Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e de Computação

Métodos de Reinício Aplicados ao Seqüenciamento em Uma Máquina com Tempos de Preparação e Datas de Entrega

Aluno: **Luciano Marcelo Christofoletti**
Orientador: **Vinícius Amaral Armentano**

Dissertação apresentada à Faculdade de Engenharia Elétrica e de Computação, UNICAMP, como parte dos requisitos exigidos para a obtenção do título de Mestre em Engenharia Elétrica.

Área de Concentração: **Automação.**

Departamento de Engenharia de Sistemas
Abril de 2002

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

Christofoletti, Luciano Marcelo
C465m Métodos de reinício aplicados ao seqüenciamento em uma máquina com tempos de preparação e datas de entrega / Luciano Marcelo Christofoletti. -- Campinas, SP: [s.n.], 2002.

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

1. Memória. 2. Programação heurística. 3. Escalonamento de produção. 4. Controle de produção. 5. Pesquisa operacional. I. Armentano, Vinícius Amaral. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Resumo

Métodos de Múltiplos Reinícios envolvem a aplicação de um procedimento de busca local em diferentes soluções iniciais geradas por um processo construtivo. O procedimento clássico de Reinício Aleatório e a meta-heurística GRASP são exemplos deste tipo de abordagem. Estratégias de Memória Adaptativa, provindas da meta-heurística Busca Tabu, podem ser utilizadas neste tipo de método através do uso de informação das soluções encontradas ao fim da busca local. Uma população de soluções de elite dispersas é usada para guiar a fase construtiva visando a obtenção de soluções iniciais adequadas do ponto de vista de diversificação e intensificação da busca. Este trabalho trata da aplicação do método GRASP e variantes, e do método de reinício com uso de memória para o problema de minimizar o atraso total de tarefas em uma máquina com tempos de preparação dependentes da seqüência. Outro aspecto aqui desenvolvido consiste da estruturação da população em regiões de soluções de boa qualidade e distantes. Resultados computacionais revelam que a inclusão de estratégias de memória produzem um ganho significativo de qualidade.

Abstract

Multiple start methods involve the application of a local search procedure to start from distinct initial solutions generated by a constructive process. The classical Random Restart procedure and the metaheuristic GRASP are examples of such approaches. Adaptive memory strategies, which are used in the metaheuristic tabu search, can improve this kind of method by making use of information from the solutions found in the end of the local search. A population of solutions that have high quality and are dispersed is maintained in order to guide the constructive phase to generate suitable initial solutions in terms of diversification and intensification in the search space. This work deals with the application of the method GRASP and its variants, and a restart method with memory use to the problem of minimizing total tardiness of jobs in a single machine with sequence dependent setup times. Another aspect here developed consists of structuring the population in regions of the search space which have high quality solutions and are sufficiently distant from each other. Computational tests disclose that the inclusion of memory strategies yield a significative quality gain.

Dedicatória

Dedico este trabalho aos meus pais Ides e Expedito, e à minha esposa Flávia, que estiveram ao meu lado durante todas as etapas de desenvolvimento deste trabalho. Dedico este trabalho também às minhas irmãs Renata e Roberta que, embora estando um pouco longe, me deram o prazer de desfrutar de sua amizade e companherismo durante toda minha vida. Amo todos vocês.

Agradecimentos

Em primeiro lugar, meus sinceros agradecimentos ao Prof. Vinícius, pela amizade, dedicação e confiança que depositou em mim, desde o início do desenvolvimento deste trabalho.

Agradeço também aos colegas de departamento, em particular ao Alexandre S. Mendez, pelo fornecimento do código fonte de seu Algoritmo Memético, a Luciana S. Buriol, pela ajuda com os contatos internacionais, e ao Vinícius J. Garcia, pelas dicas de Java e LaTeX. Agradeço também ao Varvito, que me ajudou com alguns problemas de periféricos, e ao Edgar Noda, que me ajudou a montar uma configuração decente.

Um agradecimento especial também a todos aqueles que me incentivaram, direta ou indiretamente.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

O Problema de seqüenciamento de Tarefas em uma Máquina

Problemas de seqüenciamento em uma Máquina com tempos de preparação dependentes da seqüência (SUM) ocorrem com bastante freqüência em diversas áreas, dentre as quais podemos citar: sistemas de manufatura, sistemas de processamento de dados, e pousos de aeronaves em aeroportos. Geralmente, o seqüenciamento adequado das tarefas em problemas dessa natureza é considerado de vital importância para alcançar metas previstas associadas com datas de entrega e custo de operação. Neste capítulo é feita uma breve introdução ao problema de seqüenciamento em uma máquina e uma revisão de trabalhos importantes existentes na literatura que tratam especificamente deste problema.

1.1 Introdução

Problemas de seqüenciamento em uma máquina são importantes por diversas razões. Modelos de programação (*scheduling*) de tarefas em ambientes mais complexos envolvendo mais de uma máquina, como por exemplo, máquinas paralelas, flowshop e jobshop, podem ser resolvidos de forma mais rápida e eficiente através da decomposição do problema original em diversos subproblemas de uma máquina. Algoritmos ótimos e heurísticos para job shop e máquinas paralelas são exemplos que utilizam essa decomposição (Chen e Powell (1999); Adams *et al.* (1988); Carlier e Pinson (1989); Lee e Pinedo (1997)).

Um problema de seqüenciamento em uma máquina é definido pelas seguintes características das tarefas (*jobs*): consideração de datas de entrega (*due dates*), existência ou não de tempos de preparação (*setup times*), tempos de preparação dependentes ou independentes da seqüência, instantes de disponibilidade para processamento das tarefas (*release times*) e tempos de processamento das tarefas. Existe

uma situação adicional onde ocorrem famílias de tarefas em que os tempos de preparação entre as famílias se distinguem em relação aos tempos de preparação entre as tarefas da mesma família. Finalmente, uma característica essencial é a definição do critério de otimização. Como destacado por Vollmam *et al.* (1988), existem três objetivos básicos na programação de tarefas na produção. O primeiro objetivo está relacionado com datas de entrega das tarefas: basicamente, não se deseja atraso em relação a essas datas, e quando o custo de estoque é relevante, tenta-se evitar que as tarefas sejam finalizadas muito antes dessas datas. O segundo objetivo está relacionado com o tempo de fluxo de tarefas no chão de fábrica: deseja-se que esse tempo seja curto, ou equivalentemente, que o estoque em processamento seja baixo. O terceiro objetivo envolve a utilização dos centros de trabalho: deseja-se maximizar a utilização de equipamentos e de mão de obra, o que equivale a minimizar o tempo total de processamento das tarefas ou *makespan*.

Neste trabalho vamos abordar o problema de seqüenciamento em uma máquina com tempos de preparação dependentes da seqüência, tarefas disponíveis para processamento no instante $t = 0$ e datas de entrega pré-determinadas. Assumimos que tarefas processadas antes da data de entrega não incorrem em custo de estoque. O objetivo é minimizar a soma dos atrasos em relação às datas de entrega. Para tratar este problema vamos utilizar uma abordagem baseada em métodos de múltiplos reinícios (em particular o método GRASP desenvolvido por Feo e Resende (1989)).

Em problemas com apenas uma máquina, as soluções podem ser representadas através de seqüências de números inteiros (*seqüências de permutação*). Para trabalharmos com este tipo de representação podemos associar, sem perda de generalidade, um número inteiro a cada tarefa do conjunto de forma que cada solução do problema pode ser identificada por uma única permutação dessa seqüência. Dessa forma, dado um problema com n tarefas, podemos associar um conjunto de índices $J = \{1, 2, \ldots, n\}$ ao conjunto de tarefas e qualquer permutação deste conjunto representa uma solução factível para o problema. Como exemplo, para um problema com 10 tarefas a seqüência $s = (3, 8, 4, 5, 1, 9, 2, 10, 6, 7)$ representa uma solução factível onde a tarefa 3 é a primeira a ser processada, seguida da tarefa 8, e assim por diante.

Como é fácil verificar, dado um problema com n tarefas, o número de soluções ou permutações possíveis é $n!$. Du e Leung (1990) demonstraram que, mesmo na ausência de tempos de preparação dependentes da seqüência, o problema de minimizar o atraso total em uma máquina é NP-difícil. Os algoritmos ótimos para esta classe de problemas possuem uma complexidade computacional que é exponencial com o número de tarefas. Por este motivo, a utilização de heurísticas ou meta-heurísticas, torna-se uma alternativa bastante razoável.

Problemas de seqüenciamento com tempos de preparação dependentes da seqüência ocorrem com bastante freqüência em processos industriais, principalmente

em fábricas que operam com linhas de montagens que devem ser modificadas ou adaptadas com freqüência, dependendo do tipo de produto a ser processado (Allahverdi *et al.* (1999)). Como exemplo, uma fábrica de papel que produz diversos tipos de folhas com diferentes cores, espessuras e texturas realiza freqüentemente a preparação de certas máquinas para a obtenção dos diferentes tipos de produtos desejados. O tempo de preparação e os gastos variam de acordo com as atividades, tais como, limpeza, ajuste de máquina e troca de ferramentas. Em problemas como este, os tempos de preparação representam uma parcela de tempo considerável em relação ao tempo total de processamento, e portanto não podem ser negligenciados.

Datas de entrega também ocorrem com freqüência em problemas de seqüenciamento de tarefas. Neste caso, cada tarefa possui uma data de entrega prédeterminada e o processamento da tarefa, idealmente, deve terminar antes dessa data. Caso a tarefa não seja entregue até a data determinada podemos atribuir uma penalização proporcional ao atraso ocorrido. Também podemos atribuir pesos ao conjunto de tarefas de forma a penalizar fortemente o atraso de certas tarefas, consideradas prioritárias com relação a não ocorrência de atraso.

1.2 Descrição do problema

Neste trabalho vamos considerar o problema de seqüenciamento de tarefas em uma máquina com tempos de preparação dependentes da seqüência, com o objetivo de minimizar a soma total dos atrasos com relação às datas de entrega das tarefas. Seja $J = \{1, \ldots, n\}$ um conjunto de n índices (rótulos) associados às tarefas a serem processadas. Neste caso podemos definir o conjunto de dados de entrada do problema da seguinte maneira:

Conjunto de Dados (SUM):

- $P = \{p_1, \ldots, p_n\}$ vetor de tempos de processamento das tarefas.
- $D = \{d_1, \ldots, d_n\}$ vetor de datas de entrega das tarefas.
- $S = \{s_{ij}\}$ matriz de tempos de preparação (setup).
- $IS = \{is_1, \ldots, is_n\}$ vetor de tempos de preparação inicial.

Os vetores P e D fornecem respectivamente os tempos de processamento e datas de entrega das tarefas. A matriz S fornece o tempo de preparação que deve ocorrer entre o processamento de tarefas consecutivas na seqüência, onde o elemento s_{ij} representa o tempo necessário para preparar a produção da tarefa j após o término do processamento da tarefa i (note que s_{ij} não é necessariamente igual à s_{ji}). O vetor

IS fornece o tempo de preparação para a primeira tarefa processada na seqüência, e seus elementos são também denotados por s_{0j}, onde 0 representa uma tarefa fictícia. Os elementos dos vetores são, em geral, inteiros positivos.

Uma vez definidos os dados de entrada do problema, definimos a função objetivo da seguinte maneira:

Objetivo: Encontrar a seqüência de processamento das tarefas que minimiza a soma dos atrasos em relação as datas de entrega dada por

$$T = \sum_{j=1}^{n} \max(c_j - d_j, 0), \tag{1.1}$$

onde c_j é o instante de término de processamento da tarefa j.

Para o problema de seqüenciamento de tarefas definimos o instante de término de processamento da tarefa j como

$$c_j = \sum_{k=1}^{p(j)} s_{r_{k-1}r_k} + p_{r_k}, \tag{1.2}$$

onde r_k fornece o índice (rótulo) da k-ésima tarefa processada na seqüência e $p(j)$ é a posição da tarefa j na seqüência. Também definimos $r_0 = 0$, por motivos de consistência com a notação utilizada. Note que na equação (1.1) estamos assumindo que todas as tarefas possuem a mesma prioridade de processamento, e que não estamos considerando penalizações para tarefas adiantadas em relação às datas de entrega. Também vamos assumir que todas as tarefas estão disponíveis para processamento no instante $t = 0$.

A seguir mostramos um exemplo de conjunto de dados numéricos para um problema com cinco tarefas.

tarefas	1	2	3	4	5
p_j	3	3	4	6	2
d_j	13	4	25	16	22
is_j	3	2	3	1	5

A matriz de tempos de preparação para o problema é mostrada a seguir.

tarefas	1	2	3	4	5
s_{1j}	-	3	2	2	4
s_{2j}	2	-	4	2	1
s_{3j}	4	1	-	5	6
s_{4j}	2	3	3	-	6
s_{5j}	2	3	1	5	-

A seguir, na Figura 1.1, mostramos uma representação gráfica da solução ótima para o conjunto de dados apresentado. Para este exemplo notamos que a solução ótima, obtida por enumeração completa, possui um atraso total igual a 9 unidades de tempo, onde as tarefas 2, 3 e 4 possuem atrasos respectivamente iguais a 1, 3 e 5. Neste caso a seqüência ótima é dada por $s^* = (2, 5, 1, 4, 3)$.

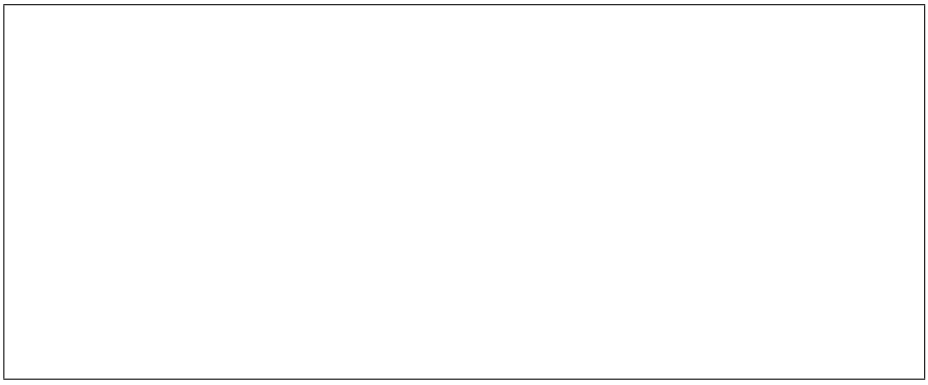

Figura 1.1: Esquema de representação gráfica para o problema de SUM.

1.3 Revisão bibliográfica

Embora a ocorrência de problemas de seqüenciamento de tarefas com tempos de preparação dependentes da seqüência seja bastante comum em diversos problemas de programação de tarefas, a sua abordagem científica só começou a receber maior atenção recentemente. A maioria dos trabalhos publicados que tratam especificamente deste problema realizam simplificações do modelo real simplesmente desconsiderando os tempos de preparação, ou ainda, considerando-os como parte do tempo de processamento. Esta abordagem simplifica bastante o modelo, porém a sua aplicação prática pode comprometer a qualidade final das soluções obtidas em relação a solução ótima do problema original.

Dentre os trabalhos pioneiros que tratam do problema aqui abordado, podemos citar o método *branch-and-bound* proposto por Ragatz (1993), o algoritmo de busca genética (BG) proposto por Rubin e Ragatz (1995) e a abordagem heurística utilizando Simulated Annealing (SA) proposta por Tan e Narasimhan (1997). Ainda em seu trabalho de 1995, Rubin e Ragatz comparam o desempenho da busca genética em relação ao método de reinício aleatório (RA) com vizinhança de trocas de tarefas adjacentes. Também em 1997, Lee *et al.* (1997) propõem uma regra de despacho para o problema de seqüenciamento de tarefas com tempos de preparação dependentes da seqüência. Entre os trabalhos mais recentes, baseados em métodos populacionais, podemos citar o Algoritmo Genético proposto por Armentano e Mazzini (2000) e

o Algoritmo de busca genética (também conhecido como algoritmo memético) proposto por França *et al.* (2001).

Ragatz (1993) desenvolve um limitante inferior e testes de dominância eficientes para utilização em um algoritmo branch-and-bound. O método foi aplicado a um conjunto de 160 problemas testes com tamanhos variando entre 8, 10, 12, 14 e 16 tarefas. O autor reconhece a dificuldade do método em resolver problemas de grande porte, principalmente aqueles com pequena dispersão das datas de entrega e/ou uma alta variância dos tempos de processamento.

Em seu trabalho de 1995, Rubin e Ragatz desenvolveram um novo operador de crossover específico para problemas de seqüenciamento onde a posição dos genes (tarefas) do cromossomo, assim como a ordem relativa em que eles aparecem, é importante. Os resultados obtidos com a BG foram comparados com os resultados obtidos pelo branch-and-bound e reinício aleatório, e os autores concluíram que o método de múltiplos reinícios teve um desempenho ligeiramente superior à busca genética na maioria dos casos, considerando o tempo total de execução e a qualidade das soluções obtidas. Vale destacar que a implementação do método de múltiplos reinícios utilizada para comparação é bastante rudimentar e não necessita de nenhuma calibragem de parâmetros. Neste trabalho os autores utilizaram um conjunto teste composto de 32 problemas, gerados por eles mesmos de acordo com três características consideradas relevantes: variância dos tempos de processamento das tarefas (alta e baixa), fator de atraso das tarefas (baixo e moderado) e distribuição das datas de entrega (ampla e reduzida). Este conjunto de características fornece um total de $2 \times 2 \times 2 = 8$ combinações possíveis para cada tamanho de problema gerado: 15, 25, 35 e 45 tarefas.

Tan e Narasimhan (1997) propuseram um método baseado em simulated annealing para o problema em questão. A implementação apresentada é bastante simples e a calibragem de parâmetros é bem rápida. Apesar de sua aparente simplicidade, os autores concluíram que o método proposto é bastante viável quando comparado aos demais métodos existentes na literatura. Porém, vale ressaltar que, assim como no trabalho de Rubin e Ragatz (1995), os autores utilizaram apenas problemas de tamanho reduzido (10, 30 e 50 tarefas)[1]. A conclusão dos autores neste trabalho é que o método proposto supera o reinício aleatório em quase todos os problemas-testes utilizados, embora o ganho máximo do simulated annealing em relação ao reinício aleatório não tenha sido superior a 6%.

Em um trabalho recente, Tan *et al.* (2000) realizam uma comparação entre as quatro abordagens propostas: branch-and-bound, busca genética, simulated annealing e reinício aleatório. Eles concluíram que os métodos de reinício aleatório e

[1]Neste trabalho parte das análises apresentadas foram feitas utilizando-se apenas os problemas com 10 tarefas.

simulated annealing superam os outros dois em quase todos os casos. Eles utilizaram o mesmo conjunto de 32 instâncias proposto por Rubin e Ragatz (1995).

Armentano e Mazzini (2000) propõem um algoritmo genético que inclui testes estatísticos para a calibração dos parâmetros. Neste trabalho os autores utilizaram três conjuntos de dados para teste: o conjunto de 32 problemas proposto por Rubin e Ragatz (1995), um conjunto de 90 problemas com tamanhos variando entre 6, 8 e 10 tarefas, utilizado para comparar os resultados obtidos pelo AG com as soluções ótimas (ou próximas à otimalidade) obtidas pelo software comercial CPLEX, e finalmente um conjunto de 150 problemas com tamanhos variando entre 30, 50, 70, 90 e 110 tarefas. Aqui, assim como no trabalho de Rubin e Ragatz (1995), os autores utilizaram uma série de combinações de parâmetros considerados relevantes para a geração dos problemas-testes. Os resultados do AG obtidos com o conjunto de dados proposto por Rubin e Ragatz são comparados com os obtidos pela regra ATCS, e também com os resultados obtidos por Rubin e Ragatz (1995), onde os autores concluíram que o AG proposto teve um desempenho semelhante à busca genética.

Outro trabalho recente, proposto por França *et al.* (2001), também apresenta uma abordagem populacional para o problema de seqüenciamento de tarefas. Neste trabalho os autores propõem um algoritmo memético onde as características marcantes são a utilização de uma população estruturada e a avaliação de três tipos diferentes de operadores de recombinação. A população é hierarquicamente estruturada segundo uma árvore ternária. A população é dividida em clusters e o operador de recombinação é aplicado em cada cluster. Os resultados computacionais apresentados indicam um aumento considerável de desempenho em relação a outros métodos populacionais que não utilizam nenhum tipo de estrutura de população. Neste trabalho os autores utilizaram um conjunto de dados gerados por eles mesmos, com problemas variando entre 20, 40, 60, 80 e 100 tarefas, além do conjunto de dados utilizado por Rubin e Ragatz (1995). As regras utilizadas para gerar os problemas seguem o mesmo estilo utilizado por Lee *et al.* (1997) e Tan e Narasimhan (1997).

Outro método proposto recentemente, baseado em busca local com perturbação, é o algoritmo de busca local de Carvalho (2002). Este trabalho, ainda em fase de conclusão, apresenta um método de busca local, baseado em movimentos *3-opt* com exploração completa de vizinhança, o qual é reiniciado após a obtenção de um mínimo local. O procedimento de reinício funciona através da aplicação de um movimento de perturbação na solução incumbente, obtida durante o processo. Este movimento de perturbação realiza uma troca aleatória de dois seguimentos de tarefas (movimento *4-opt*), gerando uma nova solução, a qual é utilizada como ponto de partida para a realização de uma nova busca local.

Neste trabalho tratamos do problema de seqüenciamento de tarefas em uma única máquina onde propomos uma abordagem baseada em métodos de múltiplos

reinícios. O conteúdo do trabalho foi dividido da seguinte maneira: no Capítulo 2 apresentamos as principais propostas de heurísticas existentes na literatura desenvolvidas especificamente para o problema em questão. No Capítulo 3 descrevemos os fundamentos do método GRASP e algumas extensões do mesmo. No Capítulo 4 são desenvolvidas algumas propostas para inclusão de memória no método GRASP. Finalmente, no Capítulo 5 apresentamos alguns resultados computacionais obtidos a partir de quatro versões diferentes da abordagem proposta.

Capítulo 2

Heurísticas Construtivas

A utilização de heurísticas construtivas para obtenção de soluções aproximadas em problemas combinatoriais é uma prática bastante comum, motivada em geral, pelo baixo custo computacional envolvido e pela facilidade de implementação. Este tipo de heurística é de fundamental importância em problemas de programação de tarefas, onde em geral, o horizonte de tomada de decisão é muito curto. Além disso, soluções geradas por heurísticas construtivas são utilizadas como ponto de partida para heurísticas de busca. Para o problema de seqüenciamento de tarefas com tempos de preparação dependentes da seqüência uma boa alternativa é a utilização da regra ATCS. Esta regra foi proposta por Lee *et al.* em 1997 como uma extensão da heurística ATC-modificada de Raman *et al.* de 1989 que é uma generalização da regra ATC de Vepsalainen e Morton (1987) para problemas de seqüenciamento de tarefas que não envolvem tempos de preparação dependentes da seqüência.

2.1 Introdução

Uma categoria de heurísticas construtivas muito usada em programação de tarefas consiste das *regras de despacho*. Uma regra de despacho é um procedimento que seleciona a próxima tarefa a ser processada a partir de uma lista ordenada de tarefas ainda não processadas, segundo um criterio de prioridade. A aplicação repetida da regra de despacho ao conjunto de tarefas resulta em uma solução completa.

Regras de despacho, em geral, pertencem a uma categoria de heurísticas comumente denominadas por *heurísticas gulosas*. Estes procedimentos, salvo raras exceções, não fornecem soluções ótimas ao final de sua execução, de modo que a utilização de procedimentos de melhoria para o pós-processamento das soluções obtidas torna-se bastante recomendada.

Para o problema de seqüenciamento de tarefas abordado neste trabalho, foram propostas na literatura as regras de despacho ATCS e ATC-modificada. A seguir,

descrevemos a regra de despacho ATC, a qual representa um exemplo típico de heurística gulosa e serve como ponto de partida para a introdução das demais regras citadas anteriormente.

2.2 A regra ATC

A regra ATC (*Apparent Tardiness Cost*) foi proposta em 1987 por Vepsalainen e Morton para problemas de seqüenciamento de tarefas que não consideram tempos de preparação dependentes da seqüência. Embora esta regra não tenha sido desenvolvida especificamente para o problema abordado neste trabalho, a sua apresentação serve como ponto de partida para a introdução da regra ATC-modificada e, posteriormente, da regra ATCS. O funcionamento desta regra é baseado no cálculo do índice de prioridade para cada tarefa ainda não processada, dado pela expressão

$$I_j(t) = \frac{w_j}{p_j} \cdot \exp\left(-\frac{\max(d_j - p_j - t, 0)}{k\bar{p}}\right), \tag{2.1}$$

onde p_j e d_j são, respectivamente, o tempo de processamento e a data de entrega da tarefa j, t é o instante de tempo de término do processamento da última tarefa processada e w_j é a penalidade por unidade de tempo de atraso da tarefa j (neste trabalho vamos considerar um caso particular onde $w_j = 1$ para todas as tarefas). O parâmetro k é um parâmetro de escala (também chamado de *look-ahead*) obtido de forma empírica de acordo com os dados do problema e $\bar{p}$ é a média dos tempos de processamento das tarefas ainda não processadas. Note que o termo exponencial na equação 2.1 prioriza a escolha das tarefas mais próximas de suas respectivas datas de entrega.

A execução do processo de seqüenciamento é bastante simples: Inicialmente o contador de tempo t é colocado em zero. Em seguida ocorre a aplicação da regra de despacho, onde são calculados os índices de prioridade dados pela equação (2.1) para todas as tarefas disponíveis para processamento. Após estes cálculos a tarefa com o maior índice é selecionada para ser a primeira a ser processada. Em seguida ocorrem as atualizações do contador de tempo t, que recebe o instante de término de processamento da tarefa selecionada, do valor médio dos tempos de processamento $\bar{p}$ e do conjunto de tarefas disponíveis para processamento. Após estas atualizações os índices $I_j(t)$ são recalculados para todas as tarefas disponíveis e novamente a tarefa com o maior índice é selecionada para ser processada. Este processo se repete até que o conjunto de tarefas disponíveis fique vazio.

Para o bom funcionamento da regra o parâmetro k deve ser ajustado de forma apropriada, possivelmente em função do problema-teste utilizado. O ajuste deste parâmetro, em geral, é feito através de conhecimentos empíricos, mas pode ser feito

de forma automática através de uma análise estatística inicial dos dados do problema teste.

Existem diversos fatores que podem ser utilizados para ajudar na caracterização dos problemas, como por exemplo os fatores de *aperto* e *dispersão* das datas de entrega. O fator de aperto τ é definido por

$$\tau = 1 - \frac{\bar{d}}{C_{max}},$$

onde $\bar{d}$ é a média das datas de entrega e C_{max} é o tempo necessário para o processamento de todas as tarefas do problema teste (*makespan*). Valores de τ próximos de 1 indicam que as datas de entrega são apertadas, ou seja, grande parte das tarefas possivelmente será entregue com atraso, enquanto que valores de τ próximos de 0 indicam datas de entrega com folga. O fator de dispersão das datas de entrega R é definido como

$$R = \frac{d_{max} - d_{min}}{C_{max}}, \tag{2.2}$$

onde d_{max} e d_{min}, representam a data de entrega máxima e mínima, respectivamente. Valores altos para R indicam uma alta dispersão das datas de entrega, enquanto que valores próximos de zero indicam datas de entrega bem próximas umas das outras.

Vepsalainen e Morton (1987) sugerem que valores apropriados para k podem ser obtidos através de experimentos computacionais. Neste caso podemos utilizar uma série de problemas-testes, gerados de forma estratégica para obter combinações de valores τ e R consideradas relevantes, para ajustar o parâmetro k de acordo com uma função do tipo $k = f(\tau, R)$. Para esta regra os autores sugerem que, em geral, o valor de k deve ficar restrito ao intervalo $[1, 3]$.

2.3 A regra ATC-modificada

A regra ATC-modificada foi proposta por Raman *et al.* (1989) para incluir os tempos de preparação no cálculo dos índices de prioridade. Neste caso, o índice de prioridade de cada tarefa j é calculado não apenas em função de t, mas também em função da última tarefa escolhida l. A inclusão deste fator no cálculo dos índices de prioridade é importante pois, em geral, desejamos que os tempos de preparação entre as tarefas sejam mínimos. A equação (2.3) fornece os valores dos índices de prioridade para o problema com tempos de preparação dependentes da seqüência.

$$I_j(t, l) = \frac{w_j}{p_j + s_{lj}} \cdot \exp\left(-\frac{\max(d_j - p_j - s_{lj} - t, 0)}{k\bar{p}}\right) \tag{2.3}$$

Os parâmetros p_j, d_j, $\bar{p}$ e k são os mesmos utilizados pela regra ATC, porém temos agora a influência dos valores dos tempos de preparação s_{lj} no cálculo dos

índices. Neste caso, l representa a última tarefa processada e o valor s_{lj} é o tempo de preparação que irá ocorrer, caso a tarefa j seja escolhida para ser processada após a tarefa l. O processo construtivo funciona basicamente da mesma forma que para a regra ATC, porém devemos tomar o cuidado de atualizar o índice l cada vez que uma tarefa é selecionada. Note que para a escolha da primeira tarefa a ser processada fazemos $l = 0$, de forma que s_{0j} representa o tempo de preparação inicial, ou seja, $s_{0j} = is_j$.

Em seu trabalho sobre o estudo comparativo das regras ATCS e ATC-modificada, Lee *et al.* (1997) afirmam que Raman *et al.* (1989) não fornecem regra alguma para a determinação dos possíveis valores de k. Para efeito de uma comparação justa eles realizam uma série de experimentos computacionais para ajustar o parâmetro k em função dos fatores τ e R citados anteriormente. Eles sugerem ainda a utilização de uma terceira estatística para o cálculo de k, chamado *fator de severidade* dos tempos de preparação, definido por

$$\eta = \frac{\bar{s}}{\bar{p}},$$

onde $\bar{s}$ é a média dos tempos de preparação. Em seu artigo sobre o estudo comparativo das heurísticas ATC-modificada e ATCS eles sugerem a utilização da seguinte regra para o cálculo de k, obtida através de um ajuste de curva baseado em experimentos computacionais feitos a partir de um conjunto de problemas-testes devidamente elaborado:

$$k = 5,5 - \tau - R + \eta. \tag{2.4}$$

Um ponto importante a ser notado é que para o problema com tempos de preparação dependentes da seqüência o valor de C_{max} varia de acordo com seqüenciamento das tarefas. Dessa forma devemos lançar mão de um estimador $\hat{C}_{max}$ para o cálculo dos fatores τ e R. Uma estimativa simples do makespan para problemas de seqüenciamento em uma máquina é dada por

$$\hat{C}_{max} = \sum_{j=1}^{n} p_j + n\bar{s}. \tag{2.5}$$

Esta estimativa geralmente é um pouco maior que o valor real do makespan, dado que o seqüenciamento final tende a tomar vantagem dos tempos de preparação que são menores que a média. Para contornar este inconveniente, podemos utilizar uma medida da variabilidade dos tempos de preparação para determinar um fator de redução β que modifica o estimador da seguinte maneira:

$$\hat{C}_{max} = \sum_{j=1}^{n} p_j + n\beta\bar{s}, \qquad \text{com } \beta \leq 1. \tag{2.6}$$

De acordo com Lee *et al.* (1997) o valor de β pode ser calculado de forma empírica em função do coeficiente de variação dos tempos de preparação C_v, que é dado por

$$C_v = \frac{Var(s)}{\bar{s}^2}, \tag{2.7}$$

onde $Var(s)$ é a variância dos tempos de preparação. Com base em experimentos computacionais eles obtiveram a seguinte expressão para o cálculo de β

$$\beta = -1,88c_v + 0,92.$$

Com esta modificação conseguimos obter uma estimativa mais precisa para o valor de C_{max} e, por conseqüência, uma precisão maior também no cálculo de τ e R.

2.4 A regra ATCS

Como dito anteriormente a regra ATCS nada mais é que um aprimoramento da regra ATC-modificada. A modificação mais significativa ocorre no cálculo dos índices de prioridade, que consideram os tempos de preparação e as folgas das tarefas em relação às datas de entrega de forma desacoplada através do produto de duas funções exponenciais. A equação (2.8) fornece o valor dos índices de prioridade dados pela regra ATCS.

$$I_j(t,l) = \frac{w_j}{p_j} \cdot \exp\left(-\frac{\max(d_j - p_j - t, 0)}{k_1\bar{p}}\right) \cdot \exp\left(-\frac{s_{lj}}{k_2\bar{s}}\right). \tag{2.8}$$

Os parâmetros p_j, d_j, s_{lj}, $\bar{p}$ e $\bar{s}$ são os mesmos utilizados anteriormente, porém ao invés de um único parâmetro de escala k, temos agora dois novos parâmetros, k_1 e k_2. Neste caso a utilização de dois parâmetros de escala é necessaria para que possamos considerar as folgas de processamento e os tempos de preparação de forma desacoplada, ao contrário do que ocorre com a regra ATC-modificada. Como podemos notar a primeira exponencial prioriza as tarefas que estão mais próximas das respectivas datas de entrega (tarefas com pouca folga), enquanto que a segunda prioriza as tarefas que possuem um baixo valor de tempo de preparação, em relação à média dos tempos de preparação restantes $\bar{s}$.

De acordo com o trabalho de Lee *et al.* os parâmetros de escala k_1 e k_2 podem ser calculados em função dos três fatores τ, R e η citados anteriormente. Estudos experimentais da regra ATCS, embora inconclusivos, sugerem algumas regras para o cálculo destes parâmetros. Para o cálculo do parâmetro k_1 a seguinte regra pode ser utilizada:

$$\begin{aligned} k_1 &= 4,5 + R, \qquad \text{se } R \leq 0,5 \\ k_1 &= 6 - 2R, \qquad \text{se } R \geq 0,5 \end{aligned}$$

Note que os valores de τ e η não são considerados para o cálculo de k_1. Lee *et al.* sugerem que experimentos computacionais mais extensos podem ser feitos para modificar ou refinar o cálculo deste parâmetro através da utilização dos demais fatores. Para o parâmetro k_2, a seguinte expressão é sugerida:

$$k_2 = \frac{\tau}{2\sqrt{\eta}}.$$

Estas regras para o cálculo de k_1 e k_2, embora bastante simples, têm sido amplamente utilizadas em diversos trabalhos que utilizam a regra ATCS e, em geral, têm funcionado de forma satisfatória para várias classes de problemas teste.

A Figura 2.1 ilustra, de forma bem geral, o pseudo-código da regra ATCS. O procedimento recebe como parâmetro de entrada o conjunto de tarefas do problema-teste e retorna como resultado a seqüência de processamento dada pela regra ATCS. O procedimento inicia com uma chamada da função que calcula as constantes τ, R, η, k_1 e k_2, que serão utilizadas na avaliação dos índices de prioridade $I_j(t,l)$. Em seguida, nos passos 2, 3 e 4, ocorrem as inicializações das variáveis t e l, assim como a inicialização da seqüência que será construída.

```
Procedure ATCS(ConjuntoTarefas) {
 1: CalculaConstantesATCS();
 2: Sequencia ← {};
 3: t ← 0;
 4: l ← 0;
 5: while (ConjuntoTarefas ≠ φ) do
 6:    AvaliaIndices(ConjuntoTarefas, t, l);
 7:    j ← SelecionaTarefa(ConjuntoTarefas);
 8:    Sequencia ← Sequencia ∪ {j};
 9:    ConjuntoTarefas ← ConjuntoTarefas\{j};
10:    t ← t + s_lj + p_j;
11:    l ← j;
12:    Atualiza(p̄, s̄);
13: end while
14: return Sequencia;
}
```

Figura 2.1: Pseudo-Código da Regra ATCS.

Feitas as inicializações o procedimento entra no laço principal, o qual irá construir a seqüência de processamento (passos 6 à 12). No passo 6 a função *AvaliaIndices*() é responsável pelo cálculo dos índices de prioridade $I_j(t,l)$ para cada tarefa $j \in$

$ConjuntoTarefas$. Após a avaliação dos índices, a função $SelecionaTarefa()$ seleciona a tarefa j que apresenta o maior valor entre os índices calculados. Em seguida (passos 8 e 9) a tarefa selecionada é retirada do conjunto de tarefas disponíveis e adicionada à seqüência parcial em construção. Finalmente, nos passos 10, 11 e 12, ocorrem as atualizações dos valores de t, l, $\bar{p}$ e $\bar{s}$. Note que na atualização de t os valores s_{lj} e p_j são respectivamente os tempos de preparação e processamento da tarefa selecionada, como definidos na Seção 1.2.

Embora a heurística ATCS seja capaz de produzir boas soluções para problemas de seqüenciamento em uma máquina, na maioria das vezes estas soluções ainda podem ser melhoradas através de um procedimento de busca local. Os próprios autores da regra sugerem a aplicação do método em três etapas: 1) Cálculo dos coeficientes, 2) Aplicação da regra de despacho e 3) Pós-processamento (melhoria). A seguir, mostramos um exemplo de aplicação da regra ATCS ao conjunto de dados apresentado na Seção 1.2.

Exemplo: Considere o seguinte exemplo numérico do problema de seqüenciamento de tarefas com tempos de preparação dependentes da seqüência mostrado no Capítulo 1. Para este exemplo temos:

tarefas	1	2	3	4	5
p_j	3	3	4	6	2
d_j	13	4	25	16	22
is_j	3	2	3	1	5

A matriz de tempos de preparação para este problema é mostrada a seguir.

tarefas	1	2	3	4	5
s_{1j}	-	3	2	2	4
s_{2j}	2	-	4	2	1
s_{3j}	4	1	-	5	6
s_{4j}	2	3	3	-	6
s_{5j}	2	3	1	5	-

Vamos então aplicar a regra ATCS ao problema dado.

Fase 1: Cálculo dos coeficientes.

Para este problema temos $\bar{p} = 3,6$, $\bar{s} = 3,0$ e $Var(s) \approx 2,24$. Com base nestes fatores podemos calcular os fatores C_v e β:

$$C_v = \frac{Var(s)}{\bar{s}^2} = \frac{2,24}{3^3} \approx 0,25$$

$$\beta = -1,88c_v + 0,92 = -1,88 \times 0,25 + 0,92 = 0,45$$

Podemos agora calcular a estimativa para o makespan, dada pela equação (2.6):

$$\hat{C}_{max} = \sum_{j=1}^{n} p_j + n\beta\bar{s} = 18 + 5 \times 0,45 \times 3,0 \approx 25.$$

Finalmente, com o estimador $\hat{C}_{max}$ calculado, podemos obter os fatores R, τ e η para determinar as constantes k_1 e k_2. Neste caso temos: $R = (25-4)/25 \approx 0,85$, $\tau = 1-(16/25) \approx 0,35$ e $\eta = 3,0/3,6 \approx 0,83$. Agora, utilizando as regras sugeridas por Lee *et al.* (1997), temos: $k_1 = 4,31$ e $k_2 = 0,19$. Podemos agora passar à segunda fase do procedimento.

Fase 2: seqüenciamento das tarefas.

Para determinar qual tarefa deve ser a primeira da seqüência devemos calcular os índices $I_j(0,0)$, para $j = 1, \ldots, 5$. Neste caso temos:

$$\begin{aligned}
I_1(0,0) &= \frac{1}{3} \cdot \exp\left(-\frac{13-3}{15,5}\right) \cdot \exp\left(-\frac{3}{0,53}\right) \approx 0,33 \times 0,52 \times 0,0040 = 7,0 \times 10^{-4}, \\
I_2(0,0) &= \frac{1}{3} \cdot \exp\left(-\frac{4-3}{15,5}\right) \cdot \exp\left(-\frac{2}{0,53}\right) \approx 0,33 \times 0,93 \times 0,0252 = 7,9 \times 10^{-3}, \\
I_3(0,0) &= \frac{1}{4} \cdot \exp\left(-\frac{25-4}{15,5}\right) \cdot \exp\left(-\frac{3}{0,53}\right) \approx 0,25 \times 0,26 \times 0,0040 = 2,5 \times 10^{-4}, \\
I_4(0,0) &= \frac{1}{6} \cdot \exp\left(-\frac{16-6}{15,5}\right) \cdot \exp\left(-\frac{1}{0,53}\right) \approx 0,17 \times 0,52 \times 0,1588 = 1,4 \times 10^{-2}, \\
I_5(0,0) &= \frac{1}{2} \cdot \exp\left(-\frac{22-2}{15,5}\right) \cdot \exp\left(-\frac{5}{0,53}\right) \approx 0,5 \times 0,27 \times 0,0001 = 1,3 \times 10^{-5}.
\end{aligned}$$

Como podemos notar o índice $I_4(0,0)$ possui o maior valor, de modo que a Tarefa 4 é selecionada para a primeira posição da seqüência. Note que nesta etapa a regra ATCS fornece os maiores índices para as Tarefas 2 e 4, que são as tarefas que possuem a data de entrega mais próxima e o menor tempo de preparação inicial respectivamente. Após a seleção da primeira tarefa as variáveis t e l são atualizadas de forma que $t = 1 + 6 = 7$ (preparação mais processamento da tarefa 4) e $l = 4$. Em seguida os valores de $\bar{p}$ e $\bar{s}$ são atualizados e os índices $I_j(7,4)$ são calculados,

Figura 2.2: Seqüência de processamento dada pela regra ATCS.

para $j \in \{1, 2, 3, 5\}$. Note que a data de entrega da Tarefa 4 é $d_4 = 16$, de modo que, até este ponto, não está ocorrendo nenhum atraso.

Continuando com este processo obtemos a seqüência $s_{ATCS} = (4, 1, 3, 2, 5)$, a qual possui um atraso total de 21. A Figura 2.2 ilustra os tempos de preparação e os atrasos ocorridos para este seqüenciamento das tarefas. Note que somente as tarefas 2 e 5 possuem atraso, e que neste caso, a estimativa para o makespan foi bastante precisa.

Fase 3: Melhoria.

Nesta etapa, Lee *et al.* sugerem um procedimento de busca local que realiza movimentos de troca de tarefas adjacentes para tentar melhorar a solução inicial obtida no passo 2. O procedimento de busca local percorre a vizinhança da solução s até a obtenção de um movimento que melhore o valor da função objetivo. Após a execução do movimento de melhoria a solução atual é re-avaliada, e o processo de busca de movimentos continua. Este processo se repete até que não seja mais possível a obtenção de movimentos de melhoria.

Após a aplicação da busca local encontramos a solução $s = (2, 5, 1, 4, 3)$, a qual possui um atraso total de 9 unidades de tempo. A representação gráfica deste seqüenciamento pode ser vista na Figura 1.1 da Seção 1.2.

Capítulo 3

Métodos de Múltiplos Reinícios

Métodos de Múltiplos Reinícios são procedimentos iterativos que operam através da geração de diferentes soluções iniciais que são posteriormente melhoradas através de um algoritmo de busca local. O método GRASP (*Greedy Randomized Adaptive Search Procedure*), desenvolvido por Tom Feo e Maurício Resende no final da década de 80, é um exemplo clássico deste tipo de abordagem. Neste capítulo vamos descrever a maneira como os componentes básicos de GRASP podem ser aplicados ao problema de seqüenciamento de tarefas.

3.1 Introdução

Um método de múltiplos reinícios é um processo iterativo onde cada iteração consiste de duas fases: construção e melhoria. A fase construtiva fornece soluções iniciais factíveis que posteriormente são submetidas a um procedimento de busca local (fase de melhoria). A melhor solução obtida durante o processo corresponde ao resultado final. Para caracterizar um método de múltiplos reinícios, precisamos especificar a estrutura do procedimento construtivo, caracterizado pelo tipo e quantidade de informações que ele utiliza, definir o tipo de vizinhança para a busca local e a estratégia de exploração da vizinhança.

Métodos de reinício com soluções iniciais geradas aleatoriamente são bastante utilizados em otimização global de funções contínuas. Torn e Zilinskas (1987) e Hu *et al.* (1994) apresentam exemplos de trabalhos que utilizam este tipo de abordagem. Em otimização combinatória, Lin e Kernighan (1973) foram os primeiros a usar este tipo de método para o problema do caixeiro viajante, com soluções iniciais geradas de forma aleatória e submetidas à uma busca local com movimentos *k-opt*. Soluções aleatórias iniciais geradas de forma uniforme em métodos de reinício tentam prover pontos de partida diversos para obter soluções de boa qualidade através de busca local. A grande deficiência deste tipo de abordagem é a total ausência de uso de

informação na construção da solução inicial.

Feo e Resende (1989) propuseram uma heurística probabilística para problemas complexos de recobrimento de conjuntos. Esta heurística já continha os ingredientes básicos da meta-heurística GRASP, proposta formalmente por Feo e Resende (1995). A grande novidade no método foi a introdução da idéia de uma fase construtiva com aleatorização controlada. GRASP tem sido aplicado a diversos tipos de problema de otimização combinatória, incluindo problemas de programação de tarefas em máquinas paralelas, como descrito em Festa e Resende (2001).

3.2 O Método GRASP

O metodo GRASP possui uma fase construtiva e uma fase de melhoria caracterizada por uma busca local. A cada iteração da fase construtiva, uma função gulosa avalia o conjunto de elementos candidatos a serem incorporados na solução parcial. A partir desta avaliação constrói-se uma lista restrita de candidatos (LRC) e um dos candidatos é escolhido probabilisticamente.

```
Procedure GRASP(MaximoIteracoes, FatorAleatorizacao) {
 1: MelhorSolucao ← {};
 2: for k ← 1 to MaximoIteracoes do
 3:    Solucao ← GeraSolucao(FatorAleatorizacao);
 4:    Solucao ← BuscaLocal(Solucao);
 5:    AtualizaSolucao(Solucao, MelhorSolucao);
 6: end for
 7: return MelhorSolucao;
}
```

Figura 3.1: Pseudo-Código do Método GRASP.

Em sua forma mais simples, o método GRASP necessita basicamente de dois parâmetros: o número máximo de iterações e um parâmetro que controla o nível de aleatorização das soluções geradas. A Figura 3.1 ilustra as linhas gerais de um pseudocódigo para o procedimento GRASP. A função *GeraSolucao*() é reponsável pela construção das soluções iniciais, que posteriormente são submetidas ao procedimento de busca local, denotado aqui por *BuscaLocal*(). O parâmetro *FatorAleatorizacao* é responsável pelo controle do grau de aleatorização na fase construtiva. A função *AtualizaSolucao*() é responsável apenas pela atualização e armazenamento da melhor solução encontrada durante o processo. Este processo é repetido até que o número máximo de iterações seja atingido.

3.2.1 Construção da lista restrita de candidatos

De uma maneira geral, a função objetivo faz o papel da função gulosa na avaliação dos candidatos e para um problema de otimização o custo incremental de cada elemento a ser adicionado em uma solução parcial é calculado. Para o problema aqui abordado, a função objetivo não se adequa à uma função gulosa por ser não decrescente. Dada uma seqüência parcial, o incremento no valor da soma dos atrasos de algumas tarefas não seqüenciadas pode ser zero, e desta forma não conseguimos definir uma ordenação de acordo com os incrementos. Uma boa alternativa consiste em ordenar as tarefas não seqüênciadas pelos índices da regra ATCS a elas associadas. A Figura 3.2 apresenta a fase construtiva com avaliação das tarefas candidatas através da regra ATCS. Note que este procedimento é basicamente o mesmo mostrado na Figura 2.1. A única alteração significativa ocorre no Passo 7, onde a tarefa é escolhida de forma probabilística. Neste caso o parâmetro α determina o nível de aleatoriedade do processo, definindo um limite mínimo de qualidade das tarefas que irão compor a lista LRC.

Procedure ATCS($ConjuntoTarefas$, α) {
1: $CalculaConstantesATCS()$;
2: $Sequencia \leftarrow \{\}$;
3: $t \leftarrow 0$;
4: $l \leftarrow 0$;
5: **while** ($ConjuntoTarefas \neq \phi$) **do**
6: $AvaliaIndices(ConjuntoTarefas, t, l)$;
7: $j \leftarrow SelecionaTarefa(ConjuntoTarefas, \alpha)$;
8: $Sequencia \leftarrow Sequencia \cup \{j\}$;
9: $ConjuntoTarefas \leftarrow ConjuntoTarefas \backslash \{j\}$;
10: $t \leftarrow t + s_{lj} + p_j$;
11: $l \leftarrow j$;
12: $Atualiza(\bar{p}, \bar{s})$;
13: **end while**
14: **return** $Sequencia$;
}

Figura 3.2: Pseudo-Código da Regra ATCS (versão aleatorizada).

Para a construção da LRC, considere uma seqüência parcial $\hat{s}$ obtida através da fase construtiva onde k tarefas foram seqüênciadas e seja I_j o índice ATCS associado a cada tarefa $j \in Q$, onde Q representa o conjunto de $n - k$ tarefas ainda não seqüênciadas. Os índices ATCS das tarefas com maior e menor prioridade são representados por I_{max} e I_{min}, respectivamente. A lista restrita de candidatos LRC

é composta das tarefas $j \in Q$ com os melhores indices I_j. Esta lista pode ser construída de duas formas: através da seleção das p tarefas melhores, onde p é um parâmetro, ou através de um parâmetro de qualidade $\alpha \in [0,1]$, que define um conjunto de tarefas que possuem índices ATCS I_j dentro de um intervalo específico: $I_j \in [I_{max} - \alpha(I_{max} - I_{min}); I_{max}]$. De acordo com esta definição o valor $\alpha = 0$ corresponde à construção puramente gulosa, pois neste caso a lista LRC irá conter apenas um elemento, enquanto que $\alpha = 1$ equivale a uma contrução aleatória (todas as tarefas contidas em Q irão fazer parte da lista LRC). A escolha de uma das formas, baseado em valor ou cardinalidade, dependente do problema em particular. No nosso caso vamos utilizar uma construção baseada no parâmetro α.

Para efeito de ilustração de aplicação deste procedimento considere o exemplo numérico apresentado no Capítulo 2 onde calculamos os índices $I_j(0,0)$ para cada tarefa do conjunto. Neste caso teríamos: $I_{max} = 1,4 \times 10^{-2}$, $I_{min} = 1,3 \times 10^{-5}$ e, para um valor de $\alpha = 0,5$, o intervalo de definição da lista de candidatos fica $[7,0 \times 10^{-3}; 1,4 \times 10^{-2}]$. De acordo com esta definição as tarefas 2 e 4 seriam selecionadas para a lista LRC.

3.2.2 Escolha de tarefas em LRC

Como mencionado anteriormente, a escolha de uma tarefa em LRC é feita de forma probabilística. Em grande parte das implementações de GRASP utiliza-se a distribuição uniforme, isto é, todos os elementos da LRC têm a mesma probabilidade de escolha. Bresina (1996) sugere outras distribuições de probabilidade para favorecer a escolha de alguns elementos. Essas distribuições são baseadas em um "rank" $r(\sigma)$ atribuído a cada elemento candidato σ em LRC. Assumindo que a lista de candidatos LRC esteja ordenada do melhor para o pior elemento, onde o elemento com menor rank é considerado o melhor da lista, Bresina sugeriu as seguintes *funções de bias*:

- *linear bias:* bias(r)=$1/(r+1)$, para $r \in$ LRC
- *log bias:* bias(r)=$\log^{-1}(r+1)$, para $r \in$ LRC
- *exponential bias:* bias(r)=e^{-r}, para $r \in$ LRC
- *polynomial(n) bias:* bias(r)=r^{-n}, para $r \in$ LRC

Lembramos ainda que a distribuição uniforme pode ser caracterizada pela função bias(r)=1. A partir dos valores de uma dada função de bias, a probabilidade $\pi(\sigma)$ de selecionar o elemento σ é dada por:

$$\pi(\sigma) = \frac{\text{bias}(r(\sigma))}{\sum_{\sigma' \in \text{LRC}} \text{bias}(r(\sigma'))}. \tag{3.1}$$

Em seu trabalho sobre amostragem estocástica, Bresina realiza uma série de experimentos computacionais onde ele compara o impacto de 6 tipos de funções de bias diferentes, classificadas em três níveis de "agressividade" (fracas: log e linear, médias: poly(2) e exponencial e fortes: poly(3) e poly(4)). Para o problema de seqüenciamento abordado em seu trabalho o autor verificou que os melhores resultados foram obtidos pelas funções de bias exponencial e polinomial de segunda ordem, classificadas como funções de bias de nível médio.

Neste trabalho utilizamos um esquema de seleção probabilística, baseada nos valores atribuídos pela regra de despacho. Este processo de atribuição de probabilidades favorece a escolha das tarefas com maior valor de índice prioridade ATCS. Neste caso as probabilidades de seleção dos elementos de LRC são diretamente proporcionais ao valor do índice ATCS a elas atribuído de modo que a seleção da tarefa a partir da lista LRC é feita com a seguinte probabilidade

$$\pi(\sigma) = \frac{I_\sigma}{\sum_{\sigma' \in \mathrm{LRC}} I_{\sigma'}}$$

onde σ representa um elemento (tarefa) da lista restrita de candidatos LRC e I_σ o valor ATCS atribuído ao elemento σ.

3.2.3 GRASP reativo

Um valor adequado para o parâmetro α, que controla o nível de aleatorização, é fundamental para o bom desempenho do método GRASP. De uma maneira geral, não existe um único valor satisfatório de α para todos os problemas-teste, e o melhor valor depende do tamanho e/ou de parâmetros do problema. Prais e Ribeiro (2000b) propuseram uma alternativa interessante para determinar valores adequados deste parâmetro, que é periodicamente ajustado de acordo com a qualidade das soluções obtidas, caracterizando um processo adaptativo denominado GRASP Reativo.

A proposta deste método é a utilização de um conjunto $A = \{\alpha_1, \alpha_2, \ldots, \alpha_m\}$, de valores pré-definidos para α, os quais são selecionados com probabilidades proporcionais à qualidade das soluções obtidas com cada parâmetro. Inicialmente, as probabilidades de escolha de cada valor são idênticas e dadas por $p_i = 1/m$ para todo $i = 1, 2, \ldots, m$. Estas probabilidades são alteradas periodicamente, a cada bloco de k reinícios efetuados, da seguinte maneira: seja $T(s^*)$ o atraso total da solução incumbente s^*, e $\overline{T_i}$ a média dos valores da função objetivo obtidos com a utilização do valor $\alpha = \alpha_i$ durante a fase construtiva. Inicialmente, calcula-se os valores

$$q_i = \left(\frac{T(s^*)}{\overline{T_i}} \right)^\delta , \tag{3.2}$$

para todo $i = 1, 2, \ldots, m$. O expoente δ pode ser utilizado para reforçar ou atenuar as diferenças entre os valores das probabilidades (valores de δ maiores que 1 tendem a reforçar as diferenças entre os valores q_i). Após o cálculo dos valores q_i as probabilidades atualizadas p_i são dadas pela seguinte expressão:

$$p_i = \frac{q_i}{\sum_{j=1}^{m} q_j}. \tag{3.3}$$

A determinação do período de atualização k é feita com base em experimentos computacionais. Em geral utiliza-se um valor $k = 10 \cdot |A|$.

Este processo de controle automático do parâmetro α é importante por duas razões: em primeiro lugar, evitamos o trabalho de calibração de parâmetros, o que pode tomar bastante tempo dependendo do problema. Em segundo lugar, a utilização de valores diferentes de α durante o processo permite a geração de soluções que provavelmente não seriam obtidas através da utilização de um único parâmetro α fixo. Um problema aparente que surge com esta abordagem é a determinação dos valores adequados α_i que irão compor a lista A e também o período de atualização dos valores da lista.

3.3 Procedimento de melhoria

Esta fase consiste da aplicação de um procedimento de busca local, caracterizado pelo tipo de *vizinhança* e pela estratégia de escolha dos movimentos dentro desta. No problema aqui abordado vamos utilizar três tipos de vizinhanças caracterizadas pelos seguintes movimentos: troca de duas tarefas, onde as posições de duas tarefas distintas são trocadas entre si, inserção de tarefa, onde uma tarefa é retirada de sua posição original e inserida em outra, e movimentos *3-opt*. A utilização deste último tipo de movimento citado é motivada por sua aplicação em problemas de caixeiro viajante assimétrico: no nosso caso a distância entre cidades corresponde ao tempo de preparação entre duas tarefas. A Figura 3.3 ilustra um exemplo de movimento de troca e inserção aplicados à uma seqüência de 8 tarefas.

É fácil mostrar que o tamanho da vizinhança de troca é $n(n-1)/2$ e o tamanho da vizinhança de inserção é $(n-1)^2$, e portanto ambas vizinhanças têm complexidade $O(n^2)$. Com efeito, considere a seguinte seqüência original de tarefas: $s = (1, 2, \ldots, n)$. No caso da vizinhança de troca, a primeira tarefa da seqüência pode trocar de posição com as tarefas 2, 3 ..., n, totalizando a geração de $n-1$ novas seqüências diferentes da original. Para a Tarefa 2 as possíveis trocas envolvem as tarefas 3, 4, ..., n, já que a troca entre as tarefas 2 e 1 é equivalente à troca entre 1 e 2, a qual já foi contabilizada. Seguindo este raciocínio temos um total de trocas possíveis (sem que ocorra a repetição de um movimento) igual à

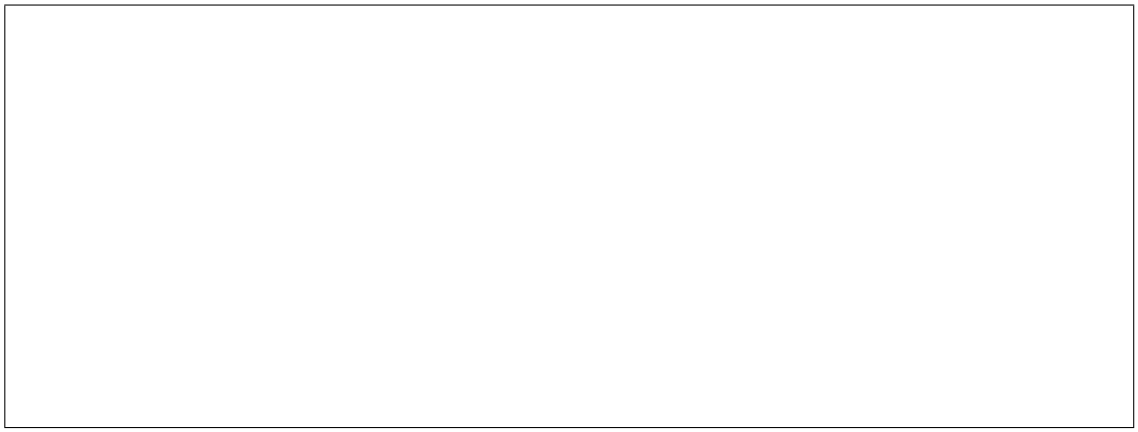

Figura 3.3: Movimentos de troca e inserção.

$NT = (n-1)+(n-2)+\ldots+1 = \sum_{k=1}^{n-1} k = n(n-1)/2$. Para o caso da vizinhança de inserção o procedimento de dedução do tamanho da vizinhança é análogo.

A seguir mostramos que o movimento *3-opt* é equivalente a uma inserção de bloco de tarefas em cada posição da seqüência. A implementação computacional do movimento foi realizada através desta *inserção generalizada*.

Seja uma seqüência:

$$s = (0, \cdots i, j, \cdots k, l, \cdots r, s, \cdots, 0).$$

Podemos fazer uma analogia entre os tempos de preparação que ocorrem entre as tarefas da seqüência e as distâncias percorridas entre cidades em problemas de caixeiro viajante. De acordo com esta analogia as tarefas seriam equivalentes às cidades e os tempos de preparação equivalentes às distâncias entre elas. Podemos então considerar os tempos de preparação como "arcos" que ligam as tarefas. Retornando à seqüência s, retirando os arcos (i, j), (k, l) e (r, s) as únicas substituições possíveis são (i, l), (k, s) e (r, j), resultando na seguinte seqüência

$$s' = (0, \cdots i, l, \cdots r, j, \cdots k, s, \cdots, 0)$$

que é equivalente à inserção da subseqüência de tarefas $(l, \cdots r)$ entre as tarefas i e j. A seguir, na Figura 3.4, mostramos um exemplo da execução de um movimento *3-opt* em uma seqüência de tamanho 16. Neste exemplo, os arcos (3,4), (7,8) e (12,13) foram substituídos pelos arcos (3,8), (12,4) e (7,13), o que equivale à inserção da subseqüência (8, 9, 10, 11, 12) entre as tarefas 3 e 4. Vale a pena lembrar que estes movimentos estão definidos para os índices $i = 1, \ldots, n-2$, com $i < j \leq n$ e $j < k \leq n$, onde n é o número de tarefas. O tamanho da vizinhança *3-opt* é dado por C_3^{n+1} (combinação $n+1$ três a três), o que caracteriza um vizinhança de tamanho igual à $n(n-1)(n+1)/6$, isto é, com complexidade $O(n^3)$.

Figura 3.4: Movimento *3-opt* aplicado ao problema de seqüenciamento.

Como podemos notar a vizinhança *3-opt* possui um tamanho relativamente grande, quando comparada com as vizinhanças de troca e inserção. Para contornar este problema podemos lançar mão de técnicas de redução de vizinhança. Estas técnicas definem um conjunto de regras que pode ser aplicado antes da avaliação de cada movimento, a fim de excluir os movimentos que *provavelmente* não irão causar melhoria no valor da função objetivo. Com isso conseguimos reduzir o tamanho da vizinhança e, consequentemente, o número total de soluções que devem avaliadas até a obtenção da melhor solução. Redução de vizinhança é comumente utilizada com o intuito de explorar um número maior de vizinhanças avaliando o menor número de soluções possíveis, e com isto obter soluções melhores.

Uma alternativa para reduzir o tamanho da vizinhança *3-opt*, utilizada no problema de seqüenciamento de tarefas, é a aplicação de um teste de redução baseado na verificação dos tempos de preparação que "entram" e "saem" da seqüência. Esta estratégia consiste em verificar se a soma dos tempos de preparação que estão saindo (arcos retirados) é menor que a soma dos que estão entrando (arcos inseridos). Como definido anteriormente, os "arcos" representam os tempos de preparação que ocorrem entre o processamento das tarefas. O teste de redução, baseado nos valores dos tempos de preparação, é definido da seguinte maneira: calcule

$$\delta \cdot \sum_{a_i \in A_1} s_{a_i} > \sum_{a_j \in A_2} s_{a_j}, \qquad \delta \geq 0, \tag{3.4}$$

onde A_1 representa o conjunto arcos que entram e A_2 o conjunto de arcos que saem da solução atual. Se a condição (3.4) for verificada o movimento deve ser descartado, ou seja, a solução resultante da aplicação deste movimento não deve ser avaliada. O parâmetro δ controla o nível de redução desejado, sendo que para $\delta = 0$ não temos nenhuma redução (pois a condição $0 > \sum_{a_j \in A_2} s_{a_j}$ nunca é verificada), enquanto que para $\delta \geq 1$ grande parte dos movimentos são descartados. Deve-se destacar que este teste de redução pode excluir movimentos de melhoria da vizinhança.

3.4 Extensões de GRASP

Em trabalho recente, Prais e Ribeiro (2000a) fazem um levantamento do estado da arte de GRASP e destacam inclusão de novas estratégias em GRASP para obter soluções de melhor qualidade. Gostariamos de destacar duas estratégias que provém da busca tabu:

i) Utilização de path-relinking proposto por Glover para explorar trajetórias entre mínimos locais encontrados por GRASP, é uma estratégia que tem produzido bons resultados.

ii) Utilização de memória de longo prazo em métodos de reinício, como proposto por Fleurent e Glover (1999). Este é o tema do próximo capítulo.

Em alguns trabalhos recentes, Fleurent e Glover (1999) e Binato *et al.* (2000) apresentam algumas propostas para a utilização de memória durante a etapa construtiva do método GRASP. Os resultados apresentados indicam um aumento de desempenho significativo em relação aos métodos que não dispõe de nenhum tipo de memória.

No capítulo seguinte apresentamos uma adaptação das idéias propostas por Fleurent e Glover (1999) para exploração de memória no processo construtivo do método GRASP aplicado ao problema de seqüenciamento de tarefas.

Capítulo 4

Métodos de Múltiplos Reinícios com Memória

Neste capítulo desenvolvemos um método de reinício com utilização de memória de longo prazo, seguindo as linhas gerais sugeridas por Fleurent e Glover (1999). Nosso objetivo é testar várias estratégias associadas com a inclusão deste tipo de memória e verificar a existência de um ganho de qualidade em relação às implementações de GRASP descritas no Capítulo 3.

4.1 Introdução

Como vimos no Capítulo 3 o método GRASP, em sua forma convencional, não dispõe de nenhum mecanismo de aprendizagem a partir das soluções obtidas ao fim de cada iteração. Fleurent e Glover (1999) propõem a incorporação de estratégias de memória adaptativa, propostas em busca tabu, na fase construtiva de métodos de múltiplos reinícios. O objetivo evidente desta proposta é prover um mecanismo de realimentação de informação para guiar a fase construtiva de forma a torná-la mais eficiente. Essas estratégias foram testadas com sucesso em problemas de designação quadrática (Fleurent e Glover) e de programação de tarefas em job shop (Binato *et al.* (2000)).

No esquema proposto por Fleurent e Glover, mantém-se uma população de soluções de alta qualidade (*soluções de elite*), suficientemente distintas, para influenciar a fase construtiva. O enfoque utilizado envolve as noções de variáveis *consistentes* e variáveis *fortemente determinadas* (Glover (1977)). Uma variável consistente é uma variável que recebe um determinado valor em um número significativo das soluções de elite. Variáveis fortemente determinadas, relativas às soluções de elite, são definidas como variáveis cujo valor não pode ser alterado sem que ocorra uma alta degradação do valor da função objetivo ou uma significativa alteração nos

valores de outras variáveis. A seguir vamos especificar as regras para selecionar as soluções que irão compor o conjunto de elite, e como as informações nele contidas podem ser utilizadas para guiar a construção da LRC durante a fase construtiva.

4.2 Construção do conjunto de elite

A construção do conjunto de elite envolve dois parâmetros: o tamanho do conjunto, que determina o número máximo de soluções a serem armazenadas, e a distância mínima entre soluções para manter um nível mínimo de diversidade no conjunto. Seja E o conjunto de soluções de elite e r o número máximo de soluções que podem ser armazenadas. Inicialmente, o conjunto E é composto por r soluções "nulas", todas com um valor de função objetivo infinito (dado que estamos trabalhando com um problema de minimização). Seja $T(s)$ o atraso total de cada solução $s \in E$, T_{max} o valor da pior solução presente no conjunto, ou seja, a solução de maior atraso, e T_{min} o atraso da melhor solução presente no conjunto E. Note que, enquanto o conjunto de elite não contiver r soluções completas (não nulas), o valor T_{max} é infinito.

De forma semelhante a Fleurent e Glover (1999), utilizamos um conjunto de regras simples para determinar se uma solução candidata deve entrar para o conjunto elite. Estas regras são baseadas na verificação da qualidade da solução candidata, porém com restrições de *distância mínima*. Se, ao final da busca local, uma solução s com atraso $T(s) < T_{max}$ for obtida, então s é uma solução candidata a entrar para o conjunto de elite, substituindo a solução de maior atraso, que pode ou não ser uma solução nula. Neste caso, a solução s entra para o conjunto E, se for suficientemente distante das soluções não nulas presentes no conjunto. Caso uma solução s seja classificada como muito próxima de alguma outra do conjunto E, ela é descartada, a menos que um *critério de aspiração* seja satisfeito: s é melhor que as demais soluções do conjunto, isto é, $T(s) < T_{min}$. Neste caso a solução s entra para o conjunto de elite independente da distância dela para as demais soluções do conjunto E, substituindo a pior solução.

A Figura 4.1 ilustra o esquema do funcionamento do método de múltiplos reinícios com utilização de memória. O fluxo de execução do código é representado pelas setas escuras (linhas cheias), enquanto que o destino das soluções obtidas ao final da busca local é indicado pelas setas com linhas tracejadas. O conjunto de setas horizontais paralelas em cinza indica a ocorrência de uma interação entre o conjunto de elite e a etapa construtiva. Nesta representação, os retângulos sem preenchimento indicam as verificações feitas em cada etapa do processo seletivo, onde o fluxo normal é desviado, caso a condição não seja satisfeita. Neste esquema, $d(s, E)$ indica que o procedimento verifica a distância entre s e cada solução s_k do conjunto elite E e

d_{min} representa a distância mínima pré-estabelecida para controle do nível de diversidade entre as soluções do conjunto. Mais adiante veremos como atribuir valores adequados ao parâmetro d_{min}.

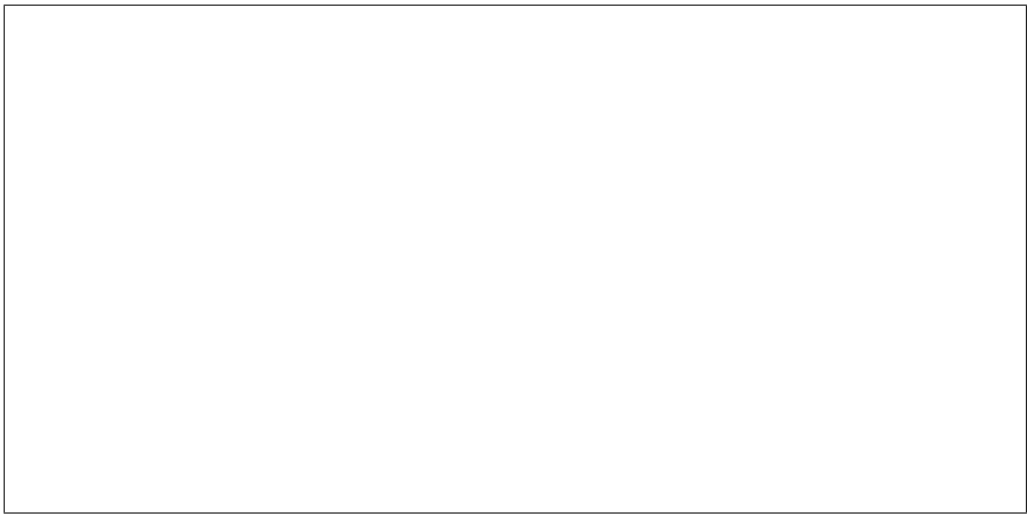

Figura 4.1: Fluxo de execução do método GRASP com utilização de memória.

Para medir a distância entre duas seqüências vamos utilizar duas infomações: a posição das tarefas nas seqüências e a relação de precedência entre duas tarefas nas seqüências. Esta última informação é utilizada devido à dependência do tempo de preparação entre duas tarefas consecutivas.

4.2.1 Medidas de distância

A distância de posição, ou distância de Hamming adaptada, consiste em contar o número de tarefas da primeira seqüência que não ocupam a mesma posição na segunda. A distância máxima neste caso é n, onde n é o número de tarefas do problema. A distância de precedência entre duas seqüências corresponde ao número de relações de precedência distintas nas duas seqüências. Em outras palavras, se a tarefa j não é imediatamente precedida pela tarefa i, em ambas as seqüências, então temos a ocorrência de uma diferença de precedência. Note que a distância máxima de precedência entre duas seqüências é $n-1$. Isto ocorre porque as relações de precedência distintas entre a tarefa fictícia 0 e a primeira tarefa são equivalentes a tarefas distintas na primeira posição, já contabilizadas no cálculo de distância de posição.

De forma semelhante a Laguna e Glover (1993), definimos então a distância total entre duas seqüências como a soma das distâncias de posição e precedência, de modo que a distância total máxima possível entre duas seqüências distintas é $2n-1$. Note

que de acordo com este critério podemos definir a distância mínima mostrada na Figura 4.1 como sendo $d_{min} = \rho(2n-1)$, onde $0 < \rho < 1$. Neste caso o parâmetro ρ é responsável pelo controle da distância mínima permitida entre as soluções do conjunto de elite.

Figura 4.2: Distância total entre seqüências de números.

A Figura 4.2 ilustra duas seqüências A e B onde mostramos destacadas as tarefas cujas posições não coincidem em ambas as seqüências. Neste exemplo a distância de posição entre as seqüências A e B é 5. Também é fácil notar a ocorrência da diferença de precedência das tarefas 2, 3, 4 e 7, de modo que a distância total (posição+precedência) entre as seqüências A e B é $d(A,B) = 5 + 4 = 9$.

4.3 Intensificação do processo construtivo

Nesta seção descreveremos uma estratégia de intensificação baseada em variáveis consistentes e fortemente determinadas, para modificar a construção da lista restrita de candidatos. A estratégia envolve uma função de intensidade $h(e)$, que representa uma medida das características destas variáveis, para cada elemento $e \in Q$, onde Q representa o conjunto de elementos disponíveis para serem adicionados à atual solução em construção (no nosso caso cada elemento e representa uma tarefa). Fleurent e Glover (1999) sugerem uma função de avaliação $K(e)$ para modificar a função gulosa que seleciona os elementos para a lista restrita de candidatos, dada por

$$K(e) = F(g(e), h(e)),$$

onde F é uma função monótona crescente que depende das funções $g(e)$ e $h(e)$. A função $g(e)$ depende da função gulosa, isto é, do custo incremental associado à adição do elemento e à solução em construção. Para o problema de seqüenciamento de tarefas abordado a função $g(e)$ é simplesmente o valor do índice ATCS associado ao elemento (tarefa) e.

A função $h(e)$ corresponde à função de intensidade de memória e depende da freqüência em que o elemento e aparece numa determinada posição em soluções de

elite, assim como da freqüência de relações de precedência entre tarefas que ocorrem em soluções de elite. Esta função também depende da atratividade do custo (valor da função objetivo) das soluções em que este elemento aparece. Após a construção da lista restrita de candidatos, a probabilidade de seleção do elemento e é dada por $\pi(e) = K(e)/\sum_{i\in \mathrm{LRC}} K(i)$.

Para o problema de designação quadrática (minimização), Fleurent e Glover (1999) propõem a seguinte função de intensidade:

$$h(e) = \sum_{s\in E: e\in s} \frac{Custo(s^*)}{Custo(s)},$$

onde E é o conjunto de soluções de elite, $Custo(s)$ é o valor da função objetivo da solução s (para o problema de minimização quadrática) e s^* representa a melhor solução do conjunto de elite. Note que a intensidade aumenta com a freqüência de e em soluções do conjunto de elite e também depende do valor da solução $s \in E$ que contém o elemento e (quanto menor o "custo" das soluções s onde ocorre o evento e, maior será o valor atribuído pela função $h(e)$). Além disso, é sugerida uma função linear para $K(e)$ com a seguinte forma

$$K(e) = \lambda \cdot g(e) + h(e), \tag{4.1}$$

onde $g(e) = c_{min}/c(e)$, o termo $c(e)$ representa o custo incremental da inclusão do elemento e na solução atual e c_{min} representa o menor custo incremental para todo elemento $e \in$ LRC. Note que $g(e) \leq 1$ e $h(e) \leq r$, onde r é o número de soluções de elite. Portanto, um valor aproximado de λ para o equilíbrio das componentes de $K(e)$ é r. Uma estratégia para controlar o valor de λ, proposta por Fleurent e Glover, consiste em dar maior ênfase para $g(e)$ nas iterações iniciais, através da atribuição de altos valores para λ, e mudar gradualmente a ênfase para $h(e)$, através da redução do parâmetro para valores da ordem de r. Este procedimento para o controle de λ é necessário, pois no início do processo não se dispõe de r soluções completas de boa qualidade, de modo que é razoável que o processo construtivo o tenha maior influência da componente gulosa.

O controle dinâmico do parâmetro λ varia de acordo com a diversidade das soluções geradas. Fleurent e Glover (1999) utilizam uma medida de entropia $\theta \in [0, 1]$, que reflete a freqüência em que um elemento e ocorre em uma determinada posição nas últimas m soluções geradas. De acordo com a definição utilizada pelos autores, valores de entropia próximos de 0 indicam uma baixa variabilidade das soluções geradas, enquanto que valores próximos de 1 indicam uma alta diversidade. Para o problema de designação quadrática os autores utilizam um valor inicial fixo $\lambda = 100r$, durante as 100 primeiras soluções geradas, de modo a privilegiar a seleção dos elementos de acordo com a função gulosa. Para as iterações restantes o valor de

λ é colocado inicialmente em $10r$, e reavaliado a cada $m = 100$ iterações para dar maior ou menor ênfase à componente de intensidade. A regra utilizada pelos autores é: Se o valor de entropia , calculado em função das últimas 100 soluções geradas, for maior que 0,5 (alta diversidade) o parâmetro λ é diminuido de r ($\lambda \leftarrow \lambda - r$). Caso contrário, se $\theta < 0,3$ (baixa diversidade), o valor de λ é aumentado de r ($\lambda \leftarrow \lambda + r$).

Binato *et al.* (2000) utilizam o mesmo procedimento de memória acima descrito para minimizar o makespan em um problema de job shop. A diferença reside no controle do parâmetro λ, o qual é baseado na variância do makespan obtido nas m últimas iterações. Eles utilizam um valor inicial de $\lambda = 100r$ durante as primeiras $2m$ iterações[1], passando a variar o parâmetro de acordo com a seguinte regra: se a variância calculada nas últimas m iterações aumenta, então o valor de λ aumenta de 10, caso contrário, λ decresce de 10.

Para o problema de seqüenciamento de tarefas propomos um esquema de memória também baseado nas idéias de Fleurent e Glover (1999), porém com algumas modificações. Seja Q o conjunto de tarefas disponíveis para processamento num dado instante de tempo t, durante o processo de sequênciamento das tarefas, e $I_j(t,l)$ o índice de prioridade ATCS da tarefa j a ser inserida na posição k, após a última tarefa seqüenciada l, no instante t. A função g é definida como:

$$g(k,l,j) = \frac{I_j(t,l)}{I_{max}}, \tag{4.2}$$

onde I_{max} é o maior índice (ATCS) de prioridade das tarefas ainda não seqüênciadas no instante t. Neste caso, quanto maior o valor atribuído pela regra ATCS à tarefa j, maior será o valor da componente g. Note que, embora o índice ATCS associado à tarefa j esteja definido em função do tempo t, podemos definir implicitamente g em função da posição k em que a tarefa j será incluída. Por este motivo podemos escrever $g(k,l,j)$, ao invés de $g(t,l,j)$. A função de intensidade h depende da posição e das relações de precedência das tarefas nas soluções de elite. Vamos definir inicialmente uma função $H(k,l,j)$ como

$$H(k,l,j) = \sum_{s \in Elite} (\delta_1(s,k,j) + \delta_2(s,l,j)) \cdot \frac{T(s^*)}{T(s)}, \tag{4.3}$$

onde as funções δ_1 e δ_2 são definidas por

$$\delta_1(s,k,j) = \begin{cases} 1 & \text{se a tarefa } j \text{ ocupa a posição } k \text{ na seqüência } s \\ 0 & \text{caso contrário} \end{cases}$$

$$\delta_2(s,l,j) = \begin{cases} 1 & \text{se a tarefa } j \text{ é precedida pela tarefa } l \text{ na seqüência } s \\ 0 & \text{caso contrário} \end{cases}$$

[1]Em geral os autores utilizaram $m = 100$ em todos os experimentos realizados.

Note que a função H reflete a freqüência em que a tarefa j ocupa a posição k e a freqüência em que a tarefa l precede a tarefa j em soluções do conjunto de elite. Finalmente, definimos a função h como

$$h(k,l,j) = \frac{H(k,l,j)}{H_{max}}, \tag{4.4}$$

onde $H_{max} = \max\{H(k,l,j) : j \in Q\}$. A normalização da função de intensidade H não foi usada por Fleurent e Glover (1999) mas revelou-se importante para facilitar o controle da ênfase das funções g e h, ambas com valores no intervalo $[0,1]$. A partir destas funções, a função de avaliação K é dada por

$$K(k,l,j) = (1-\lambda)\cdot g(k,l,j) + \lambda \cdot h(k,l,j), \qquad \lambda \in [0,1]. \tag{4.5}$$

A partir dos valores de $K(k,l,j)$ podemos calcular a probabilidade $p(k,l,j)$:

$$p(k,l,j) = \frac{K(k,l,j)}{\sum_{i \in \mathrm{LRC}} K(k,l,i)}.$$

Da mesma forma que na Equação (4.1) proposta por Fleurent e Glover (1999), o parâmetro λ controla a influência de cada componente, porém os possíveis valores de λ, neste caso, ficam restritos ao intervalo $[0,1]$, onde para $\lambda = 0$ temos o método GRASP sem a influência da componente de memória, enquanto que para $\lambda = 1$ temos um processo construtivo baseado apenas nos valores da componente de intensidade. Note que a utilização de componentes normalizadas na equação (4.5) facilita a obtenção do equilíbrio entre as componentes de intensidade e valor, bastando para isso utilizar $\lambda = 0,5$.

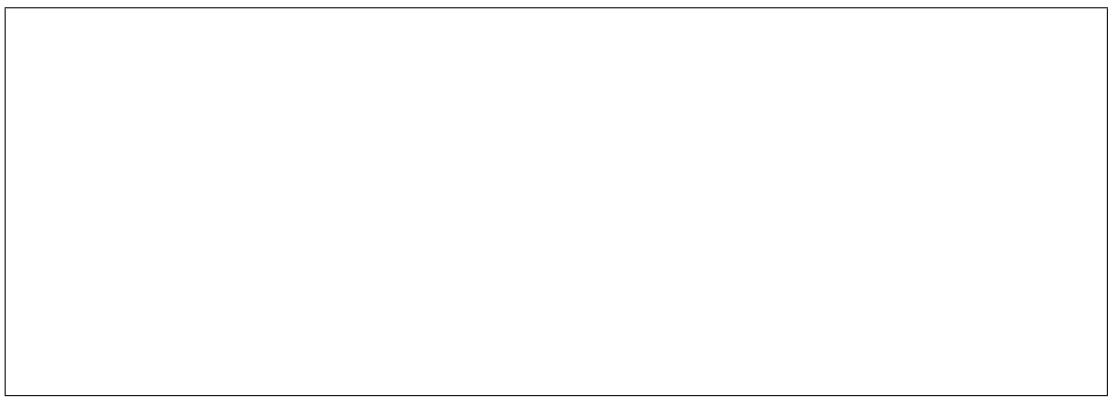

Figura 4.3: Exemplo de conjunto de soluções de Elite.

Para tornar mais clara a exposição do esquema de utilização de memória vamos examinar um pequeno exemplo. Considere a etapa do processo construtivo mostrada na Figura 4.3 onde aparecem o conjunto de soluções de elite, a seqüência em

construção e o conjunto de tarefas que ainda não foram alocadas. Neste exemplo o conjunto de elite já está completo e possui 4 soluções armazenadas ($r = 4$) e a seqüência em construção possui 6 tarefas alocadas, sendo que o número total de tarefas é dez. Ao lado direito das soluções do conjunto de elite temos os valores de atraso total de cada solução. Nesta etapa do processo construtivo o método deve selecionar uma das tarefas disponíveis para ser inserida após a última tarefa alocada na seqüência (neste caso a tarefa selecionada irá ocupar a posição 7, sendo precedida pela tarefa 4).

De acordo com o exemplo mostrado os valores de $H(k, l, j)$, para $k = 7$, $l = 4$ e $j \in \{6, 7, 8, 10\}$, são dados por:

$$\begin{aligned}
H(7,4,6) &= (0+0)\frac{100}{108} + (0+0)\frac{100}{100} + (0+0)\frac{100}{134} + (0+1)\frac{100}{120} = 0,83 \\
H(7,4,7) &= (0+1)\frac{100}{108} + (1+0)\frac{100}{100} + (0+0)\frac{100}{134} + (0+0)\frac{100}{120} = 1,93 \\
H(7,4,8) &= (1+0)\frac{100}{108} + (0+0)\frac{100}{100} + (1+1)\frac{100}{134} + (0+0)\frac{100}{120} = 2,42 \\
H(7,4,10) &= (0+0)\frac{100}{108} + (0+0)\frac{100}{100} + (0+0)\frac{100}{134} + (0+0)\frac{100}{120} = 0,0
\end{aligned}$$

Note que os valores de $H(k, l, j)$ são calculados, para cada tarefa j disponível, somando-se os valores da razão $T(s^*)/T(s)$ para cada coincidência de posição/precedência nas soluções do conjunto de elite. Como exemplo, a tarefa 6 possui apenas uma coincidência de precedência, ou seja, ela aparece após a tarefa 4 apenas uma vez em uma solução de elite (solução de custo 120). Já a tarefa 8 aparece na posição 7 em duas soluções do conjunto e também aparece após a tarefa 4 em uma solução. Neste exemplo a tarefa 8 possui o maior valor de intensidade, seguida das tarefas 7, 6 e 10. Note ainda que o valor máximo de intensidade que ocorre neste exemplo é $2,42$, e que após a aplicação da equação (4.4) obtemos os valores de intensidade 1,0, 0,79, 0,34 e 0,0, para as tarefas 8, 7, 6 e 10 respectivamente.

4.3.1 Utilização de memória

A memória não é utilizada nas primeiras m iterações do metódo de reinício. Isto porque precisamos obter um conjunto elite mais refinado e ao mesmo tempo coletar e armazenar informação suficiente de posição e precedência de tarefas das soluções geradas, para então fazer uso da memória. A partir desta fase inicial de "aquecimento", a memória passa a ser utilizada com um controle dinâmico do parâmetro λ, de forma que a ênfase dada às funções g e h seja adaptativa. A idéia consiste

em utilizar o controle reativo, descrito no Capitulo 3, onde atualizamos de forma simultânea os parâmetros de aleatorização α e intensidade λ. Definimos então uma lista $B = \{(\alpha_1, \lambda_1), \ldots, (\alpha_p, \lambda_p)\}$, com p pares de valores adequados, obtidos experimentalmente, e atribuímos probabilidade a cada par de valores dos parâmetros, de maneira idêntica à descrita no Capítulo 3. O motivo para a realização de um controle simultâneo dos parâmetros, ao invés de um controle separado para cada um, é a dificuldade em ajustar as probabilidades de cada parâmetro em separado (um mesmo valor de α pode ser responsável pela obtenção de soluções com diversos níveis de qualidade, dependendo do valor de λ).

4.4 Utilização de população estruturada

De um modo geral, é desejável que uma população contenha soluções de alta qualidade e que estas se situem em regiões distantes dentro do espaço de busca. Isto é válido para o método aqui descrito, como também para algoritmos genéticos (Michalewicz (1996)) e scatter search (Glover (1997) e Glover *et al.* (2000)). Algoritmos genéticos mantém na população as melhores soluções em termos de uma função de mérito (*fitness*), na maioria das vezes associada à função objetivo, e em geral não fazem uso explícito de distância entre soluções. No entanto, a divisão da população pode ser feita da maneira proposta por França *et al.* (2001). Scatter search divide sua população em duas sub-populações: uma contém soluções de alta qualidade, independente de distância, e a outra contém soluções dispersas, não necessariamente de boa qualidade.

Nesta seção o nosso objetivo é estruturar a população através de soluções de boa qualidade situadas em regiões (*clusters*) distantes. A motivação para identificar essas regiões no método de reinício é influenciar a fase construtiva com informação de regiões específicas e não com informação sobre todas as soluções de elite, como proposto por Fleurent e Glover (1999). Esta mesma estrutura de população pode ser usada em scatter search para combinar soluções de duas ou mais regiões, e em path-relinking, para guiar uma solução de uma região para uma solução de outra região. A seguir vamos apresentar uma estratégia para construir estas regiões, assim como um procedimento para utilizá-las na fase construtiva.

4.4.1 Construção de clusters de elite

Nesta seção apresentamos as regras utilizadas para a construção de um conjunto de elite estruturado, dividido em regiões (*clusters*). Considere a seguinte notação utilizada para apresentação das regras de construção deste conjunto:

- $d(s_i, s_j)$: distância entre duas soluções s_i e s_j.

- $d_{max} = 2n - 1$: distância total máxima que pode ocorrer entre duas soluções, onde n é o número de tarefas.
- sr_i: solução de referência, responsável pela demarcação da região que compõe o cluster i.
- $\epsilon = \rho \cdot d_{max}$: "raio" de definição dos clusters, onde $0 \leq \rho \leq 1$.
- $Max\_Clusters$: número máximo de clusters que podem ser criados (abertos).
- $Numero\_Clusters$: número atual de clusters abertos.
- $Tamanho\_Cluster$: número máximo de soluções em cada cluster.

Inicialmente, antes da obtenção de qualquer solução inicial, o conjunto de elite é preenchido com soluções "nulas" (vetores de zeros), todas com um valor de atraso total infinito. Até este ponto não existe nenhuma solução de referência, e a variável $Numero\_Clusters$ possui o valor zero. Após a obtenção da primeira solução "completa" s_1, o primeiro cluster é aberto, e s_1 corresponde à solução referência deste cluster, ou seja, $sr_1 = s_1$.

Figura 4.4: Representação gráfica de um cluster.

Na Figura 4.4 o sinal "+" representa a solução de referência, enquanto a região em cinza indica o espaço de soluções Ω. A circunferência ao redor do ponto representa o raio de definição do cluster. Considere agora uma segunda solução s_2, candidata a entrar para o conjunto de elite. Se $d(s_2, sr_1) \leq \epsilon$, então s_2 entra para o cluster 1, substituindo a solução de pior custo dentro do cluster (com exceção da solução de referência), caso contrário, um novo cluster é aberto tendo s_2 como

solução de referência. De modo geral, um novo cluster é aberto por uma solução s_j quando $d(s_j, sr_i) > \epsilon, \forall sr_i \in E$.

Suponha agora que os clusters $c_1, \ldots, c_q$ estão abertos com soluções de referência denotadas por $sr_1, \ldots, sr_q$, com $q \leq Max\_Clusters$, e considere uma solução candidata s_k. Se s_k pertence a vários clusters, então s_k é candidata a entrar para o cluster tal que $d(s_k, sr_i), i = 1, \ldots, q$, é mínima. Isto pode ocorrer pois a delimitação dos raios não garante a não sobreposição de regiões de definição dos clusters, conforme mostra a representação esquemática da Figura 4.5.

Figura 4.5: Sobreposição de regiões de delimitação de clusters.

Seja c_m o cluster tal que $d(s_k, sr_m)$ é mínima. Então s_k entra em c_m, substituindo uma solução de c_m, com exceção da solução de referência, se uma das seguintes condições é satisfeita:

i) s_k substitui a pior solução de c_m se s_k é melhor que s' e $d(s_k, s') > \frac{\epsilon}{10}$. A motivação para esta restrição de distância é evitar minimos locais muito próximos.

ii) s_k substitui a solução mais próxima dela em c_m se s_k é melhor que a melhor solução atual de c_m (critério de aspiração regional).

Suponha agora que todos os clusters estão abertos, ou seja, $Numero\_Clusters = Max\_Clusters$, e uma solução candidata s_k não pertence a nenhum cluster. Se a qualidade desta solução é melhor que a pior solução de referência de um cluster, então este cluster é eliminado e a solução candidata abre um novo cluster, tendo como solução de referência s_k. Caso contrário, a solução candidata é descartada. O inconveniente óbvio desta estratégia é a eliminação de um cluster com soluções potencialmente boas. Uma forma de contornar isto, seria retirar a restrição de número máximo de clusters (na realidade, limitar este número ao número de soluções

obtidas após a busca local), mas isto não foi explorado neste trabalho. Note que a solução de referência serve apenas para demarcar uma região e por este motivo ela não sai do cluster, com exceção do caso acima.

```
Procedure AtualizaConjuntoElite(s_k, T(s_k)) {
 1: Distancia_Minima ← ∞;
 2: c ← 0;
 3: for i ← 1 to Numero_Clusters do
 4:    if (d(s_k, sr_i) < Distancia_Minima) then
 5:       Distancia_Minima ← d(s_k, sr_i);
 6:       c ← i;
 7:    end if
 8: end for
 9: if (Distancia_Minima ≤ ρ · d_max) then
10:    Substitui_Pior_Solucao(s_k, T(s_k), c)
11: else
12:    if (Numero_Clusters < Max_Clusters) then
13:       Abre_Novo_Cluster(s_k);
14:    else
15:       Substitui_Pior_Cluster(s_k, T(s_k));
16:    end if
17: end if
}
```

Figura 4.6: Pseudo-Código para a construção de clusters.

A Figura 4.6 mostra as linhas gerais do pseudo código utilizado para a construção dos clusters. O procedimento recebe como parâmetros a solução candidata s_k e o atraso total $T(s_k)$ da seqüência. O procedimento é dividido em duas partes principais: inicialmente, passos 1 a 8, o método determina a distância mínima entre s_k e as soluções de referência sr_i, assim como o índice do cluster cuja solução de referência está mais próxima à s_k. Em seguida o procedimento determina se a solução candidata deve entrar para um cluster já aberto ou se ela deve abrir um novo cluster (passos 9 à 17). Neste esquema a função *Substitui_Pior_Solucao*() recebe como parâmetros a solução s_k, o atraso total $T(s_k)$ e o índice c, que indica qual cluster deve receber a solução s_k, caso ela atenda aos critérios de distância mínima e qualidade. Note que nesta figura omitimos as verificações de distância mínima e qualidade referente à solução candidata s_k, ficando estas tarefas a cargo da função *Substitui_Pior_Solucao*(). A função *Abre_Novo_Cluster*(), como o próprio nome indica, é responsável pela criação de um novo cluster, caso a solução s_k não pertença a nenhum cluster aberto e o número de clusters abertos não tenha atingido o

seu valor máximo $Max\_Clusters$. Finalmente, a função $Substitui\_Pior\_Cluster()$ é responsável pela eliminação do cluster que contém a pior solução de referência, caso a solução s_k tenha um custo melhor que a pior solução de referência sr_i, seguida da criação de um novo cluster, o qual terá como solução de referência a solução s_k.

4.4.2 Utilização das informações contidas nos clusters

Assuma um número q de clusters abertos, cada um deles com número de soluções limitado por $Tamanho\_Cluster$. A utilização da informação contida nos clusters é feita através de um processo cíclico. Sejam $c_1, c_2, \ldots, c_q$, os clusters abertos. Então, a utilização de informação segue a seqüência: $c_1, c_2, \ldots, c_q, *, c_1, c_2, \ldots, c_q, *, c_1, c_2, \ldots$, onde o sinal "$*$" indica uma etapa de diversificação do processo. Iniciando o ciclo por c_1, utilizamos apenas as informações das soluções deste cluster para executar uma fase construtiva. A seguir, utilizamos as soluções do cluster c_2 para outra fase construtiva e assim por diante até o cluster c_q. Neste ponto, indicado por "$*$", selecionamos uma solução de cada cluster e usamos q soluções para influenciar a nova fase construtiva. Em outras palavras, utilizamos ciclicamente informações regionais e a seguir combinamos a informação de cada região. Isto permite que o método intensifique a busca em cada região e diversifique a busca pela combinação de soluções das regiões. A seleção das soluções que serão utilizadas durante a etapa "$*$" ocorre de forma probabilística, onde a probabilidade de seleção de cada solução é inversamente proporcional ao atraso total da mesma.

Capítulo 5

Resultados Computacionais

Neste capítulo apresentamos os resultados computacionais obtidos com os métodos de múltiplos reinícios implementados. Inicialmente, realizamos uma série de experimentos computacionais para a calibração dos parâmetros de aleatorização e redução de vizinhança. Em seguida, realizamos uma série de experimentos para comparar o desempenho das diversas versões da abordagem proposta com o algoritmo memético proposto por França *et al.* (2001) e o algoritmo de busca local proposto por Carvalho (2002).

5.1 Introdução

Os métodos implementados neste trabalho foram escritos em linguagem Java 1.3 e processados em um computador pessoal com um único processador Athlon de 1.2 GHz, 256 Mb de memória DDR-RAM e sistema operacional *Windows Me*. O critério de parada utilizado foi o mesmo em todos os casos: o método é finalizado quando o número de soluções avaliadas atinge o valor $250N_s$, onde N_s é o tamanho da vizinhança *3-opt*. A única exceção a esta regra ocorre na comparação do método proposto por Carvalho (2002). Neste caso o critério de parada utilizado também foi baseado no número de soluções avaliadas, porém com um número máximo de avaliações que não leva em consideração o tamanho da vizinhança. Este critério de parada, baseado em número de avaliações, é conveniente pois permite que as análises feitas neste trabalho sejam reproduzidas em outras configurações, independente de linguagens ou arquiteturas de processador.

O desempenho dos métodos propostos foi avaliado através da aplicação de 4 conjuntos de dados, onde avaliamos a robustez dos métodos em relação aos seguintes fatores, considerados importantes:

- n: número de tarefas do problema.

- τ: fator de aperto das datas de entrega.
- R: fator de dispersão das datas de entrega.
- η: fator de severidade dos tempos de preparação.
- C_v: coeficiente de variação dos tempos de preparação.

5.2 Conjuntos de problemas teste

Quatro conjuntos de dados foram utilizados para avaliar o desempenho dos métodos apresentados. O primeiro conjunto (C1) consiste de 32 problemas utilizados por Rubin e Ragatz em seu método de Busca Genética (1995). Os detalhes das distribuições utilizadas na geração destes problemas podem ser encontrados no artigo citado. O segundo conjunto de dados utilizado (C2) é o conjunto gerado por Carvalho (2002) o qual consiste de 40 problemas com tamanhos variando entre 20 e 100 tarefas. O terceiro conjunto de dados utilizado (C3) foi gerado neste trabalho e envolve um conjunto de 252 problemas com tamanhos variando entre 50 e 150 tarefas. Finalmente, o quarto conjunto (C4), contendo 225 problemas foi gerado de forma semelhante à França *et al.* (2001).

O conjunto de problemas-teste (C2) utilizado por Carvalho foi gerado utilizando-se apenas distribuições uniformes. Os tempos de processamento foram todos gerados com distribuição no intervalo $[0, 100]$. Já os tempos de preparação das tarefas foram gerados a partir de distribuições uniformes no intervalo $[0, 100\eta]$, para $\eta = 0,3$ e $\eta = 0,7$. Neste caso o fator η controla o nível de severidade dos tempos de preparação desejado. As datas de entrega foram geradas utilizando-se o seguinte procedimento, conforme Lee *et al.* (1997):

- A média das datas de entrega é definida como $\mu = (1 - \tau)n\bar{p}$, onde n é o número de tarefas e $\bar{p}$ é a média dos tempos de processamento.
- calcula-se $\delta = Rn\bar{p}$, onde R é o fator de dispersão das datas de entrega, conforme definido pela equção 2.2 do Capítulo 2.
- As datas de entrega são geradas a partir de uma distribuição uniforme com intervalo $[\mu - \delta/2, \mu + \delta/2]$. Valores negativos são substituídos por 0.

Foram utilizados dois níveis de fator de aperto das datas de entrega, médio ($\tau = 0,3$), e alto ($\tau = 0,7$). O fator de dispersão das datas de entrega também foi variado de modo a gerar problemas em duas categorias, datas com baixa e alta dispersão, $R = 0,3$ e $R = 0,7$ respectivamente. Este conjunto de parâmetros fornece

um total de $2 \times 2 \times 2 = 8$ combinações possíveis para cada tamanho de problema: 20, 40, 60, 80 e 100 tarefas. Isto totaliza um conjunto de 40 problemas.

Para gerar os problemas teste do terceiro conjunto (C3), utilizamos dois tipos de distribuições: uniforme $U[a, b]$, onde a e b são os limites do intervalo da distribuição, e normal $N[\mu, \sigma]$, onde μ é a média e σ o desvio padrão da distribuição. De forma semelhante a Armentano e Mazzini (2000), utilizamos o seguinte procedimento para gerar os problemas: inicialmente geramos os tempos de preparação e processamento, onde os parâmetros das distribuições utilizadas foram escolhidos de forma estratégica para gerar problemas com diferentes níveis de fator de severidade e coeficiente de variação dos tempos de preparação. Em seguida calculamos a soma dos tempos de processamento TP e utilizamos dois fatores, α_1 e α_2, para determinar os parâmetros da distribuição que será utilizada para gerar as datas de entrega.

Para a distribuição uniforme utilizamos os fatores α_1 e α_2 para determinar os intervalos da distribuição uniforme $U[\alpha_1 TP; \alpha_2 TP]$. Já para os problemas com datas de entrega geradas através de uma distribuição normal, os fatores α_1 e α_2 controlam, respectivamente, a média e o desvio padrão da distribuição. Neste caso a distribuição normal tem os seguintes parâmetros: $N[\mu; \sigma]$, onde $\mu = \alpha_1 TP$ e $\sigma = \alpha_2 TP$.

As combinações de parâmetros utilizadas para gerar os tempos de preparação e processamento foram as seguintes: tempos de processamento com distribuições $U[0; 100]$, $U[50; 100]$ e $N[50; 10]$. Para os tempos de preparação também utilizamos três distribuições: $U[1; 20]$, $U[20; 100]$ e $U[100; 500]$. A motivação para utilizar estas combinações para os tempos de preparação é verificar o comportamento do teste de redução de vizinhança frente à diversas situações tais como tempos de preparação altos, da ordem ou maiores que os tempos de processamento, e baixa variância dos tempos de preparação. Finalmente, para as datas de entrega utilizamos 7 distribuições diferentes, modificadas de acordo com a distribuição de tempos de preparação utilizada:

- Problemas com tempos de preparação $U[1; 20]$: as datas de entrega dos problemas que utilizam esta distribuição para os tempos de preparação foram geradas através das seguintes distribuições: $U[0; 0,7TP]$, $U[0; TP]$, $U[0; 1,2TP]$, $U[0, 3TP; 0,8TP]$, $U[0, 5TP; TP]$, $N[0, 5TP; 0,1TP]$ e $N[0, 8TP; 0,1TP]$.

- Problemas com tempos de preparação $U[20; 100]$: neste caso as datas de entrega foram geradas através das seguintes distribuições: $U[0; 0,8TP]$, $U[0; 1,2TP]$, $U[0; 1,6TP]$, $U[0, 5TP; TP]$, $U[0, 8TP; 1,2TP]$, $N[0, 7TP; 0,1TP]$ e $N[TP; 0,1TP]$.

- Problemas com tempos de preparação $U[100; 500]$: neste caso as distribuições utilizadas foram: $U[0; 1,5TP]$, $U[0; 2TP]$, $U[0; 3TP]$, $U[TP; 1,5TP]$, $U[1,5TP; 2TP]$, $N[1,5TP; 0,2TP]$ e $N[2TP; 0,2TP]$.

A motivação para utilizar diferentes distribuições para as datas de entrega, dependendo dos tempos de preparação, é evitar a geração de problemas considerados não realistas, conforme destacado por Tan e Narasimhan (1997): Problemas com datas de entrega muito apertadas refletem a necessidade de um aumento do número de máquinas, ou ainda, um planejamento mais adequado das promessas de datas de entrega aos clientes. Por outro lado, problemas com bastante folga refletem situações onde o equipamento pode estar sendo sub-utilizado. Como exemplo, para os problemas com tempos de processamento $U[0;100]$ e tempos de preparação $U[100;500]$, a distribuição de datas de entrega $U[0;0,7TP]$ é considerada muito apertada.

Dado que utilizamos 3 distribuições diferentes para os tempos de processamento, 3 tipos para os tempos de preparação e 7 distribuições diferentes para as datas de entrega, temos então um total de $3 \times 3 \times 7 = 63$ combinações possíveis para cada tamanho de problema desejado, 50, 75, 100 e 150 tarefas, totalizando 252 problemas.

5.3 Calibração de parâmetros

Vamos apresentar agora alguns resultados iniciais obtidos com o método GRASP aplicado ao problema de seqüenciamento de tarefas, para a determinação dos valores mais adequados para os parâmetros de aleatorização e redução de vizinhança. A implementação utilizada para a calibração destes parâmetros é basicamente a mesma apresentada na Seção 3.2.1. Neste caso o procedimento construtivo utiliza a regra de despacho ATCS para construir a lista restrita de cadidatos, sem a utilização de memória, e a fase de melhoria utiliza movimentos *3-opt* com redução de vizinhança. As análises para as vizinhanças de troca e inserção não foram incluídas neste trabalho, dado o baixo desempenho apresentado na maioria dos casos analizados.

Como mencionado no Capítulo 3 o método GRASP, em sua forma mais simples, necessita da calibração de um único parâmetro: o fator de aleatorização α. Este parâmetro controla o nível de diversidade (e qualidade) das soluções geradas durante a fase construtiva. É interessante ressaltar aqui a importância de uma calibração adequada deste parâmetro. Em muitos casos ele pode ter uma forte dependência do tamanho ou das características do conjunto de dados do problema, de modo que a utilização de um controle reativo pode mostrar-se uma alternativa bastante viável. A utilização de valores inadequados para α pode acarretar em uma perda de diversidade das soluções geradas, o que pode comprometer o resultado final do processo.

Outro parâmetro importante, introduzido para reduzir a complexidade da vizinhança durante a fase de melhoria, é o fator de redução de vizinhança δ. Este parâmetro tem uma influência direta sobre a qualidade das soluções finais, obtidas após a aplicação do procedimento de busca local. Valores de δ próximos de zero

tendem a fazer com que o número de soluções avaliadas necessário para obter um mínimo local seja muito grande. Por outro lado, se a redução da vizinhança for muito acentuada, corremos o risco de comprometer o poder de melhoria da vizinhança e, por conseqüência, a qualidade final das soluções obtidas.

Como podemos notar, temos um par de parâmetros que controlam, em conjunto, a qualidade e diversidade das soluções obtidas. Dessa forma, é necessário verificar a influência de cada parâmetro através da análise de um conjunto de pares de valores (α, δ). Esta análise é válida para determinar um conjunto de valores mais adequados destes parâmetros para o método GRASP em sua forma mais básica, sem controle reativo de parâmetros. A avaliação do comportamento do parâmetro α também é útil para a determinação dos valores mais adequados da lista $A = \{\alpha_1, \alpha_2, \ldots, \alpha_m\}$, utilizada pelo método com controle reativo.

Os conjuntos de dados utilizados para a calibração dos parâmetros foram divididos em três categorias, de acordo com o tamanho dos problemas: problemas pequenos, 15, 20, 25 e 35 tarefas, médios, 40, 45, 50 e 60 tarefas e grandes, 75, 80, 100 e 150 tarefas. Nesta análise foram incluídos todos os problemas gerados por Rubin e Ragatz (1995), todos os problemas gerados por Carvalho (2002), e parte dos problemas gerados neste trabalho. Para cada categoria foram utilizados 40 problemas teste com diferentes características, tais como datas de entrega apertadas, datas de entrega com folga, diferentes fatores de severidade dos tempos de preparação e ainda diversos níveis de dispersão de datas de entrega, totalizando 120 problemas.

A seguir mostramos três tabelas com médias dos ganhos percentuais do método GRASP em relação à regra ATCS. O ganho percentual é definido como

$$G(s) = \frac{T_{ATCS}(s) - T_{grasp}(s)}{T_{ATCS}(s)} \cdot 100, \tag{5.1}$$

onde $T_{ATCS}(s)$ e $T_{grasp}(s)$ são os valores de atraso total da solução s obtidos pela regra ATCS e pelo método GRASP, respectivamente. Nos casos em que $T_{ATCS}(s) = 0$ e $T_{grasp}(s) \geq 0$, definimos o ganho como 0%.

A Tabela 5.1 apresenta os resultados para o conjunto de problemas pequenos. Nela apresentamos os ganhos médios obtidos para cada combinação de valores α e δ, e também as médias por linhas (α fixo) e colunas (δ fixo). O critério de parada utilizado foi o mesmo em todos os casos, baseado no número de soluções avaliadas, conforme citado anteriormente.

Como podemos verificar os ganhos percentuais foram bem parecidos em quase todos os casos. Também é fácil notar que temos um aumento de desempenho do método para valores de α próximos de 0,5. Para $\alpha > 0,5$ o método não apresentou nenhum aumento de desempenho significativo. Como esperado, o fator de redução também possui um papel importante no processo. Note que o pior resultado ocorre

Tabela 5.1: Calibração de parâmetros: problemas pequenos.

	$\delta = 0,3$	$\delta = 0,4$	$\delta = 0,5$	$\delta = 0,6$	$\delta = 0,7$	Média
$\alpha = 0,1$	14,82	14,73	14,14	14,32	12,95	14,19
$\alpha = 0,2$	15,22	15,46	15,07	14,78	14,81	15,07
$\alpha = 0,3$	15,50	15,41	15,44	15,03	15,42	15,36
$\alpha = 0,4$	15,41	15,83	15,74	15,73	15,41	15,62
$\alpha = 0,5$	15,63	15,76	15,74	15,50	15,61	15,65
Média	15,32	15,44	15,23	15,07	14,84	

para a combinação $\alpha = 0,1$ e $\delta = 0,7$, que são exatamente os valores de menor aleatorização e maior redução utilizados. Para este par de valores, o método apresentou um desempenho relativamente fraco, sendo que mais da metade dos piores casos, dentre os 40 problemas pequenos, ocorreram para esta configuração.

Em relação ao número de vitórias, as combinações mais promissoras foram aquelas em que valor de α estava em torno de 0,5 e δ em torno de 0,4. Vale a pena ressaltar que, salvo raras exceções, o desvio máximo entre o melhor e o pior valor de atraso total obtido, entre todas as combinações avaliadas, foi inferior à 10%, sendo que a diferença média ficou em torno de 4%.

A seguir, na Tabela 5.2, mostramos os resultados obtidos para o conjunto de problemas de porte médio. Novamente os ganhos percentuais foram bem parecidos, com uma ligeira tendência de melhoria para valores de α em torno de 0,3 e valores de δ em torno de 0,5.

Tabela 5.2: Calibração de parâmetros: problemas médios.

	$\delta = 0,3$	$\delta = 0,4$	$\delta = 0,5$	$\delta = 0,6$	$\delta = 0,7$	Média
$\alpha = 0,1$	25,22	25,05	26,12	25,44	24,69	25,30
$\alpha = 0,2$	25,53	26,09	26,47	26,27	26,15	26,10
$\alpha = 0,3$	26,27	26,66	26,69	26,43	26,91	26,59
$\alpha = 0,4$	25,79	26,51	26,78	26,83	26,73	26,53
$\alpha = 0,5$	25,67	26,31	26,70	26,90	26,72	26,46
Média	25,70	26,12	26,55	26,37	26,24	

Um fato interessante a ser notado é que o número de reinícios efetuados e o tempo computacional necessário para realizar o mesmo número de avaliações de soluções podem aumentar de forma significativa com o aumento de δ. O aumento do número total de reinícios com o aumento de δ decorre do fato de que uma grande redução da vizinhança implica em uma diminuição do número de soluções que devem ser avaliadas para a obtenção de um mínimo local. O aumento do número de reinícios acarreta em um aumento do número de vezes que o procedimento construtivo é executado, o que aumenta o esforço computacional do método. A proporção do

número de chamadas da função de teste de redução de vizinhança que ocorrem para cada movimento efetuado, também aumenta em função do fator δ. Com isso o aumento do tempo computacional, em alguns casos, pode chegar a ser 30% maior em relação ao método que não utiliza redução de vizinhança.

Finalmente, na Tabela 5.3, apresentamos os ganhos percentuais para o conjunto de problemas grandes. Novamente, assim como para o conjunto de problemas médios, o melhor valor de α parece ficar em torno de 0,3. Já o fator de redução teve um comportamento um pouco diferente em relação aos casos anteriores.

Tabela 5.3: Calibração de parâmetros: problemas grandes.

	$\delta = 0,3$	$\delta = 0,4$	$\delta = 0,5$	$\delta = 0,6$	$\delta = 0,7$	Média
$\alpha = 0,1$	27,71	27,99	28,32	28,32	28,25	28,12
$\alpha = 0,2$	27,94	27,93	28,29	28,18	28,62	28,19
$\alpha = 0,3$	27,92	28,00	28,44	28,21	28,65	28,24
$\alpha = 0,4$	27,44	27,72	28,34	28,31	28,42	28,05
$\alpha = 0,5$	27,72	27,88	28,09	28,16	28,01	27,97
Média	27,74	27,90	28,30	28,24	28,39	

Um fato interessante a ser notado em relação aos resultados da Tabela 5.3, é o bom desempenho do método para as vizinhanças com redução entre $\delta = 0,5$ e $\delta = 0,7$. Este fato sugere que o valor adequado de δ pode variar não apenas em função do número de tarefas, mas também em função das características dos problemas teste analisados. De fato, uma análise mais detalhada dos resultados revelou que o valor adequado para o fator de redução tem uma grande correlação também com o coeficiente de variação dos tempos de preparação. Dessa forma achamos conveniente utilizar uma regra simples para determinar o nível de redução de vizinhança, dependendo das características do conjunto de dados: o fator de redução de vizinhança foi fixado em $\delta = 0,5$ para problemas pequenos (até 40 tarefas) e para problemas com coeficiente de variação dos tempos de preparação maior que 0,2. Para problemas com coeficiente de variação menor que 0,2 nós fixamos $\delta = 0,7$. Note que esta regra para determinar o valor de δ foi mantida para todas as versões de métodos de reinício utilizados. Ressaltamos que, embora bastante simples, esta regra funcionou de forma satisfatória na maioria dos casos.

5.4 Comparação entre métodos

Nesta seção apresentamos os resultados computacionais obtidos com as quatro versões de métodos de múltiplos reinícios implementadas, aplicadas aos conjuntos de dados anteriormente apresentados. Também comparamos o desempenho relativo

dos métodos frente ao algoritmo memético proposto por França *et al.* (2001), e ao algoritmo de busca local com perturbação proposto por Carvalho (2002). As versões de métodos de múltiplos reinícios propostas para comparação são as seguintes:

- GRASP: método GRASP com parâmetro de aleatorização fixo.
- GRASPR: método GRASP com controle reativo de aleatorização.
- RM: método de reinício com memória não estruturada e controle reativo de aleatorização e intensidade de memória.
- RME: idem ao anterior, porém com o uso de população estruturada (clusters).

Em todos os casos citados acima utilizamos uma vizinhaça $3-opt$ durante a etapa de melhoria. Quanto aos demais parâmetros utilizados em cada método temos: para o método GRASP utilizamos um fator de aleatorização $\alpha = 0,5$ para problemas pequenos (até 40 tarefas) e $\alpha = 0,3$ para problemas com mais de 40 tarefas. O fator de redução de vizinhança foi fixado de acordo com a regra citada anteriormente.

Para o método GRASPR utilizamos a seguinte lista de valores para a lista de valores de α: A={0,2; 0,25; 0,3; 0,35; 0,4; 0,45; 0,5}. Neste caso as probabilidades de seleção dos elementos de A foram atualizadas a cada bloco de 100 iterações GRASP. O expoente δ, que modifica as probabilidades de seleção dos elementos de A (equação (3.2)), foi fixado em $\delta = 10$.

O método de reinício com memória utilizou um conjunto de elite de tamanho $r = 20$, independente do tamanho do problema-teste utilizado. Este valor para r foi determinado com base em experimentos computacionais. Valores de r maiores que 20 em geral não melhoraram a qualidade das soluções obtidas. Porém, o tempo computacional pode aumentar com o aumento do tamanho do conjunto de elite, dado que a atualização do conjunto envolve o cálculo de distâncias entre a solução candidata e as soluções presentes no conjunto. Para o controle do fator de aleatorização e intensidade de memória utilizamos uma lista de pares (α_i, λ_i) composta de 12 pares de valores, correspondentes às combinações dos seguintes valores: $\alpha = \{0,2; 0,3; 0,4; 0,5\}$ e $\lambda = \{0,5; 0,6; 0,7\}$. Inicialmente, durante as 100 primeiras iterações, fazemos $\lambda = 0$ de modo que nenhuma informação referente ao conjunto de elite é utilizada. Durante este período o procedimento construtivo utiliza apenas o valor de α_i para gerar as soluções iniciais. Após a ocorrência de 100 iterações (reinícios) o procedimento construtivo passa a utilizar também o parâmetro λ_i, para dar ênfase à componente de memória. A distância mínima entre as soluções do conjunto de elite foi fixada em $d_{min} = 0,5\dot{(}2n-1)$, ou seja, $\rho = 0,5$. Este valor mostrou-se bastante apropriado para todos os problemas teste analisados. Valores de $\rho > 0,5$, em geral, implicam em um conjunto de elite mais diversificado, porém

com o possível comprometimento da qualidade das soluções. Por outro lado, valores de ρ muito pequenos podem levar à uma rápida convergência, comprometendo a exploração de outras regiões.

Para o método de reinício com memória estruturada utilizamos basicamente os mesmos parâmetros do método anterior. A diferença mais significativa ocorre na determinação do parâmetro de distância mínima. Neste caso o parâmetro ρ controla a distância mínima entre soluções de referência, de modo que valores mais altos que os utilizados anteriormente são mais adequados. Dessa forma utilizamos $\rho = 0,75$ para determinar a distância mínima entre as soluções de referência. Este valor mostrou-se bastante adequado, permitindo a abertura de vários clusters, mesmo em estágios avançados da busca. Em relação ao conjunto de elite, fixamos o número máximo de clusters que podem ser abertos em 5, e o número de soluções por clusters também em 5. Isto caracteriza um conjunto de elite com no máximo 25 soluções. Outras combinações para número de clusters e tamanho de cluster foram avaliadas, porém a combinação 5×5 foi a que se mostrou mais robusta.

5.4.1 Conjunto de problemas teste C1

Nesta seção apresentamos os resultados computacionais obtidos com as quatro versões de métodos de múltiplos reinícios implementadas, aplicadas ao conjunto de dados proposto por Rubin e Ragatz (1995). Também comparamos o desempenho relativo dos métodos frente à regra ATCS e ao algoritmo memético proposto por França *et al.* (2001).

O critério para comparar o desempenho dos métodos foi medir o ganho percentual em relação aos resultados obtidos pelo método branch-and-bound proposto por Ragatz (1993). Os valores brutos dos atrasos obtidos por Ragatz podem ser vistos no trabalho de Rubin e Ragatz (1995). Vale a pena lembrar que os resultados apresentados por Ragatz (1993) não são necessariamente os valores ótimos de cada problema, dado que o método branch-and-bound foi executado impondo-se um limite máximo para o número de nós explorados: o método foi executado até que o número de nós explorados atingisse 2 milhões. Nos casos em que este número de nós foi atingido, o método foi interrompido e a solução incumbente tomada como resultado final do processo.

Como podemos notar o método de múltiplos reinícios com população estruturada obteve o melhor desempenho médio. Esta versão também foi responsável pelo maior número de vitórias: obteve o melhor resultado, considerando-se os empates, em 30 dos 32 problemas do conjunto. O desempenho do método de múltiplos reinícios com memória também foi considerado bom. Ele conseguiu obter os melhores resultados em 27 problemas, novamente considerando-se os empates, e foi superior ao

Tabela 5.4: Conjunto de problemas teste C1.

Problema	n	ATCS	Memético	GRASP	GRASPR	RM	RME
prob401.sms	15	-38,89	-3,33	0,0	0,0	0,0	0,0
prob402.sms	15	0,0	0,0	0,0	0,0	0,0	0,0
prob403.sms	15	-7,23	-1,14	0,0	0,0	0,0	0,0
prob404.sms	15	-8,34	0,0	0,0	0,0	0,0	0,0
prob405.sms	15	0,0	0,0	0,0	0,0	0,0	0,0
prob406.sms	15	0,0	0,0	0,0	0,0	0,0	0,0
prob407.sms	15	-5,10	0,0	0,0	0,0	0,0	0,0
prob408.sms	15	-1,61	0,0	0,0	0,0	0,0	0,0
prob501.sms	25	-7,52	1,88	1,50	1,50	1,88	1,88
prob502.sms	25	0,0	0,0	0,0	0,0	0,0	0,0
prob503.sms	25	-1,40	0,14	0,40	0,40	0,20	0,40
prob504.sms	25	0,0	0,0	0,0	0,0	0,0	0,0
prob505.sms	25	0,0	0,0	0,0	0,0	0,0	0,0
prob506.sms	25	0,0	0,0	0,0	0,0	0,0	0,0
prob507.sms	25	-6,55	0,0	0,0	0,0	0,0	0,0
prob508.sms	25	12,85	17,71	17,71	17,71	17,71	17,71
prob601.sms	35	-23,33	58,33	76,67	78,33	80,0	80,0
prob602.sms	35	0,0	0,0	0,0	0,0	0,0	0,0
prob603.sms	35	1,08	3,24	4,06	4,16	4,22	4,32
prob604.sms	35	-0,36	1,47	1,32	1,57	1,86	1,86
prob605.sms	35	-30,93	8,25	19,24	16,15	21,31	20,96
prob606.sms	35	0,0	0,0	0,0	0,0	0,0	0,0
prob607.sms	35	-5,27	2,08	2,52	2,68	2,77	2,77
prob608.sms	35	20,91	32,68	33,06	32,95	33,06	33,06
prob701.sms	45	-44,17	3,33	15,00	16,67	16,67	18,33
prob702.sms	45	0,0	0,0	0,0	0,0	0,0	0,0
prob703.sms	45	-1,07	1,53	1,58	1,78	2,03	2,23
prob704.sms	45	-2,43	4,99	4,39	4,44	4,84	4,84
prob705.sms	45	-16,24	-0,85	8,55	10,26	11,11	14,53
prob706.sms	45	0,0	0,0	0,0	0,0	0,0	0,0
prob707.sms	45	1,99	5,72	5,75	5,93	6,12	6,28
prob708.sms	45	-0,60	5,25	5,21	5,05	6,19	6,19
Média		-5,13	4,41	6,16	6,24	6,56	6,73

método com memória estruturada em apenas 1 caso. Já o algoritmo memético foi capaz de obter os melhores resultados em 18 problemas, tendo superado os demais métodos no problema prob704.sms. Vale a pena lembrar que todos os métodos apresentados forneceram bons resultados, chegando bem próximo aos melhores valores conhecidos, em quase todos os casos. O desempenho da regra ATCS também pode ser considerado bom, dado que o método é capaz de obter soluções de qualidade razoável com um custo computacional muito baixo. Em relação ao tempo computacional, os métodos de reinício e o algoritmo memético, em geral, apresentaram um comportamento parecido. Os tempos computacionais variaram entre 0,5 segundos, para cada problema com 15 tarefas, e entre 15 e 20 segundos para cada problema com 45 tarefas.

5.4.2 Conjunto de problemas teste C2

Vamos apresentar agora os resultados computacionais obtidos através da aplicação do conjunto de dados proposto por Carvalho. Para este conjunto comparamos o desempenho relativo dos métodos propostos frente ao algoritmo Mono-A, proposto por Carvalho (2002) e frente ao algoritmo memético proposto por França *et al.* (2001). A Tabela 5.5 mostra os resultados computacionais obtidos para o conjunto de problemas-teste C2, composto por 40 problemas, pelas diferentes implementações propostas. Neste caso optamos por apresentar os resultados em função do desvio da melhor solução conhecida. A equação para o cálculo do desvio é dada por

$$D(s) = \frac{T_{alg}(s) - T(s^*)}{T(s^*)} \cdot 100, \tag{5.2}$$

onde $T_{alg}(s)$ é o atraso total da solução s obtido pelo método alg e $T(s^*)$ é o atraso da melhor solução conhecida s^*.

Analisando os resultados da Tabela 5.5 podemos verificar o bom desempenho do método proposto por Carvalho para os problemas grandes, acima de 100 tarefas. Já para os problemas pequenos e médios, o método de reinício com memória estruturada obteve os melhores resultados, sendo que o desvio médio ficou abaixo dos 2%. Para os problemas grandes o maior desvio ocorreu para o problema p36.sms. O alto valor de desvio, neste caso, deve-se ao baixo valor de atraso total obtido para a melhor solução encontrada. Este fato acarreta grandes variações de desvio, mesmo para pequenas variações nos valores finais brutos obtidos pelos métodos. Note que mesmo os métodos Mono-A e Memético obtiveram um alto valor de desvio para este problema. Excluindo-se este problema no cálculo das médias dos ganhos percentuais obtemos os os valores de desvio mostrados na Tabela 5.6.

Nesta nova situação notamos que os desvios percentuais foram bem parecidos para o algoritmo Mono-A e os métodos de reinício com memória. Quanto ao número

Tabela 5.5: Conjunto de problemas teste C2.

	n	Memético	Mono-A	GRASP	GRASPR	RM	RME
p01.sms	20	4,93	0,0	0,0	0,0	0,0	0,0
p02.sms	20	0,0	0,0	0,0	0,0	0,0	0,0
p03.sms	20	4,96	0,0	0,0	0,0	0,0	0,0
p04.sms	20	0,0	1,51	4,19	4,19	0,0	0,0
p05.sms	20	0,0	0,0	0,0	0,0	0,0	0,0
p06.sms	20	0,0	0,66	0,0	0,0	0,0	0,0
p07.sms	20	0,97	1,18	0,0	0,0	0,0	0,0
p08.sms	20	0,78	0,0	0,12	0,12	0,12	0,0
p09.sms	40	4,06	0,0	1,82	1,44	0,0	0,0
p10.sms	40	125,0	36,11	36,11	27,78	11,11	5,56
p11.sms	40	20,79	1,75	6,05	3,05	1,36	1,75
p12.sms	40	34,66	0,39	3,59	5,26	0,39	0,0
p13.sms	40	2,61	0,16	0,59	0,05	0,05	0,0
p14.sms	40	1,68	0,39	1,18	0,98	0,0	0,0
p15.sms	40	3,62	0,0	1,21	0,34	0,34	0,0
p16.sms	40	4,94	0,0	1,08	1,84	0,60	0,0
p17.sms	60	8,01	0,29	1,14	2,21	0,49	0,97
p18.sms	60	24,23	0,0	10,96	6,59	5,22	5,91
p19.sms	60	18,72	1,95	8,99	7,01	0,0	0,92
p20.sms	60	63,94	12,42	15,58	13,73	5,56	2,18
p21.sms	60	2,45	0,0	1,26	1,13	0,36	0,22
p22.sms	60	2,71	0,69	1,75	2,04	0,44	0,86
p23.sms	60	5,53	0,95	1,50	1,97	0,47	0,72
p24.sms	60	5,57	1,38	2,10	1,59	0,74	0,22
p25.sms	80	12,23	0,61	4,22	3,78	2,03	2,22
p26.sms	80	74,58	0,42	47,03	37,71	23,73	5,08
p27.sms	80	21,02	2,80	5,02	4,98	0,0	2,21
p28.sms	80	60,2	0,0	19,52	18,71	5,60	3,86
p29.sms	80	4,17	0,88	1,03	1,48	0,20	0,10
p30.sms	80	2,59	0,89	2,26	2,02	0,81	0,05
p31.sms	80	9,30	0,92	3,05	3,30	0,53	2,29
p32.sms	80	6,58	1,09	2,67	1,68	1,23	0,43
p33.sms	100	11,26	0,81	4,86	4,24	1,97	2,44
p34.sms	100	156,71	11,69	52,38	58,87	38,53	27,71
p35.sms	100	32,43	2,02	8,49	8,91	2,16	4,03
p36.sms	100	1098,32	42,86	296,64	246,22	219,33	215,97
p37.sms	100	3,04	0,12	1,34	1,50	0,48	0,58
p38.sms	100	5,42	0,58	2,55	2,88	0,0	0,66
p39.sms	100	11,28	0,68	3,23	3,90	1,50	1,09
p40.sms	100	13,15	1,32	6,24	6,17	1,95	1,09
Média		46,56	3,19	13,99	12,19	8,18	7,21

Tabela 5.6: Valores médios de desvio: conjunto de problemas teste C2.

	Memético	Mono-A	GRASP	GRASPR	RM	RME
Média	19,59	2,17	6,75	6,19	2,77	1,86

de vitórias os métodos de reinício com memória estruturada e Mono-A ficaram empatados com 20 vitórias cada um, considerando-se os casos onde ambos os métodos forneceram os mesmos valores. Para este conjunto de dados o algoritmo memético não conseguiu superar em nenhum caso os demais métodos.

Os resultados do algoritmo memético apresentados foram obtidos pelo próprio autor do método e fornecidos para comparação com os resultados de Carvalho (2002). Para estas análises o critério de parada utilizado por Carvalho foi baseado nas seguintes quantidades de soluções avaliadas:

Tabela 5.7: Número de soluções avaliadas: conjunto C2.

número de tarefas	20	40	60	80	100
avaliações (em milhões)	0.40	3.79	46.45	83.91	106.70

Note que estas quantidades de soluções avaliadas foram estimadas por Carvalho (2002) para o propósito de comparação com o algoritmo memético citado. Estes valores são bem maiores que os utilizados nas demais análises feitas neste trabalho.

5.4.3 Conjunto de problemas teste C3

Nesta seção apresentamos uma análise de desempenho de robutez dos métodos de reinício propostos. Desta forma decidimos agrupar os resultados em função de quatro fatores considerados relevantes: coeficiente de variação dos tempos de preparação, fator de severidade dos tempos de preparação, fator de atraso das datas de entrega e fator de dispersão das datas de entrega. Os resultados também foram divididos de acordo com o tamanho dos problemas. Todas as tabelas apresentadas mostram o desvio percentual em relação ao melhor resultado conhecido.

Na Tabela 5.8, comparamos o desempenho dos métodos em relação ao coeficiente de variação dos tempos de preparação. Os resultados foram divididos em duas categorias: coeficientes de variação baixos, $C_v \approx 0,15$, e coeficientes de variação médios, $C_v \approx 0,3$. Problemas com alto coeficiente de variação não foram gerados neste conjunto de dados.

Como podemos notar o desempenho dos métodos de reinício com memória (RM) e memória estruturada (RME) permaneceu praticamente invariável frente às duas

Tabela 5.8: Conjunto C3: Variação do fator C_v

n	C_v	Memético	GRASP	GRASPR	RM	RME
50	baixo	25,31	6,87	2,49	0,63	0,40
	médio	14,66	3,52	1,91	0,67	0,56
75	baixo	33,14	6,01	2,45	0,48	0,57
	médio	13,39	3,34	1,33	0,61	0,69
100	baixo	33,51	3,54	3,80	2,32	0,05
	médio	20,93	2,78	1,68	0,64	0,81
150	baixo	51,34	3,04	2,97	3,07	0,00
	médio	3,94	2,54	1,88	2,10	1,90

classes de problemas. Já os métodos GRASP, GRASPR e Memético apresentaram uma variação considerável de desempenho em função de C_v.

A seguir, na Tabela 5.9, apresentamos os resultados dos métodos em função do parâmetro η. Neste caso os problemas foram agrupados em três categorias: fator de severidade baixo, $\eta << 1$, médio, $\eta \approx 1$ e alto, $\eta >> 1$. Novamente podemos notar o bom desempenho dos métodos de reinício, principalmente o método de reinício com clusters. Neste caso os métodos Memético, GRASP e GRASPR apresentaram uma tendência de queda de desempenho com o aumento do fator de severidade. Note ainda que esta queda de desempenho foi mais acentuada para o algoritmo memético.

Tabela 5.9: Conjunto C3: Variação do fator η

n	η	Memético	GRASP	GRASPR	RM	RME
	baixo	14,66	3,52	1,91	0,67	0,56
50	médio	17,03	5,09	1,52	0,84	0,11
	alto	33,58	8,65	3,45	0,42	0,68
	baixo	13,39	3,34	1,33	0,61	0,69
75	médio	23,99	5,36	1,88	0,43	0,70
	alto	42,29	6,65	3,02	0,53	0,44
	baixo	20,93	2,78	1,68	0,64	0,81
100	médio	31,93	3,47	3,64	1,94	0,03
	alto	35,09	3,61	3,95	2,70	0,06
	baixo	3,94	2,54	1,88	2,10	1,90
150	médio	51,95	3,56	3,26	3,77	0,0
	alto	50,44	2,25	2,53	2,01	0,0

Vamos apresentar agora os resultados referentes ao comportamento dos métodos em relação ao fator de atraso τ. Novamente três categorias de problemas foram analisadas: fator de atraso baixo, $\tau < 0,4$, médio, $0,4 < \tau < 0,7$ e alto, $\eta > 0,7$. Como podemos notar, os métodos de reinício com memória e reinício com memória

estruturada conseguiram manter um bom desempenho frente às variações deste fator. Vale lembrar que neste caso não podemos concluir que os demais métodos tiveram um aumento de desempenho com o aumento de τ. Isto porque, assim como mostrado na seção 5.4.2, pequenas variações de atraso total em problema com baixo fator de atraso podem gerar grandes diferenças de desvio em relação à melhor solução. Os resultados referentes à esta análise podem ser vistos na Tabela 5.10.

Tabela 5.10: Conjunto C3: Variação do fator τ

n	τ	Memético	GRASP	GRASPR	RM	RME
50	baixo	36,01	6,24	2,99	1,14	0,73
	médio	19,97	7,75	2,64	0,51	0,53
	alto	10,37	2,91	1,21	0,34	0,09
75	baixo	41,47	6,93	2,44	0,76	1,04
	médio	23,02	4,63	2,03	0,41	0,50
	alto	16,59	3,99	1,77	0,45	0,34
100	baixo	41,01	3,36	2,0	1,07	0,95
	médio	28,51	3,71	4,10	2,45	0,08
	alto	21,26	2,86	3,03	1,67	0,01
150	baixo	31,67	3,04	1,36	2,89	1,87
	médio	53,65	3,34	4,21	3,84	0,03
	alto	21,30	2,23	2,25	1,51	0,0

Finalmente, na Tabela 5.11, apresentamos os resultados dos métodos em função do fator de dispersão das datas de entrega. Novemente podemos verificar a invariância do desempenho do método em função deste fator.

Tabela 5.11: Conjunto C3: Variação do fator R

n	R	Memético	GRASP	GRASPR	RM	RME
50	baixo	33,58	6,08	2,70	0,61	0,33
	médio	21,25	5,01	2,46	0,84	0,48
	alto	10,44	6,18	1,73	0,48	0,55
75	baixo	34,30	4,87	2,80	0,49	0,48
	médio	30,27	4,39	1,58	0,65	0,71
	alto	15,10	6,09	1,85	0,43	0,65
100	baixo	35,75	2,14	2,85	1,71	0,09
	médio	26,98	3,36	2,29	0,97	0,72
	alto	24,69	4,31	4,12	2,50	0,09
150	baixo	68,06	2,47	1,78	2,50	0,0
	médio	10,26	1,11	1,36	1,01	0,03
	alto	30,56	5,20	4,77	5,14	1,87

Uma análise geral dos resultados obtidos nesta seção revela a robustez dos métodos de reinício que utilizam memória. Como podemos notar, o comportamento destes métodos foi muito bom para todas as combinações de fatores apresentadas. O método de múltiplos reinícios com clusters também obteve um bom desempenho em relação ao número de vitórias: obteve o melhor resultado em 167 dos 252 problemas avaliados. Já o método de múltiplos reinícios com memória obteve os melhores resultados em 77 problemas. O método GRASP reativo conseguiu os melhores resultados em 16 problemas.

5.4.4 Conjunto de problemas teste C4

Nesta seção apresentamos uma série de tabelas contendo os resultados comparativos entre os métodos de reinício apresentados e o algoritmo memético proposto por França *et al.* (2001). Os problemas utilizados para teste foram gerados de acordo com os procedimentos descritos por França *et al.* (2001), onde foram utilizados três valores diferentes para τ e R: $\tau = \{0,2; 0,6; 1,0\}$ e $R = \{0,2; 0,6; 1,0\}$, totalizando 9 combinações possíveis para cada tamanho de problema $n = \{20, 40, 60, 80, 100\}$.

Nas Tabelas 5.12 a 5.20 apresentamos os valores de desvio médio em relação ao melhor resultado conhecido (Equação 5.2). Como podemos notar, os métodos de reinício com memória superam os demais em quase todos os casos analisados.

Tabela 5.12: Conjunto C4: $\tau = 0,2$ $R = 0,2$

n	Memético	GRASP	GRASPR	RM	RME
20	7,77	0,54	0,00	0,00	0,00
40	35,98	3,20	3,23	0,57	1,14
60	67,24	5,69	5,09	0,30	1,43
80	85,79	8,58	9,36	1,27	3,74
100	100,45	6,07	8,01	0,45	3,25
Média	59,45	4,82	5,14	0,52	1,91

Tabela 5.13: Conjunto C4: $\tau = 0,2$ $R = 0,6$

n	Memético	GRASP	GRASPR	RM	RME
20	11,23	0,00	0,58	0,00	0,58
40	107,95	32,10	31,84	11,04	9,14
60	1078,65	107,39	130,90	56,29	37,62
80	585,51	37,37	53,09	13,98	24,14
100	1254,42	78,18	83,03	11,95	7,83
Média	607,55	51,01	59,89	18,65	15,86

Tabela 5.14: Conjunto C4: $\tau = 0,2$ $R = 1,0$

n	Memético	GRASP	GRASPR	RM	RME
20	23,05	29,81	32,94	9,23	6,44
40	10,73	5,15	2,08	3,50	0,19
60	0,00	0,00	0,00	0,00	0,00
80	0,00	0,00	0,00	0,00	0,00
100	0,00	0,00	0,00	0,00	0,00
Média	6,76	6,99	7,00	2,55	1,33

Tabela 5.15: Conjunto C4: $\tau = 0,6$ $R = 0,2$

n	Memético	GRASP	GRASPR	RM	RME
20	1,29	0,00	0,02	0,00	0,02
40	9,69	2,40	2,57	0,13	0,29
60	17,29	3,63	3,52	0,63	1,71
80	23,15	3,26	2,60	0,35	1,05
100	21,55	2,81	2,69	0,00	1,74
Média	14,59	2,42	2,28	0,22	0,96

Tabela 5.16: Conjunto C4: $\tau = 0,6$ $R = 0,6$

n	Memético	GRASP	GRASPR	RM	RME
20	1,65	0,00	0,11	0,00	0,00
40	8,85	2,37	2,45	0,88	0,18
60	20,86	4,09	4,00	0,89	0,66
80	28,56	5,97	5,22	2,16	1,51
100	29,69	5,42	5,52	1,32	2,94
Média	17,92	3,57	3,46	1,05	1,06

Tabela 5.17: Conjunto C4: $\tau = 0,6$ $R = 1,0$

n	Memético	GRASP	GRASPR	RM	RME
20	3,06	0,00	0,00	0,00	0,00
40	5,31	3,79	3,31	0,56	0,33
60	14,95	3,81	3,99	1,41	0,31
80	23,41	4,29	4,58	1,21	1,07
100	29,45	4,63	4,90	1,17	2,62
Média	15,24	3,30	3,36	0,87	0,87

Tabela 5.18: Conjunto C4: $\tau = 1,0$ $R = 0,2$

n	Memético	GRASP	GRASPR	RM	RME
20	0,25	0,00	0,00	0,00	0,00
40	3,50	0,74	0,58	0,20	0,08
60	7,54	1,46	1,34	0,47	0,35
80	11,31	1,95	1,65	0,51	0,60
100	11,15	1,82	1,93	0,33	0,49
Média	6,75	1,19	1,10	0,30	0,31

Tabela 5.19: Conjunto C4: $\tau = 1,0$ $R = 0,6$

n	Memético	GRASP	GRASPR	RM	RME
20	1,46	0,05	0,05	0,05	0,05
40	4,69	1,26	1,30	0,04	0,11
60	7,78	1,74	1,62	0,81	0,17
80	11,13	2,15	2,31	0,39	0,42
100	12,95	2,14	1,99	0,33	0,82
Média	7,60	1,47	1,45	0,33	0,32

Tabela 5.20: Conjunto C4: $\tau = 1,0$ $R = 1,0$

n	Memético	GRASP	GRASPR	RM	RME
20	0,34	0,00	0,00	0,00	0,00
40	4,35	1,50	1,54	0,64	0,38
60	7,94	1,95	1,89	0,79	0,24
80	11,31	2,16	1,86	0,70	0,54
100	14,30	2,47	2,57	0,48	0,62
Média	7,65	1,62	1,57	0,52	0,36

Capítulo 6

Conclusões

Neste trabalho propusemos um método de múltiplos reinícios para o problema de seqüenciamento de tarefas em uma máquina com datas de entrega e tempos de preparação dependentes da seqüência. Diferente da maioria dos métodos GRASP propostos na literatura, que em geral utilizam a função objetivo para guiar o processo construtivo, as versões aqui apresentadas utilizam uma regra de despacho como função gulosa para selecionar os elementos da lista de candidatos. Como foi possível verificar, a regra de despacho ATCS mostrou-se bastante apropriada para este propósito.

Outro diferencial das abordagens propostas foi o tipo de vizinhança utilizada. Em geral, métodos que tratam de problemas de seqüenciamento utilizam vizinhanças de troca e inserção. Neste trabalho optamos por utilizar uma vizinhança *3-opt*, que mostrou-se bastante adequada para a estrutura do problema e, em geral, superior às duas citadas anteriormente. Este resultado também é verificado por Carvalho (2002). O teste de redução de vizinhança proposto para a vizinhança *3-opt* também mostrou-se bastante eficiente, chegando em alguns casos a reduzir o tamanho da vizinhança em 90%, sem comprometer o poder de exploração do método. Trabalhos futuros podem ser feitos para tentar estabelecer uma relação entre os fatores de severidade e covariância dos tempos de preparação e o parâmetro que controla a redução da vizinhança.

Uma constatação que ficou evidente neste trabalho foi a superioridade dos métodos que utilizam memória frente aos métodos de reinício convencionais. O procedimento adaptativo proposto para o controle da ênfase de memória, em conjunto com o fator de aleatorização, também mostrou-se bastante eficiente. Esta adaptação do controle reativo, proposto por Prais e Ribeiro (2000b), forneceu resultados bastante satisfatórios e possibilitou um controle de intensidade independente de parâmetros extras, tal como a taxa aumento ou decréscimo do fator de ênfase de memória.

Quatro versões diferentes do método foram propostas, sendo que em duas delas

propomos a utilização de memória adaptativa. Um aspecto interessante abordado neste trabalho foi a utilização de uma população dividida em regiões (clusters). Este tipo de estrutura mostrou-se eficiente para a diversificação do processo construtivo, conseguindo obter bons resultados em relação ao procedimento que não considera a divisão da população. Lembramos ainda que existem outras técnicas que podem ser utilizadas para a delimitação das regiões de definição dos clusters. Uma alternativa que pode ser explorada é a utilização de soluções de referência dinâmicas (centróide) para delimitar estas regiões. Neste caso o referencial do cluster pode ser adaptado regularmente, em função das soluções contidas dentro do cluster.

Embora os resultados computacionais tenham revelado um ganho percentual do método com população estruturada em relação ao esquema proposto por Fleurent e Glover (1999), ressaltamos que uma análise mais extensa deve ser feita para que possamos concluir a superioridade da abordagem proposta. Quanto ao desempenho, todas as versões de métodos de reinícios apresentadas mostraram-se bastante promissoras na obtenção de soluções de boa qualidade. A robustez dos métodos, frente às diversas classes de problemas analisadas, também pôde ser verificada. Para o conjunto de dados proposto por Rubin e Ragatz (C1) todas as quatro abordagens propostas apresentaram bons resultados. Já para os conjuntos C2, C3 e C4, os métodos com memória apresentaram, em geral, os melhores resultados.

Referências Bibliográficas

Adams, J., Balas, E., D., Z., 1988. The shifting bottleneck procedure for job shop scheduling. Management Science 34e, 391–401.

Allahverdi, A., Gupta, J. N. D., Aldowaisan, T., 1999. A review of scheduling research involving setup considerations. The International Journal of Management Science 27, 219–239.

Armentano, V. A., Mazzini, R., 2000. A genetic algorithm for scheduling on a single machine with set-up times and due dates. Production Planning & Control 11, n. 7, 713–720.

Binato, S., Hery, W. J., Loewenstern, D. M., Resende, M. G. C., 2000. A greedy randomized adaptive search procedure for job shop scheduling. Essays and Surveys on Metaheuristics , 1–19.

Bresina, J. L., 1996. Heuristic-biased stochastic sampling. Em: Proccedings of the AAAI-96. Vol. 1. pp. 271–278.
URL `citeseer.nj.nec.com/bresina96heuristicbiased.html`

Carlier, J., Pinson, E., 1989. An algorithm for solving the job-shop problem. Management Science 35, 164–176.

Carvalho, R. M., 2002. Minimização da soma dos atrasos e do makespan para o problema de seqüênciamento de tarefas em uma máquina com tempos de preparação dependentes da seqüência. *Dissertação de mestrado em andamento*, Departamento de Engenharias e Sistemas - Unicamp.

Chen, Z. L., Powell, W. B., 1999. Solving parallel machine scheduling problems by column generation. Informs Journal on Computing 11 (1), 78–94.

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

Feo, T., Resende, M., 1989. A probabilistic heuristic for a computationally difficult set covering problem. Operations Research Letters 8, 67–71.

Feo, T., Resende, M., 1995. Greedy randomized adaptive search procedures. Journal of Global Optimization 2, 1–27.

Festa, P., Resende, M., 2001. GRASP: An annotated bibliography. Em: Ribeiro, C., Hansen, P. (Eds.), Essays and Surveys on Metaheuristics. Kluwer Academic Publishers, pp. 325–367.

Fleurent, C., Glover, F., 1999. Improved constructive multistart strategies for the quadratic assignment problem using adaptive memory. Informs Journal on Computing 11, 198–204.

França, P. M., Mendes, A., Moscato, P., 2001. A memetic algorithm for the total tardiness single machine scheduling problem. European Journal of Operational Research 132, 224–242.

Glover, F., 1977. Heuristics for integer programming using surrogate constraints. Decision Sciences 8, 156–166.

Glover, F., 1997. A template for scatter search and path relinking. in Lecture Notes in Computer Science , 13–54.

Glover, F., Laguna, M., Martí, R., 2000. Fundamentals of scatter search and path relinking. Control and Cybernetics 39, n.3, 653–684.

Hu, X., Shonkwiler, R., Spruill, M. C., 1994. Random restarts in global optimization. Research report, Georgia Institute of Technology, Atlanta .

Laguna, M., Glover, F., 1993. Integrating target analysis and tabu search for improved scheduling systems. Expert Systems with Applications 6, 287–297.

Lee, Y. H., Bhaskaran, K., Pinedo, M., 1997. A heuristic to minimize the total weighted tardiness with sequence-dependent setups. IIE Transactions 29, 45–52.

Lee, Y. H., Pinedo, M., 1997. Scheduling jobs on parallel machines with sequence-dependent setup times. European Journal of Operational Research 100, 464–474.

Lin, S., Kernighan, B. W., 1973. An effective heuristic for the traveling-salesman problem. Operations Research 21, 498–516.

Michalewicz, Z., 1996. Genetic Algorithms + Data Structure = Evolution Programs, 3rd Edition. Springer-Verlag.

Prais, M., Ribeiro, C. C., 2000a. Parameter variation in GRASP procedures. Investigación Operativa , 9:1–20.

Prais, M., Ribeiro, C. C., 2000b. Reactive GRASP: An application to a matriz decomposition problem in tdma traffic assignment. Informs Journal on Computing 12, 164–176.

Ragatz, G. L., 1993. A branch-and-bound method for minimum tardiness sequencing on a single processor with sequence dependent setup times. In Proceedings: Twenty-fourth annual meeting of the Decision Sciences Institute , 1375–7.

Raman, N., Rachamadugu, R. V., Talbot, F. B., 1989. Real time scheduling of an automated manufacturing center. European Journal of Operational Research 40, 222–242.

Rubin, P. A., Ragatz, G. L., 1995. Scheduling in a sequence dependent setup environment with genetic search. Computers and Operations Research 22(1), 85–99.

Tan, K. C., Narasimhan, R., 1997. Minimizing tardiness on a single processor with sequence-dependent setup times: A simulated annealing approach. OMEGA – International Journal of Management Science 25 (6), 619–634.

Tan, K. C., Narasimhan, R., Rubin, P. A., Ragatz, G. L., 2000. A comparison of four methods for minimizing total tardiness on a single processor with sequence dependent setup times. The International Journal of Management Science 28, 313–326.

Torn, A., Zilinskas, A., 1987. Global Optimization. Springer-Verlag.

Vepsalainen, A., Morton, T. E., 1987. Priority rules for jobshops with weighted tardiness costs. Management Science 33, 1035–1047.

Vollmam, T. E., Berry, W. L., Whybark, D. C., 1988. Manufacturing Planning and Control Systems, 2nd Edition. Dow Jones-Irwin.