCHANGE, procurando evitar alocações iniciais pobres, ocasionadas pela distribuição inadequada de tarefas com tempos de processamento grandes, uma vez que a heurística de trocas não consegue movê-las para outros processadores. O algoritmo por ele desenvolvido ficou conhecido como IMPROVED 0/1-INTERCHANGE. Assim que a fase inicial foi incluída, onde as m maiores tarefas são alocadas uma para cada processador, selecionam-se outras m maiores tarefas ainda não alocadas. Elas são classificadas em ordem decrescente de tempos de processamento.

Então, o algoritmo de LIST SCHEDULING é aplicado a estas m tarefas, ou seja, elas são alocadas, sucessivamente, ao processador menos carregado. A inclusão desta fase intermediária não muda o limitante de tempo computacional, que continua sendo igual a $O(nlogm)$, porém o limitante de desempenho de pior caso do algoritmo IMPROVED 0/1-INTERCHANGE cai para 4/3 - 1/3 m.

No que tange a resolução exata do $P \mid\mid Cmax$, pouco tem sido feito. Para pequenos valores de m e UB (um limitante superior sobre C^*max, onde C^*max representa o valor da solução ótima do $P \mid\mid Cmax$), o problema pode ser resolvido com exatidão em um tempo computacional de $O(n\mathrm{UB}^m)$ através de programação dinâmica, como descrito em BLAZEWICZ (1987). Este procedimento de programação dinâmica se mostrou bastante ineficiente não podendo ser usado para problemas de grande dimensão, uma vez que seu tempo de execução é exponencial em m (DELL'AMICO e MARTELLO - 1990). Um algoritmo exato para uma generalização do $P \mid\mid Cmax$, no qual as tarefas têm restrições de tempo para o início e término de sua execução, foi proposto por BRATLEY, FLORIAN e ROBILLARD (1975).

Outro algoritmo para solução exata do $P \mid\mid Cmax$ pode ser definido à partir de algoritmos exatos que resolvem o Problema de Empacotamento (PE). Tendo um limite inferior LB e um limite superior UB sobre C^*max, pode-se resolver o $P \mid\mid Cmax$ determinando, através de uma busca dicotômica, o menor valor de CAP - capacidade de cada caixa ($LB \leq CAP \leq UB$), tal que o PE tenha uma solução usando não mais que m caixas. O PE é *NP*-difícil num sentido forte, porém MARTELLO e TOTH (1990) apresentaram um algoritmo de branch & bound eficiente para sua solução. O código FORTRAN (denominado de MTP) mostrou que, em muitos casos, encontra-se a solução ótima para o PE. DELL´AMICO e MARTELLO (1990) mostraram que o procedimento MTP resolve vários exemplos para os quais programação dinâmica é inadequada. Porém, neste mesmo artigo eles apresentaram um procedimento de branch & bound para a solução exata do $P \mid\mid Cmax$ diretamente. Este novo algoritmo supera os algoritmos baseados em resoluções de PE.

Nesta tese são propostos dois novos algoritmos, um heurístico e um exato, para o $P \mid\mid Cmax$. O algoritmo heurístico, chamado de 3-FASES, pode ser classificado, à luz dos métodos relatados nesta revisão bibliográfica do $P \mid\mid Cmax$, como sendo uma heurística de melhoramento. Numa primeira fase, as tarefas são designadas aos processadores para a obtenção de uma solução inicial. Posteriormente, nas duas fases seguintes, as soluções são melhoradas através de procedimentos de trocas entre tarefas de diferentes processadores.

É provado que o algoritmo termina em um número finito de passos e um limitante de desempenho de pior caso é derivado.

O esquema ótimo proposto, também é baseado numa busca dicotômica, onde uma série de Problemas de Empacotamento são resolvidos. O LB usado é calculado de uma maneira descrita no Capítulo 2, enquanto que o UB usado é a solução encontrada pelo algoritmo 3-FASES, também apresentado no Capítulo 2. Para a solução exata do PE foi usado o código MTP de MARTELLO e TOTH (1990), cujo funcionamento é descrito de um modo sucinto ao final do Capítulo 2.

1.4. O PROBLEMA *P | tpds | Cmax*

Analogamente ao *P| |Cmax*, o problema consiste em alocar n tarefas J_1 , ... , J_n a m processadores paralelos idênticos M_1 , ... , M_m de uma forma não preemptiva (isto é, uma vez que determinada tarefa esteja alocada a um processador, ela deverá permanecer no mesmo até o seu término, sem interrupções), objetivando minimizar o tempo de finalização de todas as tarefas (*makespan*).

Porém, no *P|tpds|Cmax* o tempo de processamento de uma tarefa J_j é composto da soma de duas parcelas: um *tempo de execução* inteiro e positivo p_j e um *tempo de preparação* inteiro e positivo s_{ij} para a tarefa J_j , se ela for executada imediatamente após a tarefa i. Portanto o *tempo de processamento* pode ser expresso como $t_{ij} = p_j + s_{ij}$: é o tempo necessário para executar a tarefa j imediatamente após a tarefa i . Não suporta relação de simetria, isto é, $t_{ij} \neq t_{ji}$. Note que o problema pode ser visto como sendo o *P| |Cmax* com *tempos de processamento dependentes da seqüência* (*P|tpds|Cmax*). Este problema não trata simplesmente de alocar as tarefas aos processadores, mas procura encontrar também a melhor seqüência de execução das mesmas, de modo a minimizar o tempo máximo de execução. O *P|tpds|Cmax* é *NP*-difícil num sentido forte, pois inclui um *problema do caixeiro viajante*, que é *NP*-completo (KARP - 1972), como um caso especial (BAKER - 1974).

1.4.1. Caso $m = 1$

Inicialmente vamos revisar a literatura que trata o caso particular em que o número de processadores m é igual a 1. Neste caso o *P|tpds|Cmax* é equivalente ao Problema do Caixeiro Viajante (TSP) (BAKER - 1974). A definição clássica do TSP é que um caixeiro viajante deve visitar n cidades, passando em cada cidade uma única vez e retornando a cidade de origem. Dadas as distâncias entre todos os pares de cidades, a tarefa do vendedor é encontrar uma rota que minimize a distância total percorrida. O TSP pode ser simétrico ($t_{ij} = t_{ji}$) ou assimétrico ($t_{ij} \neq t_{ji}$), onde t_{ij} é a distância entre as cidades i e j. No problema *P|tpds|Cmax*, com $m = 1$, t_{ij} é o tempo de processamento da tarefa J_j se ela é executada imediatamente após a tarefa J_i, ou seja, t_{ij} corresponde à distância entre a cidade i e a cidade j. Igualmente, o *P|tpds|Cmax* com m=1 pode ser simétrico ou assimétrico.

Neste subitem, onde $m = 1$, seguiremos o seguinte plano de revisão de literatura. Inicialmente são apresentados os principais métodos exatos e heurísticos que explicitamente encaram o problema *P|tpds|Cmax* como um problema de sequenciamento. Obviamente, por ele ser equivalente ao TSP, qualquer método proposto para o TSP serve para o *P|tpds|Cmax*, com $m = 1$. Porém, não faremos uma revisão extensa dos métodos para o TSP. Por ser um dos problemas que mais têm recebido atenção dentro do universo dos problemas de programação matemática, limitamo-nos apenas a apresentar os algoritmos clássicos de k-OPT e o de LIN-KERNIGHAN e as mais recentes contribuições à resolução do TSP, devido a relação que eles apresentam com os métodos que são propostos nesta tese. Para uma revisão ampla dos métodos de solução do TSP, remetemos o leitor a algumas das excelentes revisões que foram publicadas (o livro de LAWLER et al. - 1985, o artigo de LAPORTE - 1992 para os métodos exatos, o artigo de JOHNSON - 1990, entre outros).

Iniciando, então, com os métodos endereçados diretamento ao *P|tpds|Cmax*, LOCKETT e MUHLEMANN (1972) desenvolveram um método, baseado no algoritmo de branch & bound para a solução do TSP desenvolvido por LITTLE et al. (1963),adaptando a escolha dos limitantes usados ao problema de sequenciamento. Porém, o método se mostrou ineficiente para todos os problemas testados, com exceção dos problemas de menor tamanho. Esta ineficiência foi resultado da distribuição dos tempos de preparação e da qualidade dos limitantes inferiores usados. LOCKETT e MUHLEMANN também fizeram testes com outras cinco heurísticas em problemas com 15 a 35 tarefas. As cinco heurísticas por eles testadas podem ser resumidamente descritas como segue: 1)Ordenação Aleatória: as tarefas são alocadas ao processador em uma ordem qualquer;

2)Caixeiro Viajante com Busca em Profundidade: é uma heurística derivada da solução do branch & bound. Está baseada numa matriz de penalidades previamente definida com a aplicação do algoritmo de LITTLE et al. (1963); 3)Caixeiro Viajante Ótimo para a Matriz de Tempos de Preparação: é também baseado na solução de um branch & bound tomando-se a solução do TSP para a matriz de penalidades; 4)Minimização da Troca de Ferramentas: é um algoritmo guloso que aloca, sucessivamente, ao processador a tarefa que requer um menor número de trocas de ferramentas para sua execução e 5)Vizinho mais Próximo: baseado na heurística descrita em CONWAY et al. (1967). Todas as heurísticas encontraram soluções para o problema. A única exceção foi o procedimento ótimo (número 3) que não resolveu nenhum problema, pois estava limitado a 30 minutos de CPU. Portanto, não se tem um parâmetro para avaliar a qualidade das soluções encontradas pelas outras heurísticas em relação à solução ótima.

WHITE e WILSON (1977) desenvolveram um método heurístico para a solução do $P|tpds|Cmax$ com m=1, onde os valores dos tempos de preparação estão baseados em dados reais e na experiência do usuário. A heurística funciona objetivando minimizar as mudanças que provoquem altos tempos de preparação. O conjunto total de tarefas é sucessivamente particionado usando-se a suposição que, uma vez definido este subconjunto, nenhuma tarefa fora dele será executada até que todas as tarefas pertencentes a ele estejam terminadas. Após a finalização de todas as tarefas pertencentes a um dado subconjunto, busca-se, dentre as tarefas ainda não alocadas, a primeira tarefa que dará seqüência à execução. Esta busca é feita usando um método para previsão dos tempos de preparação baseado em observações reais e estatísticas e nos valores dos tempos de preparação. Então, com base nas características da tarefa selecionada, um novo subconjunto de tarefas é identificado e separado do conjunto de tarefas ainda não alocadas. O processo de identificar alternadamente subconjuntos de tarefas e buscar a primeira tarefa que sucede estas tarefas continua até que todas as tarefas estejam alocadas. Esta heurística mostrou-se de fácil uso e requer poucos cálculos, além de usar as características dos tempos de preparação do processador em questão, dando ao problema um tratamento mais real.

A seguir revisaremos alguns métodos clássicos para resolução do TSP. Uma das mais conhecidas heurísticas para o TSP é aquela que envolve troca de arcos, conhecida como k-OPT (LIN - 1965 e LIN e KERNIGHAN - 1973). Na realidade trata-se de uma família de heurísticas que funcionam da seguinte maneira:

Passo 1: Encontre uma rota inicial. Geralmente esta rota é escolhida de uma maneira aleatória (porém isto não é uma imposição) dentre o conjunto de todas as rotas possíveis.

Passo 2: Melhore a rota através de uma das heurísticas de troca de arcos.
Passo 3: Repita o Passo 2 até que nenhuma melhoria seja possível.

Os procedimentos de troca de arcos terminam em um ótimo local. Para um dado k, define-se uma *k-troca* de uma rota como sendo a retirada de k arcos da rota e sua substituição por outros k arcos formando uma nova rota. Então, esta rota é k-ótima (k-OPT) se não é possível uma melhoria na rota através de uma k-troca. Os Passos 2 e 3 geram esta rota k-ótima. O procedimento de troca 2-OPT, geralmente, produz um ótimo local inferior ao 3-OPT. Do mesmo modo, um procedimento k-OPT encontrará um ótimo local melhor que o 3-OPT, para $k \geq 4$. A figura 1.1 mostra uma troca 2-OPT. Note que a troca está bem definida em cada passo. Uma vez decidido retirar os arcos (A,B) e (E,F) da solução atual, com o intuito de reduzir o tamanho da rota, deve-se introduzir os arcos (A,E) e (B,F), uma vez que introduzir (A,F) produz uma solução com subrotas e introduzir (A,B) reproduz a solução original. Note que, neste exemplo, a simetria é exigida, uma vez que a orientação dos arcos (B,C), (C,D) e (D,E) fica invertida.

A figura 1.2 mostra um exemplo onde três arcos são retirados de uma rota, deixando 3 subrotas desconexas, estas subrotas podem ser conectadas de oito modos diferentes. Quando este exercício é repetido para todas as combinações possíveis de retirar três arcos, uma rota 3-OPT é obtida para este caso (CHRISTOFIDES e EILON - 1969).

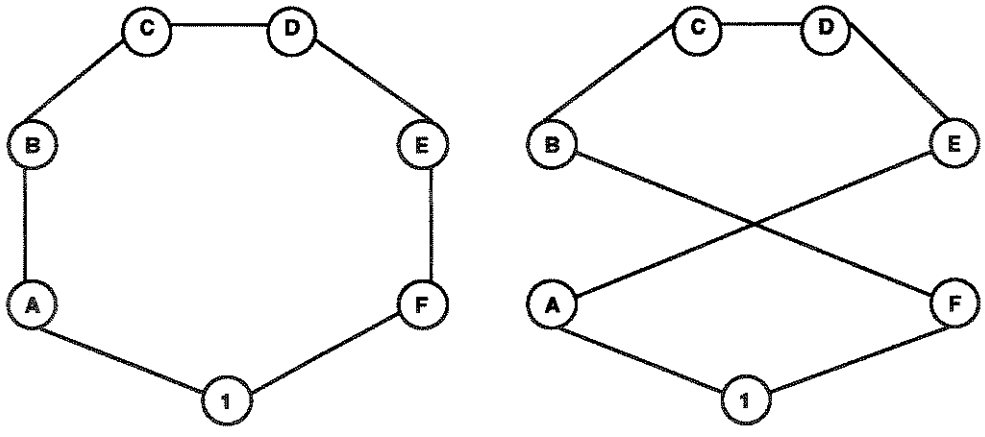


Figura 1.1 - Rota Inicial e uma Troca 2-OPT Factível

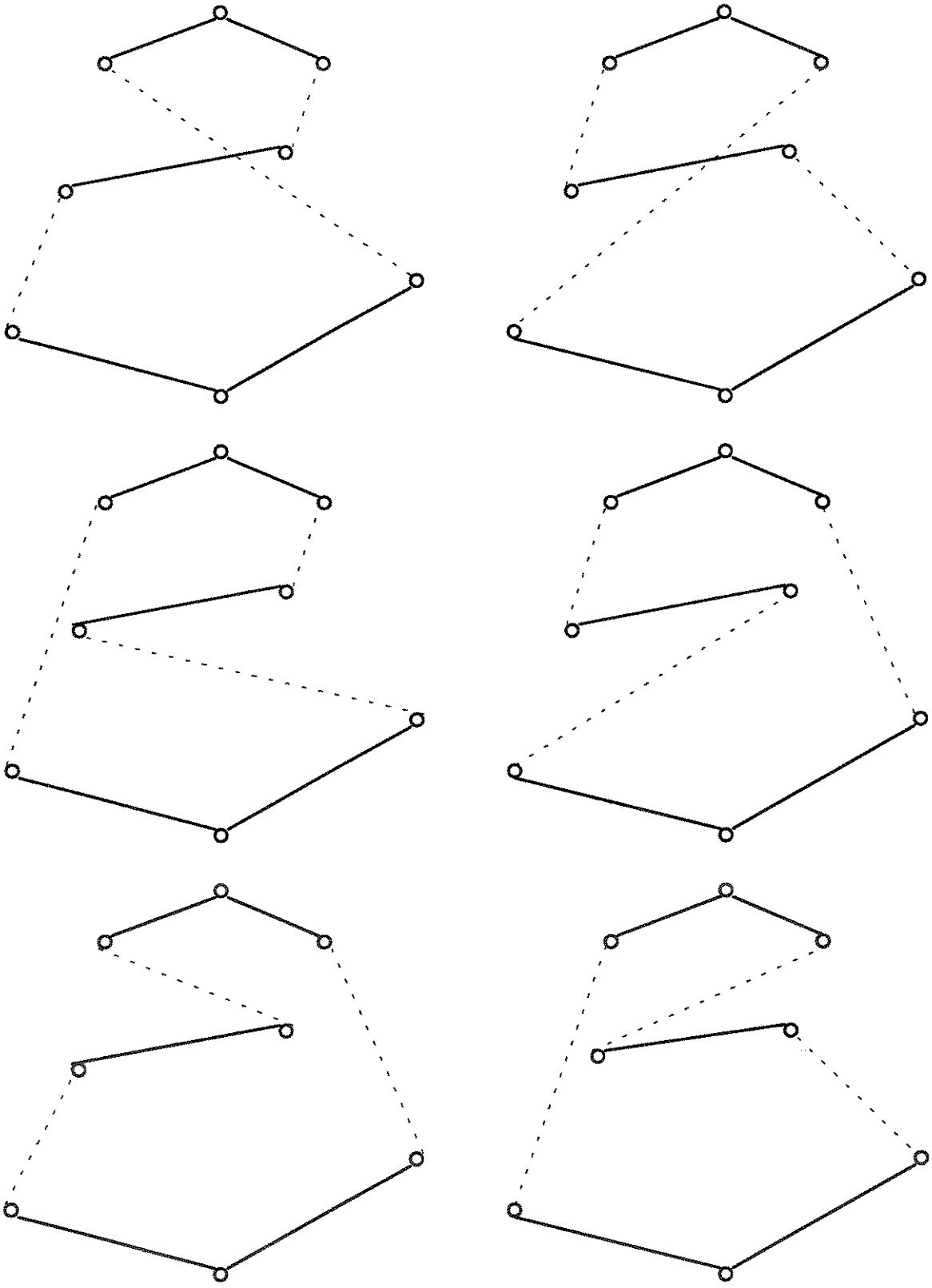


Figura 1.2 - Oito Maneiras de Conectar Três Cadeias Subrotas

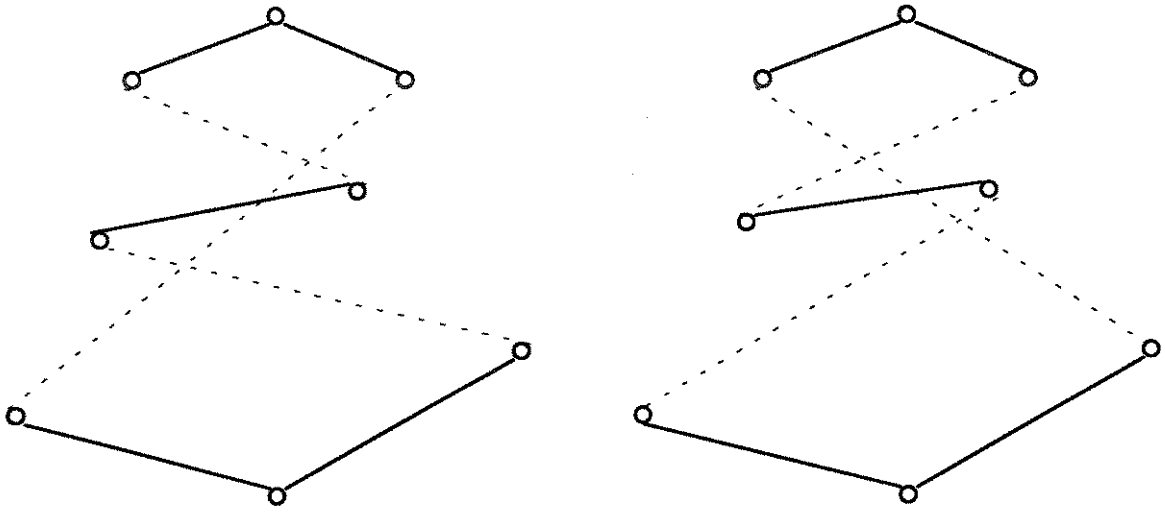


Figura 1.2 cont.- Oito Maneiras de Conectar Três Subrotas

Estas heurísticas de trocas são importantes, uma vez que ilustram um procedimento heurístico geral para problemas de otimização combinatória que podem ser representados por um grafo. Elas podem ser usadas em conjunto com outros procedimentos heurísticos ou mesmo numa seqüência delas próprias (por exemplo uma troca 2-OPT seguida de uma 3-OPT, BODIN et al. - 1983).

O procedimento *k*-OPT de LIN e KERNIGHAN (1973) foi inicialmente proposto para o caso simétrico e o valor de *k* é modificado dinâmicamente através de um algoritmo que produz soluções próximas da ótima. Dentre as heurísticas já propostas para o TSP, a de LIN e KERNIGHAN tornou-se a mais utilizada, principalmente devido às boas soluções que geralmente produz.

Mais recentemente JOHNSON (1990) desenvolveu um método "LIN-KERNIGHAN aleatóriamente integrado", que usa técnicas de *simulated annealing* (METROPOLIS et al. - 1953 e KIRKPATRICK et al. - 1983) e algoritmos genéticos (BRADY - 1985 e MÜHLENBEIN et al. - 1988), chegando a soluções ainda mais próximas da ótima.

Também em 1990, FIECHTER desenvolveu um algoritmo usando conceitos de computação paralela e técnicas de Busca Tabu (GLOVER - 1989 e 1990a) que é um procedimento heurístico geral para otimização global que vem sendo aplicado a vários tipos de problemas de otimização combinatória. O algoritmo proposto mostrou-se eficiente, obtendo soluções próximas da ótima. Ele usa conceitos de memória de médio e longo prazo da Busca Tabu, bem como utiliza um novo tipo de movimento baseado na combinação de, no mínimo, quatro movimentos 2-OPT. Foram realizados testes para

problemas de tamanho entre 500 e 100000 cidades. A implementação deste algoritmo paralelo foi feita em uma rede de transputers e seus resultados numéricos comprovaram sua eficiência.

Ainda no caso simétrico GENDREAU, HERTZ e LAPORTE (1992) desenvolveram o procedimento GENIUS que mistura um procedimento geral de inserção denominado GENI e um procedimento de pós-otimização denominado US (*unstringing e stringing*). O procedimento de inserção GENI foi inicialmente desenvolvido para o TSP simétrico . O enfoque de GENI é a integração de passos de otimização local embutidos num procedimento de inserção. Idéias semelhantes foram apresentadas por STEIGLITZ e WEINER (1968) cujo método 3-OPT dinâmico executa uma seqüência de movimentos consistindo da melhor inserção simples seguida de uma busca 3-OPT. GENI é menos míope e mais poderoso que os procedimentos padrão de inserção (por exemplo, 2-OPT). GENI permite que uma cidade seja incluída numa rota contendo pelo menos uma de suas vizinhas mais próximas, e cada inserção é executada simultaneamente com uma reotimização local da rota em questão.

O algoritmo GENIUS foi testado exaustivamente em problemas gerados aleatóriamente e também em problemas retirados da literatura (todos eles simétricos e Euclideanos). Os testes revelaram que GENIUS produz melhores soluções, em tempos computacionais menores, que a maioria dos algoritmos propostos para o TSP até o momento (LAPORTE - 1992). Deve-se ressaltar que GENIUS permite uma adaptação para o caso assimétrico, como poderemos ver no Capítulo 3 desta tese, onde GENIUS é adaptado para resolver o *P|tpds|Cmax* assimétrico.

A maioria dos trabalhos desenvolvidos para resolução do TSP simétrico está concentrada no desenvolvimento de heurísticas de bom desempenho médio. Entretanto, um ramo da pesquisa consiste no desenvolvimento de heurísticas com garantia de limitante de desempenho de pior caso. Dentre estas podemos citar o algoritmo desenvolvido por CHRISTOFIDES (1976) que tem um limitante de pior caso igual a 3/2. Também deve-se mencionar a não existência de uma heurística com limitante de desempenho de pior caso para o caso assimétrico.

Dentre os métodos exatos para o caso simétrico, geralmente baseados em técnicas de branch & bound e no tratamento de restrições violadas (restrições de eliminação de subrotas, desigualdades combinadas, desigualdades de árvores clique e desigualdades 2-matching), pode-se citar PADBERG e RINALDI (1987 e 1990) e GRÖTSCHEL e HOLLAND (1991). Eles desenvolveram procedimentos eficientes para identificação de

restrições violadas, incorporando estes procedimentos em um código de programação linear inteira. Os autores resolveram na otimalidade problemas contendo entre 17 e 2392 vértices (cidades). Acredita-se que o problema com 2392 vértices seja o maior problema não-aleatório simétrico resolvido na otimalidade até a presente data.

Para o caso assimétrico pode-se citar o método de CARPANETO e TOTH (1980) que é um método de branch & bound baseado na resolução de um problema de alocação obtido pela relaxação da restrição que impede a formação de subrotas na formulação do TSP (No modelo de programação linear inteira utilizado, x_{ij} é uma variável binária associada ao arco (i, j), igual a 1 se e somente se (i, j) é usado na solução ótima, $i \neq j$). O problema resolvido em um nó genérico da árvore de busca é um problema de alocação modificado (isto é, x_{ii} é fixo em 0 para todo i) no qual algumas variáveis x_{ij} são fixadas em 0 ou em 1. Se a solução do problema de alocação consiste de uma única rota sobre todos os vértices, então ela é factível para o TSP. Caso contrário, ela consiste de um número de subrotas. Uma destas subrotas é escolhida e particionada pela criação de subproblemas, nos quais cada um dos arcos da subrota é considerado proibido de aparecer na rota. Usando este algoritmo, os autores resolveram TSPs gerados aleatoriamente com até 240 vértices em menos de 1 minuto, num CDC 6600. A principal limitação dele parece ser espaço de memória, ao invés de tempo de CPU.

Em 1981, BALAS e CHRISTOFIDES usaram uma relaxação mais forte que a usada por CARPANETO e TOTH (1980). A descrição do processo é muito mais complexa que o caso anterior e o esforço computacional para o cálculo do limitante é muito maior. Porém, as árvores de busca resultantes são menores e o procedimento como um todo é mais poderoso. Os autores relatam a resolução de problemas aleatoriamente gerados com até 325 vértices, em menos de um minuto num CDC 7600.

Mais recentemente MILLER e PEKNY (1991) propuseram um algoritmo de branch & bound ainda baseado na relaxação de um problema de alocação, porém ainda mais poderoso que os anteriores. Os autores aplicaram este procedimento a problemas gerados aleatóriamente com até 5000 vértices, que foram resolvidos dentro de 40 segundos em um computador SUN 4/330. O maior problema resolvido por este procedimento tem 500000 vértices e necessitou de 12623 segundos de CPU em um supercomputador CRAY 2. Vale a pena mencionar que os mesmos autores (PEKNY e MILLER - 1992) desenvolveram um algoritmo paralelo de branch & bound, baseado na relaxação de um problema de alocação. Os autores mostraram a resolução exata de TSPs assimétricos gerados aleatoriamente com até 10000 vértices.

Em 1991, FISCHETTI e TOTH desenvolveram um procedimento chamado de *limitante aditivo* que usa a combinação de cinco limitantes diferentes. O sucesso deste algoritmo é dependente do tipo de problema considerado. Para o tipo mais fácil testado, os autores resolveram um problema aleatóriamente gerado com 2000 vértices, em um tempo médio de 8329 segundos num computador HP 9000/840.

1.4.2. Caso $m > 1$

No caso em que o número de processadores é maior que 1, pouco foi publicado. Para o caso simétrico, SUMICHRAST e BAKER (1987) e SUMICHRAST (1987) publicaram métodos baseados em programação linear inteira, fundamentados no trabalho de DEARING e HENDERSON (1984). Neste caso restrições de demanda também são consideradas. O método se mostrou mais apropriado para realizar o sequenciamento em locais onde existe um grande número de tarefas distintas e altos custos de preparação. Os tempos de preparação considerados são simétricos e satisfazem a desigualdade do triângulo. O método não resolve o problema de programação linear inteira associado como um todo, pois isto tornaria o tempo computacional proibitivo. Ao invés disto, ele considera somente os processadores onde produtos necessitam ser trocados, gerando uma série de problemas de programação linear 0/1. O método trabalha bem quando não há um número grande de produtos deficientes (isto é, que não alcançam a demanda no horizonte de tempo determinado), pois isto aumenta o número de processadores a serem considerados, aumentando o número de problemas de programação linear 0/1 a serem resolvidos.

Para o caso assimétrico, GUINET (1991) publicou vários algoritmos para problemas encontrados na indústria têxtil, onde um deles é semelhante ao *P|tpds|Cmax*. Porém, o objetivo é minimizar o somatório dos tempos de finalização de cada tarefa, ao invés do tempo máximo de processamento (*makespan*).

À primeira vista poder-se-ia concluir que os dois objetivos são equivalentes. O exemplo a seguir ilustra que isto nem sempre ocorre. Considere a figura 1.3 que representa a matriz de tempos de processamento de um exemplo contendo dois processadores e cinco tarefas. Nota-se a existência de uma tarefa zero que representa o ponto de partida, gerando uma primeira linha que representa os tempos de preparação iniciais de cada tarefa e a primeira coluna desta matriz é preenchida de zeros, pois não se tem nenhum gasto de tempo no momento em que a última tarefa termina. As figuras 1.4a e 1.4b mostram que um mesmo conjunto de tarefas pode fornecer diferentes soluções ótimas quando o objetivo é minimizar o somatório dos tempos de finalização de cada tarefa (caso de GUINET - 1991) e quando o objetivo é minimizar o tempo máximo de finalização (caso tratado nesta

tese). Pode-se observar que da figura 1.4a para a 1.4b tivemos uma diminuição do makespan e um aumento no somatório dos tempos de finalização, para um mesmo conjunto de tarefas, o que mostra a não equivalencia entre as funções objetivo dos dois modelos.

t_{ij}	0	1	2	3	4	5
0	-	30	30	30	30	100
1	0	-	50	90	95	91
2	0	94	-	99	101	200
3	0	93	98	-	30	26
4	0	56	121	147	-	30
5	0	97	100	131	140	-

Figura 1.3 - Matriz de Tempos de Processamento para o Exemplo das Figuras 1.4a e 1.4b

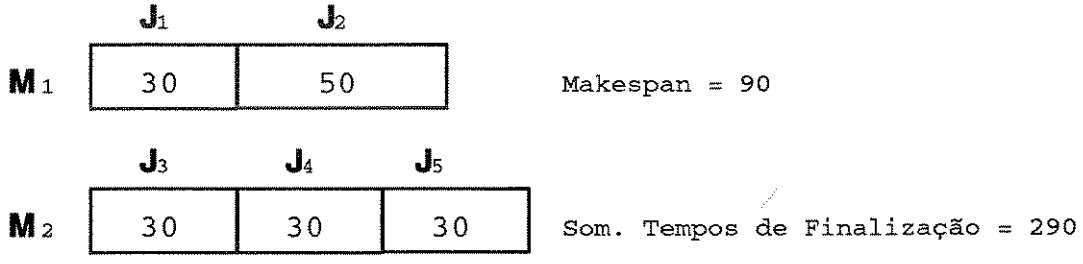


Figura 1.4a - Configuração Ótima Considerando como Objetivo Minimizar o Somatório dos Tempos de Finalização

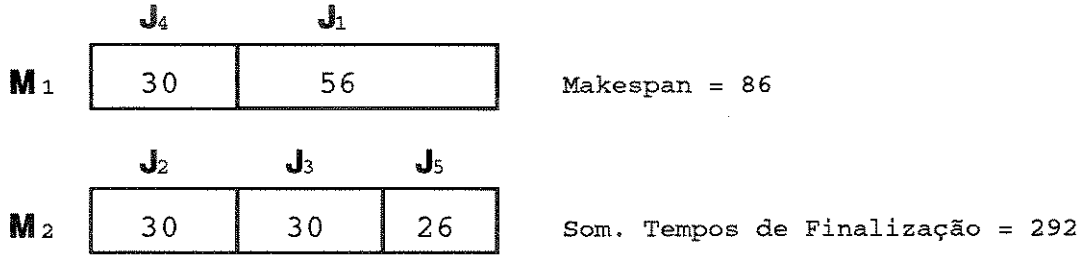


Figura 1.4b - Configuração Ótima Considerando como Objetivo Minimizar o Makespan

PARKER, DEANE e HOLMES (1977) também publicaram um algoritmo com o objetivo de minimizar os tempos de preparação, porém os tempos de processamento das tarefas não estão a eles incorporados, como no caso tratado nesta tese. Eles impõem uma restrição de carga que limita a capacidade de cada processador (esta restrição está baseada apenas nos tempos de processamento, não levando em conta os tempos de preparação).

Então eles propõem uma equivalência com o Problema de Roteamento de Veículos (PRV) que é resolvido por uma heurística baseada no algoritmo clássico de CLARKE e WRIGHT (1964). O PRV será descrito no decorrer deste capítulo.

Como o *P|tpds|Cmax* com $m = 1$ é equivalente ao TSP, poder-se-ia imaginar uma equivalência entre o *P|tpds|Cmax* com $m > 1$ e o Problema dos m Caixeiros Viajantes (m-TSP ou *Minsum* m-TSP), que é definido como sendo o problema de determinar m rotas disjuntas de modo que a distância total percorrida pelos m caixeiros viajantes seja minimizada. Isto nos permite dizer que o problema *Minsum* m-TSP é equivalente ao problema de sequenciamento que tenha como objetivo minimizar o tempo total de processamento, ou seja, minimizar a distância total de todas as rotas. No caso aqui tratado e mantendo a analogia descrita acima, o *P|tpds|Cmax* com $m > 1$ procuraria minimizar a rota mais longa (*makespan*).

De acordo com a terminologia de TSP, o que se deseja é encontrar m rotas em que todos os viajantes percorram distâncias aproximadamente iguais, isto é, a distância da rota mais longa deve ser minimizada. Este problema foi batizado com a denominação de ***Minmax m*-TSP**. A equivalência perfeita com o *P|tpds|Cmax* se dá porém, com o *Minmax* m-TSP *assimétrico* (até onde se sabe, inédito na literatura de roteamento), visto que deve-se impor que $t_{ij} \neq t_{ji}$.

O *Minsum* m-TSP é bem estudado na literatura e pode ser transformado em um 1-TSP pela adição de m depósitos fantasmas não conectados entre si e conectados aos arcos restantes do mesmo modo que o original. Posteriormente, encontra-se a solução do m-TSP quebrando a rota encontrada em m subrotas (LENSTRA e RINNOOY KAN - 1975). Porém, LAPORTE e NOBERT (1980) mostraram que é mais eficiente resolver o m-TSP diretamente, usando uma aproximação por planos de corte.

O *Minmax* m-TSP simétrico foi estudado por FREDERICKSON, HETCH e KIM (1978), que sugeriram um algoritmo heurístico com desempenho de pior caso igual a $5/2 - 1/m$.

Na pesquisa de um algoritmo exato para resolver o *P|tpds|Cmax* (*Minmax* m-TSP assimétrico) foi observada uma certa semelhança deste problema com um problema já resolvido por algoritmos exatos. Trata-se do Problema de Roteamento de Veículos (PRV) com Restrição de Tamanho da Rota - DVRP (*Distance-Constrained Vehicle Routing Problem*). O PRV pode ser visto como um m-TSP onde à cada rota de caixeiro viajante está associada uma determinada capacidade. O DVRP é semelhante ao *Minsum* m-TSP

exceto que nele, cada rota é restrita a não exceder uma determinada distância constante DIST. Assim, um esquema ótimo pode ser concebido através da resolução de uma série de DVRPs ajustando a restrição sobre as rotas (distância máxima de cada rota) a cada DVRP resolvido. Detalhando a idéia, assuma que C^* é a solução ótima do *P|tpds|Cmax* . Se a solução do DVRP para um dado valor de DIST (distância máxima permitida para a rota mais longa) é factível, então DIST é um limitante superior de C^*; se S é a soma de todos os arcos (ligações entre as tarefas, t_{ij}) usados na solução do DVRP, então $\lceil S/m \rceil$ é um limitante inferior de C^* , onde $\lceil\ \rceil$ significa o maior inteiro menor ou igual a. Se o DVRP é infactível , então DIST+1 é um limitante inferior de C^*. No Capítulo 3 desta tese é apresentado um esquema ótimo baseado nesta idéia.

Para a resolução do DVRP simétrico há o algoritmo de LAPORTE, DESROCHER e NOBERT (1984) e para o caso assimétrico o algoritmo de LAPORTE, NOBERT e TAILLEFER (1987). No caso simétrico, são resolvidos problemas Euclideanos ($t_{ij} \leq t_{ik} + t_{kj}$) e não-Euclideanos através de algoritmos de planos de corte e branch & bound.

O algoritmo proposto por LAPORTE, NOBERT e TAILLEFER (1987) está baseado nos trabalhos anteriores de LAPORTE, MERCURE e NOBERT (1986) que trata do PRV assimétrico com restrição de capacidade e de CARPANETO e TOTH (1980a) que trata do TSP assimétrico. O algoritmo está baseado na relaxação de subrotas e no problema de eliminação de rotas ilegais. O problema resultante é totalmente unimodular e é, na verdade, um problema de alocação modificado, para o qual existem algoritmos eficientes para sua solução (por exemplo, CARPANETO e TOTH - 1980b). Subrotas ilegais são eliminadas particionando-se o subproblema infactível, em um dado nó da árvore de busca, em um número de descendentes. Este algoritmo foi usado no Capítulo 3 deste trabalho no desenvolvimento do esquema ótimo proposto para a solução do *P|tpds|Cmax*.

CAPÍTULO 2

Algoritmos para Solução do Problema de Sequenciamento em Processadores Paralelos - *P | | Cmax*

2.1. INTRODUÇÃO

O problema é alocar *n* tarefas a *m* processadores idênticos, quando cada tarefa requer apenas uma operação que pode ser realizada em qualquer um dos processadores. A parte da Teoria de Sequenciamento em Máquinas (*machine scheduling*) que trata deste assunto é conhecida como Sequenciamento em Processadores Paralelos (*multiprocessor scheduling).*

De acordo com o esquema de classificação proposto por GRAHAM et al. (1979), existem três classes de problemas, dependendo se os processadores são *idênticos, uniformes* ou *não relacionados*. Quando os processadores são idênticos, o tempo de processamento de cada tarefa é igual em todos os processadores. Quando os processadores são uniformes, o tempo de processamento varia de acordo com um fator constante de velocidade associado a cada processador. Quando os processadores são não relacionados, o tempo de processamento de uma tarefa também varia de processador para processador sem, porém, qualquer fórmula estabelecida.

O trabalho aqui desenvolvido está concentrado no problema de alocar *n* tarefas com tempos de execução conhecidos, a *m* processadores paralelos e idênticos, de uma forma não-preemptiva (isto é, uma vez que uma determinada tarefa esteja alocada a um processador, ela deverá permanecer no mesmo até o seu término, sem interrupções), objetivando minimizar o tempo máximo de finalização de todas as tarefas (*makespan*). Usando a classificação de três campos proposta por GRAHAM et al. (1979), o problema descrito pode ser representado como *P| |Cmax.*

É proposto um algoritmo heurístico composto de três fases distintas. Também é apresentado um esquema ótimo para solução do problema que resolve uma sequência de

problemas de empacotamento (*Bin Packing Problem*) dentro de um procedimento de busca dicotômica.

Este capítulo está estruturado da seguinte forma: inicialmente é apresentada uma descrição formal do problema a ser resolvido; a seguir é descrito o algoritmo heurístico para a resolução do mesmo e, finalmente, a estratégia ótima de resolução. A estratégia ótima objetiva a análise de desempenho da heurística desenvolvida.

2.2. MODELO MATEMÁTICO DO PROBLEMA $P\,|\,|\,Cmax$

O problema a ser resolvido é alocar n tarefas independentes, $J_1, J_2, ..., J_n$, a m processadores paralelos idênticos, $M_1, M_2, ..., M_m$, de uma forma não preemptiva. Sabe-se também que $n \geq m \geq 2$ e que cada tarefa J_j tem um tempo de execução inteiro e positivo p_j . Então o problema é minimizar o tempo de finalização total das tarefas (*makespan*), definido como: $C_{max} = \max_{i=1,\ldots,m} \{C_i\}$, onde C_i é o tempo de execução de todas as tarefas alocadas ao processador M_i .

Uma possível formulação matemática do $P\,|\,|Cmax$ é:

$$x_{ij} = \begin{cases} 1 & \text{se a tarefa } j \text{ é alocada ao processador } i \\ 0 & \text{caso contrário} \end{cases}$$

$$Min \ \sigma$$

$$tal\ que \ \sum_j p_j x_{ij} \leq \sigma, \forall i$$

$$\sum_i x_{ij} = 1, \forall j$$

$$x_{ij} \in \{0,1\}, \forall i, j$$

Como já observado no capítulo 1, o problema $P\,|\,|Cmax$ tem sido muito pesquisado. Está provado que ele é um problema NP-difícil num sentido forte quando o número de processadores é arbitrário. Porém, o problema $P\,|\,|Cmax$ passa a ser solúvel em um tempo

pseudo-polinomial quando o número de processadores é fixo, sendo assim NP-difícil somente no sentido ordinário (GAREY e JOHNSON - 1978). Seguindo a conjectura de ser pouco provável o desenvolvimento de um algoritmo que resolva o *P| |Cmax* em um tempo polinomial, desenvolveu-se um algoritmo heurístico de três fases para a resolução deste problema.

Sabe-se que em problemas onde espera-se que o tempo computacional cresça exponencialmente com o tamanho do problema, os algoritmos heurísticos se apresentam como uma boa alternativa, encontrando soluções próximas da ótima em tempos computacionais razoáveis, o que permite a resolução de problemas complexos e/ou de grande porte. Na próxima seção será descrito um algoritmo heurístico desenvolvido para a resolução do *P| |Cmax* cuja principal caraterística é possuir um esquema de busca local sofisticado que não compromete o desempenho computacional.

2.3. ALGORITMO HEURÍSTICO *3-FASES*

O algoritmo 3-FASES pode ser resumido do seguinte modo. Na FASE 1, as tarefas são classificadas de acordo com os tempos de processamento e alocadas aos processadores de maneira a obter um carregamento razoavelmente balanceado entre eles. Na FASE 2, a divisão de carga é melhorada, movendo-se, sucessivamente, uma tarefa do processador mais carregado para o menos carregado usando o valor de um limitante pré-calculado como alvo. Por fim, na FASE 3, tenta-se um balanço de cargas ainda melhor, através da troca simultânea de duas tarefas pertencentes a processadores diferentes. O detalhamento do algorítmo 3-FASES será feito a seguir.

FASE 1 - Alocação Inicial

Esta fase pode ser vista como um procedimento construtivo, com a característica de não necessitar uma pré-ordenação das tarefas, geralmente requerida na maioria dos algoritmos construtivos conhecidos. A única exigência é que se conheça um limitante

inferior ($\underline{p}$) e superior ($\overline{p}$) dos tempos de processamento das tarefas a serem alocadas. Por exemplo, eles podem ser obtidos como $\underline{p} = \min_j \{p_j\}$ e $\overline{p} = \max_j \{p_j\}$.

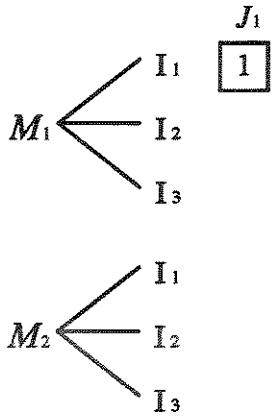
De posse destes dois limitantes, a idéia básica subjacente ao algoritmo heurístico é dividir o intervalo $[\underline{p}, \overline{p}]$ em r intervalos aproximadamente iguais e associar cada tarefa ao intervalo que corresponde ao seu tempo de processamento.

Chamando estes r intervalos de I_1 ,..., I_r , onde r é um parâmetro de entrada do algoritmo, chama-se uma tarefa J_j de l-tarefa se $p_j \in I_l$. A alocação inicial proposta funciona do seguinte modo: primeiro inicializa-se $s_1 = ... = s_r = 0$, o índice do processador no qual a última l-tarefa foi alocada ($l = 1,...,r$). Então, para cada uma das n tarefas J_j : se J_j é uma l-tarefa aloque-a ao processador $s_l \pmod m + 1$ e faça $s_l = s_l \pmod m + 1$, onde $s_l \pmod m = 0$ quando s_l é múltiplo de m e igual ao resto da divisão (s_l / m), caso contrário. O que se objetiva é alocar um mesmo número de tarefas a cada intervalo de cada processador e, com isso, assegurar uma alocação equilibrada entre os processadores.

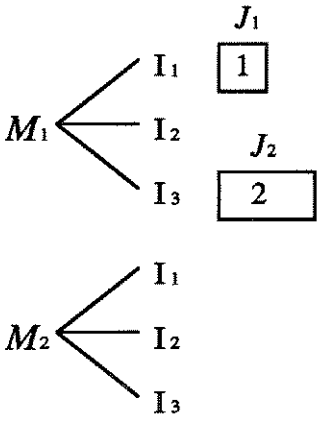
Exemplificando o funcionamento da alocação inicial, considere a divisão de $[\underline{p}, \overline{p}]$ em r intervalos e $m = 2$, $r = 3$.

Supondo o conjunto de tarefas com tempos de processamento $p_j = \{ 1 , 2 , 1 , 2 \}$, tem-se $\underline{p} = 1$, $\overline{p} = 2$ e $I_1 = [1,4/3]$, $I_2 = (4/3,5/3]$ e $I_3 = (5/3,2]$. Faz-se $s_1 = s_2 = s_3 = 0$.

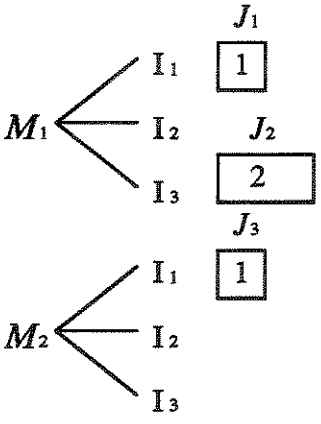
A tarefa J_1 é uma l-tarefa, onde $l = 1$. Como $s_1 = 0$, então $s_1 \pmod 2 + 1 = 1$ e aloca-se a tarefa J_1 ao processador 1 e faz-se $s_1 = 1$.



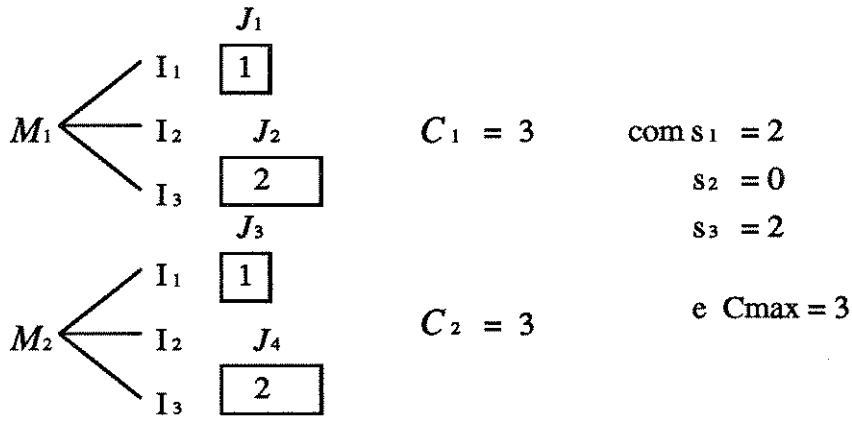
Passando para a tarefa J_2, tem-se que ela é uma l-tarefa, onde $l = 3$. Então, a tarefa J_2 é alocada ao processador 1, visto que $s_3 \pmod 2 + 1 = 1$ e faz-se $s_3 = 1$.



Agora a tarefa considerada é J_3 , que como J_1 , é uma l-tarefa, onde $l = 1$. Então a tarefa J_3 é alocada ao processador 2, visto que $s_1 \pmod 2 + 1 = 2$ e faz-se $s_1 = 2$.



Como a tarefa J_4 é uma é uma *l*-tarefa, onde $l = 3$, aloca-se J_4 ao processador 2, uma vez que $s_3 \pmod 2 + 1 = 2$ e faz-se $s_3 = 2$. E assim chega-se a seguinte configuração ao final da alocação inicial.



Como pode-se notar neste exemplo, a alocação inicial atingiu a solução ótima do problema, mas o que realmente se busca com este procedimento é, sem a necessidade de qualquer ordenação das tarefas, alocar o mesmo número de tarefas em cada intervalo I_i, nos *m* processadores. Pode-se notar que quanto maior a razão (*n/mr*) - número médio de tarefas por intervalo, por processador - maiores as chances de atingir estes objetivos. Note também que cada processador contém praticamente todo o espectro de tipos de tarefas. Estas características se mostrarão bastante úteis nas duas fases seguintes do algoritmo, principalmente quando se procura eficiência na busca das melhores trocas de tarefas entre processadores.

FASE 2 - Fase de Balanceamento

Esta fase pode ser vista como um procedimento de melhoramento, pois, a partir da solução encontrada pela FASE 1, ela procura diminuir o tempo de finalização, trocando tarefas do processador mais carregado para o menos carregado, até que não existam mais trocas que atinjam este objetivo.

A escolha das tarefas a serem trocadas é orientada pelo *tempo médio de finalização*, definido como $\overline{C} = \sum_{i=1}^{m} \frac{C_i}{m}$, que independe de como as tarefas foram alocadas aos processadores. O valor de $\overline{C}$, também conhecido como limitante de McNaughton (McNAUGHTON - 1959), é um limitante inferior para a solução ótima do problema. Ele

corresponde à solução ótima quando a hipótese de tarefas não-preemptivas é relaxada. Este valor de $\overline{C}$ servirá como um alvo para decidir em qual intervalo será feita a busca da tarefa a ser trocada e qual será esta tarefa. Deve-se notar que o uso do alvo não restringe a busca , pois quando não existem tarefas que satisfazem o alvo, a FASE 2 permite uma ampliação no espaço de busca, liberando trocas que ultrapassem o alvo, desde que a solução melhore.

A figura abaixo ilustra o funcionamento da FASE 2 do algorítmo.

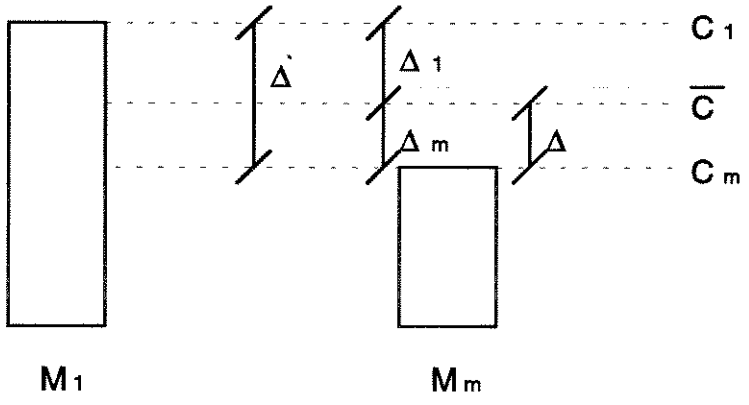


Figura 2.1 - Representação Gráfica dos Principais Elementos da FASE 2 do Algoritmo.

O objetivo da FASE 2 é, através de trocas simples de tarefas, procurar levar o tempo de finalização dos processadores a um valor o mais próximo possível de $\overline{C}$ (valor ótimo preemptivo). Identificados M_1 e M_m e calculados $\Delta_1 = C_1 - \overline{C}$, $\Delta_m = \overline{C} - C_m$, $\Delta = \min\{\Delta_1, \Delta_m\}$ e $\Delta' = C_1 - C_m$, passa-se a procurar uma tarefa em M_1 que, ao ser movida para M_m , leve o valor de C_1 e C_m para valores próximos de $\overline{C}$. É evidente que a melhor delas será aquela com tempo de processamento mais próximo a Δ. Primeiramente encontra-se o intervalo I_l ao qual Δ pertence e busca-se nele uma tarefa com tempo de processamento menor que Δ. Se esta tarefa não for encontrada, permite-se ainda que tarefas com tempos de processamento menores que Δ´ sejam selecionadas.

No caso de não existirem tarefas em I_l que possam ser trocadas, tenta-se encontrar uma nos intervalos l-1 a 1, pois sabe-se que todas as tarefas pertencentes a estes intervalos satisfazem a condição $p_j < \Delta'$. Se, ainda assim, não for encontrada uma tarefa que possa ser trocada, tenta-se uma busca nos intervalos l+1 a r por uma tarefa tal que $p_j < \Delta'$ (note

que esta busca se estende até que se encontra o primeiro intervalo cujo limitante inicial seja maior que Δ', pois, a partir deste ponto, a busca se torna sem sentido). Quando não existirem mais tarefas a serem trocadas, o algoritmo passa para a FASE 3.

É importante salientar que a estrutura de intervalos não é alterada, uma vez que quando identificada a tarefa a ser trocada, ela muda de processador, mas permanece em seu intervalo original. Esta estruturação das tarefas em intervalos de acordo com seus tempos de processamento facilita bastante a busca das tarefas que são as melhores candidatas à realocação.

Os passos desta fase são descritos a seguir:

Passo 1 - Identifique os processadores mais e menos carregados e chame-os de M_1 e M_m, repectivamente.

Passo 2 - Calcule $\Delta_1 = C_1 - \overline{C}$, $\Delta_m = \overline{C} - C_m$, $\Delta = \min\{\Delta_1, \Delta_m\}$ e $\Delta' = C_1 - C_m$.
Se $\Delta < \underline{p}$, faça $l = 0$ e vá para o PASSO 5. Caso contrário, Δ pertence a algum intervalo I_l , ou $\Delta > \overline{p}$, caso no qual se faz $l = r$.

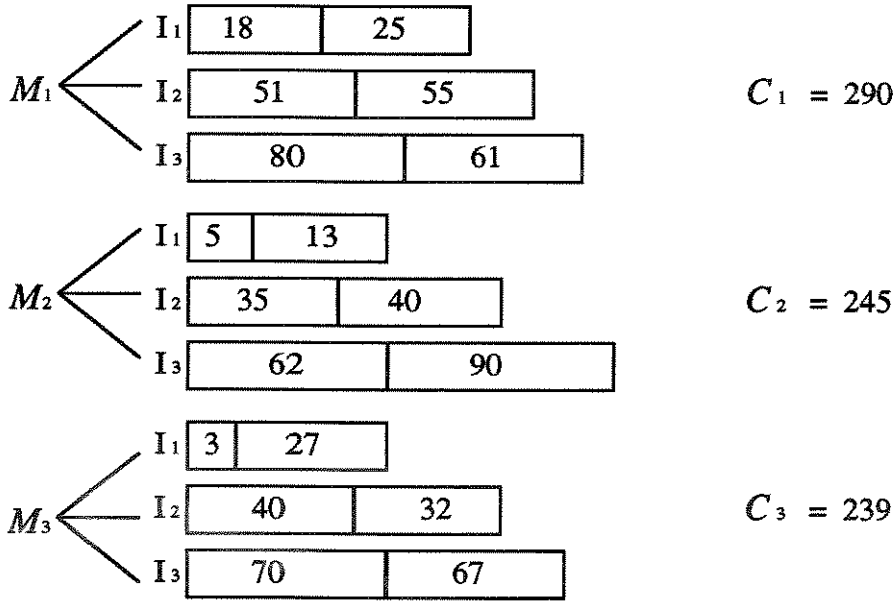
Passo 3 - Se não existe em M_1 uma tarefa J_j com $p_j \in \mathrm{I}_l$ e $p_j < \Delta'$, vá para o PASSO 4. Caso contrário, realoque uma tarefa pertencente ao intervalo I_l , de M_1 para M_m e vá para o PASSO 1.

Passo 4 - Se não existe em M_1 uma tarefa J_j com $p_j \in \mathrm{I}_{\bar{1}} = \mathrm{I}_1 \cup \ldots \cup \mathrm{I}_{l-1}$, vá para o PASSO 5. Caso contrário, realoque uma tarefa pertencente ao intervalo $\mathrm{I}_{\bar{1}}$, de M_1 para M_m e vá para o PASSO 1.

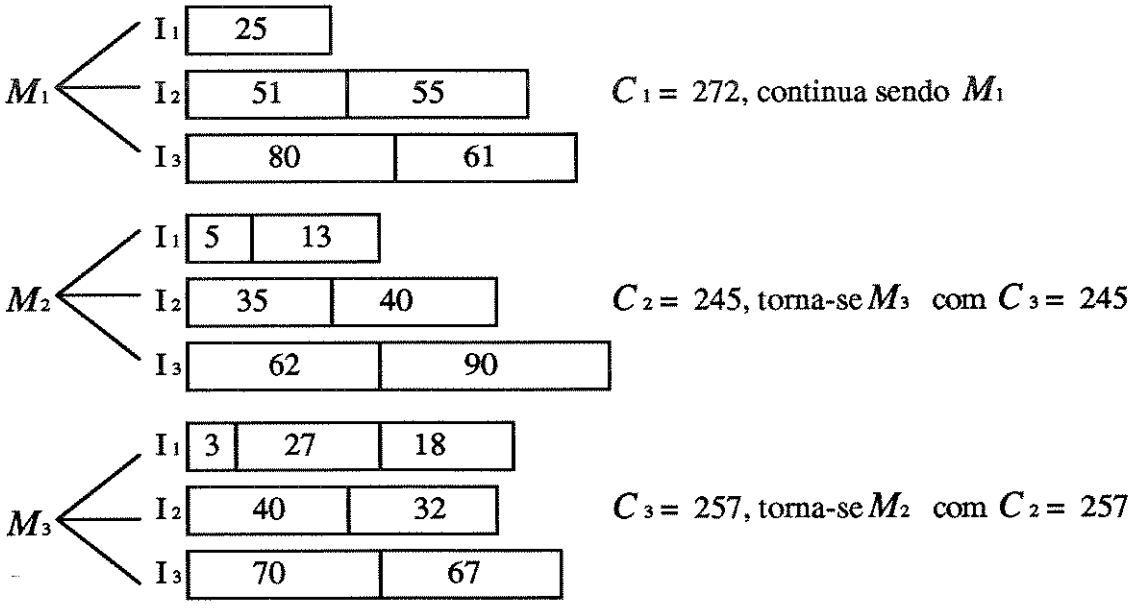
Passo 5 - Se $l = r$, vá para a FASE 3. Caso contrário procure em M_1 uma tarefa J_j em $\mathrm{I}_{l+1}, \mathrm{I}_{l+2}, \ldots, \mathrm{I}_r$, sucessivamente, tal que $p_j < \Delta'$ (Note que esta busca deve parar no primeiro intervalo no qual todas as tarefas tenham tempos de processamento maiores ou iguais a Δ'). Se não encontrar esta tarefa, vá para a FASE 3. Caso contrário, realoque a tarefa J_j de M_1 para M_m e vá para o PASSO 1.

Para um melhor entendimento dos passos do algoritmo, apresentaremos um exemplo de como a FASE 2 do algoritmo 3-FASES trabalha. Supondo que após a FASE 1 tem-se a seguinte configuração:

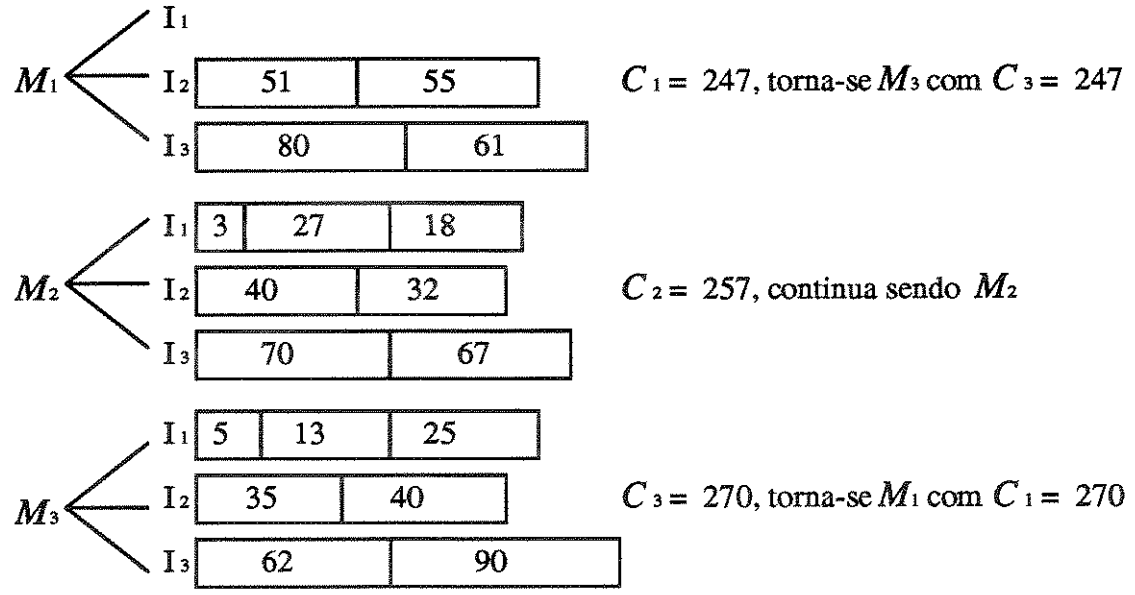
$r = 3, m = 3, n = 18, \underline{p} = 1, \overline{p} = 90$ e $I_1 = [1,30]$, $I_2 = [31,60]$ e $I_3 = [61,90]$.



Após o PASSO 1, os processadores já estão na classificação correta, com M_1 sendo o mais carregado e M_3 o menos. Sabe-se que $\overline{C} = 258$ e, aplicando-se o PASSO 2, tem-se : $\Delta_1 = C_1 - \overline{C} = 32$, $\Delta_m = \overline{C} - C_m = 19$, $\Delta = \min\{\Delta_1, \Delta_m\} = 19$, $\Delta' = C_1 - C_m = 51$ e $\Delta \in I_1$ Aplicando-se o PASSO 3, procura-se em M_1 $p_j \in I_1$ tal que $p_j < \Delta'$. Então escolhe-se a tarefa com $p_j = 18$ e realiza-se a troca.

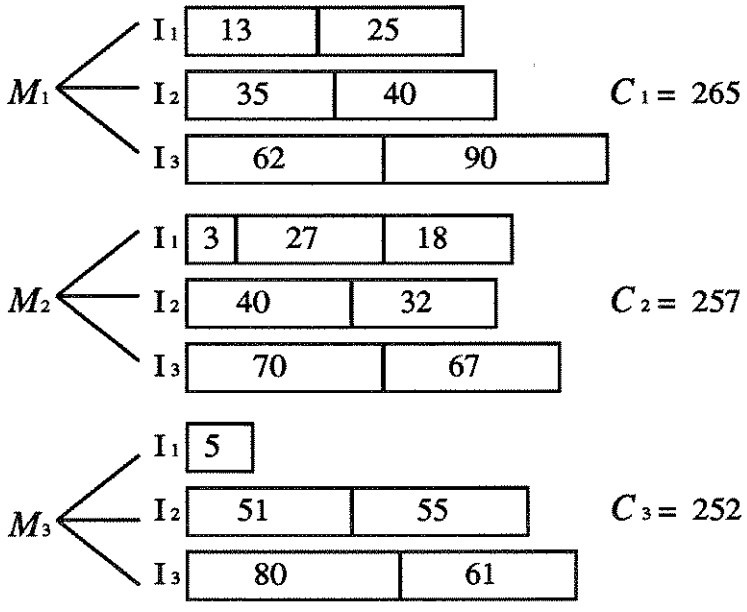


Aplica-se o PASSO 1 para identificar o processador mais e menos carregado e atualizam-se os índices como mostrado acima. Aplica-se o PASSO 2 e obtem-se : $\Delta_1 = C_1 - \overline{C} = 14$, $\Delta_m = \overline{C} - C_m = 13$, $\Delta = \min\{\Delta_1, \Delta_m\} = 13$ e $\Delta' = C_1 - C_m = 27$. $\Delta \in I_1$ e aplica-se o PASSO 3. Procura-se em M_1 a tarefa $p_j \in I_1$ tal que $p_j < \Delta'$. Então escolhe-se a tarefa com $p_j = 25$ e realiza-se a troca.



Novamente aplica-se o PASSO 1 e atualizam-se os índices dos processador mais e menos carregado. Aplica-se o PASSO 2 e obtem-se : $\Delta_1 = C_1 - \overline{C} = 12$, $\Delta_m = \overline{C} - C_m = 11$, $\Delta = \min\{\Delta_1, \Delta_m\} = 11$ e $\Delta' = C_1 - C_m = 23$. $\Delta \in I_1$ e aplica-se o PASSO 3. Procura-se

em M_1 a tarefa $p_j \in \mathrm{I}_1$ tal que $p_j < \Delta'$. Então escolhe-se a tarefa com $p_j = 5$ e realiza-se a troca, resultando a seguinte configuração.



A aplicação do PASSO 1 não traz nenhuma alteração nos índices dos processadores mais e menos carregados, então aplica-se o PASSO 2 e obtem-se : $\Delta_1 = C_1 - \overline{C} = 7$, $\Delta_m = \overline{C} - C_m = 6$, $\Delta = \min\{\Delta_1, \Delta_m\} = 6$, $\Delta' = C_1 - C_m = 13$ e $\Delta \in \mathrm{I}_1$. No PASSO 3 vê-se que não existem tarefas em M_1 tal que $p_j \in \mathrm{I}_1$ e $p_j < \Delta'$. Logo, vai-se ao PASSO 4. No PASSO 4, nota-se que $\mathrm{I}_i = \{\ \}$ e não há nenhuma tarefa em M_1 para ser realocada. Do PASSO 5, vê-se que o início do intervalo $\mathrm{I}_{i+1} = 31 > \Delta'$, portanto, também não existem tarefas a serem trocadas e a FASE 2 termina.

A FASE 2 realiza a mudança de uma tarefa de um processador para outro, de acordo com certos critérios que foram apresentados. Já na FASE 3, que será apresentada a seguir, será feita uma busca por duplas trocas, envolvendo tarefas de diferentes processadores.

FASE 3 - Duplas Trocas

De modo a incorporar novas soluções ao espaço de busca, desenvolveu-se outra fase, que também pode ser vista como um algoritmo de melhoramento. Esta fase busca realizar duplas trocas, envolvendo uma tarefa do processador mais carregado e uma de outro processador. Esta fase é um pouco mais elaborada, pois além de envolver duas tarefas,

procura a melhor troca dentre todas as possíveis, aproveitando a estrutura de armazenamento das tarefas em intervalos para reduzir as combinações à pesquisar.

Novamente M_1 e M_m representam os processadores mais e menos carregados, repectivamente. Os processadores restantes são arbitrariamente nomeados. Nesta fase tenta-se a troca de uma tarefa de M_1 (digamos, J_j) com uma tarefa de outro processador M_h (J'_j), partindo de $h = m, m-1, \ldots, 2$, até que uma troca vantajosa seja identificada, isto é, uma troca tal que $0 < \theta = p_j - p_j' < d = C_1 - C_h$.

A figura abaixo ajuda a compreensão da FASE 3.

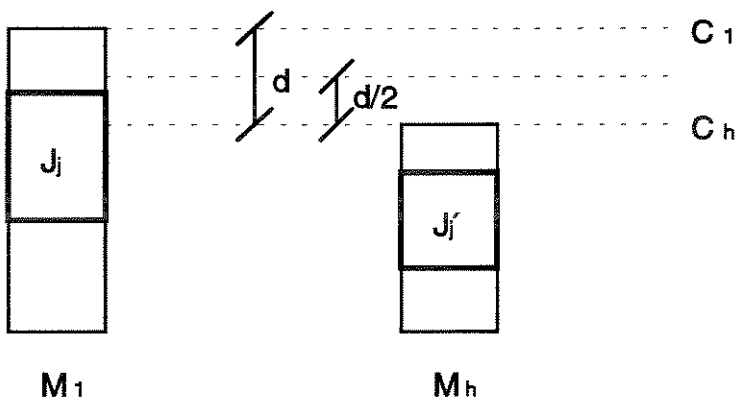


Figura 2.2 - Representação Gráfica dos Principais Elementos da FASE 3 do Algoritmo.

A idéia do uso de um alvo para direcionar a busca da melhor troca, como por exemplo o uso de $\overline{C}$ na FASE 2, é novamente utilizada aqui. Usa-se *d*/2 como valor alvo para a diferença dos tempos de processamento entre as duas tarefas a serem trocadas. Para se chegar a este valor, observe que após a troca, o valor de C_1 diminuirá para $C_1' = C_1 - \theta$, e que C_h aumentará para $C_h' = C_h + \theta$. Portanto, $\left|C_1' - C_h'\right| = \left|C_1 - C_h - 2\theta\right| = \left|d - 2\theta\right|$ e este valor é minimizado para $\theta = d/2$.

Usando este alvo e a estrutura de intervalos, vê-se que, para achar a troca que mais se aproxima de θ, não é necessário enumerar todas as trocas possíveis, pois a busca fica restrita a pares de tarefas pertencentes a dois intervalos localizados a aproximadamente *d*/2 unidades de distância.

Ao contrário da FASE 2, onde se realiza o primeiro movimento que satisfaça as condições dadas, a FASE 3 busca pelo *melhor* movimento dentre os analisados. Para isto, primeiro se identifica o intervalo que está distante *d*/2 unidades do limitante superior do intervalo *r* (intervalo de ordem mais alta). A partir destes dois intervalos e, considerando também os outros pares de intervalos que estejam à esta mesma distância, busca-se a melhor troca possível, envolvendo as tarefas p_j e p'_j tal que $0 < p_j - p'_j < d$, entre os processadores mais e menos carregados. Se não existir nenhum par de tarefas a ser trocado com o processador menos carregado, busca-se em outro processador, até que todos os processadores sejam analisados ou a melhor troca seja encontrada.

Os passos desta fase são apresentados a seguir.

Passo 1 - Faça $h = m$.

Passo 2 - Faça $t = r$; $f_t = C_1$ e $d = C_1 - C_h$.

Se $\left(\overline{p} - d/2\right) \leq \underline{p}$, faça $s = 1$. Caso contrário, faça s tomar o valor do intervalo ao qual $\left(\overline{p} - d/2\right)$ pertence.

Passo 3 - Se existem as tarefas J_j e J'_j , com $p_j \in \mathrm{I}_t$, $p_j' \in \mathrm{I}_s$, tal que J_j está alocado a M_1 J'_j está alocado a M_h e $0 < p_j - p_j' < d$, calcule:

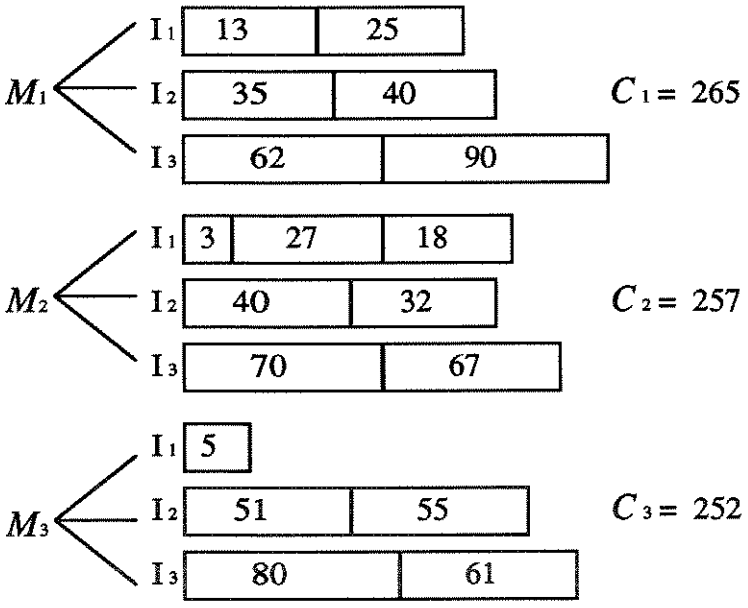
$$f_t = \min_{J_j, J_j'} \left\{ \max \left[C_1 - p_j + p_j' \, , C_h + p_j - p_j' \right] \right\}$$

Caso contrário faça $f_t = C_1$.
Se $s = 1$, faça $\underline{t} = t$ e vá para o PASSO 5.

Passo 4 - Faça $s = s - 1$; $t = t - 1$ e vá para o PASSO 3.

Passo 5 - Faça $f^* = \min_{\underline{t} \leq t \leq r} \{f_t\}$. Se $f^* < C_1$, troque as duas tarefas J_j e J'_j que produziram f^* e vá para a FASE 2. Se $f^* = C_1$, faça $h = h - 1$. Se $h > 1$ vá para o PASSO 2. Se $h = 1$, FIM.

Novamente apelamos para um exemplo para ajudar o entendimento da FASE 3 do algoritmo 3-FASES. Tomando como ponto de partida a configuração final encontrada pela aplicação da FASE 2 ao exemplo anterior, tem-se:



Do PASSO 1 faz-se $h = 3$.

Aplica-se o PASSO 2 e obtem-se $t = 3$, $f_3 = C_1 = 265$, $d = C_1 - C_3 = 13$, $\left(\overline{p} - d/2\right)$=90 - 6,5 = 83,5. Como $\underline{p} = 3 < 83{,}5$ faz-se $s = 3$, pois $\left(\overline{p} - d/2\right) \in I_3$. Note que , se $\underline{p} > 83{,}5$ a distância alvo estaria abaixo de $\underline{p}$, portanto, tenta-se a troca entre I_r e I_1 , que é a maior distância entre intervalos, o que leva a trocas de alto valor, como necessário neste caso.

No PASSO 3 procura-se por $J_j \in M_1$ e $J_j' \in M_3$ com $p_j \in I_3$ e $p_j' \in I_3$ e $0 < p_j - p_j' < d$.

Possibilidade 1) $p_j - p_j' = 62 - 61 = 1 \Rightarrow C_1 - p_j + p_j' = 264$ e $C_3 + p_j - p_j' = 253$;

Possibilidade 2) $p_j - p_j' = 90 - 80 = 10 \Rightarrow C_1 - p_j + p_j' = 255$ e $C_3 + p_j - p_j' = 262$;

Possibilidade 3) $p_j - p_j' = 90 - 61 = 29 > d \Rightarrow$ desconsiderada.

Então f_3 assume o valor de $f_3 = 262$.

O PASSO 4 atualiza $s = s - 1 = 2$ e $t = t - 1 = 2$; e volta-se ao PASSO 3 procurando por $J_j \in M_1$ e $J_j' \in M_3$ com $p_j \in \mathrm{I}_2$ e $p_j' \in \mathrm{I}_2$ e $0 < p_j - p_j' < d$. Vê-se que não existe $p_j - p_j' > 0$ neste caso, então f_2 assume o valor de C_1, $f_2 = 265$.

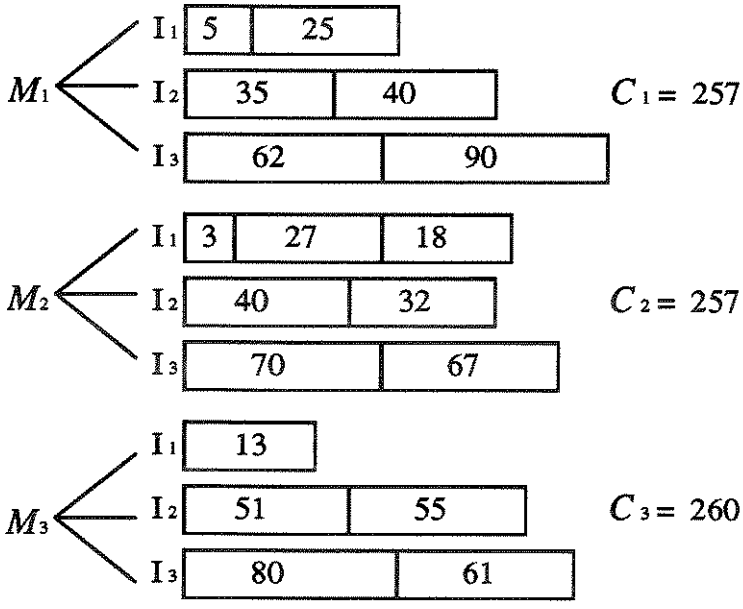
Pelo PASSO 4 atualizam-se novamente $s = s - 1 = 1$ e $t = t - 1 = 1$; e agora o PASSO 3 procura por $J_j \in M_1$ e $J_j' \in M_3$ com $p_j \in \mathrm{I}_1$ e $p_j' \in \mathrm{I}_1$ e $0 < p_j - p_j' < d$.

Possibilidade 1) $p_j - p_j' = 13 - 5 = 8 \Rightarrow C_1 - p_j + p_j' = 257$ e $C_3 + p_j - p_j' = 260$;

Possibilidade 2) $p_j - p_j' = 25 - 5 = 20 > d \Rightarrow$ desconsiderada.

Então f_1 assume o valor de $f_1 = 260$.

Como $s = 1$, faz-se $t = 1$ e aplica-se o PASSO 5. No PASSO 5, vê-se que $f^* = f_1 = 260 < C_1$ e J_j é a tarefa com $p_j = 13$ e J_j' é a tarefa com $p_j' = 5$, portanto, realiza-se a troca obtendo a configuração abaixo e retorna-se à FASE 2 do algoritmo 3-FASES.



A solução encontrada pela FASE 3 é ótima para este problema. Note que a dupla troca, em conjunto com a estrutura de intervalos, realmente proporciona um ajuste fino na qualidade da solução.

Vale observar que o maior esforço de cálculo se concentra na FASE 3, pois a busca envolve pares de tarefas. Porém, a distribuição das tarefas em r intervalos proporciona uma

redução considerável nos cáculos de f_t no PASSO 3 e portanto, influi no desempenho total da FASE 3.

Assumindo-se tempos de processamento uniformemente distribuidos, pode-se notar que o número médio de tarefas por máquina, com tempo de processamento em um dado intervalo, é aproximadamente igual a n/rm. Isto implica que, no PASSO 3, em torno de $(n/rm)^2$ pares de tarefas são considerados no cálculo de f_t para cada valor de t. Portanto, o cálculo de f^* requer, aproximadamente, $2r(n/rm)^2$ comparações, ao invés de $2(n/m)^2$ quando nenhuma partição em intervalos é utilizada.

A seguir será feita uma análise sobre a garantia de parada, após um número finito de passos, do algoritmo 3-FASES, uma vez que não há nenhum limitante no número máximo de iterações permitidas ou no número máximo de trocas permitidas.

2.4. PROVA DE PARADA

Para provar que o algoritmo termina, usa-se uma função quadrática $\tilde{f}$, que é definida com base em $\overline{C}$ - teoricamente o menor tempo de finalização total (*makespan*) que poderia ser encontrado para qualquer conjunto de n tarefas alocadas a m processadores, admitindo preempção - e no tempo de finalização de cada processador.

Para que o algoritmo termine, deve-se provar que a solução encontrada a cada passo do algoritmo vai-se tornando cada vez mais próxima de $\overline{C}$. Se isto é provado e não existe mais nenhuma troca disponível, a solução corrente é a melhor encontrada e o algoritmo acaba após um número finito de passos.

Definindo $\tilde{f}$ como sendo a soma dos quadrados das diferenças entre o tempo de finalização de cada processador e $\overline{C}$, tem-se:

$$\overline{C} = \sum_{i=1}^{m} C_i / m \quad \text{e} \quad \tilde{f} = \sum_{i=1}^{m} (C_i - \overline{C})^2$$

Observa-se que, se $\tilde{f}$ assume o valor zero, o algoritmo termina e a solução encontrada é ótima. Caso contrário o algoritmo tenta uma troca simples ou dupla entre tarefas e,

portanto, deve-se analisar a diferença entre o novo valor de $\tilde{f}$, chamado $\tilde{h}$, e o antigo valor de $\tilde{f}$, chamado $\tilde{g}$.

Então, se a diferença $\tilde{g}$ - $\tilde{h}$ for positiva, pode-se afirmar que a cada troca realizada produz-se uma redução no valor de $\tilde{f}$. Sendo $\tilde{f}$ uma função com valor inicial não negativo e limitado inferiormente em zero, se $\tilde{f}$ decresce a cada troca, pode-se concluir que o algoritmo termina em um número finito de passos. Será mostrado a seguir que nas FASES 2 e 3 do algoritmo 3-FASES o valor de $\tilde{f}$ é reduzido.

Inicialmente vai-se analisar a FASE 2, onde é definido: $\Delta_1 = C_1 - \overline{C}$, $\Delta_m = \overline{C} - C_m$, $\Delta = \min\{\Delta_1, \Delta_m\}$ e $\Delta' = C_1 - C_m$. A tarefa J_j é escolhida de modo que $p_j < \Delta'$. Montando a expressão $\tilde{g}$ - $\tilde{h}$, tem-se:

$$\tilde{g} - \tilde{h} = (C_1 - \overline{C})^2 + (C_m - \overline{C})^2 - (C_1 - p_j - \overline{C})^2 - (C_m + p_j - \overline{C})^2$$

Caso $C_1 = \overline{C}$, segue-se que $C_2 = ... = C_m = \overline{C}$ e nenhuma tarefa J_j tal que $p_j < \Delta'$ existe. Portanto, encontrou-se a solução ótima do problema e o valor de $\tilde{g}$ - $\tilde{h}$ é zero. Caso contrário, desenvolvendo a expressão $\tilde{g}$ - $\tilde{h}$ resulta:

$$\begin{aligned}
\tilde{g} - \tilde{h} &= (C_1 - \overline{C} + C_1 - p_j - \overline{C})(C_1 - \overline{C} - C_1 + p_j + \overline{C}) + \\
&+ (C_m - \overline{C} + C_m + p_j - \overline{C})(C_m - \overline{C} - C_m - p_j + \overline{C}) = \\
&= (2C_1 - 2\overline{C} - p_j)(p_j) + (2C_m - 2\overline{C} + p_j)(-p_j) = \\
&= (p_j)(2C_1 - 2\overline{C} - p_j - 2C_m + 2\overline{C} - p_j) = \\
&= (p_j)(2(C_1 - C_m - p_j))
\end{aligned}$$

Sabe-se que, por hipótese, $p_j > 0$ e $\Delta' = C_1 - C_m > p_j$, então $C_1 - C_m - p_j > 0$ e, portanto, $\tilde{g}$ - $\tilde{h} > 0$, o que mostra que a FASE 2 faz com que a função $\tilde{f}$ decresça.

Na FASE 3 do algoritmo tem-se: $d = C_1 - C_h, h = m, m-1, \ldots, 2$ e a escolha das tarefas envolvidas na dupla troca é feita de modo que $J_j \in M_1$, $J_j' \in M_h$ e $p_j > p_j'$ com $0 < p_j - p_j' < d$.

Novamente, desenvolve-se a expressão $\tilde{g}$ - $\tilde{h}$, observando porém que agora o incremento (ou decremento) do valor de C_m (ou C_1) é $p_j - p_j'$. As considerações quanto a $C_1 = \overline{C}$ são as mesmas do caso anterior (FASE 2), portanto passa-se à análise de $\tilde{g}$ - $\tilde{h}$.

$$\tilde{g} - \tilde{h} = (C_1 - \overline{C})^2 + (C_h - \overline{C})^2 - \left(C_1 - \left(p_j - p_j'\right) - \overline{C}\right)^2 - \left(C_h + \left(p_j - p_j'\right) - \overline{C}\right)^2 =$$

$$= (C_1 - \overline{C})^2 + (C_h - \overline{C})^2 - \left(C_1 - p_j + p_j' - \overline{C}\right)^2 - \left(C_h + p_j - p_j' - \overline{C}\right)^2 =$$

$$= \left(C_1 - \overline{C} + C_1 - p_j + p_j' - \overline{C}\right)\left(C_1 - \overline{C} - C_1 + p_j - p_j' + \overline{C}\right) +$$

$$+ \left(C_h - \overline{C} + C_h + p_j - p_j' - \overline{C}\right)\left(C_h - \overline{C} - C_h - p_j + p_j' + \overline{C}\right) =$$

$$= \left(2C_1 - 2\overline{C} - p_j + p_j'\right)\left(p_j - p_j'\right) + \left(2C_h - 2\overline{C} + p_j - p_j'\right)\left(-p_j - p_j'\right) =$$

$$= \left(p_j - p_j'\right)\left(2C_1 - 2\overline{C} - p_j + p_j' - 2C_h + 2\overline{C} - p_j + p_j'\right) =$$

$$= \left(p_j - p_j'\right)\left(2\left(C_1 - C_h\left(p_j - p_j'\right)\right)\right)$$

Sabe-se que, por hipótese, $\left(p_j - p_j'\right) > 0$ e $C_1 - C_h = d > p_j - p_j'$, então $\left(C_1 - C_h - \left(p_j - p_j'\right)\right) > 0$ e $\tilde{g}$ - $\tilde{h} > 0$, o que permite concluir que a FASE 3 também faz com que o valor de $\tilde{f}$ decresça. Portanto, todas as trocas simples e duplas realizadas na execução do algoritmo fazem com que $\tilde{f}$ decresça, o que permite concluir que o algoritmo *termina* após um número finito de passos. O que permite dizer que o algoritmo também converge para valores de tempos de processamento reais.

2.5. LIMITANTE DE PIOR CASO

Para se chegar a um limitante de pior caso para o algoritmo são necessários alguns resultados preliminares:

Teorema 1 - Seja C_i a carga do processador M_i após uma passagem pela FASE 2 do algoritmo 3-FASES. Seja C^* o valor ótimo do problema. Então todas as tarefas no processador M_1 têm tempos de processamento maiores que C_1 - C^*.

Prova: Tem-se que $\overline{C} = \sum_{i=1}^{m} C_i / m$ e $\overline{C} \leq C^*$. O alvo $\Delta < \Delta' = C_1 - C_m$ é usado na FASE 2 para a escolha da tarefa a ser retirada de M_1 e realocada em M_m. A FASE 2 só pára quando não existir mais tarefa, em M_1, que satisfaça $p_j < \Delta'$.

Como $C_1 - C^* < C_1 - C_m$ e todo $J_j \in M_1$ tem $p_j \geq \Delta'$, segue-se que $p_j > C_1 - C^*, \forall J_j \in M_1$ após uma passagem pela FASE 2 do algoritmo 3-FASES. ■

Teorema 2 - Após uma passagem pela FASE 2 do algoritmo 3-FASES, se o processador M_1 tem k tarefas, então $\frac{C_1}{C^*} < \frac{k}{(k-1)}, k > 1$.

Prova: Assuma que $\frac{C_1}{C^*} \geq \frac{k}{(k-1)}$. Isto implica em $C_1 - C^* \geq \frac{C_1}{k}$.

Pelo TEOREMA 1, sabe-se que as k tarefas tem tempos de processamento maiores que $C_1 - C^*$, portanto $C_1 = p_{(\text{TAREFA } 1)} + \ldots + p_{(\text{TAREFA } k)} > k(C_1 - C^*) \geq C_1$ e o teorema está provado por contradição. ■

Pode-se afirmar que se $k = 1$ então, $C_1 - C^* = 0$ e o algoritmo encontrou a solução ótima.

Corolário - O limitante de pior caso para a FASE 2 do algoritmo 3-FASES é 2.

Este limitante de pior caso para a FASE 2 do algoritmo 3-FASES é o mesmo encontrado por FINN e HOROWITZ (1979) para o algoritmo 0-1 INTERCHANGE. Há, porém, uma forte conjectura, calcada na superioridade da FASE 3 sobre a FASE 2, que a

FASE 3 forneceria um limitante de pior caso de melhor qualidade. Porém, não foi possível encontrar este limitante, uma vez que duas tarefas estão envolvidas e há uma iteração entre a FASE 2 e a FASE 3 até que o critério de parada seja satisfeito.

Discussões mais aprofundadas sobre o desempenho do algoritmo 3-FASES, bem como comparações com outros algoritmos clássicos serão apresentadas no Capítulo 4, onde são apresentados os resultados computacionais. Em seguida será apresentado um esquema ótimo para a solução do problema *P*| |*Cmax*, que permitirá comparar a qualidade das soluções encontradas pelo algoritmo 3-FASES.

2.6. ESQUEMA ÓTIMO PARA A RESOLUÇÃO DO PROBLEMA *P* | | *Cmax*

O procedimento ótimo está baseado na solução de uma seqüência de Problemas de Empacotamento (*bin packing*) dentro de um esquema de busca dicotômica. Sendo dados n itens (tarefas), cada um com um tamanho (tempo de processamento) p_j ($j = 1,...,n$), e um número limitado de caixas (*bins*) idênticas de capacidade CAP, o Problema de Empacotamento (PE) é acomodar os itens nas caixas sem exceder suas capacidades, de modo que o número de caixas usado seja mínimo.

De posse de um limitante inferior LB e um limitante superior UB sobre *Cmax*, pode-se resolver *P*| |*Cmax* determinando, através de uma busca dicotômica, o menor valor de CAP (LB $\leq$ CAP $\leq$ UB), tal que o PE tenha uma solução usando não mais que m caixas. Neste caso usou-se a solução obtida pelo algoritmo 3-FASES, como limitante superior (UB) inicial. O limitante inferior foi escolhido seguindo a seqüência sugerida por DELL´AMICO E MARTELLO (1990), que produz um limitante com desempenho de pior caso igual a 2/3.

O limitante inferior mais imediato para o *P*| |*Cmax* é o valor da solução do problema quando permite-se preempção (limitante de McNaughton) e pode-se notar que este valor é o tempo médio de finalização $\overline{C}$ descrito na FASE 2 do algoritmo 3-FASES. Chamemos este limitante de $L_0 = \left\lceil \sum_{j=1}^{n} p_j / m \right\rceil$. Porém, este limitante tem um desempenho de pior caso arbitrariamente ruim, ou seja, assume valor 0 em casos patológicos. Um limitante melhor pode ser obtido quando retira-se a permissão de preempção, ou seja,

$L_1 = \max\left(L_0, \max_j \{p_j\} \right)$. Este limitante já apresenta um desempenho de pior caso um pouco melhor, assumindo o valor de 1/2. A melhoria deste limitante pode ser conseguida ao se considerar a relaxação do $P|\ |Cmax$ obtida pela eliminação das n-m-1 menores tarefas $J_{m+2},\ldots,J_n$ (supondo uma classificação em ordem decrescente de tempos de processamento). Isto produz uma solução ótima não menor que $p_m + p_{m+1}$, fornecendo um limitante com desempenho de pior caso igual a 2/3. Para a implementação da busca dicotômica foi utilizado, então, como limitante inferior LB = $\max\{L_1, p_m + p_{m+1}\}$.

De posse destes limitantes, calcula-se a capacidade das caixas como:

$$\text{CAP} = \left\lceil (\text{UB} - \text{LB})/2 \right\rceil + \text{LB} \qquad (1)$$

Resolve-se um PE, usando temporariamente CAP como a capacidade das caixas. O PE é resolvido através do algoritmo MTP desenvolvido por MARTELLO e TOTH (1990).

O seguinte esquema de busca dicotômica é aplicado. Enquanto o critério de parada não é satisfeito, ou seja LB = UB, a atualização dos limitantes é feita da seguinte maneira:

- se o número de caixas > m, então LB = CAP;
- se o número de caixas ≤ m, então UB = CAP - 1.

Com os novos limitantes, recalcula-se CAP a partir de (1) e resolve-se um novo PE até que o critério de parada seja satisfeito ou algum dispositivo externo de parada seja acionado (por exemplo, um limitante no tempo computacional).

O algoritmo ótimo MTP, usado para resolução do PE dentro do esquema de busca dicotômica, é um algoritmo de *branch & bound* baseado na estratégia de ramificação FFD (*first-fit decreasing*). Os itens são inicialmente classificados em ordem decrescente de tamanho (p_j). O algoritmo MTP identifica as caixas com um índice crescente de acordo com a ordem em que elas são inicializadas (isto é, quando o primeiro item é alocado à ela). A cada nó de decisão, o primeiro (isto é, o maior) item livre é alocado, sucessivamente, à uma caixa factível inicializada (em ordem crescente de índice) ou, quando esta não existir, à uma nova caixa. A cada passo de avanço (*forward step*) do algoritmo: (a) procedimentos de sondagem são utilizados com objetivo de reduzir o espectro de busca; (b) quando não ocorre sondagem, os algoritmos de aproximação FFD (descrito acima), BFD (*best-fit decreasing* - os itens estão ordenados em ordem decrescente de tamanho e a alocação é feita na caixa factível que produza a melhor solução corrente ou, quando esta não existir, à

uma nova caixa) e WFD (*worst-fit decreasing* - os itens estão ordenados em ordem decrescente de tamanho e a alocação é feita na caixa factível de maior capacidade residual ou, quando esta não existir, à uma nova caixa) são aplicados no sentido de tentar uma melhoria na solução corrente. Um passo de retorno (*backtracking*) implica na remoção de um item corrente J_j^* da sua caixa atual M_i^* e sua realocação na próxima caixa factível (o passo de retorno ocorre se M_i^* foi inicializada com J_j^*, uma vez que inicializando M_{i+1}^* com J_j^* poderia produzir uma situação idêntica a atual). Considerando Z o valor da solução ótima atual, sempre que um passo de retorno ocorrer, ele é realizado no último item alocado à caixa de índice não maior que Z - 2, uma vez que um passo de retorno de um item à uma caixa de índice Z ou Z - 1 poderá produzir soluções requerendo no mínimo Z bins.

Um *critério de dominância* também é usado entre os nós de decisão. Quando o item corrente J_j^* é alocado à caixa M_i^* cuja capacidade residual $\overline{C}_i^*$ é menor que $p_j^* + p_n$, , esta alocação domina todas as alocações em M_i^* de itens $J_j > J_j^*$, pois não é permitida a alocação de nem mais um item além dele. Portanto, esta alocação "*fecha*" a caixa, no sentido que, após um passo de retorno envolvendo J_j^* , nenhum item $J_j \in \left\{ J_k > J_j^* \,/\, p_k + p_n > \overline{C}_i^* \right\}$ é alocado a M_i^* . A caixa é considerada "*reaberta*" quando, um item $J_j > J_j^*$ para o qual $p_j + p_n \leq \overline{C}_i^*$, é candidato a alocação e, caso não houver algum item satisfazendo estas condições, quando for realizado o primeiro passo de retorno envolvendo um item $J_l < J_j^*$.

CAPÍTULO 3

Algoritmos para Solução do Problema de Sequenciamento em Processadores Paralelos com Tempos de Processamento Dependentes da Sequência - *P / tpds / Cmax*

3.1. INTRODUÇÃO

Como visto no capítulo anterior, o problema de alocar n tarefas a m processadores idênticos com o objetivo de minimizar o tempo máximo de finalização é conhecido como o Problema de Sequenciamento em Processadores Paralelos. Entretanto, pode ocorrer que para executar uma tarefa distinta da anterior num processador seja necessário realizar operações de modo que o processador fique devidamente preparado para receber e executar a próxima tarefa. Como exemplo destas operações estão: preparação do processador, limpeza, troca de peças, ferramentas ou desenhos, aquecimento/resfriamento, ajustes de um modo geral, etc ... Estas operações feitas no processador consomem um tempo total que se denomina *tempo de preparação* (*setup time*).

O tempo de preparação pode representar, em certas situações, uma parcela significativa do tempo necessário para um processador executar um dado conjunto de tarefas. A redução destes tempos de preparação é freqüentemente alcançada quando uma fábrica adota um *lay out* que aloca os processadores em células de produção usando tecnologia de grupo e classificação por famílias de tarefas (BURBIDGE - 1975).

Em muitos casos estes tempos de preparação são dependentes do estado anterior em que o processador se encontrava e da tarefa que virá a ser realizada. Nestas situações recebem o nome de *tempos de preparação dependentes da seqüência*. Eles ocorrem normalmente quando um mesmo equipamento produz uma gama variada de produtos, ou onde um processador multi tarefa realiza uma variedade de tarefas.

O uso de um sistema simples para produzir, por exemplo, diferentes componentes químicos pode necessitar que um certo tempo seja necessário para a limpeza do

equipamento no período existente entre o processamento de compostos diferentes, com o objetivo de assegurar ao processo níveis baixos de impureza. Além disso, não é difícil vizualizar situações em que a duração deste processo de limpeza depende, tanto do último composto químico processado, quanto do composto que será processado.

Exemplos típicos ocorrem na produção de tintas de diferentes cores, detergentes com níveis de concentração diferentes e misturas de combustíveis. Na manufatura há inúmeros processos onde a troca de ferramentas, inspeções ou rearranjo dos processadores impõe um tempo de preparação (BAKER - 1974).

Tendo um conjunto de processadores idênticos que, por hipótese, têm o mesmo estado inicial, os tempos de processamento envolvem a soma de duas parcelas: o tempo de preparação do processador para a troca da tarefa *i* para *j*, e o tempo de processamento da tarefa *j* propriamente dito. Portanto, o problema agora, diferentemente do *P*| |*Cmax*, não é simplesmente alocar as tarefas aos processadores, mas deve-se encontrar também a melhor *seqüência* de execução das mesmas de modo a minimizar o tempo de execução total.

Neste capítulo é apresentada uma estratégia heurística para a resolução deste problema. A estratégia é composta de três fases: a primeira consiste de uma alocação inicial das tarefas aos processadores; a segunda fase consiste na aplicação de um algoritmo heurístico, governado por uma metaheurística de tipo Busca Tabu, que tenta melhorar a solução inicial realocando tarefas e, a cada realocação, examinando várias alternativas para a seqüência de execução das tarefas. Finalmente, uma terceira fase de pós-otimização procura melhorar ainda mais a solução anterior agindo exclusivamente na seqüência de execução das tarefas. Além disso, uma estratégia ótima também foi desenvolvida para a solução deste problema.

O capítulo está estruturado da seguinte forma: inicialmente o problema a ser resolvido é formalmente apresentado, seguindo-se o desenvolvimento da estratégia heurística e da estratégia ótima de resolução.

3.2. O PROBLEMA *P* | *tpds* | *Cmax*

O problema é alocar n tarefas, $J_1, J_2, ..., J_n$, a m processadores paralelos idênticos, $M_1, M_2, ..., M_m$, de uma forma não preemptiva (isto é, uma vez que determinada tarefa

esteja alocada a um processador, ela deverá permanecer no mesmo até o seu término, sem interrupções), objetivando minimizar o tempo máximo de finalização de todas as tarefas (*makespan*).

O tempo de processamento de uma tarefa J_j é composto da soma de duas parcelas: um tempo de execução inteiro e positivo p_j e um tempo de preparação inteiro e positivo para a tarefa J_j, se ela foi executada imediatamente após a tarefa i, chamado s_{ij}. Portanto, o tempo de processamento pode ser expresso como $t_{ij} = p_j + s_{ij}$. Assume-se que não existe nenhuma relação de simetria, isto é, $t_{ij} \neq t_{ji}$. Como já observado no Capítulo 1, o problema continua essencialmente sendo o $P|\ |Cmax$, porém com tempos de processamento dependentes da seqüência ($P|tpds|Cmax$).

3.3. ALGORITMO HEURÍSTICO PARA A SOLUÇÃO DO PROBLEMA *P | tpds | Cmax*

Inicialmente o algoritmo proposto constrói uma solução de partida. Então, dentro de um esquema de busca local e com a supervisão da metaheurística Busca Tabu, soluções vizinhas são exploradas removendo uma tarefa de um processador e reinserindo-a em outro processador. Por fim, um procedimento de pós-otimização é aplicado a cada processador em separado.

As inserções são feitas usando a idéia proposta por GENDREAU, HERTZ e LAPORTE (1992) de um procedimento geral de inserção denominado GENI (de *generalized insertion*), que é menos míope e mais poderoso que os procedimentos padrão de inserção (por exemplo, 2-OPT). GENI permite que uma tarefa seja incluída no processador contendo pelo menos um de seus vizinhos mais próximos, e cada inserção é executada simultaneamente com uma reotimização local da seqüência de tarefas do processador em questão.

A fase de pós-otimização aplica o procedimento US (*unstringing - stringing*) também proposto pelos mesmos autores de GENI. O procedimento US remove, sucessivamente, tarefas de um processador, usando o reverso de GENI e reinsere-a no mesmo processador usando GENI.

Como já mencionado, nesta tese faz-se uso de Busca Tabu (BT). BT é uma metaheurística de otimização global proposto por GLOVER (1989 e 1990a) e, paralelamente por HANSEN (1986), que tem sido amplamente aplicada com sucesso na resolução de vários problemas difíceis de otimização combinatória (MÜLLER e FRANÇA - 1990, PUREZA e FRANÇA - 1991, SILVEIRA - 1993, LAGUNA, BARNES e GLOVER - 1991, BARNES e LAGUNA - 1992, DELL´AMICO e TRUBIAN - 1992, WOODRUFF e SPEARMAN - 1992, GENDREAU, HERTZ e LAPORTE - 1991, SEMET e TAILLARD - 1992, BLAND e DAWSON - 1991, HERTZ e de WERRA - 1987, HERTZ, JAUMARD e ARAGÃO - 1992, de WERRA e HERTZ - 1989, TAILLARD - 1991, OLIVEIRA e STROUD - 1989, GLOVER e LAGUNA - 1992b, GENDREAU, SORIANO e SALVAIL - 1992, MALEK, GURUSWAMY, PANDYA e OWENS - 1989).

A seguir é apresentada a adaptação da idéia dos procedimentos GENI/US , originalmente propostos para problemas de roteamento de veículos, para o problema de sequenciamento que estamos tratando. Os fundamentos da Busca Tabu serão apresentados na forma de apêndice.

3.3.1. Procedimento de Inserção GENI (GENeralized Insertion Procedure)

O procedimento de inserção GENI foi inicialmente proposto para o problema do caixeiro viajante simétrico (GENDREAU, HERTZ e LAPORTE - 1992). O enfoque de GENI é a integração de passos de otimização local embutidos num procedimento de inserção. Idéias semelhantes foram apresentadas por STEIGLITZ e WEINER (1968) cujo método 3-OPT dinâmico executa uma seqüência de movimentos consistindo da melhor inserção simples seguida de uma busca 3-OPT.

De um ponto de vista puramente conceitual, existem no procedimento básico de GENI dois tipos de inserções. As inserções do primeiro tipo (que englobam as inserções **Tipo I** e **Tipo II**, detalhadas a seguir) são equivalentes a selecionar o melhor entre vários movimentos, cada um consistindo de uma inserção simples seguida de somente uma troca

3-OPT. Nas inserções do segundo tipo (que compreendem as inserções **Tipo III** e **Tipo IV**) realiza-se, de fato, uma troca 4-OPT.

O que faz GENI eficiente é o fato de que o número de inserções candidatas fica restrito à vizinhança do elemento a ser adicionado. Outro fator importante é que a progressão da busca é limitada aos movimentos mais promissores, uma vez que a reordenação dos elementos, bem como suas reconecções são feitas levando-se em conta a proximidade dos mesmos.

GENI mostrou ser, no contexto do problema do caixeiro viajante simétrico, um procedimento de inserção muito eficaz. A seguir será apresentada uma adaptação deste procedimento ao problema aqui tratado ($P \mid tpds \mid Cmax$) que, dentre outras características, é assimétrico, ou seja, $t_{ij} \neq t_{ji}$. Primeiramente vai-se apresentar a estrutura de dados utilizada para a representação das tarefas alocadas a cada processador e , após, será apresentada a implementação do procedimento GENI para o caso aqui estudado.

Suponha que as tarefas J_1 , J_2 , J_3 , J_4 e J_5 estão alocadas ao processador M_1 conforme a figura 3.4.

Figura 3.4 - Disposição das tarefas no processador M_1

Cria-se uma lista encadeada, tendo como cabeça um indicativo do processador em questão. A última tarefa tem um apontador para a cabeça da lista. Esta estrutura pode ser representada como um grafo orientado (veja figura 3.5).

UNICAMP
BIBLIOTECA CENTRAL

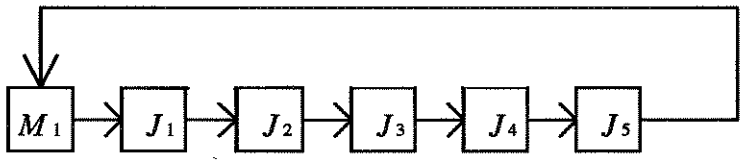


Figura 3.5 - Representação da distribuição das tarefas como um grafo

A seqüência de execução das tarefas é dependente da orientação dos apontadores, ou seja, J_1 é a primeira tarefa a ser executada, seguida de J_2 , J_3 e assim por diante até J_5 . Esta estrutura é muito apropriada, pois como os tempos de processamento são dependentes da seqüência e assimétricos, fica muito fácil a inversão de parte da seqüência, ou mesmo da seqüência toda.

Outro ponto que deve ser ressaltado é que a matriz de tempos de processamento inclui uma coluna 0, onde o elemento t_{0i} representa o tempo de execução da tarefa i, a primeira tarefa da seqüência. Consideramos $t_{j0} = 0, \forall\ j, j = 1,...,n.$

A principal característica do GENI é que a inserção de uma tarefa J_x em um processador não necessita ser feita entre duas tarefas consecutivas. Entretanto, após a inserção, estas duas tarefas se tornam adjacentes a J_x . Para uma dada orientação da seqüência das tarefas, considere J_k uma tarefa na subseqüência de J_j para J_i e J_l uma tarefa na subseqüência de J_i para J_j . Devido a assimetria, a inserção de J_x entre J_i e J_j pode ser feita de quatro maneiras.

Inserção Tipo I: Aqui $J_k \neq J_i$ e $J_k \neq J_j$. Inserir J_x na seqüência resulta na deleção das ligações (J_i , J_{i+1}), (J_j , J_{j+1}) e (J_k , J_{k+1}), que serão substituidas pelas ligações (J_i , J_x), (J_x , J_j), (J_{i+1} , J_k) e (J_{j+1} , J_{k+1}). Isto resulta na inversão de duas subseqüências $(J_{i+1} , ..., J_j)$ e $(J_{j+1} ,..., J_k)$. Isto pode ser melhor visto na figura 3.6. Note que, neste caso, a tarefa J_x precede J_k na seqüência final. As convenções sobre os arcos das figuras estão mostradas na figura na figura 3.7.

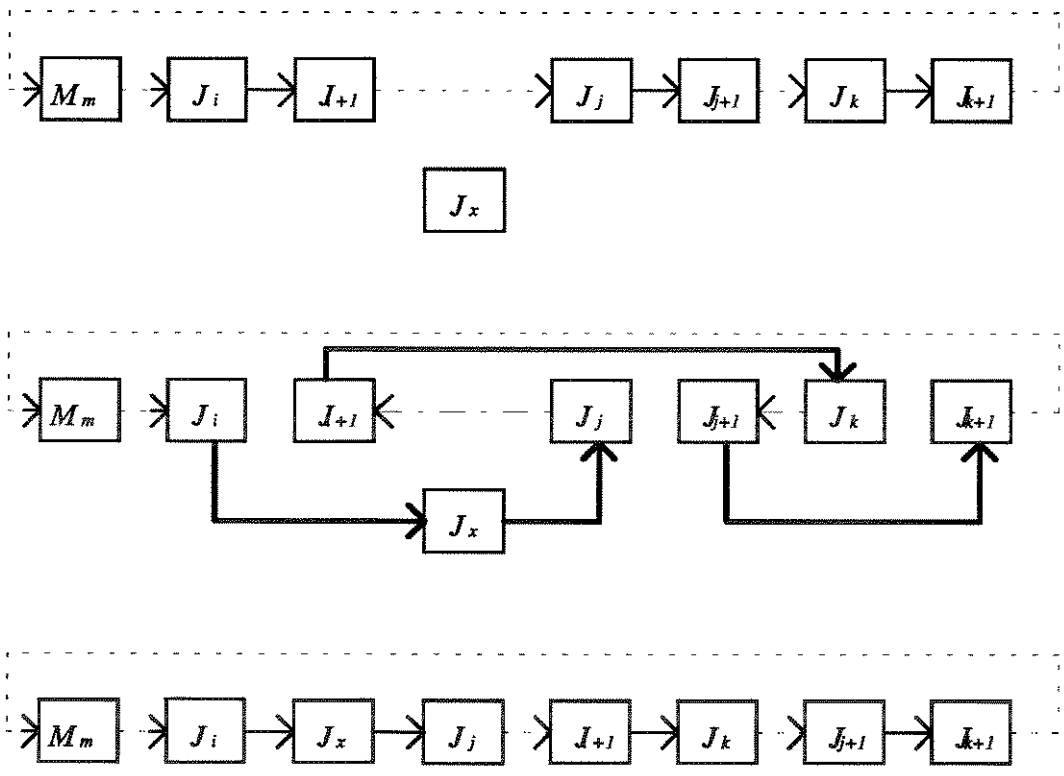


Figura 3.6 - Inserção do Tipo I

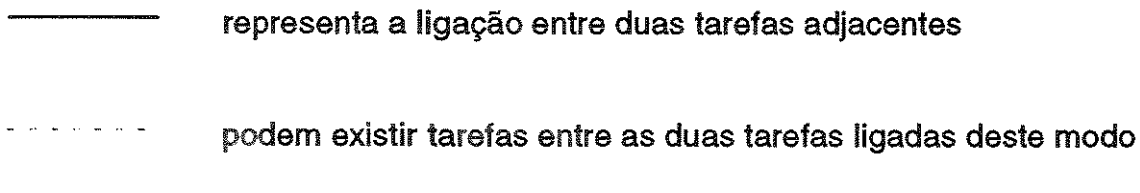


Figura 3.7 - Convenções usadas

Inserção do Tipo II (Reverso da Inserção Tipo I): A inserção Tipo II será feita de um modo que a seqüência original resultará praticamente invertida após a inserção, com isto procurando explorar melhores soluções levando em conta a assimetria do problema. Esta inversão pode ser melhor notada ao se observar que, neste caso, J_x é precedido por J_k. Note que as possibilidades envolvendo duas tarefas estão esgotadas. Novamente tem-se $J_k \neq J_i$ e $J_k \neq J_j$. Inserir J_x na seqüência resulta na deleção das ligações (J_{i-1}, J_i), (J_{j-1}, J_j) e (J_{k-1}, J_k) que serão substituidas pelas ligações (J_i, J_x), (J_x, J_j), (J_k, J_{j-1}), (J_{k-1}, J_{i-1}). Isto resulta na inversão de duas subseqüências $(J_i, \ldots, J_{j-1})$ e $(J_k, \ldots, J_{i-1})$. Observe a figura 3.8 para uma melhor visualização do processo.

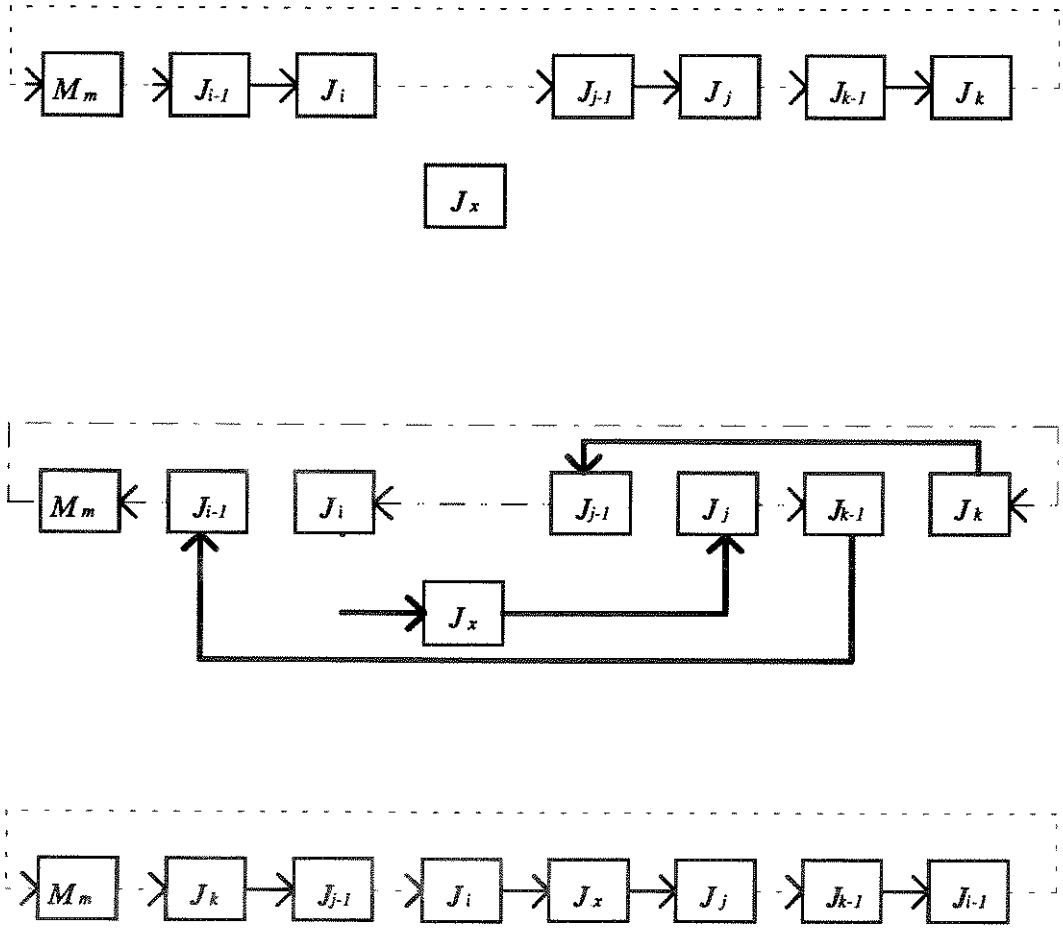


Figura 3.8 - Inserção do Tipo II

Inserção do Tipo III: Aqui $J_k \neq J_j$ e $J_k \neq J_{j+1}$; $J_l \neq J_i$ e $J_l \neq J_{i+1}$. Inserir J_x na seqüência resulta na deleção das ligações (J_i , J_{i+1}) , (J_{l-1} , J_l) , (J_j , J_{j+1}) e (J_{k-1} , J_k) que serão substituidas pelas ligações (J_i , J_x) , (J_x , J_j) , (J_l , J_{j+1}) , (J_{k-1} , J_{l-1}) e (J_{i+1} , J_k). Isto resulta na inversão de duas subseqüências $(J_{i+1} ,..., J_{l-1})$ e $(J_l ,..., J_j)$. Note que, na seqüência final J_x precede J_l que, por sua vez, precede J_k . A figura 3.9 mostra o processo de inserção.

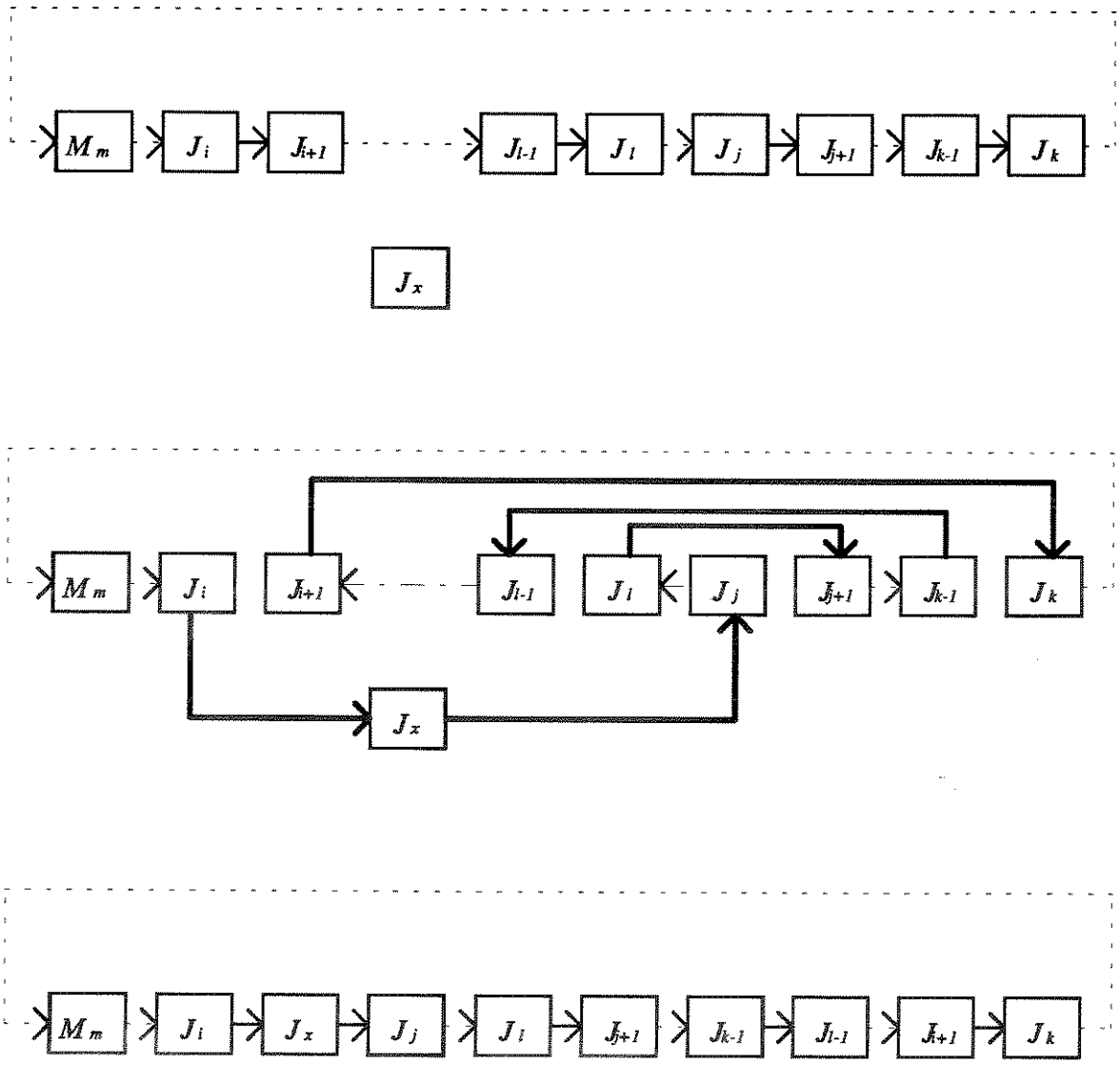


Figura 3.9 - Inserção do Tipo III

Inserção do Tipo IV (Reverso da Inserção do Tipo III): Novamente, este tipo de inserção levará em conta a assimetria do problema e a orientação da seqüência original resultará bastante modificada. Aqui $J_k \neq J_j$ e $J_k \neq_{j+1}$; $J_l \neq J_i$ e $J_l \neq J_{i+1}$. Inserir J_x na seqüência resulta na deleção das ligações (J_{i-1}, J_i), (J_l, J_{l+1}), (J_{j-1}, J_j) e (J_k, J_{k+1}) que serão substituidas pelas ligações (J_i, J_x), (J_x, J_j), (J_{i-1}, J_l), (J_{l+1}, J_{k+1}) e (J_k, J_{j-1}). Isto resulta na inversão de duas subseqüências $(J_i, ..., J_l)$ e $(J_{l+1}, ..., J_{j-1})$. A figura 3.10 mostra o processo de inserção. Note que, na seqüência final, J_l precede J_x que, por sua vez, precede J_k. Poder-se-ia pensar em uma ordem de seqüência na qual J_l precedesse J_k, que precedesse J_x. Porém, este é o mesmo caso que ocorre na inserção do Tipo III, pois o referencial do processador é como se fosse um nó fantasma, funcionando apenas como referencial e elo de ligação. Portanto, se partirmos de J_l, percorrendo a seqüência, veremos que J_l precede J_k, que precede J_x e, assim, as possibilidades envolvendo três tarefas estão esgotadas.

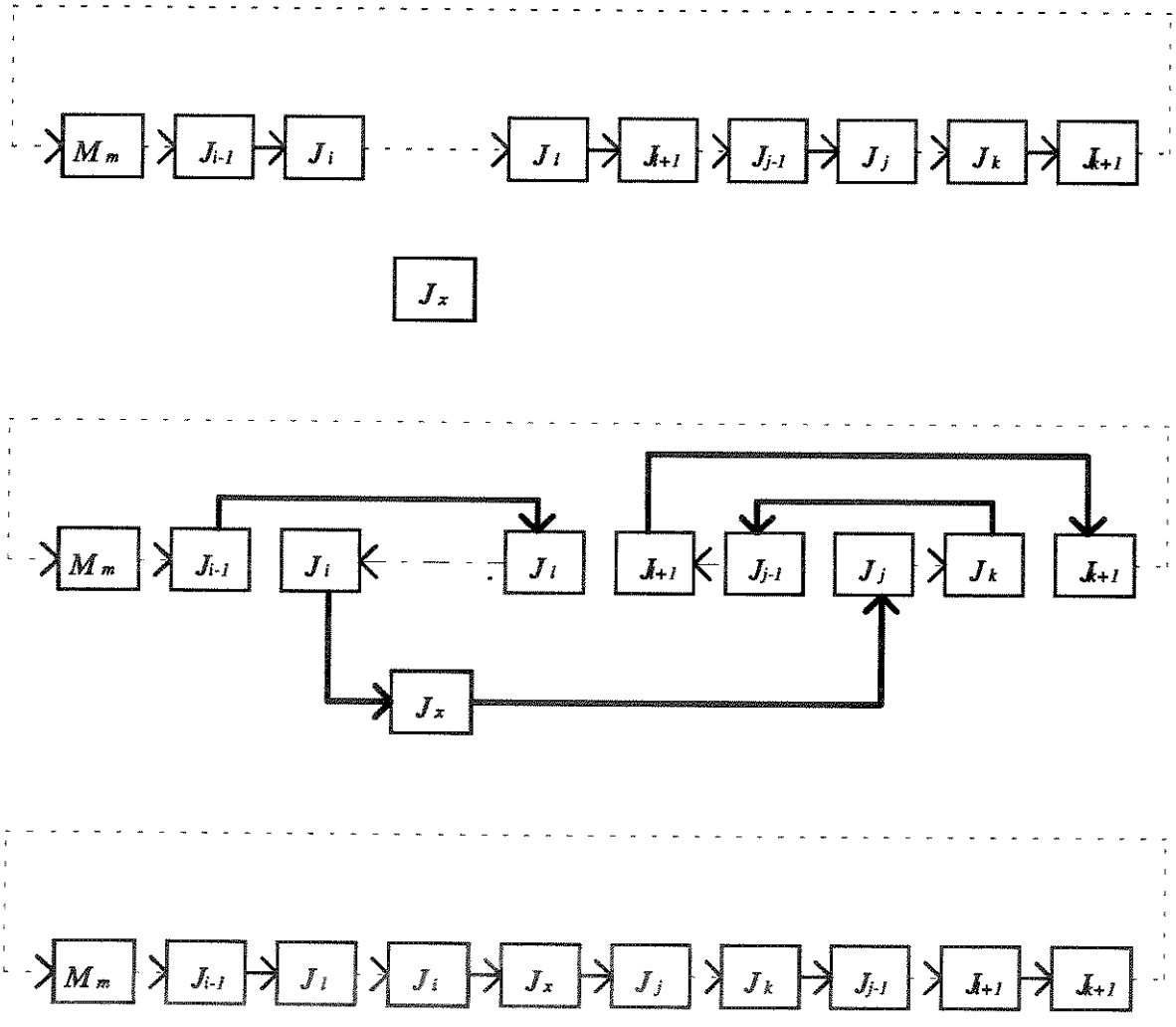


Figura 3.10 - Inserção do Tipo IV

Uma questão não levada em conta pelo procedimento original GENI é o caso da inserção quando só existe uma ou nenhuma tarefa alocada ao processador. No problema de sequenciamento aqui tratado, devido à retirada e inserção de tarefas, este caso é possível, gerando um quinto e um sexto tipos de inserção.

Inserção do Tipo V (Somente uma tarefa alocada ao processador): Inserir a tarefa J_x quando o processador só tem uma tarefa J_i alocada a ele, pode ser feito de duas maneiras: ou J_x é inserido antes de J_i, ou depois. O algoritmo testa ambas as possibilidades e decide qual a inserção é mais vantajosa. Veja na figura 3.11.

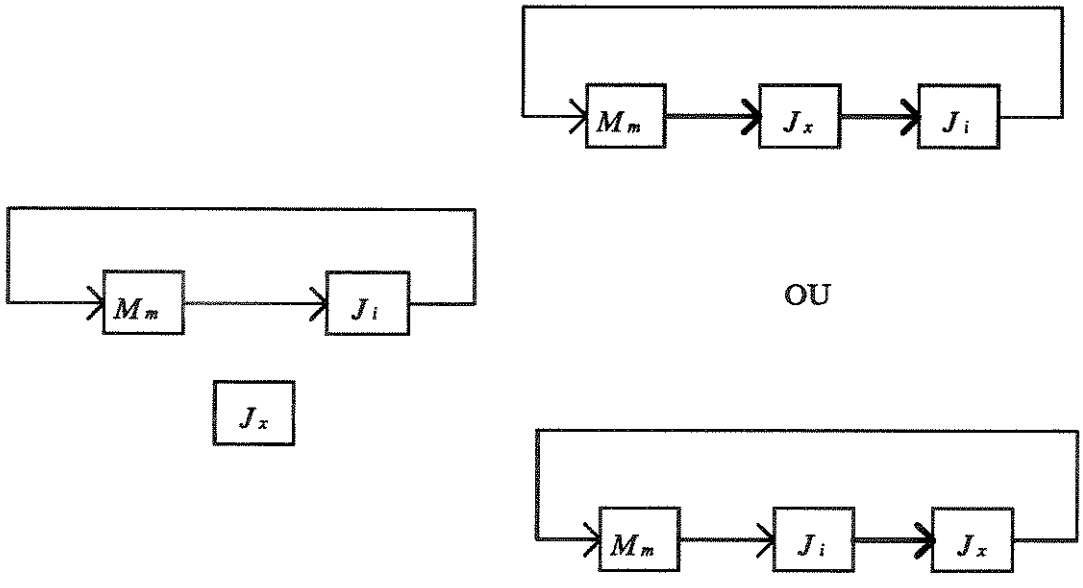


Figura 3.11 - Inserção do Tipo V

Inserção do Tipo VI (Processador vazio): Inserir a tarefa J_x no processador vazio requer somente que se façam as ligações dele ao processador, como mostrado na figura 3.12.

Figura 3.12 - Inserção do Tipo VI

Determinada como é feita a inserção de uma tarefa em um processador, agora será mostrado como fazer a retirada de uma tarefa de um determinado processador. O procedimento de retirada de tarefas descrito a seguir é utilizado tanto na rotina que realoca tarefas aos processadores (Fase 2 do método), quanto na rotina de pós-otimização (Fase 3 do método).

3.3.2. Procedimento de Retirada US (Unstringing and Stringing)

A retirada de tarefas está baseada no procedimento de pós-otimização US (*Unstringing and Stringing*) de GENDREAU, HERTZ e LAPORTE (1992), mais precisamente no procedimento U (*Unstringing*) do método. US usa o conceito de *vizinhança local* e *vizinhança global* que tem um papel fundamental para o bom desempenho computacional do método.

Vizinhos globais de uma tarefa J_i são as suas mais próximas tarefas sucessoras e predecessoras dentre todas as tarefas, exceto ela própria (J_i). Entenda-se por *mais próxima sucessora de uma tarefa* J_i aquela tarefa J_j que tiver o menor t_{ij} ; e por *mais próxima predecessora de uma tarefa* J_i a tarefa J_j com menor t_{ji} . O número de vizinhos globais é um inteiro p, que é um parâmetro do algoritmo.

Vizinhos locais de uma tarefa J_i , relativa ao processador k, são os mais próximos sucessores e predecessores da tarefa J_i , dentre as tarefas já alocadas ao processador em questão. O número de predecessores e sucessores locais é limitado a um inteiro q, um parâmetro do algoritmo.

O conjunto de vizinhos locais de uma tarefa deve ser dinamicamente atualizado no decorrer do algoritmo, ao contrário do conjunto de vizinhos globais que é imutável. A vizinhança local é usada para facilitar as inserções e retiradas de tarefas de um processador em particular. Por este motivo ela é definida em cada processador (isto é, cada tarefa tem m vizinhanças locais). Poder-se-ia ter usado o conceito de uma *vizinhança global expandida*, mas isso traria o problema da identificação de qual tarefa está em qual processador e se gastaria mais tempo para gerenciar este tipo de estrutura. A vizinhança local funciona para identificar o local das inserções e retiradas, enquanto que a vizinhança global serve para restringir o espaço de busca, uma vez que só são considerados para análise os processadores com um número mínimo de tarefas pertencentes a esta vizinhança. As estruturas de vizinhança local (considerando 5 vizinhos locais) e global (considerando 7 vizinhos globais), para uma dada tarefa da matriz de tempos de processamento mostrada na figura 3.13, podem ser vistas na figura 3.14.

t_{ij}	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	-	32	30	54	9	42	38	42	5	47	47	55	34	55	55	39	56	45	54	25	54
1	0	-	63	47	25	45	7	50	35	59	78	33	41	87	37	64	54	13	55	10	24
2	0	21	-	76	38	60	69	56	28	55	24	83	52	29	79	28	70	76	67	55	84
3	0	33	26	-	54	16	44	23	59	32	76	25	24	86	84	92	100	48	99	41	65
4	0	58	61	37	-	45	32	46	10	52	55	52	37	64	46	41	50	38	48	19	46
5	0	49	49	25	13	-	45	8	48	18	60	35	9	70	82	79	94	50	92	37	67
6	0	23	29	18	36	27	-	51	42	60	83	28	42	92	40	71	60	7	60	14	22
7	0	74	82	61	25	38	54	-	47	10	53	43	9	63	86	77	95	57	94	41	73
8	0	16	12	36	69	58	34	88	-	51	47	60	39	55	54	33	53	48	50	29	56
9	0	53	59	38	10	17	31	23	66	-	48	53	18	57	94	79	102	66	100	50	83
10	0	40	24	17	52	39	33	77	36	54	-	90	53	10	100	51	94	90	91	69	101
11	0	37	21	16	52	39	31	76	33	54	3	-	39	100	66	93	87	28	88	33	44
12	0	38	27	7	41	28	26	65	39	43	11	11	-	63	77	70	86	48	85	32	64
13	0	67	67	42	12	18	45	30	76	22	54	55	44	-	108	54	100	99	96	78	110
14	0	49	43	18	26	14	29	51	54	31	28	28	17	26	-	66	28	38	31	45	26
15	0	40	54	73	94	87	61	105	42	87	77	74	77	105	89	-	49	77	45	60	79
16	0	17	17	41	72	62	37	90	5	68	41	38	44	80	58	37	-	61	4	58	52
17	0	71	85	71	45	54	57	28	87	37	87	86	77	55	67	92	88	-	62	21	17
18	0	71	79	58	23	35	51	3	85	20	74	73	63	29	49	102	87	28	-	58	54
19	0	9	24	28	50	41	14	65	24	44	38	36	34	59	42	47	25	63	62	-	34
20	0	48	53	32	11	13	26	29	60	6	48	48	37	23	26	83	63	41	26	39	-

Figura 3.13 - Matriz de Tempos de Processamento para $m = 2$ e $n = 20$.

Alocação Inicial

Processador 1 : 0 -15 - 1 - 19 - 6 - 17 - 20 - 5 - 7 -12 - 3 - 11 (Carga = 189)

Processador 2 : 0 - 4 - 18 - 14 - 16 - 8 - 2 - 10 - 13 - 9 (Carga = 207)

Sucessores Locais : Processador 1 = { 0 , 19 , 17 , 6 , 20 }
Processador 2 = { 0 , 14 , 4 , 8 , 16 }

Predecessores Locais : Processador 1 = { 0 , 3 , 19 , 6 , 11 }
Processador 2 = { 0 , 2 , 10 , 8 , 16 }

Sucessores Globais : { 0 , 6 , 19 , 17 , 20 , 4 , 11 }

Predecessores Globais : { 19 , 8 , 16 , 2 , 6 , 0 , 3 }

Figura 3.14 - Resultado da Alocação Inicial e Vizinhança Local e Global da Tarefa J_1 .

Assim como no procedimento GENI, aqui existem quatro maneiras de se fazer a retirada de uma tarefa J_i de um determinado processador.

Retirada do Tipo I: Seja J_j pertencente à vizinhança local de J_{i+1} e, para uma dada orientação da seqüência, seja J_k pertencente à vizinhança local de J_{i-1} , com J_k fazendo parte da subseqüência (J_{i+1} ,...., J_{j-1}). Então para fazer a retirada de J_i , as ligações (J_{i-1} , J_i) (J_i , J_{i+1}) , (J_k , J_{k+1}) e (J_j , J_{j+1}) são eliminadas e substituidas pelas ligações (J_{i-1} , J_k) , (J_{i+1} , J_j) , (J_{k+1} , J_{j+1}), enquanto que as seqüências (J_{i+1} ,...., J_k) e (J_{k+1} ,...., J_j) são invertidas. Observe na figura 3.15.

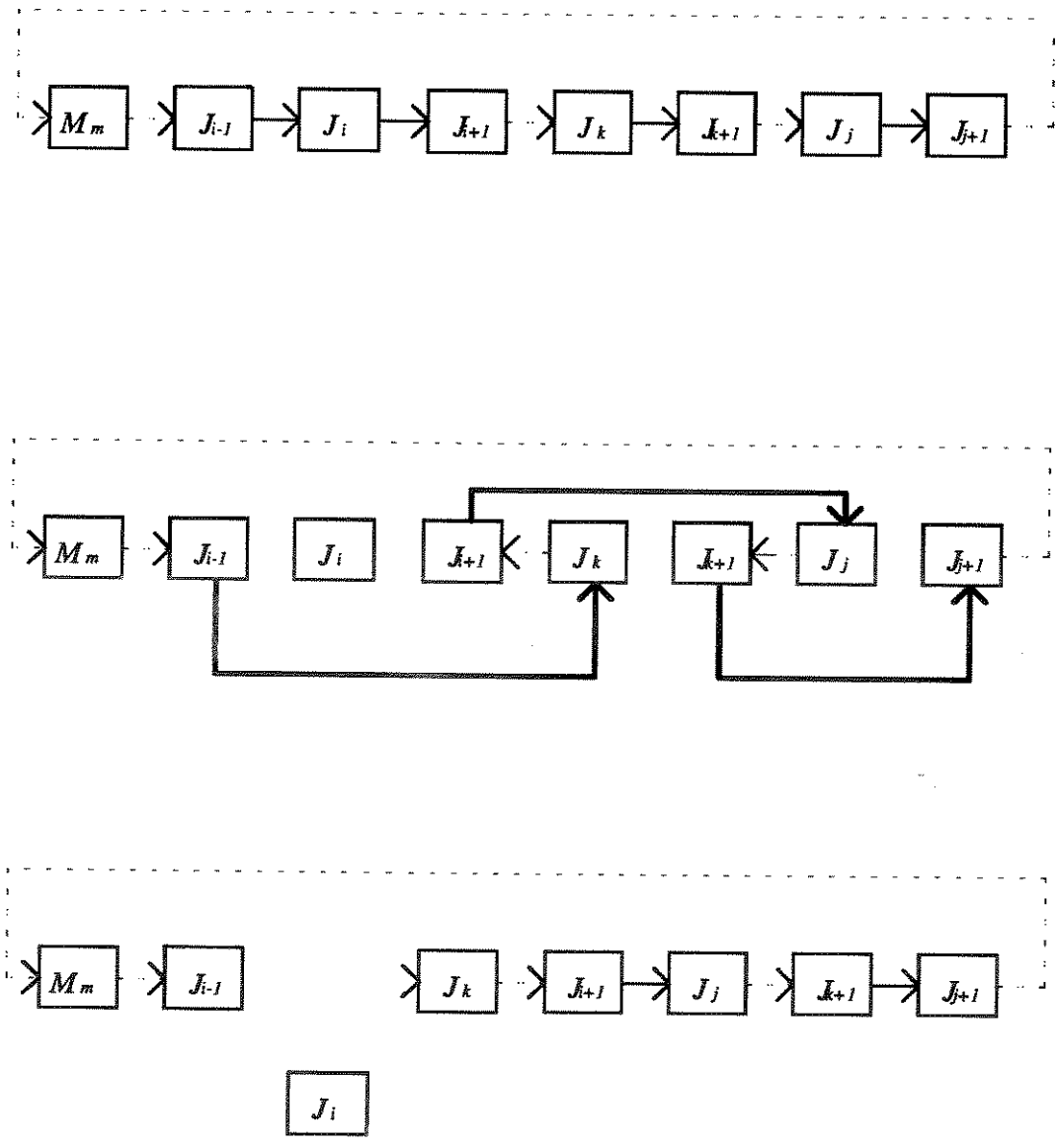


Figura 3.15 - Retirada do Tipo I

Retirada do Tipo II (Inverso da Retirada do Tipo I): Novamente a questão da assimetria é levada em conta, trazendo certas vantagens a este tipo de retirada que, praticamente inverte a seqüência original. Como anteriormente, J_j pertence à vizinhança local de J_{i+1} e, para uma dada orientação da seqüência, J_k pertence à vizinhança local de J_{i-1} e faz parte da subseqüência (J_{i+2} ,..., J_{j-1}). Para retirar J_i , as ligações (J_{i-1} , J_i) (J_i , J_{i+1}) , (J_{k-1} , J_k) e (J_{j-1} , J_j) são eliminadas e substituidas pelas ligações (J_j , J_{i+1}) , (J_{k-1} , J_{j-1}) e (J_k , J_{i-1}). Os caminhos (J_j ,..., J_{i-1}) e (J_k ,..., J_{j-1}) são invertidos. Observe na figura 3.16.

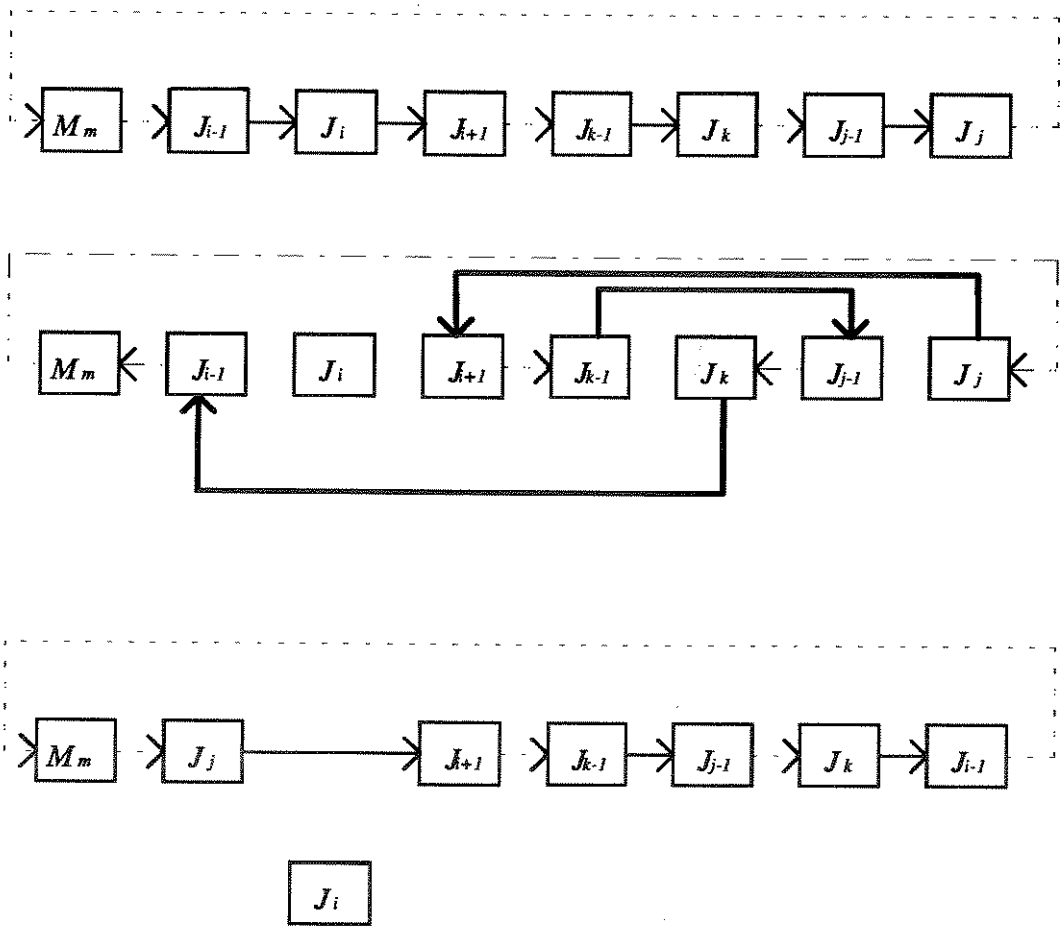


Figura 3.16 - Retirada do Tipo II

Retirada do Tipo III: Seja J_j pertencente à vizinhança local de J_{i+1} e, para uma dada orientação da seqüência, J_k pertence à vizinhança local de J_{i-1} e faz parte da subseqüência (J_{j+1} ,..., J_{i-2}); seja também J_l pertencente à vizinhança local de J_{k+1} na subseqüência (J_j ,..., J_{k+1}). Para retirar J_i , as ligações (J_{i-1} , J_i) (J_i , J_{i+1}) , (J_{j-1} , J_j) , (J_l , J_{l+1}) e (J_k , J_{k+1}) são eliminadas e substituidas pelas ligações (J_{i-1} , J_k) , (J_{l+1} , J_{j-1}) , (J_{i+1} , J_j) e (J_l , J_{k+1}). Os caminhos (J_{i+1} ,..., J_{j-1}) e (J_{l+1} ,..., J_k) são invertidos. A figura 3.17 mostra esta retirada.

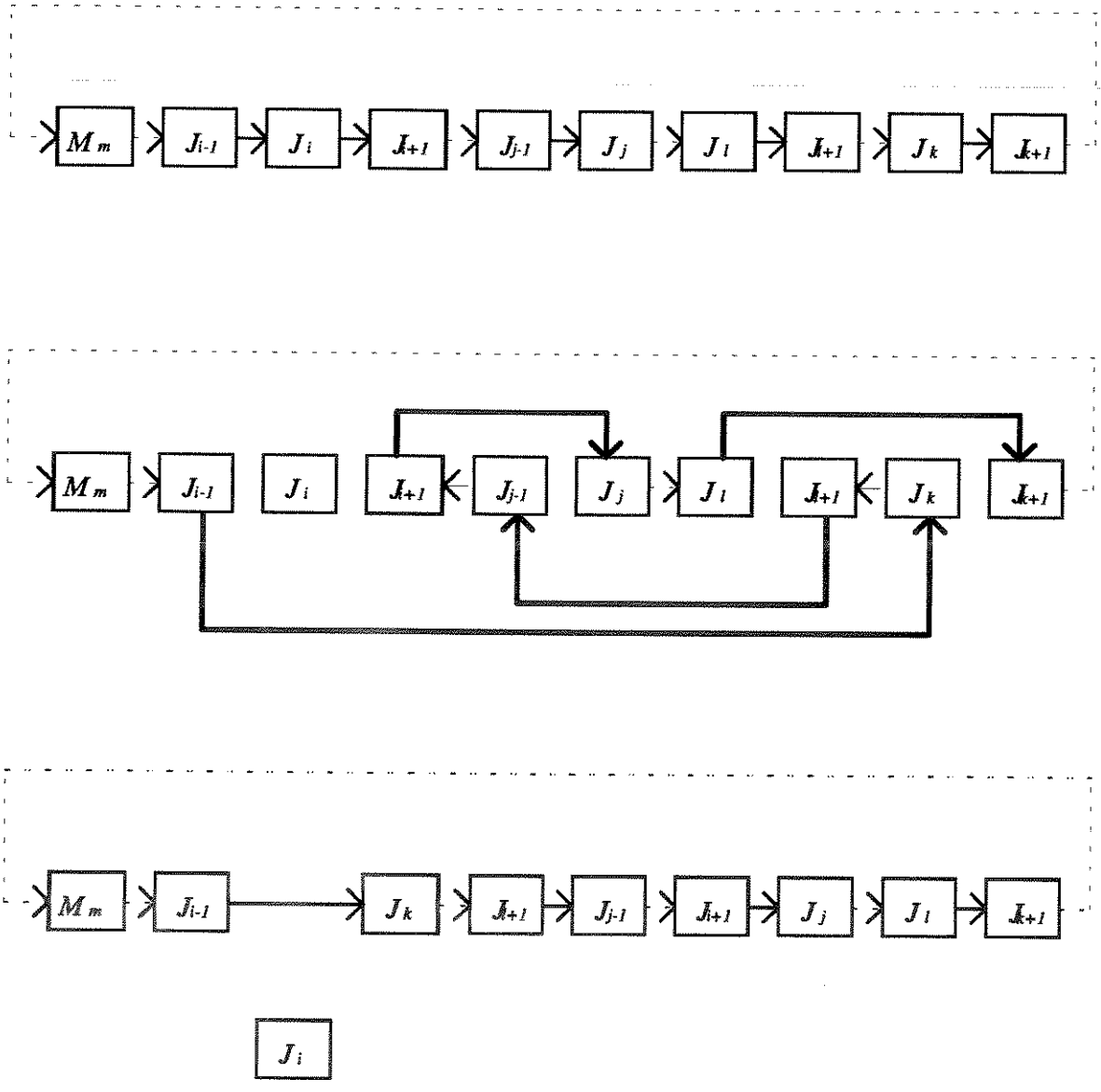


Figura 3.17 - Retirada do Tipo III

Retirada do Tipo IV (Inverso da Retirada do Tipo III): Aqui, a assimetria é levada em conta praticamente invertendo a seqüência original. Tem-se que J_j pertence à vizinhança local de J_{i+1} e, para uma dada orientação da seqüência, J_k pertence à vizinhança local de J_{i-1} e faz parte da subseqüência $(J_{i+1}, \ldots, J_{i-2})$; seja também J_l pertencente à vizinhança local de J_{j-1} na subseqüência $(J_{j+1}, \ldots, J_{k-1})$. Para retirar J_i, as ligações (J_{i-1}, J_i) (J_i, J_{i+1}), (J_{j-1}, J_j), (J_{l-1}, J_l) e (J_k, J_{k+1}) são eliminadas e substituidas pelas ligações (J_k, J_{i-1}), (J_{j-1}, J_l), (J_{k+1}, J_{l-1}) e (J_j, J_{i+1}). Os caminhos $(J_{k+1}, \ldots, J_{i-1})$ e $(J_j, \ldots, J_{l-1})$ são invertidos. A figura 3.18 ilustra o procedimento.

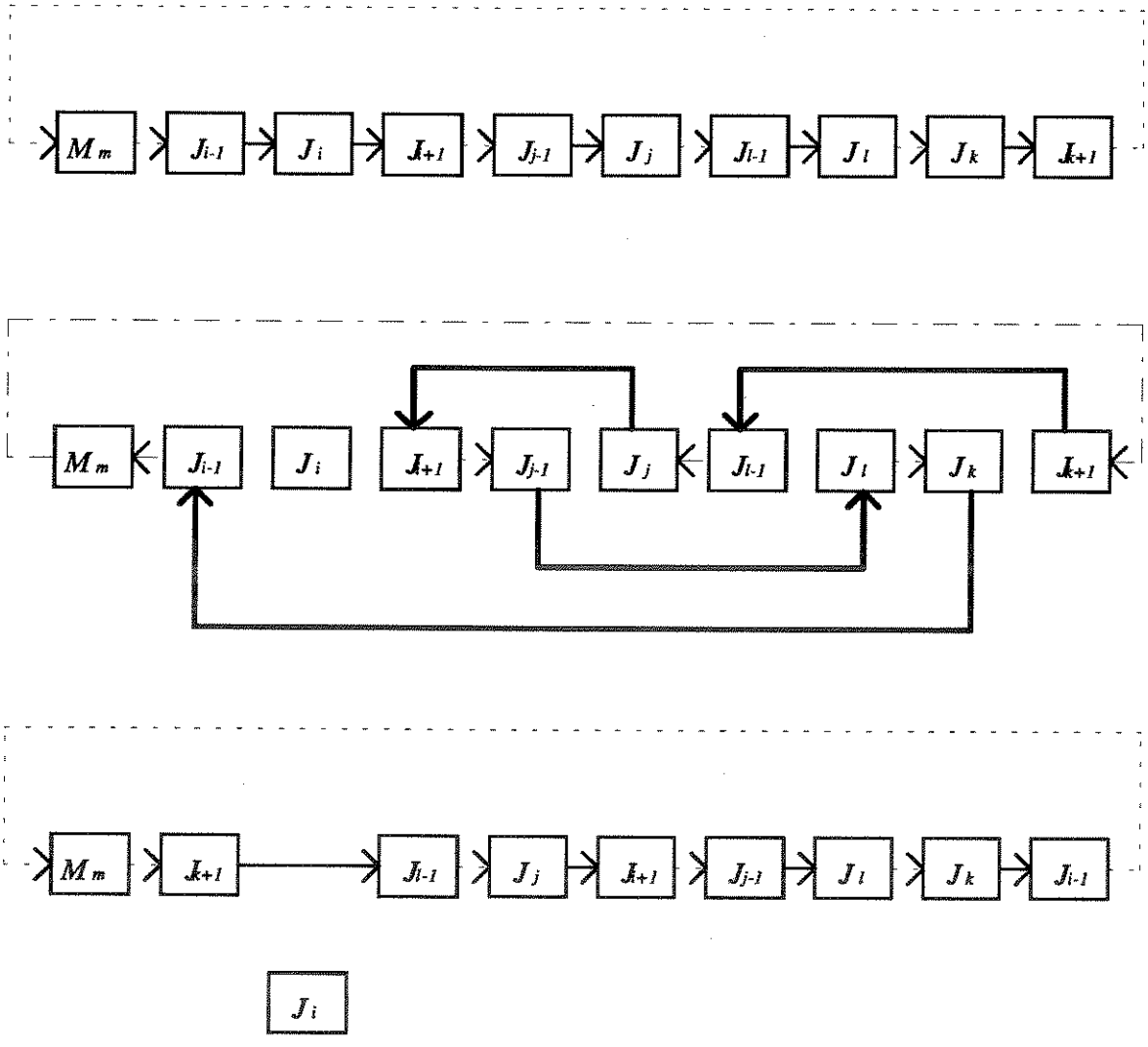


Figura 3.18 - Retirada do Tipo IV

Os comentários feitos para cada tipo de inserção são tambem válidos para as retiradas. Há mais dois tipos de retiradas a considerar.

Retirada do Tipo V (Somente duas tarefas estão alocadas ao processador): Para retirar J_i , dado que só existem duas tarefas alocadas ao processador, simplesmente retiram-se as ligações de J_i com sua tarefa adjacente, que ficará como a única tarefa da máquina. Observe na figura 3.19.

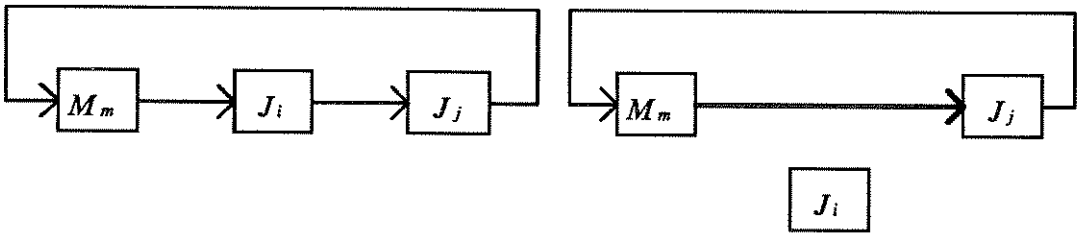


Figura 3.19 - Retirada do Tipo V

Retirada do Tipo VI (Somente uma tarefa está alocada ao processador): Neste caso, simplesmente desfaz-se as ligações desta tarefa com o processador, deixando-o vazio.

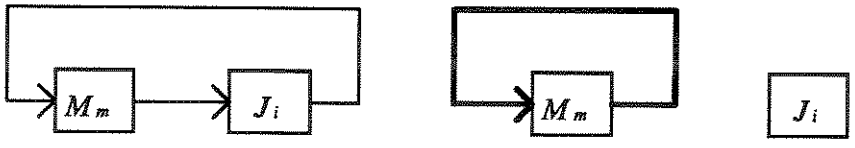


Figura 3.20 - Retirada do Tipo VI

3.3.3. Descrição do Algoritmo e Exemplo Ilustrativo

Os procedimentos de inserção e retirada combinam características dos movimentos *k*-OPT e, ao mesmo tempo, procuram levar em conta o objetivo global do problema (no caso, minimizar o tempo máximo de finalização das tarefas). A escolha de qual tipo de retirada ou inserção que será realizado é feita tomando-se aquele tipo que forneça o menor valor de C_i para M_i , no momento da operação. Então, sempre que se falar de inserção ou retirada, a partir de agora estaremos falando da *melhor* inserção ou da *melhor* retirada

dentre os tipos apresentados, considerando todas as alternativas possíveis de inserção ou retirada, ou seja, todos os possíveis lugares de inserção ou retirada definidos pela vizinhança local são testados.

Conhecidos os procedimentos de inserção e retirada de tarefas é possível passar a descrição do algoritmo proposto. Como já foi observado, o algoritmo é dividido em três fases: *alocação inicial , fase tabu e fase de pós-otimização.* A Fase de Alocação Inicial inicia encontrando a vizinhança global de cada tarefa (dividida em sucessores e predecessores globais, devido à assimetria do problema). Após isto as n tarefas são inseridas nos m processadores de uma maneira gulosa (inserção no processador que cause o menor acréscimo no tempo de finalização) e, a cada inserção, atualiza a vizinhança local (também dividida em predecessores e sucessores).

Após a Fase de Alocação Inicial, passa-se à Fase Tabu, onde o processador mais carregado é identificado. Então, para cada tarefa deste processador, ela verifica as mudanças causadas no tempo máximo de finalização pela retirada e reinserção em outro processador (que deve possuir um número mínimo de vizinhos globais) desta tarefa. Depois disto, ela escolhe o movimento que cause o mínimo tempo máximo de finalização e não seja considerado tabu. Este procedimento é repetido por um número pré-determinado de iterações (neste caso: $n \times 10$) sem melhoria na solução incumbente.

Devido às várias inserções e retiradas feitas nos processadores, ao final da Fase Tabu é possivel ocorrer casos em que retirando e reinserindo uma tarefa no mesmo processador haja uma diminuição no seu tempo de finalização (o que pode levar a uma diminuição no tempo máximo de finalização). Isto é causado devido às características de inserção e retirada, que realizam uma otimização local a cada operação. À realização destas operações de inserção e retirada no mesmo processador dá-se o nome de Fase de Pós-Otimização. As fases deste algoritmo serão detalhadas a seguir. Os parâmetros utilizados no algoritmo serão discutidos posteriormente.

ALOCAÇÃO INICIAL

Passo 1 - Encontre os p predecessores globais e os p sucessores globais de cada tarefa J_i .

Passo 2 - Escolha aleatoriamente m tarefas e insira uma em cada um dos m processadores. Inicialize as listas de vizinhos locais de cada tarefa em cada processador.

Passo 3 - Insira uma tarefa ainda não alocada no processador que produz o menor acréscimo no tempo de finalização atual. Se nao há mais tarefas a alocar, PARE.
Após cada inserção atualize as listas de vizinhos locais.
Guarde a solução inicial encontrada como a solução incumbente (melhor solução encontrada até o momento) e vá para a FASE TABU.

FASE TABU

Suponha que $\overline{M}_i$ é o processador mais carregado (com maior tempo de finalização). Considere:

$C_{M_i J_j}$ = tempo de finalização (tempo necessário para finalizar todas as tarefas) do processador M_i se a tarefa J_j é inserida nele.

$C_{\overline{M}_i J_j}$ = tempo de finalização do processador $\overline{M}_i$ se a tarefa J_j é retirada dele.

Passo 1 - Faça o contador de iterações $t = 0$.

Passo 2 - Para cada tarefa J_j pertencente ao processador mais carregado, encontre:

- o conjunto de processadores candidatos a receber a tarefa J_j, definido como:
  $M(J_j) = \{M_i / M_i \neq \overline{M}_i$ e M_i inclua entre as suas tarefas pelo menos uma sucessora global ou uma predecessora global de $J_j\}$
- o menor tempo de finalização causado pela inserção de J_j em um dos processadores do conjunto $M(J_j)$, definido como:

$$C_{M_i^* J_j} = \min_{M_i \in M(J_j)} \{C_{M_i J_j}\}$$

- o tempo máximo de finalização entre M_i^* e $\overline{M}_i$ se a tarefa J_j fosse transferida de $\overline{M}_i$ para M_i^*, definido como:

$$T_{J_j} = \max \{C_{\overline{M}_i J_j}, C_{M_i^* J_j}\}$$

Passo 3 - Escolha a tarefa J_j^* a ser retirada de $\overline{M}_i$ como sendo $J_j^* = \text{argmin}\{T_{J_j}\}$ e J_j^* não tabu (a menos que o tempo de finalização produzido por este movimento seja melhor que o da solução incumbente - *CRITÉRIO DE ASPIRAÇÃO*).

Então, retire J_j^* de $\overline{M}_i$ e insira em M_i^*.

Proiba o movimento reverso por θ iterações.

Passo 4 - Atualize a lista das q tarefas predecessoras e sucessoras locais afetadas pelas trocas nos processadores $\overline{M}_i$ e M_i^*.

Identifique o processador mais carregado e, se necessário, atualize a solução incumbente.

Passo 5 - Incremente o contador de iterações $t = t + 1$.

Se o número máximo de iterações sem melhoria da solução não foi alcançado, vá para o PASSO 2. Caso contrário vá para a FASE DE PÓS-OTIMIZAÇÃO.

FASE DE PÓS-OTIMIZAÇÃO

Passo 1 - Considere M_i o processador mais carregado na solução incumbente e C_{M_i} o tempo de finalização deste processador.

Passo 2 - Para cada uma das tarefas alocadas a M_i realize uma retirada e uma reinserção no mesmo processador gerando um novo tempo de finalização $\overline{C}_{M_i}$.

Defina $C^*_{M_i} = \underset{J_j \in M_i}{\text{MIN}} \{\overline{C}_{M_i}\}$ como sendo o tempo de finalização mínimo encontrado após as retiradas e reinserções.

Passo 3 - Se $C^*_{M_i} \geq C_{M_i}$, o tempo mínimo de finalização não se altera e PARE.

Se $C^*_{M_i} < C_{M_i}$, mas M_i ainda é o processador mais carregado, atualize a solução incumbente e PARE. Caso contrário, atualize a solução incumbente e vá para o PASSO 1.

Seguem alguns comentários sobre os parâmetros utilizados no algoritmo. O número de vizinhos locais q e globais p é escolhido como:

$$p = \lceil \alpha\sqrt{n} \rceil \text{ e } q = \max\left\{3, \left\lceil \alpha\sqrt{\frac{n}{m}} \right\rceil\right\}$$

onde α é um parâmetro de entrada do algoritmo cuja escolha de valores será discutida no capítulo de resultados computacionais. A escolha da fórmula utilizada para encontrar p e q, procurou levar em conta a dimensão do problema. Como, para se encontrar os p vizinhos globais, todas as tarefas são consideradas, então $\sqrt{n}$ nos fornece um bom número, nem exagerado, nem extremamente pequeno para o número de vizinhos globais, restando ao parâmetro α realizar o ajuste fino deste valor. Na vizinhança local, somente as tarefas alocadas a um determinado processador são consideradas. Portanto, optou-se por trabalhar com o número médio de tarefas por processador, na forma de $\sqrt{\frac{n}{m}}$.

O parâmetro θ da FASE TABU é um número escolhido aleatoriamente no intervalo inteiro [3,7] e usado para proibir a tarefa J_j^* de ser reinserida em $\overline{M_i}$ até a iteração $(t+\theta)$, a menos que o critério de aspiração seja satisfeito. O intervalo [3,7] foi escolhido após vários testes que serão discutidos no capítulo de resultados computacionais.

Para um melhor entendimento do algoritmo proposto, vamos apresentar um exemplo da primeira iteração da Fase Tabu do algoritmo. Utilizaremos a matriz de tempo de processamento apresentada na figura 3.13 e um valor de $\alpha = 0{,}6$ (o que gera um valor para o número de vizinhos globais de 3 e de vizinhos locais também de 3)

A alocação inicial chega a seguinte configuração:

Processador 1 : 0 - 16 - 1 - 6 - 17 - 20 - 19 - 14 - 3 - 5 - 7 - 9
(Carga = 237)
Processador 2 : 0 - 4 - 8 - 2 - 15 - 18 - 12 - 11 - 10 - 13
(Carga = 191)

O processador mais carregado é o processador número 1. Portanto, busca-se em M_1 uma tarefa que será retirada dele e inserida em outro processador (no caso, só existe M_2) de modo a diminuir o tempo máximo de finalização (atualmente com valor de 237). As

retiradas de M_1 e as inserções em M_2 , bem como o valor dos tempos de finalização de cada processador e *makespan* são apresentados na tabela 3.1.

Tarefa	Retirada Tipo	C_1	Inserção Tipo	C_2	*Cmax*
16	3	143	2	199	199
1	1	250	1	239	239
6	1	219	1	234	234
17	2	235	2	233	233
20	2	202	1	247	247
19	2	182	4	218	218
14	1	205	2	194	205
3	2	200	4	208	208
5	2	236	2	172	236
7	2	237	2	140	237
9	1	227	2	166	227

Tabela 3.1 - Valores das Cargas para as Retiradas de M_1 e Inserções em M_2

Como pode-se ver ressaltado na tabela 3.1 a tarefa cuja movimentação proporciona o menor valor de tempo máximo de finalização é a tarefa J_{16} , que movendo-se de M_1 para M_2 diminui este tempo de 237 para 199. Para realização da operação de retirada e inserção de J_{16} é necessário definir sua vizinhança global e local.

Sucessores Globais : { 0 , 18 , 8 }
Predecessores Globais : { 19 , 14 , 15 }

Processador 1 - Sucessores Locais : { 19 , 1 , 14 }
Predecessores Locais : { 0 , 1 , 6 }
Processador 2 - Sucessores Locais : { 15 , 8 , 4 }
Predecessores Locais : { 0 , 18 , 8 }

A retirada de J_{16} de M_1 é do tipo 3 onde : $J_{i-1} = 0$, $J_i = 16$, $J_{i+1} = 1$, $J_{j-1} = 1$, $J_j = 6$, $J_l = 20$, $J_{l+1} = 19$, $J_k = 19$ e $J_{k+1} = 14$. Então, de acordo com o procedimento descrito na figura 3.17, a seqüência resultante, no processador M_1 , é a seguinte:

Processador 1 : 0 - 19 - 1 - 6 - 17 - 20 - 14 - 3 - 5 - 7 - 9 (Carga = 143)

Tem-se como restrições que J_j deve pertencer à vizinhança local de J_{i+1} e J_k deve pertence à vizinhança local de J_{i-1} . Portanto vai-se mostrar a vizinhança local de J_{i+1} e J_{i-1} a fim de verificar se estas restrições estão satisfeitas.

Vizinhança Local da Tarefa $J_{i+1} = 1$
Processador 1 - Sucessores Locais : { 0 , 6 , 19 }
Predecessores Locais : { 19 , 16 , 6 }
Processador 2 - Sucessores Locais : { 0 , 11 , 4 }
Predecessores Locais : { 0 , 2 , 8 }

Vizinhança Local da Tarefa $J_{i-1} = 0$
Processador 1 - Sucessores Locais : { 1 , 6 , 19 }
Predecessores Locais : { 1 , 3 , 5 }
Processador 2 - Sucessores Locais : { 2 , 4 , 8 }
Predecessores Locais : { 2 , 4 , 8 }

Como pode-se ver, $J_j = 6$ e $J_k = 19$ pertencem à vizinhança local de $J_{i+1} = 1$ e de $J_{i-1} = 0$, satisfazendo as restrições impostas pelo tipo de retirada.

Parte-se agora para a análise da inserção de J_{16} em M_2 , que é do tipo 2, onde : $J_{i-1} = 2$, $J_i = 15$, $J_z = 16$, $J_{j-1} = 15$, $J_j = 18$, $J_{k-1} = 8$ e $J_k = 2$. Então, de acordo com o procedimento descrito na figura 3.8, a seqüência resultante, no processador M_2 ,é a seguinte:

Processador 2 : 0 - 4 - 8 - 2 - 15 - 16 - 18 - 12 - 11 - 10 - 13 (Carga = 199)

Observa-se que $J_k \neq J_i$ e $J_k \neq J_j$, como requerido pelo tipo de inserção realizado. A nova solução tem o processador M_2 (tempo máximo de finalização atual igual a 199) como o mais carregado e o procedimento de busca por uma tarefa a ser trocada é iniciado em M_2 do mesmo modo anterior. O procedimento pára quando o número de iterações limite é alcançado.

3.4. ALGORITMO ÓTIMO PARA A SOLUÇÃO DO PROBLEMA *P | tpds | Cmax*

A idéia do esquema ótimo está baseada na equivalência do problema em questão com o problema Minmax *m*-caixeiro viajante (*Minmax m*-TSP), já mostrada no Capítulo 1.

Também foi mostrado no Capítulo 1 a semelhança entre o *Minmax*-TSP e o DVRP(Distance-Constrained Vehicle Routing Problem). Relembrando, o DVRP é semelhante ao *Minsum m*-TSP exceto que neste, cada rota é restrita a não exceder uma determinada constante DIST. Assim, um esquema ótimo pode ser concebido através da resolução de uma série de DVRPs ajustando a restrição sobre as rotas (distância máxima de cada rota) a cada DVRP resolvido. Detalhando a idéia, assuma que C^* é a solução ótima do *P|tpds|Cmax* . Se a solução do DVRP para um dado valor de DIST (distância máxima permitida para a rota mais longa) é factível, então DIST é um limitante superior de C^*; se *S* é a soma de todos os arcos (ligações entre as tarefas, t_{ij}) usados na solução do DVRP, então $\lceil S/m \rceil$ é um limitante inferior de C^* . Se o DVRP é infactível , então DIST+1 é um limitante inferior de C^*.

Partindo destes conceitos desenvolveu-se dois esquemas ótimos para a solução do problema *P|tpds|Cmax* : um baseado em uma *Busca Dicotômica* e outro baseado em um esquema de diminuição da distância máxima permitida de uma unidade por vez. A este último esquema chamaremos de *Busca Subtrativa*.

BUSCA DICOTÔMICA

Passo 1 - Usando a Heurística de Busca Tabu, obtenha um limitante superior UB do valor ótimo da solução do problema *P|tpds|Cmax* .

Passo 2 - Resolva o DVRP associado usando como restrição de máxima distância por rota o limite UB; portanto, DIST = UB.

Passo 3 - Faça UB assumir o valor do tamanho da rota mais longa na solução do DVRP. Sendo *S* a soma do custo de todos os arcos (ligações entre as tarefas, t_{ij}) usados na solução do DVRP, faça LB = $\lceil S/m \rceil$.

Passo 4 - Se UB=LB, a solução ótima do *P|tpds|Cmax* foi obtida e PARE. Caso contrário faça:

$$DIST = LB + \left\lfloor \frac{UB - LB}{2} \right\rfloor$$

Passo 5 - Resolva o DVRP associado usando DIST como restrição de máxima distância por rota.

- Se o problema é FACTÍVEL faça:
  UB = tamanho da rota mais longa na solução do DVRP
  $LB = \lceil S/m \rceil$

- Se o problema é INFACTÍVEL, faça LB = DIST + 1.

Volte ao PASSO 4.

O esquema acima apresentado necessita resolver O(log (UB-LB+1)) DVRPs (SEDEWICK - 1990), onde UB e LB são definidos no PASSO 3. Porém, se a solução heurística obtida no PASSO 1 do esquema não é muito distante de C^* (solução ótima do *P|tpds|Cmax*) um outro esquema pode ser empregado. Este esquema parte do mesmo limitante superior encontrado no PASSO 1 do algoritmo anterior, mas é atualizado pela diminuição de uma unidade a cada DVRP resolvido, até que o problema se torne infactível. O esquema de Busca Subtrativa será mostrado agora.

BUSCA SUBTRATIVA

Passo 1 - Usando a Heurística de Busca Tabu, obtenha um limitante superior UB do valor ótimo da solução do problema *P|tpds|Cmax* . Faça DIST = UB.

Passo 2 - Resolva o DVRP associado usando como restrição de máxima distância por rota DIST = DIST - 1.

Passo 3 - Se o problema é infactível, a solução anterior é a ótima para o *P|tpds|Cmax* e PARE.
Se o problema é factível, vá para o PASSO 2.

Como se pode notar, neste esquema o número de DVRPs resolvido está intimamente ligado à qualidade da solução heurística do PASSO 1. Quanto mais próxima do ótimo ela estiver, menos DVRPs terão que ser resolvidos e mais rapidamente se atingirá uma solução ótima.

O próximo capítulo é dedicado aos resultados computacionais referentes aos testes dos algoritmos propostos para o $P|\ |Cmax$ e para o $P|tpds|Cmax$.

CAPÍTULO 4

Resultados Computacionais

4.1. INTRODUÇÃO

Neste capítulo são apresentados os resultados computacionais obtidos na resolução dos problemas *P*| |*Cmax* e *P*|*tpds*|*Cmax*. Inicialmente, são avaliados a heurística e o esquema exato, propostos no capítulo 2, para o *P*| |*Cmax* e a seguir faz-se o mesmo para o *P*|*tpds*|*Cmax*.

O algoritmo heurístico 3-FASES proposto para o problema *P*| |*Cmax* é comparado com outras três heurísticas clássicas: LPT (*Longest Processing Time first*) desenvolvida por GRAHAM (1969); 0/1 INTERCHANGE de FINN e HOROWITZ (1979) e IMPROVED 0/1 INTERCHANGE de LANGSTON (1982). Todas as soluções heurísticas são também comparadas com as soluções ótimas encontradas pelo procedimento exato de busca dicotômica descrito no capítulo 2.

O algoritmo TABU desenvolvido para o problema *P*|*tpds*|*Cmax* tem três versões diferentes, denominadas TABU ACELERADO, TABU LONGO e TABU AMOSTRAL, que são comparadas com o procedimento ótimo de BUSCA DICOTÔMICA descrito no capítulo 3 e duas destas versões (TABU ACELERADO e TABU AMOSTRAL) são comparadas com o procedimento de BUSCA SUBTRATIVA também descrito no capítulo 3. Comparações também são feitas com uma adaptação do procedimento heurístico de vizinho mais próximo de FREDERICKSON, HECHT e KIM (1978) para encontrar uma solução do *P*|*tpds*|*Cmax*.

A seguir são apresentados, detalhados e discutidos os resultados obtidos para os dois problemas de sequenciamento tratados nesta tese.

4.2. RESULTADOS COMPUTACIONAIS DO PROBLEMA *P* | | *Cmax*

O algoritmo 3-FASES para a solução do *P*| |*Cmax* foi implementado na linguagem C e foi comparado com os algoritmos LPT, 0/1 INTERCHANGE e IMPROVED 0/1 INTERCHANGE, também implementados na linguagem C. Os testes foram realizados numa estação de trabalho SUN Sparc 2.

A heurística LPT foi escolhida para a comparação, pois além de seu bom desempenho numa variada gama de problemas, tem sido usada como heurística de referência para quase todos os novos algoritmos propostos para a solução do *P*| |*Cmax*. O procedimento LPT, inicialmente ordena as tarefas em ordem decrescente de tempo de processamento e depois aloca-as, sucessivamente, ao processador menos carregado (empates são decididos arbitrariamente).

As outras duas heurísticas foram escolhidas por seu bom desempenho computacional e também por pertencerem à classe dos algoritmos baseados em trocas, como o aqui proposto. O procedimento 0/1 INTERCHANGE é fundamentalmente a FASE 2 do algoritmo 3-FASES, porém não divide os tempos de processamento em intervalos e a alocação inicial é feita aleatoriamente.

Já o procedimento IMPROVED 0/1 INTERCHANGE é uma extensão do 0/1 INTERCHANGE e procura evitar alocações iniciais pobres ocasionadas pela distribuição inadequada de tarefas com tempos de processamento grandes, uma vez que a heurística de trocas não consegue movê-las para outros processadores. Assim, uma fase intermediária é incluída, onde selecionam-se as m tarefas com maiores tempos de processamento, alocando-as uma para cada processador.

O algoritmo 3-FASES não foi comparado com o procedimento MULTIFIT de COFFMAN, GAREY e JOHNSON (1978) pois, como observado em FRIESEN (1984), o seu desempenho computacional é comparável à heurística LPT.

As quatro heurísticas foram comparadas umas com as outras e as soluções obtidas foram também comparadas com a solução ótima obtida pelo algoritmo exato de busca dicotômica descrito no capítulo 2.

4.2.1. Geração dos Conjuntos de Teste

Para se avaliar o desempenho do novo procedimento em relação aos demais e em relação ao ótimo, forem gerados problemas testes com m = 5, 10, 25 e n = 10, 50, 100, 500 e 1000 e tempos de processamento no intervalo $[\underline{p}, \overline{p}]$ que assume os valores de: [1,100], [1,1000] e [1,10000]. Para cada combinação dos valores acima foram gerados 10 problemas de acordo com as seguintes distribuições (HASTINGS e PEACOCK - 1974):

- DISTRIBUIÇÃO UNIFORME: nos intervalos $[\underline{p}, \overline{p}]$;

- DISTRIBUIÇÃO EXPONENCIAL:

$$0{,}632\frac{(\underline{p}+\overline{p})}{2}\log R$$

onde R é uma variável retangular unitária, isto é, tem uma distribuição uniforme contínua no intervalo [0,1]. Os valores fora do intervalo $[\underline{p}, \overline{p}]$ são desprezados;

- DISTRIBUIÇÃO NORMAL:

$$\left(\sum_{i=1}^{12} R_i - 6\right)\times(\overline{p}-\underline{p})+\underline{p}$$

onde R é uma variável retangular unitária. Os valores fora do intervalo $[\underline{p}, \overline{p}]$ são desprezados.

Também foram gerados problemas teste com m = 5, 10, 15, 20 e 25 de acordo com uma distribuição chamada DEGRAU, sugerida por GRAHAM (1969). É uma distribuição de pior caso para o algoritmo LPT. A distribuição degrau obedece a seguinte definição:

n = $2m + 1$
Para i = 1 até $2m$
$p_i = 2m - ((i+1)\text{DIV}2)$
$p_n = m$

No total foram gerados 1175 problemas distintos.

4.2.2. Escolha do Parâmetro *r*

O parâmetro *r* representa o número de intervalos nos quais são divididos os tempos de processamento das tarefas. Um pequeno valor de *r* tende a produzir uma boa solução na FASE 1 com poucos passos esperados nas FASES 2 e 3. Entretanto, a FASE 3 deve ser dispendiosa computacionalmente, pois existirão muitas duplas trocas para serem testadas. Por outro lado, para um valor grande de *r* resultam soluções iniciais não tão boas, mas as fases de melhoramento serão mais velozes, permitindo um refinamento maior da solução. Faz-se então necessário uma análise computacional para indicar a melhor escolha de compromisso para *r*.

Testes com *r* variando de 1 a 20 indicaram que os *"melhores"* valores de *r* estão relacionados com a razão (*n/m*). Chegou-se a uma tabela de valores recomendados para *r* (r^*) (tabela 4.1) que, geralmente, produzem valores de soluções bom a tempos computacionais factíveis.

n/m	r^*
(1,10)	2
[10,50)	5
[50,100)	10
[100,200)	15
[200,100)	20

Tabela 4.1 - Valores Recomendados de *r*

Todos os problemas foram rodados usando $r = r^*$ e limitados em 1000 segundos de CPU por problema. Exemplos fora desta especificação são considerados sem sucesso e descartados da análise. Os resultados obtidos são uma média de 10 problemas testados, a não ser quando o número de problemas resolvidos na otimalidade tenha sido menor que 10, quando então este número vem indicado entre parênteses, com as médias sendo tomadas em relação a ele.

As variáveis usadas nas tabelas 4.2 a 4.11 são definidas do seguinte modo:

- *m* : número de processadores;
- *n* : número de tarefas;

- $Z : \left(\frac{Z' - Z^*}{Z^*}\right) \times 100$;
- $Z_1 : \left(\frac{Z_1' - Z^*}{Z^*}\right) \times 100$;
- $Z_2 : \left(\frac{Z_2' - Z^*}{Z^*}\right) \times 100$;
- $Z_3 : : \left(\frac{Z_3' - Z^*}{Z^*}\right) \times 100$;

onde: Z^* : valor ótimo global;
Z' : melhor solução encontrada pelo algoritmo 3-FASES;
Z_1' : melhor solução encontrada pelo algoritmo 0/1 INTERCHANGE;
Z_2' : melhor solução encontrada pelo algoritmo IMPROVED 0/1 INTERCHANGE;
Z_3' : melhor solução encontrada pelo algoritmo LPT;

- tZ^*: tempo de CPU (em segundos) consumido pelo algoritmo ótimo;
- tZ : tempo de CPU (em segundos) consumido pelo algoritmo 3-FASES;
- tZ_1 : tempo de CPU (em segundos) consumido pelo algoritmo 0/1 INTERCHANGE;
- tZ_2 : tempo de CPU (em segundos) consumido pelo algoritmo IMPROVED 0/1 INTERCHANGE;
- tZ_3 : tempo de CPU (em segundos) consumido pelo algoritmo LPT.

Nas tabelas 4.2 a 4.11, a primeira coluna indica o número de processadores; a segunda o número de tarefas; a terceira, quarta, quinta e sexta colunas representam a diferença percentual em relação ao ótimo das soluções encontradas pelos algoritmos 3-FASES, 0/1 INTERCHANGE, IMPROVED 0/1 INTERCHANGE e LPT, respectivamente. Se não foi encontrada solução ótima para nenhum dos 10 problemas testados, a quarta, quinta e sexta colunas representam a diferença percentual entre a solução encontrada pelos algoritmos 0/1 INTERCHANGE, IMPROVED 0/1 INTERCHANGE e LPT e a solução encontrada pelo algoritmo 3-FASES. As colunas restantes representam o tempo computacional consumido pelos algoritmos ótimo, 3-FASES, 0/1 INTERCHANGE, IMPROVED 0/1 INTERCHANGE e LPT, respectivamente.

m	n	Z	Z_1	Z_2	Z_3	tZ^*	tZ	tZ_1	tZ_2	tZ_3
5	10	1,8	4,5	4,5	1,8	0,003	0,002	0,032	0,023	0,032
	50	0,0	0,8	0,8	0,5	0,003	0,002	0,032	0,032	0,035
	100	0,0	0,1	0,1	0,1	0,008	0,007	0,030	0,038	0,030
	500	0,0	0,0	0,0	0,0	0,024	0,023	0,112	0,125	0,063
	1000	0,0	0,0	0,0	0,0	0,063	0,062	0,215	0,217	0,152
10	50	0,3	2,8	2,8	1,6	0,013	0,007	0,035	0,032	0,033
	100	0,0	0,7	0,7	0,4	0,008	0,007	0,038	0,042	0,032
	500	0,0	0,0	0,0	0,0	0,014	0,013	0,137	0,132	0,057
	1000	0,0	0,0	0,0	0,0	0,034	0,033	0,295	0,298	0,102
25	50	1,1	9,6	9,6	1,4	0,017	0,013	0,032	0,038	0,037
	100	0,2	4,4	4,4	2,4	0,023	0,010	0,042	0,045	0,037
	500	0,0	0,1	0,1	0,1	0,026	0,025	0,160	0,157	0,055
	1000	0,0	0,0	0,0	0,0	0,041	0,040	0,425	0,422	0,083

Tabela 4.2 - Resultados da Distribuição Uniforme - Intervalo [1,100]

m	n	Z	Z_1	Z_2	Z_3	tZ^*	tZ	tZ_1	tZ_2	tZ_3
5	10	1,0	4,8	4,8	0,9	0,002	0,001	0,030	0,033	0,033
	50	0,1	0,5	0,5	0,3	0,092	0,005	0,038	0,033	0,037
	100	0,0	0,2	0,2	0,1	0,022	0,005	0,043	0,037	0,035
	500	0,0	0,0	0,0	0,0	0,021	0,020	0,170	0,165	0,067
	1000	0,0	0,0	0,0	0,0	0,046	0,045	0,527	0,528	0,150
10	50	0,2	2,2	2,2	1,5	3,962	0,002	0,035	0,037	0,033
	100	$0,0^{(7)}$	$0,7^{(7)}$	$0,7^{(7)}$	$0,4^{(7)}$	$3,524^{(7)}$	0,010	0,040	0,037	0,033
	500	0,0	0,0	0,0	0,0	0,037	0,020	0,178	0,180	0,052
	1000	0,0	0,0	0,0	0,0	0,040	0,038	0,567	0,562	0,100
25	50	1,1	8,6	8,6	1,1	0,017	0,012	0,035	0,033	0,037
	100	$0,3^{(3)}$	$5,5^{(3)}$	$5,5^{(3)}$	$3,4^{(3)}$	$11,43^{(3)}$	0,015	0,038	0,037	0,032
	500	0,0	0,2	0,2	0,1	0,187	0,035	0,168	0,173	0,050
	1000	0,0	0,1	0,1	0,0	0,064	0,063	0,565	0,567	0,087

Tabela 4.3 - Resultados da Distribuição Uniforme - Intervalo [1,1000]

m	n	Z	Z_1	Z_2	Z_3	tZ^*	tZ	tZ_1	tZ_2	tZ_3
5	10	1,3	5,0	5,0	1,6	0,005	0,001	0,033	0,033	0,030
	50	0,1(7)	0,5(7)	0,5(7)	0,5(7)	111,4(7)	0,002	0,043	0,035	0,040
	100	0,0(8)	0,2(8)	0,2(8)	0,2(8)	22,33(8)	0,012	0,047	0,037	0,037
	500	0,0	0,0	0,0	0,0	0,912	0,035	0,188	0,190	0,067
	1000	0,0	0,0	0,0	0,0	0,053	0,052	0,627	0,628	0,150
10	50	0,4(5)	2,6(5)	2,6(5)	2,0(5)	239,7(5)	0,005	0,028	0,032	0,038
	100	0,0(1)	1,0(1)	1,0(1)	0,5(1)	2,333(1)	0,008	0,042	0,035	0,035
	500	0,0(7)	0,0(7)	0,0(7)	0,0(7)	1,238(7)	0,040	0,178	0,175	0,055
	1000	0,0(9)	0,0(9)	0,0(9)	0,0(9)	0,066(9)	0,065	0,627	0,635	0,105
25	50	0,4	5,2	5,2	0,8	0,010	0,007	0,030	0,035	0,043
	100	1,1(1)	6,0(1)	6,0(1)	4,5(1)	0,100(1)	0,017	0,040	0,038	0,040
	500	- (0)	0,2	0,2	0,1	- (0)	0,057	0,177	0,182	0,050
	1000	0,0(4)	0,1(4)	0,1(4)	0,0(4)	0,142(4)	0,137	0,587	0,592	0,090

Tabela 4.4 - Resultados da Distribuição Uniforme - Intervalo [1,10000]

m	n	Z	Z_1	Z_2	Z_3	tZ^*	tZ	tZ_1	tZ_2	tZ_3
5	10	2,3	10,6	10,6	3,9	0,002	0,001	0,032	0,030	0,038
	50	0,0	1,0	1,0	0,9	0,005	0,003	0,030	0,032	0,035
	100	0,0	0,2	0,2	0,1	0,004	0,003	0,042	0,040	0,038
	500	0,0	0,0	0,0	0,0	0,016	0,015	0,105	0,107	0,058
	1000	0,0	0,0	0,0	0,0	0,048	0,047	0,203	0,205	0,152
10	50	0,1	3,9	3,9	3,1	0,008	0,007	0,038	0,035	0,033
	100	0,0	1,3	1,3	0,8	0,008	0,007	0,038	0,035	0,038
	500	0,0	0,1	0,1	0,1	0,014	0,013	0,123	0,127	0,048
	1000	0,0	0,0	0,0	0,0	0,048	0,047	0,293	0,293	0,102
25	50	1,5	11,9	11,9	1,8	0,017	0,015	0,028	0,037	0,038
	100	0,3	6,8	6,8	4,4	0,027	0,010	0,035	0,045	0,030
	500	0,0	0,4	0,4	0,2	0,021	0,020	0,152	0,153	0,045
	1000	0,0	0,1	0,1	0,1	0,048	0,047	0,433	0,435	0,082

Tabela 4.5 - Resultados da Distribuição Normal - Intervalo [1,100]

m	n	Z	Z_1	Z_2	Z_3	tZ^*	tZ	tZ_1	tZ_2	tZ_3
5	10	1,2	6,1	6,1	1,5	0,002	0,001	0,030	0,032	0,033
	50	0,1(7)	1,1(7)	1,1(7)	0,7(7)	0,050(7)	0,002	0,033	0,033	0,030
	100	0,0(8)	0,4(8)	0,4(8)	0,3(8)	22,02(8)	0,007	0,038	0,043	0,028
	500	0,0	0,0	0,0	0,0	0,019	0,018	0,173	0,177	0,067
	1000	0,0	0,0	0,0	0,0	0,056	0,055	0,527	0,523	0,148
10	50	0,3(6)	3,2(6)	3,2(6)	2,6(6)	13,33(6)	0,002	0,035	0,033	0,032
	100	0,0(2)	0,6(2)	0,6(2)	0,4(2)	0,450(2)	0,005	0,048	0,037	0,035
	500	0,0	0,1	0,1	0,1	0,023	0,022	0,168	0,172	0,053
	1000	0,0	0,0	0,0	0,0	0,044	0,043	0,555	0,558	0,100
25	50	0,4	12,9	12,9	1,0	0,017	0,010	0,033	0,038	0,033
	100	- (0)	6,6	6,6	3,9	- (0)	0,017	0,037	0,047	0,028
	500	0,0	0,4	0,4	0,3	0,036	0,035	0,167	0,167	0,057
	1000	0,0	0,1	0,1	0,1	0,058	0,057	0,560	0,562	0,090

Tabela 4.6 - Resultados da Distribuição Normal - Intervalo [1,1000]

m	n	Z	Z_1	Z_2	Z_3	tZ^*	tZ	tZ_1	tZ_2	tZ_3
5	10	1,4	6,4	6,4	2,8	0,003	0,002	0,032	0,035	0,037
	50	0,1(4)	1,2(4)	1,2(4)	1,0(4)	2,133(4)	0,007	0,033	0,038	0,028
	100	0,0(4)	0,3(4)	0,3(4)	0,2(4)	62,41(4)	0,008	0,040	0,037	0,040
	500	0,0	0,0	0,0	0,0	0,083	0,042	0,192	0,200	0,062
	1000	0,0	0,0	0,0	0,0	0,062	0,060	0,647	0,643	0,152
10	50	0,3(5)	3,8(5)	3,8(5)	2,6(5)	23,40(5)	0,008	0,032	0,032	0,035
	100	- (0)	1,2	1,2	0,6	- (0)	0,007	0,045	0,040	0,035
	500	0,0(7)	0,1(7)	0,1(7)	0,1(7)	0,048(7)	0,047	0,188	0,183	0,053
	1000	0,0	0,0	0,0	0,0	0,071	0,070	0,622	0,612	0,107
25	50	0,9	13,9	13,9	1,9	0,015	0,008	0,035	0,037	0,035
	100	- (0)	6,3	6,3	3,7	- (0)	0,020	0,035	0,042	0,030
	500	- (0)	0,4	0,4	0,3	- (0)	0,063	0,177	0,177	0,053
	1000	0,0(8)	0,1(8)	0,1(8)	0,1(8)	0,138(8)	0,137	0,605	0,605	0,090

Tabela 4.7 - Resultados da Distribuição Normal - Intervalo [1,10000]

m	n	Z	Z_1	Z_2	Z_3	tZ^*	tZ	tZ_1	tZ_2	tZ_3
5	10	0,9	1,5	1,5	0,8	0,002	0,001	0,030	0,032	0,032
	50	0,0	0,5	0,5	0,3	0,002	0,001	0,037	0,035	0,032
	100	0,0	0,1	0,1	0,1	0,007	0,005	0,038	0,037	0,040
	500	0,0	0,0	0,0	0,0	0,019	0,018	0,108	0,108	0,065
	1000	0,0	0,0	0,0	0,0	0,047	0,045	0,213	0,213	0,153
10	50	0,5	2,4	2,4	1,4	0,005	0,003	0,037	0,030	0,035
	100	0,1	0,4	0,4	0,2	0,010	0,003	0,043	0,040	0,035
	500	0,0	0,0	0,0	0,0	0,018	0,017	0,145	0,132	0,057
	1000	0,0	0,0	0,0	0,0	0,038	0,037	0,308	0,312	0,098
25	50	0,0	0,0	0,0	0,0	0,004	0,003	0,028	0,037	0,033
	100	0,6	3,6	3,6	1,6	0,010	0,008	0,040	0,037	0,035
	500	0,0	0,1	0,1	0,1	0,018	0,017	0,177	0,172	0,058
	1000	0,0	0,0	0,0	0,0	0,044	0,043	0,475	0,478	0,085

Tabela 4.8 - Resultados da Distribuição Exponencial - Intervalo [1,100]

m	n	Z	Z_1	Z_2	Z_3	tZ^*	tZ	tZ_1	tZ_2	tZ_3
5	10	0,0	0,3	0,3	0,0	0,002	0,001	0,037	0,035	0,030
	50	0,1(9)	0,6(9)	0,6(9)	0,4(9)	0,156(9)	0,002	0,035	0,028	0,035
	100	0,0	0,1	0,1	0,1	0,015	0,007	0,048	0,038	0,032
	500	0,0	0,0	0,0	0,0	0,019	0,018	0,210	0,203	0,067
	1000	0,0	0,0	0,0	0,0	0,051	0,050	0,660	0,657	0,147
10	50	0,6	2,3	2,3	1,6	40,26	0,008	0,033	0,040	0,040
	100	0,1(8)	0,4(8)	0,4(8)	0,3(8)	0,421(8)	0,008	0,038	0,042	0,033
	500	0,0	0,0	0,0	0,0	0,023	0,022	0,218	0,208	0,053
	1000	0,0	0,0	0,0	0,0	0,036	0,035	0,667	0,672	0,102
25	50	0,0	0,0	0,0	0,0	0,011	0,010	0,037	0,032	0,028
	100	0,6(7)	3,9(7)	3,9(7)	2,3(7)	0,500(7)	0,017	0,042	0,042	0,040
	500	0,0	0,1	0,1	0,1	0,110	0,033	0,210	0,217	0,050
	1000	0,0	0,0	0,0	0,0	0,056	0,055	0,703	0,698	0,088

Tabela 4.9 - Resultados da Distribuição Exponencial - Intervalo [1,1000]

m	n	Z	Z_1	Z_2	Z_3	tZ^*	tZ	tZ_1	tZ_2	tZ_3
5	10	0,0	0,5	0,5	0,2	0,002	0,001	0,035	0,037	0,032
	50	0,1(9)	0,5(9)	0,5(9)	0,4(9)	7,298(9)	0,005	0,033	0,035	0,031
	100	0,0(8)	0,1(8)	0,1(8)	0,1(8)	8,175(8)	0,012	0,040	0,045	0,037
	500	0,0	0,0	0,0	0,0	0,031	0,030	0,222	0,235	0,063
	1000	0,0	0,0	0,0	0,0	0,068	0,067	0,827	0,810	0,148
10	50	0,8(7)	1,7(7)	1,7(7)	1,0(7)	0,176(7)	0,007	0,035	0,037	0,030
	100	0,0(4)	0,4(4)	0,4(4)	0,2(4)	20,64(4)	0,013	0,042	0,047	0,033
	500	0,0(7)	0,0(7)	0,0(7)	0,0(7)	0,038(7)	0,042	0,247	0,237	0,055
	1000	0,0	0,0	0,0	0,0	0,198	0,070	0,777	0,782	0,107
25	50	0,0	0,0	0,0	0,0	0,008	0,007	0,033	0,035	0,040
	100	0,7(6)	3,9(6)	3,9(6)	1,9(6)	0,136(6)	0,015	0,042	0,042	0,035
	500	- (0)	0,1	0,1	0,1	- (0)	0,065	0,215	0,210	0,053
	1000	0,0(6)	0,0(6)	0,0(6)	0,0(6)	18,48(6)	0,127	0,738	0,738	0,085

Tabela 4.10 - Resultados da Distribuição Exponencial - Intervalo [1,10000]

m	n	Z	Z_1	Z_2	Z_3	tZ^*	tZ	tZ_1	tZ_2	tZ_3
5	11	0,0	20,0	20,0	26,7	0,002	0,001	0,033	0,050	0,033
10	21	0,0	16,7	16,7	30,0	0,002	0,001	0,017	0,033	0,017
15	31	0,0	17,8	17,8	31,1	0,002	0,001	0,017	0,033	0,033
20	41	0,0	16,7	16,7	31,7	0,002	0,001	0,033	0,033	0,017
25	51	0,0	17,3	17,3	32,0	0,002	0,001	0,033	0,017	0,050

Tabela 4.11 - Resultados da Distribuição Degrau

Tabela	Distribuição	$[\underline{p}, \overline{p}]$
4.2	uniforme	[1,100]
4.3		[1,1000]
4.4		[1,10000]
4.5	exponencial	[1,100]
4.6		[1,1000]
4.7		[1,10000]
4.8	normal	[1,100]
4.9		[1,1000]
4.10		[1,10000]
4.11	degrau	[m,2m]

Tabela 4.12 - Tabelas e Distribuições para o $P|\ |Cmax$

No total foram testados 1175 problemas e o algoritmo ótimo encontrou solução para 999 deles, sendo que o algoritmo 3-FASES encontrou a solução ótima para 68,6 % deles. Para os problemas em que o algoritmo 3-FASES não encontrou a solução ótima, 22,3 % estavam abaixo de 1% da mesma; 8,1% entre 1 e 2% e apenas 1% entre 2 e 3%.

Heurística	Sol. Ótima	< 1%	1% a 2%	2% a 3%	> 3%
3-FASES	68,6	22,3	8,1	1,0	0,0
0/1 INTER	39,0	32,3	4,9	4,5	19,3
IMP 0/1 INT	39,0	32,3	4,9	4,5	19,3
LPT	41,4	35,7	14,7	4,3	3,9

Tabela 4.13 - Distribuição das Soluções Heurísticas em Relação ao Ótimo

Como observado na tabela 4.13 o algoritmo 3-FASES apresentou um comportamento superior às heurísticas a que foi comparado, superando ou igualando em todos os casos as heurísticas 0/1 INTERCHANGE, IMPROVED 0/1 INTERCHANGE e LPT, com exceção de dois casos onde a heurística LPT foi 0,1 % superior a 3-FASES (veja tabelas 4.3 e 4.8).

Em termos de qualidade de solução, é conclusivo que a heurística 3-FASES produz soluções bastante boas e robustas, pois mesmo para distribuições reconhecidamente difíceis para os outros algoritmos (distribuição degrau - tabela 4.11) o algoritmo 3-FASES encontrou a solução ótima para todos os problemas, enquanto que as outras heurísticas ficaram, no mínimo, à 16,7 % do valor ótimo.

Quanto à eficiência, o algoritmo 3-FASES também foi mais rápido que os outros três, geralmente em uma ordem de magnitude. É importante ressaltar aqui que a heurística LPT necessita de uma fase de ordenação que domina o tempo computacional enquanto que o algoritmo 3-FASES, 0/1 INTERCHANGE e IMPROVED 0/1 INTERCHANGE não necessitam nenhuma ordenação preliminar dos tempos de processamento.

Nota-se tambem que quando os tempos de processamento $p_j \in [1,100]$, o tempo de execução do algoritmo ótimo é comparável ao do algoritmo 3-FASES, porém esta vantagem desaparece para os outros dois intervalos usados nas comparações, pois a resolução do problema de empacotamento se torna mais complexa, o que resulta em alguns tempo extremamente grandes em relação aos outros (isto ocorre quando o algoritmo ótimo para o PE tem que explorar um grande número de nós da árvore de busca, ou seja, a qualidade dos limitantes superior e inferior não é tão boa). Outro ponto importante a comentar é o comportamento do tempo computacional do algoritmo 3-FASES, pelas tabelas 4.2 a 4.12, pode-se notar que o tempo computacional apresenta um crescimento abaixo do linear com o crescimento do tamanho do problema (n).

Os resultados do algoritmo 3-FASES foram publicados em FRANÇA e MÜLLER (1992) e FRANÇA et al. (1993a).

4.3. RESULTADOS COMPUTACIONAIS DO PROBLEMA *P | tpds | Cmax*

Os testes computacionais enfocando a solução do *P|tpds|Cmax* foram feitos com três variações do algoritmo proposto no capítulo 3: TABU ACELERADO, TABU LONGO e TABU AMOSTRAL.

As versões TABU ACELERADO e TABU LONGO estão baseadas na escolha do valor do parâmetro α que definirá o número de vizinhos locais e globais a ser considerado. Sabe-se que quanto maior o número de vizinhos, mais dispendiosa computacionalmente se torna a busca pelo melhor movimento a ser realizado. A versão TABU AMOSTRAL está baseada na escolha de um grupo de diferentes soluções iniciais às quais são aplicadas um número finito de passos do algoritmo, reinicializando-se o mesmo com a melhor solução parcial dentre todas as encontradas, e retornando ao algoritmo até o seu final normal. A versão embutida dentro de TABU AMOSTRAL é a versão TABU LONGO do algoritmo.

Antes da apresentação dos resultados computacionais, deve-se mostrar como foram feitos os testes para a escolha do parâmetro α, que serve para fazer o ajuste fino dos valores p e q que representam o número de vizinhos globais e locais, respectivamente. Este ajuste fino faz com que p e q, dependentes do tamanho do problema, assumam valores que gerem um bom compromisso entre qualidade de solução e tempo computacional.

Foram gerados 10 problemas para cada combinação de n = 25, 50, 100 e 200 e $m = n/5$, $n/10$ e $n/25$. Pode-se ver na tabela 4.14 os valores que p e q assumem quando α varia de 0,1 a 2,0. A geração dos problemas é feita de uma maneira que chamaremos de *estruturada*. Neste modo, primeiro geram-se aleatóriamente $2(n + 1)$ pontos P_0, P_1, ...,P_n , Q_0 ,Q_1 ,...., Q_n dentro do quadrado [0,100]. Para $i < j$, define-se t_{ij} como sendo a distância Euclideana truncada entre P_i e P_j ; para $i > j$, t_{ij} é igual a distância Euclideana truncada entre Q_i e Q_j . Com isto constrói-se a matriz assimétrica de tempos de processamento dependentes da seqüência.

α	p			q				
	n=25	n=100	n=200	n/m=25/2	n/m=25/3	n/m=5	n/m=25	n/m=10
0,1	1	1	2	1	1	1	1	1
0,2	1	2	3	1	1	1	1	1
0,3	2	3	5	2	1	1	2	1
0,4	2	4	6	2	2	1	2	2
0,5	3	5	8	2	2	2	3	2
0,6	3	6	9	3	2	2	3	2
0,7	4	7	10	3	3	2	4	3
0,8	4	8	12	3	3	2	4	3
0,9	5	9	13	4	3	3	5	3
1,0	5	10	15	4	3	3	5	4
1,1	6	11	16	4	4	3	6	4
1,2	6	12	17	5	4	3	6	4
1,3	7	13	19	5	4	3	7	5
1,4	7	14	20	5	5	4	7	5
1,5	8	15	22	6	5	4	8	5
1,6	8	16	23	6	5	4	8	6
1,7	9	17	25	7	5	4	9	6
1,8	9	18	26	7	6	5	9	6
1,9	10	19	27	7	6	5	10	6
2,0	10	20	29	8	6	5	10	7

Tabela 4.14 - Valores de $p = \lceil \alpha\sqrt{n} \rceil$ e $q = \left\lceil \alpha\sqrt{n/m} \right\rceil$ em função de α

Testes foram feitos com α variando de 0,1 a 2,0 e foram construidas várias tabelas nas quais foi analisado o comportamento do valor da solução encontrada pela heurística em relação ao parâmetro α. Um exemplo deste estudo pode ser visto na tabela 4.15 para $\alpha = 0{,}6$ e $q \geq 3$ onde podem ser vistos os resultados de uma média de 10 problemas testados para cada combinação. Estes testes mostraram que quanto maior α, isto é, quanto maior a vizinhança considerada, melhor a qualidade da solução, porém os tempos computacionais tendem a se tornar elevados. Portanto, o que se procurou foi, novamente, um valor de compromisso para α que fornecesse boas soluções com tempos computacionais suportáveis.

Daí surgiram duas versões do algoritmo, a primeira, TABU ACELERADO, usa um valor fixo de $\alpha = 0,6$ e $q \geq 3$, de modo a produzir bons resultados com baixo esforço computacional. Na versão TABU LONGO buscam-se soluções melhores, em detrimento do tempo computacional, escolhendo quatro valores de $\alpha = 0,6$; 0,8 ; 1,5 e 2,0 e $q \geq 3$, para os quais o algoritmo é executado e a melhor solução encontrada é a escolhida.

n	m	Solução Inicial	Solução Tabu	Solução Pós-Otim.	Tempo de CPU(seg)
25	2	245,0	180,4	180,3	14,3
	3	167,5	118,4	117,7	9,5
	5	115,4	81,3	81,3	8,7
50	2	424,8	254,0	254,0	95,0
	5	170,7	110,3	110,2	43,3
	10	111,2	68,0	68,0	19,5
100	4	368,2	197,4	197,4	458,5
	10	153,0	88,5	88,5	122,9
	20	107,1	56,9	56,9	60,2
200	8	303,3	154,7	154,7	1560,0
	20	128,0	71,2	71,2	389,3
	40	101,9	50,1	50,1	171,1

Tabela 4.15 - Resultados do Algoritmo para $\alpha = 0,6$ e $q \geq 3$.

Outro parâmetro do algoritmo que necessita de ajustes é θ, o número de iterações para as quais o movimento fica declarado tabu (proibido). Foi implementado o conceito de usar um valor aleatório para θ e não uma constante. Para isso, consideramos uma variável aleatória distribuida uniformemente em um intervalo $[\underline{\theta}\ ,\ \overline{\theta}]$. Os valores de $\underline{\theta}$ e $\overline{\theta}$ foram escolhidos de modo a proporcionar um balanço equilibrado entre constranger a busca (valores muito altos de θ) e liberá-la (valores muito pequenos de θ). Por isso, foram testados vários intervalos dentro dos limites máximos de [3 , 20] e concluiu-se que a concentração maior de boas soluções se dava com $\underline{\theta} = 3$ e $\overline{\theta} = 7$, que foi o valor utilizado pelo algoritmo.

Nota-se que estes valores estão em conformidade com os valores de intervalos propostos por GLOVER e LAGUNA (1992a) para regras dinâmicas simples de tabu.

Também, no sentido de reduzir a quantidade de espaço computacional necessário, nenhuma lista tabu é mantida na implementação. Ao invés disto, usa-se uma etiqueta associada a cada tarefa, de valor θ para indicar por quantas iterações um dado movimento é declarado tabu.

Finalizando o estudo dos parâmetros do algoritmo, deve-se definir o número máximo de iterações sem melhoria, que é usado como critério de parada para a busca tabu. Utilizou-se o conceito proposto por CRAINIC, GENDREAU, SORIANO e TOULOUSE (1991) onde se deve definir este número em função do tamanho do problema. Testes mostraram que fazer este número igual a $10n$ proporciona um bom compromisso entre qualidade de solução e esforço computacional.

Para efeito de comparação, foi implementado também o procedimento de vizinho mais próximo de FREDERICKSON, HETCH e KIM (1978) adaptado para resolver o $P|tpds|Cmax$. O procedimento, inicialmente, aloca aleatoriamente m tarefas aos processadores e vai, sucessivamente, alocando ao processador menos carregado aquela tarefa que trás o menor acréscimo no tempo de execução.

Resumindo, há três versões para a heurística proposta e uma heurística clássica, TABU ACELERADO, TABU LONGO, TABU AMOSTRAL e vizinho mais próximo. Como o objetivo é comparar o desempenho delas, usando-se o algoritmo ótimo proposto no capítulo 3, fez-se necessário gerar um novo conjunto de problemas teste, onde o limite máximo de tarefas foi fixado em 50, devido as limitações impostas pelo método ótimo.

Foram gerados 10 problemas para cada uma das combinações de $m = 2, 3, 4$ e 5 e $n = 20, 30, 40$ e 50. Uma série foi gerada da maneira estruturada, como já mencionado, e outra série de uma maneira que denominou-se *não-estruturada*, pois não obedece a desigualdade do triângulo, como no caso estruturado. O que se faz é gerar cada t_{ij} como sendo um número aleatório uniformemente distribuido no intervalo [1 , 100].

O algoritmo ótimo foi implementado na linguagem FORTRAN 77 e o procedimento ótimo de resolução do DVRP é o proposto por LAPORTE, NOBERT e TAILLEFER

(1987). As versões heurísticas foram implementadas na linguagem C. Todos os testes foram realizados numa estação SUN Sparc 2.

Foram feitas as seguintes comparações entre os procedimentos heurísticos e ótimo, para cada uma das séries, estruturada e não-estruturada:

- VMP / BD Tabelas 4.16 e 4.22;
- ACEL / BD Tabelas 4.17 e 4.23;
- LONG / BD Tabelas 4.18 e 4.24;
- AMOST / BS Tabelas 4.19 e 4.25;
- AMOST / BS Tabelas 4.20 e 4.26;
- ACEL / BS Tabelas 4.21 e 4.27.

onde: VMP : algoritmo de vizinho mais próximo de FREDERICKSON et al. (1978);
ACEL : algoritmo TABU ACELERADO;
LONG : algoritmo TABU LONGO;
AMOST : algoritmo TABU AMOSTRAL;
BD : algoritmo ótimo de BUSCA DICOTÔMICA;
BS : algoritmo ótimo de BUSCA SUBTRATIVA.

O cabeçalho das tabelas 4.16 a 4.27 usam a seguinte legenda:

- n : número de tarefas;
- m : número de processadores;
- $\overline{Z}/Z^*$: valor da solução heurística ÷ valor da solução ótima;
- Z/Z^* : limite inferior fornecido pela primeira solução do DVRP ÷ valor da solução ótima;
- #DVRP : número de DVRPs resolvidos na otimalidade;
- HEUR : tempo de CPU (em segundos) necessário para executar a heurística;
- TOTAL: tempo total (em segundos) de execução do algoritmo ótimo.

Todos os resultados são a média de 10 problemas testados, exceto indicação em contrário e o procedimento ótimo foi limitado a 1000 segundos de CPU para cada DVRP a ser resolvido. Exemplos que não se enquadraram nesta especificação foram considerados sem sucesso e descartados da análise

n	m	$\overline{Z}/Z^*$	$\underline{Z}/Z^*$	# DVRP	HEUR	TOTAL
20	2	1,53	0,95	3,8	0,001	35,8
	3	1,42	0,93	3,9	0,004	729,5
	4(7)	1,60	0,90	3,9	0,001	1494,1
	5(8)	1,57	0,88	4,5	0,005	3386,9
30	2	1,42	0,98	2,7	0,001	30,2
	3	1,54	0,95	4,7	0,007	2784,8
	4(2)	1,72	0,95	4,5	0,001	1712,5
40	2	1,42	0,99	3,4	0,006	294,7
	3(8)	1,61	0,96	4,5	0,011	2338,6
50	2	1,47	0,98	4,4	0,006	763,0
	3(6)	1,50	0,97	4,2	0,006	2937,3
	5(1)	2,15	0,97	5,0	0,017	607,6

Tabela 4.16 - Resultados Computacionais VMP / BD - Problemas Estruturados

n	m	$\overline{Z}/Z^*$	$\underline{Z}/Z^*$	# DVRP	HEUR	TOTAL
20	2	1,03	0,98	2,7	7,3	42,8
	3	1,02	0,97	2,8	6,0	254,7
	4(9)	1,04	0,95	2,7	4,5	1192,9
	5(8)	1,03	0,95	3,1	4,7	2660,4
30	2	1,05	0,99	2,1	24,8	50,9
	3(9)	1,06	0,97	3,3	22,6	2649,6
	4(2)	1,11	0,97	3,5	13,4	1628,2
40	2	1,08	0,99	2,5	54,2	276,0
	3(8)	1,10	0,97	3,0	34,1	2376,8
50	2	1,12	0,98	3,4	90,3	841,7
	3(6)	1,14	0,98	3,2	69,6	2969,9
	5(1)	1,17	0,98	3,0	67,7	736,2

Tabela 4.17 - Resultados Computacionais ACEL / BD - Problemas Estruturados

n	m	$\overline{Z}/Z^*$	$\underline{Z}/Z^*$	# DVRP	HEUR	TOTAL
20	2	1,01	0,99	2,1	46,7	55,9
	3	1,01	0,97	2,7	23,7	135,1
	4(9)	1,01	0,96	2,9	7,4	1961,3
	5(9)	1,01	0,95	2,7	7,7	2062,1
30	2	1,03	0,99	2,0	266,4	294,5
	3	1,03	0,97	3,1	142,5	2528,1
	4(2)	1,06	0,97	3,0	41,5	1692,2
40	2	1,04	0,99	1,9	1765,9	1906,3
	3	1,06	0,98	2,8	772,2	3521,8
50	2	1,06	0,99	3,0	5730,3	6456,3
	3(6)	1,08	0,98	2,8	2197,3	4966,5
	5(1)	1,08	0,98	3,0	100,1	697,2

Tabela 4.18 - Resultados Computacionais LONG / BD - Problemas Estruturados

n	m	$\overline{Z}/Z^*$	$\underline{Z}/Z^*$	# DVRP	HEUR	TOTAL
20	2	1,01	0,99	2,0	449,0	459,2
	3	1,01	0,97	2,7	129,9	288,2
	4(8)	1,01	0,96	2,9	47,8	1676,9
	5(8)	1,01	0,95	2,8	59,5	2486,3
30	2	1,03	0,99	1,9	1801,9	1826,1
	3(9)	1,03	0,97	3,2	1335,6	3834,6
	4(2)	1,03	0,97	3,0	933,1	2556,8
40	2	1,04	0,99	2,2	21556,7	21701,9
	3(8)	1,04	0,98	2,5	5048,8	7283,7
50	2	1,07	0,99	3,1	24875,3	25635,6
	3(6)	1,07	0,98	2,7	9883,7	12668,3
	5(1)	1,10	0,98	3,0	411,9	1010,5

Tabela 4.19 - Resultados Computacionais AMOST / BD - Problemas Estruturados

n	m	$\overline{Z}/Z^*$	$\underline{Z}/Z^*$	# DVRP	HEUR	TOTAL
20	2	1,01	0,99	1,7	449,0	456,5
	3	1,00	0,97	1,9	129,9	196,3
	4(9)	1,01	0,96	2,1	57,2	797,0
	5(8)	1,01	0,95	2,3	59,5	1812,7
30	2	1,03	0,99	1,8	1801,9	1824,9
	3	1,03	0,97	2,6	1210,4	1876,1
	4(2)	1,03	0,97	2,5	933,1	1074,4
40	2	1,04	0,99	2,4	21556,7	21700,1
	3(9)	1,04	0,98	3,7	5776,7	7401,6
50	2	1,07	0,99	3,2	24875,3	25294,8
	3(6)	1,07	0,98	3,7	9883,7	13282,2
	5(1)	1,10	0,98	4,0	411,9	1041,7

Tabela 4.20 - Resultados Computacionais AMOST / BS - Problemas Estruturados

n	m	$\overline{Z}/Z^*$	$\underline{Z}/Z^*$	# DVRP	HEUR	TOTAL
20	2	1,03	0,98	2,4	7,3	19,8
	3	1,02	0,97	2,1	6,0	84,0
	4(9)	1,04	0,95	2,3	4,5	714,3
	5(8)	1,03	0,95	2,6	4,7	1787,3
30	2	1,05	0,99	2,0	24,8	49,2
	3	1,06	0,97	2,9	24,7	677,6
	4(2)	1,11	0,97	3,0	13,4	158,5
40	2	1,08	0,99	2,8	54,2	277,9
	3(9)	1,10	0,97	4,7	33,0	1753,9
50	2	1,12	0,98	3,7	90,3	988,1
	3(4)	1,14	0,98	5,3	75,8	6412,3
	5(1)	1,17	0,98	5,0	67,7	757,4

Tabela 4.21 - Resultados Computacionais ACEL / BS - Problemas Estruturados

n	m	$\overline{Z}/Z^*$	$\underline{Z}/Z^*$	# DVRP	HEUR	TOTAL
20	2	1,75	0,94	4,0	0,003	18,6
	3	2,00	0,87	4,1	0,003	204,5
	4	1,67	0,86	4,8	0,001	251,7
	5	1,75	0,82	4,7	0,003	472,9
30	2	2,02	0,96	3,9	0,003	79,8
	3	1,85	0,93	4,0	0,006	173,9
	4(7)	1,97	0,91	4,3	0,001	1508,3
	5(3)	2,28	0,91	4,3	0,006	770,6
40	2	2,14	0,98	3,7	0,006	46,7
	3(9)	2,13	0,94	4,8	0,005	1248,0
	4(4)	1,74	0,94	4,0	0,013	791,3
	5(2)	2,32	0,92	4,5	0,017	1716,2
50	2	2,31	0,98	3,7	0,006	204,4
	3(8)	2,32	0,96	4,4	0,011	1611,1
	4(4)	2,09	0,95	4,5	0,013	2090,8

Tabela 4.22 - Resultados Computacionais VMP / BD - Problemas Não-Estruturados

n	m	$\overline{Z}/Z^*$	$\underline{Z}/Z^*$	# DVRP	HEUR	TOTAL
20	2	1,10	0,96	3,1	8,7	34,8
	3	1,12	0,91	3,1	7,2	187,0
	4	1,12	0,89	3,6	5,0	781,5
	5	1,09	0,87	3,5	5,1	136,3
30	2	1,27	0,96	3,5	21,5	120,4
	3	1,26	0,94	3,1	20,9	214,4
	4(7)	1,22	0,91	3,6	13,9	1872,4
	5(3)	1,23	0,91	3,3	18,0	1004,3
40	2	1,39	0,98	3,2	46,0	137,3
	3(9)	1,41	0,95	4,3	36,1	1315,3
	4(4)	1,40	0,94	3,5	28,9	865,2
	5(2)	1,50	0,92	4,0	20,9	1765,2
50	2	1,52	0,98	3,5	86,2	378,5
	3(8)	1,48	0,96	4,0	67,2	1739,2
	4(4)	1,45	0,96	4,3	75,7	2270,8

Tabela 4.23 - Resultados Computacionais ACEL / BD - Problemas Não-Estruturados

n	m	$\overline{Z}/Z^*$	$\underline{Z}/Z^*$	# DVRP	HEUR	TOTAL
20	2	1,04	0,97	2,9	68,6	83,2
	3	1,08	0,91	3,6	24,8	212,5
	4	1,06	0,91	3,1	14,8	150,5
	5	1,06	0,88	3,3	14,5	121,4
30	2	1,15	0,97	2,9	883,9	959,2
	3	1,18	0,94	3,2	132,3	339,4
	4(8)	1,17	0,92	3,6	84,8	2681,7
	5(4)	1,19	0,92	3,3	62,9	2879,8
40	2	1,26	0,98	3,0	1718,5	1763,8
	3(9)	1,30	0,95	4,2	320,7	1920,4
	4(4)	1,28	0,94	3,5	763,3	1537,3
	5(2)	1,33	0,94	4,0	104,1	1866,8
50	2	1,35	0,98	3,2	2037,1	2252,9
	3(8)	1,36	0,97	3,8	1531,4	3479,4
	4(4)	1,35	0,96	3,8	942,8	2994,0

Tabela 4.24 - Resultados Computacionais LONG / BD - Problemas Não-Estruturados

n	m	$\overline{Z}/Z^*$	$\underline{Z}/Z^*$	# DVRP	HEUR	TOTAL
20	2	1,03	0,97	2,9	517,5	534,6
	3	1,09	0,91	3,3	155,7	362,5
	4	1,05	0,93	3,1	47,7	194,5
	5	1,06	0,87	3,6	87,6	251,6
30	2	1,14	0,97	3,0	3315,1	3391,3
	3	1,17	0,95	2,9	1411,3	1582,4
	4(8)	1,18	0,92	3,5	543,4	2958,8
	5(4)	1,18	0,92	3,3	337,5	3181,8
40	2	1,22	0,98	2,9	13871,6	13917,4
	3(9)	1,27	0,95	3,9	3229,2	4487,2
	4(4)	1,28	0,94	3,5	4308,3	5099,5
	5(2)	1,35	0,94	4,0	1912,2	3664,2
50	2	1,35	0,98	3,0	30317,5	30503,4
	3(8)	1,39	0,96	3,9	10260,1	11822,6
	4(4)	1,39	0,96	4,0	4951,8	6941,5

Tabela 4.25 - Resultados Computacionais AMOST / BD - Problemas Não-Estruturados

n	m	$\overline{Z}/Z^*$	$\underline{Z}/Z^*$	# DVRP	HEUR	TOTAL
20	2	1,03	0,97	2,5	517,5	532,1
	3	1,09	0,91	2,8	155,7	225,3
	4	1,05	0,93	2,4	47,7	144,1
	5	1,06	0,87	2,9	87,6	166,5
30	2	1,14	0,97	3,3	3315,1	3352,6
	3	1,17	0,95	3,6	1411,3	1616,0
	4(9)	1,18	0,92	4,4	492,0	2044,5
	5(5)	1,19	0,91	4,6	418,8	2575,8
40	2	1,22	0,98	3,3	13871,6	13899,3
	3(9)	1,27	0,95	5,1	3229,2	3273,1
	4(5)	1,31	0,94	4,8	3913,2	4730,3
	5(2)	1,35	0,94	8,5	1912,2	3903,2
50	2	1,35	0,98	3,3	30317,5	30371,9
	3(8)	1,39	0,96	5,0	10260,1	11379,2
	4(4)	1,39	0,96	6,8	4951,8	8002,6

Tabela 4.26 - Resultados Computacionais AMOST / BS - Problemas Não-Estruturados

n	m	$\overline{Z}/Z^*$	$\underline{Z}/Z^*$	# DVRP	HEUR	TOTAL
20	2	1,10	0,96	2,7	8,7	24,0
	3	1,12	0,91	2,7	7,2	76,8
	4	1,12	0,89	3,5	5,0	111,9
	5	1,09	0,87	3,2	5,1	83,4
30	2	1,27	0,96	4,2	21,5	64,0
	3	1,26	0,94	3,8	20,9	229,5
	4(9)	1,24	0,91	4,9	13,4	1642,3
	5(5)	1,27	0,90	5,2	18,3	2167,3
40	2	1,39	0,98	4,3	46,0	110,2
	3(9)	1,41	0,95	6,4	36,1	879,9
	4(5)	1,39	0,94	4,6	29,1	1652,1
	5(2)	1,50	0,92	10,0	20,9	4073,7
50	2	1,52	0,98	3,9	86,2	198,7
	3(7)	1,49	0,97	4,9	69,0	2396,9
	4(4)	1,45	0,96	7,8	75,7	6430,6

Tabela 4.27 - Resultados Computacionais ACEL / BS - Problemas Não-Estruturados

Analisando-se os resultados, pode-se dizer que o algoritmo proposto produz soluções próximas do ótimo para os problemas estruturados na grande maioria dos casos, porém para problemas não-estruturados, tendem a deteriorar mais rapidamente à medida que a dimensão dos problemas aumenta. LAPORTE, DESROCHERS e NOBERT (1984) e LAPORTE, NOBERT e TAILLEFER (1987) já haviam observado que em problemas não-estruturados, os valores de solução tem uma variação maior que no caso estruturado, e isto resulta em árvores de busca mais limitadas, no algoritmo que resolve o DVRP. Pode-se então concluir que, geralmente, problemas estruturados são mais fáceis de resolver.

Já a heurística VMP produz soluções bastante pobres, porém o tempo computacional necessário para executá-la é baixo. As versões de busca tabu encontram soluções muito melhores às custas, porém, de um maior esforço computacional.

Nota-se também que os problemas não-estruturados, apresentam soluções mais pobres, particularmente para valores maiores de n, porém observa-se que a qualidade da solução para este tipo de problema parece não depender da relação n/m, enquanto que nos problemas estruturados a medida que a relação n/m diminui (isto é, matrizes maiores de tempos de processamento são geradas) a dificuldade de resolução do problema aumenta.

Quanto ao algoritmo ótimo, foram testadas várias combinações usando as heurísticas como pontos de partida e pode-se dizer que quanto melhor a qualidade da solução heurística encontrada, menos DVRPs são necessários resolver para encontrar a solução ótima. Porém, algumas vezes, o tempo gasto para se encontrar a solução heurística de melhor qualidade possível torna-se proibitivo. Portanto, pode-se notar que, em certos casos, é melhor iniciar o procedimento ótimo de BUSCA DICOTÔMICA com soluções de baixa qualidade, como a da heurística VMP, pois a busca dicotômica tende a diminuir a distância entre os limites superior e inferior muito rapidamente. Entretanto, se o investimento na busca de soluções iniciais de boa qualidade for grande, gerando soluções muito próximas do ótimo, o procedimento de BUSCA SUBTRATIVA parece ser o mais indicado. O procedimento de BUSCA SUBTRATIVA tem a vantagem de encontrar somente um DVRP infactível, ou seja, ele só explora totalmente a árvore de busca uma vez, enquanto que o procedimento de BUSCA DICOTÔMICA pode, em certos casos, explorar um número maior de problemas infactíveis até chegar a solução ótima, podendo gerar um acréscimo expressivo no tempo computacional gasto para encontrar a solução ótima do problema.

A tabela 4.28 mostra as melhores combinações dos algoritmos heurísticos e exatos com relação a tempos computacionais.

n	*m*	Problemas Estruturados	Problemas Não-Estruturados
20	2	ACEL / BS	VMP / BD
	3	ACEL / BS	ACEL / BS
	4	ACEL / BS	ACEL / BS
	5	ACEL / BS	ACEL / BS
30	2	VMP / BD	ACEL / BS
	3	ACEL / BS	VMP / BD
	4	ACEL / BS	VMP / BD
	5	-	VMP / BD
40	2	ACEL / BD	VMP / BD
	3	ACEL / BS	ACEL / BS
	4	-	VMP / BD
	5	-	VMP / BD
50	2	VMP / BD	ACEL / BS
	3	VMP / BD	VMP / BD
	4	-	VMP / BD
	5	VMP / BD	-

Tabela 4.28 - Combinações de Algoritmos Heurísticos e Exatos produzindo os Menores Tempos Computacionais

Da tabela 4.28, pode-se notar que as melhores combinações de heurísticas e métodos exatos são ACEL / BS que foi melhor em 58 % dos problemas estruturados e 40 % dos problemas não-estruturados e VMP / BD que forneceu melhores resultados em 33 % dos problemas estruturados e 60 % dos problemas não-estruturados.

Pode-se ver que a combinação VMP / BD foi melhor para problemas não-estruturados, porém a combinação ACEL / BS tem um desempenho computacional semelhante para estes mesmos problemas, mas resolve 4 problemas a mais na otimalidade o que aumenta em 3,5 % o número de problemas resolvidos na otimalidade, sem um aumento considerável de esforço computacional. Portanto, conclui-se que a heurística TABU ACELERADO fornece soluções de boa qualidade e também proporciona um bom

ponto de partida para o algoritmo ótimo de busca subtrativa, tanto para problemas estruturados quanto não-estruturados.

Os métodos TABU LONGO e TABU AMOSTRAL, fornecem soluções de qualidade ainda melhor que TABU ACELERADO. Porém, a medida que a dimensão dos problemas aumenta, o tempo computacional passa a ser um fator limitante para seu uso, apesar de um número maior de problemas ser resolvido na otimalidade.

Vale a pena ressaltar também que problemas com até 50 tarefas foram resolvidos na otimalidade e problemas com até 200 tarefas foram testados no algoritmo heurístico de busca tabu, o que permite dizer que tem-se em mãos um algoritmo heurístico de busca tabu poderoso e um procedimento ótimo para o *Pltpds|Cmax* que, pelo que se sabe da literatura especializada, ainda não havia sido proposto.

Os resultados referentes ao algoritmo proposto para resolver o *Pltpds|Cmax* podem ser encontrados em FRANÇA et al. (1994) e apresentado sob o enfoque do problema Minmax m-caixeiros viajantes podem ser encontrados em FRANÇA et al. (1993b).

CONCLUSÃO

O trabalho desta tese concentrou-se na resolução de dois problemas de sequenciamento: o *P*| |*Cmax* e o *P*|*tpds*|*Cmax*.

O *P*| |*Cmax* é um dos problemas mais estudados na literatura de sequenciamento, constituindo um desafio propor novos algoritmos para ele. O objetivo que norteou o trabalho foi o de desenvolver um algoritmo heurístico que fosse eficiente em termos computacionais e fornecesse soluções de boa qualidade. Outra preocupação que se teve foi a de não executar uma pré-ordenação das tarefas a serem executadas.

Através de testes computacionais, pôde-se notar que o algoritmo satisfez e até mesmo superou as expectativas, pois apresentou um desempenho muito bom para todos os tipos de problemas testados, inclusive para aqueles casos considerados patológicos para outros algoritmos. Não foi identificada uma classe de problemas patológicos onde o algoritmo apresentasse desempenho sistematicamente ruim. Isso lhe confere uma característica de robustez. Observou-se também que o tempo gasto pelo algoritmo cresce de uma maneira abaixo da linear com o crescimento do tamanho do problema (número de tarefas n), permitindo que se trabalhasse com problemas contendo um número grande de tarefas.

Também pôde-se ver que as soluções encontradas ficaram sempre muito próximas da solução ótima (sempre abaixo de 3% em relação a solução ótima), sendo que em torno de 70% das soluções encontradas pelo algoritmo heurístico correspondiam à solução ótima. Portanto, conclui-se que a estrutura de intervalos usada, juntamente com o uso de alvos na escolha das trocas a serem realizadas, reduz de uma maneira bastante racional o espectro de busca, fazendo com que o algoritmo concentre sua atenção em trocas que tenham maior possibilidade de fornecer melhores ganhos.

Todos estes comentários permitem-nos concluir que o método 3-FASES é indicado para a solução do *P*| |*Cmax*, podendo inclusive ser usado como um método para determinação de um limitante superior para a solução do problema dentro de um método ótimo de resolução mais sofisticado que o apresentado neste trabalho.

Um ponto que talvez pudesse ser trabalhado, no sentido de deixar este algoritmo ainda mais eficiente, seria um estudo maior da estrutura de dados utilizada na sua implementação, de forma a diminuir o tempo de busca por trocas factíveis na fase 3 do

algoritmo 3-FASES. Esse tempo ainda é um pouco elevado se comparado ao restante do processo.

Quanto ao algoritmo ótimo desenvolvido para o $P|\ |Cmax$, pode-se dizer que foi de grande utilidade no sentido de provar a qualidade do método heurístico. Não houve, porém, intenção de construir um método que fosse excelente, mas apenas um que cumprisse os propósitos e não fosse extremamente ineficiente.

A outra parte do trabalho esteve concentrada na solução de outro problema de sequenciamento, o $P|\ tpds|Cmax$ assimétrico, que não tem sido muito estudado na sua forma explícita mas, como foi mostrado, guarda estreita relação com o Problema do Caixeiro Viajante e suas variantes.

O objetivo aqui foi desenvolver um algoritmo heurístico para o $P|\ tpds|Cmax$ que usasse técnicas avançadas de busca, por exemplo Busca Tabu, e tirasse vantagem da semelhança com o TSP no processo de alocação e transferências de tarefas entre os processadores. Para isto foi desenvolvido um algoritmo heurístico que usou técnicas de Busca Tabu para dirigir as trocas entre tarefas. As trocas foram realizadas usando uma adaptação de um procedimento denominado GENIUS, que é uma rotina geral de inserção inicialmente desenvolvida para o Problema do Caixeiro Viajante.

Seguindo-se uma tendência da literatura que trata de métodos aplicando Busca Tabu, a maioria dos parâmetros utilizados no processo foi idealizada de forma a ficar dependente do problema. Isso torna o algoritmo bastante fácil de aplicar, bastando apenas que se conheçam as dimensões do problema em questão e sua matriz de tempos de processamento.

O algoritmo mostrou um desempenho muito bom, quando se trata dos problemas gerados de maneira estruturada, porém, perde um pouco em qualidade quando tenta resolver os problemas gerados de maneira não estruturada. Isto pode ser explicado pelo fato que o procedimento de inserção foi, inicialmente, desenvolvido para o TSP e o caso que mais se aproxima de um TSP são os problemas estruturados, pois os tempos de processamento são gerados tomando-se a distância Euclideana entre dois pontos gerados aleatoriamente em um espaço pré-determinado.

Ao nosso ver, os problemas que ocorrem com o algoritmo ao tentar resolver os problemas não-estruturados poderiam ser solucionados, ou pelo menos minimizados, se o procedimento de inserção não fosse tão intimamente ligado ao TSP e não estivesse

exclusivamente baseado em uma estrutura de vizinhança para escolha dos movimentos candidatos. Se fosse aplicado um procedimento de diversificação da busca, para que o algoritmo pudesse explorar regiões diferentes e talvez mais promissores do que as já exploradas, poderiam ser esperados melhores resultados.

Foram desenvolvidas três versões do algoritmo, duas relacionadas com a vizinhança considerada (TABU LONGO e TABU ACELERADO) e outra procurando pontos de partida melhores para o método (TABU AMOSTRAL). TABU LONGO e TABU AMOSTRAL foram os que resolveram maior número de problemas com soluções de melhor qualidade. Porém, o tempo computacional se mostrou demasiadamente elevado para o que se procurava. Então, seguindo a máxima da busca heurística, que é obter boas soluções a um baixo custo computacional, sacrificou-se um pouco a qualidade da solução, diminuindo-se o espectro de busca e, consequentemente, ficando o tempo computacional reduzido. A esta variante do método deu-se o nome de TABU ACELERADO.

TABU ACELERADO fornece boas soluçoes em tempos computacionais razoáveis, servindo muito bem como ponto de partida para o procedimento exato desenvolvido neste trabalho.

O procedimento exato tem duas versões, uma de BUSCA DICOTÔMICA e outra de BUSCA SUBTRATIVA. Pode-se dizer que a Busca Dicotômica é mais adequada para trabalhar com resultados iniciais pobres, pois ela rapidamente reduz a diferença entre o resultado inicial pobre e a solução ótima. Porém, pode ser necessária a identificação de mais de uma solução infactível do problema, processo que tem alto custo computacional. Já a Busca Subtrativa tem um melhor desempenho quando parte de soluções próximas da ótima e garante a identificação de, no máximo, um problema infactível para encontrar a solução ótima do problema.

O algoritmo ótimo desenvolvido para o *P| tpds|Cmax* , inicialmente, teve o mesmo propósito do desenvolvido para o *P| |Cmax*, ou seja, demonstrar o desempenho da heurística, porém com o desenrolar do trabalho investiu-se mais tempo no seu desenvolvimento, criando mais uma ferramenta na resolução exata deste tipo de problemas.

Finalmente, vale ressaltar que a heurística desenvolvida para o *P| tpds|Cmax* pode ser usada, sem nenhuma modificação, para resolver o *Minmax m-TSP*, uma vez que são equivalentes. O problema *No-wait Flow Shop* com o objetivo de minimizar o *makespan* e

sem restrições, que pode ser transformado em um *Minsum TSP*, pode ser resolvido pelo procedimento GENIUS, embutido na heurística.

O que se pode sugerir como trabalho nesta área é a implementação de uma estratégia de diversificação no algoritmo tabu para resolução do *P| tpds|Cmax*.

Outro ponto a ser desenvolvido seria a inclusão de limitantes nos tempos de finalização de cada tarefa (*due dates*), pois há casos onde o cumprimento dos prazos de entrega é prioridade. Isto faria com que, ao invés de minimizar o tempo máximo de finalização (*Cmax*), se pensasse em termos de minimizar a demora (*Lmax*) ou o atraso (*Tmax*), que estão definidos no Capítulo 1 desta tese.

APÊNDICE

Fundamentos da Busca Tabu

Busca Tabu tem seus antecedentes em métodos projetados para superar os limitantes da factibilidade ou otimalidade local, normalmente tratados como barreiras, e para sistematicamente impor e relaxar restrições permitindo a exploração de regiões até então não alcançáveis. A forma atual de BT deriva de GLOVER (1986). Idéias geradoras do método foram também desenvolvidas por PIERRE HANSEN (1986) em uma formulação de *subida acentuada / descida suave (steepest ascent / mildest descent).* O desenvolvimento da BT tem trazido melhorias e refinamentos à idéia original (GLOVER e LAGUNA (1992a). Porem, a intenção aqui é apenas mostrar a idéia geral do método, fazendo uso dos procedimentos mais simples e gerais.

O dicionário Aurélio define tabu como *"proibição aos profanos, nas sociedades primitivas, de se relacionarem com determinadas pessoas, objetos ou lugares, supostamente sagrados"*. BT, porem, tem pouco envolvimento com sociedades primitivas, e muito menos com o sagrado. Ela apenas se baseia na imposição de restrições que servem para dirigir o processo de busca de modo a evitar ciclagem, pois ela permite movimentos que deterioram a função objetivo. Para isso, soluções que foram recentemente visitadas ficam temporariamente proibidas de ser exploradas e são armazenadas em uma lista, chamada de *lista tabu.*

A filosofia de BT é encontrar e explorar um conjunto de princípios de resolução inteligente de problemas. O elemento fundamental da BT é o uso de *memória flexível.* Do ponto de vista de BT, memória flexível incorpora o processo dual de criar e explorar estruturas levando em conta seu passado (deste modo combinando as atividades de adquirir e tirar proveito da informação). As estruturas de memória da BT operam fazendo referência a quatro dimensões principais: momento do surgimento, freqüência, qualidade e influência. Estas dimensões são comparadas a uma base de conhecimento de estrutura lógica e conectividade. O papel destes elementos na criação de processos efetivos de solução de problemas é o fundamento da BT.

BT pode ser vista como uma metaheurística que pode ser superimposta a outros procedimentos com o objetivo de evitar as armadilhas criadas pelos *ótimos locais.* O

método pode ser usado por qualquer processo que aplique um conjunto de *movimentos* para transformar uma solução em outra, e que forneça uma função de avaliação para medir a atratividade destes movimentos. A forma de dirigir a busca é bastante flexível e geralmente induz a criação de novos tipos de movimentos e critérios de avaliação que tiram proveito desta habilidade de adaptar-se a diferentes estruturas de problemas.

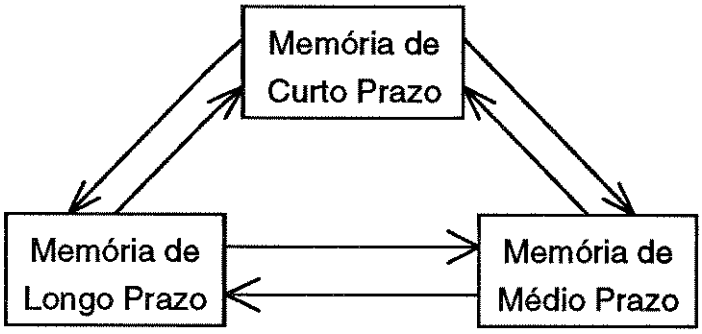
BT está fundamentada em três aspectos principais:

- o uso de estruturas de memória flexíveis, projetadas para permitir um critério de avaliação e busca de informação passada, que explore o espectro de busca de forma mais abrangente que em formas mais rígidas de memória (como em Branch and Bound e A^*) ou que em sistemas sem memória (como em Simulated Annealing, que tem suas origens no antigo trabalho de METROPOLIS et al. - 1953 e foi primeiramente aplicado ao campo de otimização combinatória por KIRKPATRICK, GELATT e VECCHI -1983, e outros procedimentos aleatórios);

- um mecanismo de controle associado que faz uso das estruturas de memória; este sistema está baseado na inter-relação existente entre as condições que restringem e liberam o processo de busca ;

- a incorporação de funções de memória com diferentes tempos de duração, com o objetivo de implementar estratégias de *intensificação* (dizem respeito a procedimentos que dão ênfase a combinação de movimentos e caracteristicas de solução que se mostraram boas) e *diversificação* da busca (procura levar o processo a explorar regiões novas). As ligações entre os diferentes tipos de memória e seus componentes estratégicos associados pode ser vista na Figura A.1.

Funções de Memória



Componentes Estratégicos

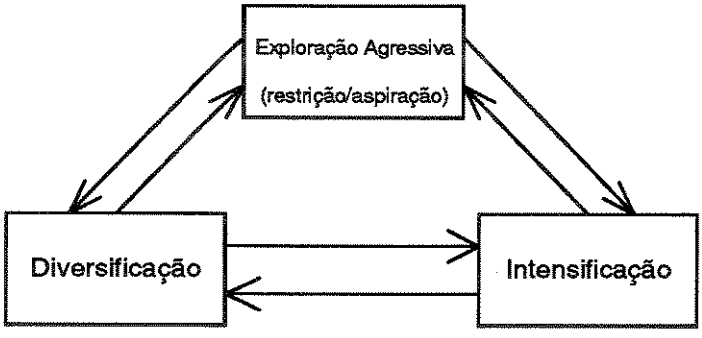


Figura A.1 - Tipos de Memória da BT e seus Componentes Estratégicos

Dentre estas ligações a que mais se destaca é a definida entre memória de curto prazo e a estratégia de exploração agressiva, que se realiza através da inter-relação entre certas restrições (chamadas restrições tabu) e o critério de aspiração (que permite a realização de um movimento proibido sob certas condições).

A divisão em duas classes representada por *restrição* e *aspiração* assemelha-se à representada por *intensificação* e *diversificação*. A chave para estabelecer um balanço efetivo entre tais polaridades e a motivação para que estruturas flexíveis de memória guiem o processo de busca, é permitir que o método realize três coisas:

- incorporação de certos tipos de movimentos aparentemente inúteis, que anteriormente não podiam ser aplicados por métodos altamente restritivos ou sem memória (por exemplo, na memória de curto prazo um movimento que venha a piorar a solução atual);

- aplicação de um intensivo conhecimento na busca (tirando vantagem de informação vinda de observações anteriores do processo de busca ou de problemas da mesma classe ou correlatos);

- anexação do poder heurístico de especialização (incluindo regras que tirem proveito das estruturas específicas do problema).

A ação fundamental da BT está apoiada na memória de curto prazo e muitas observações sobre esta memória reaparecem em memórias de prazos mais longos, amplificadas em grau, mas sem mudanças no tipo.

A memória de curto prazo da BT busca o melhor movimento possível, requerendo que as escolhas disponíveis satisfaçam certas restrições (restrições tabu). O objetivo principal destas restrições é fazer com que o método vá além dos pontos de otimalidade local, enquanto continua escolhendo o melhor movimento a cada passo. Em geral as restrições tabu têm como objetivo procurar prevenir um comportamento cíclico induzindo o método a seguir uma nova trajetória (evitando retornar a soluções já visitadas anteriormente). Estas restrições não operam de maneira isolada, mas são contrabalanceadas pela aplicação de *critérios de aspiração*, que permitem que estas restrições sejam relaxadas mediante

alguma condição especial ou um conjunto de condições especiais, deixando o processo de busca com uma maior flexibilidade (um exemplo de critério de aspiração é permitir que um movimento proibido (tabu) seja realizado sempre que ele levar a uma solução melhor que a melhor solução já encontrada).

A escolha do *melhor movimento* possível (que pode ou não melhorar a solução atual) está baseada no fato que movimentos com avaliações mais altas tem maior possibilidade de levarem a soluções ótimas ou quase-ótimas. Estas avaliações são feitas em função de um estado de busca, que envolve um conjunto de características da solução atual, em conjunto com o conhecimento adquirido pela resolução do problema até o momento e pela experiência em resolver certas classes de problemas.

O funcionamento da BT pode ser resumido na figura A.2, onde os passos de maior importância estão destacados por linhas duplas no diagrama. A escolha do melhor movimento pode ser vista com detalhes na figura A.3.

O critério de parada é dado por um número limite de iterações, que pode ser reduzido para permitir um maior uso das fases de intensificação e diversificação, relacionadas com as funções de memória de médio e longo prazo. Este limite pode ser aumentado, permitindo que a componente de memória de curto prazo guie a maior parte da busca, o que em muitas aplicações tem produzido soluções superiores às até então encontradas, eliminando o uso de memórias de prazos mais longos.

A forma modular do processo da BT torna fácil a implementação inicial da componente de memória de curto prazo e uma posterior incorporação das componentes restantes, caso seja necessário uma solução de melhor qualidade (caso que ocorre principalmente em problemas de alto grau de dificuldade).

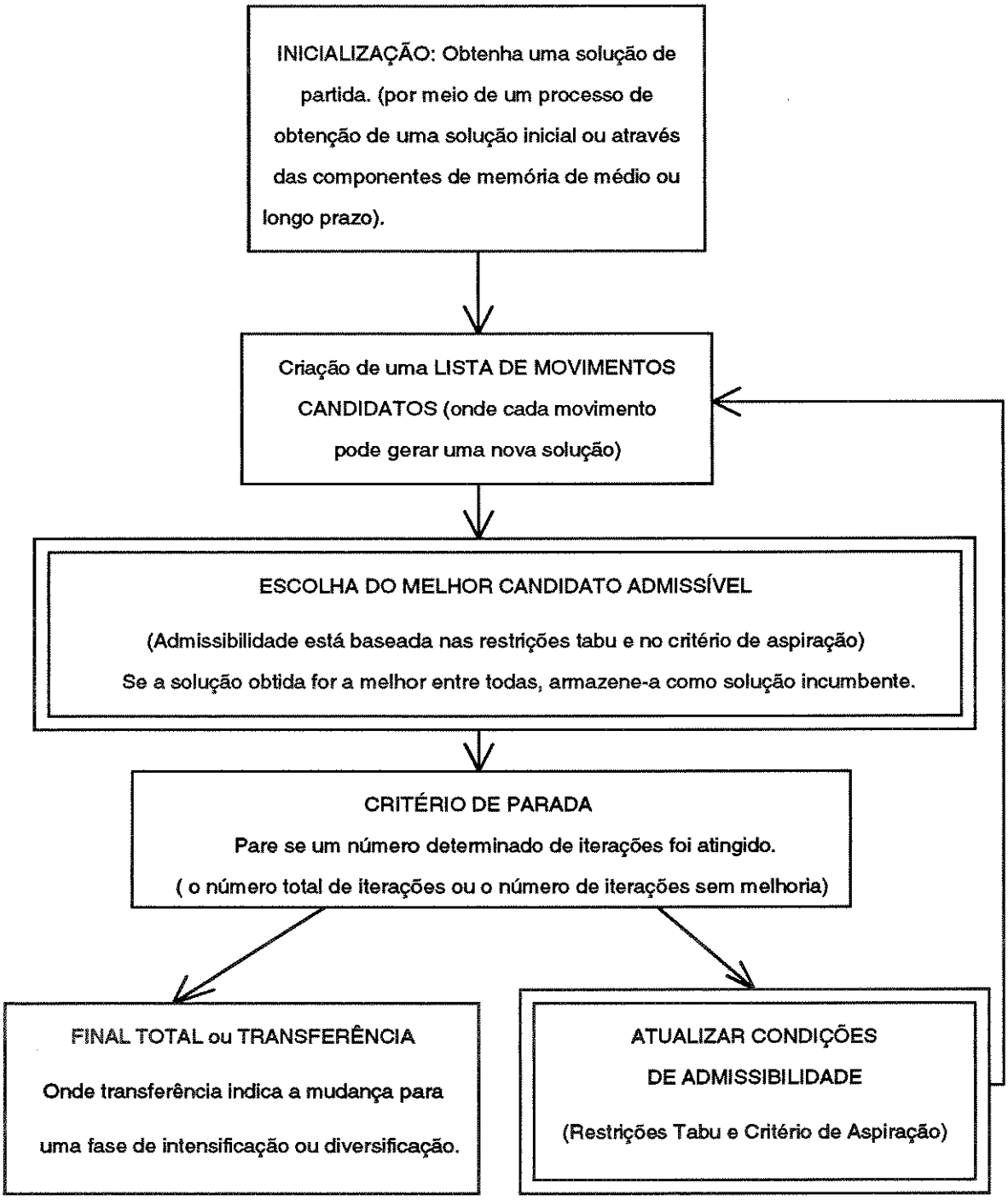


Figura A.2 - Passos Básicos da Busca Tabu

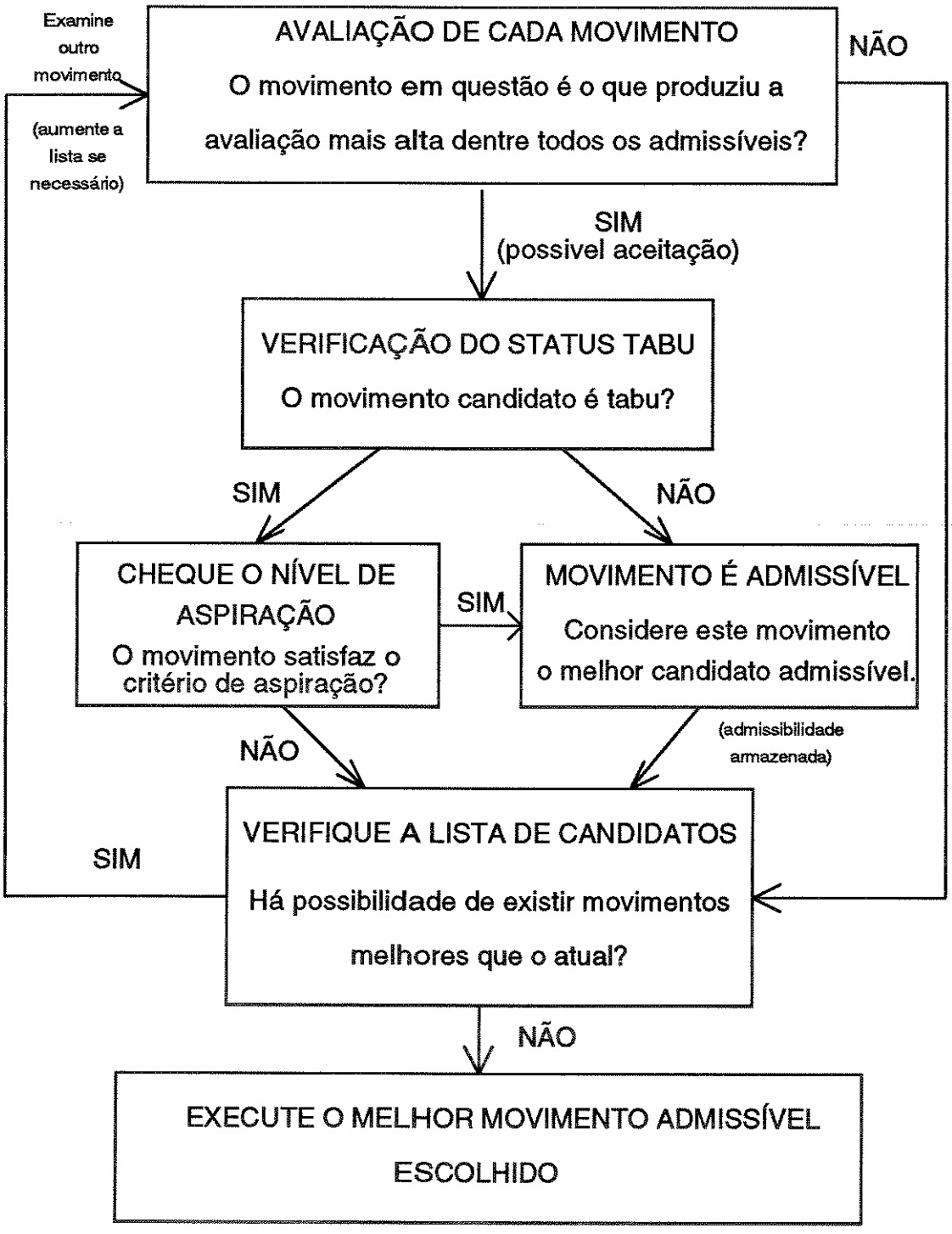
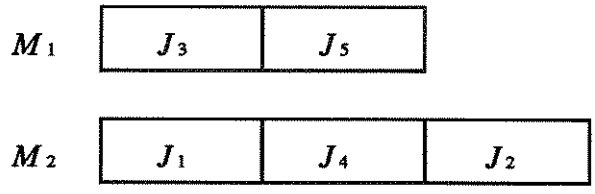


Figura A.3 - Escolha do Melhor Movimento Admissível

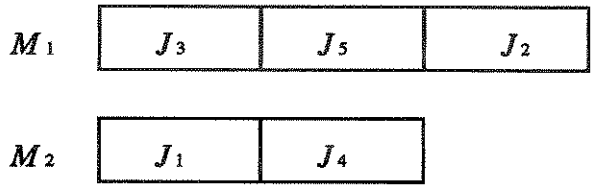
As restrições tabu e o critério de aspiração da BT operam pela identificação dos atributos dos movimentos que, se classificados tabu, fazem com que os movimentos que os contenham adquiram o status tabu.

Com o objetivo de prevenir que um movimento seja revertido (procurando evitar a ciclagem) são criadas as restrições tabu, selecionando um conjunto de atributos e armazenando-os na lista tabu. Numa implementação simples, estes atributos são armazenados na lista tabu por t iterações, onde t é um parâmetro que identifica o tamanho da lista. A cada novo atributo ou conjunto de atributos adicionados à lista, retira-se o atributo mais antigo (adicionado à lista há $t+1$ iterações passadas), podendo esta lista ser tratada como uma lista circular de tamanho t. Uma vez que o atributo é retirado da lista, ele perde seu status tabu, e conseqüentemente, os movimentos que continham estes atributos ficam liberados para serem selecionados desde que não contenham outros atributos na lista.

Para exemplificar estes conceitos apresenta-se um problema simples de *P|tpds|Cmax* consistindo de dois processadores (M_1 e M_2) e cinco tarefas (J_1, J_2, J_3, J_4 e J_5). Suponha que em um determinado estágio da busca tem-se a seguinte configuração:



O movimento escolhido para ser realizado é trocar J_2 do processador M_2 para M_1 resultando em:



Portanto o movimento proibido por um determinado número de iterações será o retorno de J_2 para M_2 .

Para uma maior detalhamento de BT pode-se recorrer às referências citadas nesta seção, principalmente GLOVER (1989, 1990a e 1990b).

E SE TUDO ESTIVER ERRADO...?

Referências Bibliográficas

BALAS, E. e CHRISTOFIDES, N., A restricted lagrangean approach to the traveling salesman problem, Mathematical Programming, Vol. 21, pp. 19-46, 1981.

BAKER, K. R., Introduction to Sequencing and Scheduling, John Wiley & Sons Inc., 1974.

BARNES, J. W. e LAGUNA, M., Solving the multiple-machine weighted flow time problem using tabu search, a ser publicado em IIE Transactions, 1992.

BLAND, J. A. e DAWSON, G. P., Tabu search and design optimization, Computer-Aided Design, Vol. 23(3), pp. 195-202, 1991.

BLAZEWICZ, J., Selected topics in scheduling theory, Annals of Discrete Mathematics, Vol. 31, pp. 1-60, 1987.

BODIN, L., GOLDEN, B., ASSAD, A. e BALL, M., Routing and scheduling of vehicles and crews: the state of the art, Computers & Operations Research, Vol. 10(2), pp. 65-211, 1983.

BRADY, R.M., Optimization strategies gleaned from biological evolution, Nature, Vol. 317, pp. 804-806, 1985.

BRATLEY, P., FLORIAN, M. e ROBILLARD, P., Scheduling with earliest start and due date constraints on multiple machines, Naval Research Logistics Quarterly, Vol. 22, pp. 165-173, 1975.

BURBIDGE, J. L., The Introduction of Group Technology, John Wiley & Sons Inc., 1975.

BURKARD, R. E. e RENDL, F., A thermodynamically motivated simulation procedure for combinatorial optimization problems, European Journal of Operations Research, Vol. 17, pp. 169-174, 1984.

CARPANETO, G. e TOTH, P., Some new branching and bounding criteria for the asymetric traveling salesman problem, Management Science, Vol. 26(7), pp. 736-743, 1980a.

CARPANETO, G. e TOTH, P., Solution of the assignment problem, ACM Trans. math. Softw., Vol. 6(1), pp. 104-111, 1980b.

CHENG, T.C.E. e SIN, C.C.S., A state-of-the-art review of parallel-machine scheduling research, European Journal of Operational Research, Vol. 47, pp. 271-292, 1990.

CLARKE, G. e WRIGHT, J., Scheduling vehicles from a central depot to a number of delivery points, Operations Research, Vol. 12, pp. 568-581, 1964.

COFFMAN Jr., E. G., GAREY, M. R. e JOHNSON, D. S., An application of bin-packing to multiprocessor scheduling, SIAM Journal on Computing, Vol. 7, pp. 1-17, 1978.

COFFMAN Jr., E. G., GAREY, M. R. e JOHNSON, D. S., Approximation algorithms for bin-packing - an updated survey, Algorithm Design for Computer System Design, G. AUSIELLO, M. LUCERTINI e P. SERAFINI (eds.), pp. 49-106, Springer, Viena, 1984.

CONWAY, R.W., MAXWELL, W.L. e MILLER, L.W., Theory of Scheduling, Addison-Wesley Reading, Mass., 1967.

COOK, S.A., The complexity of theorem proving procedures, Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, pp. 151-158, 1971.

CRAINIC, T. G., GENDREAU, M., SORIANO, P. e TOULOUSE, M., A tabu search procedure for multicommodity location/allocation with balancing requirements, publicação número 797, CRT - Centre de Recherche sur les Transports, Montreal - Canadá, 1991.

CHRISTOFIDES, N. e EILON, S., Algorithm for the vehicle-dispatching problem, Operational Research Quarterly, Vol. 20(3), pp. 309-318, 1969.

CHRISTOFIDES, N., Worst-case analysis of a new heuristic for the traveling salesman problem, Report 388, Graduate School of Industrial Administration, Carnegie-Mellow University, Pittsburgh, PA, 1976.

DEARING, P.M. e HENDERSON, R.A., Assigning looms in a textile weaving operation with changeover limitations, Production and Inventory Management, Vol. 25, pp. 23-31, 1984.

DELL´AMICO, M. e MARTELLO, S., Optimal scheduling of tasks on identical parallel processors, Research Report OR/90/7, DEIS, University of Bologna, Italy, 1990.

DELL´AMICO, M. e TRUBIAN, M., Applying tabu search to the job-shop scheduling problem, Annals of Operations Research, 1992.

FIECHTER, C-N., A parallel tabu search algorithm for large traveling salesman problems, Working Paper 90/1, Département de Mathématiques, École Polytechnique Fédérale de Lausanne, 1990.

FINN, G. e HOROWITZ, E., A linear time approximation algorithm for multiprocessor scheduling, BIT, Vol. 19, pp. 312-320, 1979.

FISCHETTI, M. e TOTH, P., An additive bounding procedure for the asymetric travelling salesman problem, Mathematical Programming, 1991.

FRANÇA, P. M., GENDREAU, M., LAPORTE, G. e MÜLLER, F. M., A composite heuristic for the identical parallel machine scheduling with minimum makespan objective, a ser publicado em Computers & Operations Research, 1993a.

FRANÇA, P. M., GENDREAU, M., LAPORTE, G. e MÜLLER, F. M., The m-traveling salesman problem with minmax objective, submetido à Transportation Science, 1993b.

FRANÇA, P. M., GENDREAU, M., LAPORTE, G. e MÜLLER, F. M., A tabu search heuristic for multiprocessor scheduling with sequence dependent setup times, submetido ao XXXII TIMS, Anchorage, Alaska, EUA, 1994.

FRANÇA, P. M. e MÜLLER, F.M., Programação de N tarefas em M processadores paralelos: resultados de uma nova heurística de trocas, Anais do 9º Congresso Brasileiro de Automática, Vitória - ES, 1992.

FREDERICKSON, G. N., HETCH, M. S. e KIM, C. E., Approximation algorithms for some routing problems, SIAM Journal on Computing, Vol. 7, pp. 178-193, 1978.

FRENCH, S., Sequencing and Scheduling: An Introduction to the Mathematics of the Job Shop, John Wiley & Sons, Chichester, 1986.

FRIESEN, D. K., Tighter bounds for the MULTIFIT processor scheduling, SIAM Journal on Computing, Vol. 13, pp. 170-181, 1984.

FRIESEN, D. K. e LANGSTON, M.A., Evaluation of a MULTIFIT-based scheduling algorithm, Journal of Algorithms, Vol. 7, pp. 35-59, 1986.

GAREY, M. R. e JOHNSON, D. S., Strong NP-completeness results: motivation, examples and implications, Journal of the Association for Computing Machinery, Vol. 25, pp. 499-508, 1978.

GAREY, M. R. e JOHNSON, D. S., Computers and Intractibility: a Guide to the Theory of NP-Completeness, Freeman, San Francisco, 1979.

GENDREAU, M., HERTZ, A. e LAPORTE, G., A tabu search heuristic for the vehicle routing problem, Management Science, 1991.

GENDREAU, M., HERTZ, A. e LAPORTE, G., New insertion and post-optimization procedures for the traveling salesman problem, Operations Research, Vol. 40(6), pp. 1086-1094, 1992.

GENDREAU, M., SORIANO, P. e SALVAIL, L., Solving the maximum clique problem using a tabu search approach, Annals of Opeartions Research, 1992.

GLOVER, F., Future paths for integer programming and links with artificial intelligence, Computers and Operations Research, Vol. 5, pp. 533-549, 1986.

GLOVER, F., Tabu search: part I, ORSA Journal on Computing, Vol. 1, pp. 190-206, 1989.

GLOVER, F., Tabu search: part II, ORSA Journal on Computing, Vol. 2, pp. 4-32, 1990a.

GLOVER, F., Tabu Search: a Tutorial, CAAI Report, University of Colorado, Boulder, 1990b.

GLOVER, F. e LAGUNA, M., Tabu search, Modern Heuristic Techniques for Combinatorial Problems, C. REEVES (ed.), Blackwell Scientific Publications, Oxford, 1992a.

GLOVER, F. e LAGUNA, M., Bandwidth packing: a tabu search approach, Management Science, 1992b.

GOLDEN, B.L. e STEWART Jr., W.R., Empirical analysis of heuristics, The Traveling Salesman Problem. A Guided Tour of Combinatorial Optimization, E.L. LAWLER, J.K. LENSTRA, A.H.G. RINNOOY KAN e D.B. SHMOYS (eds.), Wiley, Chichester, pp. 207-249, 1985.

GRÖTSCHEL, M. e HOLLAND, O., Solution of large-scale symetric traveling salesman problems, Mathematical Programming, Vol. 51, pp. 141-202, 1991.

GRAHAM, R. L., Bounds for certain multiprocessing anomalies, Bell System Tech. Journal, Vol. 45, pp. 1563-1581, 1966.

GRAHAM, R. L., Bounds on multiprocessing timing anomalies, SIAM Journal on Applied Mathematics, Vol. 17, pp. 416-429, 1969.

GRAHAM, R. L., LAWLER, E. L., LENSTRA, J. K. e RYNNOOY KAN, A. H. G., Optimization and approximation in deterministic sequencing and scheduling: a survey, Annals of Discrete Mathematics, Vol. 5, pp. 287-326, 1979.

GUINET, A., Textile production systems: a succession of non-identical parallel processor shops, Journal of the Operational Reasearch Society, Vol. 42(8), pp. 655-671, 1991.

HANSEN, P., The steepest ascent / mildest descent heuristic for combinatorial programming, trabalho apresentado no Congress on Numerical Methods in Combinatorial Optimization, Capri, Itália, 1986.

HASTINGS, N. A. J. e PEACOCK, J. B., Statistical Distributions, John Wiley & Sons, New York, 1974.

HERTZ, A., JAUMARD, B. e ARAGÃO, M. P., Topology of local optima for the K-coloring problem, Discrete Applied Mathematics, 1992.

HERTZ, A. e de WERRA, D., Using tabu search for graph coloring, Computing, Vol. 29, pp. 345-351, 1987.

HERTZ, A. e de WERRA, D., The tabu search metaheuristic: how to use it, Annals of Mathematics and Artificial Intelligence, 1990.

HOCHBAUM, D.S. e SHMOYS, D.B., Using dual approximation algorithms for scheduling problems: a theoretical and practical results, Journal of the Association for Computing Machinery, Vol. 34, pp. 144-162, 1987.

JOHNSON, D.S., Local optimization and the traveling salesman problem, Proceedings of the 17th International Colloquium on Automata, Languages and Programming, M.S. PATERSON (ed.), Lecture Notes in Computer Science, Springer-Verlag, Berlin, pp. 446-461, 1990.

KARMARKAR, N. e KARP, R.M., The differencing method of set particioning, Report UCB/CSD 82/113, Computer Science Division, University of California, Berkeley, 1988.

KARP, R.M., Reductibility among combinatorial problems, Complexity of Computer Computations, R.E. MILLER e J.W. THATCHER (eds.), Plenum Press, New York, pp. 85-103, 1972.

KARP, R.M., On the computational complexity of combinatorial problems, NETWORKS, Vol. 5, pp. 45-68, 1975.

KIRKPATRICK, S., GELATT Jr., C. D. e VECCHI, M. P., Optimization by simulated annealing, Science, Vol. 220, pp. 671-680, 1983.

LAGUNA, M. , BARNES, J. W. e GLOVER, F., Tabu search methods for a single machine scheduling problem, Journal of Intelligent Manufacturing, Vol. 2, pp. 63-74, 1991.

LANGSTON, M. A., Improved 0/1 interchange scheduling, BIT, Vol. 22, pp. 282-290, 1982.

LAPORTE, G. e NOBERT, Y., A cutting planes algortihm for the m-salesmen problem, Journal of the Operational Research Society, Vol. 31(11), pp. 1017-1023, 1980.

LAPORTE, G., DESROCHERS, M. e NOBERT, Y., Two exact algorithms for the distance-constrained vehicle routing problem, NETWORKS, Vol. 14, pp. 161-172, 1984.

LAPORTE, G., MERCURE, H. e NOBERT, Y., An exact algorithm for the asymmetrical capacitated vehicle routing problem, NETWORKS, Vol. 16, pp. 33-46, 1986.

LAPORTE, G., NOBERT, Y. e TAILLEFER, S., A branch-and-bound algorithm for the asymetrical distance-constrained vehicle routing problem, Mathematical Modelling, Vol. 9(12), pp. 857-868, 1987.

LAPORTE, G., The traveling salesman problem: an overview of exact and approximate algorithms, European Journal of Operational Research, Vol. 59, pp. 231-247, 1992.

LAWLER, E.L., LENSTRA, J.K., RINNOOY KAN, A.H.G. e SHMOYS, D.B., The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization, Wiley, Chichester, 1985.

LAWLER, E.L., LENSTRA, J.K., RINNOOY KAN, A.H.G. e SHMOYS, D.B., Sequencing and scheduling: algorithms and complexity, Report BS-R8909, Center for Mathematics and Computer Science, Amsterdan, 1989.

LEE, C. e MASSEY, D., Multiprocessor scheduling: combining LPT e MULTIFIT, Discrete Applied Mathematics, Vol. 20, pp. 233-242, 1988.

LENSTRA, J. e RINNOOY KAN, A.H., Some simple applications of the traveling salesman problem, Operational Research Quarterly, Vol. 24(4), pp. 717-733, 1975.

LENSTRA, J. e RINNOOY KAN, A.H., Computational complexity of discrete optimization problems, Annals of Discrete Mathematics, Vol. 4, pp. 121-140, 1979.

LEVIN, L.A., Universal sequential search problems, Problems Inform. Transmission, Vol. 9 , pp. 265-266, 1975.

LIN, S., Computer solutions of the traveling salesman problem, Bell System Computer Journal, Vol. 44, pp. 713-733, 1965.

LIN, S. e KERNIGHAN, B., An effective heuristic algorithm for the traveling salesman problem, Operations Research, Vol. 21, pp. 498-516, 1973.

LITTLE, J.D.C., MURTY, K.G. e KAREL, C., An algorithm for the traveling salesman problem, Operations Research, Vol. 11, pp. 972-989, 1963.

LOCKET, A.G. e MUHLEMAN, A.P., A scheduling problem involving sequence dependent changeover times, Operations Research, Vol. 20, pp. 895-902, 1972.

MALEK, M., GURUSWAMY, M., PANDYA, M. e OWENS, H., Serial and parallel simulated annealing and tabu search algorithms for the traveling salesman problem, Annals of Operations Research, Vol. 21, pp. 59-84, 1989.

MARTELLO, S. e TOTH, P., Knapsack Problems: Algorithms and Computer Implementations, Wiley, Chinchester, 1990.

McNAUGHTON, R., Scheduling with deadlines and loss functions, Management Science, Vol. 6(1), pp. 1-12, 1959.

METROPOLIS, N., ROSENBLUTH, A. W., ROSENBLUTH, M. N., TELLER, A. H. e TELLER, E., Equation of state calculations by fast computing machines, Journal of Chemical Physics, Vol. 21, pp. 1087-1091, 1953.

MILLER, D.L. e PEKNY, J.F., Exact solution of large asymmetric traveling salesman problems, Science, Vol. 251, pp. 754-761, 1991.

MÜHLENBEIN, H., GORGES-SCHLEUTER, M. e KRÄMER, O., Evolution algorithms in combinatorial optimization, Parallel Computing, Vol. 7, pp. 65-85, 1988.

MÜLLER, F. M., Busca Tabu na Solução de Problemas de Programação Zero-Um, Tese de Mestrado, UNICAMP, 1990.

OLIVEIRA, S. e STROUD, G., A parallel version of tabu search and the path assignment problem , Heuristics for Combinatorial Optimization, Vol. 4, pp. 1-24, 1989.

OR, I., Traveling Salesman-type Combinatorial Problems and their Relation to the Logistics of Regional Blood Banking, Ph.D. Dissertation, Northwestern University, Evanston, IL, 1976.

PADBERG, M.W. e RINALDI, G., Optimization of a 532-city symetric traveling salesman problem by branch and cut, Operations Research Letters, Vol. 6, pp. 1-7, 1987.

PADBERG, M.W. e RINALDI, G., Facet identification for the symetric traveling salesman problem, Mathematical Programming, Vol. 47, pp. 219-257, 1990.

PAPADIMITRIOU, C.H. e STEIGLITZ, K., Combinatorial Optimization: Algorithms and Complexity, Prentice-Hall, New Jersey, 1982.

PARKER, R.G., DEANE, R.H. e HOLMES, R.A., On the use of a vehicle routing algorithm for the parallel processors problems with sequence dependent changeover costs, AIIE Transactions, Vol. 9(2), pp. 155-160, 1977.

PEKNY, J.F. e MILLER, D.L., A parallel branch and bound algorithm for solving large asymmetric traveling salesman problems, Mathematical Programming, Vol. 55, pp. 17-33, 1992.

PUREZA, V. M. e FRANÇA, P.M., Vehicle routing problems via tabu search metaheuristic, publicação número 747, CRT - Centre de Recherche sur les Transports, Montreal - Canadá, 1991.

SANHI, S.K., Algorithms for scheduling independent tasks, Journal of the Association for Computing Machinery, Vol. 23, pp. 116-127, 1976.

SEDEWICK, R., Algorithms in C, Addison-Wesley, USA, 1990.

SEMET, F. e TAILLARD, E., Solving real-life vehicle routing problems efficiently using taboo search, a ser publicado em Annals of Operations Research, 1992.

SHMOYS, D.B. e TARDOS, E., Computational complexity of combinatorial problems, Handbook in Combinatorics, R.L. GRAHAM, M. GRÖTSCHEL e L. LVOÁSZ (eds.), North Holland, Amsterdan, 1989.

SILVEIRA, P. D. B., Algoritmos para a Resolução do Problema de Estoque e Roteamento de Veículos, Tese de Mestrado, UNICAMP, 1992.

STEIGLITZ, K. e WEINER, P., Some improved algorithms for computer solution of the traveling salesman problem, Proceedings of the 6th Annual Allerton Conference on Circuit Theory, pp. 814-821, 1968.

STOCKMEYER, L.J., Complexity theory, Handbooks in Operations Research and Management Science; Volume 3: Computation, E.G. COFFMAN Jr., J.K. LENSTRA e A.H.G. RINNOOY KAN (eds.), Cap. 8, North-Holland, Amsterdan, 1990.

SUMICHRAST, R.T., Scheduling parallel processors to minimize setup time, Computers & Operations Research, Vol. 14(4), pp. 305-313, 1987.

SUMICHRAST, R.T. e BAKER, J.R., Scheduling parallel processors: an integer linear programming based heuristic for minimizing setup time, International Journal of Production Research, Vol. 25(5), pp. 761-771, 1987.

TAILLARD, E., Robust taboo search for the quadratic assignment problem, Parallel Computing, Vol. 17, pp. 473-455, 1991.

de WERRA, D. e HERTZ, A., Tabu search techniques: a tutorial and an applications to neural networks, OR Spectrum, Vol. 11, pp. 131-141, 1989.

WHITE, C.H. e WILSON, R.C., Sequence dependent set-up times and job sequencing, International Journal of Production Research, Vol. 15(2), pp. 191-202, 1977.

WOODRUFF, D. L. e SPEARMAN, M. L., Sequencing and batching for two classes of jobs with deadlines and setup times, Production and Operations Management, Vol. 1, pp. 87-102, 1992.