UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

DEPARTAMENTO DE ENGENHARIA DE SISTEMAS

BUSCA TABU PARA A PROGRAMAÇÃO DE TAREFAS EM JOB SHOP COM DATAS DE ENTREGA

CINTIA RIGÃO SCRICH

Orientador:

Prof. Dr. Vinicius Amaral Armentano

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

- AGOSTO 1997 -

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

Scrich, Cintia Rigão
~~Sc31b~~ Busca Tabu para a programação de tarefas em job shop com datas de entrega / Cintia Rigão Scrich.--Campinas, SP: [s.n.], 1997.

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

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

AGRADECIMENTOS

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

- ao Vinicius, pela orientação irrepreensível e pela amizade;

- aos professores, pela minha formação;

- à minha família, pelo apoio e incentivo;

- ao Zake pelo amor, interesse e paciência;

- à Regina, Débora, Denise e Sacchi pelas discussões sobre este trabalho;

- à Fran por deixar meus programas mais "rápidos" e ao Sacchi pelas dicas computacionais;

- à Renata pelas constantes ajudas na escrita;

- à Lúcia, Vitória, Hamilton, Mauro, Pablo, Regina, Débora, Ernesto, Fran, Renata, Gelson, Cássio, Walcir, Jussara, Gonzaga, pelos momentos agradáveis, pelos cafés, almoços e pelas saídas animadas;

- à Márcia pela paciência e prontidão;

- ao Walcir, Suze e Luciana pelo apoio computacional;

- aos funcionários da BAE e do serviço de comutação bibliográfica COMUT, pela ajuda e eficiência;

- ao CNPq pelo apoio financeiro.

RESUMO

Este trabalho trata do problema de programação de tarefas nos ambientes *job shop* tradicional e *job shop* flexível com o objetivo de minimizar o atraso total das tarefas. A principal diferença do *job shop* flexível em relação ao *job shop* tradicional é que cada operação possui um conjunto de máquinas alternativas onde pode ser processada. Para cada um dos problemas é desenvolvida uma heurística guiada pela metaheurística Busca Tabu. Estratégias de diversificação e intensificação para a busca são sugeridas. Testes e resultados computacionais obtidos são apresentados. Para o *job shop* tradicional dois métodos heurísticos encontrados na literatura foram implementados e comparados com a heurística proposta.

ABSTRACT

This work addresses the traditional and the flexible job shop scheduling problems with the objective of minimizing total tardiness. The main difference between the flexible job shop and the traditional job shop is that each operation of a job can be processed in alternative machines. A heuristic method guided by the tabu search metaheuristic is developed for each problem. Diversification and intensification strategies are suggested. Tests and computational results are reported. For the traditional job shop two heuristic methods from the literature are implemented and compared with the proposed heuristic.

SUMÁRIO

INTRODUÇÃO

À medida que a concorrência mundial se intensifica, aumentam os esforços para fornecer produtos e serviços com o maior valor possível pelo menor custo e no menor período de tempo. Competir globalmente não se trata apenas de fazer negócios internacionalmente ou de ter subsidiárias no exterior. O desenvolvimento de uma cultura organizacional global envolve a formação de valores, mecanismos e processos de integração entre os vários setores da empresa que lhe permitam reagir às constantes mudanças que ocorrem neste mercado global. E isto significa tornar global tanto sua cultura organizacional bem como suas operações de pesquisa e fabricação e atividades de marketing e distribuição.

Assim, as principais características a serem resolvidas pelas empresas que querem sobreviver no mercado hoje são conseguir fornecer variedade de produtos, o que demanda flexibilidade de produção, a baixo custo e entregá-lo sem atraso e com qualidade ao consumidor.

Em termos práticos, pode-se enxergar dois grandes problemas a serem equacionados: organizar um sistema de produção, capaz de fabricar uma variedade razoável de produtos, o que conseqüentemente significa que o volume de cada variedade é baixo ou médio, e entregar o produto sem atraso, o que gera uma necessidade de produzí-lo também sem atraso.

Um sistema de produção pode ser dividido em três níveis de planejamento distintos. O nível mais alto é responsável por decidir as políticas adequadas visando os objetivos de longo prazo da empresa, e deve especificar os recursos necessários para atingir tais objetivos, as estratégias para aquisição de equipamentos, localização e tamanho de fábricas, dentre outros. O segundo nível está relacionado à implementação das estratégias definidas no nível superior e seu principal objetivo é a alocação eficiente de recursos para satisfazer a demanda, levando em conta os custos envolvidos. O terceiro nível trata das decisões de curto prazo, associadas à execução dos planos definidos no nível anterior. Neste nível é tratada a programação

detalhada do fluxo de materiais no chão de fábrica. Essa programação envolve a alocação de operações de produção a recursos disponíveis e o roteamento de cada produto.

O problema abordado neste trabalho se ocupa das duas questões associadas à programação de tarefas (*scheduling*) em um *job shop*, que é um ambiente que se caracteriza por baixo volume de produção e alta variedade de produtos, e tem por objetivo minimizar o atraso das tarefas em relação às datas de entrega. Um *job shop* pode ser definido por um conjunto de tarefas e por um conjunto de máquinas, onde cada tarefa consiste de um conjunto de operações que devem ser processadas pelas máquinas. Cada operação deve ser processada por uma única máquina e cada máquina só pode processar uma operação de cada vez. O problema de programação consiste portanto, em determinar o instante de início de processamento de cada operação das tarefas. No trabalho são estudados os ambientes *job shop* tradicional e *job shop* flexível. O *job shop* flexível difere do tradicional, pois cada operação possui um conjunto de máquinas alternativas onde pode ser processada. Por causa da dificuldade intrínseca de programação em um *job shop*, este trabalho apresenta uma resolução heurística para o problema em cada um dos ambientes. As heurísticas desenvolvidas consistem basicamente de dois procedimentos. O primeiro consiste de aplicação de regras de despacho para obtenção de uma solução inicial. O segundo procedimento busca novas soluções através da metaheurística Busca Tabu.

Para apresentar a abordagem ao problema acima descrito e os métodos de solução propostos, este trabalho está dividido em 6 capítulos que são apresentados sucintamente a seguir.

O **Capítulo 1** consiste na apresentação do problema de programação de tarefas nos ambientes *job shop* tradicional e *job shop* flexível e na revisão bibliográfica de ambos. No **Capítulo 2** são apresentados os fundamentos básicos da metaheurística Busca Tabu e também é feita uma revisão bibliográfica das aplicações desta metaheurística para *job shop*. No **Capítulo 3** apresenta-se uma descrição detalhada das heurísticas propostas para os problemas, acompanhada de exemplos que ilustram os procedimentos envolvidos. Nos **Capítulos 4** e **5** são descritos a geração dos problemas de teste, os testes efetuados e os resultados obtidos para o ambiente tradicional e flexível, respectivamente, e por fim, o **Capítulo 6** contém as conclusões, discussões e propostas futuras.

CAPÍTULO 1

O PROBLEMA DE *JOB SHOP* COM DATAS DE ENTREGA

1.1. Introdução

O problema de programação de tarefas em um *job shop* consiste em determinar a seqüência e o instante de início de processamento de cada tarefa, composta por operações ordenadas, em um conjunto de máquinas de modo a otimizar um ou mais critérios. Neste capítulo são apresentados os problemas de *job shop* tradicional e de *job shop* flexível bem como a revisão bibliográfica de ambos.

1.2. O Problema Tradicional

Um *job shop* tradicional pode ser definido por um conjunto de **m** máquinas e um conjunto de **n** tarefas, onde cada tarefa consiste de uma seqüência ordenada de **m** operações. Cada operação deve ser processada em uma única máquina por um determinado tempo sem interrupção e uma máquina pode processar somente uma operação de cada vez. As operações de uma mesma tarefa devem ser executadas em máquinas diferentes e cada tarefa possui uma ordem particular de processamento nas máquinas. O problema consiste em programar as operações das tarefas de forma a otimizar algum critério.

Este problema pode ser representado através de um grafo disjuntivo (Balas, 1969) onde:

- para cada operação de cada tarefa é criado um nó com peso igual ao seu tempo de processamento;

- dois nós artificiais com peso nulo são criados correspondendo às operações inicial e final;

- um arco é criado do nó inicial ao nó correspondente à primeira operação de cada tarefa; para cada operação de cada tarefa é criado um arco do nó correspondente àquela operação ao nó correspondente à próxima operação. Tais arcos representam as restrições de precedência entre operações de uma mesma tarefa;

- os nós correspondentes a operações a serem executadas na mesma máquina são unidos por arcos disjuntivos (direção ainda não estabelecida). Uma vez estabelecida a direção, o arco define a precedência das operações na máquina. A figura a seguir ilustra um grafo disjuntivo para um problema com 3 tarefas e 4 máquinas.

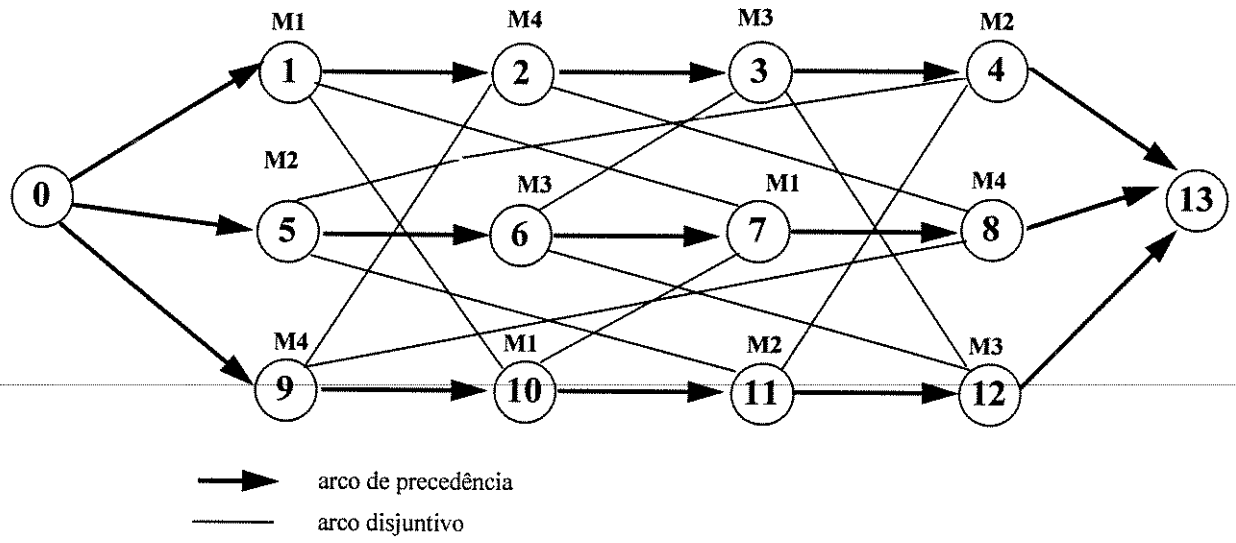


Figura 1.1. - Grafo Disjuntivo

Na Figura 1.1, a tarefa 1 é composta das operações 1, 2, 3 e 4 que são processadas nas máquinas M1, M4, M3 e M2, respectivamente. As tarefas 2 e 3 são compostas das operações 5 a 8 e 9 a 12, respectivamente. Os arcos disjuntivos mostram por exemplo, que as operações 1, 7 e 10 são processadas pela máquina 1.

Quando as direções dos arcos disjuntivos são escolhidas, obtém-se um grafo direcionado. Se o grafo for acíclico, a configuração obtida caracteriza uma solução factível e pode-se então calcular o maior caminho de cada tarefa que é o caminho mais longo do nó 0 até seu último nó. Tal caminho determina o instante de término da tarefa. Note que um grafo direcionado cíclico não estabelece precedência e portanto, o caminho mais longo é infinito. Neste trabalho é adotado que caminho crítico do grafo é o maior caminho do nó inicial ao nó final do grafo e caminho crítico da tarefa é o maior caminho do nó inicial ao nó final da tarefa.

Como exemplo, na Figura 1.2, o caminho crítico do grafo é o maior caminho do nó 0 ao nó 7, e passa pelos nós 0-1-4-5-6-7, e o caminho crítico da tarefa 2 é o maior caminho do nó 0 ao nó 4, e passa pelos nós 0-1-4.

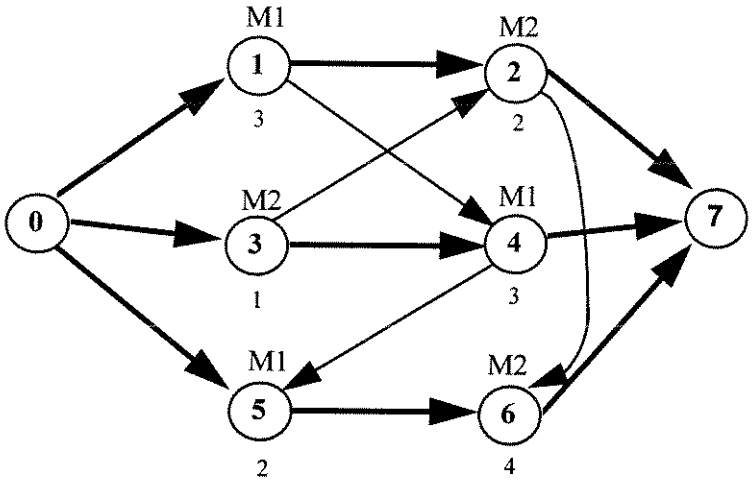


Figura 1.2. - Grafo direcionado com 3 tarefas e 2 máquinas

O objetivo deste trabalho é encontrar a programação das operações nas máquinas que minimize o atraso total das tarefas, isto é, encontrar a configuração do grafo acíclico que resulte no menor atraso. O atraso de cada tarefa é calculado em relação ao seu instante de término e sua data de entrega.

1.2.1. Revisão Bibliográfica

Sabe-se que o problema de programação em um *job shop* tradicional é um dos problemas mais difíceis de otimização combinatória (Lawler et al., 1993) e tem sido amplamente estudado, principalmente para a medida de desempenho associada ao tempo total para completar todas as tarefas, isto é, o *makespan*. Quando o objetivo é minimizar a soma dos atrasos das tarefas em relação às datas de entrega, o problema é, em geral, atacado através de regras de despacho.

A seguir é apresentada a revisão bibliográfica do problema. Como o objetivo deste trabalho é minimizar o atraso total das tarefas, a revisão para outros objetivos contém somente os trabalhos mais importantes. Vale ressaltar que nem todos os trabalhos consideram, para o problema tradicional, a existência de **m** operações por tarefa e que operações de uma mesma

tarefa devam ser executadas em máquinas diferentes, porém, consideram que duas operações consecutivas de uma tarefa nunca podem ser feitas pela mesma máquina.

Para o caso onde considera-se a minimização do *makespan* muita pesquisa já foi realizada, sendo a maior parte direcionada a métodos heurísticos de resolução devido a dificuldade do problema. Quanto aos métodos ótimos a maioria dos trabalhos utiliza a técnica *branch-and-bound.* Vaessens et al. (1996) apresentam resultados sobre a complexidade computacional do problema que é mostrada na tabela a seguir, onde l_j denota o número de operações da tarefa j, t(v) o valor inteiro do tempo de processamento da operação v, e comprimento é o comprimento do caminho crítico do grafo.

Tabela 1.1.- A Complexidade do Problema de *Job Shop* (extraída de Vaessens et al., 1996)

Resolvidos em tempo polinomial	*NP-Hard* (*in the strong sense**)
(1a) m=2, todos $l_j \leq 2$	(1b) m=2, todos $l_j \leq 3$ m=3, todos $l_j \leq 2$
(2a) m=2, todos t(v)=1	(2b)* m=2, todos t(v) ≤ 2 m=3, todos t(v)=1
(3a) n=2	(3b) n=3
(4a) comprimento ≤ 3	(4b)* comprimento ≤ 4

(1a) Jackson, 1956; (1b) Lenstra et al., 1977; (2a) Hefetz e Adiri, 1982;
(2b) Lenstra e Rinnoy Kan, 1979; (3a) Akers, 1956 e Brucker, 1988;
(3b) Sotskov, 1991; (4a,b) Williamson et al..

Carlier e Pinson (1989) e Applegate e Cook (1991) desenvolveram algoritmos *branch-and-bound* baseados na resolução do problema de seqüenciamento de tarefas em uma máquina. Tem-se uma árvore com m nós, um para cada máquina, e a cada um deles está associado um problema de seqüenciamento que deve ser resolvido. A cada nó não sondado, dois novos problemas são criados onde uma tarefa crítica e um conjunto crítico de tarefas são escolhidos tal que em um dos problemas a tarefa é escolhida para ser processada antes de todas as outras do conjunto crítico e no outro, depois de todas do conjunto crítico. A principal diferença entre os dois métodos está na escolha da tarefa e do conjunto críticos. Carlier e Pinson testaram problemas onde o número de tarefas varia de 6 a 20 e o número de máquinas de 4 a 10, sendo que os maiores problemas possuem dimensão 100 (tarefas×máquinas). Já Applegate e Cook só testaram problemas com 10 tarefas e 10 máquinas.

Quanto aos métodos heurísticos, encontra-se na literatura métodos construídos especificamente para o problema e métodos baseados em busca local, além de regras de despacho e suas combinações (Blackstone et al., 1982).

Adams et al. (1988) desenvolveram um algoritmo heurístico, onde utiliza-se um método ótimo para resolver um problema de seqüenciamento para cada máquina. Cada vez que uma máquina é seqüenciada, reotimiza-se a seqüência de cada máquina anteriormente seqüenciada buscando melhorias. A principal contribuição de tal método é em relação à decisão da ordem na qual as máquinas serão seqüenciadas. Os problemas de uma máquina são resolvidos pelo algoritmo de Carlier (1982). Applegate e Cook (1991) também desenvolveram um algoritmo heurístico baseado nesta mesma idéia. Em Dauzere-Peres e Lassere (1993), o algoritmo de Carlier para uma máquina foi modificado, para considerar a dependência entre as tarefas, e aplicado novamente no algoritmo de Adams et al.. Os problemas testados por Adams et al. (1988) variam de 4 a 50 tarefas e 5 a 15 máquinas com a maior dimensão igual a 500. Dauzere-Peres e Lassere (1993) utilizam problemas de 6 a 20 tarefas e de 5 a 10 máquinas com a maior dimensão sendo 200.

Quando se trata de algoritmos baseados em busca local e, portanto, que necessitam da definição de vizinhança, a maioria dos trabalhos considera uma vizinhança gerada pelo caminho crítico do grafo. Van Laarhoven et al. (1992) desenvolveram um algoritmo de *simulated annealing* e os resultados foram comparados com Adams et al. (1988) que se mostrou superior, pois o algoritmo de *simulated annealing* é muito custoso computacionalmente, apesar de sua simplicidade de implementação. Dell'Amico e Trubian (1993), Taillard (1994), Barnes e Chambers (1995) e Nowicki e Smutnicki (1996) apresentam algoritmos de busca tabu, os quais são detalhados no capítulo 2.

Della Croce et al. (1995) apresentam um algoritmo genético onde cada cromossomo é composto por m subcromossomos, um para cada máquina, e cada subcromossomo é um registro de números onde cada número identifica uma operação a ser feita na máquina. Os subcromossomos não descrevem a ordem das operações nas máquinas, apenas sugerem uma precedência baseados numa lista de preferência. O uso de tal lista de preferência vem do fato que nem sempre a ordem dos subcromossomos resulta em uma solução factível. O procedimento através do qual a programação das operações é deduzida do cromossomo é semelhante à regras de despacho, porém pode ser usada uma regra diferente para cada

máquina, isto é, para cada subcromossomo. Segundo os autores os resultados obtidos são comparáveis aos de Adams et al. (1988), Van Laarhoven et al. (1992) e Dell'Amico e Trubian (1993), só que com um custo computacional muito maior. Dagli e Sittisathanchai (1995) também apresentam um algoritmo genético onde o cromossomo mostra a seqüência das operações e a interpretação desta seqüência é que define os instantes de início e término para cada operação. Os cromossomos são construídos levando-se em consideração as restrições de precedência das operações, isto é, uma operação só pode ser seqüenciada se a sua predecessora naquela tarefa já acabou de ser executada, e isso implica que só são geradas soluções factíveis. O algoritmo foi comparado com regras de despacho, as quais se mostraram inferiores.

Em Aarts et al. (1994) são apresentadas comparações de *simulated annealing*, algoritmo genético, algoritmo de limiar (*threshold algorithm*) e algoritmo de múltiplo início (*multi start algorithm*), todos estes implementados pelos autores, porém baseados em outros trabalhos. Também são apresentados resultados dos trabalhos de Applegate e Cook (1991) e de busca tabu de Dell'Amico e Trubian (1993). Dos resultados obtidos os autores concluem que busca tabu é o melhor algoritmo tanto em relação à qualidade das soluções finais como em tempo de execução para obter tais soluções.

Vaessens et al. (1996) também apresentam uma comparação de vários algoritmos ótimos e heurísticos, só que neste caso os resultados foram retirados da literatura. Os autores concluem que os melhores resultados em termos de qualidade de solução e tempo computacional são obtidos por busca tabu e afirmam que, para *job shop*, *simulated annealing* não parece ser atrativa, a menos que tempo computacional não seja um limitante, e que algoritmos genéticos obtiveram resultados muito pobres até agora.

Todos esses trabalhos baseados em busca local usam problemas-testes com número de tarefas variando de 5 a 30 e número de máquinas de 5 a 15 sendo a dimensão máxima igual a 300. As instâncias encontradas em Fisher e Thompson (1963), Lawrence (1984), Adams et al. (1988) e Applegate e Cook (1991) são, em geral, as usadas para comparação.

Outros trabalhos, ainda para *makespan*, podem ser encontrados em Blazewicz et al. (1996), onde é apresentada uma extensa revisão dos métodos ótimos e heurísticos já desenvolvidos.

Em relação ao objetivo de minimização do atraso total das tarefas, foi provado por Du e Leung (1990) que o problema para uma única máquina com **n** tarefas é *NP-Hard*. Koulamas (1994) provou que para o mesmo objetivo o problema *flowshop* com 2 máquinas é *NP-Hard* (*in the strong sense*) e que tal resultado pode ser usado para provar que para *job shop* também o é. A literatura para este objetivo é bastante escassa e a maioria dos trabalhos trata de regras de despacho que sabe-se serem computacionalmente eficientes, apesar de apresentarem desempenho pouco previsível (Kanet e Hayya, 1982; Baker e Kanet, 1983; Baker, 1984; Vepsalainen e Morton, 1987; Han e McGinnis, 1989; Anderson e Nyirenda, 1990; Raghu e Rajendran, 1993). Métodos heurísticos mais elaborados que procuram superar as deficiências de regras de despacho são encontrados em He et al. (1993) e Raman e Talbot (1993).

A heurística proposta por He et al. parte de uma solução inicial, dada por uma regra de despacho, e tenta mudar a programação escolhendo determinadas operações de tarefas atrasadas para serem iniciadas antes do seu instante atual de início para rearranjar outras operações a fim de obter uma melhoria na medida de desempenho. O método foi testado para problemas com 60, 70, 80, 90 e 100 tarefas e 30 máquinas. Os resultados mostram que a heurística obteve um pequeno ganho em relação às soluções iniciais, obtidas por várias regras de despacho. Raman e Talbot apresentam um método de melhoria da programação das operações gerada pela regra de despacho *modified operation due date* - MOD (Baker e Kanet, 1983). Para cada máquina é feita uma tentativa de melhorar sua programação através de modificações apropriadas das datas de entrega de operações pertencentes a tarefas atrasadas. Para cada modificação é feita uma nova programação para o problema todo com a regra MOD. Este método heurístico foi testado para problemas com 15 e 35 tarefas e 5 e 10 máquinas e segundo os autores os resultados mostram que, utilizando-se um esforço computacional extra, melhorias significativas em relação a regras de despacho são obtidas pelo método proposto. No trabalho também foi desenvolvido um algoritmo ótimo de enumeração implícita testado apenas para problemas bem pequenos, de 5 tarefas e 3 máquinas.

Pelos trabalhos descritos pode-se notar que o objetivo de minimização do atraso total foi bem pouco estudado e que as aplicações de busca tabu para *job shop* para o critério *makespan* obtiveram bastante sucesso. Tais considerações foram portanto um estímulo para o nosso trabalho que consiste na aplicação de busca tabu no problema considerando atraso. Os

detalhes da heurística desenvolvida são apresentados no capítulo 3.

1.3. O Problema Flexível

A principal diferença do *job shop* flexível em relação ao tradicional é que para cada operação existe um conjunto de máquinas onde esta pode ser executada e ainda, em geral, o número de operações por tarefa é distinto. Portanto o problema de programação de tarefas em um *job shop* flexível consiste de:

- um subproblema de roteamento que consiste em alocar cada operação de uma tarefa a uma máquina;

- um subproblema de programação de operações em cada máquina de modo a obter um programa (*schedule*) que minimize a medida de desempenho considerada. Este subproblema consiste na resolução de um problema tradicional de *job shop*, porém com a particularidade que duas ou mais operações de uma mesma tarefa podem ser executadas numa mesma máquina consecutivamente.

O critério de otimalidade aqui considerado também é a minimização da soma dos atrasos das tarefas.

1.3.1. Revisão Bibliográfica

Como para este problema pouco há na literatura, a revisão bibliográfica a seguir engloba várias medidas de desempenho. Regras de despacho adaptadas do problema tradicional e outras desenvolvidas para sistemas flexíveis de manufatura são utilizadas para resolver o problema (Montazeri e Van Wassenhove, 1990).

Para minimização do *makespan*, Chang et al. (1989) apresentam um método heurístico do tipo enumeração parcial (*beam search*). O método tenta eficientemente explorar a árvore de busca onde cada nó da árvore é uma seqüência parcial de operações, que são escolhidas por possuírem determinadas características. Os experimentos computacionais consideraram problemas de 10 tarefas e 10 máquinas, com variação da flexibilidade de 1 a 10, isto é, o número de máquinas que são alternativas a cada operação e com tempos de processamento das

operações nas máquinas alternativas iguais e diferentes. O método foi comparado com regras de despacho e em geral se mostrou superior, porém com um tempo significativamente maior. Em Chang e Sullivan (1990), é apresentado um algoritmo para geração de todos os programas ativos para o problema flexível. Programas ativos são programas factíveis onde nenhuma operação pode ser iniciada antes de seu instante atual sem que outras operações sejam deslocadas para um instante de início maior que o atual. A conclusão dos autores é que o algoritmo desenvolvido é impraticável mesmo para problemas pequenos, o que motivou-os a desenvolver um outro algoritmo que reduz o número de tais programas. Os dois algoritmos foram então testados para problemas onde o número de máquinas e de tarefas são iguais e variam de 2 a 5, o número de máquinas alternativas para cada operação é igual a m e as operações possuem tempos de processamento diferentes para máquinas alternativas. Pelos resultados, pode-se notar que o segundo algoritmo tem um desempenho muito bom e com um tempo computacional bem menor, em comparação com o primeiro.

Ainda para *makespan*, métodos heurísticos baseados em busca tabu são apresentados por Brandimarte (1993), Hurink et al. (1995) e Logendran e Sonthinen (1997). Brandimarte utiliza uma técnica hierárquica de resolução, que inicia com uma solução gerada através de regras de despacho, o que resulta numa fixação de um roteiro e numa programação das operações nas máquinas, e então são feitas tentativas de melhorar a programação, mantendo o roteiro inicial fixo. Um novo roteamento é feito depois de algumas tentativas de obter uma melhor programação. Técnicas hierárquicas baseiam-se na decomposição do problema original a fim de reduzir sua complexidade e são muito usadas em sistemas flexíveis de manufatura (Sawik, 1990; Hutchison et al. 1991; Brandimarte e Calderini, 1995). Brandimarte (1993) também propõe outro método que pode ser considerado como um algoritmo de busca tabu de múltiplo início e que foi testado também para a medida de soma ponderada dos atrasos. Em Hurink et al. (1995), as principais diferenças em relação ao método hierárquico de Brandimarte (1993) são a vizinhança utilizada e a consideração do roteamento das operações em cada passo do algoritmo. Logendran e Sonthinen (1997) consideram para o problema flexível, que além das operações possuírem um conjunto de máquinas onde podem ser processadas, também as tarefas possuem planos de processamento alternativos. Como exemplo, dois planos de processamento alternativos, para uma tarefa qualquer, podem ser processar somente as operações 1 e 2 da tarefa (plano A) ou processar somente as operações 2

e 3 da tarefa (plano B). A resolução do problema também é feita de maneira hierárquica, onde para um determinado plano de processamento, escolhe-se o melhor roteamento, isto é, aquele que resulta no menor valor para a medida de desempenho. Maiores detalhes sobre estes métodos de busca tabu são apresentados no capítulo 2.

Nasr e Elsayed (1990) apresentam um método heurístico que resolve o problema através de sua decomposição em subproblemas de designação (*assignment*) que são mais fáceis de resolver. O critério de otimalidade considerado é a minimização do tempo médio de fluxo das tarefas. Os resultados apresentados são para um problema de 10 tarefas e 10 máquinas e com variação da flexibilidade de 1 a 6, ou seja, cada operação tendo de 1 a 6 máquinas alternativas. Em Hutchison et al. (1991) três métodos são apresentados. O primeiro é um método ótimo onde os dois subproblemas - roteamento e programação - são resolvidos simultaneamente. O problema é formulado por programação inteira mista com o objetivo de minimizar o *makespan* e resolvido através de um método *branch-and-bound*. O segundo método resolve otimamente cada subproblema também através de algoritmos baseados em *branch-and-bound*, sendo que o objetivo do subproblema de roteamento é minimizar o máximo carregamento das máquinas e o de programação é minimizar o *makespan*. O terceiro método consiste da aplicação da regra de despacho *shortest processing time* (SPT) considerando também o carregamento das máquinas. Os testes para comparação dos métodos foram feitos para problemas pequenos, de 8 tarefas e 7 máquinas com no máximo duas máquinas alternativas por operação, principalmente por causa das características do primeiro método. Pelos resultados, os autores concluem que o primeiro e o segundo método tiveram um desempenho qualitativo muito melhor que o terceiro. A qualidade de solução do primeiro método é um pouco melhor que o segundo, mas dada a imensa diferença em termos de tempo computacional, o segundo método se mostrou mais apropriado para o ambiente flexível. Isto mostra que a perda em desempenho resultante da utilização de uma técnica hierárquica pode ser bem pequena. Hoitomt et al. (1993) apresentam um método que utiliza a relaxação Lagrangeana aumentada com relação às restrições de precedência das operações e de capacidade das máquinas. O problema lagrangeano resultante é decomposto em n subproblemas, um para cada tarefa, que são resolvidos e fornecem o instante de início do processamento das operações e a máquina que vai processá-las. Como esses valores obtidos podem levar a um programa infactível, pois algumas restrições de precedência ou de

capacidade podem estar violadas, eles são usados somente como base para a aplicação de um algoritmo do tipo regra de despacho que gera uma solução factível para o problema. A função objetivo considerada é a minimização do atraso quadrático ponderado. O método foi implementado na prática para problemas com 228 tarefas, cada uma com no máximo 7 operações, 32 máquinas e cada operação com 6 máquinas alternativas. Segundo os autores o método obteve resultados muito bons.

Estes trabalhos existentes para *job shop* flexível sugerem que esquemas hierárquicos apresentam bons resultados e novamente pode-se constatar que pouco há para o objetivo de minimizar o atraso das tarefas. Isso motivou o desenvolvimento de uma heurística de resolução hierárquica para o problema. Esta heurística também é baseada em busca tabu e no capítulo 3 ela é descrita com detalhes.

CAPÍTULO 2

BUSCA TABU E SEUS FUNDAMENTOS

2.1. Introdução

Busca Tabu é uma metaheurística baseada em busca local que procura explorar o espaço de soluções movendo-se sucessivamente de uma solução para a melhor dentre sua vizinhança. Com o objetivo de superar ótimos locais o valor da função objetivo pode deteriorar e para prevenir ciclagem, movimentos recentemente realizados são proibidos. Busca Tabu (BT) foi desenvolvida por Glover (1989) para resolver problemas de otimização combinatória e tem sido aplicada com sucesso a uma variedade de problemas (Glover e Laguna, 1993; Glover, 1996). Os principais elementos para o sucesso de BT são a incorporação de memória adaptativa, que basicamente pode ser dividida em memória de curto e de longo prazo, e de estratégia de busca, que pressupõe que uma busca informada é melhor que uma boa escolha aleatória. Neste capítulo os componentes de BT são apresentados e também é feita uma revisão detalhada dos trabalhos onde ela foi aplicada para o problema de *job shop*.

2.2. Memória de Curto Prazo

O método de BT requer que cada solução tenha associada a ela uma vizinhança que pode ser determinada a partir de um movimento, que é uma operação feita para se levar da solução a uma outra. A BT age buscando sempre a melhor solução na vizinhança e também proibindo movimentos que possuam certos atributos (características dos movimentos) com o objetivo de prevenir ciclagem e explorar regiões novas. A proibição ocorre através do armazenamento dos atributos dos movimentos já realizados em uma lista tabu que é

consultada a cada vez que se avalia um novo movimento. Caso os atributos deste movimento estejam presentes na lista tabu, ele é proibido de ser executado e, portanto, é considerado um movimento tabu. Uma maneira de retirar a condição tabu de um movimento é pelo critério de aspiração que libera um movimento considerado suficientemente atrativo naquele momento da busca. Todos estes elementos da BT estão relacionados à memória de curto prazo que é assim chamada, pois baseia-se na história recente da busca.

2.3. Memória de Longo Prazo

Os principais componentes da memória de longo prazo são intensificação e diversificação. Depois de um certo tempo e de algumas soluções armazenadas, pode-se tentar diversificar a busca para regiões não exploradas (por exemplo, incentivando/penalizando o aparecimento de características não freqüentes/freqüentes nas soluções) ou intensificar a busca em regiões atrativas (por exemplo, incentivando regiões onde as melhores soluções apresentam características comuns). Tais estratégias são usadas pois nem sempre a memória de curto prazo é suficiente para alcançar soluções de boa qualidade e, segundo Glover (1995) e Laguna (1995), em geral, o uso de memória de longo prazo não requer muito tempo de execução para que seu benefício se torne visível.

A estratégia de diversificação está relacionada com a tentativa de guiar a busca para novas regiões. Em geral, é baseada em medidas de freqüência de atributos das soluções obtidas durante o processo de busca. Os atributos que freqüentemente ocorreram até então são penalizados, tentando assim atingir novas soluções. Segundo Glover (1995), a estratégia de diversificação não deve ser aplicada arbitrariamente, e sim quando a busca parece falhar em encontrar soluções melhores que a melhor solução encontrada até o momento, isto é, quando a busca não consegue atualizar a solução incumbente por um certo período de tempo.

Para a estratégia de intensificação, a idéia principal é retornar a busca para regiões consideradas promissoras. Para isto também é usada uma medida de freqüência de atributos das soluções, porém, neste caso, a freqüência está relacionada somente às melhores soluções encontradas durante a busca, denominadas soluções de elite.

Existem várias maneiras de se implementar tais estratégias e elas podem atuar

combinadas ou sozinhas. A seguir são detalhados os trabalhos onde foram feitas implementações de BT para o problema de *job shop*.

2.4. Descrição dos Trabalhos de BT para *Job Shop*

Para o problema tradicional, os trabalhos citados a seguir tem por objetivo minimizar o *makespan*. Os elementos de BT utilizados em tais trabalhos e que são apresentados a seguir se baseiam na representação do problema pelo grafo, e a notação (i,j) representa a direção do arco disjuntivo, isto é, a operação i é processada pela máquina antes da operação j. Antes porém, são apresentadas duas propriedades em relação ao grafo (Balas, 1969; Van Laarhoven et al., 1992). Dada uma solução factível para um grafo:

1) a inversão de um arco disjuntivo que pertença ao caminho crítico deste grafo resulta em uma outra solução factível.

2) se a inversão de um arco disjuntivo que não pertence ao caminho crítico do grafo resultar em uma solução factível, então o comprimento do caminho crítico do grafo desta nova solução é maior ou igual ao comprimento do caminho crítico do grafo da solução factível anterior.

Tais propriedades são importantes quando se trabalha com *makespan* e com base nelas define-se os elementos da seguinte forma:

- movimento - corresponde à inversão de um arco disjuntivo no caminho crítico do grafo. A cada movimento associa-se um valor, chamado de valor do movimento, que representa a mudança no valor da função objetivo em conseqüência da sua realização, isto é, a diferença entre a solução atual e a solução anterior.
- atributo do movimento - o par de operações i,j que compõe o arco disjuntivo.
- vizinhança (V1)- uma solução vizinha é obtida pela execução de um movimento, portanto, a vizinhança é composta por todas as soluções alcançadas pela inversão de arcos disjuntivos no caminho crítico do grafo.
- lista tabu - é composta por atributos dos arcos que foram invertidos. Por exemplo, se o arco (2,8) foi invertido então o par 2,8 é armazenado na lista tabu por uma certa duração.

- restrição tabu - um movimento está proibido se o atributo reverso dele está na lista tabu. Por exemplo, ao tentar inverter o arco (8,2) deve-se consultar a lista para saber se o seu atributo reverso 2,8 está presente. Em caso afirmativo, o arco (8,2) não pode ser invertido.
- critério de aspiração - um movimento tabu é liberado se o valor da função objetivo resultante de sua execução superar o valor da solução incumbente.

Estes elementos constituem a forma básica da memória de curto prazo da BT utilizada nos trabalhos, porém, em alguns são apresentadas outras vizinhanças que são consideradas extensões ou reduções da vizinhança V1.

Em Dell'Amico e Trubian (1993) são desenvolvidas extensões da vizinhança V1, onde considera-se a inversão de mais de um arco disjuntivo simultaneamente, e no trabalho é utilizada uma vizinhança baseada na troca de operações que pertencem a um bloco. A idéia de bloco foi desenvolvida por Grabowski (1983) para o problema de *flowshop*. Um bloco é a maior seqüência de operações adjacentes que são processadas na mesma máquina e que pertencem ao caminho crítico do grafo. A vizinhança utilizada pelos autores consiste em inserir uma operação de um bloco imediatamente antes ou depois de todas as operações do bloco. Caso as soluções resultem infactíveis, insere-se a operação dentro do bloco o mais próximo possível do início ou fim do bloco, tal que a factibilidade seja preservada. Isto é feito para todas as operações de todos os blocos e para prevenir a geração de soluções infactíveis um teste de factibilidade foi elaborado. Além disso, é sugerida uma implementação dinâmica para a lista tabu, isto é, onde a duração da proibição varia de acordo com a região da busca. Por exemplo, se a solução encontrada tiver melhor valor de função objetivo que a anterior os autores consideram que a região é promissora e por isso a duração da proibição do atributo deve diminuir e vice-versa. Apresentam ainda uma estratégia de intensificação que reinicia a busca toda vez que a solução incumbente não é atualizada por um certo número de iterações. O reinício parte sempre da solução incumbente. Esta aplicação de BT foi comparada com regras de despacho e se mostrou bem superior a elas.

Taillard (1994) utiliza memória baseada em freqüência para diversificar a busca e, segundo o autor, tal estratégia foi bem sucedida sem aumento do tempo computacional. Também apresenta uma versão paralela para BT e utiliza lista tabu aleatória onde a duração da proibição é sorteada num determinado intervalo. Barnes e Chambers (1995) apresentam uma

implementação de BT com diversificação e intensificação. As soluções vão sendo armazenadas durante a busca e a diversificação ocorre, pois é feito um reinício da busca em várias soluções enquanto que a intensificação é alcançada porque tais soluções são as de elite. A cada reinício da busca a lista tabu é esvaziada. Nowicki e Smutnicki (1996) também apresentam uma estratégia de intensificação, onde depois de um certo número de iterações e registro de algumas soluções de elite e suas trajetórias de busca, reinicia-se a busca por uma das soluções de elite e utiliza-se a informação de sua trajetória para que novos caminhos sejam tentados. No trabalho é utilizada uma vizinhança baseada na troca de operações que pertencem a um bloco. Esta vizinhança é significativamente reduzida pois os movimentos que não melhoram imediatamente o *makespan* são removidos, isto é, pode-se mostrar que a troca de determinadas operações de um bloco resultam em uma solução com pelo menos o mesmo valor de *makespan*. Os resultados obtidos são comparados com outros trabalhos clássicos e se mostraram excelentes, tanto em tempo computacional como em qualidade de solução.

Os problemas-testes usados nestes trabalhos possuem de 5 a 30 tarefas e de 5 a 20 máquinas, com a maior dimensão sendo igual a 300. Os únicos trabalhos onde problemas maiores foram testados são de Taillard (1994) e de Nowicki e Smutnicki (1996) com número de tarefas até 1.000 e de máquinas até 20 sendo a maior dimensão igual a 10.000.

Para o problema flexível, o trabalho de Brandimarte (1993) usa uma resolução hierárquica, onde o problema é resolvido através de regra de despacho que define um roteiro para o subproblema de roteamento e uma solução inicial para o subproblema de programação. A partir desta solução a BT básica descrita anteriormente é aplicada sobre o subproblema de programação, ou seja, considerando fixo o roteiro. Depois de algum tempo sem melhoria na programação, volta-se ao subproblema de roteamento onde tenta-se trocar de máquina uma operação que pertença ao caminho crítico do grafo. Dentre as trocas escolhe-se a melhor mas considerando também que ela deve ser factível, pois desta troca deve-se obter uma nova solução inicial para o subproblema de programação. Volta-se então a resolver o subproblema de programação agora para este novo roteiro. Para evitar ciclagem de soluções no nível de resolução do problema de roteamento também é aplicado um procedimento tabu, onde o par {operação, máquina} fica proibido por um certo número de iterações. Vale ressaltar que as soluções infactíveis obtidas pelas trocas de operações de máquinas são descartadas. Esta implementação hierárquica foi testada para *makespan*. No trabalho também é apresentada uma

implementação de múltiplo início, onde tem-se uma solução inicial, para ambos os subproblemas, dada por uma regra de despacho e daí é aplicada BT básica somente no subproblema de programação. Depois de algumas iterações sem que haja atualização da solução incumbente, aplica-se novamente uma regra de despacho (outra regra diferente ou a mesma com outros parâmetros, se for uma regra parametrizada) e obtém-se, em geral, uma nova solução inicial com um novo roteiro. Volta-se então a resolver o subproblema de programação. Para esta segunda implementação a vizinhança está relacionada com uma amostra de todas as trocas possíveis das operações nas máquinas e também aqui as trocas que causarem soluções infactíveis são descartadas. Esta implementação foi testada tanto para *makespan* como para o atraso ponderado. Os problemas testados possuem de 10 a 30 tarefas e de 4 a 15 máquinas, com número de máquinas alternativas variando para cada operação e com tempo de processamento das operações diferentes em máquinas alternativas. Segundo o autor, os experimentos computacionais feitos são limitados e os resultados estão longe de serem conclusivos. Em Hurink et al. (1994) também é apresentada uma aplicação de BT para *makespan*. Dada uma solução inicial, aplica-se BT sobre uma vizinhança que leva em conta tanto trocas de operações nas máquinas como troca de operações entre máquinas. Tal vizinhança é também baseada nos blocos e faz tanto a inserção apresentada por Dell'Amico e Trubian (1993) como a inserção de operações pertencentes a um bloco em suas máquinas alternativas. Esta vizinhança pode gerar soluções infactíveis e quando isto acontece, elas são descartadas. Para os problemas-testes o número de máquinas varia de 5 a 10 e de tarefas de 6 a 30, com cada tarefa possuindo m operações e cada operação tendo no máximo 80% do total de máquinas como máquinas alternativas e com tempo igual de processamento nas mesmas. Segundo os autores, os resultados obtidos pelo método se mostraram muito bons. Logendran e Sonthinen (1997) apresentam uma heurística, baseada em busca tabu, para o problema flexível onde considera-se que as tarefas possuem planos de processamento alternativos. A heurística envolve dois níveis de resolução. No primeiro nível, escolhe-se um plano de processamento para cada tarefa, e no segundo nível, seleciona-se a melhor opção de processamento nas máquinas, isto é, aquela que resulta no menor *makespan*. A seleção é feita através de uma enumeração completa. Depois de escolhida a melhor opção, volta-se ao primeiro nível onde novos planos de processamento das tarefas são fixados. Para evitar ciclagem de soluções, aplica-se um procedimento tabu nos dois níveis. No trabalho, utiliza-se memória de longo prazo, para diversificar a busca, e lista tabu dinâmica. Além disso, os autores apresentam uma

formulação inteira do problema e tentam resolvê-la pelo pacote SuperLINDO, porém para um problema de 3 máquinas e 3 tarefas, não conseguem obter a solução ótima.

Em geral, os autores destes trabalhos de BT afirmam que a implementação dos elementos de memória de curto prazo é muito simples, e aqueles que utilizaram memória de longo prazo sugerem que, pelos resultados obtidos e também pela facilidade de implementação, a incorporação de diversificação e intensificação é um campo bastante promissor e novas estratégias devem ser pesquisadas.

CAPÍTULO 3

HEURÍSTICAS PARA O PROBLEMA DE *JOB SHOP* COM DATAS DE ENTREGA

3.1. Introdução

Este capítulo apresenta as heurísticas desenvolvidas para os problemas de *job shop* tradicional e flexível com o objetivo de minimizar o atraso total das tarefas em relação às datas de entrega. As heurísticas são compostas por duas fases, uma para obtenção de uma solução inicial e outra que busca novas e melhores soluções. Ambas heurísticas utilizam BT com memória de curto e de longo prazo.

3.2. Heurística proposta para o problema tradicional

A heurística desenvolvida para o problema tradicional, como já dito, é composta de duas fases onde a solução inicial é obtida através de regras de despacho que consideram datas de entrega, e a busca de melhores soluções consiste de uma busca na vizinhança gerada pelos caminhos críticos das tarefas. Esta segunda fase é guiada pela BT, para a qual elementos de diversificação e intensificação foram incorporados.

3.2.1. Solução inicial

A solução factível inicial é obtida através da aplicação de regras de despacho apropriadas para problemas com datas de entrega. O mecanismo para regras de despacho funciona da seguinte maneira: a cada instante que uma máquina fica desocupada, calcula-se uma prioridade para cada operação que está pronta para ser processada nesta máquina e então,

escolhe-se para processar a operação de maior prioridade. O cálculo de cada prioridade é dependente da regra em questão. Neste trabalho as regras implementadas são MDD (Baker e Kanet, 1983), MOD (Baker e Kanet, 1983; Baker, 1984), CR+SPT e S/RPT+SPT (Anderson e Nyirenda, 1990). Para a apresentação delas, considere a seguinte notação:

t - instante em que se considera a alocação de uma operação em uma máquina;

k - k-ésima operação da tarefa j que está esperando para ser processada em t;

g_j - número total de operações da tarefa j;

t_{ij} - tempo de processamento da operação i da tarefa j;

d_j - data de entrega da tarefa j;

d_{kj} - data de entrega da operação k da tarefa j;

• MDD (*modified due date*): combinação de "earliest due date" (EDD) com "least remaining processing time" (LRPT). Define-se a data de entrega da tarefa j da seguinte maneira:

$$d_j' = \max \{d_j, t + \sum_{i=k}^{gj} t_{ij}\}$$

A expressão mostra que se d_j for menor que o tempo de processamento restante da tarefa no instante t, então d_j' assume o valor do segundo termo da expressão, dado que a tarefa vai atrasar. Caso contrário, d_j' assume o valor de d_j.

• MOD (*modified operation due date*): é a versão da MDD considerando-se como base as operações e não a tarefa. A data de entrega da k-ésima operação da tarefa j é definida da seguinte maneira:

$$d_{kj}' = \max \{d_{kj}, t + t_{kj}\}$$

A data de entrega da operação k, d_{kj}, é calculada através de fórmulas recursivas e neste trabalho é usada a seguinte:

$$d_{kj} = d_{k-1,j} + d_j \cdot t_{kj} / \sum_{i=1}^{gj} t_{ij}$$

• CR+SPT: combinação de "critical ratio" (CR) com "shortest processing time" (SPT). Esta regra foi desenvolvida a partir da regra MOD para uma máquina e a data de entrega da k-ésima operação da tarefa j é definida como:

$$d_{kj} = \max\{t + \beta(t).t_{kj},\ t + t_{kj}\} \qquad \text{com} \qquad \beta(t) = a_j(t)/\sum_{i=k}^{gj} t_{ij} \ \text{ e } \ a_j(t) = d_j - t$$

O termo $\beta(t)$ representa a "critical ratio" que é a razão entre o tempo para atingir a data de entrega e o tempo de processamento restante da tarefa. Quando $\beta(t)$ for menor que 1, d_{kj} assume o valor do segundo termo da expressão, dado que a tarefa vai atrasar.

• S/RPT + SPT: combinação de "slack per remaining processing time" (S/RPT) com SPT. Esta regra difere da anterior, pois no lugar de CR usa-se S/RPT e, neste caso, a data de entrega da k-ésima operação é definida como:

$$d_{kj} = \max\{t + \gamma(t).t_{kj},\ t + t_{kj}\} \qquad \text{com} \qquad \gamma(t) = (d_j - t - \sum_{i=k}^{gj} t_{ij}) / \sum_{i=k}^{gj} t_{ij} = \beta(t) - 1$$

O termo $\gamma(t)$ representa a "slack per remaining processing time" que é a razão entre a data de entrega menos o tempo de término da tarefa e o tempo de processamento restante da tarefa.

A escolha de tais regras se baseou na simplicidade e eficiência apresentada ao se trabalhar com atraso (Anderson e Nyirenda, 1990).

O procedimento de obtenção da solução inicial pode ser resumido nos seguintes passos. Considere para isso, t[m] como o instante em que a máquina m está disponível, M o número total de máquinas e que uma máquina está marcada se ela está ociosa. Além disso, define-se que a operação de maior prioridade é aquela associada ao menor valor para a regra de despacho.

Passo 1 - Faça t[k]=0, para k=1,...,M; t=0 e m=1.

Passo 2 - Se t[m]≤t ou se a máquina m estiver marcada, calcule a prioridade, de acordo com a regra de despacho escolhida, das operações que estão prontas para serem seqüenciadas na máquina m e aloque a esta máquina a operação de maior prioridade (operação r). Atualizar $t[m]=t+t_{rj}$. Se não existe nenhuma operação que possa ser processada pela máquina m no instante t, então marque a máquina m e faça t[m]=G, onde G é um número grande.

Passo 3 - Faça m=m+1 (se m>M faça m=1), $t = \min_{k=1,..,M}\{t[k]\}$ e volte ao passo 2 até que todas as operações sejam seqüenciadas.

No passo 2, a menção às operações que estão prontas significa que as suas predecessoras já foram executadas nas suas respectivas máquinas.

Este procedimento obtém uma programação das operações nas máquinas que é usada como partida para a busca de melhores soluções. O exemplo a seguir ilustra o procedimento. Considere 3 tarefas, 2 máquinas (M1 e M2) e cada tarefa com 2 operações, e os seguintes dados:

Tabela 3.1. Tempos de processamento e data de entrega das tarefas

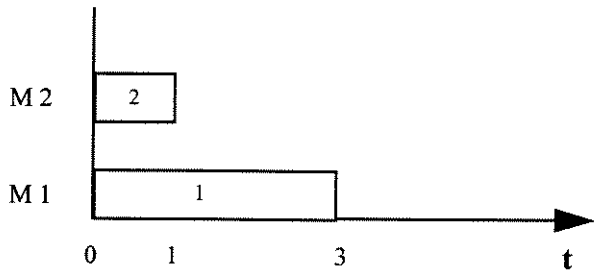
tarefa	tempo de processamento da operação		data de entrega
	1	2	
1	3	2	5
2	1	3	3
3	2	4	3

Tabela 3.2. Roteiro das tarefas

tarefa	operação	
	1	2
1	M1	M2
2	M2	M1
3	M1	M2

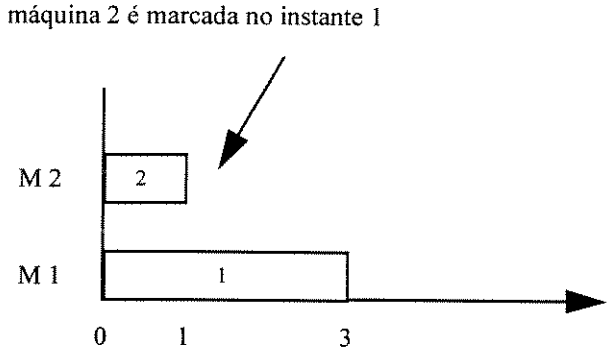
As operações são seqüenciadas pela regra MDD e os programas parciais são representados através de gráficos de Gantt.

Inicia-se a alocação em t = 0. Neste instante, M1 e M2 estão disponíveis. A operação 1 das tarefas 1 e 3 estão prontas para serem processadas em M1. Como a data de entrega da tarefa 1 é d_1'=max{5,5}=5 e da tarefa 3 é d_3'=max{6,6}=6, então em M1 é processada a operação 1 da tarefa 1. Para M2 só está disponível a operação 1 da tarefa 2, que é então escolhida para ser processada.

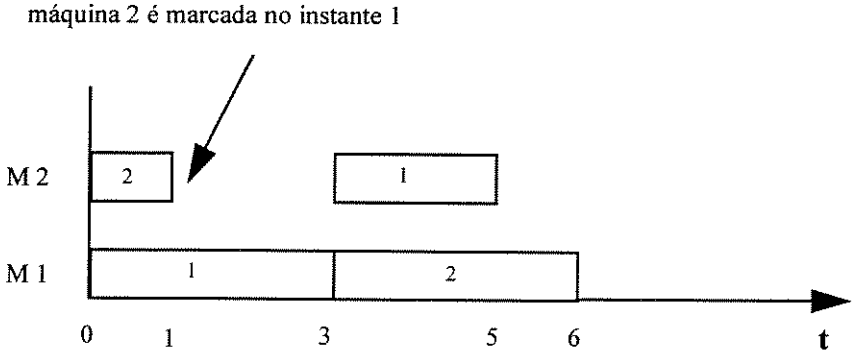


- o próximo instante que uma máquina fica disponível é t=1, para M2. Como não existe

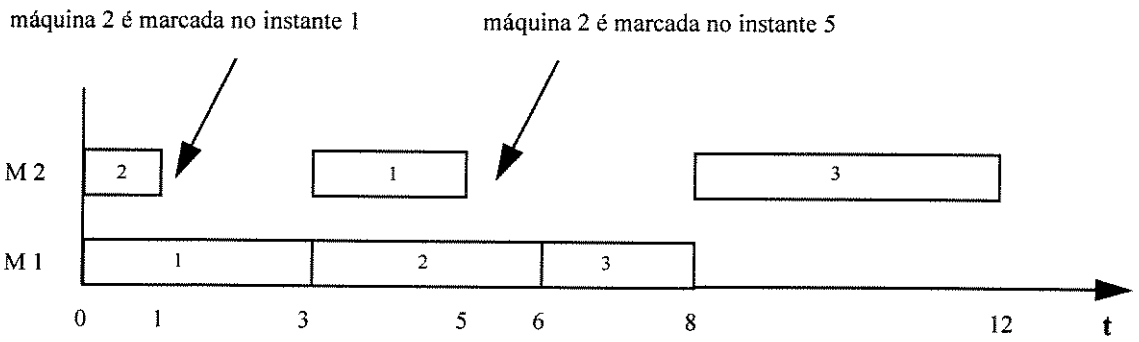
nenhuma operação disponível neste instante para ser processada nesta máquina, M2 recebe uma marca.



- em t=3, M1 está disponível e M2 está marcada. Portanto neste instante, tenta-se alocar operações nas duas máquinas. Para a máquina M1, a operação 2 da tarefa 2 e a operação 1 da tarefa 3 estão prontas para serem processadas. A data de entrega da tarefa 2 é d_2'=max{3,6}=6 e da tarefa 3 é d_3'=max{3,9}=9, portanto, em M1 é processada a operação 2 da tarefa 2. Para M2 só está disponível a operação 2 da tarefa 1 e, portanto, ela é processada.



E assim continuam-se as alocações. A solução obtida no final do procedimento pode ser vista a seguir.



Esta solução inicial possui um atraso de 12 unidades, causado pelas tarefas 2 e 3

.

3.2.2. Busca de novas e melhores soluções

A busca de novas e melhores soluções é baseada na representação do problema pelo grafo disjuntivo. Com a solução obtida na primeira fase da heurística, pode-se construir um grafo direcionado acíclico e então calcular o caminho crítico de cada uma das tarefas. Novas soluções podem ser obtidas através da inversão de arcos disjuntivos pertencentes a estes caminhos. A motivação para se usar os caminhos críticos das tarefas se baseia nas propriedades apresentadas no capítulo 2 para o caminho crítico do grafo e que podem ser facilmente estendidas para os caminhos críticos das tarefas. Isso implica que a inversão de arcos disjuntivos que pertencem ao caminho crítico de uma tarefa também não causa ciclos no grafo e que a inversão de arcos disjuntivos que não pertençam aos caminhos críticos das tarefas, se resultar em soluções factíveis, não causa melhoria na medida de desempenho. A seguir, apresenta-se a extensão do Lema enunciado em Van Laarhoven et al. (1992).

Lema 3.1: Suponha que a = (v,w) é um arco disjuntivo pertencente ao caminho crítico de uma tarefa T do grafo acíclico D_i. Seja D_j o grafo direcionado obtido de D_i pela inversão do arco a. Então D_j também é acíclico.

Prova: Suponha que D_j é cíclico. Como D_i é acíclico, o arco (w,v) é parte do ciclo em D_j. Consequentemente, existe um caminho (v,x,y,...,w) em D_j. Mas este caminho também pode ser encontrado em D_i e, claramente, é um caminho maior de v a w que o do arco (v,w) presente no caminho crítico da tarefa T em D_i. Portanto, D_j, é acíclico.

A segunda propriedade pode ser estendida diretamente e, portanto, segue-se que a inversão de um arco disjuntivo (x,y) do grafo acíclico D_i que não pertence aos caminhos críticos das tarefas, se resultar em um grafo acíclico D_j, não reduz o atraso total, pois os caminhos críticos anteriores ainda continuam em D_j, o que implica que o atraso das tarefas continua maior ou igual ao anterior.

É importante ressaltar que para o problema de *job shop* as soluções infactíveis, isto é, as configurações que acarretam ciclo no grafo, são descartadas. Sabe-se que existem algoritmos que detectam quais os arcos e, conseqüentemente, quais os nós que pertencem a um ciclo num grafo cíclico (Brassard e Bratley, 1988), porém, durante este trabalho, não foi

encontrado na literatura um algoritmo que retire um ciclo em um grafo. A inversão de um dos arcos que compõe o ciclo nem sempre é capaz de retirá-lo, como mostra o exemplo a seguir. No grafo, os arcos disjuntivos redundantes são omitidos, por exemplo o arco (3,8) é omitido pois a relação de precedência das operações na máquina M1 já fica estabelecida somente com os arcos (3,4) e (4,8).

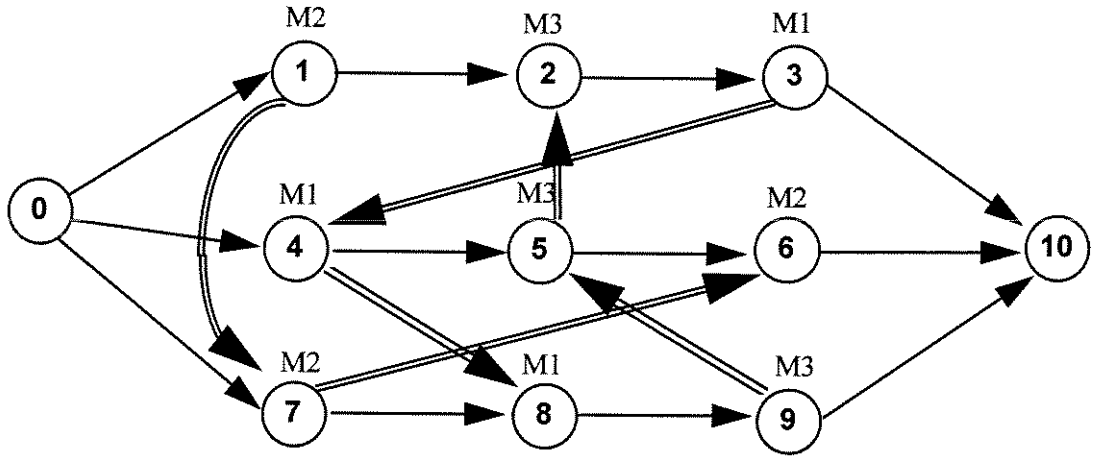


Figura 3.1. Grafo direcionado contendo ciclo

Pela Figura 3.1 tem-se que as operações 1, 7 e 6 são processadas na máquina M2, nesta ordem, as operações 9, 5 e 2 são processadas na máquina M3, nesta ordem e as operações 3, 4 e 8 são processadas na máquina M1, nesta ordem. Note que a ordem de processamento não estabelece uma precedência no grafo, compondo um ciclo através dos arcos (2,3), (3,4), (4,8), (8,9), (9,5), (5,2). A inversão de quaisquer um dos arcos disjuntivos, separadamente, que pertencem ao ciclo resulta novamente num ciclo no novo grafo. Note que a inversão dos arcos (2,3) e (8,9) não é possível, pois ambos são relacionados à precedência de operações em tarefas. Observe na figura a seguir o que ocorre quando o arco (3,4) é invertido.

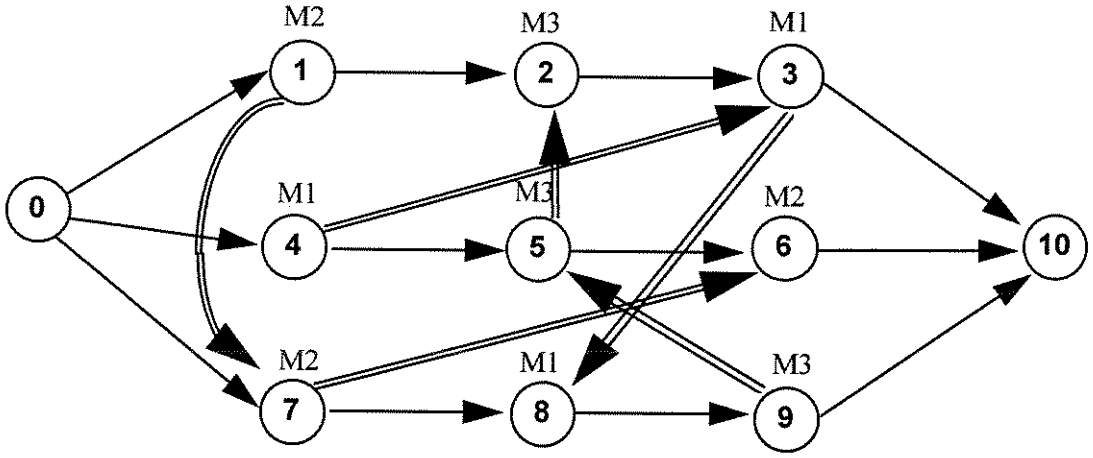


Figura 3.2. Novo grafo direcionado contendo ciclo

Ao se fazer a inversão do arco (3,4), o arco (3,8), que é omitido na solução anterior, aparece agora na Figura 3.2, pois a ordem na máquina M1 fica 4, 3 e 8, e o arco redundante passa a ser (4,8), que é omitido. O ciclo agora está nos arcos (2,3), (3,8), (8,9), (9,5), (5,2). Por este pequeno exemplo, pode-se notar que retirar um ciclo de um grafo pode não ser trivial.

O procedimento de busca de melhores soluções através dos caminhos críticos das tarefas é guiado pela BT, onde considera-se que os elementos de memória de curto prazo usados são:

- movimento - inversão de um arco disjuntivo no caminho crítico de uma dada tarefa. A cada movimento tem-se associado um valor que representa a diferença causada no atraso total devido à sua execução;
- atributo do movimento - o par de operações i,j do arco;
- vizinhança - todas as soluções obtidas pela inversão de arcos que pertencem ao caminho crítico de uma dada tarefa;
- lista tabu - é composta pelos atributos dos arcos que foram invertidos;
- restrição tabu - um movimento está proibido se o seu atributo reverso está na lista tabu;
- critério de aspiração - se um movimento tabu causar uma diminuição no valor da solução incumbente, então ele é liberado momentaneamente.

O algoritmo resumido da heurística pode então ser apresentado da seguinte maneira. Considere que J é o número total de tarefas, atraso atual é o valor do atraso total das tarefas

para a solução atual, atraso incumbente é o melhor valor para o atraso total das tarefas encontrado até o momento e atraso inicial é o valor do atraso total das tarefas da solução inicial.

Passo 1: Construa a solução inicial usando alguma regra de despacho. Faça j=1, iter=0, inicialize o atraso incumbente e o atraso atual usando o atraso inicial.

Passo 2: a) Calcule os caminhos críticos das tarefas e o atraso atual. Atualize o atraso incumbente.

b) Se nenhuma tarefa estiver atrasada ou se o número máximo de iterações for satisfeito, PARE.

Passo 3: a) Se j>J, faça j=1;

b) Se a tarefa j estiver atrasada, vá ao Passo 4.
Senão, j=j+1 e volte ao Passo 3a).

Passo 4: a) Avalie todas as inversões possíveis no caminho crítico da tarefa j e escolha uma de acordo com alguma regra (ver abaixo).

b) Execute a inversão escolhida. Faça iter=iter+1, j=j+1 e volte ao Passo 2.

c) Se não existir inversão possível, faça j=j+1 e volte ao Passo 3.

A escolha da inversão a ser executada é sempre através da regra da melhor, isto é, escolhe-se a que obtiver uma maior melhoria na função objetivo e caso não existam movimentos de melhoria, escolhe-se o de menor pioria. A escolha é sempre feita em relação às inversões não proibidas e às que satisfazem o critério de aspiração. Se todas as inversões possíveis são tabu e não passam pelo critério de aspiração, então escolhe-se a que for melhor mesmo sendo tabu. Isto é feito para não interromper a busca, e assim continuar a procurar outras soluções. O algoritmo também pára se não existirem inversões possíveis para todas as tarefas, ou seja, se todas as operações forem precedidas nos caminhos críticos pela sua própria operação predecessora da tarefa. Para se avaliar uma inversão deve-se utilizar um algoritmo que calcule os caminhos críticos das tarefas e, conseqüentemente, o atraso total.

O algoritmo que determina os caminhos críticos das tarefas foi adaptado do algoritmo apresentado por Taillard (1994) que calcula as datas de início de processamento das operações. Seus passos são resumidos a seguir, onde r_i denota o início de processamento da

operação i, t_i o seu tempo de processamento, PM[i] a operação que precede i na mesma máquina, PJ[i] a operação que precede i na sua tarefa, SM[i] a operação que sucede i na mesma máquina, SJ[i] a operação que sucede i na sua tarefa e cc[i] a operação que precede i no caminho crítico.

Passo 0) Coloque em Q as operações i que não possuem operação predecessora na tarefa ou na máquina.

Passo 1) Repita até que Q=∅:

a) escolher e marcar $i \in Q$;

b) $Q \leftarrow Q/\{i\}$;

c) calcule r_i onde $r_i = \max(r_k, r_j)$, $r_k = r_{PJ[i]}+t_{PJ[i]}$ e $r_j = r_{PM[i]}+t_{PM[i]}$;

Se $r_k \geq r_j$ então $r_i=r_k$ e cc[i]=k;

senão $r_i=r_j$ e cc[i]=j;

d) se PM[SJ[i]] não existe ou está marcado então $Q \leftarrow Q \cup \{SJ[i]\}$;

e) se PJ[SM[i]] não existe ou está marcado então $Q \leftarrow Q \cup \{SM[i]\}$;

Para obter o caminho crítico de uma dada tarefa basta percorrer cc[i], iniciando-se da última operação desta tarefa até sua operação inicial.

O exemplo a seguir, usando a solução inicial obtida no exemplo da seção 3.1.1., mostra como é calculado o caminho crítico de cada tarefa. Os passos do algoritmo são seguidos através do grafo da solução. Na representação pelo grafo, tem-se que os nós 1 e 2 representam as operações da tarefa 1, os nós 3 e 4 representam as operações da tarefa 2 e os nós 5 e 6 representam as operações da tarefa 3. No grafo, o tempo de processamento de cada operação está representado logo abaixo de seu respectivo nó e os arcos disjuntivos redundantes não foram desenhados.

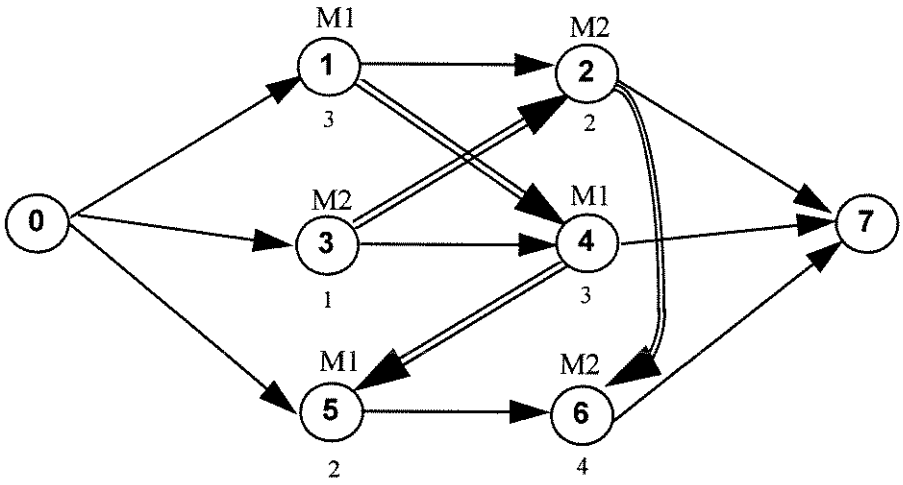


Figura 3.3. - Grafo direcionado com 3 tarefas e 2 máquinas

Passo 0 - Q = {1,3};
Passo 1 - Como Q ≠∅,

a) i = 1 e marcar a operação 1;
b) Q = {3};
c) $r_1 = \max(r_0) = 0$ e cc[1] = 0;
d) Q = {3};
e) Q = {3};

Como Q ≠∅,

a) i = 3 e marcar a operação 3;
b) Q = ∅;
c) $r_3 = \max(r_0) = 0$ e cc[3] = 0;
d) Q = {4};
e) Q = {4,2};

Como Q ≠∅,

a) i = 4 e marcar a operação 4;
b) Q = {2};
c) $r_4 = \max(r_3, r_1) = 3$ e cc[4] = 1;
d) Q = {2};
e) Q = {2,5};

Como Q ≠∅,

a) i = 2 e marcar a operação 2;
b) Q = {5};
c) $r_2 = \max(r_1, r_3) = 3$ e cc[2] = 1;
d) Q = {5};
e) Q = {5};

Como Q ≠ ∅,

a) i = 5 e marcar a operação 5;
b) Q = ∅;
c) $r_5 = \max(r_0, r_4) = 6$ e cc[5] = 4;
d) Q = {6};

e) Q = {6};
Como Q ≠ ∅, repetir;
a) i = 6 e marcar a operação 6;
b) Q = ∅;
c) $r_6 = \max(r_5, r_2) = 8$ e cc[6] = 5;
d) Q = ∅;
e) Q = ∅;

FIM.

O caminho crítico da tarefa 1 é composto pelos nós 2-1-0 e, portanto, pelos arcos (0,1) e (1,2), da tarefa 2 pelos nós 4-1-0, ou seja, arcos (0,1) e (1,4) e da tarefa 3 pelos nós 6-5-4-1-0 e arcos (0,1), (1,4), (4,5) e (5,6). O instante de término da tarefa 1 é $r_2+t_2=3+2=5$, da tarefa 2 é $r_4+t_4=3+3=6$, e da tarefa 3 é $r_6+t_6=8+4=12$.

3.2.3. Aplicação das estratégias de diversificação e intensificação da BT

Nesta seção são descritas as estratégias de memória de longo prazo que são incorporadas à heurística. Tais estratégias se basearam tanto em trabalhos para *job shop* com o objetivo de minimizar o *makespan* (Dell'Amico e Trubian, 1993; Taillard, 1994; Barnes e Chambers, 1995; Nowicki e Smutnicki, 1996) como em trabalhos de BT para outros problemas (Chakrapani e Skorin-Kapov, 1993; Kelly et al., 1994), e que em ambos os casos se mostraram eficientes ao serem incorporadas à BT de curto prazo.

Para intensificação são feitos dois tipos de implementação. Na primeira, usa-se sempre a solução incumbente para reiniciar a busca. Os instantes destes reinícios podem ser escolhidos de acordo com um dos seguintes critérios: quando uma nova solução incumbente é obtida ou se a busca falha em obter uma solução melhor que a incumbente por um certo número de iterações. A segunda implementação utiliza as soluções de elite que vão sendo armazenadas durante a busca. Estas soluções podem ser usadas para se fazer um reinício da busca a partir de cada uma delas ou podem ser armazenadas em uma matriz de residência. Essa matriz guarda o número de vezes que uma operação ocupa uma determinada posição na máquina em que deve ser processada e constitui uma memória baseada em freqüência. A partir desta matriz, pode-se reiniciar a busca incentivando as operações que mais aparecem em determinadas posições, isto é, incentivando os elementos da matriz que possuem maior freqüência. Para esta segunda implementação, os instantes de aplicação da intensificação

ocorrem a cada número fixo de iterações ou depois de algum tempo sem que a busca consiga atualizar a solução incumbente. A intensificação deve durar um certo número de iterações ou até que o conjunto de soluções de elite se altere.

Para a diversificação também é usada memória baseada em freqüência, porém, neste caso, armazena-se em uma matriz de residência o número de vezes que uma operação ocupa uma posição na máquina, em relação à todas as soluções obtidas durante a busca. Calcula-se então uma freqüência relativa dos elementos da matriz de residência e continua-se a busca usando uma penalidade sobre estas freqüência, visando alcançar soluções onde os elementos de maior freqüência não estejam presentes. Os instantes de aplicação da diversificação ocorrem a cada número fixo de iterações ou depois de um certo número de iterações sem atualizar a solução incumbente. A duração da diversificação segue um dos seguintes critérios: um número fixo de iterações ou até a obtenção de uma solução que seja melhor que sua predecessora imediata.

Vale ressaltar que a intensificação e a diversificação, se ambas estão sendo aplicadas, nunca ocorrem simultaneamente durante a busca e que, em geral, a intensificação é aplicada depois da diversificação.

A heurística proposta e a incorporação das estratégias foram amplamente testadas, e detalhes da implementação, dos testes elaborados e os resultados são apresentadas no capítulo 4.

3.3. Heurística proposta para o problema flexível

Para o problema flexível optou-se por uma resolução hierárquica onde os subproblemas de roteamento e programação das operações são resolvidos separadamente. A solução inicial é obtida através de regra de despacho que fixa um roteiro e estabelece uma programação das operações nas máquinas. A partir daí, a fase de busca de melhores soluções é dividida em dois níveis, onde num nível superior trabalha-se com o subproblema de roteamento e num nível inferior com o subproblema de programação.

3.3.1. Solução inicial

A solução factível inicial é obtida pelas mesmas regras anteriores que são MDD, MOD, CR+SPT, S/RPT+SPT. Como cada operação pode ser processada por várias máquinas, deve-se considerar aqui como um objetivo importante, o balanceamento do carregamento das máquinas. Entende-se por carregamento a soma dos tempos de processamento das operações que já foram processadas pela máquina até o momento. Stecke (1983) e Brandimarte (1994) citam a importância de se fazer o balanceamento quando se trabalha em um ambiente flexível. Portanto, para se levar em consideração este elemento é feita uma modificação no algoritmo da solução inicial, onde agora as máquinas disponíveis no instante t são ordenadas de acordo com o valor crescente de seus carregamentos. Resumindo, tem-se:

Passo 1 - Faça t[k]=0, para k=1,...,M; t=0.

Passo 2 - Para as máquinas m=1,...,M, se t[m]≤t ou se m estiver marcada, calcule e ordene crescentemente seus carregamentos. Para cada máquina ordenada, calcule a prioridade das operações que estão prontas para serem seqüenciadas e aloque a ela a operação de maior prioridade (operação r). Atualizar $t[m]=t+t_{rj}$. Se não existe nenhuma operação que possa ser processada pela máquina m no instante t então a máquina m recebe uma marca e t[m]=G, onde G é um número grande.

Passo 3 - Faça $t = \min_{k=1,..,M}\{t[k]\}$ e volte ao passo 2 até que todas as operações sejam seqüenciadas.

Esta solução inicial pode então ser usada na busca de novas e melhores soluções.

3.3.2. Busca de novas e melhores soluções

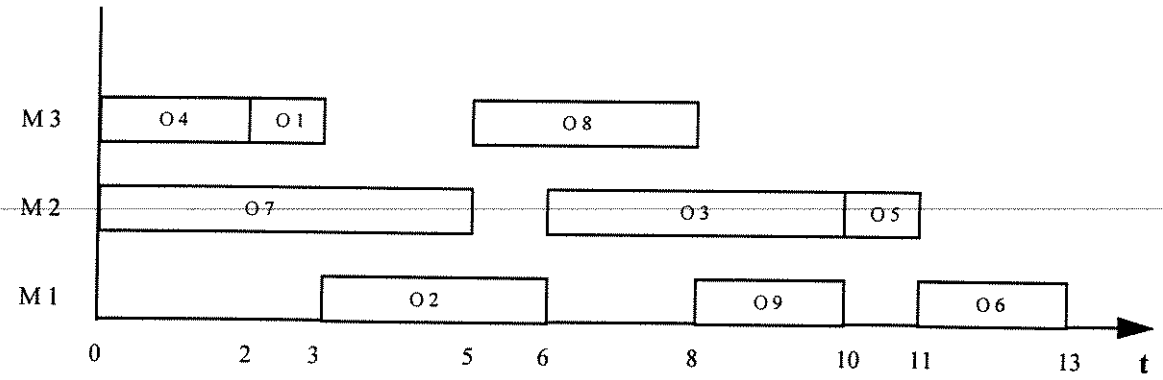
A busca de novas e melhores soluções é dividida em dois procedimentos: um que considera a inversão de arcos nos caminhos críticos das tarefas atrasadas, como feito no problema tradicional e que está relacionado ao problema do nível inferior e outro que considera a troca de uma operação da máquina em que seria processada para uma de suas máquinas alternativas, que diz respeito ao nível superior. Os elementos de memória de curto prazo da BT utilizados para o primeiro procedimento são os mesmos citados para o problema tradicional, e para o segundo são os descritos a seguir.

- movimento - troca de uma determinada operação de máquina. A operação deve pertencer ao caminho crítico de pelo menos uma tarefa.
- atributo do movimento - o par {op, m_o}, onde op é a operação que foi trocada de máquina e m_o é a máquina origem, isto é, aquela onde op era processada.
- vizinhança - todas as soluções factíveis obtidas pela troca de determinadas operações de máquinas.
- lista tabu - é composta pelos atributos {op, m_o} dos movimentos.
- restrição tabu - proíbe os movimentos que resultarem nos pares {operação, máquina} que estão presentes na lista tabu. Por exemplo, se {2,3} está presente na lista tabu, então a troca da operação 2 para a máquina 3 está proibida.
- critério de aspiração - relacionado também à melhoria do valor da solução incumbente, isto é, se a execução do movimento tabu resultar em uma solução com valor melhor que o valor incumbente, então este movimento é liberado.

É muito importante ressaltar que deste procedimento de trocas de operação de máquina, deve resultar uma solução factível, que representa uma nova solução inicial para o subproblema de programação. Para se garantir que uma solução infactível não é gerada, são utilizadas duas condições suficientes de factibilidade, que foram propostas por Dell'Amico e Trubian (1993). Tais condições foram anteriormente usadas pelos autores para impedir ciclo no grafo, isto é, soluções infactíveis no *job shop* tradicional para a vizinhança associada ao bloco, que está descrita no capítulo 2. Neste trabalho, estas mesmas condições foram então adaptadas ao contexto de troca de operação de máquina, também para impedir ciclo no grafo. As condições são baseadas no seguinte: considere que C_i denota o instante de término da operação i, e que temos no grafo direcionado a seguinte ordem para uma máquina (b', x, b"), onde b' representa o conjunto de operações que precedem a operação x na máquina e b" representa o conjunto de operações que sucedem a operação x na máquina. Para existir um ciclo no grafo é necessário que para algum $k \in b"$ a desigualdade $C_{SJ[k]} \leq C_{PJ[x]} - t_{PJ[x]}$ seja satisfeita, ou seja que a operação SJ[k] seja seqüenciada antes da operação PJ[x], ou então que para algum $k \in b'$ ocorra $C_{SJ[x]} \leq C_{PJ[k]} - t_{PJ[k]}$. Estas desigualdades são usadas para testar cada troca de uma operação candidata em cada posição de cada uma de suas máquinas alternativas. Note que a operação candidata só pode ser inserida em outra máquina em uma posição que

não viole as restrições de precedência em relação às operações de sua tarefa que são processadas naquela máquina. Esta verificação não é necessária no *job shop* tradicional. Neste trabalho, se uma operação não possui predecessor ou sucessor de tarefa, então o sucessor ou predecessor inexistente é denotado por NI com C_{NI}=G e t[NI]=G, onde G é um número grande.

A seguir, é apresentado um exemplo de tentativa de trocar uma operação de máquina e como as condições são verificadas para um problema com 3 máquinas e 3 tarefas, onde O1, O2 e O3 são operações da tarefa 1, O4, O5 e O6 operações da tarefa 2 e O7, O8 e O9 operações da tarefa 3 e qualquer operação pode ser processada em qualquer máquina. Considere que o gráfico de Gantt a seguir, representa uma solução factível para o problema.



Suponha agora que a operação O6 é candidata a ser trocada da máquina M1 para a máquina M3. Existem 3 posições onde pode-se tentar inserir a operação O6 na máquina M3. A primeira é entre as operações O4 e O1 e para essa, faz-se a seguinte verificação:

- k=O4 e x=O6. Como PJ[O4]=NI e SJ[O6]=NI, a condição $C_{NI} \leq C_{NI} - t_{NI}$ é falsa;
- x=O6 e k=O1. Como PJ[O6]=O5 e SJ[O1]=O2, a condição $C_{O2} \leq C_{O5} - t_{O5}$ é verdadeira e isso indica que se a operação O6 for inserida entre as operações O4 e O1 pode causar um ciclo e portanto isso nem será tentado. Neste caso causaria mesmo um ciclo com as operações O1-O2-O3-O5-O6-O1....

A segunda possibilidade é inserir a operação O6 entre as operações O1 e O8 e tem-se que:

- k=O4 e x=O6. Como PJ[O4]=NI e SJ[O6]=NI, a condição $C_{NI} \leq C_{NI} - t_{NI}$ é falsa;

- k=O1 e x=O6. Como PJ[O1]=NI e SJ[O6]=NI, a condição $C_{NI} \leq C_{NI} - t_{NI}$ é falsa;
- x=O6 e k=O8. Como PJ[O6]=O5 e SJ[O8]=O9, a condição $C_{O9} \leq C_{O5} - t_{O5}$ é verdadeira e isso indica que essa inserção pode causar um ciclo. Neste caso, não causaria um ciclo mas como uma das condições foi satisfeita a troca para esta posição não é tentada. Note que a inserção de O6 antes de O8, se fosse feita, causaria um deslocamento das operações O8 e O9 que iniciariam no instante 13 e 16, respectivamente.

A última tentativa é colocar a operação O6 depois da operação O8 e então:

- k=O4 e x=O6. Como PJ[O4]=NI e SJ[O6]=NI, a condição $C_{NI} \leq C_{NI} - t_{NI}$ é falsa;
- k=O1 e x=O6. Como PJ[O1]=NI e SJ[O6]=NI, a condição $C_{NI} \leq C_{NI} - t_{NI}$ é falsa;
- k=O8 e x=O6. Como PJ[O8]=O7 e SJ[O6]=NI, a condição $C_{NI} \leq C_{O7} - t_{O7}$ é falsa.

Como as condições para se ter um ciclo para esta última tentativa são falsas, então esta inserção pode ser feita.

Portanto, das 3 possibilidades existentes para inserir a operação O6 na máquina M3 só uma é considerada, pois é a única que satisfaz as condições de factibilidade.

Pode-se então resumir a heurística desenvolvida da seguinte maneira:

Passo 1: Construa a solução inicial usando alguma regra de despacho. Faça j=1, iter=0, inicialize o atraso incumbente e o atraso atual usando o atraso inicial.

Passo 2: a) Calcule os caminhos críticos das tarefas e o atraso atual. Atualize o atraso incumbente.

b) Se nenhuma tarefa estiver atrasada ou se o número máximo de iterações for satisfeito, PARE.

Passo 3: Se o critério para troca de operações entre máquinas for verdadeiro, então:

a) Avalie para determinadas operações todas as trocas possíveis para suas máquinas alternativas e escolha através da regra da melhor (ver abaixo).

b) Execute a troca escolhida e calcule os caminhos críticos das tarefas, o atraso e atualize o atraso incumbente.

c) Se não existir troca possível, vá ao Passo 4.

Passo 4: a) Se j>J, faça j=1;

b) Se a tarefa j estiver atrasada, vá ao Passo 5.

Senão, j=j+1 e volte ao Passo 4a).

Passo 5: a) Avalie todas as inversões possíveis no caminho crítico da tarefa j e escolha de acordo com a regra da melhor.

b) Execute a inversão escolhida. Faça iter=iter+1, j=j+1 e volte ao Passo 2.

c) Se não existir inversão possível, faça j=j+1 e volte ao Passo 3.

A principal diferença desta heurística em relação à do problema tradicional é a inclusão do Passo 3, que é o passo responsável por fazer a troca de operações entre máquinas, ou seja, por apresentar uma nova solução para o subproblema de roteamento. Neste passo, o critério para acionar a troca de máquinas é um dos seguintes: um certo número de iterações sem que haja atualização da solução incumbente ou um número fixo de iterações. Note que a cada vez só é trocada uma operação de máquina. As operações candidatas para serem avaliadas também seguem uma regra que é apresentada detalhadamente no capítulo 4. A cada vez que se executa a troca de uma operação de máquina um novo roteamento é obtido e a partir daí continua-se a fazer inversões das operações para se tentar melhorar a programação para este novo roteiro.

A regra da escolha da melhor troca é a mesma apresentada para a heurística do problema tradicional, porém a avaliação no Passo 3 vai também levar em consideração o carregamento das máquinas, ou seja, a função objetivo avaliada para escolher a operação que vai trocar de máquina é uma ponderação do atraso total das tarefas e do carregamento total das máquinas, isto é:

$$(\alpha \times \text{atraso total} + (1-\alpha) \times \text{carregamento total}) \text{ com } \alpha \in (0,1).$$

Em Brandimarte e Calderini (1995) é discutido o fato de se utilizar uma combinação convexa de objetivos. Sabe-se que para problemas contínuos pode-se calcular os pontos eficientes dessa combinação e escolher um deles como solução (Goicoechea et al., 1982). Entretanto, para o caso discreto, os pontos eficientes podem não ser gerados pela combinação convexa. Portanto, a ponderação adotada não tem um enfoque multi-objetivo e é utilizada para refletir a importância dos dois objetivos que estão sendo considerados.

3.3.3. Estratégias de intensificação e diversificação

Aqui são apresentadas as estratégias de intensificação e diversificação incorporadas à memória de curto prazo da BT para o problema flexível, que são basicamente as mesmas já descritas para o problema tradicional.

Para intensificação, as soluções de elite vão sendo armazenadas durante o processo de busca e pode-se usá-las para reiniciar a busca a partir de cada uma delas ou como uma forma de memória baseada em freqüência. Como neste problema as operações podem ser processadas por várias máquinas, a informação referente à máquina é muito importante. Portanto, para se usar memória baseada em freqüência armazena-se em uma matriz de residência o número de vezes que uma operação ocupa uma determinada posição em uma determinada máquina. Essa matriz é usada para tentar obter outras soluções que possuam as características mais freqüentes das soluções de elite.

Quanto à diversificação, também usa-se memória baseada em freqüência onde características de todas as soluções encontradas durante a busca são armazenadas em uma matriz, que guarda o número de vezes que uma operação ocupa uma posição numa máquina. A partir desta matriz pode-se calcular a freqüência relativa de cada um de seus elementos. A freqüência é então usada para tentar obter soluções onde as características de maior freqüência não sejam encontradas.

Os testes elaborados, parâmetros utilizados e detalhes importantes de implementação da heurística e das estratégias são apresentados no capítulo 5, juntamente com os resultados.

CAPÍTULO 4

TESTES E RESULTADOS PARA JOB SHOP TRADICIONAL

4.1. Introdução

Neste capítulo são descritas as implementações e os testes realizados para avaliar o desempenho da heurística desenvolvida para o problema de *job shop* tradicional. O conjunto de problemas usado para estes testes se encontra explicado na próxima seção.

A implementação da heurística foi feita em linguagem C e os testes realizados em estações de trabalho SUN, cujos modelos são descritos na apresentação dos resultados e dos tempos computacionais obtidos.

Os resultados preliminares dizem respeito aos testes exaustivos de ajuste dos elementos da BT com memória de curto prazo. A incorporação de memória de longo prazo da BT faz parte de outra fase de resultados onde, partindo da melhor configuração dos elementos, testou-se várias estratégias aplicando diversificação e intensificação, separadamente e em conjunto. Finalmente, a heurística é comparada com outros métodos heurísticos, e para um grupo de problemas de dimensão pequena, com a solução ótima. Uma versão resumida destes resultados é apresentada em Scrich e Armentano (1995).

4.2. Geração das instâncias de teste

As instâncias usadas neste trabalho para os testes computacionais foram geradas aleatoriamente e são baseadas em outras gerações encontradas na literatura (Adams et al., 1988; Chang et al., 1989; Hutchison et al., 1991;Van Laarhoven et al., 1992; Aarts et al., 1994; Taillard, 1994).

Cada instância pode ser caracterizada pelos seguintes dados de entrada: número de tarefas (J), número de máquinas (M), número de operações por tarefa (OPJ), tempo de processamento das operações (t_{ij}), máquina que pode processar as operações (m_{ij}), data de entrega das tarefas (d_j) e fator de atraso das tarefas (β).

A quantidade de tarefas e de máquinas foram pré-definidas, enquanto que os outros dados ou foram gerados aleatoriamente de acordo com uma distribuição uniforme U[a,b], ou tiveram que ser calculados. A seguir, são mostrados os dados utilizados para o conjunto C1 de problemas-testes.

Tabela 4.1. - Dados de geração dos problemas-testes

Dados de entrada	Conjunto C1
J	5, 10, 15, 30, 50, 100
M	5, 10, 15
OPJ	M
m_{ij}	U[1,M]
t_{ij}	U[1,99]
d_j	$\beta \times \sum_i t_{ij}$
β	$\beta_1 = ((J \times M)/1000) + 0,5$
	$\beta_2 = \beta_1 + 1$

De acordo com a tabela 4.1, o parâmetro β_1 utilizado para o cálculo das datas de entrega foi obtido de He et al. (1993) e β_2 é utilizado para gerar datas de entrega mais folgadas.

As dimensões utilizadas nos testes, ou seja, as combinações usadas de J×M são 5×5, 10×5, 50×5, 10×10, 15×10, 50×10, 100×10, 15×15, 30×15, 50×15, sendo que para cada dimensão são geradas 5 instâncias e para cada uma tem-se duas formas de calcular as datas de entrega. Tem-se então que o conjunto C1 é constituído de 100 instâncias ao todo, onde as 50

primeiras utilizam β_1 e as restantes β_2. Como os resultados, em geral, aparecem em relação à média das 5 instâncias de cada dimensão para um dado valor de β então, para facilitar a notação, cada dimensão é denominada de Grupo e portanto, tem-se 20 grupos onde Gp1, Gp2, Gp3, Gp4, Gp5, Gp6, Gp7, Gp8, Gp9, Gp10 constituem, respectivamente, o conjunto de instâncias de cada dimensão onde utilizou-se β_1 e Gp11, Gp12, Gp13, Gp14, Gp15, Gp16, Gp17, Gp18, Gp19, Gp20 onde utilizou-se β_2.

4.3. Apresentação dos resultados

Os resultados a seguir estão divididos em três etapas. A primeira onde estão descritos os testes feitos para ajuste dos parâmetros básicos da BT, a segunda onde descreve-se as várias estratégias de memória de longo prazo que foram incorporadas à heurística e a terceira diz respeito à comparação da heurística com outros métodos heurísticos existentes para o problema considerando atraso das tarefas. Todos os testes foram feitos sobre um subconjunto de C1 formado pelas instâncias dos grupos Gp1, Gp3, Gp4, Gp6, Gp7, Gp9, Gp11, Gp13, Gp14, Gp16, Gp17, Gp19.

4.3.1. Testes preliminares

Aqui são apresentados os diversos testes efetuados para ajuste dos componentes da heurística, relacionados à memória de curto prazo da BT. Os elementos testados são apresentados a seguir e todos os testes utilizaram 250 iterações da heurística.

- atributo do movimento e restrição tabu - a escolha do atributo e da restrição tabu devem levar em conta se está se querendo uma busca mais ou menos restritiva. Por exemplo, considere o movimento de inversão de um arco disjuntivo (x,y). Cada um dos exemplos a seguir descreve um atributo e uma regra de proibição diferente associado a este movimento, e que foram testados durante este trabalho:

Tabela 4.2. - Exemplos de atributo e restrição tabu

	Atributo	Restrição tabu
(1)	x	proíbe inversão que contenha o nó x
(2)	y	proíbe inversão que contenha o nó y
(3)	x,y	proíbe inversão que resulte no arco (x, y)
(4)	x,y	proíbe inversão que contenha o nó x ou y

Note que os exemplos (1), (2) e (4) podem restringir um número maior de movimentos que o (3). Nos testes, o desempenho obtido pelas 4 versões foi praticamente equivalente e aqui, por causa das características do problema de *job shop*, optou-se pela condição que parece restringir menos a busca, que é a (3).

- duração da restrição tabu - isto está relacionado ao número de iterações que um atributo fica presente na lista tabu. Vale ressaltar que, em termos computacionais, a lista tabu é representada por uma matriz, onde o elemento da matriz representa o arco proibido e guarda o número de iterações que o movimento fica tabu. A matriz é inicializada com zero em todos os seus elementos. Para a duração tabu, vários tipos de implementação e valores foram testados. Em geral, foram baseados nos trabalhos de *job shop* para *makespan* (Dell'Amico e Trubian, 1993; Taillard, 1994; Barnes e Chambers, 1995), e são descritos a seguir:

- duração tabu fixa em um número qualquer, que pode ou não ser dependente de algum dado do problema;

- duração aleatória sorteada entre $[t_{min},t_{max}]$ para cada movimento, onde t_{min} e t_{max} podem ser valores fixos ou não. Quando t_{min} e t_{max} são valores fixos, a duração é chamada de duração tabu aleatória, e quando t_{min} e t_{max} variam é chamada de duração tabu dinâmica.

Neste trabalho, testou-se os três tipos de duração tabu e os valores usados são mostrados abaixo:

1. duração tabu fixa em 7 e N/2, onde N é o número total de operações (N=J×M);
2. duração tabu aleatória, com os seguintes valores:

- $t_{min} = 20$ e $t_{max} = 30$;
- $t_{min} = 5$ e $t_{max} = 15$;
- $t_{min} = 0{,}8 \times L$ e $t_{max} = 1{,}2 \times L$, onde $L=((J+M)/2).e^{-J/5.M} + N/2.e^{-5.M/J}$ (Taillard, 1994);
- $t_{min} \in U[a,b]$, onde $a = 2$ e $b = a+\lfloor (J+M)/3 \rfloor$ e $t_{max} \in U[A,B]$, onde $A = t_{min}+6$ e $B = A+\lfloor (J+M)/3 \rfloor$ (Dell'Amico e Trubian, 1993), onde $\lfloor x \rfloor$ é o maior inteiro menor ou igual a x.

3. duração tabu dinâmica:

Para $t_{min} \in U[a,b]$, onde $a=2$ e $b=a+\lfloor (J+M)/3 \rfloor$ e $t_{max} \in U[A,B]$, onde $A=t_{min}+6$ e $B=A+\lfloor (J+M)/ /3 \rfloor$ utilizou-se duas estratégias:

- sorteia-se um valor para cada movimento entre t_{min} e t_{max} e a cada 50 iterações sorteia-se um novo valor para t_{min} e t_{max}.
- gera-se um valor entre t_{min} e t_{max}. Se a solução atual é melhor que a solução incumbente, então a duração tabu fica sendo 1; se a solução atual for melhor que a anterior, então decresce em uma unidade o valor gerado (se este for maior que o limite mínimo de geração); se a solução atual for pior ou igual a anterior, então aumenta em uma unidade o valor gerado (se este for menor que o limite máximo de geração). A cada 20 iterações gera-se novamente os limites t_{min} e t_{max} (Dell'Amico e Trubian, 1993).

Para os outros intervalos de t_{min} e t_{max}, que foram testados para duração aleatória acima, utilizou-se as estratégias descritas a seguir:

- calcula-se no início da busca a faixa de geração $t_{max}-t_{min}$; se a solução for melhor que a anterior, então aumenta-se t_{min} e t_{max} de 5% do valor da faixa; se a solução for pior que a anterior, então diminui-se t_{min} e t_{max} de 5% do valor da faixa. A cada iteração gera-se aleatoriamente um valor entre t_{min} e t_{max}.
- inversa a anterior, isto é, aumenta-se a faixa se a solução é pior e caso contrário, diminui-se.

Depois de todos esses testes, notou-se que a qualidade das soluções parece ser insensível à duração tabu e estratégia utilizadas. A escolha então baseou-se no critério de

prevenção de ciclagem de soluções, e quem obteve mais sucesso foi a duração tabu aleatória gerada no intervalo usado por Dell'Amico e Trubian (1993).

- vizinhança - no capítulo 3, a vizinhança usada na heurística (Vh), é determinada da seguinte maneira: escolhe-se uma tarefa atrasada e cada inversão possível no caminho crítico desta tarefa constitui uma solução vizinha. Pelos passos da heurística a tarefa atrasada escolhida segue uma ordem, sempre começando da tarefa 1 e indo até a última tarefa. A primeira modificação testada foi em relação à escolha da tarefa, onde uma das alternativa testada foi escolher aleatoriamente uma tarefa atrasada e a outra foi escolher sempre a tarefa mais atrasada. Estas alternativas não apresentaram melhoria de qualidade. A segunda modificação foi desenvolver outras vizinhanças baseadas em Vh. A primeira vizinhaça estendida, Ve1, considera que deve-se avaliar todas as inversões possíveis no caminho crítico de cada uma das tarefas atrasadas, para se escolher aquela que será executada. Embora esta vizinhança seja maior do que Vh, ela não mostrou melhoria em termos de qualidade das soluções, além de apresentar um tempo computacional extremamente alto. Por estes motivos esta vizinhança foi descartada. Outra vizinhança testada (Ve2), é a apresentada por Dell'Amico e Trubian (1993). Nesta vizinhança, considera-se a possibilidade de inverter simultaneamente mais de um arco disjuntivo no caminho crítico de uma dada tarefa, ou seja, dado um arco disjuntivo (x,y) no caminho crítico de uma tarefa, considera-se todas as possíveis permutações de {PM[x],x,y} e {x,y,SM[y]} nas quais a operação y precede x na máquina. Como esta vizinhança não garante a geração de soluções factíveis, as condições suficientes de factibilidade, explicadas no capítulo 3, foram usadas para descartar as soluções infactíveis. Dell'Amico e Trubian não usam essas condições quando trabalham com esta vizinhança, pois, é possível calcular um limitante inferior do *makespan* resultante das inversões e pode-se mostrar que uma inversão infactível vai sempre ter um limitante maior que uma inversão factível, e como sempre se escolhe a inversão que resulte no menor limitante, é claro que as inversões que causam infactibilidade são automaticamente descartadas. Para o caso de atraso total das tarefas, não é possível calcular este limitante inferior e por isso optou-se por usar as condições de factibilidade. Os testes com esta vizinhança também foram desanimadores, pois não se obteve melhoria significativa na qualidade das soluções e o tempo computacional foi bem maior. Portanto, esta vizinhança também foi descartada. Foi

escolhida então a vizinhança Vh e alguns testes foram feitos para avaliar o tamanho desta vizinhança. A conclusão é que cada solução tem em torno de J/2 vizinhos.

Vale ressaltar que aqui foram feitos testes com até 500 iterações para cada uma das vizinhanças Vh, Ve1 e Ve2, a fim de comprovar se o número de iterações era um fator de bloqueio para se obter soluções melhores. Notou-se que o tempo computacional foi bem maior e o ganho desprezível em relação ao obtido com 250 iterações. Este número fixo 250 foi estipulado por ter se mostrado bom para todos os grupos e também para poder tornar mais fiel a comparação da heurística com memória de curto prazo com a heurística onde há a incorporação da memória de longo prazo.

- critério de aspiração: para este elemento foram feitos dois testes, onde em um usou-se o critério mais comum nos trabalhos, que libera um movimento de sua condição tabu, se o valor da solução resultante de sua execução for melhor que o valor da solução incumbente, e no outro, um movimento é liberado se o valor da solução resultante de sua execução for, no máximo, 10% maior que o valor da solução incumbente. Pelos resultados obtidos, o primeiro critério se mostrou melhor em relação à qualidade das soluções obtidas e também em relação à prevenção de ciclagem das soluções.

- primeiro e melhor movimento - este parâmetro está relacionado à escolha do movimento. O movimento a ser executado pode ser o melhor movimento da vizinhança, isto é, aquele que resulte no melhor valor da função objetivo dentre todos ou o primeiro movimento que obtenha um valor de função objetivo melhor que o valor atual (Taillard, 1990). No trabalho, depois de alguns testes, optou-se por escolher sempre o melhor movimento da vizinhança.

Na tabela a seguir são apresentados os elementos de curto prazo da BT que são utilizados na heurística.

Tabela 4.3. - Elementos usados para memória de curto prazo da BT

Parâmetro	valor
atributo e restrição tabu	x,y → proíbe inversão que resulte no arco (x,y)
duração tabu aleatória	sorteia aleatoriamente um valor entre $[t_{min}, t_{max}]$ para cada movimento, com $t_{min} \in U[a,b]$, a=2 e $b=a+\lfloor(J+M)/3\rfloor$ e $t_{max} \in U[A,B]$, $A=t_{min}+6$ e $B=A+\lfloor(J+M)/3\rfloor$;
vizinhança	uma solução vizinha é obtida pela inversão de um arco disjuntivo no caminho crítico de uma dada tarefa (Vh)
critério de aspiração	libera um movimento tabu se a solução resultante da sua execução for melhor que a solução incumbente
melhor movimento	movimento que resulta no melhor valor da função objetivo dentre todos os da vizinhança é executado
número de iterações	250

A heurística com esses elementos da BT de curto prazo é denotada por **HCP**, e denota-se por HCP1, HCP2, HCP3 e HCP4 a heurística que tem como solução inicial a solução obtida pela regra MDD, MOD, CR+SPT e S/RPT + SPT, respectivamente. A tabela a seguir apresenta os resultados das soluções iniciais e das heurísticas para o conjunto de instâncias C1.

Tabela 4.4. - Valores de atraso obtidos pela heurística HCP e pelas regras de despacho

Gp	SI1	SI2	SI3	SI4	HCP1	HCP2	HCP3	HCP4	T(s)
1	1.056,6	**1.037,4**	**1.037,4**	**1.037,4**	**900,8**	921,6	921,6	921,6	1,3
2	**3.641,0**	3.646,0	3.646,0	3.646,0	**3.396,6**	3.406,4	3.406,4	3.406,4	5,0
3	**61.618,4**	71.415,8	71.415,8	71.415,8	**60.977,0**	70.645,8	70.645,8	70.645,8	98,6
4	4.857,6	**4.594,0**	**4.594,0**	**4.594,0**	4.273,4	**4.190,8**	**4.190,8**	**4.190,8**	10,2
5	9.215,4	**8.855,0**	**8.855,0**	**8.855,0**	8.425,0	**8.320,8**	**8.320,8**	**8.320,8**	30,9
6	**66.954,6**	73.859,6	73.859,6	73.859,6	**65.640,6**	72.866,0	72.866,0	72.866,0	255,2
7	**229.558,8**	273.764,0	272.649,4	273.425,2	**227.739,2**	272.118,6	271.016,4	271.708,4	977,8
8	9.339,6	**9.011,4**	**9.011,4**	**9.011,4**	8.680,8	**8.460,2**	**8.460,2**	**8.460,2**	46,5
9	**27.112,4**	28.448,4	28.448,4	28.448,4	**26.201,8**	27.683,4	27.683,4	27.683,4	157,8
10	**60.902,4**	66.391,6	65.101,4	66.454,6	**59.557,0**	65.158,6	63.997,6	65.271,2	494,9
11	99,0	104,4	**76,8**	91,4	3,2	**2,8**	**2,8**	**2,8**	0,5
12	**1.100,8**	1.191,4	1.156,4	1.225,8	**887,2**	934,4	960,6	964,4	6,3
13	**48.626,6**	58.699,2	58.921,2	58.777,6	**48.122,2**	57.937,8	57.946,2	58.012,6	115,8
14	407,4	**218,0**	268,8	261,4	13,0	**8,0**	19,4	20,2	10,1
15	2.180,6	1.659,4	**1.542,2**	1.616,0	1.286,6	1.267,2	**1.079,6**	1.094,0	31,5
16	**42.560,4**	48.659,6	49.446,2	48.994,4	**41.417,6**	47.260,8	48.243,2	48.078,2	297,7
17	**179.572,0**	222.411,6	222.234,2	223.764,6	**177.981,2**	220.406,2	220.544,6	222.050,8	981,4
18	454,0	173,0	**121,6**	247,4	42,2	**0,0**	9,0	26,2	31,4
19	7.218,4	6.776,2	**6.627,8**	6.937,6	6.175,0	5.981,4	**5.612,8**	6.353,0	197,1
20	**26.638,6**	28.937,6	29.660,0	27.418,2	**25.565,2**	27.948,4	28.449,6	26.593,4	573,0

Notação:

Gp - Grupo de problemas.

SIi - Valor da solução inicial obtida pela regra i, onde:

i=1 Regra MDD;

i=2 Regra MOD (com as datas de entrega das operações definidas no capítulo 3);

i=3 Regra CR+SPT;

i=4 Regra S/RPT+SPT;

HCPi - Valor da solução da heurística partindo da solução inicial obtida pela regra i.

T - Tempo de execução de 250 iterações da melhor heurística (valores em negrito). Este tempo foi obtido da execução da heurística em estações SUN SPARCclassic com clock de 50MHz, 48MB de RAM, sistema operacional SUNOS 4.1.3.

Em relação às regras pode-se notar, pelos valores em negrito da tabela 4.4, que a MDD é a que fornece os melhores resultados, principalmente para problemas com atraso grande. Quando comparamos as soluções encontradas pela heurística com as respectivas soluções iniciais, as melhorias obtidas variam de 1 a 100%, onde as maiores melhorias ocorrem para os problemas com menor atraso, como mostra a tabela 4.5. Nota-se que quando aumentamos o número de tarefas e fixamos o número de máquinas há uma redução na melhoria e um aumento no tempo computacional e quando fixamos o número de tarefas e aumentamos o número de máquinas há um aumento na melhoria e também no tempo computacional. **M-HCPi** é definida como a melhoria em porcentagem da solução da heurística em relação à solução inicial dada pela regra i, isto é, $\left(\frac{\text{SIi} - \text{HCPi}}{\text{SIi}}\right) \times 100$. As instâncias onde a heurística obtém solução com atraso zero, e portanto onde a melhoria é de 100%, não são incluídas no cálculo da média da porcentagem, e tais instâncias pertencem aos grupos marcados com *.

Tabela 4.5. - Valores de melhoria obtidas pela heurística HCP

Gp	J×M	M-HCP1 (%)	M-HCP2 (%)	M-HCP3 (%)	M-HCP4 (%)
1	5×5	13,33	10,67	10,67	10,67
2	10×5	6,74	6,56	6,56	6,56
3	50×5	1,05	1,09	1,09	1,09
4	10×10	11,94	8,70	8,70	8,70
5	15×10	8,34	5,96	5,96	5,96
6	50×10	1,97	1,34	1,34	1,34
7	100×10	0,79	0,60	0,60	0,63
8	15×15	7,09	6,06	6,06	6,06
9	30×15	3,35	2,67	2,67	2,67
10	50×15	2,20	1,86	1,69	1,78
11	5×5	84,47 *	92,00 *	87,83 *	87,83 *
12	10×5	19,60	21,41	16,19	20,24
13	50×5	1,04	1,31	1,66	1,32
14	10×10	95,14 *	90,42 *	75,99 *	86,26 *
15	15×10	40,74	23,51	30,85	32,70
16	50×10	2,71	2,86	2,45	1,88
17	100×10	0,88	0,91	0,76	0,76
18	15×15	83,20 *	----- *	82,28 *	85,55 *
19	30×15	14,69	11,79	15,18	8,42
20	50×15	4,04	3,41	4,08	2,98

Pode-se notar pela Tabela 4.5 que, em geral, a heurística consegue melhorar a solução inicial na mesma proporção, ou seja, a escolha de uma boa solução inicial parece ser importante. Isso condiz com a afirmação feita por Taillard (1994) que, quando se trabalha com memória de curto prazo, é importante que a solução inicial seja de boa qualidade para que a busca seja eficiente. Por este motivo optou-se aqui por trabalhar com a regra MDD e a incorporação de diversificação e intensificação só foi testada para esta regra. Em relação ao fator de atraso β, pode-se notar que as mesmas dimensões de instâncias dos grupos Gp1 a Gp10 tem valores de atraso maiores que aquelas de Gp11 a Gp20. Porém, tanto para β_1 como para β_2 existem problemas com pouco e com bastante atraso.

A complexidade computacional de HCP é da ordem de $O(J^3 \times M^2)$ e os cálculos se encontram no apêndice A.

4.3.2. Testes com diversificação e intensificação

Para tentar melhorar ainda mais os resultados obtidos pela heurística, foram implementadas as estratégias de memória de longo prazo, descritas no capítulo 3. Essas estratégias são incorporadas à heurística com memória de curto prazo. A maioria das implementações das estratégias de intensificação e diversificação feitas aqui se baseia em informações da freqüência de características das soluções e, basicamente, quando se tem tais informações, são usadas penalizações para desestimular características que apareçam com freqüência, e incentivos para favorecer regiões promissoras. Essas penalizações e incentivos agem alterando a avaliação da escolha dos movimentos ou alterando o valor da prioridade da regra de despacho para que se calcule uma nova solução de partida da busca, chamada de estratégia de reinício. As idéias usadas aqui se baseiam nos trabalhos de Dell'Amico e Trubian (1993), Chakrapani e Skorin-Kapov (1993), Kelly et al. (1994), Taillard (1994), Barnes e Chambers (1995), Nowicki e Smutnicki (1996). Cada um dos testes executados utilizou como critério de parada 250 iterações. A seguir é mostrada, de forma geral, como estas estratégias são incorporadas à heurística.

- Obtenção da solução inicial.
- Busca de melhores soluções. Ao final do passo 2 do algoritmo da heurística apresentado no capítulo 3, testar:
    1. Se critério para iniciar diversificação for verdadeiro, então diversifique.
    2. Se critério para iniciar intensificação for verdadeiro, então intensifique.

4.3.2.1. Intensificação

Para intensificação foram testadas duas estratégias, onde em uma delas usou-se informações da solução incumbente e na outra, informações das soluções de elite que vão sendo armazenadas durante a busca. Em geral, a idéia da intensificação é interromper a busca em um certo momento e recomeçá-la de outro ponto.

1) utilização da solução incumbente para reiniciar a busca. Os instantes escolhidos para este reinício foram:

- cada vez que se obtém uma nova solução incumbente, reinicia-se a busca e esvazia-se a lista tabu.
- depois de k iterações sem que a busca consiga atualizar a solução incumbente, reinicia-se a busca a partir da solução incumbente e esvazia-se a lista tabu. Os valores de k utilizados foram 15, 30, 50 e $\sqrt{N}$. Vale ressaltar que se nas próximas k iterações depois do reinício a solução incumbente não for atualizada, não é feito um novo reinício para evitar uma possível ciclagem. No entanto, a busca continua normalmente com a memória de curto prazo.

2) utilização das soluções de elite. Durante a execução da heurística c soluções de elite vão sendo armazenadas, e podem ser usadas das seguintes maneiras:

a) para se fazer um reinício da busca a partir de cada uma ou de algumas das soluções guardadas. Esta estratégia deve ser iniciada depois de várias iterações da busca, para que as soluções armazenadas sejam realmente de boa qualidade. Foi testado, depois de 200 iterações da heurística somente usando memória de curto prazo, reiniciar a busca de uma solução de elite a cada 10 iterações, ou seja, a busca reinicia 5 vezes de 5 soluções de elite diferentes. A cada reinício, a lista tabu é esvaziada.

b) como memória baseada em freqüência. A partir das soluções de elite, pode-se calcular uma matriz de freqüência fE, que guarda o número de vezes que uma operação ocupa uma determinada posição na máquina em que deve ser processada, de acordo com as soluções. Por exemplo, considere um problema de 2 máquinas, 2 tarefas e 2 operações por tarefa (1 e 2 operações da tarefa 1 e 3 e 4 operações da tarefa 2) e as seguintes soluções de elite:

solução 1	solução 2	solução 3
op 1 - M1 - posição 1	op 1 - M1 - posição 1	op 1 - M1 - posição 2
op 2 - M2 - posição 2	op 2 - M2 - posição 1	op 2 - M2 - posição 2
op 3 - M2 - posição 1	op 3 - M2 - posição 2	op 3 - M2 - posição 1
op 4 - M1 - posição 2	op 4 - M1 - posição 2	op 4 - M1 - posição 1

Destas soluções obtém-se a seguinte matriz de freqüência fE:

posição / operação	1	2	3	4
1	2	1	2	1
2	1	2	1	2

A partir da matriz fE pode-se reiniciar a busca, incentivando as operações que mais aparecem em determinadas posições. Esse incentivo pode ser usado tanto para se tentar obter uma nova solução de partida (reinício), como na escolha do melhor movimento.

O uso do incentivo no procedimento de obtenção da solução inicial consiste na alteração do cálculo da prioridade das operações para

valor_prioridade=valor_prioridade+I*fE_op(r)_pos(p),

onde I indica o fator de incentivo e fE_op(r)_pos(p) denota a freqüência com que a operação r ocupa nas soluções de elite a posição p da máquina. Como exemplo, suponha que no procedimento de obtenção da solução inicial está sendo alocada uma operação para a posição 2 da máquina 1. Para cada operação r que está disponível na fila da máquina 1 o valor da prioridade calculado é valor_prioridade=valor_prioridade+I*fE_op(r)_pos(2).

Para a escolha do melhor movimento, o incentivo é incorporado ao cálculo do valor do movimento, da seguinte forma:

valor_movimento=valor_movimento−I*(fE_op(x)_pos(p1)+fE_op(y)_pos(p2)), onde p1 e p2 são as posições que as operações x e y vão ocupar depois da inversão do arco (x,y). Por exemplo, suponha que esteja sendo avaliado o movimento de inversão do arco disjuntivo (1,4), onde a operação 1 ocupa a primeira posição na máquina e a operação 4 ocupa a segunda posição na máquina. Ao se considerar a inversão o valor do movimento será valor_movimento=valor_movimento−I*(fE_op(1)_pos(2)+fE_op(4)_pos(1)).

Para esta implementação de memória baseada em freqüência, os instantes de aplicação da intensificação podem ser:

- depois de um número fixo k de iterações, e testou-se para k os valores 30, 50 e 200.
- depois de q iterações sem que a busca consiga atualizar a solução incumbente, com

q testado em 15, 30 e 50.

O uso do incentivo sobre a escolha dos movimentos deve durar um número fixo de iterações, para o qual foram testados 10 e 20 iterações, ou até que a matriz das soluções de elite se altere.

Para o número de soluções de elite c, testou-se os valores 5 e 20, sendo que 5 foi quem obteve melhores resultados. Isso pode ser explicado, pois o conjunto de 20 soluções contém soluções de qualidade inferior. Para o fator de incentivo I foram testados os valores 10 e 100, e I=10 obteve melhores resultados.

Durante os testes com estas estratégias, pôde-se constatar que o tempo computacional se manteve praticamente o mesmo. As estratégias que tiveram melhor desempenho foram aquelas onde utilizou-se memória baseada em freqüência mas, na média, nenhuma delas conseguiu superar os ganhos obtidos por HCP. Por este motivo não são apresentados resultados da aplicação da intensificação sozinha.

4.3.2.2. Diversificação

A idéia principal da diversificação é evitar regiões muito pesquisadas através da penalização de características muito freqüentes das soluções. Para as estratégias de diversificação também foi usada memória baseada em freqüência mas, neste caso, de todas as soluções obtidas durante a busca. Para isso armazena-se em uma matriz A o número de vezes que uma operação ocupa uma posição na máquina em que deve ser processada. Por exemplo, seja um problema com 3 máquinas e 2 tarefas e cada tarefa com 3 operações (1, 2, 3 são operações da tarefa 1; 4, 5, 6 são operações da tarefa 2) com a seguinte matriz A associada, até a iteração corrente:

posição / operação	1	2	3	4	5	6
1	19	12	7	8	1	13
2	1	8	13	12	19	7

A partir da iteração de início da diversificação e a cada iteração subsequente, calcula-

se uma matriz de freqüência fA onde cada elemento é a razão entre o elemento equivalente de A e o maior elemento de A. Para o exemplo tem-se a seguinte matriz de freqüência fA, da iteração corrente:

posição / operação	1	2	3	4	5	6
1	19/19	12/19	7/19	8/19	1/19	13/19
2	1/19	8/19	13/19	12/19	19/19	7/19

Tal matriz é utilizada para penalizar movimentos através do uso de um fator de penalidade P. Essa penalização pode ser aplicada no procedimento de cálculo da solução inicial, para se obter uma nova solução de partida, e na escolha do movimento, tentando sempre desfavorecer características bastante encontradas durante a busca, até aquele momento.

No procedimento da solução inicial, modifica-se o cálculo da prioridade da operação da seguinte maneira:

valor_prioridade=valor_prioridade–P*fA_op(o)_pos(p);

e para o cálculo da escolha do melhor movimento na fase de busca de soluções melhores tem-se:

valor_movimento=valor_movimento+P*(fA_op(x)_pos(p1)+fA_op(y)_pos(p2)).

Os instantes de aplicação da diversificação devem seguir algum critério, e aqui testou-se iniciá-la :

- a cada número fixo k de iterações. Para k usou-se 20, 50 e 200.
- depois de um certo número q de iterações sem que a solução incumbente seja atualizada. Para q testou-se 15, 30, 50 e $\sqrt{N}$.
- depois de 200 iterações, e repetindo o reinício a cada 10 iterações.

A aplicação da diversificação deve durar um certo tempo, e para essa duração testou-se:

- um número fixo de iterações, com valores em 15, 20 e 30.

- a cada k iterações alternadamente, aplica-se penalidade sobre os movimentos, com k testado em 30 e 50.
- até que a solução encontrada seja melhor que a sua predecessora.

Outro parâmetro testado foi o valor do fator P, para o qual utilizou-se 10, 100, 150, 200 e 10% do valor da solução inicial. E ainda mais importante foi a escolha da normalização da matriz de freqüência fA. Foram feitos testes normalizando a matriz fA da forma como mostrada anteriormente e também fazendo uma transformação linear nos elementos da matriz de freqüência fA para valores inteiros de 1 a 10. A combinação da segunda versão de fA com a penalidade P=10 apresentou os melhores resultados, e ilustrando, através do exemplo, temos que a matriz de freqüência fA, calculada de A, é:

posição / operação	1	2	3	4	5	6
1	10	6	4	4	1	7
2	1	4	7	6	10	4

Depois de testadas estas estratégias, concluiu-se que a melhor delas, em média, é reiniciar a busca a cada 20 iterações com uma nova solução de partida, obtida através de penalização e freqüência no procedimento da solução inicial, e aplicar a penalização na escolha dos movimentos de 50 em 50 iterações, ou seja, da iteração 0 a 49 não há penalização, da iteração 50 até 99 há penalização, da 100 a 149 não há penalização, e assim por diante. Os resultados para esta estratégia estão apresentados na Tabela 4.6, onde **HLPD** denota o valor da solução da heurística de longo prazo com essa melhor estratégia de diversificação, **M-HLPD** é a melhoria, em porcentagem, obtida em relação à solução inicial dada pela regra MDD e **T** é o tempo de execução de 250 iterações da heurística HLPD em estações SUN SPARCclassic.

Tabela 4.6. - Resultados da heurística com diversificação HLPD

Gp	(J x M)	HLPD	M-HLPD (%)	T(s)
1	5x5	920,2	11,69	1,4
2	10x5	3.424,8	5,91	4,9
3	50x5	59.499,0	3,45	106,6
4	10x10	4.120,0	15,08	11,9
5	15x10	8.300,4	9,87	30,8
6	50x10	64.005,6	4,40	273,0
7	100x10	224.461,6	2,20	1.095,9
8	15x15	8.588,6	7,94	49,4
9	30x15	25.997,4	4,11	190,2
10	50x15	58.528,6	3,89	524,5
11	5x5	3,2	84,47 *	0,6
12	10x5	904,4	17,93	6,0
13	50x5	47.588,6	2,10	134,5
14	10x10	30,4	89,33 *	15,4
15	15x10	1.511,2	29,56	33,8
16	50x10	40.139,4	5,71	237,9
17	100x10	175.925,4	2,02	1.088,9
18	15x15	47,2	85,41 *	49,8
19	30x15	5.745,8	20,68	213,2
20	50x15	24.448,2	8,23	609,4

Pode-se notar que para problemas com maior atraso conseguiu-se resultados bem melhores que os mostrados nas Tabelas 4.4 e 4.5. Por exemplo, para Gp7 obteve-se um acréscimo de melhoria de 0,79% para 2,20% com um tempo computacional equivalente. Já para problemas com atraso pequeno (<10.000) o comportamento não é uniforme, obtendo-se para alguns grupos resultados de melhor qualidade e para outros, de pior qualidade.

Os gráficos a seguir mostram, para algumas instâncias, uma comparação da evolução das soluções obtidas pela heurística com e sem a diversificação.

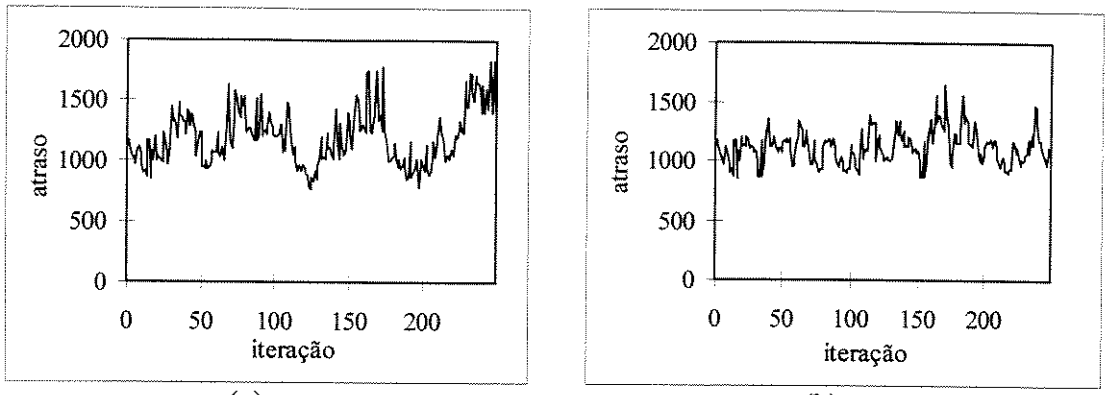


Gráfico 4.1. Heurística sem diversificação (a) e com diversificação (b) para instância de Gp1

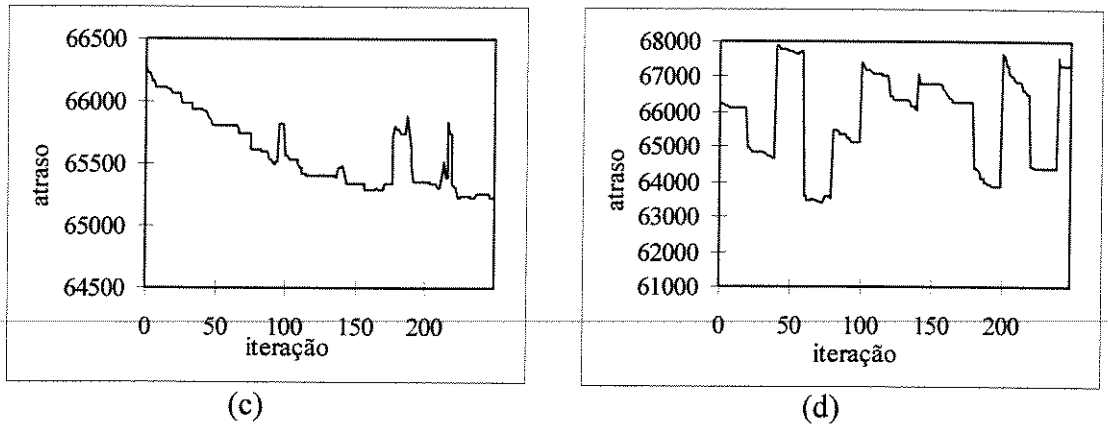


Gráfico 4.2. Heurística sem diversificação (c) e com diversificação (d) para instância de Gp6

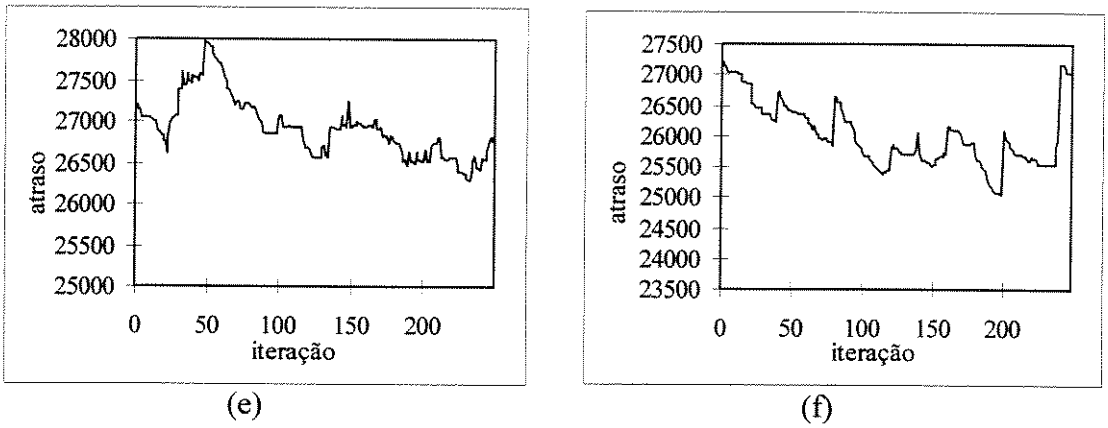
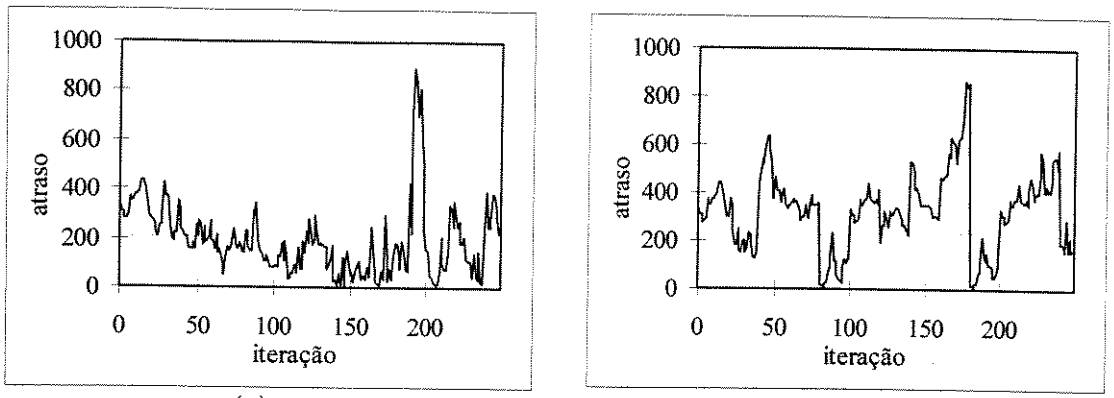


Gráfico 4.3. Heurística sem diversificação (e) e com diversificação (f) para instância de Gp9



(g) (h)

Gráfico 4.4. Heurística sem diversificação (g) e com diversificação (h) para instância de Gp14

Pelos gráficos percebe-se que, para as instâncias de Gp6 e Gp9 a busca alcança soluções que com a memória de curto prazo não alcançava. Para as instâncias de Gp1 e Gp14 não há muita diferença nos dois gráficos, o que leva a concluir que para esses problemas a memória de curto prazo parece ser suficiente.

4.3.2.3. Diversificação + Intensificação

Outras estratégias combinando as de diversificação com as de intensificação também foram implementadas. O esquema geral abaixo mostra com mais detalhes como se incorporou esses elementos de memória de longo prazo.

- Obtenção da solução inicial;
- Busca de melhores soluções. Ao final do passo 2 apresentado no algoritmo da heurística testar,

    1. Se critério para iniciar diversificação for verdadeiro, então reinicie a busca com uma solução de partida obtida com a aplicação da freqüência e penalidade sobre a regra no procedimento de obtenção da solução inicial e/ou use a freqüência e a penalidade para escolher o movimento, na fase de busca de melhores soluções, por um certo número de iterações.
    2. Se critério para iniciar intensificação for verdadeiro, então inicie a intensificação de acordo com a implementação em questão, isto é:
        - utilize a incumbente para reiniciar a busca ou

- recupere uma das soluções de elite para reiniciar a busca ou
- calcule uma nova solução de partida com a aplicação do incentivo e da freqüência no procedimento de obtenção da solução inicial e/ou use o incentivo e a freqüência para escolher o movimento, na fase de busca de melhores soluções, por um certo número de iterações.

Vale ressaltar que a intensificação e a diversificação, quando ambas estão sendo aplicadas, nunca ocorrem simultaneamente durante a busca e que, em geral, a intensificação é aplicada depois da diversificação.

A melhor estratégia de combinação da diversificação e intensificação foi aplicar a melhor diversificação (descrita na seção anterior) até a iteração 200 e a partir daí recomeçar a busca a cada 10 iterações de cada uma das soluções de elite, esvaziando a lista tabu e usando somente a memória de curto prazo. Os resultados a seguir mostram a pequena superioridade desta combinação, denotada por **HLPDI**, em relação à melhor diversificação HLPD, e com um tempo equivalente. O tempo **T** é obtido da execução de 250 iterações da heurística em estações SUN SPARCclassic.

Tabela 4.7. - Resultados de HLPD e HLPDI

Gp	(J x M)	M-HLPDI(%)	T(s) - HLPDI	M-HLPD(%)	T(s) - HLPD
1	5x5	12,08	1,4	11,69	1,4
2	10x5	6,64	5,0	5,91	4,9
3	50x5	3,40	107,9	3,45	106,6
4	10x10	15,68	11,6	15,08	11,9
5	15x10	10,58	25,9	9,87	30,8
6	50x10	4,57	240,4	4,40	273,0
7	100x10	2,28	938,1	2,20	1.095,9
8	15x15	8,88	44,1	7,94	49,4
9	30x15	4,42	155,9	4,11	190,2
10	50x15	4,02	426,0	3,89	524,5
11	5x5	86,41 *	0,4	84,47 *	0,6
12	10x5	19,75	5,6	17,93	6,0
13	50x5	1,99	109,3	2,10	134,5
14	10x10	92,41 *	11,0	89,33 *	15,4
15	15x10	32,77	32,2	29,56	33,8
16	50x10	5,58	258,4	5,71	237,9
17	100x10	2,03	995,6	2,02	1.088,9
18	15x15	89,01 *	39,2	85,41 *	49,8
19	30x15	22,09	189,7	20,68	213,2
20	50x15	8,40	527,6	8,23	609,4

Os gráficos a seguir mostram as soluções obtidas para duas instâncias do conjunto C1 para esta melhor combinação. Pelos gráficos, claramente percebe-se que a partir da iteração 200, ocorre uma intensificação na região onde as melhores soluções foram sendo encontradas durante a busca.

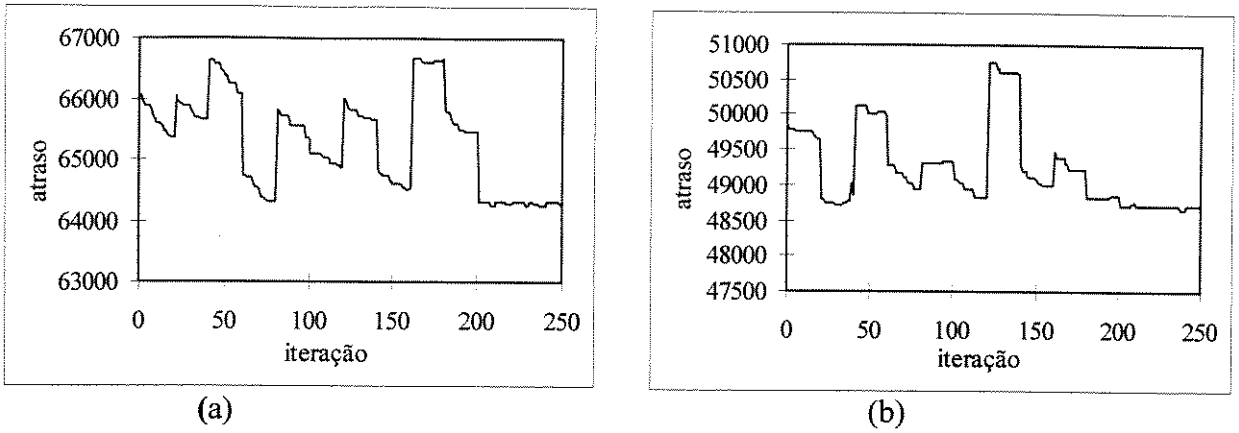


Gráfico 4.5. Soluções obtidas por HLPDI para instância de (a) Gp6 e de (b) Gp13

Considerando então a incorporação da memória de longo prazo, pode-se concluir que a busca se torna mais eficiente na maioria dos casos, pois os resultados mostram que a melhoria aumenta, principalmente para esta melhor estratégia que combina os dois elementos, enquanto o tempo computacional se mantém equivalente ao apresentado usando memória de curto prazo.

4.3.3. Comparação com outros algoritmos

Para fins de comparação da heurística, além das regras de despacho foram implementados os dois métodos heurísticos desenvolvidos para *job shop* considerando atraso, e que estão descritos no capítulo 1. Os algoritmos dos métodos se encontram no apêndice B. A implementação de ambos também foi feita em linguagem C e os testes efetuados no conjunto C1, em estações SUN SPARCclassic. O método proposto por Raman e Talbot (1995), denotado por **GSP**, pára, no trabalho apresentado, de acordo com um certo tempo no equipamento utilizado pelos autores (IBM 3090-600 mainframe) ou porque não se consegue mais melhorar a solução. Neste trabalho, os tempos de execução usados como critério de parada para o GSP são os tempos médios obtidos pela heurística HCP, que se encontram na Tabela 4.4. O método proposto por He et al. (1993), chamado de **MEHA**, pára quando não se consegue mais melhorar a solução. O método GSP utiliza como solução de partida a solução da regra MOD, com as datas de entrega das operações geradas de acordo com o exposto no apêndice B, e MEHA utiliza a solução dada pela regra MDD. A seguir são apresentadas tabelas comparativas da heurística com memória de curto prazo HCP1 e da heurística com

memória de longo prazo HLPDI com os dois métodos, onde as melhorias mostradas foram calculadas em relação à solução da regra MDD.

Tabela 4.8. - Valores de atraso obtidos pelos métodos heurísticos MEHA e GSP

Gp	(J x M)	MEHA	GSP	T(s) - MEHA	T(s) - GSP
1	5x5	1.017,6	**1.015,8**	0,02	0,28
2	10x5	3.595,8	**3.477,4**	0,09	1,40
3	50x5	**61.526,8**	71.149,8	12,48	134,84
4	10x10	4.830,4	**4.435,2**	0,38	15,51
5	15x10	9.114,8	**8.410,6**	1,46	35,38
6	50x10	**66.695,8**	71.995,2	78,26	363,50
7	100x10	**229.381,6**	275.962,2	703,74	1.443,44
8	15x15	9.267,6	**8.769,8**	3,38	58,95
9	30x15	**26.968,8**	26.988,8	35,89	293,96
10	50x15	**60.608,2**	63.740,2	219,08	563,45
11	5x5	90,0	**71,2**	0,01	0,19
12	10x5	1.088,0	**969,2**	0,08	5,03
13	50x5	**48.536,4**	58.227,4	11,98	182,26
14	10x10	397,6	**138,2**	0,17	14,46
15	15x10	2.152,6	**1.563,8**	1,01	38,05
16	50x10	**42.370,4**	48.185,0	66,72	328,99
17	100x10	**179.409,4**	219.670,6	600,71	1.429,45
18	15x15	397,2	**78,4**	1,40	59,81
19	30x15	7.054,2	**6.700,4**	20,69	336,82
20	50x15	**26.463,0**	29.768,0	134,53	717,36

Por esta tabela pode-se notar que para problemas com bastante atraso, o método GSP é inferior ao método MEHA, além de possuir um tempo muito maior.

Tabela 4.9. - Valores de melhorias obtidas pela heurística de curto e longo prazo e pelos métodos MEHA e GSP

Gp	M-HCP1(%)	M-HLPDI(%)	M-MEHA(%)	M-GSP(%)
1	13,33	12,08	3,43	3,57
2	6,74	6,64	1,13	4,58
3	1,05	3,40	0,15	-15,46
4	11,94	15,68	0,56	8,71
5	8,34	10,58	1,07	8,38
6	1,97	4,57	0,39	-7,52
7	0,79	2,28	0,08	-20,24
8	7,09	8,88	0,79	5,92
9	3,35	4,42	0,53	0,54
10	2,20	4,02	0,48	-4,68
11	84,47 *	86,41 *	3,40	16,18
12	19,60	19,75	1,39	12,02
13	1,04	1,99	0,18	-19,93
14	95,14 *	92,41 *	2,04	62,83
15	40,74	32,77	1,35	27,25
16	2,71	5,58	0,45	-13,24
17	0,88	2,03	0,09	-22,38
18	83,20 *	89,01 *	10,73	62,56 *
19	14,69	22,09	2,44	7,85
20	4,04	8,40	0,66	-11,79

Como pode-se notar, a heurística com a solução inicial dada pela regra MDD, se mostrou muito superior a ambos os métodos. Mesmo sendo o método MEHA mais rápido, a qualidade de solução cai muito quando comparada a HCP1 e HLPDI.

Os números negativos que aparecem na Tabela 4.9 em relação ao método GSP vem do fato da solução de partida do método, ser em geral pior que a regra MDD e mesmo o método conseguindo melhorar a solução inicial, ele não é capaz de alcançar a solução da regra MDD. Por causa disso é apresentada também uma comparação da melhoria obtida pelo método GSP e pela heurística HCP, em relação à solução inicial dada pela MOD com as datas de entrega geradas de acordo com a fórmula usada por Raman e Talbot, exposta no apêndice B.

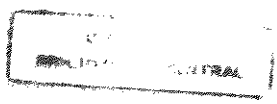


Tabela 4.10. - Valores de melhoria obtidas pela heurística HCP e pelo método GSP em relação à solução inicial usada por Raman e Talbot (1995)

Gp	M-HCP(%)	M-GSP(%)
1	14,54	2,24
2	4,76	4,64
3	1,00	0,38
4	8,48	3,45
5	4,87	5,04
6	1,45	2,53
7	0,51	-1,41
8	6,48	2,69
9	3,13	5,13
10	1,65	1,79
11	79,71 *	13,81
12	24,89	15,41
13	1,80	-1,55
14	50,62 *	-2,41
15	41,94	11,13
16	1,68	-0,85
17	0,84	0,38
18	----- *	-21,57
19	13,01	8,63
20	3,81	1,92

Por esta tabela, nota-se que a heurística é melhor que o método GSP na grande maioria dos grupos. Os números negativos na coluna de melhoria do GSP ocorrem porque a solução inicial não é armazenada pelo método (ver apêndice B).

Desta comparação pode-se concluir que a heurística, tanto a que utiliza BT com memória de curto prazo (HCP) como com memória de longo prazo (HLPDI), se mostrou bem mais eficiente na obtenção de soluções de qualidade do que os métodos MEHA e GSP, e em um tempo computacional razoável. Esses resultados são muito animadores pois sabe-se que estes métodos são os únicos encontrados na literatura para o problema de *job shop* tradicional, quando se considera datas de entrega. Portanto, pode-se afirmar que o uso de BT neste

problema é bastante eficiente tendo-se obtido excelentes resultados.

4.3.4. Comparação com solução ótima

A solução ótima do problema foi obtida através da resolução da formulação inteira pelo pacote CPLEX[1]. A formulação inteira clássica, apresentada em Baker (1974) para o problema de *job shop* com minimização do makespan, foi adaptada para o critério do atraso total, e é mostrada a seguir. Considere a seguinte notação:

J - número total de tarefas;

M - número total de máquinas;

p_{jk} - tempo de processamento da tarefa j na máquina k;

p_{j1} - tempo de processamento da tarefa j na sua primeira máquina;

d_j - data de entrega da tarefa j;

G - valor positivo muito grande.

C_{jk} - instante de término da tarefa j na máquina k;

C_{j1} - instante de término da tarefa j na sua primeira máquina;

C_{jm_j} - instante de término da tarefa j na sua última máquina m_j;

T_j - atraso da tarefa j;

X_{jpk} : assume valor 1 caso a tarefa j seja processada antes da tarefa p na máquina k;

caso contrário, assume valor 0.

Suponha também que a operação i da tarefa j requer a máquina k e a operação (i–1) da tarefa j requer a máquina h.

[1] Versão 3.0 de "Using the CPLEX Callable Library including Using the CPLEX Linear Optimizer with CPLEX Barrier and Mixed Integer Solvers", CPLEX Optimization, Inc., 1989-1994.

Formulação:

$$\text{minimizar} \quad \sum_{j=1}^{J} T_j \tag{1}$$

$$\text{sujeito a} \quad T_j \geq 0 \qquad j = 1,..., J \tag{2}$$

$$T_j \geq C_{jm_j} - d_j \qquad j = 1,..., J \tag{3}$$

$$C_{j1} \geq p_{j1} \qquad j = 1,..., J \tag{4}$$

$$C_{jk} - C_{jh} \geq p_{jk} \qquad (j, i-1, h) \rightarrow (j, i, k) \tag{5}$$

$$C_{pk} - C_{jk} \geq p_{pk} - G.(1 - X_{jpk}) \qquad j = 1,..., J;\ p = 1,..., J;\ k = 1,..., M; \tag{6}$$

$$C_{jk} - C_{pk} \geq p_{jk} - G.(X_{jpk}) \qquad j = 1,..., J;\ p = 1,..., J;\ k = 1,..., M; \tag{7}$$

$$C_{jk} \geq 0 \qquad j = 1,..., J;\ k = 1,..., M \tag{8}$$

$$X_{jpk} \in \{0,1\} \qquad j = 1,..., J;\ p = 1,..., J\ k=1,..., M \tag{9}$$

A função objetivo (1) representa a soma dos atrasos. Os conjuntos de restrições (2) e (3) correspondem à linearização de $T_j = \max (C_{jm_j} - d_j , 0)$. Os conjuntos (4) e (5) indicam que uma operação de uma tarefa não pode ser concluída antes que a operação precedente a esta tenha sido terminada. As restrições disjuntivas (6) e (7) garantem que não mais de uma operação é processada em cada máquina em qualquer instante de tempo. O conjunto de restrições (8) e (9) estabelecem, respectivamente, que as variáveis C_{jk} são contínuas e não negativas, e que as variáveis X_{jpk} são binárias. O número de variáveis contínuas não negativas é J+J.M, e o número de variáveis binárias X_{jpk} é M.J.(J–1)/2, dado que X_{pjk} não precisa ser definida se X_{jpk} esta' na formulação. O número total de restrições é $J+J.M+J.(J-1).M=J+J^2.M$.

A formulação foi resolvida em uma SUN SPARCstation IPX com clock de 40 MHz, 64MB de RAM e sistema operacional SUNOS 4.1.1, para 10 dimensões (J×M) diferentes, com 10 instâncias cada, geradas de acordo com a Tabela 4.1. Quando, para este problema, o

número de variáveis inteiras ultrapassou 80, o pacote CPLEX não foi capaz de resolver em um tempo computacional razoável. Como exemplo, uma instância 8×3 com 84 variáveis binárias, consumiu mais de 18 horas de execução.

A seguir são mostrados o tempo computacional do pacote e da heurística HLPDI, com 250 iterações, na estação SUN IPX e o desempenho da heurística HLPDI em relação à solução ótima. Para as dimensões de 1 a 10 foi usado β_1 para gerar as datas de entrega e para as dimensões de 11 a 20, β_2. Na tabela estão anotadas a quantidade de instâncias que se encontram a um determinado *gap* da solução ótima.

Tabela 4.11. - Comportamento da heurística HLPDI em relação à solução ótima

Dimensões (J×M)	**T(s) Cplex**	**T(s) HLPDI**	**ótimo**	**0,1 a 2%**	**2 a 5%**	**5 a 10%**	**> 10%**	**pior caso**
1- 3×3	0,13	0,39	10	0	0	0	0	---
2- 4×3	0,86	0,66	10	0	0	0	0	---
3- 5×3	4,84	0,86	6	3	1	0	0	4,63
4- 6×3	54,35	1,19	4	3	2	1	0	5,27
5- 7×3	1.295,25	1,55	5	1	2	2	0	7,62
6- 4×4	1,13	0,84	9	1	0	0	0	1,54
7- 5×4	9,10	1,20	4	4	2	0	0	3,23
8- 6×4	140,82	1,64	4	2	2	1	1	12,46
9- 5×5	10,53	1,53	5	2	1	0	2	10,23
10- 6×5	115,87	2,04	4	2	1	2	1	11,40
11- 3×3	0,11	0,16	10	0	0	0	0	---
12- 4×3	0,56	0,64	10	0	0	0	0	---
13- 5×3	3,69	0,94	10	0	0	0	0	---
14- 6×3	37,38	1,27	7	1	1	0	1	17,14
15- 7×3	626,87	1,60	3	2	2	2	1	11,68
16- 4×4	0,77	0,66	9	0	0	0	1	63,60
17- 5×4	5,22	1,32	9	0	0	1	0	9,02
18- 6×4	59,21	1,80	7	0	1	1	1	17,02
19- 5×5	4,47	0,44	10	0	0	0	0	---
20- 6×5	51,37	1,82	9	0	0	0	1	29,37

Pode-se notar por esta comparação que a heurística com memória de longo prazo (HLPDI), consegue obter a solução ótima em 61% das instâncias para β_1 e em 84% para β_2. Portanto, quando as datas de entrega são mais folgadas a heurística consegue chegar na solução ótima com mais eficiência. Outra conclusão interessante é que à medida que aumenta-se o número de tarefas e mantém-se fixo o número de máquinas, o número de instâncias onde a heurística obtém a solução ótima diminui.

A Tabela 4.12 apresenta o valor da solução inicial dada pela regra de despacho MDD (**SI-MDD**) e a porcentagem de melhoria da heurística em relação à solução inicial (**M-HLPDI**). Os valores são a média das 10 instâncias para cada dimensão.

Tabela 4.12.- Valor de atraso da regra MDD e porcentagem de melhoria de HLPDI

Dimensões	**SI-MDD**	**M-HLPDI(%)**
1- 3×3	363,0	4,86
2- 4×3	619,7	7,65
3- 5×3	879,5	9,25
4- 6×3	1.192,9	8,48
5- 7×3	1.564,2	6,59
6- 4×4	716,9	7,62
7- 5×4	1.032,9	8,11
8- 6×4	1.396,5	8,28
9- 5×5	1.087,3	11,69
10- 6×5	1.457,9	11,66
11- 3×3	23,3	30,01
12- 4×3	68,1	26,37
13- 5×3	172,6	26,14
14- 6×3	333,6	24,51
15- 7×3	520,9	15,63
16- 4×4	51,1	53,38
17- 5×4	124,0	38,69
18- 6×4	248,1	31,04
19- 5×5	90,7	73,41
20- 6×5	174,9	62,61

De acordo com os resultados apresentados, pode-se notar que a heurística HLPDI se comporta bem para as dimensões testadas em relação à solução ótima e, além disso, ela consegue uma melhoria razoável em relação à regra de despacho, sendo que para problemas com datas de entrega mais folgadas (de 11 a 20) a melhoria é maior que para os que possuem datas de entrega mais apertadas (de 1 a 10).

CAPÍTULO 5

TESTES E RESULTADOS PARA JOB SHOP FLEXÍVEL

5.1. Introdução

Neste capítulo são apresentados os resultados obtidos pela heurística desenvolvida para o problema flexível. Os testes para ajustar os componentes da BT de curto prazo foram feitos somente para o subproblema de roteamento, enquanto que para o de programação usou-se os componentes que já haviam sido ajustados e usados na heurística HCP, e que se encontram resumidos na Tabela 4.3. A incorporação de memória de longo prazo foi feita de acordo com as estratégias e implementações que apresentaram os melhores resultados no caso tradicional.

A heurística foi implementada em linguagem C e os testes realizados em estações de trabalho SUN SPARCclassic com clock de 50 MHz, 48 MB de RAM e sistema operacional SUNOS 4.1.3. Um conjunto de instâncias C2 usado para os testes se encontra explicado a seguir.

5.2. Geração das instâncias de teste

As instâncias geradas para o problema flexível diferem um pouco das geradas para o caso tradicional. Neste caso, cada instância é caracterizada por dois parâmetros a mais que no caso tradicional, que são número de máquinas alternativas para cada operação (MA) e máquinas que podem processar as operações (m_{ij}). A seguir são mostrados os dados utilizados para o conjunto C2. Considere que $\lceil x \rceil$ denota o menor inteiro maior ou igual a x, $t_{i,j,mkl}$ representa o tempo de processamento da operação i da tarefa j na máquina m_{kl} e $\bar{t}_{ij}$ denota o tempo médio da operação i da tarefa j sobre as máquinas.

Tabela 5.1. Dados de geração das instâncias

Dados de entrada	Conjunto C2
J	5, 10, 15, 30, 50, 100
M	5, 10, 15
OPJ	U[M/2,M]
MA	$\lceil 0{,}2\times M \rceil, \lceil 0{,}5\times M \rceil, \lceil 0{,}8\times M \rceil$
m_{ij}	U[1,M]
t_{ij}	U[1,99] para m_{k1} $U[t_{i,j,mk1}, 3\times t_{i,j,mk1}]$ para $m_{k2},\ldots,m_{kMA}$
d_j	$\beta \times \sum_i \bar{t}_{ij}$
β	$\beta_1 = ((J\times M)/1000) + 0{,}5$
	$\beta_2 = \beta_1 + 0{,}5$

Na Tabela 5.1, os valores do parâmetro MA para o conjunto C2 definem problemas de alta, média e baixa flexibilidade, onde 80%, 50% e 20% das máquinas do sistema podem processar uma operação, respectivamente. Por causa dessa variação de flexibilidade, a geração dos tempos de processamento para uma operação segue uma regra diferente no conjunto C2, explicada a seguir. Cada operação tem MA máquinas onde pode ser processada, que são sorteadas de acordo com U[1,M]. Sorteia-se então a primeira máquina alternativa e o tempo de processamento da operação na mesma de acordo com U[1,99]. A partir daí, os tempos da operação nas outras máquinas alternativas são sorteados dependentes deste primeiro valor v e portanto, para as próximas máquinas alternativas, sorteia-se o tempo de processamento da operação entre U[v,3×v] e não entre U[1,99]. Note que $(3\times v) \leq 99$. Este procedimento foi adotado para evitar uma discrepância muito grande dos tempos da operação em máquinas alternativas.

As dimensões (J×M) usadas nos testes são as mesmas definidas para o problema tradicional, ou seja, 5×5, 10×5, 50×5, 10×10, 15×10, 50×10, 100×10, 15×15, 30×15, 50×15, e

para cada dimensão são geradas 5 instâncias. Os grupos são formados da mesma forma que já apresentado anteriormente no capítulo 4, sendo 10 deles (Gp1 a Gp10) gerados com β_1 e os outros 10 (Gp11 a Gp20) com β_2. Para o conjunto C2 tem-se então 100 instâncias para cada grau de flexibilidade, e portanto ao todo são 300 instâncias.

5.3. Ajuste dos componentes da heurística e alguns resultados

Nesta fase de ajustes, os testes foram feitos usando as instâncias dos grupos Gp1, Gp3, Gp4, Gp6, Gp7, Gp9, Gp11, Gp13, Gp14, Gp16, Gp17, Gp19. A heurística basicamente consiste da obtenção de uma solução factível e da busca de novas soluções, através dos procedimentos de troca de operações entre máquinas e de troca de operações na mesma máquina. A resolução do subproblema de roteamento consiste do procedimento de troca de operações entre máquinas. Neste passo, é preciso determinar quantas e quais operações são candidatas a serem avaliadas para trocar de máquina. Sabe-se que operações que pertencem ao caminho crítico de uma tarefa são as responsáveis pelo instante de término daquela tarefa e, levando isso em conta, só foram consideradas candidatas para trocarem de máquinas as operações que pertencem a algum caminho crítico. Para obter essa informação, associou-se a cada operação um contador (n_cc[operação]) que indica o número de caminhos críticos nos quais a operação aparece. O maior valor do contador, dentre as N operações, foi armazenado em mcc. Vale lembrar que para o problema flexível, N, em geral, não é igual a J×M, pois para cada tarefa o número de operações é gerado entre M/2 e M. Portanto, em relação à quantidade de operações que vão ser avaliadas testou-se:

- todas as operações que aparecem em pelo menos um caminho crítico das tarefas, ou seja, que satizfazem n_cc[operação]≥ 1.
- todas as operações que aparecem em pelo menos um caminho crítico das tarefas e são processadas na máquina com maior carregamento.
- um número de operações dependente da instância, isto é, são consideradas candidatas as operações r que satisfazem n_cc[r] ≥ critério. Os testes feitos para critério foram k_1.N, k_2.(J/M) e k_3.mcc. Os valores testados para k_1 foram 0,1; 0,2 e 0,5, para k_2 0,8; 1 e 1,5, e para k_3 0,7; 0,8 e 1.

Em relação a estes testes, os melhores resultados foram obtidos com os critérios k_2.(J/M) com k_2=1,5, e k_3.mcc com k_3=0,7, sendo que o primeiro apresenta resultados de melhor qualidade, porém com um tempo maior que o segundo. Resolveu-se então continuar os testes com estas duas possibilidades de escolha das operações candidatas.

O outro parâmetro, ainda neste procedimento, que deve ser ajustado são os pesos da ponderação usada na escolha da melhor operação para trocar de máquina. Como a função objetivo usada nesta fase é composta por (α×atraso total+(1−α)×carregamento total), testou-se para α os valores 0,5; 0,6; 0,7; 0,8; 0,9 e 1. Dentre estes valores observou-se que os melhores resultados foram obtidos quando se usou $\alpha \geq 0,7$, mas pela pequena diferença entre os resultados, optou-se por continuar os testes com todos os valores.

Além disso, foram testados dois critérios para acionar a troca de operações entre máquinas:

- a cada k iterações, com k testado em 15 e 20;
- depois de k iterações sem que haja atualização da solução incumbente, com k testado nos valores 10 e 15.

Deste teste observou-se que acionar a troca de operações entre máquinas a cada 20 iterações foi quem apresentou os melhores resultados, em média.

É muito importante ressaltar que toda vez que se inicia o procedimento de troca de operação entre máquinas, a solução incumbente é recuperada, e todas as informações usadas no procedimento são obtidas desta solução. Foi também testada a utilização da solução atual mas os resultados com a solução incumbente foram bem superiores.

Em relação ao atributo usou-se o par {op, m_o}, onde op indica a operação que trocou de máquina e m_o a máquina origem da troca. A restrição tabu proíbe o movimento que resultar num par {operação, máquina} que é tabu.

Para a duração da restrição tabu, testou-se duração aleatória, com os seguintes valores para o intervalo [t_{min}, t_{max}]:

- $t_{min} = 20$ e $t_{max} = 30$;
- $t_{min} = 0,8 \times L$ e $t_{max} = 1,2 \times L$, onde $L=((J+M)/2).e^{-J/5.M} + N/2.e^{-5.M/J}$ (Taillard, 1994);

° $t_{min} \in U[a,b]$, onde $a = 2$ e $b = a+\lfloor(J+M)/3\rfloor$ e $t_{max} \in U[A,B]$, onde $A = t_{min}+6$ e $B = A+\lfloor(J+M)/3\rfloor$ (Dell'Amico e Trubian, 1993).

O último valor do intervalo foi o escolhido.

Todos estes ajustes foram feitos executando 250 iterações da heurística com a solução inicial dada pela regra MDD. Esta regra foi escolhida pois foi quem mostrou um melhor comportamento, como pode se ver pelos resultados apresentados a seguir, que são as soluções iniciais obtidas pelas regras MDD, MOD, CR+SPT, S/RPT+SPT, para os três níveis de flexibilidade.

Tabela 5.2. Regras de despacho para instâncias de baixa flexibilidade

Gp	(J x M)	SI1 - 20%	SI2 - 20%	SI3 - 20%	SI4 - 20%
1	5x5	607,8	**567,2**	**567,2**	**567,2**
2	10x5	**1.918,8**	1.959,8	1.959,8	1.959,8
3	50x5	**36.719,6**	43.198,4	43.198,4	43.198,4
4	10x10	2.446,8	**2.393,8**	**2.393,8**	**2.393,8**
5	15x10	4.433,6	**4.346,2**	**4.346,2**	**4.346,2**
6	50x10	**39.420,8**	46.137,8	45.778,2	46.137,8,
7	100x10	**149.971,2**	186.502,4	186.857,8	186.463,8
8	15x15	3.671,8	**3.569,4**	**3.569,4**	**3.569,4**
9	30x15	**11.185,6**	12.053,6	12.159,0	12.053,6
10	50x15	**27.586,0**	32.374,0	33.144,4	32.521,0
11	5x5	120,8	**96,6**	131,4	**96,6**
12	10x5	**916,2**	960,4	1.041,8	954,0
13	50x5	**32.214,6**	38.238,2	37.745,6	38.174,4
14	10x10	233,0	218,0	210,8	**204,4**
15	15x10	1.234,6	1.123,0	**1.092,8**	1.154,0
16	50x10	**29.647,2**	34.904,4	34.225,0	34.851,6
17	100x10	**129.075,4**	165.011,2	163.544,6	164.577,4
18	15x15	5,6	**4,8**	11,6	9,2
19	30x15	2.718,4	3.081,0	**2.276,2**	3.325,2
20	50x15	**14.568,0**	16.647,0	15.457,2	17.362,8

Notação:

SIi-k% - Valor da solução inicial obtida pela regra i, para grau k de flexibilidade no sistema, onde: i=1 MDD; i=2 MOD, I=3 CR+SPT; i=4 S/RPT+SPT.

Tabela 5.3. Regras de despacho para instâncias de média flexibilidade

Gp	(J x M)	SI1 - 50%	SI2 - 50%	SI3 - 50%	SI4 - 50%
1	5x5	**586,8**	612,2	612,2	612,2
2	10x5	**1.832,4**	1.857,2	1.857,2	1.857,2
3	50x5	**37.791,4**	44.812,6	44.812,6	44.812,6
4	10x10	2.188,0	**2.170,4**	**2.170,4**	**2.170,4**
5	15x10	4.417,0	**4.338,2**	**4.338,2**	**4.338,2**
6	50x10	**38.231,4**	45.705,0	45.863,8	45.705,0
7	100x10	**149.995,0**	189.812,0	190.772,0	191.285,4
8	15x15	**3.037,8**	3.188,8	3.188,8	3.188,8
9	30x15	**9.914,8**	11.126,4	11.066,6	11.177,0
10	50x15	**26.326,4**	32.150,4	31.804,6	32.151,4
11	5x5	**95,8**	106,2	101,0	106,2
12	10x5	**813,0**	839,4	918,2	800,8
13	50x5	**32.550,8**	39.284,6	39.783,4	39.306,2
14	10x10	**40,8**	82,2	81,0	58,8
15	15x10	1.139,2	846,0	**830,0**	950,2
16	50x10	**28.421,0**	34.790,2	34.229,2	34.760,0
17	100x10	**128.259,4**	168.257,4	166.888,4	169.329,0
18	15x15	**0,0**	**0,0**	**0,0**	**0,0**
19	30x15	2.599,4	2.591,8	**1.647,2**	3.308,4
20	50x15	**13.466,6**	16.449,2	13.945,0	17.102,2

Tabela 5.4. Regras de despacho para instâncias de alta flexibilidade

Gp	(J x M)	SI1 - 80%	SI2 - 80%	SI3 - 80%	SI4 - 80%
1	5x5	**504,0**	517,4	517,4	517,4
2	10x5	**1.805,4**	1.938,4	1.938,4	1.938,4
3	50x5	**37.777,6**	44.255,4	44.255,4	44.255,4
4	10x10	1.959,4	**1.958,4**	**1.958,4**	**1.958,4**
5	15x10	4.100,0	**3.948,0**	**3.948,0**	**3.948,0**
6	50x10	**38.660,0**	48.624,8	48.611,8	48.909,8
7	100x10	**150.686,0**	195.073,8	194.487,8	194.634,2
8	15x15	2.928,2	**2.925,8**	**2.925,8**	**2.925,8**
9	30x15	**9.250,4**	10.476,8	10.368,0	10.476,8
10	50x15	**25.955,0**	31.707,8	31.476,6	32.286,8
11	5x5	21,2	**14,4**	**14,4**	**14,4**
12	10x5	**675,4**	872,8	850,8	846,8
13	50x5	**32.422,4**	38.804,4	38.700,0	38.811,4
14	10x10	**1,8**	4,0	5,0	4,0
15	15x10	1.052,0	729,4	**584,2**	900,8
16	50x10	**28.374,2**	36.444,4	36.014,6	36.776,2
17	100x10	**130.360,8**	170.565,8	169.206,8	171.706,4
18	15x15	**0,0**	**0,0**	**0,0**	**0,0**
19	30x15	2.399,8	2.108,8	**915,4**	2.783,4
20	50x15	**12.837,0**	15.585,2	13.235,8	16.821,0

Pelas Tabelas 5.2, 5.3, 5.4, pode-se notar que a regra MDD é a que obtém os melhores resultados, principalmente quando se tem média e alta flexibilidade no sistema e para problemas com bastante atraso. Vale ressaltar aqui que nada se pode afirmar a respeito das regras de despacho em relação ao grau de flexibilidade, pois para cada operação estão sendo usados tempos de processamento diferentes para cada máquina, e portanto maior flexibilidade pode não implicar necessariamente em menor atraso.

Os resultados a seguir mostram os valores obtidos pela heurística do problema flexível, denotada por **HFCP,** com a solução inicial dada pela regra MDD, e também a melhoria obtida pela heurística (**M-HFCP**) em relação à regra. As instâncias onde a heurística obtém solução com atraso zero, e portanto onde a melhoria é de 100%, não são incluídas no

cálculo da média da porcentagem, e tais instâncias pertencem aos grupos marcados com *. Os grupos marcados somente com ---- indicam instâncias onde a solução inicial possui atraso zero.

Para os parâmetros de HFCP que ainda não haviam sido ajustados, utilizou-se:

- $n\_cc[r] \geq 1{,}5.(J/M)$ para o critério de escolha das operações candidatas (r);
- $\alpha = 0{,}9$ na ponderação dos objetivos.

Tabela 5.5. Resultados da heurística HFCP

Gp	HFCP - 20%	T(s)	HFCP - 50%	T(s)	HFCP - 80%	T(s)
1	521,0	1,42	492,8	2,73	504,0	0,00
2	1.858,6	2,69	1.804,8	2,84	1.805,0	1,56
3	36.686,0	43,77	37.790,2	68,89	37.745,6	24,99
4	2.139,2	30,78	1.950,4	63,27	1.705,2	221,66
5	4.183,0	75,53	4.183,2	163,84	3.755,0	398,94
6	39.405,8	305,17	38.231,0	477,63	38.660,0	635,22
7	149.958,8	1.305,64	149.995,0	2.323,53	150.686,0	2.685,87
8	3.333,0	175,74	2.682,4	497,01	2.526,8	1.467,66
9	11.088,2	344,03	9.912,4	607,27	9.249,2	831,02
10	27.444,2	1.307,92	26.321,8	3.882,20	25.955,0	3.944,72
11	51,2	2,24	33,4	1,78	21,2	0,01
12	836,8	2,70	811,0	3,65	675,4	1,65
13	32.127,2	40,85	32.550,8	59,83	32.422,4	16,16
14	63,4	94,76	23,0	196,87	0,0	2,05
15	958,2	92,22	901,4	234,51	633,2	921,68
16	29.574,0	466,56	28.416,4	619,89	28.374,2	794,91
17	129.074,6	1.396,42	128.259,4	2.941,20	130.360,8	2.268,96
18	0,0	5,92	0,0	0,18	0,0	0,19
19	2.648,0	708,50	2.591,2	1.313,47	2.399,6	1.714,34
20	14.510,4	1.939,69	13.465,6	4.994,66	12.835,8	5.585,36

Notação:

HFCP-k% - Valor da solução da heurística partindo da solução inicial obtida pela regra MDD para grau k de flexibilidade;

T - Tempo de 250 iterações da heurística.

Tabela 5.6. Melhorias obtidas pela heurística HFCP

Gp	(J x M)	M-HFCP - 20%	M-HFCP - 50%	M-HFCP - 80%
1	5x5	13,11	15,27	0,00
2	10x5	3,14	1,49	0,00
3	50x5	0,10	0,00	0,08
4	10x10	12,51	10,87	13,47
5	15x10	5,65	6,44	8,44
6	50x10	0,04	0,00	0,00
7	100x10	0,01	0,00	0,00
8	15x15	9,26	11,90	13,80
9	30x15	0,85	0,02	0,01
10	50x15	0,54	0,02	0,00
11	5x5	44,54	54,42 *	0,00
12	10x5	8,17	0,18	0,00
13	50x5	0,28	0,00	0,00
14	10x10	65,28	25,42 *	----- *
15	15x10	22,78	21,78	41,95
16	50x10	0,27	0,02	0,00
17	100x10	0,00	0,00	0,00
18	15x15	----- *	-----	-----
19	30x15	2,69	0,29	0,01
20	50x15	0,40	0,01	0,01

Destes resultados pode-se perceber que a heurística HFCP consegue melhorias maiores em relação a problemas com flexibilidade menor. Para se tentar melhorar os resultados obtidos pela heurística, foram incorporadas estratégias de intensificação e diversificação da BT.

5.4. Estratégias de intensificação e diversificação

As estratégias de intensificação e diversificação incorporadas à memória de curto prazo da BT para o problema flexível são basicamente as mesmas já mostradas para o problema tradicional, isto é, agem sobre o subproblema de programação. A diferença

fundamental é que as matrizes que armazenam as freqüências, tanto das soluções de elite quanto das soluções que são obtidas durante a busca, são matrizes tridimensionais que armazenam (operação, posição, máquina), pois neste caso a informação referente à máquina é muito importante, já que uma operação pode ser processada por várias máquinas. Vale ressaltar que estas matrizes são compostas pelas soluções que vão sendo obtidas através dos dois níveis de resolução.

As estratégias de diversificação e intensificação foram usadas de duas maneiras. Na primeira considera-se a aplicação de tais estratégias sobre a heurística desenvolvida para o problema flexível e na segunda, sobre a heurística desenvolvida para o problema tradicional, ou seja, sem o procedimento de troca de operações entre máquinas.

A primeira maneira é apresentada a seguir, e para facilitar ela é incluída (em negrito) no algoritmo de passos da heurística já apresentado no capítulo 3.

Passo 1: Construa a solução inicial usando alguma regra de despacho. Faça j=1, iter=0, inicialize o atraso incumbente e o atraso atual usando o atraso inicial.

Passo 2: a) Calcule os caminhos críticos das tarefas e o atraso atual. Atualize o atraso incumbente.

b) Se nenhuma tarefa estiver atrasada ou se o número máximo de iterações for satisfeito, PARE.

Passo 3: Se o critério para troca de operações entre máquinas for verdadeiro, então:

a) Avalie para determinadas operações todas as trocas possíveis para suas máquinas alternativas e escolha através da regra da melhor.

b) Execute a troca escolhida e calcule os caminhos críticos das tarefas, o atraso e atualize o atraso incumbente.

c) Se não existir troca possível, vá ao **Passo 3a**.

Passo 3a: Se o critério para iniciar a diversificação for verdadeiro, reinicie a regra utilizando a penalização e/ou use a penalidade na escolha dos movimentos do **Passo 5** por um certo número de iterações.

Passo 3b: Se o critério para iniciar a intensificação for verdadeiro, reinicie a regra usando o incentivo no cálculo da prioridade e/ou use o incentivo na escolha dos movimentos do **Passo 5** por um certo número de iterações.

Passo 4: a) Se j>J, faça j=1;

b) Se a tarefa j estiver atrasada, vá ao Passo 5.
Senão, j=j+1 e volte ao Passo 4a).

Passo 5: a) Avalie todas as inversões possíveis no caminho crítico da tarefa j e escolha de acordo com a regra da melhor (**incluindo penalização/incentivo se estiver diversificando/intensificando**).

b) Execute a inversão escolhida. Faça iter=iter+1, j=j+1 e volte ao Passo 2.

c) Se não existir inversão possível, faça j=j+1 e volte ao Passo 3.

O cálculo dos valores da prioridade da regra e do movimento, usando a penalidade e o incentivo é:

valor_prioridade=valor_prioridade+I*fE_op(o)_pos(p)_maq(m)

valor_prioridade=valor_prioridade–P*fA_op(o)_pos(p)_maq(m)

valor_movimento=valor_movimento–I*fE_op(x)_pos(p1)_maq(m1)+
–I*fE_op(y)_pos(p2)_maq(m1)).

valor_movimento=valor_movimento+P*fA_op(x)_pos(p1)_maq(m1)+
+P*fA_op(y)_pos(p2)_maq(m1)).

Nesta versão, pode-se notar que um novo roteamento pode ser obtido tanto da troca de operações de máquinas, como da diversificação e/ou intensificação. Deve-se ressaltar que o roteiro resultante do procedimento que faz a troca de operação de máquina só difere do anterior por uma operação que trocou de máquina, enquanto que o reinício da regra a partir da aplicação da diversificação ou intensificação pode resultar em um roteiro onde várias operações podem trocar de máquinas simultaneamente. É importante deixar claro que se o Passo 3 acabou de ser executado com sucesso, isto é, foi feita uma troca de operação entre máquinas, então mesmo que o critério para diversificação, no Passo 3a, ou para intensificação, no Passo 3b, seja verdadeiro, não é permitido fazer um reinício da regra para obter uma nova solução de partida. Neste caso, a diversificação ou intensificação só consiste em usar a

penalização ou incentivo sobre a escolha do movimento no Passo 5.

De todas as implementações feitas de diversificação e intensificação e de ambas combinadas, a que apresentou melhores resultados foi quando incorporou-se somente a seguinte estratégia de diversificação: depois de 10 iterações sem atualização da solução incumbente, inicia-se a diversificação, calculando uma nova solução de partida através do uso da penalidade e da freqüência no cálculo da prioridade da regra, e aplica-se penalização na escolha dos movimentos por 5 iterações. Os resultados para esta melhor versão são mostrados na tabela a seguir, onde **HFLP** denota o valor da solução da heurística com memória de longo prazo para o problema flexível e **T** o tempo de execução de 250 iterações da heurística HFLP.

Tabela 5.7. Valores obtidos pela melhor estratégia de memória de longo prazo

Gp	(J x M)	HFLP - 20%	T(s)	HFLP - 50%	T(s)	HFLP - 80%	T(s)
1	5x5	526,4	1,20	479,8	1,81	504,0	0,01
2	10x5	1.830,6	2,21	1.741,4	2,28	1.701,4	1,35
3	50x5	36.117,4	34,45	36.549,8	38,73	37.127,0	33,70
4	10x10	2.102,2	21,82	1.910,6	46,03	1.672,0	179,82
5	15x10	3.974,6	56,52	4.029,6	122,76	3.618,0	315,69
6	50x10	38.038,2	239,10	37.625,2	391,26	38.004,6	500,09
7	100x10	147.037,4	1.131,07	147.568,2	2.279,02	148.387,8	2.398,93
8	15x15	3.229,6	136,08	2.653,0	374,64	2.455,8	902,8
9	30x15	10.269,0	251,99	9.313,2	510,48	8.699,0	588,79
10	50x15	26.342,4	1.195,05	25.862,6	3.102,60	25.155,0	3.538,45
11	5x5	51,6	1,56	12,2	2,68	21,2	0,01
12	10x5	822,6	2,44	729,6	2,40	636,0	1,18
13	50x5	31.196,0	37,21	32.125,4	43,00	31.685,2	25,25
14	10x10	77,2	56,38	9,6	108,79	0,0	0,13
15	15x10	912,2	85,69	774,8	194,54	574,2	655,90
16	50x10	28.162,6	333,10	27.451,8	462,94	27.574,0	345,16
17	100x10	127.117,4	1.383,52	126.009,4	2.167,8	127.192,0	2.665,19
18	15x15	0,0	0,64	0,0	0,55	0,0	0,56
19	30x15	2.336,8	475,46	2.369,2	1.080,19	2.058,4	1.262,46
20	50x15	13.654,0	1.775,36	12.606,0	4.086,53	12.052,8	5.532,01

Tabela 5.8. Melhorias obtidas por HFLP

Gp	(J x M)	M-HFLP-20%	M-HFLP-50%	M-HFLP-80%
1	5x5	12,02	17,35	0,00
2	10x5	4,38	5,03	5,90
3	50x5	1,71	3,23	1,71
4	10x10	13,83	12,72	14,72
5	15x10	10,38	9,70	11,83
6	50x10	3,55	1,54	1,69
7	100x10	1,95	1,62	1,52
8	15x15	12,25	12,87	16,13
9	30x15	7,76	6,05	5,91
10	50x15	4,43	1,69	3,01
11	5x5	44,04	83,97 *	0,00
12	10x5	9,01	9,23	6,47
13	50x5	3,21	1,30	2,25
14	10x10	61,66	45,24 *	----- *
15	15x10	26,76	32,24	46,35
16	50x10	5,03	3,40	2,78
17	100x10	1,53	1,75	2,40
18	15x15	----- *	-----	-----
19	30x15	13,49	8,36	14,03
20	50x15	6,18	6,43	5,86

Pode-se constatar por esses resultados que os ganhos com a aplicação da diversificação são bem maiores que os ganhos obtidos pelo uso somente da memória de curto prazo. A diversificação, como no caso tradicional, parece agir mais sobre os grupos que apresentam atraso grande.

Em relação ao carregamento, pelas tabelas a seguir, pode-se perceber que em todos os grupos o carregamento médio da melhor solução obtida por HFLP (**C_melhor**) teve uma redução em relação ao da solução inicial (**C_ini**). Além disso, tem-se um bom balanceamento na melhor solução, pois os valores do carregamento médio (C_melhor), do maior carregamento das máquinas (**Max_C**) e do menor carregamento das máquinas (**Min_C**) são muito próximos. Vale lembrar que o carregamento de uma máquina consiste na soma dos

tempos de processamento das operações que são executadas na mesma.

Tabela 5.9. Valores de carregamento para instâncias de baixa flexibilidade

Gp	(J x M)	C_ini - 20%	C_melhor - 20%	Max_C - 20%	Min_C - 20%
1	5x5	200,8	192,8	267,0	85,0
2	10x5	407,8	399,4	448,0	355,0
3	50x5	1.948,0	1.932,6	1.994,0	1.862,0
4	10x10	462,6	438,6	576,0	313,0
5	15x10	661,3	626,5	755,0	530,0
6	50x10	2.228,4	2.198,5	2.276,0	2.110,0
7	100x10	4.454,6	4.393,3	4.459,0	4.312,0
8	15x15	688,2	663,1	771,0	546,0
9	30x15	1.341,3	1.304,1	1.376,0	1.230,0
10	50x15	2.238,8	2.212,8	2.271,0	2.133,0
11	5x5	200,8	195,2	273,0	129,0
12	10x5	407,4	398,2	454,0	338,0
13	50x5	1.959,0	1.937,8	2.003,0	1.865,0
14	10x10	460,2	444,3	538,0	314,0
15	15x10	664,0	633,8	750,0	477,0
16	50x10	2.246,5	2.208,0	2.274,0	2.146,0
17	100x10	4.451,2	4.426,6	4.490,0	4.346,0
18	15x15	690,3	687,3	749,0	612,0
19	30x15	1.341,2	1.318,3	1.416,0	1.239,0
20	50x15	2.252,9	2.218,5	2.287,0	2.119,0

Tabela 5.10. Valores de carregamento para instâncias de média flexibilidade

Gp	(J x M)	C_ini - 50%	C_melhor - 50%	Max_C - 50%	Min_C - 50%
1	5x5	214,2	199,8	249,0	150,0
2	10x5	419,2	408,4	456,0	364,0
3	50x5	2.047,6	2.003,2	2.043,0	1.948,0
4	10x10	481,1	458,6	551,0	338,0
5	15x10	708,4	665,6	787,0	524,0
6	50x10	2.296,5	2.278,7	2.336,0	2.227,0
7	100x10	4.608,7	4.578,3	4.622,0	4.533,0
8	15x15	692,8	665,7	761,0	566,0
9	30x15	1.384,6	1.362,0	1.400,0	1.317,0
10	50x15	2.336,1	2.318,9	2.366,0	2.270,0
11	5x5	214,2	204,4	249,0	158,0
12	10x5	424,6	416,2	452,0	378,0
13	50x5	2.044,0	2.029,4	2.079,0	1.968,0
14	10x10	484,2	472,7	541,0	392,0
15	15x10	712,8	677,2	788,0	578,0
16	50x10	2.330,7	2.290,6	2.341,0	2.241,0
17	100x10	4.600,7	4.564,5	4.621,0	4.514,0
18	15x15	698,1	698,1	752,0	652,0
19	30x15	1.383,0	1.368,3	1.418,0	1.322,0
20	50x15	2.350,3	2.315,1	2.364,0	2.272,0

Tabela 5.11. Valores de carregamento para instâncias de alta flexibilidade

Gp	(J x M)	C_ini - 80%	C_melhor - 80%	Max_C - 80%	Min_C - 80%
1	5x5	223,8	223,8	265,0	191,0
2	10x5	453,6	440,6	480,0	408,0
3	50x5	2.126,4	2.103,6	2.134,0	2.071,0
4	10x10	486,3	453,4	546,0	355,0
5	15x10	708,3	661,5	744,0	552,0
6	50x10	2.397,1	2.378,3	2.426,0	2.334,0
7	100x10	4.701,3	4.673,1	4.722,0	4.633,0
8	15x15	706,3	672,4	766,0	587,0
9	30x15	1.384,5	1.357,2	1.412,0	1.323,0
10	50x15	2.339,5	2.321,6	2.371,0	2.285,0
11	5x5	223,8	223,8	265,0	191,0
12	10x5	447,2	441,6	480,0	403,0
13	50x5	2.115,6	2.097,0	2.142,0	2.053,0
14	10x10	489,6	488,4	535,0	450,0
15	15x10	707,8	661,3	778,0	544,0
16	50x10	2.410,8	2.384,6	2.432,0	2.343,0
17	100x10	4.720,3	4.671,8	4.717,0	4.632,0
18	15x15	711,9	711,9	759,0	674,0
19	30x15	1.386,9	1.366,3	1.415,0	1.325,0
20	50x15	2.352,2	2.322,5	2.368,0	2.282,0

A seguir é detalhada a segunda maneira de aplicação das estratégias de diversificação e intensificação. Esta versão é a mesma descrita no capítulo 4, pois foi aplicada sobre a heurística HCP. A diferença é que aqui são usadas as matrizes de freqüência tridimensionais. O esquema desta versão pode ser apresentado da seguinte maneira:

- Obtenção da solução inicial;
- Busca de novas e melhores soluções, através dos seguintes passos:

  - se o critério para diversificação for verdadeiro, então reinicie a regra usando penalidade no cálculo da prioridade;

  - se critério para intensificação for verdadeiro, então reinicie a regra usando incentivo

no cálculo da prioridade;

- troca de operações nas máquinas. Se estiver diversificando/intensificando, use penalidade/incentivo no cálculo do valor do movimento, por um certo número de iterações.

Nesta versão sempre é feito um reinício da regra para se tentar obter uma nova solução com um novo roteamento, diferente do anterior. Este teste foi feito para se tentar medir a importância do procedimento de troca de operações entre máquinas. A melhor versão para esta aplicação foi usar somente a diversificação, da seguinte forma: inicia-se a diversificação depois de 10 iterações sem que haja atualização da solução incumbente, calculando uma nova solução de partida (reinício) e penalizando os movimentos por 5 iterações. A seguir são mostrados os resultados obtidos por esta versão, denotada por **HTLP**. O tempo **T**, em segundos, refere-se a 250 iterações de HTLP.

Tabela 5.12. Valores de atraso obtidos por HTLP para o conjunto C2

Gp	(J x M)	HTLP - 20%	T(s)	HTLP - 50%	T(s)	HTLP - 80%	T(s)
1	5x5	593,8	0,70	518,2	0,60	504,0	0,01
2	10x5	1.792,8	2,03	1.710,6	1,90	1.681,2	1,36
3	50x5	36.174,2	36,35	36.802,6	32,65	37.021,6	21,25
4	10x10	2.314,4	4,81	2.039,8	4,45	1.872,4	3,75
5	15x10	4.171,2	11,45	4.194,4	10,28	3.851,0	7,45
6	50x10	38.010,2	98,24	37.163,6	84,34	37.867,2	50,70
7	100x10	146.307,0	406,7	146.893,8	352,25	148.223,6	208,20
8	15x15	3.508,0	16,80	2.865,2	14,71	2.770,4	12,50
9	30x15	10.405,0	56,92	9.354,4	45,80	8.627,8	34,06
10	50x15	26.372,6	182,50	25.633,8	157,06	25.125,2	103,48
11	5x5	110,4	0,74	40,8	0,46	21,2	0,01
12	10x5	790,0	2,25	691,8	2,05	650,8	1,43
13	50x5	31.448,0	44,52	32.057,4	38,01	31.413,8	24,51
14	10x10	155,6	5,76	7,8	3,33	0,0	0,14
15	15x10	1.094,6	12,36	948,6	11,76	885,0	8,05
16	50x10	27.983,4	128,28	27.055,4	109,93	26.923,4	59,08
17	100x10	125.440,8	484,63	126.297,4	412,8	126.845,4	234,98
18	15x15	0,0	0,68	0,0	0,55	0,0	0,57
19	30x15	2.468,0	77,3	2.127,6	63,18	2.036,8	44,20
20	50x15	13.589,0	227,47	12.609,2	176,97	11.859,4	111,94

Tabela 5.13. Melhorias obtidas por HTLP

Gp	(J x M)	M-HTLP - 20%	M-HTLP - 50%	M-HTLP - 80%
1	5x5	2,01	11,20	0,00
2	10x5	6,16	6,71	6,92
3	50x5	1,53	2,59	2,00
4	10x10	5,21	6,80	4,05
5	15x10	5,99	5,79	5,98
6	50x10	3,50	2,74	2,04
7	100x10	2,43	2,07	1,62
8	15x15	4,60	5,71	5,15
9	30x15	6,65	5,62	6,60
10	50x15	4,33	2,61	3,17
11	5x5	10,87	55,47	0,00
12	10x5	12,94	13,14	4,29
13	50x5	2,43	1,48	3,10
14	10x10	22,03	20,77 *	----- *
15	15x10	12,22	15,45	16,03
16	50x10	5,63	4,80	5,07
17	100x10	2,84	1,52	2,66
18	15x15	----- *	-----	-----
19	30x15	8,79	17,93	15,14
20	50x15	6,61	6,43	7,52

Comparando estes resultados com os das Tabelas 5.7 e 5.8, tem-se que:

- para flexibilidade 20%, HFLP obtém melhorias maiores que HTLP nos grupos 1, 3, 4, 5, 6, 8, 9, 10, 11, 13, 14, 15, 19. Apesar dos tempos de HFLP serem maiores que os de HTLP, na maioria dos grupos, eles ainda são baixos, excluindo-se Gp10 onde o tempo de HFLP é extremamente maior que o de HTLP. Neste caso fica claro que HTLP é bem mais vantajosa.

- para flexibilidade 50%, HFLP obtém melhorias maiores que HTLP nos grupos 1, 3, 4, 5, 8, 9, 11, 14, 15, 17, sendo que para Gp17 o tempo de HFLP é muito superior ao de HTLP.

- para 80%, HFLP consegue melhorias maiores que HTLP nos grupos 4, 5, 8, 12, 15, e para todos estes HFLP ainda tem um tempo pequeno.

Pode-se notar também que para os grupos com pouco atraso, HFLP, em geral, obtém melhorias maiores que HTLP. Além disso, percebe-se que quanto maior a flexibilidade, mais as melhorias de HTLP se igualam ou superam as de HFLP. Com relação ao tamanho dos problemas, nota-se que quando a relação entre o número de tarefas e o número de máquinas é maior que 3, em geral, HFLP possui um tempo computacional bem maior que HTLP, enquanto que as melhorias são equivalentes. Isto mostra que o procedimento de trocas de máquinas é bem custoso e pode não valer a pena em certos casos.

Para uma melhor avaliação do comportamento das heurísticas, resolveu-se fazer um teste com 500 iterações. As tabelas a seguir mostram as melhorias obtidas em relação a solução inicial e também os tempos de execução para 500 iterações. Para os grupos marcados com **não**, a heurística HFLP não foi executada, pois 250 iterações já foram suficientes para que fossem tiradas conclusões.

Tabela 5.14. Melhorias e tempos obtidos por HFLP com 500 iterações

Gp	(J x M)	M-HFLP-20%	T(s)	M-HFLP-50%	T(s)	M-HFLP-80%	T(s)
1	5x5	12,02	2,75	17,35	3,90	0,00	0,01
2	10x5	5,54	4,96	5,03	5,27	7,14	3,04
3	50x5	1,74	80,19	3,23	84,36	2,15	71,12
4	10x10	14,36	46,08	14,68	100,08	16,59	393,62
5	15x10	12,54	128,83	11,35	264,74	14,48	713,34
6	50x10	3,75	572,78	2,03	889,86	2,14	1.131,78
7	100x10	**não**		**não**		**não**	
8	15x15	16,45	299,23	17,00	798,55	20,98	1.792,51
9	30x15	8,10	549,25	6,72	1.166,23	6,52	1.303,47
10	50x15	**não**		**não**		**não**	
11	5x5	44,04	3,62	83,97 *	6,31	0,00	0,00
12	10x5	9,99	5,32	12,67	5,48	6,47	2,46
13	50x5	3,54	79,64	1,99	113,06	2,53	55,0
14	10x10	61,66	130,55	45,24 *	241,94	----- *	0,13
15	15x10	28,81	178,35	36,63	453,05	51,77	1.404,62
16	50x10	5,21	775,25	4,94	1.069,58	3,69	969,42
17	100x10	**não**		**não**		**não**	
18	15x15	----- *	0,65	----	0,55	-----	0,57
19	30x15	14,94	1.177,54	14,14	2.860,76	14,82	2.942,93
20	50x15	**não**		**não**		**não**	

Tabela 5.15. Melhorias e tempos obtidos por HTLP com 500 iterações

Gp	(J x M)	M-HTLP-20%	T(s)	M-HTLP-50%	T(s)	M-HTLP-80%	T(s)
1	5x5	2,01	1,42	11,20	1,16	0,00	0,01
2	10x5	6,25	4,28	7,56	3,91	8,09	2,78
3	50x5	2,28	83,19	2,59	73,63	2,75	46,43
4	10x10	5,21	10,51	8,09	9,57	4,05	7,78
5	15x10	7,34	22,55	6,42	21,21	6,61	14,65
6	50x10	4,25	217,45	2,74	187,66	2,27	108,75
7	100x10	2,56	911,22	2,29	794,11	1,95	442,54
8	15x15	4,60	31,81	5,76	28,38	6,83	24,39
9	30x15	7,60	124,36	6,53	101,91	7,36	72,51
10	50x15	4,82	372,99	3,11	333,05	4,10	204,15
11	5x5	10,87	1,47	55,47	0,91	0,00	0,01
12	10x5	14,75	4,51	13,49	4,10	5,65	2,84
13	50x5	3,48	87,23	2,34	75,95	3,10	48,64
14	10x10	23,15	11,23	20,77 *	5,41	----- *	0,14
15	15x10	15,45	24,33	19,88	23,38	18,09	15,75
16	50x10	6,33	254,38	5,04	220,28	5,50	115,97
17	100x10	2,87	968,51	1,90	825,20	2,75	468,61
18	15x15	----- *	0,68	-----	0,55	-----	0,57
19	30x15	11,57	155,16	18,07	125,52	19,11	86,55
20	50x15	7,03	451,36	7,48	347,15	7,89	222,81

Dos resultados obtidos pelas Tabelas 5.14 e 5.15 pode-se concluir que ambas continuam tendo um desempenho semelhante, mesmo considerando o dobro de iterações. A heurística HTLP passa a obter melhorias maiores que HFLP em Gp3, Gp6 para 20% de flexibilidade. Nos grupos onde HFLP superava HTLP com uma margem grande a situação continua a mesma. Em relação ao tempo, novamente HFLP gasta mais tempo, o que já era esperado, mas às vezes isto pode compensar.

Um aspecto interessante que pode ser notado nas Tabelas 5.12 e 5.15, é que a vizinhança do problema de programação parece ficar cada vez menor quanto maior a flexibilidade, ou seja, quanto mais flexível o problema, menor é o número de inversões possíveis nos caminhos críticos das tarefas. Isto foi percebido, pois nos resultados houve uma

diminuição do tempo computacional com o aumento da flexibilidade.

Através de todos os resultados apresentados aqui para o problema flexível, pode-se concluir que é fundamental incorporar à heurística uma estratégia de diversificação, através de memória de longo prazo.

CAPÍTULO 6

DISCUSSÕES FINAIS E CONCLUSÕES

O objetivo deste trabalho foi estudar e desenvolver um método de solução para o problema de programação de tarefas nos ambientes *job shop* tradicional e *job shop* flexível, usando como medida de desempenho o atraso total das tarefas.

O problema de programação em um *job shop* tradicional é considerado um dos problemas de otimização combinatória de mais difícil resolução, e isto motivou o desenvolvimento de uma heurística para sua resolução. A heurística proposta parte de uma solução inicial obtida através de regras de despacho que consideram datas de entrega, e busca novas soluções através de uma vizinhança gerada pelos caminhos críticos das tarefas. Essa busca é guiada pela metaheurística Busca Tabu. A motivação para utilizar Busca Tabu veio dos excelentes resultados que esta técnica tem obtido para este problema com a medida de desempenho *makespan*.

A avaliação do desempenho da heurística foi feita em três fases. Na primeira, foram feitos testes para escolha da melhor regra de despacho usada para obter a solução inicial e para ajustar os elementos de memória de curto prazo da BT. Em relação à vizinhança utilizada, existem algumas discussões interessantes. Depois de feitos os testes com a ampliação da vizinhança, tentando inverter mais que um arco simultaneamente, isto é, trocar três operações adjacentes na mesma máquina, e de se notar que a melhoria obtida por esta ampliação não compensa em relação ao aumento no tempo computacional, pode-se concluir que o tipo de movimento parece muito restrito para que se consiga grandes mudanças. Porém, a utilização de uma vizinhança mais rica, do tipo inserção de uma operação ou trocas de quaisquer operações numa máquina, recai no problema da obtenção de soluções factíveis, e portanto na elaboração de testes eficientes para se garantir a factibilidade, isto é, a ausência de ciclo no grafo. Portanto, a utilização de outras vizinhanças é um ponto em aberto para ser estudado.

Em relação às regras de despacho testadas observou-se que a regra MDD foi a que obteve os melhores resultados.

A heurística proposta foi comparada com a sua respectiva solução inicial e pode-se notar que as melhorias obtidas foram boas, principalmente para problemas com pouco atraso. Tentando melhorar ainda mais o desempenho da heurística veio a segunda fase de testes, onde estratégias de diversificação e intensificação da BT foram incorporadas à heurística. De acordo com os resultados, pôde-se observar que a melhor versão da incorporação destas estratégias (HLPDI) conseguiu superar as melhorias obtidas pela versão de curto prazo (HCP1) em 80% dos grupos, e com praticamente o mesmo tempo computacional. Vale ressaltar que HLPDI parece agir melhor em problemas com bastante atraso.

Na terceira fase, os resultados de HCP1 e HLPDI foram comparados com os de outros dois métodos heurísticos, MEHA e GSP, encontrados na literatura e que também foram implementados. A conclusão destes testes é que tanto a heurística com memória de curto prazo (HCP1), como a heurística com memória de longo prazo (HLPDI), é muito superior a ambos. Isto é muito animador, pois estes métodos heurísticos foram os únicos encontrados na literatura para este problema.

Para o problema flexível, a heurística desenvolvida também parte de uma solução inicial dada por regras de despacho, que tiveram que ser adaptadas para o ambiente flexível, e busca novas soluções através de uma vizinhança gerada pelos caminhos críticos das tarefas, guiada pela Busca Tabu. Esta busca de novas soluções foi dividida em dois níveis de resolução, onde em um resolve-se o problema de roteamento das operações nas máquinas e no outro, resolve-se o problema de programação das operações.

Foram feitos testes para avaliar o desempenho das regras neste ambiente e a regra MDD novamente foi a que obteve os melhores resultados. Os testes envolvendo a heurística para este problema foram feitos da seguinte maneira. Primeiro, foram feitos testes para ajustar os elementos de memória de curto prazo da BT para o subproblema de roteamento. Os elementos usados para o subproblema de programação foram os mesmos estabelecidos na heurística do problema tradicional. Destes resultados observou-se que a heurística (HFCP) obteve melhorias em problemas com pouco atraso, porém para problemas com bastante atraso a melhoria foi quase desprezível. A partir daí, tentando então melhorar o desempenho da

heurística, foram incorporadas estratégias de diversificação e intensificação. Destes resultados observou-se que a incorporação de uma estratégia de memória de longo prazo (HFLP) foi fundamental para todos os grupos, com um aumento significativo nas melhorias obtidas.

Visando determinar a importância do procedimento de resolução do subproblema de roteamento, foram feitos testes incorporando estratégias de intensificação e diversificação à heurística desenvolvida para o problema tradicional (HTLP), isto é, somente com o nível de resolução do problema de programação, e neste caso, um novo roteamento das operações é obtido através da diversificação ou intensificação. Os resultados mostraram que o procedimento para resolver o subproblema de roteamento da heurística do problema flexível é muito custoso computacionalmente, mas mesmo assim em alguns casos é compensador, pois apresenta resultados muito bons e com um tempo razoável. Aqui outra consideração importante deve ser feita. Notou-se que a vizinhança utilizada para resolver o problema de roteamento, que consiste em mudar uma operação de máquina, é muito restrita. Portanto, também aqui fica em aberto o desenvolvimento de uma vizinhança que consiga fazer mudanças mais radicais no roteiro das operações, porém sempre considerando que para isto é necessário elaborar testes de factibilidade eficientes.

De modo geral, este trabalho contribuiu com uma heurística eficiente para o problema de programação de tarefas em um *job shop* tradicional, e com duas implementações de BT para a resolução do problema no ambiente flexível, uma um pouco mais custosa computacionalmente que a outra.

Como extensões deste trabalho, pode-se citar:

- Elaboração de vizinhanças mais ricas para os problemas de roteamento e programação, mas que sejam embasadas por testes eficientes de factibilidade;

- Elaboração de vizinhanças reduzidas para ambos os problemas, visando obter a mesma qualidade de solução porém com um tempo computacional menor.

- Outras implementações para estratégias de diversificação e intensificação da BT baseadas em outras medidas de freqüências, por exemplo, relacionadas ao valor do atraso das tarefas.

- Implementação de outros elementos de BT, como oscilação estratégica e *path relinking*. Em termos gerais, oscilação estratégica opera orientando movimentos em relação a um limiar, chamado fronteira de oscilação. Neste trabalho um possível limiar é entre regiões factíveis e

infactíveis. Regras de seleção de movimentos são elaboradas para permitir o cruzamento dessas regiões. Porém a utilização de tal elemento fica novamente restrita ao problema da infactibilidade. *Path relinking* está relacionado à geração de novas soluções, explorando trajetórias que conectam diversas soluções de elite. Consiste basicamente em construir um caminho entre duas soluções incorporando características de uma em outra. A aplicação deste elemento neste problema deve sempre manter a factibilidade das soluções, e para isto se torna necessário o desenvolvimento de testes para garantir que somente sejam incorporadas características que preservem a factibilidade.

- Factibilização de uma solução infactível, para viabilizar a aplicação dos elementos e das vizinhanças mais ricas citados anteriormente. Uma proposta seria usar um procedimento do tipo regra de despacho para obter uma solução factível, porém tentando preservar ao máximo as características da solução infactível.

APÊNDICE A

COMPLEXIDADE DA HEURÍSTICA

Neste apêndice é apresentada a complexidade da heurística do problema tradicional (HCP). Os cálculos podem ser feitos diretamente dos pseudo-códigos a seguir. A análise da complexidade é feita separadamente para cada um dos procedimentos e depois é dada a complexidade geral da heurística. Para isto, considere que M é o número total de máquinas, J é o número total de tarefas, N é o número total de operações (N=J×M), J≥M, e G é um número grande.

Heurística (HCP)

```
iter=0
procedimento de obtenção da solução inicial
cálculo do atraso total
procedimento de determinação do caminho crítico das tarefas
enquanto (iter ≠ 250) faça
    para j=1 até J faça
        se T[j] >0 então procedimento de inversões no caminho crítico de j
        se executou uma inversão então
            procedimento de determinação do caminho crítico das tarefas
            cálculo do atraso total
            iter=iter+1
        fim
        atualização da solução incumbente
    fim
fim
```

Procedimento de obtenção da solução inicial

```
k=0; t=0; t[m]=0 para todo m
enquanto (k<N) faça
    para m=1 até M  faça  t=min{t[m]}
    para m=1 até M faça
        prior_atual=0;not=0;
        se (t[m]=t ou t[m]=G) então
            para n=1 até N faça
                se (n está disponível )
                    se (n pode ser processada em m) então
                        cálculo do valor da prioridade (prior)
                        da operação n
(a)             (b)     se (prior > prior_atual) então
                            prior_atual=prior; not=n;
                        fim
                    fim
                fim
            fim
            se (not ≠ 0) então k=k+1
        fim
    fim
fim
```

De acordo com a implementação feita, o cálculo do valor da prioridade de uma operação, por uma das quatro regras, tem complexidade O(J+M).

O pior caso para o bloco (b) é quando todas as operações que são processadas na máquina m estão disponíveis. Como cada máquina tem exatamente J operações, então o pior caso do laço (a) tem complexidade O((J+M)×J).

A complexidade deste procedimento é portanto O((J+M)×J×N×M), e como J×M=N e

J≥M, então a expressão reduz-se a $O(N \times J)$ ou $O(J^3 \times M^2)$.

Cálculo do atraso total

```
atraso total=0
para j=1 até J faça
    calcule o atraso de j de acordo com o instante de término de sua última
    operação e sua data de entrega
    atraso total=atraso total + atraso da tarefa j
fim
```

Este cálculo tem complexidade O(J).

Procedimento de determinação do caminho crítico das tarefas

A complexidade deste procedimento pode ser obtida diretamente do algoritmo que está apresentado no capítulo 3, e é O(N) ou O(J×M).

Procedimento de inversões no caminho crítico de uma tarefa j

```
op=última operação da tarefa j; y=op; trocou=verdadeiro;
decrescot=G; sol_tabu=G;
enquanto (trocou)
    x=operação predecessora de y no caminho crítico
    se (x ≠ 0) então
        se (x ≠ predecessora de tarefa de y) então avaliar inversão (x,y)
        senão y=x
    senão faça
        trocou=falso
        recupere melhor inversão
        insere inversão (x,y) na lista tabu
    fim
fim
```

Avaliar inversão (x,y)

```
guarde a solução atual S
ATR=atraso da solução atual
inverta (x,y) e atualize os predecessores e sucessores de máquina de x e y
procedimento de determinação do caminho crítico das tarefas
cálculo do atraso total (AT)
decresc=AT−ATR
se (inversão não for tabu ou  passar pelo critério de aspiração) então
|       se (decresc<decrescot) então atualize decrescot e armazene a inversão
senão faça
|       se (AT<sol_tabu) então atualize sol_tabu e armazene a inversão
fim
recupere a solução S
procedimento de determinação do caminho crítico das tarefas
y=x
```

A avaliação de uma inversão tem complexidade O(N) ou O(J×M). A complexidade do procedimento de inversões no caminho crítico de uma tarefa é então $O(N^2)$ ou $O(J^2 \times M^2)$.

Depois de calculadas as complexidades de todos os procedimentos, tem-se que a complexidade da heurística HCP é $O((N^2 \times J)+J+N+(J \times (N^2+N+J)))$, e portanto $\mathbf{O(N^2 \times J)}$ ou $\mathbf{O(J^3 \times M^2)}$.

APÊNDICE B

MÉTODOS HEURÍSTICOS - ALGORITMOS

Neste apêndice são apresentados os algoritmos, em passos, dos dois métodos heurísticos, MEHA e GSP, para o problema de *job shop* tradicional com datas de entrega, que foram usados para comparar com a heurística desenvolvida neste trabalho. Os algoritmos são descritos da mesma maneira que nos artigos (He et al., 1993; Raman e Talbot, 1993), e maiores detalhes podem ser encontrados nos mesmos.

Antes da apresentação, considere a seguinte notação:

J - número total de tarefas;

M - número total de máquinas;

atraso atual - valor do atraso total das tarefas para a solução atual;

atraso incumbente - melhor valor para o atraso total das tarefas encontrado até o momento;

atraso inicial - valor do atraso total das tarefas da solução inicial;

PJ[i] - a operação predecessora de tarefa da operação i;

PM[i] - operação predecessora de máquina da operação i;

SJ[i] - operação sucessora da tarefa da operação i;

SM[i] - operação sucessora de máquina da operação i;

ii[i] - instante de início da operação i;

f[i] - instante de término da operação i;

NI - denota que a operação não existe;

d_j - data de entrega da tarefa j;

d_{ij} - data de entrega da operação i da tarefa j;

t_{ij} - tempo de processamento da operação i da tarefa j;

g_j - última operação da tarefa j;

NT - contador de tarefas;

NM - contador de máquinas.

Algoritmo MEHA

Passo 1: Construa a solução inicial usando alguma regra de despacho. Calcule o atraso de cada tarefa e ordene-as na ordem decrescente dos valores. Faça NT=0; inicialize o atraso incumbente e o atraso atual usando o atraso inicial.

Passo 2: Se não existir nenhuma tarefa atrasada, PARE.

Passo 3: Para as tarefas atrasadas, marque as operações i tal que:

a) se i é a operação inicial da tarefa e ii[i] >0.

b) se i é outra operação da tarefa e ii[i] > f[i−1].

Passo 4: a) Se NT=J, PARE.

b) Escolha a próxima tarefa j de acordo com o ordenamento. Faça NT=NT+1;

1) Dentre as operações da tarefa j, escolha a que está marcada e que possui o maior instante de término - operação_alvo.

2) Se não existir nenhuma operação_alvo, volte ao passo 4 a).

Passo 5: Se PJ[operação_alvo]=0 então LHB=0.

Senão LHB=f[PJ[operação_alvo]].

Passo 6: Se SJ[operação_alvo]=NI ou f[operação_alvo]=ii[SJ[operação_alvo]] ou f[operação_alvo]=ii[SM[operação_alvo]], faça **RSMO**.

Passo 7: Se RSMO foi bem sucedido, faça **LSMO**.

Senão, volte ao passo 4.

Passo 8: Calcule o atraso de cada tarefa e o atraso atual.

Se atraso atual > atraso incumbente, recupere a solução incumbente e volte ao passo 4.

Senão, atualize o atraso e a solução incumbente, ordene decrescentemente o atraso das tarefas, faça NT =0 e volte ao passo 2.

RSMO: este procedimento tenta empurrar o máximo possível para frente, as operações k que estão na mesma máquina da operação_alvo e que possuem ii[k] ≥ LHB. Tenta-se fazer isso para uma operação de cada vez, e só é permitido empurrar tal operação para frente se isto não acarretar um aumento no atraso de sua tarefa. O procedimento é feito na ordem decrescente dos instantes de término das operações. Esta tentativa é feita com o objetivo de deixar um espaço tal que a operação_alvo possa ser inserida no mesmo. RSMO é bem sucedido se a operação_alvo consegue ser inserida em alguma posição para a qual seu instante de início seja menor que antes. Durante este procedimento a operação_alvo é temporariamente retirada da programação.

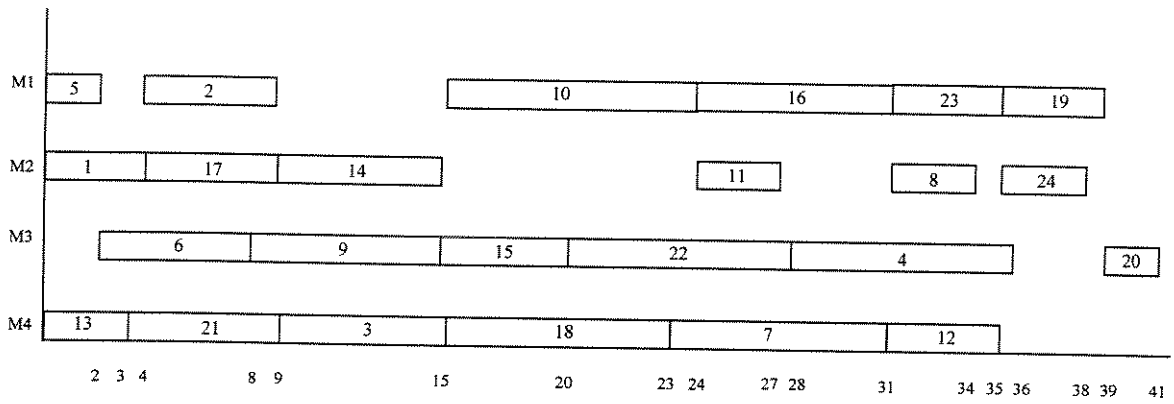
LSMO: procedimento que tenta empurrar para trás as operações k que possuem ii[k]≥LHB, a fim de iniciá-las o mais próximo possível do final de sua predecessora de tarefa ou de máquina, ou seja, fazer com que ii[k]=max{f[PJ[k]],f[PM[k]]}.

O exemplo a seguir, obtido de (He et al., 1993), ilustra uma iteração da heurística. Considere um problema com 6 tarefas e 4 máquinas e cada tarefa com 4 operações sendo que 1, 2, 3, 4 representam as operações da tarefa 1; 5, 6, 7, 8 as operações da tarefa 2; 9, 10, 11, 12 as operações da tarefa 3; 13, 14, 15, 16 as operações da tarefa 4; 17, 18, 19, 20 as operações da tarefa 5; 21, 22, 23, 24 as operações da tarefa 6. A tabela a seguir apresenta os tempos de processamento das operações e a data de entrega das tarefas.

Tabela B.1. Tempo de processamento das operações e data de entrega das tarefas

tarefa	operação	tempo de processamento	data de entrega
1	1	4	35
	2	5	
	3	6	
	4	8	
2	5	2	32
	6	6	
	7	8	
	8	3	
3	9	7	36
	10	9	
	11	3	
	12	4	
4	13	3	33
	14	6	
	15	5	
	16	7	
5	17	5	38
	18	8	
	19	4	
	20	2	
6	21	6	42
	22	8	
	23	4	
	24	3	

O roteiro e a programação das operações nas máquinas pode ser visto pelo gráfico de Gantt, que mostra a solução inicial gerada pela regra SPT.



Passo 1 - o atraso total da solução inicial é de 5 unidades, estando atrasadas as tarefas 1 (1 unidade), 2 (2 unidades) e 5 (3 unidades). Então, o ordenamento das tarefas atrasadas é 5, 2, 1, e o atraso incumbente e atraso atual são inicializados em 5.

Passo 2 - como existem tarefas atrasadas, a heurística não pára.

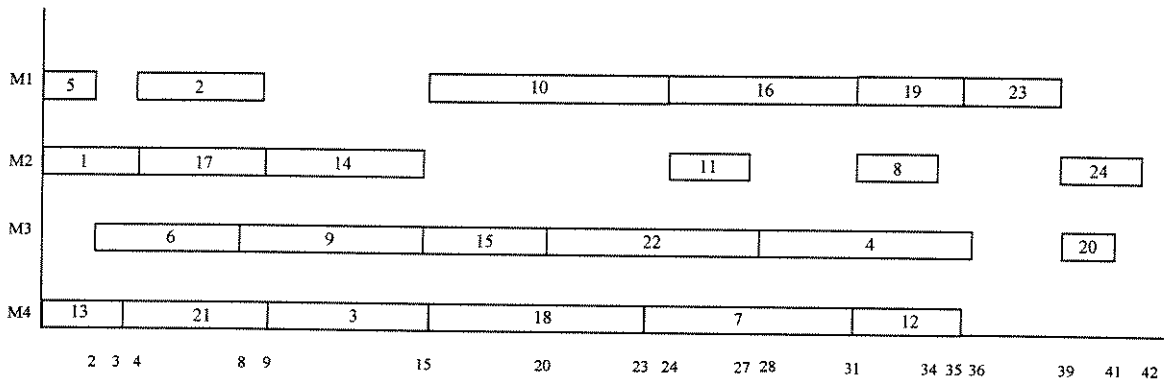
Passo 3 - são marcadas as operações 4 (da tarefa1), 7 (da tarefa 2), 17, 18, 19 (da tarefa 5).

Passo 4 - operação_alvo=19.

Passo 5 - LHB=f[PJ[19]]=f[18]=23.

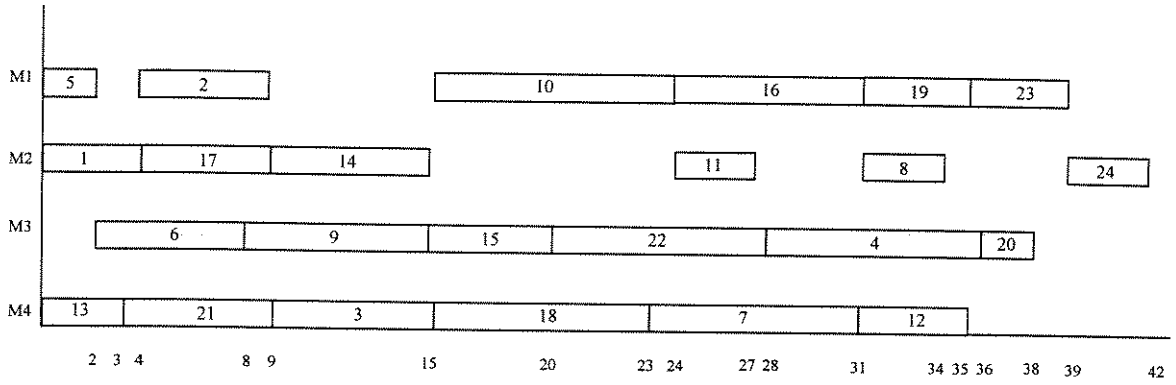
Passo 6 - como f[19]=ii[SJ[19]] → RSMO.

RSMO: as operações que estão em M1 e que possuem ii ≥ LHB são as operações 16 e 23. Inicia-se o procedimento com a operação 23, tentando empurrá-la para frente sem causar um atraso em sua tarefa. Como a operação 23 pertence à tarefa 6 que não está atrasada, então ela é empurrada para iniciar no instante 35. Note que isso só é possível, pois a operação 24 também é empurrada para iniciar no instante 39. Como a operação 19 tem tempo de processamento igual a 4 ela "cabe" exatamente no espaço aberto, iniciando no instante 31. Esta nova programação é guardada. Agora, tenta-se empurrar para frente a operação 16. Como esta operação pertence a tarefa 4 que não está atrasada, então ela é empurrada para iniciar no instante 26. A operação_alvo não cabe no espaço aberto pela operação 16. Depois de avaliar as possibilidades, recupera-se a programação guardada, onde a operação 19 inicia no instante 31 e a operação 23, no instante 35. O gráfico de Gantt abaixo ilustra a solução de RSMO.



Passo 7 - como RSMO foi bem sucedido → LSMO.

LSMO: tenta empurrar para trás as operações 7, 12, 4, 20, 11, 8, 24, 16, 19 e 23. A única operação que é empurrada para trás é a operação 20, que inicia no instante 36. O gráfico de Gantt a seguir mostra a solução obtida de LSMO.



Passo 8 - O atraso atual é de 3 unidades, estando atrasadas as tarefas 1 e 2. Como o atraso atual é menor que o atraso incumbente, então a solução incumbente é atualizada e volta-se ao Passo 2.

Algoritmo GSP

Passo 1: Construa a solução inicial usando a regra MOD, com as datas de entrega d_{kj} das operações dadas por $d_{kj} = d_j - \sum_{i=k+1}^{g_j} t_{ij}$.

Passo 2: Ordene as máquinas decrescentemente de acordo com seu carregamento. Seja z(r) o valor do atraso total na iteração r. Faça r=0, z(r)=+∞, NM=0.

Passo 3: Se não existir nenhuma tarefa atrasada, PARE. Senão r=r+1;

Passo 4: Para a próxima máquina na ordem decrescente:

a) Ordene as tarefas em ordem decrescente de seus atrasos. Faça NM=NM+1 e NT=0.

Passo 5: Para a próxima tarefa na ordem, tome a operação s desta tarefa que é processada na máquina escolhida e calcule o intervalo possível para sua data de entrega (o cálculo do intervalo está mostrado abaixo).

a) Para cada valor inteiro x no intervalo, gere as datas de entrega das outras operações desta tarefa, de acordo com a **regra_cálculo**, execute a regra MOD novamente e guarde o atraso total para cada x.

b) Para o melhor valor de atraso total obtido, recupere o valor x e gere as datas de entrega das outras operações de acordo com esse valor. Atualize z(r) de acordo com o melhor valor de atraso obtido. Fixe a data de entrega da operação s no valor $x_s^* = x$. Faça NT=NT+1.

Passo 6: Se NT=J, vá ao passo 7.

Senão, volte ao passo 4 a) e ordene somente as tarefas que ainda não foram escolhidas.

Passo 7: Se NM=M, vá ao passo 8.

Senão, volte ao passo 4.

Passo 8: Se z(r) < z(r-1), volte ao passo 3.

Senão, PARE.

A figura a seguir ilustra de forma genérica o algoritmo. Cada nó da árvore representa uma solução completa.

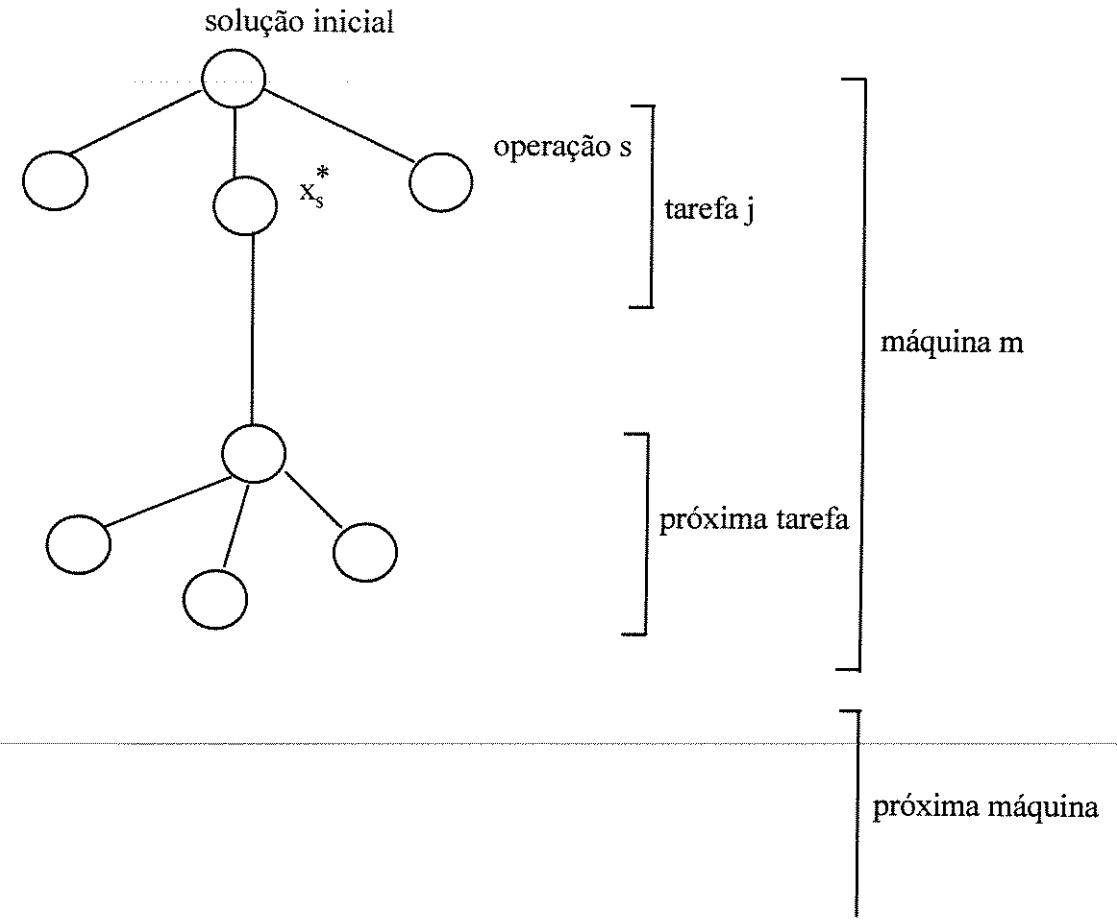


Figura B.1. Árvore de solução do GSP

O intervalo para a data de entrega de uma operação i_1 é $[L_1,U_1]$, dado por:

$$L_1=\sum_{i=1}^{i_1} t_{ij} \qquad \text{e} \qquad U_1=d_j\ .$$

Para cada x no intervalo, a data de entrega das outras operações da tarefa j são geradas de acordo com a seguinte regra:

$$\text{Regra\_cálculo}: \begin{cases} d_{kj} = d_{k-1,j} + (x - t_{i_1,j})\ .\ t_{kj} \ /\ \sum_{i=1}^{i_1-1} t_{ij} & k = 1,...i_1 - 1 \\ d_{kj} = d_{k-1,j} + (d_j - x)\ .\ t_{kj} \ /\ \sum_{i=i_1+1}^{g_j} t_{ij} & k = i_1 + 1,...,g_j \end{cases}$$

Suponha que durante o algoritmo, está sendo avaliado o ajuste da data de entrega da

operação k da tarefa j na máquina m. Suponha, também que depois de avaliar as máquinas anteriores a m no ordenamento, se encontram fixadas as datas de entrega das operações u_1, u_2, ...u_z da tarefa j. Seja k a operação processada entre u_l e u_{l+1} com datas de entrega fixadas em x_l^* e x_{l+1}^*, respectivamente. Então para a data de entrega da operação k, só é necessário considerar o intervalo $[\sum_{r=u_l+1}^{k} t_{rj} + x_l^* \ , \ x_{l+1}^* - t_{u_{l+1},j}]$. Além disso, só é necessário gerar as datas de entrega para as outras operações da tarefa que estejam entre u_l e u_{l+1}.

Considerando então a existência de um intervalo de ajuste para a data de entrega de uma operação, usou-se um procedimento de busca binária para o mesmo, que é explicado abaixo. Considere o intervalo inicial $[L_0,U_0]$. Primeiramente é calculado atraso(L_0) e atraso(U_0), onde atraso(d) é o valor do atraso total para x=d. Começando com o intervalo $[L_0,U_0]$, divide-se sucessivamente cada intervalo ao meio e calcula-se o atraso total no ponto médio. Para cada um dos subintervalos gerados, a busca neste pára quando o mesmo é sondado. Um subintervalo é sondado quando o atraso total de seu ponto médio é igual ao atraso total dos extremos do subintervalo. Dentro do intervalo $[L_0,U_0]$ a busca começa da esquerda para a direita, como mostra a figura a seguir.

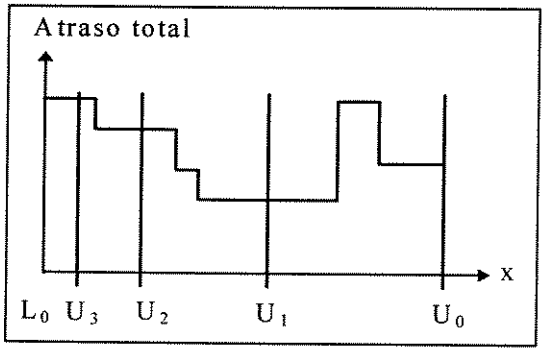


Figura B.2. - Procedimento de Busca

Pela Figura B.2 pode-se notar que o subintervalo $[L_0,U_3]$ é sondado. Depois disto, volta-se ao subintervalo anterior que é $[U_3,U_2]$. Vale ressaltar que este procedimento de busca binária nem sempre fornece o melhor valor de x.

BIBLIOGRAFIA

AARTS, E.H.L., VAN LAARHOVEN, P.J.M., LENSTRA, J.K. e ULDER, N.L.J., (1994), A Computational Study of Local Search Algorithms for Job Shop Scheduling, *ORSA Journal on Computing*, 6 (2): 118-125.

ADAMS, J., BALAS, E. e ZAWACK, D., (1988), The Shifting Bottleneck Procedure for Job Shop Scheduling, *Management Science*, 34 (3): 391-401.

AKERS, S.B., (1956), A Graphical Approach to Production Scheduling Problems, *Operations Research*, 3: 429-442.

ANDERSON, E.J. e NYIRENDA, J.C., (1990), Two New Rules to Minimize Tardiness in a Job Shop, *International Journal of Production Research*, 28 (12): 2277-2292.

APPLEGATE, D. e COOK, W., (1991), A Computational Study of the Job-Shop Scheduling Problem, ORSA *Journal on Computing*, 3 (2): 149-156.

BAKER, K.R., (1974), *Introducing to Sequencing and Scheduling*, John Wiley & Sons, Inc., New York.

BAKER, K.R., (1984), Sequencing Rules and Due-Date Assignments in a Job Shop, *Management Science*, 30: 1093-1104.

BAKER, K.R. e KANET, J.J., (1983), Job Shop Scheduling with Modified Due Dates, *Journal of Operations Management*, 4 (1): 11-22.

BALAS, E., (1969), Machine Sequencing via Disjunctive Graphs: An Implicit Enumeration Algorithm, *Operations Research*, 17 (6): 941-957.

BARNES, J.W. e CHAMBERS, J.B., (1995), Solving the Job Shop Scheduling Problem with Tabu Search, *IIE Transactions*, 27: 257-263.

BLACKSTONE, J.H., PHILLIPS, D.T. e HOGG, G.L., (1982), A State-of-the-Art Survey of Dispatching Rules for Manufacturing Job Shop Operations, *International Journal of Production Research*, 20 (1): 27-45.

BLAZEWICZ, J., DOMSCHKE, W. e PESCH, E., (1996), The Job Shop Scheduling Problem: Conventional and New Solutions Techniques, *European Journal of Operational Research*, 93: 1-33.

BRANDIMARTE, P., (1993), Routing and Scheduling in a Flexible Job Shop by Tabu Search, *Annals of Operations Research*, 41: 157-183.

BRANDIMARTE, P. e CALDERINI, M., (1995), A Hierarchical Bicriterion Approach to Integrated Process Plan Selection and Job Shop Scheduling, *International Journal of Production Research*, 33 (1): 161-181.

BRASSARD, G. e BRATLEY, P., (1988), *Algorithmics Theory and Practice*, Prentice-Hall International Editions, Great Britain.

BRUCKER, P., (1988), An Efficient Algorithm for the Job-Shop Problem with Two Jobs, *Computing*, 40: 353-359.

CARLIER, J., (1982), The One-Machine Sequencing Problem, *European Journal of Operational Research*, 11: 42-47.

CARLIER, J. e PINSON, E., (1989), An Algorithm for Solving the Job-Shop Problem, *Management Science*, 35 (2): 164-176.

CHAKRAPANI, J. e SKORIN-KAPOV, J., (1993), Massively Parallel Tabu Search for the Quadratic Assignment Problem, *Annals of Operations Research*, 41: 327-341.

CHANG, Y., MATSUO, H. e SULLIVAN, R.S., (1989), A Bottleneck-based Beam Search for Job Scheduling in a Flexible Manufacturing System, *International Journal of Production Research*, 27 (11): 1949-1961.

CHANG, Y. e SULLIVAN, R.S., (1990), Schedule Generation in a Dynamic Job Shop, *International Journal of Production Research*, 28 (1): 65-74.

DAGLI, C.H. e SITTISATHANCHAI, S., (1995), Genetic Neuro-Scheduler: A New Approach for Job Shop Scheduling, *International Journal of Production Economics*, 41: 135-145.

DAUZERE-PERES, S. e LASSERE, J.-B., (1993), A Modified Shifting Bottleneck Procedure for Job-Shop Scheduling, *International Journal of Production Research*, 31 (4): 923-932.

DELLA CROCE, F., TADEI, R. e VOLTA, G., (1995), A Genetic Algorithm for the Job Shop Problem, *Computers and Operations Research*, 22 (1): 15-24.

DELL'AMICO, M. e TRUBIAN, M., (1993), Applying Tabu Search to the Job-Shop Scheduling Problem, *Annals of Operations Research*, 41: 231-252.

DU, J. e LEUNG, J.Y.-T., (1990), Minimizing Total Tardiness on One Machine is *NP*-Hard, *Mathematics of Operations Research*, 15 (3): 483-495.

FISHER, H. e THOMPSON, G.L., (1963), Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules, em *Industrial Scheduling*, Muth, J.F. e Thompson, G.L. eds., Prentice-Hall, Englewood Cliffs, New Jersey.

GLOVER, F., (1989), Tabu Search - Part I, *ORSA Journal on Computing*, 1 (3): 190-206.

GLOVER, F., (1995), *Tabu Search Fundamentals and Uses*, Graduate School of Business, University of Colorado, working paper, University of Colorado, Boulder.

GLOVER, F., (1996), Tabu Search and Adaptive Memory Programming - Advances, Applications and Challenges, *Interfaces in Computer Science and Operations Research*, Barr, Helgason and Kennington eds., Kluwer Academic Publishers.

GLOVER, F. e LAGUNA, M., (1993), Tabu Search, capítulo do livro *Modern Heuristic Techniques for Combinatorial Problems*, C.R. Reeves ed., Blacwell, Londres.

GOICOECHEA, A., HANSEN, D.R. e DUCKSTEIN, L., (1982), *Multiobjective Decision Analysis with Engineering and Business Application*, John Wiley & Sons, New York.

GRABOWSKI, J., SKUBALSKA, E. e SMUTNICKI, C., (1983), On Flow Shop Scheduling with Release and Due Dates to Minimize Maximum Lateness, *Journal of the Operational Research Society*, 34 (7): 615-620.

HAN, M.H. e McGINNIS, L.F., (1989), Due Dates in a Manufacturing Shop An Unweighted Case, *Annals of Operations Research*, 17: 217-232.

HE, Z., YANG, T. e DEAL, D.E., (1993), A Multiple-Pass Heuristic Rule for Job Shop Scheduling with Due Dates, *International Journal of Production Research*, 31 (11): 2677-2692.

HEFETZ, N. e ADIRI, I., (1982), An Efficient Optimal Algorithm for the Two-Machine Unit-Time Jobshop Schedule-Length Problem, *Mathematics of Operations Research*, 7 (3): 354-360.

HOITOMT, D.J., LUH, P.B. e PATTIPATI, K.R., (1993), A Practical Approach to Job-Shop Scheduling Problems, *IEEE Transactions on Robotics and Automation*, 9 (1): 1-13.

HURINK, J, JURISCH, B. e THOLE, M., (1994), Tabu Search for the Job-Shop Scheduling Problem with Multi-Purpose Machines, *OR Spektrum*, 15: 205-215.

HUTCHISON J., LEONG, K., SNYDER, D. e WARD, P.,(1991), Scheduling Approaches for Random Job Shops Flexible Manufacturing Systems, *International Journal of Production Research*, 29 (5): 1053-1067.

JACKSON, J.R., (1956), An Extension of Johnson's Results on Job Lot Scheduling, *Naval Research Logistics Quarterly*, 3: 201-203.

KANET, J.J. e HAYYA, J.C., (1982), Priority Dispatching with Operation Due Dates in a Job Shop, *Journal of Operations Management*, 2 (3): 167-175.

KELLY, J.P., LAGUNA, M. e GLOVER, F., (1994) A Study of Diversification Strategies for the Quadratic Assignment Problem, *Computers and Operations Research*, 21 (8): 885-893.

KOULAMAS, C., (1994), The Total Tardiness Problem: Review and Extensions, *Operations Research*, 42 (6): 1025-1041.

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

LAWLER, E.L., LENSTRA, J.K., RINNOOY KAN, A.H.G. e SHMOYS, D.B., (1993), Sequencing and Scheduling: Algorithms and Complexity, em *Logistics of Production and Inventory*, S.C. Graves, A.H.G. Rinnoy Kan e P.H. Zipkin eds., North Holland.

LAWRENCE, S.R., (1984), *Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques*, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh.

LENSTRA, J.K. e RINNOOY KAN, A.H.G., (1979), Computational Complexity of Discrete Optimization Problems, *Annals of Discrete Mathematics*, 4: 121-140.

LENSTRA, J.K., RINNOOY KAN, A.H.G. e BRUCKER, P., (1977), Complexity of Machine Scheduling Problems, *Annals of Discrete Mathematics*, 1: 343-362.

LOGENDRAN, R. e SONTHINEN, A., (1997), A Tabu Search-based Approach for Scheduling Job-Shop type Flexible Manufacturing Systems, *Journal of the Operational Research Society*, 48: 264-277.

MONTAZERI, M. e VAN WASSENHOVE, L.N., (1990), Analysis of Scheduling Rules for an FMS, *International Journal of Production Research*, 28 (4): 785-802.

NASR, N. e ELSAYED, E.A., (1990), Job Shop Scheduling with Alternative Machines, *International Journal of Production Research*, 28 (9): 1595-1609.

NOWICKI, E. e SMUTNICKI, C., (1996), A Fast Taboo Search Algorithm for the Job Shop Problem, *Management Science*, 42 (6): 797-813.

RAGHU, T.S. e RAJENDRAN, C., (1993), An Efficient Dynamic Dispatching Rule for Scheduling in a Job Shop, *International Journal of Production Economics*, 32: 301-313.

RAMAN N. e TALBOT, F.B., (1993), The Job Shop Tardiness Problem: A decomposition Approach, *European Journal of Operational Research*, 69: 187-199.

SAWIK, T., (1990), Modelling and Scheduling of a Flexible Manufacturing System, *European Journal of Operational Research*, 45: 177-190.

SCRICH, C.R. e ARMENTANO, V.A., (1995), Busca Tabu para Job Shop com Datas de Entrega, *Anais do XXVII SBPO,* Vitória, ES, 06 a 08 de novembro; submetido a *Computers and Industrial Engineering* em julho de 1996.

SOTSKOV, Y.N., (1991), The Complexity of Shop-Scheduling Problems with Two or Three Jobs, *European Journal of Operational Research*, 53: 326-336.

STECKE, K., (1983), Formulation and Solution of Nonlinear Integer Production Planning Problems for Flexible Manufacturing Systems, *Management Science*, 29 (3): 273-288.

TAILLARD, E., (1990), Some Efficient Heuristic Methods for the Flow Shop Scheduling Sequencing Problem, *European Journal of Operational Research*, 47: 65-74.

TAILLARD, E.D., (1994), Parallel Taboo Search Techniques for the Job Shop Scheduling Problem, *ORSA Journal on Computing*, 6 (2):108-117.

VAESSENS, R.J.M., AARTS, E.H.L. e LENSTRA, J.K., (1996), Job Shop Scheduling by Local Search, *INFORMS Journal on Computing*, 8 (3): 302-317.

VAN LAARHOVEN, P.J.M., AARTS, E.H.L. e LENSTRA, J.K., (1992), Job Shop Scheduling by Simulated Annealing, *Operations Research*, 40 (1): 113-125.

VEPSALAINEN, A.P.J. e MORTON, T.E., (1987), Priority Rules for Job Shops with Weighted Tardiness Costs, *Management Science*, 33 (8): 1035-1047.

WILLIAMSON, D.P., HALL, L.A., HOOGEVEEN, J.A., HURKENS, C.A.J., LENSTRA, J.K., SEVAST'JANOV, S.V. e SHMOYS, D.B., Short Shop Schedules, *Operations Research*, a aparecer, *apud* Vaessens, R.J.M, Aarts, E.H.L. e Lenstra, J.K., (1996).