UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO
DEPARTAMENTO DE ENGENHARIA DE SISTEMAS

MINIMIZAÇÃO DO ATRASO MÉDIO NA PROGRAMAÇÃO DE MÁQUINAS PARALELAS: UMA APLICAÇÃO DE BUSCA TABU

DENISE SATO YAMASHITA

Orientador:
Prof. Dr. Vinícius Amaral Armentano

Tese apresentada à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas - UNICAMP como parte dos requisitos exigidos para o obtenção do título de MESTRE EM ENGENHARIA ELÉTRICA.

-AGOSTO 1996-

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

Y14m

Yamashita, Denise Sato
Minimização do atraso médio na programação de máquinas paralelas: uma aplicação de busca tabu / Denise Sato Yamashita.--Campinas, SP: [s.n.], 1996.

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

1. Heurística. 2. Processamento paralelo (Computadores). 3. Processamento sequencial (Computação). 4. Controle de produção. 5. Pesquisa operacional. I. Armentano, Vinícius Amaral. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Aos meus Pais.

AGRADECIMENTOS

A todos que colaboraram para a realização deste trabalho. Em especial

ao Prof. Vinícius A. Armentano, pela paciência e pela dedicação, e com quem muito aprendi;

à Fran, Renata, Vitória, Cíntia pelo carinho e amizade;

à Regina, pelas preciosas dicas em *talk, email, finger...*

à Débora pelas discussões sobre diversificação, intensificação, tabu..., que muito me ajudaram;

ao Walcir, à Márcia e à Suze, pela paciência;

ao Jesse por saber ouvir;

à Gladys por estar sempre alegre, por seus conselhos e sua presença nas horas difíceis;

ao Bernhard pelos *Emails* otimistas, pelos comentários "terríveis", e pela força;

à CAPES pelo apoio financeiro;

à minha família pelo constante interesse, incentivo e carinho;

ao Sávio por compartilhar os bons e maus momentos, pela sua paciência e por seu amor.

RESUMO

Esta dissertação trata do problema de programar n tarefas em m máquinas paralelas idênticas, com o objetivo de minimizar o atraso médio em relação às datas de entrega. Para resolver o problema, propõe-se uma aplicação de busca tabu e duas estratégias de diversificação. O desempenho das heurísticas foi comparado através de testes computacionais gerados para 900 problemas. Foram realizados testes envolvendo até 10 máquinas e 150 tarefas. Para 540 problemas os resultados são comparados com limitantes inferiores gerados por relaxação lagrangeana. Em mais de 65% desses problemas, os resultados dos métodos propostos chegaram a menos de 1% do limitante inferior.

Palavras chave: Programação de máquinas paralelas; Atraso; Heurística; Busca tabu.

ABSTRACT

This thesis deals with the problem of scheduling n jobs on m parallel identical machines with the objective of minimizing the mean tardiness. In order to solve this problem, it is proposed a tabu search approach and two diversification strategies. The performance of the heuristics was measured by computational tests for 900 problems. The tests were made in instances with up to 10 machines and 150 jobs. For 540 problems, the results are compared with lower bounds given by a lagragian relaxation. In more than 65% of these problems, the results of the proposed method are within 1% of the lower bounds.

Key words: Parallel machine scheduling; Tardiness; Heuristics; Tabu search.

CONTEÚDO

INTRODUÇÃO

Busca tabu é uma metaheurística que tem como finalidade guiar um procedimento de heurística de busca local para explorar o espaço de soluções além da otimalidade local. Nos últimos anos, a busca tabu tem sido aplicada com sucesso a diversos tipos de problema de otimização combinatória. Nem sempre os elementos básicos da busca tabu são suficientes para garantir um bom desempenho do método, devido ao confinamento da busca em uma região. Neste caso, é interessante diversificar a busca, explorando diferentes regiões do espaço de soluções. Essas estratégias de diversificação têm sido aplicadas com bons resultados a muitos problemas.

Esta tese apresenta uma aplicação de busca tabu e duas estratégias de diversificação aplicadas ao problema de programação de n tarefas em m máquinas paralelas idênticas. Cada tarefa tem um tempo de processamento e uma data de entrega. O objetivo é minimizar o atraso médio em relação às datas de entrega. As tarefas têm pesos iguais, e podem ser processadas por qualquer uma das máquinas. Não é permitido *preemption*, isto é uma vez que a tarefa começa a ser processada ela não pode ser interrompida.

A literatura para este problema é escassa. Em geral, os métodos de resolução adotados consistem de heurísticas de busca local. Não foi encontrado na literatura nenhum trabalho que aplicasse busca tabu a este problema. Entretanto, a busca tabu e estratégias de diversificação foram aplicadas com sucesso a problemas de máquinas paralelas com outros objetivos, como minimizar o tempo total de execução das tarefas e o tempo de fluxo ponderado.

A heurística desenvolvida para o problema consiste de uma fase construtiva seguida de uma aplicação da busca tabu. Foram implementadas também duas estratégias de diversificação. Na fase construtiva as tarefas são ordenadas por uma regra, e depois são alocadas nas máquinas. Aplica-se então uma heurística para minimizar o atraso em cada máquina separadamente.

Para avaliar o desempenho das heurísticas foram gerados aleatoriamente, 900 instâncias, seguindo um padrão encontrado na literatura. Além disso, para 540 instâncias foi possível gerar um limitante inferior obtido através de relaxação lagrangeana.

A apresentação da tese está organizada da seguinte forma. O Capítulo 1 consiste na apresentação formal do problema de atraso em máquinas paralelas e trata da revisão da literatura para o problema. O Capítulo 2 apresenta uma breve descrição da metodologia da busca tabu e explica como ela foi aplicada ao problema tratado nesta tese. Descrevem-se também as duas diversificações implementadas, e a heurística construtiva. O Capítulo 3 apresenta os resultados computacionais e as conclusões. Neste capítulo descreve-se o método usado para gerar as instâncias, o desempenho das heurísticas em relação à heurística construtiva e ao limitante inferior. No final do trabalho encontra-se um apêndice onde são mostrados detalhadamente o cálculo do tamanho da vizinhança, a complexidade

computacional, e a relaxação lagrangeana utilizada para se obter os limitantes inferiores.

CAPÍTULO 1

O Problema de Atraso Médio em Máquinas Paralelas

Este capítulo trata da definição do problema, e apresenta uma revisão bibliográfica sobre o assunto.

1.1 Definição do Problema

O problema abordado neste trabalho consiste em programar tarefas em máquinas paralelas idênticas de forma a minimizar o atraso médio das tarefa em relação às datas de entrega. Não é permitido interrupção *(preemption)*, ou seja, uma vez que a tarefa começa a ser processada pela máquina ela ocupa a máquina até que seja totalmente completada. As datas de entrega e os tempos de processamentos são determinísticos. Cada tarefa deve ser processada por exatamente uma máquina e o tempo de processamento da tarefa é independente da máquina e da ordem em que a tarefa está sendo processada. Todas as tarefas estão disponíveis para processamento no instante 0.

A seguinte notação é usada para o problema:

Dados:

m - número de máquinas paralelas.
n - número de tarefas a serem processadas.
p_i - tempo de processamento da tarefa i.
d_i - data de entrega da tarefa i.
M_i - máquina i.

Variáveis

C_i - tempo de término da tarefa i.
T_i - atraso da tarefa i, onde $T_i = max(0, C_i - d_i)$.

Seja S um programa *(schedule)* com a seguinte forma:

$$S = \{S_1, S_2, \ldots, S_m\}$$

onde

$$S_k = \{J_k(1), J_k(2), \ldots, J_k(n_k)\}$$

é a seqüência das tarefas na máquina k, $J_k(i)$ é o índice da tarefa que ocupa a posição i na máquina k, e n_k é o número de tarefas alocadas na máquina k, para $k = 1, 2, \ldots, m$ e

$$n = \sum_{k=1}^{m} n_k.$$

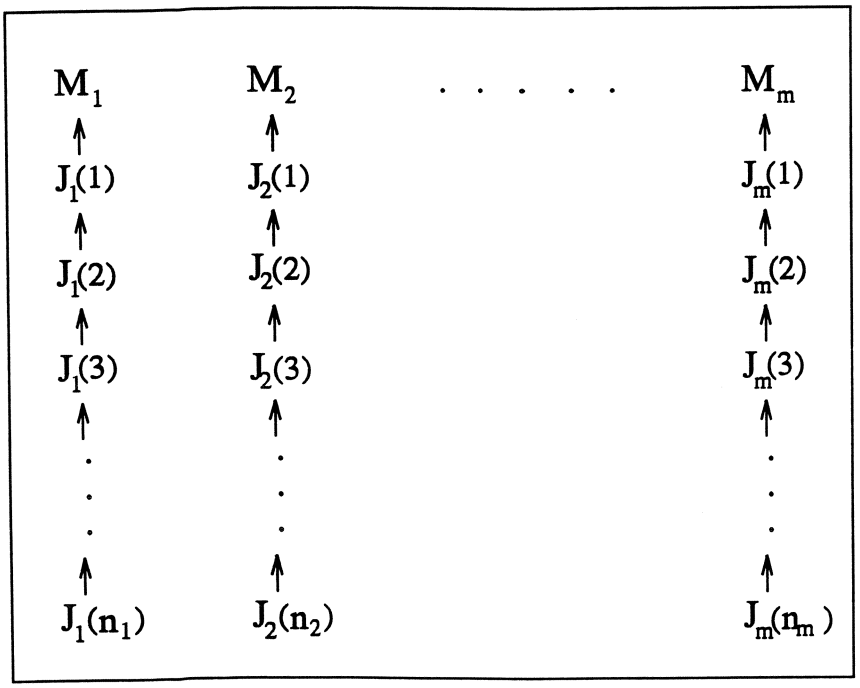
O objetivo é encontrar uma partição de tarefas (ou seja, um programa S) com a finalidade de:

$$Minimizar\, T(S) = \frac{1}{n} \sum_{k=1}^{m} T(S_k)$$

$$T(S_k) = \sum_{i=1}^{n_k} max(0, C_{J_k(i)} - d_{J_k(i)})$$

Daqui por diante, o `atraso médio` ou equivalentemente `atraso total`, será chamado de `atraso` simplesmente.

O problema pode ser ilustrado na figura a seguir.



1.2 Revisão Bibliográfica

`Koulamas (1994)` apresenta uma revisão de literatura para o problema de atraso em uma máquina, máquinas paralelas , flowshop e jobshop. A literatura sobre o problema

de minimizar o atraso em máquinas paralelas não é muito vasta. Este problema é *NP-hard*, pois o problema para uma máquina também o é (Du e Leung (1990)). Devido a sua complexidade, o problema de atraso em máquinas paralelas idênticas é geralmente abordado através de heurísticas.

Uma classe de heurísticas citadas a seguir utiliza regras de despacho para alocar tarefas a máquinas de acordo com algum critério. As regras mais usadas são: *EDD (Earliest Due Date)*, onde as tarefas são ordenadas em ordem crescente de data de entrega; *SPT (Shortest Processing Time)*, em que as tarefas são seqüenciadas em ordem crescente de tempo de processamento; *SLACK* que consiste em ordenar as tarefas em ordem ascendente de $(d_i - p_i)$.

A seguir, utilizam-se heurísticas para minimizar o atraso em cada máquina.

Wilkerson e Irwin (1971) propõem uma heurística para o problema de atraso em uma máquina e sugerem que a heurística proposta poderia ser usada em conjunto com um mecanismo para alocar tarefas em múltiplas máquinas. Baker (1973) apresentou essa heurística proposta por Wilkerson e Irwin (1970) para máquinas paralelas idênticas. Inicialmente as tarefas são ordenadas por EDD. A seguir, a tarefa é alocada na máquina o mais próximo possível de sua data de entrega. Não são apresentadas comparações dos resultados computacionais com outros métodos.

Dogramaci e Surkis (1979) utilizam três regras de despacho (EDD, SPT e $SLACK$). A seguir, alocam cada tarefa na máquina com menor tempo de término, e tratam separadamente o problema de minimização do atraso em cada máquina. Das três soluções obtidas, a melhor é escolhida. Foram testados 560 problemas gerados aleatoriamente, com tamanhos variando entre 12 e 30 tarefas e 2 a 5 máquinas. O desempenho da heurística proposta foi avaliado por $y = \frac{H-L}{M-L}$ onde H é o atraso gerado pela heurística; L é o valor do limitante inferior obtido através de programação linear; e M é o atraso médio de 500 programas gerados aleatoriamente. Para 92% dos casos, $y \leq 0.15$, o que confirma que a solução heurística está muito mais próxima do limitante inferior que a média dos programas gerados aleatoriamente.

Ho e Chang (1991) ordenam as tarefas pela regra de despacho TPI (*traffic priority index*) proposta pelos autores. A alocação de tarefas a máquinas é idêntica à descrita anteriormente, e os autores propõem uma heurística baseada em trocas de tarefas adjacentes para minimizar o atraso em cada máquina. Resultados computacionais são apresentados para problemas com tamanhos diversos, entre 20 e 200 tarefas e 1 a 10 máquinas. Os resultados computacionais indicam que a heurística proposta tem um desempenho melhor (0.22% em média) que a heurística de Wilkerson-Irwin (1971).

Koulamas (1994) apresenta uma heurística construtiva para o problema de atraso em máquinas paralelas. A heurística proposta consiste em inicialmente ordenar as tarefas em ordem crescente de tempo de processamento (SPT). A seguir são construidas m cópias dessas listas. Seleciona-se então uma tarefa de cada máquina usando a heurística PSK proposta por Panwalkar, Smith e Koulamas (1993) apresentada no Capítulo 2. Se ao menos uma dessas tarefas puder ser alocada sem atraso na respectiva máquina, selecione aquela com menor data de entrega. Se as tarefas estiverem atrasadas, aloca-se a que resultar no menor atraso. À medida que as tarefas vão sendo alocadas, elas

são retiradas das listas. São apresentados resultados computacionais para problemas de 200 tarefas e 2, 3 e 4 máquinas. Koulamas (1994) conclui através desses resultados que sua heurística tem um desempenho superior a Wilkerson e Irwin (1970), Dogramaci e Surkis (1979) e Ho e Chang (1991). São apresentados ganhos da ordem de 150% em relação às heurísticas citadas anteriormente.

Luh et al.(1990) propõem um limitante inferior fornecido por Relaxação Lagrangeana para o problema de atraso ponderado em máquinas paralelas idênticas. A seguir apresentam uma heurística baseada nesta relaxação. São realizados testes computacionais com 40 máquinas e 80 a 120 tarefas. A maior parte dos resultados ficou a menos de 1% do limitante inferior.

Existem também artigos que tratam de variações do problema de minimizar o atraso em máquinas paralelas como Sahni e Cho (1980) e Arkin e Roundy (1991). Sahni e Cho (1980) abordam o problema de minimizar o atraso em máquinas paralelas idênticas permitindo interrupção de tarefas. Arkin e Roundy (1991) propõem um algoritmo para minimizar o atraso ponderado total em máquinas paralelas idênticas. Os pesos das tarefas são proporcionais aos tempos de processamento.

Além do atraso, existem outros critérios que podem ser minimizados em máquinas paralelas, como em França et al.(1994) que minimizam o *makespan*, e Li e Cheng (1994) que tratam do problema de desvio absoluto em relação à data de entrega *(absolute lateness)*. Cheng e Sin (1990) apresentam uma revisão de literatura para o problema de máquinas paralelas.

Não foi encontrado nenhum artigo que aplica busca tabu ao problema de minimizar o atraso em máquinas paralelas idênticas. Entretanto, aplicações de Busca Tabu ao problema de máquinas paralelas minimizando outros critérios podem ser encontrados na literatura. Alguns desses artigos são citados a seguir. Hübscher e Glover (1994) tratam do problema de minimizar o *makespan*. Laguna e Velaverde (1991) abordam o problema de minimizar o avanço *(earliness)* ponderado com *deadlines*. Em França et al. (1995) aplica-se Busca Tabu ao problema de minimizar o *makespan* quando os tempos de processamento dependem da seqüência, e Barnes e Laguna (1993) tratam do problema de minimizar o tempo de fluxo ponderado.

CAPÍTULO 2

Heurística Proposta

Este capítulo apresenta a heurística proposta para a solução do problema abordado. A heurística consiste de uma fase construtiva seguida de uma aplicação de busca tabu.

A fase construtiva, apresentada na seção 2.1, é formada por três procedimentos básicos: ordenar as tarefas por uma regra de despacho, distribuir as tarefas nas máquinas, e minimizar o atraso em cada máquina utilizando uma heurística.

A seção 2.2 apresenta um resumo da metodologia da busca tabu e descreve como ela foi aplicada ao problema tratado neste trabalho, definindo o tipo de vizinhança utilizada, as regras de ativação e duração tabu.

A seção 2.3 apresenta duas estratégias de diversificação aplicadas ao problema estudado.

2.1 Heurística Construtiva

A heurística construtiva consiste de três procedimentos básicos:

I - Ordenar as n tarefas usando a regra de despacho MDD.
II - Distribuir as tarefas nas máquinas.
III - Minimizar o atraso em cada máquina aplicando a heurística PSK.

2.1.1 Procedimentos I e II

A regra MDD *(Modified Due Date)* ordena as tarefas por ordem crescente de MDD, onde MDD da tarefa i é dada por $max(C(t) + p_i; d_i)$, e $C(t)$ é o tempo de término da última tarefa seqüenciada até o instante t. Esta regra foi proposta por **Baker and Bertrand(1982)**

Depois de ordenadas por esta regra de despacho, as tarefas são alocadas na máquina com menor tempo de término. Foram realizados testes alocando-se as tarefas na máquina com menor atraso, numa tentativa de "equilibrar" os atrasos entre as máquinas. Entretanto, os resultados obtidos não foram tão bons. Uma possível justificativa é que o atraso de uma tarefa não depende dos atrasos das tarefas que a precedem na máquina, e portanto,

alocar a tarefa na máquina com menor atraso pode gerar um alto valor de atraso naquela máquina. Isto é, dada uma tarefa, suponha que deseja-se alocá-la na máquina com menor atraso (máquina k). O atraso da tarefa $J_k(i)$ é dado por: $max(C(t)+p_i-d_i,0)$. Observe que o atraso da tarefa alocada não depende do atraso da máquina k, mas essencialmente do tempo de término ($C(t)$) da máquina. Portanto, alocar tarefas na máquina com menor atraso pode levar a atrasos maiores do que na máquina com menor tempo de término.

2.1.2 Procedimento III

Panwalkar, Smith e Koulamas (1993) desenvolveram uma eficiente heurística construtiva, PSK, descrita a seguir.

Considere um conjunto de tarefas ordenadas por SPT (Os empates são resolvidos por EDD). Inicialmente, todas as tarefas não seqüenciadas estão em um conjunto ordenado $U(1,2,3,...n)$. A primeira tarefa de U é portanto a menor tarefa não seqüenciada. O conjunto S contém as tarefas seqüenciadas; este conjunto está inicialmente vazio. Seja C a soma dos tempos de processamento de todas as tarefas em S, isto é, o tempo de término da última tarefa em S.

Passo 1. Se U contém somente 1 tarefa, insira-a na última posição em S e vá para o Passo 9. Caso contrário, denomine a primeira tarefa de U como a tarefa ativa i, isto é, a próxima tarefa candidata a ser seqüenciada.

Passo 2. Se $C+p_i \geq d_i$, vá para o Passo 8.

Passo 3. Selecione a próxima tarefa de U, e denomine-a como tarefa j.

Passo 4. Se $d_i \leq C+p_j$, vá para o Passo 8.

Passo 5. Se $d_i \leq d_j$, vá para o Passo 7.

Passo 6. A tarefa j agora passa a ser a tarefa ativa i. Se esta é a última tarefa em U, vá para o Passo 8. Caso contrário retorne ao Passo 2.

Passo 7. Se j é a última tarefa em U, vá para o Passo 8. Caso contrário, retorne ao Passo 3.

Passo 8. Remova a tarefa i de U e coloque-a após a última tarefa programada em S, $C = C + p_i$ e retorne ao Passo 1.

Passo 9. Calcule o atraso total para a seqüência e encerre.

Segundo Koulamas(1994), a complexidade do PSK é $O(n^2 \log n)$. Entretanto, analisando a heurística é possível notar que essa complexidade pode ser reduzida a $O(n^2)$. O

pior caso da heurística PSK ocorre quando a tarefa ativa i é comparada com todas as tarefas subseqüentes pertencentes a U. Isto é, os passos $3, 4, 5$ e 7 são repetidos até que j seja a última tarefa em U. Neste caso, na primeira iteração esses passos são repetidos $(n-1)$ vezes. Na segunda iteração, $(n-2)$, na terceira, $(n-3)$, e assim sucessivamente, até a *n-ésima* iteração. Portanto, esses passos são executados $n(n-1)/2$ vezes no pior caso. Este laço contendo os passos $3, 4, 5$ e 7 é o de maior complexidade da heurística PSK, portanto, a complexidade da heurística pode ser reduzida para $O(n^2)$.

A seguir, apresenta-se uma explicação mais detalhada da heurística. Inicialmente as tarefas são colocadas em ordem SPT. O algoritmo realiza n passagens na lista U, começando no início dela. Em cada passagem, retira uma tarefa de U e a insere em S. Esta heurística tem alguns passos baseados em uma das condições de otimalidade para atraso em uma máquina devida a **Emmons** (1969) citada a seguir.

Sejam duas tarefas i e j, e β_j é o conjunto das tarefas que precedem a tarefa j. A tarefa i precede a tarefa j em alguma seqüência ótima se a condição abaixo é satisfeita:

$$p_i \leq p_j \text{ e } d_i \leq max(p_j + \sum_{k \in \beta_j} p_k, d_j)$$

O Passo 1 denomina a primeira tarefa de U como tarefa ativa i, isto é, a próxima tarefa candidata a ser seqüenciada. Caso exista somente uma tarefa em U, ela é seqüenciada imediatamente e o algoritmo termina no Passo 9.

O Passo 2 verifica se a tarefa i ficará atrasada se for seqüenciada imediatamente. Caso isso seja verdade, então a tarefa i é retirada de U e é seqüenciada. Se a tarefa i é a tarefa não seqüenciada com menor tempo de processamento, então a decisão deste passo é ótima. Isto porque $p_i \leq p_j$ e $d_i \leq C + p_i \leq C + p_j$, $\forall j \in U$, onde $C = \sum_{k \in \beta_j} p_k$. Pela condição de Emmons, conclui-se que a tarefa i deve preceder a tarefa j.

O Passo 3 seleciona a próxima tarefa de U, tarefa j, para ser comparada com a tarefa ativa i.

O Passo 4 verifica se a tarefa i ficará atrasada caso seja seqüenciada após j. Se isso acontecer, a tarefa i deve preceder a tarefa j. Essa decisão é ótima caso i seja a tarefa com menor tempo de processamento em U. Isso pode ser mostrado usando a condição de Emmons. Por hipótese, $p_i \leq p_j$ e $d_i \leq C + p_j$, $\forall j \in U$, onde $C = \sum_{k \in \beta_j} p_k$. Portanto a política ótima é que a tarefa i deve preceder a tarefa j.

Se a condição no Passo 5 for satisfeita, então a tarefa j é eliminada de sua posição atual e a próxima tarefa em U é selecionada para ser comparada com a tarefa i. Este passo sempre resultará em otimalidade, pois $p_i \leq p_j$ e $d_i \leq d_j$ e portanto pela condição de Emmons citada acima, a tarefa i deve preceder a tarefa j.

O Passo 6 denomina a tarefa j como a nova tarefa ativa i. Neste caso, a nova tarefa ativa i deixa de ser a tarefa (não seqüenciada) com menor tempo de processamento.

A seguir apresenta-se um exemplo da heurística PSK para o problema com 7 tarefas, cujos tempos de processamento e datas de entrega são dados na próxima tabela.

tarefa	1	2	3	4	5	6	7
p	30	59	9	39	13	60	52
d	16	127	84	104	130	105	27

Inicialmente as tarefas são ordenadas em SPT, portanto,

$$U = 3, 5, 1, 4, 7, 2, 6 \, e \, S = \{\} \, e \, C = 0.$$

Iteração 1:
P1: i=3
P2: $C + p_3 = 0 + 9$ maior ou igual a 84 ? Não.
P3: $j = 5$
P4: $d_3 = 84$ menor ou igual a $C + p_5 = 0 + 13$? Não.
P5: $(d_3 = 84) \leq (d_5 = 130)$? Sim $\Rightarrow$ vá para P7.
P7: j não é a última tarefa em $U \Rightarrow$ vá para P3.
P3: $j = 1$
P4: $(d_3 = 84) \leq (C + p_1 = 0 + 30)$? Não.
P5: $(d_3 = 84) \leq (d_1 = 16)$? Não.
P6: $i = j = 1. \Rightarrow$ vá para P2.
P2: $C + p_1 = 0 + 30 \geq (d_1 = 16)$? Sim $\Rightarrow$ vá para P8.
P8: $U = \{3, 5, 4, 7, 2, 6\}$ e $S = \{1\}$. $C = C + p_1 = 0 + 30 = 30$ Retorne a P1

Iteração 2:
P1: i=3
P2: $(C + p_3 = 30 + 9) \geq (d_3 = 84)$? Não.
P3: $j = 5$
P4: $(d_3 = 84) \leq (C + p_5 = 30 + 13)$? Não.
P5: $(d_3 = 84) \leq (d_5 = 130)$? Sim $\Rightarrow$ vá para P7.
P7: j não é a última tarefa em $U \Rightarrow$ vá para P3.
P3: $j = 4$
P4: $(d_3 = 84) \leq (C + p_4 = 30 + 39)$? Não.
P5: $(d_3 = 84) \leq (d_4 = 104)$? Sim $\Rightarrow$ vá para P7.
P7: j não é a última tarefa em $U \Rightarrow$ vá para P3.
P3: $j = 7$
P4: $(d_3 = 84) \leq (C + p_7 = 30 + 52)$? Não.
P5: $(d_3 = 84) \leq (d_7 = 27)$? Não.
P6: $i = j = 7. \Rightarrow$ vá para P2.
P2: $(C + p_7 = 30 + 52) \geq (d_7 = 27)$? Sim. $\Rightarrow$ vá para P8.
P8: $U = \{3, 5, 4, 2, 6\}$ e $S = \{1, 7\}$. $C = C + p_7 = 30 + 52 = 82$. Retorne a P1.

Iteração 3:
P1: i=3
P2: $(C + p_3 = 82 + 9) \geq (d_3 = 84)$? Sim. $\Rightarrow$ vá para P8.

P8: $U = \{5, 4, 2, 6\}$ e $S = \{1, 7, 3\}$. $C = C + p_3 = 82 + 9 = 91$. Retorne a P1.

Iteração 4:
P1: i=5
P2: $(C + p_5 = 91 + 13) \geq (d_5 = 130)$? Não.
P3: $j = 4$
P4: $(d_5 = 130) \leq (C + p_4 = 91 + 39)$? Sim. $\Rightarrow$ vá para P8.
P8: $U = \{4, 2, 6\}$ e $S = \{1, 7, 3, 5\}$. $C = C + p_5 = 91 + 13 = 104$. Retorne a P1.

Iteração 5:
P1: i=4
P2: $(C + p_4 = 104 + 39) \geq (d_4 = 104)$? Sim. $\Rightarrow$ vá para P8.
P8: $U = \{2, 6\}$ e $S = \{1, 7, 3, 5, 4\}$. $C = C + p_4 = 104 + 39 = 143$. Retorne a P1.

Iteração 6:
P1: i=2
P2: $(C + p_2 = 143 + 59) \geq (d_2 = 127)$? Sim. $\Rightarrow$ vá para P8.
P8: $U = \{6\}$ e $S = \{1, 7, 3, 5, 4, 2\}$. $C = C + p_2 = 143 + 59 = 202$. Retorne a P1.

Iteração 7:
P1: U contém somente a tarefa 6, portanto, ela é inserida na última posição de S: $U = \{\}$, $S = \{1, 7, 3, 5, 4, 2, 6\}$. $\Rightarrow$ vá para P9.
P9: atraso total da seqüência é: 347.

O desempenho da heurística construtiva utilizando o PSK foi comparado à heurística construtiva proposta em Yamashita e Armentano (1995). Nesse trabalho, ao invés do PSK, utilizou-se a heurística API *(adjacent pairwise interchange)*, baseada em troca de tarefas adjacentes, proposta por Fry et al.(1989) para minimizar o atraso em cada máquina. Em instâncias com até 20 tarefas por máquina, os dois métodos encontraram soluções de valores muito parecidos ou idênticos. Já em problemas com um maior número de tarefas por máquina, o PSK gerou soluções de melhor qualidade. Em relação aos tempos computacionais, a heurística PSK foi no mínimo, 1.3 vezes mais rápida que o API e no máximo, cerca de quatro vezes mais rápida. A escolha da heurística de construção utilizando PSK deve-se à boa qualidade dos resultados fornecidos e ao bom desempenho computacional.

Concluído o procedimento da heurística construtiva, aplica-se então a busca tabu ao problema.

2.2 Busca Tabu

A busca tabu é uma metaheurística que tem como finalidade guiar um procedimento de heurística de busca local para explorar o espaço de soluções além da otimalidade local. Nos últimos anos, a busca tabu tem sido aplicada com sucesso a diversos tipos de problema de otimização combinatória (Laguna(1994), Glover(1995)).

A busca tabu é baseada na idéia de que um método de solução do problema, para ser qualificado como inteligente, deve incorporar **memória adaptativa** e **estratégia de busca**. O uso da memória adaptativa constrasta com os modelos sem memória, e com os de memória rígida como o **Branch and Bound**. A ênfase na existência de uma estratégia de busca, seja determinística ou probabilística, deriva da suposição que uma busca informada é melhor que uma boa escolha aleatória.

Noções básicas da busca tabu podem ser descritas como se segue. Seja $f(x)$ uma função a ser otimizada sobre um conjunto X. A busca tabu começa do mesmo modo que uma busca local comum, prosseguindo iterativamente de um ponto solução a outro, até que um critério de parada seja obedecido. Cada x pertencente a X tem uma vizinhança $V(x)$ associada, que está contida em X, e cada solução x' pertencente a $V(x)$ é atingida a partir de x por uma operação chamada **movimento**. O **valor do movimento** representa a mudança no valor da função objetivo em conseqüência da realização do movimento.

A busca tabu vai além da busca local, empregando uma estratégia de modificar $V(x)$ à medida que a busca prossegue, substituindo-a por uma outra vizinhança $V^*(x)$. Como já foi dito anteriormente, um aspecto chave na busca tabu é o uso de estruturas especiais de **memória** que servem para determinar $V^*(x)$, e então organizar o modo no qual o espaço é explorado. A seguir, são descritos alguns aspectos importantes da memória na busca tabu.

A memória usada na busca tabu é explícita e atributiva. A **memória explícita** armazena soluções completas e consiste basicamente de soluções de elite visitadas durante a busca. Essas soluções especiais são introduzidas em intervalos estratégicos para ampliar $V^*(x)$. A **memória atributiva**, por outro lado, guarda informações sobre atributos de soluções que caracterizam a mudança de uma solução para outra.

Outro aspecto importante da memória da busca tabu é a diferença entre **memória de curto prazo** e **memória de longo prazo**. Cada uma delas vem acompanhada de suas próprias estratégias especiais. Entretanto, ambas podem ser vistas como uma forma de modificar a vizinhança $V(x)$ referente a uma dada solução x.

Nas estratégias de busca tabu baseadas em memória de curto prazo, $V^*(x)$ é um subconjunto de $V(x)$, e a classificação tabu serve para identificar elementos de $V(x)$ excluidos de $V^*(x)$. O tipo mais comum de memória de curto prazo é aquele que guarda atributos de soluções recentes. Isso previne certas soluções do passado recente de pertencerem a $V^*(x)$ e portanto de serem revisitadas. Em algumas aplicações, os componentes da memória de curto prazo da busca tabu não são suficientes para produzir soluções de alta qualidade. Nesse caso, a busca tabu torna-se significantemente mais eficiente ao se introduzir memória de longo prazo e suas estratégias associadas.

A memória de longo prazo armazena dados importantes como a freqüência com que os atributos mudam, ou a freqüência com que um determinado atributo participa de soluções geradas no decorrer da busca. Dois importantes componentes de memória de longo prazo da busca tabu são as estratégias de **diversificação** e **intensificação**. Estratégias de intensificação são baseadas em modificar regras de seleção a fim de encorajar combinações de características de movimentos e soluções historicamente boas. As estratégias de diversificação têm a finalidade de direcionar a busca para novas regiões. Em geral, são

baseadas em modificar regras de seleção para favorecer atributos de soluções que não são freqüentemente selecionados(usando funções de penalidade, por exemplo).

2.2.1 Vizinhança no Problema Estudado

Para definir a vizinhança $V(x)$ são considerados dois tipos de **movimentos**, **inserção** e **troca**. O **valor de movimento** é a variação do atraso total da nova solução gerada pelo movimento.

Movimento de Inserção

O movimento de inserção consiste em retirar uma tarefa de uma máquina e inserí-la em uma posição de outra máquina (diferente da máquina de origem da tarefa). Dada uma solução, a vizinhança de inserção é formada por todas as soluções que podem ser alcançadas através do movimento, ou seja, para cada tarefa do problema, considera-se a sua inserção em todas as posições de todas as máquinas com exceção de sua máquina de origem. Seja n_i o número de tarefas em M_i. Então o tamanho da vizinhança de inserção é,

$$\sum_{i=1}^{m-1} \sum_{j=i+1}^{m} (2n_i n_j + n_i + n_j)$$

Um limitante superior fraco para o tamanho da vizinhança de inserções pode ser dado por,

$$n^2 \frac{(m-1)m}{2}$$

É possível obter uma estimativa mais realista do tamanho da vizinhança considerando-se n múltiplo de m e sob a hipótese que existem n/m tarefas por máquina. Neste caso, a vizinhança formada pelos movimentos de inserção tem tamanho

$$(\frac{n}{m} + 1)(n)(m-1)$$

Um exemplo de inserção pode ser visto a seguir. Sejam os programas $S_1 = \{J_1(1), J_1(2), J_1(3), J_1(4)\}$ e $S_2 = \{J_2(1), J_2(2), J_2(3)\}$. Suponha que a tarefa $J_1(2)$ seja inserida na máquina 2 (Figura ($1a$)). Inicialmente $J_1(2)$ é retirada da máquina 1 (Figura ($1b$)). Avaliam-se então as soluções obtidas ao se inserir $J_1(2)$ em cada uma das posições da máquina 2 (Figura ($1c$)), gerando as soluções da Figura ($1e$). Portanto, as 4 soluções que podem ser alcançadas ao se inserir a tarefa $J_1(2)$ na máquina 2 são formadas pela combinação do programa de M_1 (Figura ($1d$)) com os programas da Figura ($1e$).

Movimento de troca

O movimento de troca consiste em trocar duas tarefas entre duas máquinas. As tarefas podem ocupar qualquer posição da máquina na qual estão sendo inseridas. As soluções que compõem a vizinhança dos movimentos de troca são obtidas da seguinte

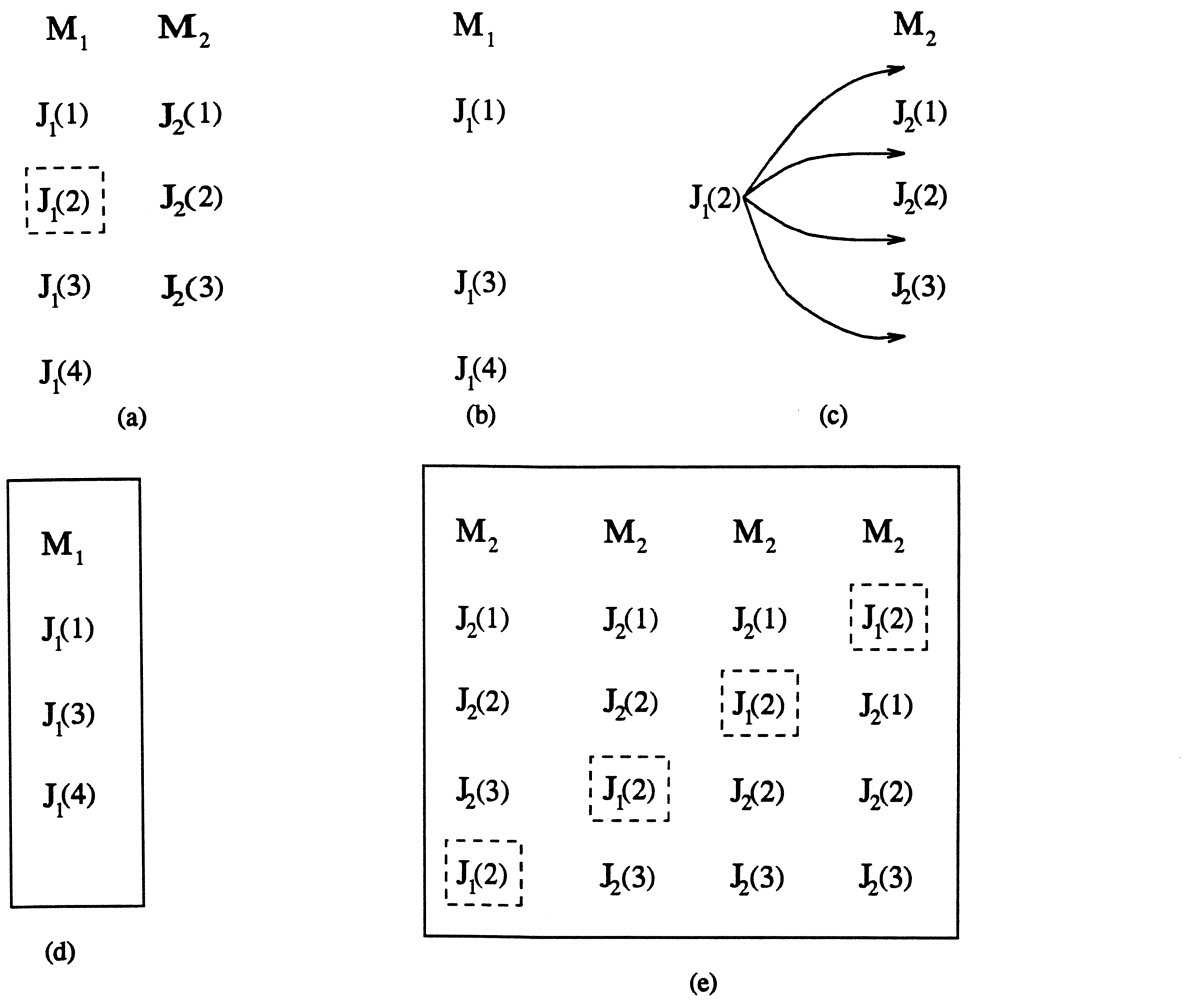


Figura 1: Inserção da tarefa $J_1(2)$ na máquina 2

forma. Examinam-se todas as trocas possíveis de duas tarefas entre duas máquinas. Da mesma forma que nos movimentos de inserção, as tarefas que estão sendo trocadas são testadas em todas as posições das respectivas máquinas. No parágrafo seguinte é apresentado um exemplo das trocas. Seja n_i a quantidade de tarefas em M_i. O tamanho da vizinhança para os movimentos de troca é dado por

$$\sum_{i=1}^{m-1} \sum_{j=i+1}^{m} (n_i n_j)^2$$

Um limitante superior fraco para essa vizinhança é,

$$\left\lfloor \left(\frac{n}{2}\right)^4 \right\rfloor \frac{m(m-1)}{2}$$

onde $\lfloor b \rfloor$ é o maior inteiro menor ou igual a b.

Considerando-se que cada máquina processa n/m tarefas e n é múltiplo de m, então a vizinhança formada por esses movimentos de troca tem tamanho de

$$(\frac{n^4}{m^4})(\frac{m(m-1)}{2})$$

Um exemplo do movimento de troca pode ser observado a seguir. Na Figura 2, sejam dois programas ilustrados em $2(a)$, $S_1 = \{J_1(1), J_1(2), J_1(3), J_1(4)\}$ e $S_2 = \{J_2(1), J_2(2), J_2(3)\}$. Deseja-se fazer a troca da tarefa $J_1(2)$ com a tarefa $J_2(3)$.

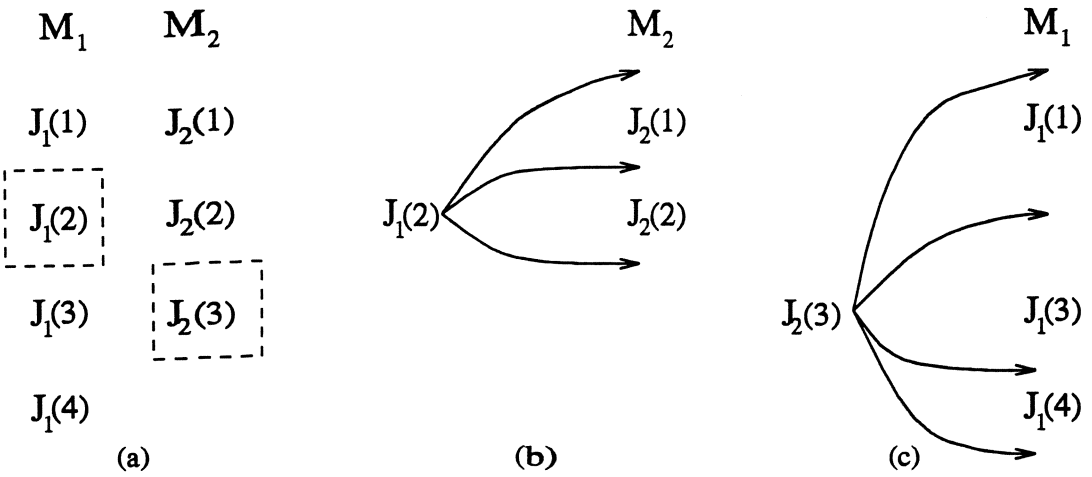


Figura 2: Troca da tarefa $J_1(2)$ com a tarefa $J_2(3)$

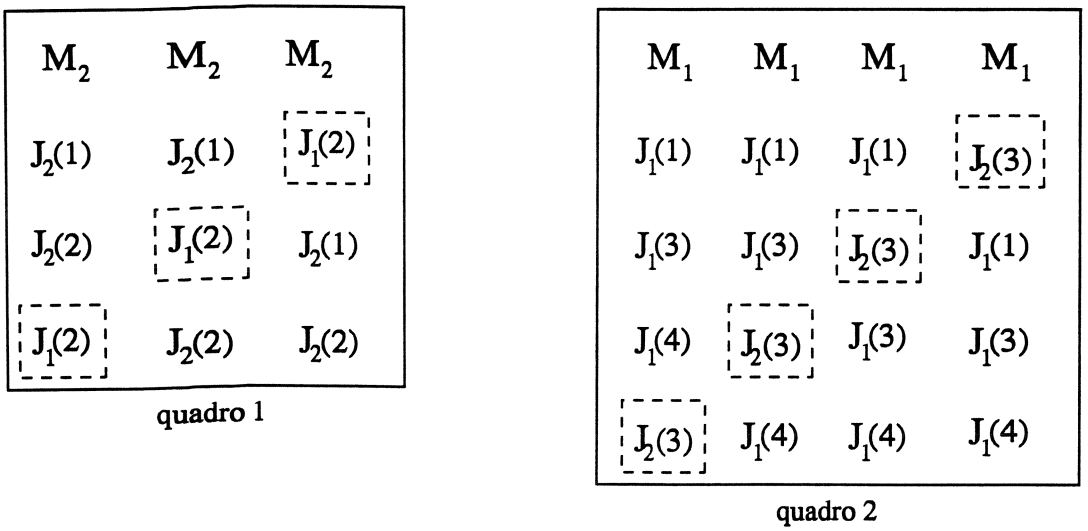


Figura 3: Conjunto de soluções gerado pela troca de $J_1(2)$ com $J_2(3)$

Inicialmente, $J_1(2)$ e $J_2(3)$ são retiradas de suas respectivas máquinas. $J_1(2)$ é então inserida em todas as posições da máquina 2 (Figura $(2b)$), gerando as soluções do quadro 1 da Figura 3. Da mesma forma, $J_2(3)$ é testada em todas as posições da máquina

1 (Figura (2c)), gerando as soluções do quadro 2 da Figura 3. Portanto, o número de soluções que podem ser alcançadas (tamanho da vizinhança) ao se trocar $J_1(2)$ com $J_2(3)$ é $3 * 4 = 12$. Observando os programas da Figura 3, é possível notar que o atraso dos programas do quadro 1 não dependem dos programas do quadro 2. Portanto, uma vez calculado o atraso para os $3 + 4 = 7$ programas da Figura 3, não é necessário recalcular o atraso para cada programa que compõe as 12 soluções geradas pela trocas de $J_1(2)$ com $J_2(3)$.

O tamanho da vizinhança completa de uma dada solução corrente é a soma do tamanho de vizinhança das trocas e inserções. O cálculo do tamanho da vizinhança é detalhado no Apêndice A1.

Complexidade Computacional

A complexidade computacional da avaliação da vizinhança é $O(n^3m) + O(n^2m^2)$, e sua dedução é detalhada no Apêndice A3. Pode-se observar que a complexidade é menor que o tamanho da vizinhança. Isso se deve à independência do cálculo do atraso das máquinas e ao uso de matrizes que armazenam informações da iteração anterior sobre as trocas (veja Apêndice A2.). De nosso conhecimento a estrutura de dados e o cálculo da complexidade computacional utilizados foram propostos neste trabalho e não encontrou-se literatura sobre estes tópicos.

2.2.2 Regra de Ativação Tabu e Duração Tabu.

Diferente da busca local, a busca tabu modifica a vizinhança $V(x)$ utilizando estruturas especiais de memória. Uma dessas estruturas é a memória de curto de prazo, que é fundamental para se determinar a vizinhança modificada $V^*(x)$. A memória de curto prazo baseia-se na história recente da busca. O principal mecanismo para explorar memória de curto prazo na busca tabu é classificar um subconjunto de movimentos numa vizinhança como proibidos (ou tabu), através de regras de ativação. A regra de ativação tabu utilizada foi a seguinte. Sempre que uma tarefa mudar de uma máquina origem para uma máquina destino (seja por um movimento de troca ou inserção) ela não poderá sair dessa última por um determinado número de iterações, ou seja, qualquer movimento envolvendo esta tarefa será considerado tabu. Fica implícito, então, que o atributo do movimento é caracterizado pela identidade da tarefa movida.

A busca tabu deve encorajar flexibilidade. Entretanto, não é sempre verdade que a condição tabu menos restritiva é a melhor. Foram testados outros critérios para deixar o movimento tabu. Por exemplo, se um movimento é executado, a tarefa fica proibida de retornar para a máquina de origem, mas pode ir para qualquer outra máquina. Este critério é muito menos restritivo que o adotado, e não produziu resultados tão bons. A eficiência da memória de curto prazo é influenciada diretamente pela seleção do atributo, a regra que torna movimentos tabu, o tempo que um elemento permanece tabu-ativo, e a estratégia que permite movimentos tabu serem realizados sob situações especiais.

O próximo passo então, é determinar por quanto tempo um movimento deve permanecer tabu, ou seja, a **duração tabu**. Uma aproximação natural é selecionar a duração tabu de acordo com o tamanho do problema. Uma duração tabu apropriada depende da característica da regra de ativação empregada, onde regras mais restritivas são geralmente combinadas com uma duração tabu mais curta.

A duração tabu é um requisito fundamental para o bom desempenho do algoritmo. Uma duração tabu pequena pode gerar ciclos ao passo que uma duração tabu muito grande pode proibir um número excessivo de movimentos, restringindo a busca a movimentos que não permitem melhorias substanciais da função objetivo. A duração tabu ideal fica na faixa intermediária entre esses dois extremos. Em geral, uma duração tabu curta permite explorar soluções "próximas" de um ótimo local enquanto uma duração longa é importante para se afastar de um ótimo local. Variar a duração tabu durante a busca, ao invés de mantê-la constante, oferece uma maneira de induzir um balanço entre examinar uma região bem de perto e mover-se para diferentes partes do espaço solução. Existem muitas formas de se implementar uma duração tabu dinâmica. Essas implementações podem ser classificadas em **duração tabu dinâmica aleatória** e **duração tabu dinâmica sistemática**. Detalhes sobre tipos de duração tabu podem ser encontradas em Laguna(1994).

Para o problema estudado, foi adotada uma duração tabu **dinâmica aleatória**. Neste caso, a duração tabu t é selecionada por uma distribuição uniforme entre dois parâmetros a e b, representada por $U(a,b)$. Um novo valor de t é escolhido para o movimento que se torna tabu em cada iteração. A questão agora é como escolher os parâmetros a e b. Através de experimentos computacionais obteve-se uma estimativa da duração tabu que tornasse, em média, todos os movimentos tabu, de tal forma que não existissem mais movimentos que pudessem ser selecionados. Segundo essa estimativa, tal fato ocorre com uma duração tabu de

$$\frac{n}{m}(m-1)$$

em média, obtida experimentalmente. Foi dado então uma redução a esse número, tomando uma porcentagem dele. Observou-se que os movimentos de inserção são bem menos freqüentes que os de troca. Com a finalidade de incentivar movimentos de inserção, foram utilizados dois tipos de duração tabu diferentes para os movimentos de inserção e troca. Após testes exaustivos foram obtidos bons resultados com as seguintes duração tabu:

Para trocas: $U(a,b)$, onde

$$a = \frac{n}{m}(m-1)0.8 - \frac{n}{m} \qquad b = \frac{n}{m}(m-1)0.8 + \frac{n}{m}$$

Para inserções: $U(a,b)$, onde

$$a = \frac{n}{m}(m-1)0.5 - \frac{n}{m} \qquad b = \frac{n}{m}(m-1)0.5 + \frac{n}{m}.$$

A estrutura computacional usada para a duração tabu consiste de um vetor unidimensional de n posições, que indica até quando a tarefa i deve ficar tabu. Esse vetor é atualizado a cada iteração. Suponha que a iteração atual é a iteração k, então $duracao\_tabu[i] = k + U(a,b)$

Restrições tabu não são invioláveis em todas as circunstâncias. Uma condição que permite suprimir a condição tabu é chamada de `critério de aspiração`. Dessa forma, evita-se que determinados movimentos considerados bons ou influentes sejam ignorados. O critério de aspiração adotado aqui consiste em retirar as restrições tabu sempre que a realização do movimento resultar em uma solução superior à melhor solução já encontrada.

A cada iteração todos os movimentos são avaliados, levando-se em conta o seu status tabu e o critério de aspiração. Seleciona-se o candidato admissível com melhor valor de movimento. O movimento é executado e a duração tabu é atualizada. O algoritmo pára após 300 iterações ou se tempo de CPU superar 1200 segundos.

Ao invés de se escolher o melhor movimento da vizinhança $V^*(x)$, pode-se alterar a política de seleção de movimentos escolhendo o primeiro movimento que melhore a função objetivo com relação a x. Nesse caso, existe grande chance de não ser preciso avaliar a vizinhança inteira, portanto, a escolha pode ser mais rápida que na estratégia do melhor movimento, onde obrigatoriamente a vizinhança inteira deve ser analisada. No caso, existe um compromisso entre o tempo gasto para se avaliar uma vizinhança e a qualidade da solução obtida. Para o problema aqui tratado, verificou-se que a estratégia mais agressiva do melhor movimento, apresenta soluções de qualidade superior a do primeiro movimento num mesmo intervalo de tempo.

Para finalizar a seção apresenta-se na Figura 4 um diagrama de blocos da busca tabu. Na seção seguinte serão apresentadas as duas estratégias de diversificação aplicadas à busca tabu. Daqui por diante, o procedimento de busca tabu sem diversificação tal como foi apresentado nesta seção, será denominado `busca tabu simples`.

2.3 Diversificação

Estratégias de diversificação têm a finalidade de direcionar a busca para novas regiões. Em geral, são baseadas na modificação das regras de seleção de movimentos para favorecer aqueles que não são freqüentemente escolhidos (usando funções de penalidade, por exemplo).

Duas estratégias de diversificação foram propostas para o problema: diversificação baseada em distância de movimento e diversificação com penalização de movimentos freqüentes. No primeiro caso, não foi utilizada memória de longo prazo, e a idéia da diversificação consiste em incentivar movimentos de inserção. Na diversificação com penalização, utiliza-se memória de longo prazo para armazenar soluções de elite e a freqüência dos movimentos. As duas estratégias de diversificação serão discutidas a seguir.

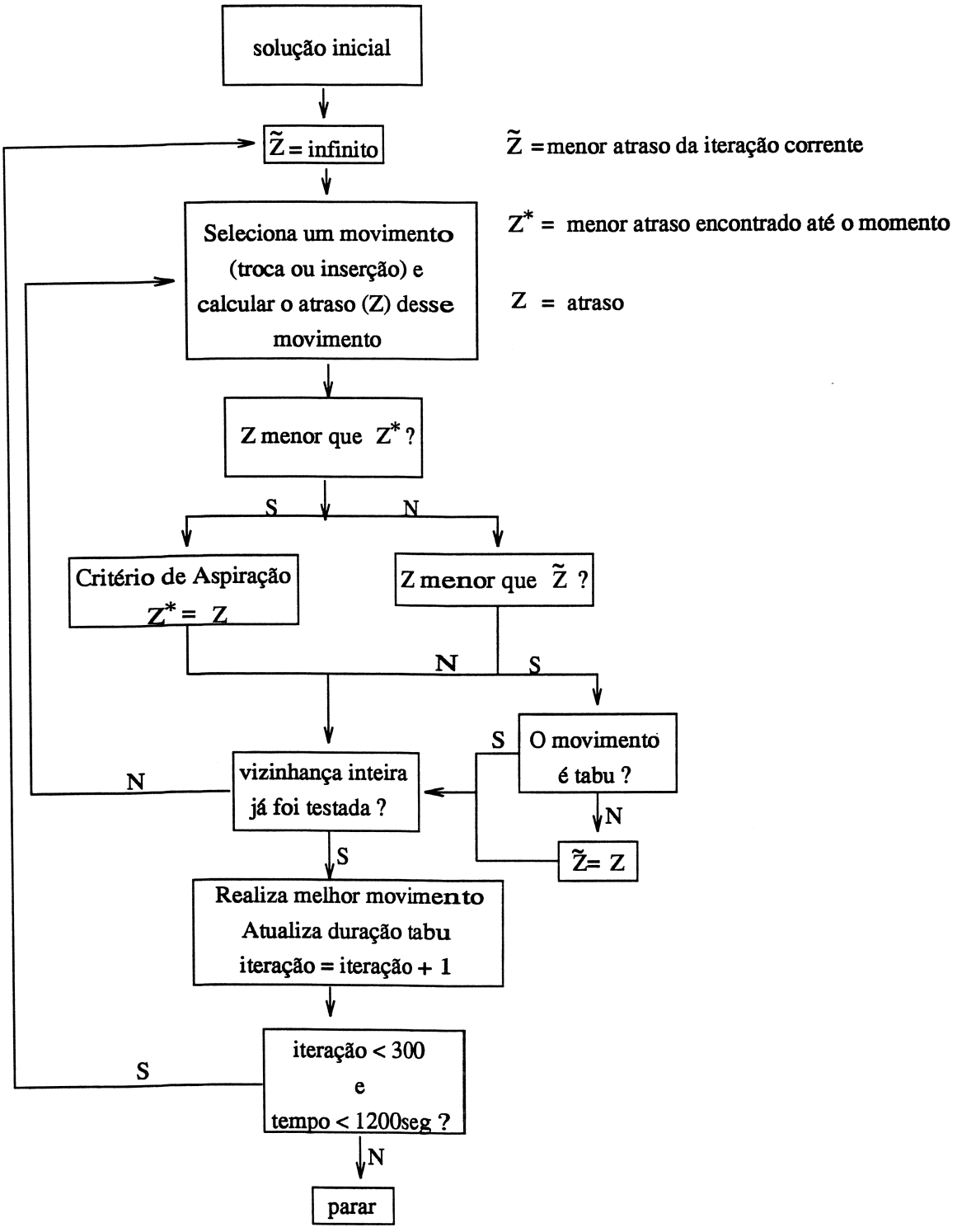


Figura 4: Fluxograma da busca tabu simples

2.3.1 Diversificação com Movimentos de Inserção

As estratégias de diversificação são úteis quando soluções melhores só podem ser alcançadas ao se cruzar barreiras na topologia do espaço de solução. A noção de **distância do movimento** está associada ao grau no qual a solução corrente é modificada ao se executar um determinado movimento. Movimentos de grandes distância produzem maiores mudanças estruturais na solução, e têm uma tendência de produzir soluções de pior qualidade. Este caminho por tais soluções é necessário para se obter soluções de melhor qualidade.

Uma interpretação simples mas poderosa deste conceito foi usada para incorporar um elemento de diversificação na busca tabu (**Barnes e Laguna, (1993)**). Observou-se que em média, a busca tabu realiza um número muito maior de movimentos de troca do que de inserção. Uma provável justificativa é que os movimentos de troca não modificam a cardinalidade da partição da solução corrente, e em geral, eles possuem uma distância de movimento menor que os movimentos de inserção. Logo, os movimentos de troca são selecionados mais freqüentemente. Portanto, em adição à regra de decisão tradicional de selecionar o melhor movimento a cada iteração, adota-se uma outra regra de seleção de movimentos com a finalidade de incentivar as inserções. Seja Z^* o valor do menor atraso obtido no decorrer da busca até o momento. Após um período de 10 iterações sem encontrar uma solução melhor que a melhor que Z^*, proibe-se a seleção de movimentos de troca que não melhoram Z^*, e o melhor movimento de inserção é executado. Um fluxograma desta estratégia de diversificação é apresentado na Figura 5. Esta estratégia tem o efeito de perturbar a solução corrente, e tende a levar a busca para uma região diferente daquela atualmente explorada.

2.3.2 Diversificação com Penalidade

Uma estratégia de diversificação muito utilizada é a penalização. Usando memória de longo prazo, é possível armazenar a freqüência dos movimentos ao longo da busca, e dessa forma, tentar evitar movimentos executados muitas vezes e favorecer movimentos pouco freqüentes. Para tanto, será utilizada uma matriz triangular inferior, de dimensão nxn, denominada *matriz de freqüência*, e que armazena na posição (i, j) quantas vezes a tarefa i foi trocada com a tarefa j. Quando o movimento é de inserção, a freqüência é armazenada na diagonal da matriz. A matriz de freqüência é utilizada na função de penalidade de movimentos. Quando a diversificação é ativada, os movimentos envolvendo as tarefas i e j passam a ser avaliados através da função de penalidade,

$$movimento\,penalizado(i,j) = valor\,do\,movimento(i,j) + \alpha * penalidade(i,j) \qquad (1)$$

onde α é o parâmetro de diversificação que deve ser ajustado e $penalidade(i,j)$ é uma medida relativa da freqüência. Nesse caso em particular, $penalidade(i,j)$ é o elemento (i,j) da matriz de freqüência dividido pelo maior elemento da matriz. Portanto, $penalidade(i,j)$ varia entre 0 e 1.

Foram usados dois fatores de penalidade distintos para movimentos de troca (α = 0.3*(*solução da heurística construtiva*)) e inserção (α = 0.05*(*solução da heurística construtiva*)), determinados após testes exaustivos. Essa distinção deve-se ao fato da quantidade de movimentos de inserção ser muito menor que os movimentos de troca. Portanto ao designarmos um peso menor aos movimentos de inserção, esses movimentos são encorajados e dessa forma, pode-se examinar espaço de soluções pouco explorados.

A diversificação usando penalidade é ativada sempre que a busca tabu não consegue encontrar melhores soluções. Através de testes computacionais foi determinado que a penalização deve ter início a cada 60 iterações sem melhoria. Parte-se então da melhor solução obtida até o momento, a duração tabu é zerada e os movimentos passam a ser avaliados pela função de penalidade (1). Quando a solução estiver suficientemente longe do mínimo local (no caso, após 50 iterações com diversificação), ou se uma nova solução for encontrada durante este estágio, o procedimento de diversificação é abandonado e a busca tabu prossegue normalmente.

Este reinício partindo da melhor solução garante que a diversificação não ocorra em torno de uma região ruim, e comprovou ser mais eficiente que partir de um ponto qualquer da busca. É importante notar que existe o risco de não se sair dessa região inicial (essa região pode ser um "vale"). Portanto, para assegurar o sucesso da diversificação é fundamental determinar um período de tempo adequado para a fase de diversificação. Esse período de diversificação deve ser suficientemente longo para que se consiga explorar novas regiões, mas não pode ser tão longo pois isso pode acarretar uma degradação muito grande na qualidade da solução. Um fluxograma da penalização é apresentado na Figura 6.

cont = contador da diversificação

Inicialmente cont=0

Z* é a melhor solução encontrada até o momento.

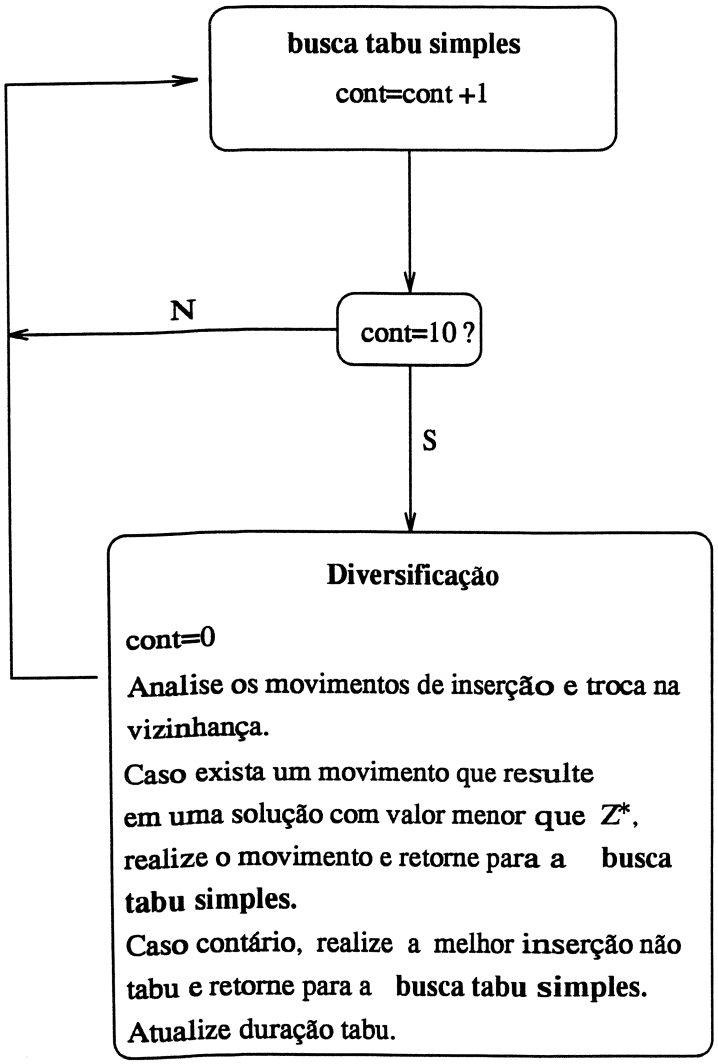


Figura 5: Fluxograma da busca tabu com diversificação com inserções

Z* é a melhor solução encontrada até o momento.

iter* = iteração em que encontrou Z*

iter_diver = iteração na qual o procedimento de **Diversificação** foi interrompido.

Inicialmente, iteracao_corrente= iter_diver = iter* = 0

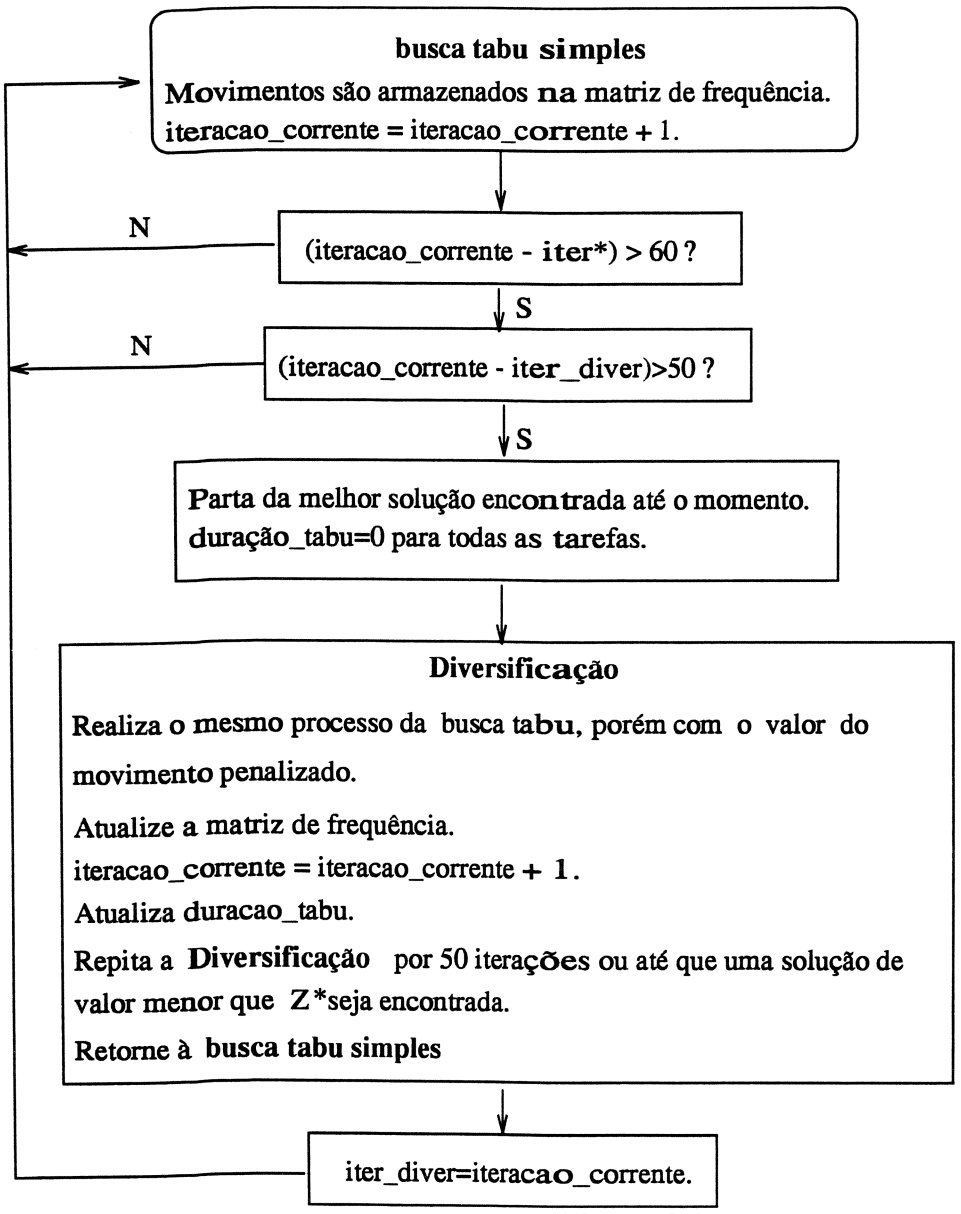


Figura 6: Fluxograma da busca tabu com diversificação com penalidade

CAPÍTULO 3

Experimentos Computacionais

Neste capítulo apresentam-se os experimentos computacionais realizados com a finalidade de avaliar o desempenho das heurísticas propostas. Na seção 3.1 descreve-se o método usado para gerar as instâncias de teste. Na seção 3.2 apresentam-se os resultados da busca tabu, assim como das duas estratégias de diversificação propostas. A qualidade das heurísticas foi avaliada através de limitantes inferiores gerados por relaxação lagrangeana. Na seção 3.4 apresentam-se as conclusões deste trabalho.

3.1 Geração do conjunto de teste

As instâncias foram geradas com os seguintes tamanhos de máquinas e tarefas: $m = 2, 3, 5, 7, 10$ e $n = 20, 50, 100, 150$, respectivamente. Os tempos de processamento foram gerados por uma distribuição uniforme entre 1 e 100. As datas de entrega são obtidas por uma distribuição uniforme entre $P(1 - \tau - R/2)$ e $P(1 - \tau + R/2)$. τ é o parâmetro de atraso médio e quanto maior o seu valor, menor é a média da distribuição e portanto, mais apertadas serão as datas de entrega. R é o parâmetro que determina a dispersão da distribuição que gera as datas de entrega, quanto maior for R, maior será a dispersão. O parâmetro P é a soma de todos os tempos de processamento dividida pelo número de máquinas (Potts e Van Wassenhove (1982), Koulamas(1994)).

Os valores de τ e R usados foram $\tau = 0,4$; $0,6$; $0,8$ e $R = 0,4$; $0,7$; $1,0$. Existem então nove categorias de instâncias formadas pelas combinações de τ e R. Para cada categoria são geradas cinco instâncias, logo, como existem cinco quantidades diferentes de máquinas e quatro de tarefas, ao todo são testadas $9 \times 5 \times 5 \times 4 = 900$ instâncias. A Tabela 1 a seguir classifica as instâncias em categorias de acordo com os valores de τ e R.

Tabela 1 - Categoria das instâncias geradas

Categoria	τ	R	$U(a,b)$
1	0,4	0,4	$U(0,40P;\ 0,80P)$
2	0,4	0,7	$U(0,25P;\ 0,95P)$
3	0,4	1,0	$U(0,10P;\ 1,10P)$
4	0,6	0,4	$U(0,20P;\ 0,60P)$
5	0,6	0,7	$U(0,05P;\ 0,75P)$
6	0,6	1,0	$U(0,00P;\ 0,90P)$
7	0,8	0,4	$U(0,00P;\ 0,40P)$
8	0,8	0,7	$U(0,00P;\ 0,55P)$
9	0,8	1,0	$U(0,00P;\ 0,70P)$

A Tabela 1 classifica as instâncias por categorias em relação aos valores de R e τ, exibidos na segunda e terceira colunas. A coluna $U(a,b)$ indica os parâmetros a e b da distribuição uniforme, usada para gerar as datas de entrega.

Todos os programas computacionais foram codificados na linguagem C e executados em estações de trabalho SUN modelo SPARCstation classic.

3.2 Resultados Computacionais

Neste trabalho, a fim de solucionar o problema de atraso em máquinas paralelas, foram propostas uma busca tabu simples e duas estratégias de diversificação, diversificação com penalização e diversificação com movimentos de inserção.

O resultado gerado pelas heurísticas foi comparado com um limitante inferior proposto por Luh at al. (1990), detalhado no Apêndice A4. O limitante inferior não foi calculado para todas instâncias devido ao alto custo computacional. Ele só pôde ser calculado completamente para instâncias de 20 e 50 tarefas. Para as instâncias com 100 e 150 tarefas, foi calculado apenas um limitante para uma instância de cada categoria. Para as instâncias de 20 e 50 tarefas, os tempos variaram de 83 segundos até 20 minutos por instância. Para 100 e 150 tarefas, utilizaram-se 3 horas para cada instância. Foram obtidos ao todo, 540 limitantes inferiores.

A seguir, apresentam-se as tabelas mostrando o desempenho das três heurísticas, busca tabu simples, diversificação com penalização e diversificação com inserções, em relação à heurística construtiva e ao limitante inferior. Na coluna 1 encontra-se a categoria a que as instâncias pertencem. A coluna 2 apresenta a média dos cinco atrasos resultantes da heurística construtiva para cada categoria. As colunas 3, 4 e 5 indicam quanto a busca tabu simples, a diversificação com penalidade e a diversificação com inserções, respectivamente, melhoraram em relação à heurística construtiva. Essa melhoria em porcentagem é calculada por:

$$100\frac{(hc-h)}{hc}$$

onde hc e h são os atrasos das heurística construtiva e heurística com busca tabu, respectivamente. A coluna 6 apresenta o pior desempenho entre as três heurísticas aplicadas

às instâncias de mesma categoria, em relação à heurística construtiva. A coluna 7 indica a distância das soluções heurísticas, em porcentagem, em relação ao limitante inferior calculada da seguinte forma,

$$100\frac{(h-L)}{L}$$

onde h é o atraso da heurística e L é o valor do limitante inferior. Como já mencionado anteriormente, não foi possível calcular o limitante para todos as instâncias. Logo, para as instâncias de 100 e 150 tarefas foi calculado somente um limitante para uma instância de cada categoria. A coluna *pior gap* representa a maior distância encontrada entre a heurística e o limitante inferior naquela categoria.

Tabela 2 - Resultados obtidos para instâncias com 2 máquinas e 20 tarefas

m x n - 2 x 20							
Categoria	Heurística Construtiva	Melhoria em (%)	Melhoria 1 em (%)	Melhoria 2 em (%)	Pior Melh	Gap em (%)	Pior Gap
1	23,61	11,29	11,29	11,29	3,43	2,64	4,64
2	17,13	26,83	26,83	26,83	3,14	4,77	8,80
3	10,65	48,87	48,87	48,87	10,00	1,36	6,78
4	61,15	5,64	5,64	5,64	2,39	0,62	1,46
5	62,71	6,59	6,59	6,59	3,07	0,60	1,08
6	44,08	8,82	8,82	8,82	2,65	1,43	2,34
7	110,68	1,23	1,23	1,23	0,05	0,09	0,20
8	132,14	1,07	1,07	1,07	0,13	0,10	0,25
9	103,27	2,85	2,89	2,85	0,76	0,44	1,12

Tabela 3 - Resultados obtidos para instâncias com 2 máquinas e 50 tarefas

m x n - 2 x 50							
Categoria	Heurística Construtiva	Melhoria em (%)	Melhoria 1 em (%)	Melhoria 2 em (%)	Pior Melh	Gap em (%)	Pior Gap
1	38,90	10,28	10,28	10,26	5,61	3,59	5,08
2	8,67	25,60	25,60	24,96	11,52	42,91	129,73
3	7,40	61,84	61,84	61,93	0,00	1,56	6,89
4	98,50	4,14	4,16	4,14	1,63	1,33	2,28
5	104,68	4,96	5,01	5,01	2,51	1,57	2,22
6	68,65	9,39	9,19	9,35	5,11	1,86	3,77
7	236,07	1,03	1,04	1,03	0,63	0,58	0,67
8	246,13	0,92	0,92	0,92	0,16	0,51	0,58
9	230,63	1,14	1,17	1,18	0,54	0,75	1,07

Tabela 4 - Resultados obtidos para instâncias com 2 máquinas e 100 tarefas

m x n - 2 x 100							
Categoria	Heurística Construtiva	Melhoria em (%)	Melhoria 1 em (%)	Melhoria 2 em (%)	Pior Melh	Gap em (%)	Pior Gap
1	60,75	6,07	5,99	6,13	4,89	31,31	31,31
2	14,23	25,31	25,30	25,62	6,36	157,37	163,16
3	0,10	20,00	20,00	20,00	0,00	0,00	0,00
4	210,09	3,24	3,24	3,25	2,09	6,32	6,34
5	172,84	3,19	3,18	3,17	1,69	3,36	3,63
6	175,55	2,00	1,92	1,95	1,62	5,01	5,12
7	499,14	0,79	0,80	0,80	0,20	2,10	2,11
8	433,29	0,53	0,54	0,55	0,36	1,54	1,61
9	397,87	0,52	0,50	0,53	0,31	2,44	2,48

Tabela 5 - Resultados obtidos para instâncias com 2 máquinas e 150 tarefas

m x n - 2 x 150							
Categoria	Heurística Construtiva	Melhoria em (%)	Melhoria 1 em (%)	Melhoria 2 em (%)	Pior Melh	Gap em (%)	Pior Gap
1	84,91	4,17	4,17	4,20	3,40	0	0,00
2	16,22	13,87	13,86	13,77	7,67	0	0,00
3	0,00	0,00	0,00	0,00	0,00	0,00	0,00
4	321,51	3,14	3,15	3,15	2,40	39,29	39,29
5	241,19	4,98	5,04	4,99	1,30	16,29	16,46
6	247,59	3,18	3,24	3,17	1,37	35,29	35,67
7	730,87	0,63	0,63	0,60	0,38	1,12	1,13
8	672,50	0,34	0,35	0,36	0,28	1,80	1,82
9	612,07	0,27	0,26	0,27	0,15	1,96	1,97

Tabela 6 - Resultados obtidos para instâncias com 3 máquinas e 20 tarefas

m x n - 3 x 20							
Categoria	Heurística Construtiva	Melhoria em (%)	Melhoria 1 em (%)	Melhoria 2 em (%)	Pior Melh	Gap em (%)	Pior Gap
1	20,61	17,62	17,62	17,62	11,86	1,48	2,51
2	15,61	24,45	24,35	24,42	9,52	1,34	6,40
3	14,19	59,54	59,54	59,54	27,93	3,70	13,16
4	47,78	7,24	7,24	7,24	2,22	0,39	1,16
5	50,49	9,39	9,36	9,39	1,68	0,42	0,74
6	39,06	13,95	13,95	14,10	8,49	0,24	0,88
7	82,29	1,54	1,54	1,54	0,97	0,07	0,14
8	97,99	2,00	2,00	2,00	1,28	0,10	0,45
9	77,53	3,42	3,46	3,43	1,40	0,20	0,68

Tabela 7 - Resultados obtidos para instâncias com 3 máquinas e 50 tarefas

m x n - 3 x 50							
Categoria	Heurística Construtiva	Melhoria em (%)	Melhoria 1 em (%)	Melhoria 2 em (%)	Pior Melh	Gap em (%)	Pior Gap
1	29,36	11,38	11,31	11,40	6,01	1,98	3,25
2	9,25	38,12	38,12	38,12	19,62	87,58	240,82
3	14,39	86,22	86,10	86,20	39,05	1,11	5,48
4	73,81	8,53	8,50	8,48	3,03	0,62	0,91
5	79,06	7,53	7,76	7,62	4,37	0,92	1,90
6	59,84	20,08	19,99	19,98	9,27	1,94	4,01
7	165,99	1,89	1,89	1,89	1,18	0,24	0,35
8	173,54	1,67	1,67	1,67	0,90	0,30	0,45
9	163,28	2,05	2,10	2,10	0,61	0,43	0,71

Tabela 8 - Resultados obtidos para instâncias com 3 máquinas e 100 tarefas

m x n - 3 x 100							
Categoria	Heurística Construtiva	Melhoria em (%)	Melhoria 1 em (%)	Melhoria 2 em (%)	Pior Melh	Gap em (%)	Pior Gap
1	44,42	9,06	9,16	9,26	5,16	5,07	5,07
2	13,56	39,50	39,11	39,16	19,80	5,23	12,11
3	0,79	80,00	80,00	80,00	0,00	0,00	0,00
4	146,68	4,42	4,42	4,42	2,01	1,07	1,19
5	125,21	6,07	6,04	6,10	3,31	1,23	1,29
6	128,12	4,88	4,88	4,96	2,55	2,31	2,38
7	342,04	1,24	1,24	1,23	0,58	0,52	0,54
8	298,01	1,01	1,00	1,00	0,68	1,84	1,85
9	275,08	1,27	1,25	1,27	0,70	0,77	0,86

Tabela 9 - Resultados obtidos para instâncias com 3 máquinas e 150 tarefas

m x n - 3 x 150							
Categoria	Heurística Construtiva	Melhoria em (%)	Melhoria 1 em (%)	Melhoria 2 em (%)	Pior Melh	Gap em (%)	Pior Gap
1	61,09	7,30	7,28	7,28	4,27	9,60	9,60
2	14,36	26,30	26,32	26,30	15,43	20,69	20,69
3	0,32	60,00	60,00	60,00	0,00	0,00	0,00
4	218,89	3,00	3,03	2,97	1,62	7,13	7,15
5	169,98	6,39	6,41	6,38	4,51	2,96	3,17
6	173,39	3,55	3,53	3,49	2,37	4,95	5,29
7	498,02	1,19	1,20	1,19	0,75	2,72	2,75
8	457,45	0,61	0,60	0,61	0,42	3,20	3,20
9	418,11	0,73	0,74	0,74	0,45	4,69	4,69

Tabela 10 - Resultados obtidos para instâncias com 5 máquinas e 20 tarefas

m x n - 5 x 20							
Categoria	Heurística Construtiva	Melhoria em (%)	Melhoria 1 em (%)	Melhoria 2 em (%)	Pior Melh	Gap em (%)	Pior Gap
1	18,92	17,90	18,00	18,17	12,72	1,01	2,48
2	18,17	22,15	22,26	22,26	16,62	0,78	1,69
3	16,56	43,64	43,64	43,64	17,53	0,50	1,24
4	38,38	7,84	7,84	7,84	1,79	0,15	0,53
5	41,27	10,00	9,98	10,00	5,24	0,08	0,24
6	32,33	11,10	11,10	11,10	7,56	0,26	0,66
7	59,80	1,45	1,45	1,45	0,58	0,03	0,17
8	70,41	2,10	2,10	2,10	0,32	0,00	0,00
9	58,35	5,21	5,21	5,21	1,73	0,08	0,27

Tabela 11 - Resultados obtidos para instâncias com 5 máquinas e 50 tarefas

m x n - 5 x 50							
Categoria	Heurística Construtiva	Melhoria em (%)	Melhoria 1 em (%)	Melhoria 2 em (%)	Pior Melh	Gap em (%)	Pior Gap
1	23,77	18,66	18,75	18,75	15,59	1,39	2,21
2	12,43	54,99	55,15	55,09	41,00	13,16	29,92
3	15,96	77,03	76,43	76,52	27,76	11,05	86,00
4	50,58	8,54	8,60	8,64	4,73	0,88	1,27
5	60,80	13,19	13,04	13,17	9,70	1,20	1,83
6	44,07	17,64	17,64	17,65	14,49	1,81	2,85
7	110,66	3,06	3,05	3,06	2,20	0,35	0,46
8	115,29	2,56	2,56	2,59	0,97	0,28	0,62
9	109,41	3,33	3,33	3,28	1,48	0,41	0,92

Tabela 12 - Resultados obtidos para instâncias com 5 máquinas e 100 tarefas

m x n - 5 x 100							
Categoria	Heurística Construtiva	Melhoria em (%)	Melhoria 1 em (%)	Melhoria 2 em (%)	Pior Melh	Gap em (%)	Pior Gap
1	32,09	14,40	14,47	14,50	11,05	26,34	27,13
2	12,78	46,85	47,05	46,91	25,32	20,73	21,76
3	6,26	100,00	100,00	100,00	100,00	0,00	0,00
4	95,94	6,43	6,39	6,44	4,22	4,30	4,43
5	87,61	10,33	10,49	10,28	7,84	2,47	2,78
6	89,08	8,04	8,05	8,09	5,83	3,93	4,19
7	216,34	2,03	2,07	2,09	1,11	1,16	1,18
8	190,13	2,05	2,05	2,06	1,49	1,26	1,32
9	176,96	2,54	2,55	2,60	1,43	1,49	1,68

Tabela 13 - Resultados obtidos para instâncias com 5 máquinas e 150 tarefas

m x n - 5 x 150							
Categoria	Heurística Construtiva	Melhoria em (%)	Melhoria 1 em (%)	Melhoria 2 em (%)	Pior Melh	Gap em (%)	Pior Gap
1	41,96	11,60	11,61	11,66	7,94	23,13	23,42
2	13,95	42,35	42,39	42,34	27,08	17,10	17,22
3	4,10	100,00	100,00	100,00	100,00	0,00	0,00
4	141,24	5,56	5,57	5,56	4,53	1,25	1,36
5	111,88	8,52	8,49	8,43	4,25	1,10	1,16
6	117,04	6,68	6,69	6,72	4,58	1,44	1,65
7	309,49	1,67	1,64	1,65	0,90	0,42	0,47
8	285,67	1,18	1,18	1,18	1,05	0,40	0,40
9	262,55	1,55	1,50	1,55	0,73	0,54	0,58

Tabela 14 - Resultados obtidos para instâncias com 7 máquinas e 20 tarefas

m x n - 7 x 20							
Categoria	Heurística Construtiva	Melhoria em (%)	Melhoria 1 em (%)	Melhoria 2 em (%)	Pior Melh	Gap em (%)	Pior Gap
1	19,85	18,11	18,38	18,38	13,01	0,40	3,47
2	19,92	21,33	21,33	21,33	14,67	0,21	0,63
3	20,06	35,65	35,61	35,59	24,38	0,00	0,65
4	34,95	6,67	6,67	6,67	3,72	0,03	0,13
5	37,08	6,06	6,06	6,06	4,14	0,06	0,18
6	30,74	12,23	12,27	12,27	7,48	0,11	0,33
7	51,16	2,35	2,35	2,35	1,14	0,00	0,00
8	59,41	2,65	2,65	2,65	0,00	0,00	0,00
9	49,68	4,08	4,08	4,08	2,28	0,04	0,21

Tabela 15 - Resultados obtidos para instâncias com 7 máquinas e 50 tarefas

m x n - 7 x 50							
Categoria	Heurística Construtiva	Melhoria em (%)	Melhoria 1 em (%)	Melhoria 2 em (%)	Pior Melh	Gap em (%)	Pior Gap
1	20,77	20,21	20,25	20,04	15,34	1,09	2,81
2	13,92	56,04	56,13	56,34	41,76	5,30	10,29
3	18,24	64,07	64,25	65,00	24,73	33,48	211,76
4	41,39	9,75	9,66	9,79	7,48	1,02	2,19
5	49,71	11,34	11,31	11,37	7,41	0,60	1,21
6	37,46	14,67	14,78	14,80	11,79	1,51	2,87
7	86,28	3,15	3,15	3,16	2,28	0,27	0,51
8	90,80	3,78	3,74	3,79	1,97	0,14	0,30
9	86,64	4,38	4,37	4,36	1,73	0,34	0,66

Tabela 16 - Resultados obtidos para instâncias com 7 máquinas e 100 tarefas

m x n - 7 x 100							
Categoria	Heurística Construtiva	Melhoria em (%)	Melhoria 1 em (%)	Melhoria 2 em (%)	Pior Melh	Gap em (%)	Pior Gap
1	25,78	14,93	14,93	15,04	10,51	4,62	4,89
2	14,82	55,51	55,38	55,07	42,41	9,89	12,42
3	9,88	99,23	99,23	99,23	98,17	0,00	0,00
4	73,87	7,29	7,24	7,31	6,01	0,88	0,98
5	69,76	11,07	11,27	11,30	8,08	0,92	1,12
6	71,17	9,25	9,28	9,20	8,13	2,33	2,77
7	161,86	2,17	2,18	2,19	1,25	0,33	0,46
8	143,18	2,42	2,44	2,40	1,72	0,51	0,61
9	135,19	3,77	3,71	3,81	1,82	0,47	0,71

Tabela 17 - Resultados obtidos para instâncias com 7 máquinas e 150 tarefas

m x n - 7 x 150							
Categoria	Heurística Construtiva	Melhoria em (%)	Melhoria 1 em (%)	Melhoria 2 em (%)	Pior Melh	Gap em (%)	Pior Gap
1	33,37	14,28	14,25	14,43	12,17	4,25	4,40
2	13,06	43,20	42,86	42,81	23,68	5,90	6,99
3	9,06	100,00	100,00	100,00	100,00	0,00	0,00
4	106,29	6,27	6,22	6,26	5,31	0,54	0,84
5	89,81	12,81	12,83	12,79	8,51	1,29	1,39
6	92,51	8,73	8,69	8,66	6,97	2,21	2,93
7	228,73	1,92	1,92	1,94	1,40	0,33	0,45
8	212,23	1,76	1,75	1,76	1,36	0,26	0,32
9	195,42	1,98	1,96	1,97	1,04	0,34	0,44

Tabela 18 - Resultados obtidos para instâncias com 10 máquinas e 20 tarefas

m x n - 10 x 20							
Categoria	Heurística Construtiva	Melhoria em (%)	Melhoria 1 em (%)	Melhoria 2 em (%)	Pior Melh	Gap em (%)	Pior Gap
1	20,17	6,71	6,71	6,71	4,48	0,05	0,24
2	21,48	13,90	13,90	13,90	7,90	0,00	0,00
3	21,29	19,20	19,20	19,20	13,46	0,00	0,00
4	32,81	0,11	0,11	0,11	0,00	0,00	0,00
5	34,62	2,94	2,94	2,94	0,00	0,00	0,00
6	28,99	7,10	7,10	7,10	4,38	0,04	0,21
7	44,05	0,00	0,00	0,00	0,00	0,00	0,00
8	50,61	0,00	0,00	0,00	0,00	0,00	0,00
9	43,56	0,59	0,59	0,59	0,00	0,00	0,00

Tabela 19 - Resultados obtidos para instâncias com 10 máquinas e 50 tarefas

m x n - 10 x 50							
Categoria	Heurística Construtiva	Melhoria em (%)	Melhoria 1 em (%)	Melhoria 2 em (%)	Pior Melh	Gap em (%)	Pior Gap
1	19,38	20,97	21,17	20,77	15,30	2,10	4,05
2	14,89	51,14	51,36	50,68	40,47	2,53	12,54
3	19,73	48,14	48,35	48,12	28,00	3,36	8,94
4	36,13	11,57	11,50	11,59	7,95	0,25	0,66
5	43,40	12,18	12,11	12,12	11,33	0,52	0,92
6	34,41	16,11	16,14	16,26	14,22	0,88	1,22
7	68,15	2,96	2,94	2,96	2,10	0,10	0,20
8	71,66	2,93	2,94	2,96	1,35	0,07	0,17
9	69,69	5,77	5,80	5,76	3,00	0,17	0,53

Tabela 20 - Resultados obtidos para instâncias com 10 máquinas e 100 tarefas

m x n - 10 x 100							
Categoria	Heurística Construtiva	Melhoria em (%)	Melhoria 1 em (%)	Melhoria 2 em (%)	Pior Melh	Gap em (%)	Pior Gap
1	22,89	21,36	21,15	21,27	19,13	1,67	2,05
2	15,51	55,82	55,89	55,73	50,06	7,31	8,04
3	12,36	95,72	95,92	95,74	93,64	0,00	0,00
4	58,02	9,17	9,09	9,19	7,45	1,01	1,15
5	57,21	12,67	12,62	12,55	7,77	1,17	1,30
6	58,87	11,84	11,71	12,02	9,64	1,74	2,68
7	121,46	2,77	2,75	2,74	1,83	0,28	0,28
8	108,88	3,63	3,63	3,60	2,35	0,48	0,63
9	102,73	4,22	4,20	4,17	2,69	0,54	0,71

Tabela 21 - Resultados obtidos para instâncias com 10 máquinas e 150 tarefas

m x n - 10 x 150							
Categoria	Heurística Construtiva	Melhoria em (%)	Melhoria 1 em (%)	Melhoria 2 em (%)	Pior Melh	Gap em (%)	Pior Gap
1	27,47	18,22	18,10	18,34	14,91	4,88	5,20
2	13,15	49,03	48,92	48,86	37,08	5,91	6,18
3	11,46	99,83	99,83	99,83	99,24	0,00	0,00
4	81,29	8,57	8,56	8,51	7,33	1,07	1,14
5	69,19	12,20	12,19	12,19	10,05	1,30	1,30
6	75,52	12,75	12,47	12,56	9,38	1,61	2,92
7	168,31	2,35	2,33	2,37	1,44	0,32	0,47
8	156,93	2,39	2,41	2,39	1,83	0,38	0,44
9	146,01	3,21	3,21	3,25	1,65	0,44	0,57

Observando as tabelas é possível notar que as maiores melhorias em relação à heurística de construção ocorrem nas instâncias das categorias 1, 2 e 3. Nessas mesmas categorias ocorrem também os maiores gaps. É importante observar que essas categorias possuem o menor valor de atraso. Isso indica que para analisar os resultados não basta considerar apenas a melhoria em porcentagem mas também o valor do atraso.

Foi possível observar também, que para instâncias de categoria com mesmo valor de τ (Tabela 1), a melhoria em relação à heurística construtiva parece crescer à medida que R cresce (Veja, por exemplo, Tabela 6).

As instâncias que apresentaram as maiores melhorias em relação à heurística construtiva foram as de 50 e 100 tarefas e 5 e 7 máquinas.

As três heurísticas tiveram um desempenho bastante parecido. Isso pode ser notado, por exemplo, na Tabela 3. Comparando as três heurísticas em relação à heurística construtiva, a diversificação com inserção foi melhor que as demais heurísticas em 20% dos casos, seguida da diversificação com penalização com 15% e busca tabu simples, que foi a melhor heurística em 14% dos casos. Na média, as três heurísticas apresentaram cerca de 16,28% de melhoria em relação à heurística construtiva, com diferenças na terceira casa decimal. Uma possível explicação para as heurísticas apresentarem resultados tão semelhantes é que a busca tabu simples sem diversificação já se encontra bem próxima dos limitantes inferiores para uma grande variedade de instâncias. A Tabela 22 apresenta o desempenho dos métodos em relação ao limitante inferior, indicando a quantidade de resultados que encontram-se a uma determinada distância, em porcentagem, do limitante inferior. Nesta análise, considerou-se o melhor resultado obtido dentre as três heurísticas. Pode-se observar que em vários casos o valor da heurística foi igual ao limitante inferior, o que implica que o ótimo global foi atingido. Além disso, mais de 65% das instâncias, estão a menos de 1% do limitante inferior. Nos casos onde o limitante inferior foi zero e a heurística teve um resultado diferente de zero, o gap foi contado como acima de 10%.

Tabela 22 - Distribuição das soluções heurísticas em relação aos limitantes inferiores.

$gap = 0$	$0 < gap < 1$	$1 \leq gap < 2$	$2 \leq gap < 3$	$3 \leq gap < 5$	$5 \leq gap < 10$	$gap \geq 10$
27,22%	38,52%	14,44%	5%	5,37%	4,07%	5,37%

Em geral, os piores gaps parecem se concentrar nas categorias 1, 2 e 3, principalmente em instâncias onde o número de tarefas é grande e existem poucas máquinas para processá-las. Isso pode ser observado na Tabela 5. Não se pode concluir, entretanto, que o método não funcionou para esses casos, pois essas diferenças podem estar ocorrendo devido à existência de um gap de dualidade.

Como mencionado no Capítulo 2, o critério de parada utilizado foi 300 iterações ou 1200 segundos de CPU, o que ocorresse primeiro. Os tempos computacionais entre as três heurísticas foram muito parecidos. Uma média dos tempos computacionais (em segundos) é apresentada na Tabela 23. A primeira coluna corresponde ao número de máquinas e a primeira linha ao número de tarefas. Examinando-se a tabela, fica bem evidente que, para um número fixo de tarefas, o tempo computacional é reduzido à medida que a quantidade de máquinas aumenta. Esse comportamento contrasta com a complexidade computacional, que sugere que o tempo deveria crescer (Veja Apêndice A3). Isso ocorre porque a complexidade leva em consideração o pior caso, em que existem n tarefas por máquina.

Tabela 23 - Média dos tempos computacionais das três heurísticas.

número de máquinas	número de tarefas			
	20	50	100	150
2	4,50	60.53	515,72	1203,1
3	4,42	53.43	470,23	1202,7
5	3,24	33.20	216,25	888,84
7	2,78	22.29	148,09	428,44
10	2,66	16.30	97,90	229,48

Observando-se os resultados da busca tabu simples, foi possível notar que o número de inserções é muito menor que o número de trocas. Uma provável explicação é que os movimentos de inserção alteram o número de tarefas em duas máquinas. Isto em geral, resulta em valores de movimento mais altos. Nas duas diversificações propostas pôde-se observar um aumento do número de inserções. Na diversificação com penalização, os movimentos de inserção subiram em $12,41\%$ e na diversificação com inserções, $151,45\%$ em relação ao tabu simples. O aumento das inserções na diversificação com penalização ocorre devido aos parâmetros de penalização α, distintos para movimentos de troca e inserção. Dessa forma, com um valor de α menor para os movimentos de inserções, o número de inserções aumentou. No caso da diversificação com inserções, esse aumento está diretamente relacionado ao tipo de estratégia adotada.

Uma interessante observação pode ser feita analisando-se os casos em que a diversificação com penalidade foi pior que a busca tabu simples. Em muito desses casos, as melhores soluções encontradas pela busca tabu não foram obtidas nas iterações iniciais, e mostram uma tendência de ocorrer nas últimas iterações. Devido a esse fato, foram realizados alguns experimentos computacionais aumentando o número de iterações do método. Observou-se que a partir de um determinado número de iterações, existe uma tendência da diversificação com penalidade apresentar melhores resultados que a busca tabu simples. Isto nos leva a crer que talvez a diversificação com penalidade precise de mais tempo para apresentar bons resultados. É bom lembrar que a penalização é baseada na matriz de freqüência que precisa de um tempo suficientemente grande para ser preenchida e ser uma fonte de informações relevante do passado da busca. Seria interessante, portanto, realizar experimentos considerando um número de iterações maior.

A diversificação com movimentos de inserção é uma das estratégias de diversificação mais simples. Ela não utiliza memória de longo prazo e não depende de parâmetros. Pode-se ver que em casos como na Tabela 15 que ela atingiu resultados muito bons. Isso indica que as inserções podem funcionar realmente como elementos diversificadores na busca tabu.

A seguir, apresenta-se um exemplo do comportamento do atraso das três heurísticas, ao longo das iterações. A Figura 1 apresenta o comportamento da busca tabu simples, diversificação com penalidade e diversificação com inserções, para uma instância de 7 máquinas e 50 tarefas. A busca tabu simples foi o método que apresentou o melhor resultado dentre as três heurísticas. A Figura 2 ilustra um exemplo para 3 máquinas e 20 tarefas, onde a diversificação com penalidade obteve o melhor resultado. Na Figura 3

pode-se observar o desempenho das três heurísticas em uma instância com 3 máquinas e 50 tarefas onde a diversificação com inserções encontrou a melhor solução.

Observando o comportamento das heurísticas nas Figuras 1 e 2 nota-se a existência de platôs bem evidentes. Este comportamento não é resultado de ciclagem, e sim de uma gradual mudança na configuração da solução. Tal comportamento é bastante comum a diversas instâncias. Isto significa que existem muitas soluções com diferentes configurações que resultam no mesmo valor de atraso.

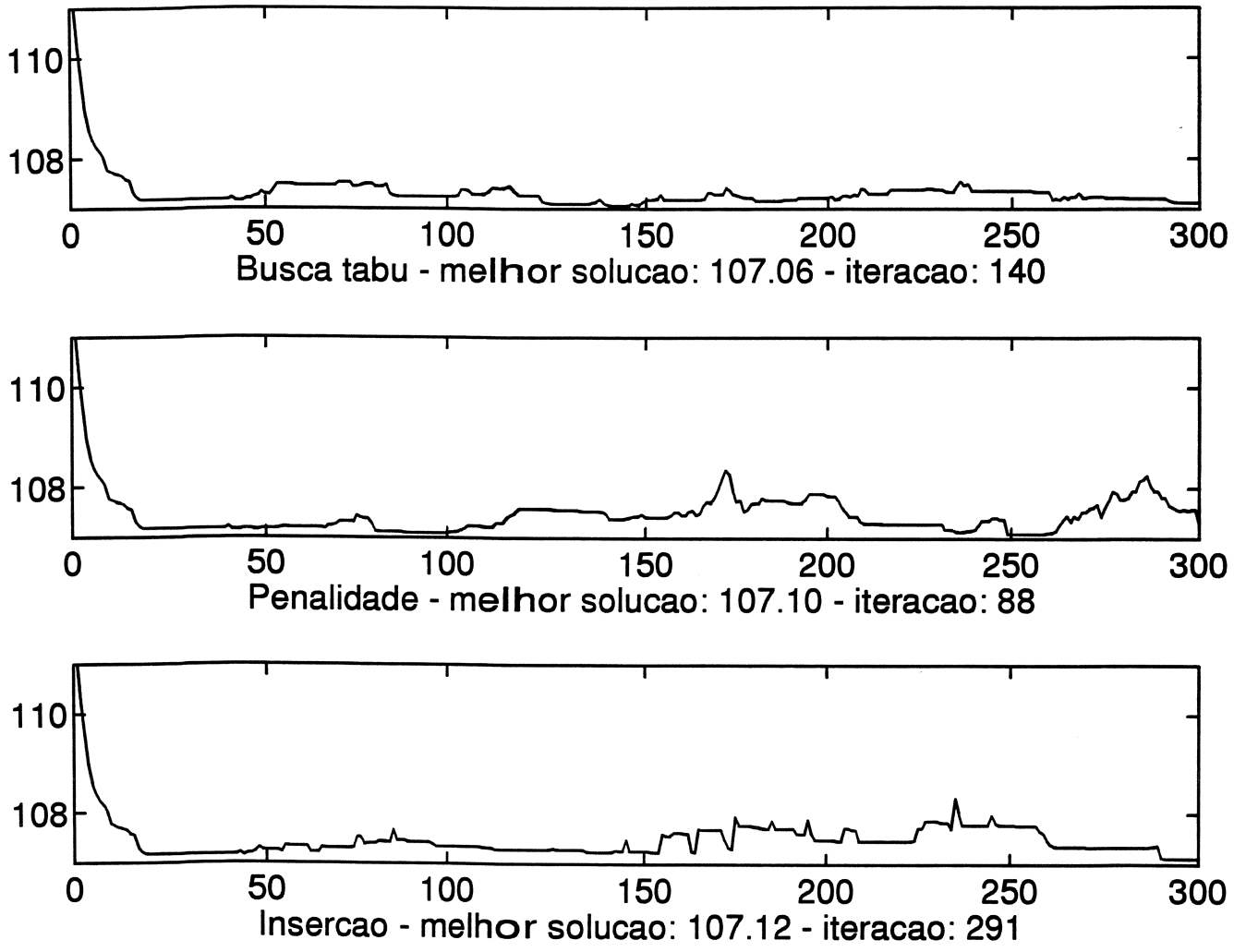


Figura 1: Desempenho das heurísticas - 7 máquinas e 50 tarefas

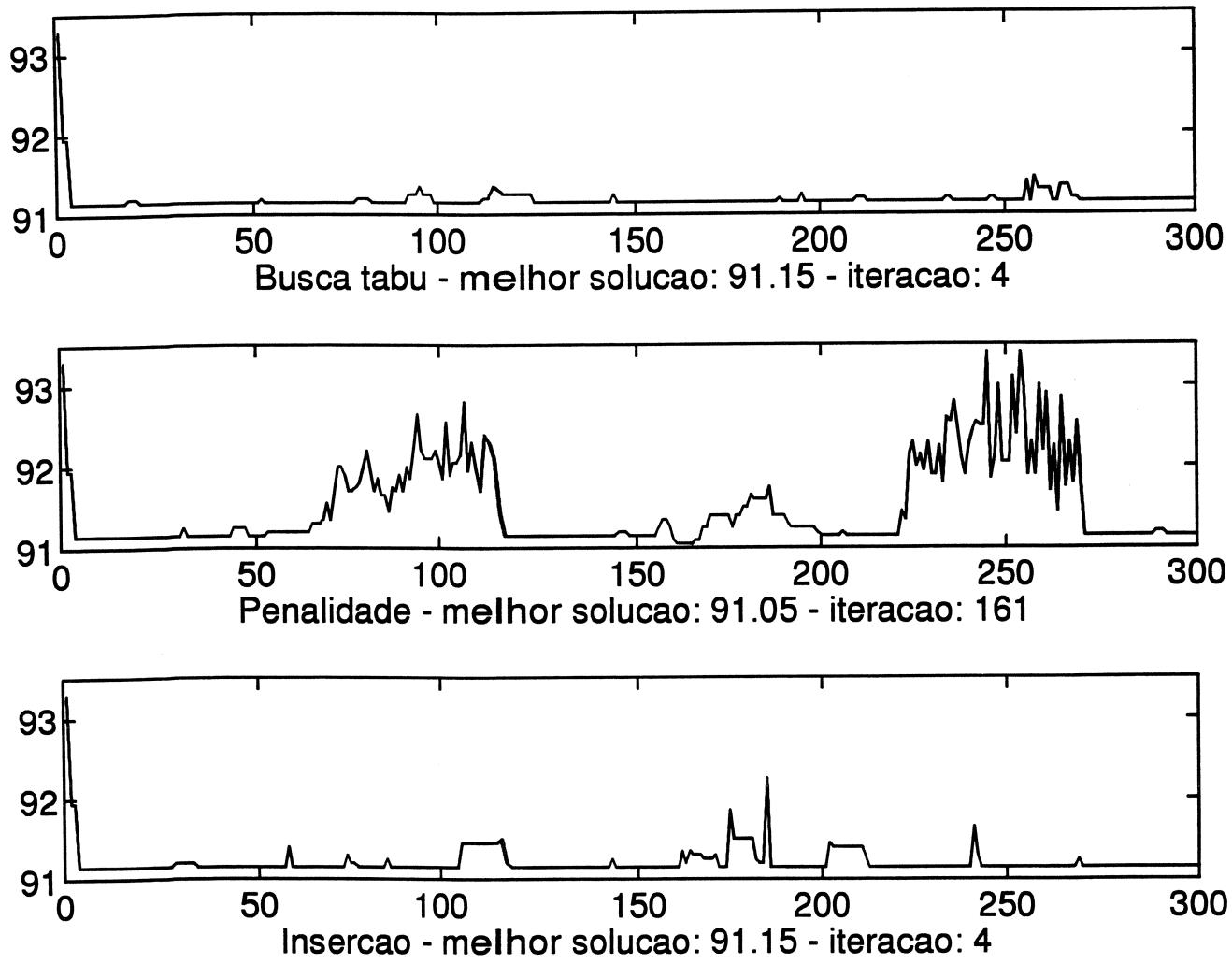


Figura 2: Desempenho das heurísticas - 3 máquinas e 20 tarefas

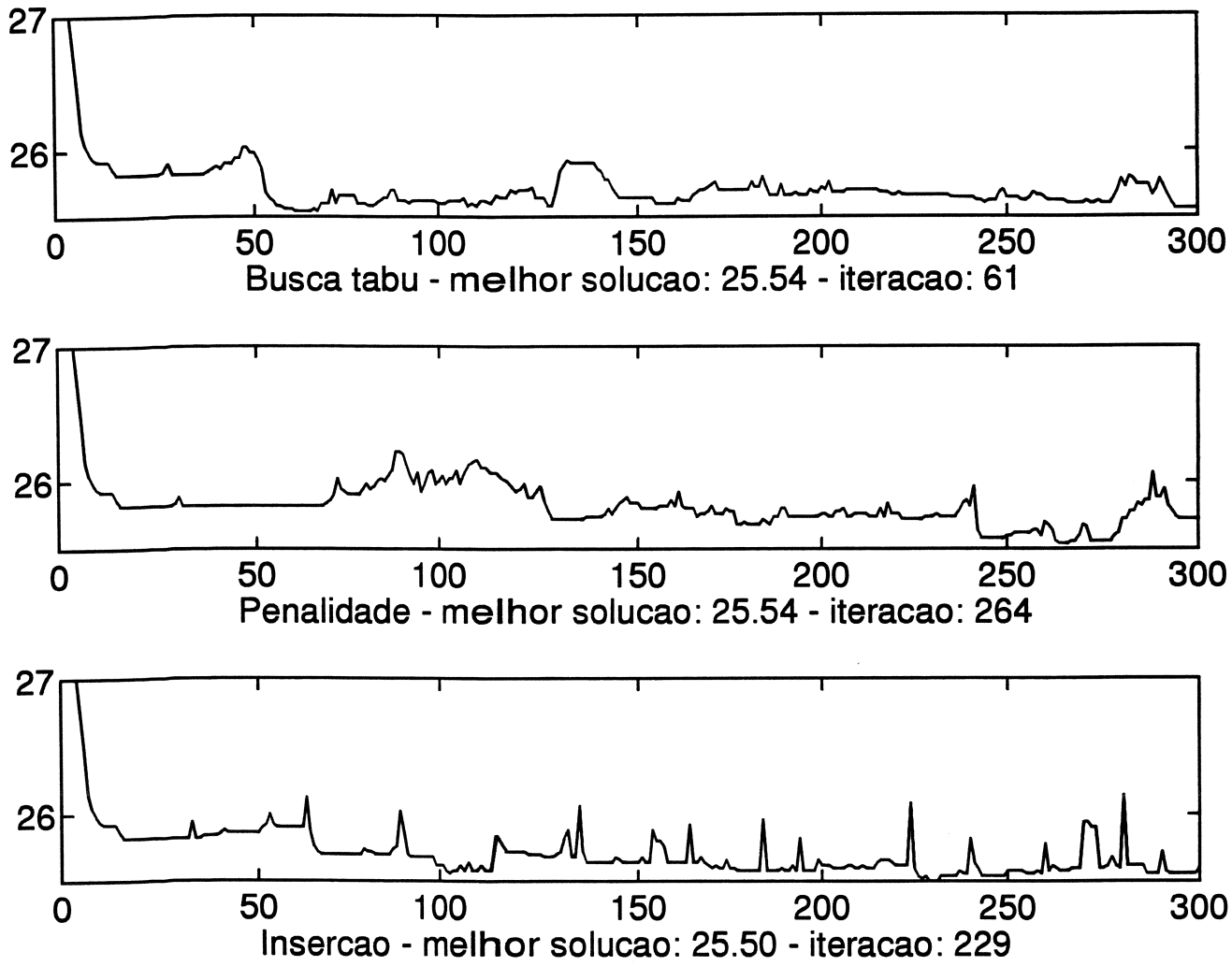


Figura 3: Desempenho das heurísticas - 3 máquinas e 50 tarefas

3.3 Conclusões

O objetivo deste trabalho foi aplicar a busca tabu ao problema de atraso em máquinas paralelas idênticas. Além da busca tabu simples foram propostas duas estratégias de diversificação. Os resultados das heurísticas foram comparados com um limitante inferior utilizando relaxação lagrangeana.

As três heurísticas tiveram um desempenho bastante semelhante. Uma possível explicação é a proximidade das soluções em relação ao limitante inferior. Para 27% das instâncias o mínimo global foi atingido. Mais de 60% dos resultados ficaram a menos de 1% dos limitantes inferiores. Portanto, é possível que em boa parte das instâncias já se tenha alcançado a otimalidade.

Uma parte dos parâmetros usados nas heurísticas foram definidos em função da dimensão do problema, como por exemplo, a duração tabu e o parâmetro de penalização α. Isso permite a aplicação do método conhecendo-se apenas o tamanho do problema.

Através dos experimentos computacionais foi possível observar que os movimentos de inserção, apesar de serem bem menos freqüentes que os movimentos de troca, têm um papel importante no desempenho da heurística. As inserções podem ser bons elementos de diversificação ao longo da busca.

Nota-se que a busca tabu já apresenta bons resultados com poucas iterações. Com cerca de 40 iterações, a heurística já está razoavelmente próxima dos resultados finais apresentados. Para obter resultados melhores, entretanto, é necessário dar uma tempo maior para a heurística, principalmente para a estratégia de diversificação com penalidade. Em pesquisas futuras, pretende-se testar o desempenho da heurística de penalização por um tempo mais longo. Além disso, será testado um valor negativo para o parâmetro de penalização relativo às inserções, com o objetivo de incentivar ainda mais os movimentos de inserção.

Outra idéia para pesquisas futuras é utilizar estratégias de lista de candidatos. Inicialmente, constrói-se uma lista com os k melhores movimentos de toda a vizinhança, e nas iterações subseqüentes, o melhor movimento da lista é executado. O processo continua até que os movimentos não forneçam mais soluções de boa qualidade ou até que um determinado número de iterações seja alcançado. Então uma nova lista é construida e processo se repete. A motivação para a escolha desse tipo de estratégia é a grande quantidade de movimentos com mesmo valor ou valor muito próximo.

Uma alternativa de estratégia de lista de candidatos consiste em avaliar a vizinhança até encontrar um determinado número de movimentos com valores dentro de um limiar estabelecido. Por exemplo, a vizinhança seria pesquisada até que se encontrasse 10 movimentos que resultassem em atrasos 5% melhores que a solução da iteração anterior. Para que a busca não se restrinja a uma região pequena da vizinhança, ou ao contrário, para que ela não explore uma região muito grande, define-se um limite mínimo e máximo para a quantidade de movimentos explorados.

APÊNDICE A

A1. Tamanho da vizinhança

Trocas

Seja m o número de máquinas, n o número de tarefas e n_i o número de tarefas processadas pela máquina i. O cálculo do **tamanho da vizinhança** gerado pelo movimento de **trocas** é apresentado em etapas. Na Etapa 1, é calculado o tamanho da **vizinhança** ao se realizar **uma troca**, por exemplo, ao se trocar a tarefa $J_1(1)$ com a tarefa $J_2(1)$, Figura $1(a)$. Na Etapa 2 apresenta-se o número de **trocas distintas** de tarefas, que é possível efetuar entre **duas** máquinas quaisquer. A seguir, calcula-se o número total de trocas.

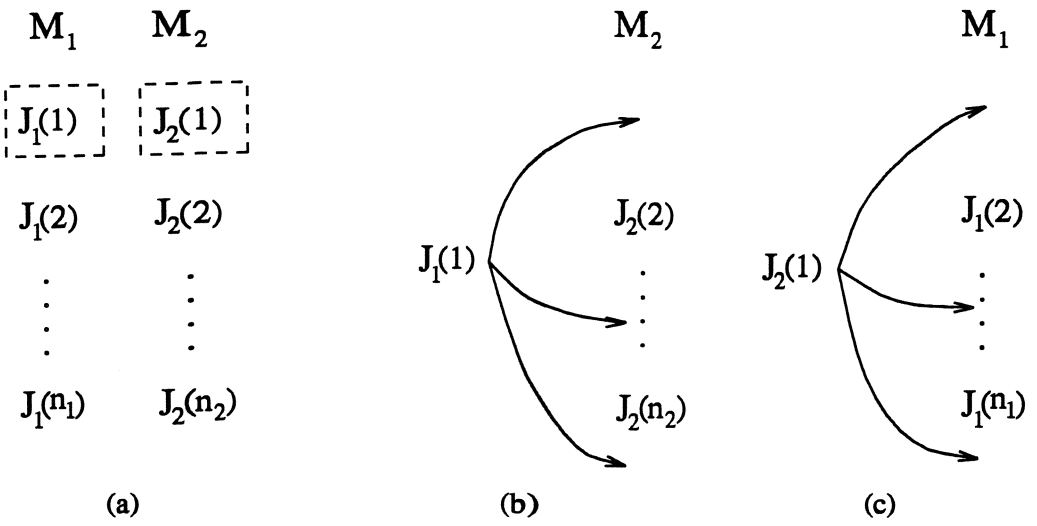


Figura 1: Troca da tarefa $J_1(1)$ com a tarefa $J_2(1)$

Etapa 1: Suponha que deseja-se calcular o tamanho da vizinhança gerada pela troca da tarefa $J_1(1)$ com a tarefa $J_2(1)$. $J_1(1)$ é retirada da máquina M_1 e $J_2(1)$ é retirada da máquina M_2. $J_1(1)$ é inserida nas n_2 posições de M_2 (Figura 1(b)). Para cada posição de $J_1(1)$ em M_2 existem n_1 posições de $J_2(1)$ em M_1. Isso pode ser visto na Figura $1(c)$. Portanto, $n_1 n_2$ soluções podem ser atingidas com a troca de duas tarefas entre duas máquinas.

Etapa 2: Uma vez que existem n_1 tarefas em M_1 que podem ser trocadas com as n_2 tarefas em M_2, então existem $n_1 n_2$ possibilidades diferentes de troca de tarefas entre duas máquinas.

Como foi calculado na Etapa 1, cada troca de tarefas resulta em $n_1 n_2$ soluções. Na Etapa 2 foi deduzido que existem $n_1 n_2$ possíveis trocas que podem ser realizadas entre duas máquinas. Portanto, para duas máquinas, o tamanho da vizinhança é $(n_1 n_2)^2$.

Para m máquinas, $M_1, M_2, ..., M_m$, a vizinhança é composta por todas as trocas que podem ser efetuadas entre duas máquinas i e j. Isto é, a vizinhança é composta pelas possíveis trocas de tarefas entre:

$(M_1, M_2), (M_1, M_3), ..., (M_1, M_m), (M_2, M_3), (M_2M_4), ..., (M_2M_m), ..., (M_{m-1}, M_m)$
Portanto, como o tamanho da vizinhança para trocas entre duas máquinas (M_i e M_j) é $(n_in_j)^2$, o tamanho da vizinhança de trocas é:

$$(n_1n_2)^2 + (n_1n_3)^2 + ... + (n_1n_m)^2 + (n_2n_3)^2 + ... + (n_{m-1}n_m)^2 = \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} (n_in_j)^2 \quad (1)$$

Um limitante superior para (1) pode ser dado por,

$$\sum_{i=1}^{m-1} \sum_{j=i+1}^{m} (n_in_j)^2 \leq \left(\frac{n}{2}\right)^4 \frac{m(m-1)}{2}, \quad (2)$$

como mostrado a seguir.

Sejam n_i, n_j o número de tarefas na máquina i e j, respectivamente, $n_i, n_j \geq 1$.

$$(n_i - n_j)^2 \geq 0$$

$$n_i^2 + n_j^2 - 2n_in_j \geq 0$$

Somando $4n_in_j$ nos dois lados da desigualdade,

$$n_i^2 + n_j^2 - 2n_in_j + 4n_in_j \geq 4n_in_j$$

$$n_i^2 + n_j^2 + 2n_in_j \geq 4n_in_j$$

Agrupando os termos do lado esquerdo da inequação, obtem-se,

$$(n_i + n_j)^2 \geq 4n_in_j$$

Dividindo tudo por 4 e elevando os dois lados da inequação ao quadrado, chega-se a,

$$(n_in_j)^2 \leq \left(\frac{n_i + n_j}{2}\right)^4$$

Como $n = n_1 + n_2 + ... + n_m$,

$$(n_in_j)^2 \leq \left(\frac{n_i + n_j}{2}\right)^4 \leq \left(\frac{n}{2}\right)^4$$

Portanto, $\left(\frac{n}{2}\right)^4$ é um limitante superior para $(n_in_j)^2$. Logo, para a vizinhança completa de trocas tem-se:

$$\sum_{i=1}^{m-1} \sum_{j=i+1}^{m} (n_in_j)^2 \leq \left(\frac{n}{2}\right)^4 \frac{m(m-1)}{2}$$

Mas n_i, n_j são inteiros e portanto isso implica em,

$$\sum_{i=1}^{m-1} \sum_{j=i+1}^{m} (n_i n_j)^2 \leq \left\lfloor \left(\frac{n}{2}\right)^4 \right\rfloor \frac{m(m-1)}{2}$$

Isto é, $\left\lfloor \left(\frac{n}{2}\right)^4 \right\rfloor \frac{m(m-1)}{2}$ é um limitante superior para a vizinhança de trocas.

No caso de 2 máquinas e um número par de tarefas, o tamanho da vizinhança de trocas atinge o limitante superior se $n_1 = n_2 = n/2$.

No entanto, para três máquinas, o seguinte exemplo mostra que o limitante (2) é fraco. Seja $n_1 = 1$, $n_2 = 2$, $n_3 = 3$. O tamanho da vizinhança é obtido substituindo esses dados em (1), isto é, $(n_1 n_2)^2 + (n_1 n_3)^2 + (n_2 n_3)^2 = 49$. Por enumeração completa é possível verificar que este é o caso que resulta na maior vizinhança. No entanto, o valor do limitante superior é $(6/2)^4 * 3 * 2/2 = 243$, que é muito maior que o pior caso.

Uma estimativa mais realista para a vizinhança de trocas pode ser obtida considerando-se que existem n/m tarefas por máquinas, com n múltiplo de m. Neste caso, basta substituir n/m em (1) para obter

$$\left(\frac{n}{m}\right)^4 \frac{m(m-1)}{2}$$

Utilizando esta estimativa mais realista para o problema anterior, o valor da vizinhança é : $2^4 * 3 * 2/2 = 48$. Pode-se observar que esta estimativa está muito mais próxima do tamanho exato da vizinhança do que o limitante superior.

Inserção

Seja m o número de máquinas, n o número de tarefas, e n_i a quantidade de tarefas na máquina i.

Vamos calcular o tamanho da vizinhança ao se inserir a tarefa $J_1(1)$ na máquina M_2 (Figura 2(b). A tarefa $J_1(1)$ é retirada de sua máquina e pode ser inserida em $(n_2 + 1)$ posições de M_2, gerando uma vizinhança de tamanho $(n_2 + 1)$. Como existem n_1 tarefas de M_1 que podem ser inseridas em M_2, o número de soluções geradas é $n_1(n_2 + 1)$. Da mesma forma, existem n_2 tarefas de M_2 que podem ser inseridas em M_1 gerando $n_2(n_1+1)$ soluções. Logo, para $m = 2$, o tamanho da vizinhança gerado pelo movimento de inserção é $n_1(n_2 + 1) + n_2(n_1 + 1)$. Portanto, para m máquinas tem-se:

$$\sum_{i=1}^{m-1} \sum_{j=i+1}^{m} (2n_i n_j + n_i + n_j) \tag{3}$$

Um limitante superior para o tamanho da vizinhança de inserções pode ser dado por,

$$n^2 \frac{(m-1)m}{2},$$

como mostrado a seguir.

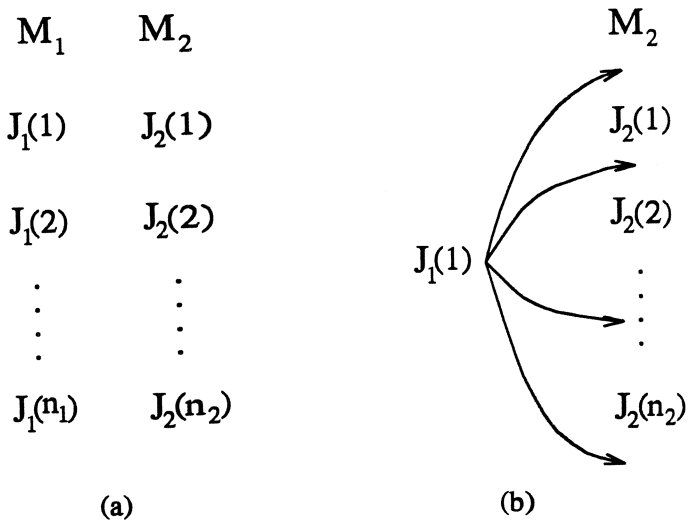


Figura 2: Inserção da tarefa $J_1(1)$ na máquina 2

Sejam duas máquinas M_i e M_j. O tamanho da vizinhança de inserção entre essas duas máquinas é dado por

$$n_i(n_j+1)+n_j(n_i+1) = 2n_in_j+n_i+n_j \leq (n_i+n_j)^2 \leq n^2$$

onde $n = n_1 + n_2 + ... + n_m$.

Isso significa que cada termo de

$$\sum_{i=1}^{m-1}\sum_{j=i+1}^{m}(2n_in_j+n_i+n_j)$$

pode ser limitado superiormente por n^2, o que implica em

$$\sum_{i=1}^{m-1}\sum_{j=i+1}^{m}(2n_in_j+n_i+n_j) \leq n^2\frac{(m-1)m}{2}$$

Para obter uma estimativa mais realista do número de tarefas por máquinas, considera-se que existem n/m tarefas em cada máquina, com n múltiplo de m. Subsistituindo n/m em (3) temos,

$$(\frac{n}{m}+1)(n)(m-1)$$

A2. Matriz de Armazenamento

Como já foi dito antes, no decorrer da busca tabu, a cada iteração seleciona-se o melhor movimento. Quando o movimento é de troca, é possível observar que apenas as duas máquinas envolvidas no processo é que sofrem modificações. As demais máquinas não têm qualquer tipo de alteração. Uma maneira de poupar cálculos computacionais é armazenar parte desses cálculos em uma matriz e recalcular apenas os movimentos que envolvem as máquinas modificadas na iteração anterior.

Seja um problema com m máquinas, M_1, M_2, M_3,..., M_m. A cada iteração, para selecionar o melhor movimento de troca avaliam-se todas as trocas de tarefas entre duas máquinas. Assim, serão avaliadas as trocas entre M_1 e M_2, M_1 e M_3, ... A melhor troca é então selecionada. Suponha que a melhor troca envolva as máquinas M_i e M_j. Para o cálculo do melhor movimento na próxima iteração não é necessário então recalcular o valor de movimento para as $m(m-1)/2$ combinações de duas máquinas, mas apenas para os movimentos que envolvem as máquinas que foram modificadas na iteração anterior,

M_i M_1, M_i M_2, M_i M_3, ..., M_i M_j,..., M_i M_m.
M_j M_1, M_j M_2, M_j M_3,...,M_j M_{i-1}, M_j M_{i+1}, ...,M_j M_m.
ou seja, é preciso calcular o valor do movimento para $(m-1)+(m-2) = 2m-3$, que é inferior a $m(m-1)/2$ combinações de duas máquinas.

A estrutura de dados utilizada para poupar os cálculos desses movimentos é uma matriz que armazena dados relativos a todas as possíveis trocas de tarefas entre duas máquinas. Dadas duas máquinas M_i e M_j, e duas tarefas pertencentes a essas máquinas, $J_i(k)$ e $J_j(l)$ é necessário que a matriz guarde qual é a melhor posição na máquina para se inserir essas tarefas após a troca, e o valor do atraso ao se realizar essa troca. Para isso, utiliza-se um vetor $M[i][j][k][l]$, composto por três campos: *posicao*1, *posicao*2 e *atraso*. A posição em que a k-ésima tarefa da máquina i deve ser inserida na máquina j é dada por $M[i][j][k][l].posicao1$, a posição em que a l-ésima tarefa da máquina j deve ser inserida na maquina i é dada por $M[i][j][k][l].posicao2$. O atraso originado por essas duas trocas é dado por $M[i][j][k][l].atraso$.

Exemplo

Sejam dois programas dados por $S_1 = \{J_1(1), J_1(2)\}$ e $S_2 = \{J_2(1), J_2(2), J_2(3)\}$. A tabela abaixo apresenta as possíveis trocas de tarefas entre as duas máquinas, a melhor posição que as tarefas devem ocupar na máquina em que estão sendo inseridas, e o valor do atraso resultante dessa troca.

Trocas	Melhor posição tarefa $J_1(*)$ na máquina 2	Melhor posição da tarefa $J_2(*)$ na máquina 1	Atraso Resultante
$J_1(1)$ $J_2(1)$	2	1	23
$J_1(1)$ $J_2(2)$	1	1	25
$J_1(1)$ $J_2(3)$	2	2	10
$J_1(2)$ $J_2(1)$	3	2	20
$J_1(2)$ $J_2(2)$	3	1	19
$J_1(2)$ $J_2(3)$	2	2	14

A matriz M é preenchida da seguinte forma para os dados da tabela anterior:
$M[1][2][1][1].posicao1 = 2$ $M[1][2][1][1].posicao2 = 1$ $M[1][2][1][1].atraso = 23$
$M[1][2][1][2].posicao1 = 1$ $M[1][2][1][2].posicao2 = 1$ $M[1][2][1][2].atraso = 25$

$\vdots$

$M[1][2][2][3].posicao1 = 2$ $M[1][2][2][3].posicao2 = 2$ $M[1][2][2][3].atraso = 14$

A3. Complexidade Computacional

A complexidade computacional depende sobretudo dos procedimentos que calculam o valor de movimento para as trocas e inserções. Portanto, o cálculo da complexidade será realizado para esses dois procedimentos.

O procedimento *Movimento_insercao* calcula todos os valores de movimentos relativo às inserções.

Movimento_insercao
{
(1)Para as máquinas M_i, $i = 1$ até $(m-1)$, faça:
(2) Para todas as tarefas $J_i(k)$ de M_i, faça:
(3) Para todas as máquinas M_j, $i \neq j$ faça:
Melhor_posicao$(J_i(k), M_j)$
}

O procedimento *Melhor_posicao*$(J_i(k), M_j)$ seleciona a melhor posição que a tarefa $J_i(k)$ deve ocupar em M_j. Como esse procedimento implica em precorrer M_j uma vez, avaliando cada posição que $J_i(k)$ pode ocupar em M_j, a complexidade é $O(n)$.

No procedimento do *Movimento_insercao*, o laço (1) é executado $(m-1)$ vezes. Para cada máquina de (1), o laço 2 é repetido, no pior caso, n vezes. Para cada tarefa de $J_i(k)$ de M_i, o laço (3) é executado $(m-1)$ vezes. Portanto, o procedimento *Melhor_posicao*$(J_i(k), M_j)$ é chamado $((m-1)^2 n)$ vezes. Conclui-se então, que o procedimento *Movimento_insercao* é da ordem $O(n^2 m^2)$.

O procedimento *Movimento_troca* descrito a seguir calcula todos os valores de movimentos de trocas.

Movimento_troca
{
(1)Para todas as máquinas M_i e M_j, $(i \neq j)$, onde M_i ou M_j foram modificadas na iteração anterior, faça:
(2) Para todas as tarefas $J_i(k)$ de M_i faça:
(3) Para todas as tarefas $J_j(l)$ de M_j faça:
(4) *Melhor_posicao*$(J_i(k), J_j(l), M_i, M_j)$
(5)Para todas as máquinas M_i e M_j, $(i \neq j)$, onde M_i e M_j não foram modificadas na iteração anterior, faça:
(6) Para todas as tarefas $J_i(k)$ de M_i faça:
(7) Para todas as tarefas $J_j(l)$ de M_j faça:
Lê valor do movimento na Matriz de Armazenamento.
}

O procedimento *Melhor_posicao*$(J_i(k), J_j(l), M_i, M_j)$ encontra qual é a melhor posição que a tarefa $J_i(k)$ deve ocupar em M_j e qual é a melhor posição que a tarefa $J_j(l)$ deve

ocupar em M_i para que se tenha o menor valor de movimento. Como esse procedimento implica em percorrer as duas máquinas uma vez, avaliando a melhor posição em que a tarefa deve entrar, então a ordem do procedimento é $O(n)$ no pior caso.

O laço em (1) é executado $(2m - 3)$ vezes (ver apêndice A2). Considerando que no pior caso existem n tarefas em cada máquina, os laços em (2) e (3) variam de 1 até n, portanto, o procedimento $Melhor\_posicao(J_i(k), J_j(l), M_i, M_j)$ será executado $n^2(2m-3)$ vezes. Logo, as operações dos laços (1), (2), (3) e procedimento *Melhor_posicao* são da ordem $O(n^3m)$.

O laço (5) determina o valor do movimento para as máquinas restantes, não consideradas no laço (1). Portanto, o laço (5) será executado $m(m-1)/2-(2m-3)$ vezes. Para cada máquina M_i e M_j de (5), avaliam-se os laços (6) e (7), que variam de 1 até n no pior caso. Portanto, a matriz de armazenamento será acessada $n^2(m^2/2 - 5m/2 + 3)$ vezes. Logo, o procedimento $Movimento\_troca$ tem ordem $O(n^3m) + O(n^2m^2)$.

Conclui-se então que a complexidade computacional da heurística é $O(n^3m)+O(n^2m^2)$.

A4. Relaxação Lagrangeana

O limitante inferior utilizado para comparação com as heurísticas é baseado em uma relaxação lagrangeana proposta por Luh et al.(1990).

O problema de minimizar o atraso em máquinas paralelas idênticas pode ser formulado como um problema de programação inteira. As variáveis usadas são:

δ_{it} - variável binária que é igual a 1 se a tarefa i é processada no instante t, e 0 caso contrário.
B_i - tempo de início do processamento da tarefa i
C_i - tempo de término da tarefa i
d_i - data de entrega da tarefa i
K - horizonte de tempo considerado
m - quantidade de máquinas disponíveis.
n - número de tarefas
p_i - tempo de processamento da tarefa i
T_i - atraso da tarefa i, $T_i = max(0, C_i - d_i) = max(0, B_i + p_i - d_i)$

As variáveis de decisão são B_i, $i = 1, ..., n$, instante em que a tarefa i começa a ser processada, pois estas determinam C_i, T_i e δ_{it}, $i = 1, 2, ..., n$ e $t = 0, 1, ..., K$. Por exemplo, se $B_i = 2$, $p_i = 3$, e $d_i = 3$, temos $\delta_{i2} = \delta_{i3} = \delta_{i4} = 1$, $C_i = B_i + p_i = 5$, e $T_i = C_i - di = 2$. Assume-se que o horizonte de tempo considerado, K, é grande o suficiente para completar todas as tarefas (isto é, $C_i \leq K$ para todo i).

O problema estudado pode ser formulado do seguinte modo:

$$(P) \quad \min_{\{B_i\}} \sum_{i=1}^{n} T_i \tag{1}$$

sujeito às restrições de capacidade,

$$\sum_{i=1}^{n} \delta_{it} \leq m, \quad t = 0, 1, ..., K \tag{2}$$

Para decompor o problema (P) de acordo com as tarefas, as restrições de capacidade (2) são relaxadas usando o multiplicador de lagrange π_t, obtendo-se assim o seguinte problema lagrangeano,

$$h(\pi_t) = \min_{\{B_i\}} L = \left\{ -\sum_{t=0}^{K} \pi_t m + \sum_{i=1}^{n} \left\{ T_i + \sum_{t=0}^{K} \pi_t \delta_{it} \right\} \right\} \tag{3}$$

O problema lagrangeano (3) se decompõe em um subproblema para cada tarefa i:

$$\min_{0 \leq B_i \leq K - p_i} L_i, \quad \text{com } L_i = \left\{ T_i + \sum_{t=0}^{K} \pi_t \delta_{it} \right\} \tag{4}$$

Então o problema dual é:

$$D : \max_{\pi} h(\pi), \tag{5}$$

$$\text{sujeito a } \pi \geq 0 \tag{6}$$

Com a finalidade de obter o limitante inferior para o atraso, é necessário resolver os subproblemas e o problema dual. Resolver os subproblemas é equivalente a encontrar os valores de B_i que minimizam (4). Para isso, L_i é calculado para cada possível valor de B_i, até que seja encontrado B_i^* que minimiza L_i. Pode-se observar que a complexidade computacional do subproblema lagrangeano é linear em K, uma vez que é preciso realizar no máximo K avaliações para se obter B^*.

Para resolver o problema dual, utiliza-se o método do subgradiente modificado proposto por **Camerini, Fratta e Maffioli (1975)**.

O método do subgradiente termina em qualquer uma das seguintes situações: número máximo de 300 iterações; máximo tempo de CPU igual a 20 min; vetor multiplicador de lagrange praticamente inalterado por um período de 20 iterações. Esse critério foi adotado para a grande maioria dos problemas. Entretanto, para problemas de 100 e 150 tarefas, onde era possível obter um acréscimo significativo na função dual, foi dado um tempo maior de CPU de aproximadamente 3 horas por problema.

REFERÊNCIAS BIBLIOGRÁFICAS

Arkin, E. M., Roundy, R. O., (1991), *Weighted-Tardiness Scheduling on Parallel Machines with Proportional Weights,* Operations Research, Vol 39, No.1, 64-81.

Baker, K. R., (1973), *Procedures for sequencing tasks with one resource type,* Int. J. Prod. Res., Vol 11, No.2, 125-138.

Baker, K. R., Bertrand, J. W. M., (1982), *A Dynamic Priority Rule for Scheduling Against Due-Dates,* Journal of Operations Management, Vol 3, No.1, 37-42.

Barnes, J. W., Laguna, M., (1993), *Solving the Multiple-Machine Weighted Flow Time Problem Using Tabu Search,* IIE Transactions, Vol 25, No.2, 121-128.

Camerini, P.M., Fratta, L., Maffioli F., (1975), *On Improving Relaxation Methods by Modified Gradient Techniques,* Mathematical Programming Study, Vol 3, 26-34.

Cheng, T. C. E., Sin, C. C. S., (1990), *A State-of-the-art Review of Parallel-Machine Scheduling Research,* European Journal of Operational Research, Vol 47, 271-292.

Dogramaci, A., Surkis, J., (1979), *Evaluation of a Heuristic for Scheduling Independent Jobs on Parallel Identical Processors,* Management Science, Vol 25, No.12, 1208-1216.

Du, J., Leung, Y.-T., (1990), *Minimizing Total Tardiness on One Machine is NP-Hard,* Mathematics of Operations Research, Vol 15, No.3, 483-495.

Emmons, H., (1969), *One Machine Sequencing to Minimize Certain Functions of Job Tardiness,* Operations Research, Vol 17, No.4, 701-715.

França, P. M., Gendreau, M., Laporte, G., Müller, F. M., (1994), *A Composite Heuristic for the Identical Parallel Machine Scheduling Problem with Minimum Makespan Objective,* Computers Operations Research, Vol 21, No.2, 205-210.

França, P. M., Gendreau, M., Laporte, G., Müller, F. M., (1995), *The m-Traveling Salesman Problem with Minmax Objective,* Transportation Science, Vol 29, No.3, 267-275.

Fry, T. D., Vicens, L., Macleod, K., Fernandez, S., (1989) *A Heuristic Solution Procedure to Minimize $\bar{T}$ on a Single Machine,* J. Opl. Res. Soc. Vol 40, No.3, 293-297.

Glover, F., (1995), *Tabu Search Fundamental and Uses,* Graduate School of Business, University of Colorado.

Gupta, J. N. D., Maykut, A. R., (1973), *Scheduling Jobs on Parallel Processors with Dynamic Programming,* Decision Sciences, Vol 4, 447-457.

Ho, C. J., Chang, Y.-L., (1991), *Heuristics for Minimizing Mean Tardiness for m Parallel Machines,* Naval Research Logistics, Vol 38, 367-381.

Hübscher, R., Glover, F., (1994), *Applying Tabu Search with Influential Diversification to Multiprocessor Scheduling,* Computers Ops. Res., Vol 21, No.8, 877-884.

Koulamas, C. P., (1994), *The Total Tardiness Problem: Review And Extensions,* Operations Research, Vol 42, No.6, 1025-1041.

Laguna, M., Velaverde, J. L. G., (1991), *A Search Heuristic for Just-in-Time Scheduling in Parallel Machines,* Journal of Intelligent Manufacturing, Vol 2, 253-260.

Laguna, M., (1994), *A Guide to Implementing Tabu Search,* Investigación Operativa, Vol 4, No.1, 5-24.

Laguna, M., (1995), *Tabu Search Tutorial,* II Escuela de Verano Latino-Americana de Investigación Operativa.

Li, C.-L., Cheng, T.C.E., (1994), *The Parallel Machine Min-Max Weighted Absolute Lateness Scheduling Problem,* Naval Research Logistics, Vol 41, 33-46.

Luh, P. B., Hoitomt, D. J., Max, E., Pattipati, K. R., (1990), *Schedule Generation and Reconfiguration for Parallel Machines,* IEEE Transactions on Robotics and Automation, Vol 6, No.6, 687-696.

Panwalkar, S. S., Smith, M. L., Koulamas, C. P., (1993), *A Heuristic for the Single Machine Tardiness Problem,* European Journal of Operational Research, Vol 70, 304-310.

Potts, C. N., Van Wassenhove, L. N., (1982), *A Decomposition Algorithm for the Single Machine Total Tardiness Problem,* Operations Research Letters, Vol 1, No.5, 177-181.

Sahni, S., Cho, Y., (1980), *Scheduling Independent Tasks with Due Times on a Uniform Processor System,* Journal of the Association for Computing Machinery, Vol 27, 550-563.

Wilkerson, L.J., Irwin, J.D., (1970), *An Improved Algorithm for Scheduling Independent Tasks,* Technical Report AU-T-15, Digital Systems Laboratory, Auburn University,

Auburn, Alabama. *apud* Baker, K. R., (1973), *Procedures for sequencing tasks with one resource type,* Int. J. Prod. Res., Vol 11, No.2, 125-138.

Wilkerson, L.J., Irwin, J.D., (1971), *An Improved Method for Scheduling Independent Tasks,* AIIE Transactions, Vol 3, No.3, 239-245.

Yamashita, D. S., Armentano, V. A., (1995), *Uma Heurística para o Problema de Atraso Médio em Máquinas Paralelas,* Anais do XXVII Simpósio Brasileiro de Pesquisa Operacional, Vitória, Brasil.