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

ESTUDO DE META-HEURÍSTICAS POPULACIONAIS PARA A PROGRAMAÇÃO DE MÁQUINAS PARALELAS COM TEMPOS DE PREPARAÇÃO DEPENDENTES DA SEQUÊNCIA E DATAS DE ENTREGA

RENATA MAZZINI

Orientador:
Prof. Dr. Vinícius Amaral Armentano

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

- OUTUBRO 1998 -

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

M459e

Mazzini, Renata
Estudo de meta-heurísticas populacionais para a programação de máquinas paralelas com tempos de preparação dependentes da sequência e datas de entrega. / Renata Mazzini.--Campinas, SP: [s.n.], 1998.

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

1. Programação heurística. 2. Planejamento da produção. 3. Multiprocessadores. 4. Algoritmos genéticos. I. Armentano, Vinícius Amaral. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

"O silêncio enchia a cela. Rubachov ouvia apenas o ranger de seus sapatos nos ladrilhos. Seis passos e meio para a porta, onde deviam vir buscá-lo, seis passos e meio para a janela, atrás da qual caia a noite. Em breve, tudo estaria acabado. Mas quando se interrogava: Por que mesmo estás morrendo? — Não achava resposta."

"Na adolescência, tivera realmente a intenção de estudar astronomia, e depois, durante quarenta anos, estivera fazendo outra coisa. Por que o promotor público não lhe perguntara: "Acusado Rubachov, que me diz do infinito?" Ele não teria respondido — e eis que era esta a fonte verdadeira de sua culpa...Poderia haver outra maior?"

Arthur Koestler, "O Zero e o Infinito"

"Se uma conclusão não é poeticamente equilibrada, não pode ser cientificamente verdadeira."

Isaac Asimov, "Os Robôs do Amanhecer"

AGRADECIMENTOS

Agradeço a todos aqueles que, direta ou indiretamente, colaboraram para a realização deste trabalho. Em especial:

- ao Professor Vinícius A. Armentano pela orientação dedicada e ajuda fundamental em todas as etapas;
- à CAPES e à FAPESP pelo apoio financeiro;
- aos professores e funcionários do DENSIS, por todo apoio e estímulo;
- à minha família pelo interesse constante e pelo conforto emocional;
- aos amigos Vitória, Fran, Cíntia, Débora, Ernesto, Cris, Pablo, Regina, Jeanne, Jussara, Gonzaga, e muitos outros, por estarem sempre ao meu lado;
- ao Ricardo, pelo apoio moral e logístico;
- à minha segunda família, Daniele, Ronald e Raquel por todo o amor, apoio e confiança.
- "Gracias a la vida, que me hay dado tanto..."

RESUMO

Neste trabalho estuda-se a aplicação de meta-heurísticas populacionais em problemas de programação da produção. O problema abordado consiste da minimização do atraso total na programação de tarefas em máquinas paralelas idênticas onde se consideram tempos de preparação de máquina dependentes da seqüência de processamento e datas de entregas distintas. São apresentadas duas implementações de Algoritmos Genéticos e quatro implementações de Busca por Espalhamento. Testes computacionais extensos foram realizados com essas implementações sobre um conjunto de problemas gerados aleatoriamente. Os resultados obtidos possibilitaram a realização de análises comparativas do desempenho e do comportamento das versões implementadas dessas duas meta-heurísticas quando aplicadas ao problema de programação da produção estudado.

Palavras-chave: Programação da Produção, Máquinas Paralelas, Atraso, Tempos de Preparação Dependentes da Seqüência, Meta-heurísticas, Algoritmos Genéticos, Busca por Espalhamento.

ABSTRACT

The application of population-based meta-heuristics on scheduling problems is studied in this work. The problem which is addressed is the minimization of total tardiness when jobs are scheduled on identical parallel machines with sequence dependent set-up times and distinct due dates. Two implementations of Genetic Algorithms and four implementations of Scatter Search are presented. These implementations were extensively tested over a set of randomly generated problems. The obtained results were used to compare the performance and the behavior of the implemented versions of these two meta-heuristics when applied to the scheduling problem studied.

Keywords: Scheduling, Parallel Machines, Tardiness, Sequence Dependent Setup Times, Meta-heuristics, Genetic Algorithms, Scatter Search.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

INTRODUÇÃO

Dentro do contexto econômico mundial, os sistemas de produção têm enfrentado uma questão bastante complexa. Não basta adquirir ou desenvolver a tecnologia direcionada à produção em si mas é preciso desenvolver estratégias de planejamento e controle de produção que garantam altos níveis de produtividade. Tendo em vista a globalização da economia, com a formação de mercados comuns na maioria dos continentes, esse tema é bastante atual. Entretanto, a idéia de se planejar a produção com vistas a utilizar racionalmente os recursos disponíveis evitando perdas e, portanto, aumentando os ganhos possíveis, não é uma idéia nova. É interessante observar que, desde a substituição da produção artesanal pela produção em massa, vários paradigmas de gerenciamento surgiram como forma de atacar o problema da produtividade (MITCHELL, 1991). Sem considerar o impacto social de determinadas estratégias propostas ao longo da evolução desses paradigmas, o fato é que os sistemas de produção de grande porte e a necessidade de otimizar a produção surgiram quase que simultaneamente.

Apesar de ser um problema antigo e bastante pesquisado, o grande volume de trabalhos realizados e a qualidade dos resultados já obtidos não garantem que a questão da produtividade tenha sido resolvida. Isso se deve principalmente à falta de homogeneidade dos sistemas de produção. Mesmo que dois sistemas estejam competindo pela mesma fatia de mercado e produzindo exatamente os mesmos produtos, eles podem apresentar grandes diferenças entre si. De acordo com o planejamento feito no passado e o histórico administrativo de cada sistema, sua configuração atual em termos de máquinas, mão de obra e restrições operacionais é praticamente única. Esse fato dificulta enormemente a criação de estratégias gerais de planejamento e programação da produção. Para viabilizar o estudo relacionado a problemas de produção e o desenvolvimento de métodos para resolvê-los, os sistemas são decompostos em partes menores que possam ser modeladas com um grau razoável de generalização, criando problemas de otimização mais tratáveis. Essa decomposição pode ser feita, a princípio, considerando-se a divisão hierárquica do processo decisório dos sistemas de produção. Entretanto, ao mesmo tempo que a necessidade de trabalhar com problemas menores e mais específicos estimula a quebra do sistema em partes, corre-se o risco de ignorar ou subestimar a influência de algumas das partes sobre as outras. Portanto, no estudo de problemas de produção, a estrutura e estabilidade da hierarquia de

tomada de decisões criada em cada sistema, assim como a eficiência da troca de informações entre os vários níveis, influenciam diretamente o sucesso da decomposição do mesmo e ajudam a minimizar os efeitos negativos de não considerar o problema completo de uma só vez.

Uma forma de divisão organizacional bastante conhecida (GELDERS e VAN WASSENHOVE, 1981) estabelece uma hierarquia de três níveis definidos de acordo com a abrangência de suas decisões ao longo do tempo. No topo dessa hierarquia está o nível estratégico de planejamento de longo prazo no qual devem ser definidas as diretrizes mestras do sistema e seus objetivos principais. Nesse nível pode-se estabelecer metas de crescimento e desenvolvimento a serem operacionalizadas pelos níveis inferiores.

O segundo nível de decisões corresponde ao planejamento de médio prazo, durante o qual é preciso adequar a aquisição e a utilização de recursos agregados de forma a cumprir as metas estabelecidas no nível estratégico. O último nível e talvez o mais crítico é o nível operacional, onde decisões são tomadas para aplicação em curto prazo, muitas vezes numa base diária, ou de turno a turno de trabalho. O fator complicante nesse nível de tomada de decisões é a grande diversidade de recursos disponíveis, a influência direta das restrições tecnológicas e administrativas, além da exigência de respostas rápidas e precisas.

As interfaces entre os níveis de decisão são tão importantes quanto a sua própria definição (MITCHELL, 1991). As vias de informação devem ser sempre bidirecionais permitindo que durante o processo decisório também ocorra um aprendizado constante sobre o sistema e sua ligação com o mercado. Para viabilizar a realimentação de informações entre os níveis definidos é preciso usar medidas de desempenho que sejam capazes de traduzir a eficiência e a eficácia de cada atividade realizada tendo em vista os objetivos gerais do sistema. Essas medidas possuem um papel extremamente importante dentro da hierarquia de decisões pois permitem que se extraia toda e somente a informação necessária evitando que um excesso de dados atrapalhe o processamento da informação. Dessa forma, o processo de reavaliação de decisões torna-se rápido, diminuindo o tempo de resposta do sistema às perturbações a que está sujeito. Claramente, a escolha de medidas representativas não é uma questão trivial pois elas devem refletir fatores internos e externos ao sistema como custos de produção, eficiência da utilização de recursos, segurança para mão de obra e satisfação do cliente, entre outros.

Talvez a maior quantidade de medidas de desempenho diferentes possa ser encontrada no nível operacional, pois elas determinam as diretrizes através das quais deve-se organizar a realização das atividades nesse nível. De acordo com o que se espera em termos de custo e tempo de produção, deve-se estabelecer exatamente o que será produzido, em que ordem e quais os instantes iniciais e finais de processamento de cada item. A tomada de decisões referentes a essas atividades é chamada de programação ("scheduling") de produção e, na literatura, encontram-se vários critérios de desempenho que servem como base para sua

realização. Entre eles pode-se citar como os mais comuns: minimização do instante de término dos ítens programados ("makespan"), minimização do atraso total (ou atraso médio) e minimização do tempo de permanência de cada item no sistema (tempo de fluxo).

A teoria clássica de programação de produção provê uma grande quantidade de modelos matemáticos para os problemas de otimização encontrados no nível operacional. O estudo desses modelos e das propriedades dos problemas possibilitaram a obtenção de muitos resultados teóricos significativos, bem como o desenvolvimento de uma série de métodos exatos, aproximados e heurísticos para a resolução dos mesmos. Apesar da pesquisa em programação da produção ter sido iniciada formalmente há praticamente cinqüenta anos, a aplicação de seus modelos e métodos na prática ainda é pequena (MACCARTHY e LIU, 1993). Um dos fatores apontados como determinantes dessa situação é que as hipóteses assumidas nos estudos teóricos não são suficientemente próximas das realidade. Dentre essas hipóteses pode-se citar: máquinas disponíveis continuamente durante todo o horizonte de planejamento e que não estejam sujeitas a quebra; tempos de preparação de máquina independentes da seqüência de processamento; e valores fixos e conhecidos *a priori* para os parâmetros do problema (tempos de processamento e datas de entrega, por exemplo). Portanto, é preciso que a pesquisa teórica avance no sentido de englobar restrições tecnológicas presentes nos sistemas de produção reais. Entretanto, isso pode levar a modelos muito complexos para serem tratados por métodos exatos, visto que, mesmo considerando hipóteses ideais, os modelos já são bastante desafiadores. Sendo assim, deve-se investir em novos métodos de resolução de problemas capazes de obter boas soluções que, mesmo não sendo ótimas, representem um ganho com relação ao estado atual do sistema, aumentando sua produtividade e, como conseqüência, sua competitividade.

Tendo sido determinado o modelo segundo o qual é descrito o nível operacional de um determinado sistema de produção, a escolha do método de solução a ser utilizado dependerá principalmente da razão entre a qualidade da solução gerada pelo método e o tempo gasto para chegar nessa solução. Nesse nível, a maioria dos problemas é intratável, ou seja, são problemas para os quais é improvável que se consiga desenvolver um algoritmo exato que possa ser executado em tempo razoável. Além disso, o fator tempo é extremamente importante visto que a programação da produção é uma atividade de curto prazo e esperar a execução de um método lento pode causar prejuízos diretos e indiretos ao sistema, mesmo que a solução obtida seja ótima. Para acelerar e, muitas vezes, até viabilizar a obtenção de soluções é preciso lançar mão de métodos heurísticos. Esses métodos, quando bem desenvolvidos e adaptados aos problema que se deseja resolver, são capazes de apresentar soluções de boa qualidade (embora subótimas) em tempo compatível com a necessidade de rapidez presente no nível operacional de um sistema de produção. A seguir analisam-se os diferentes tipos de métodos heurísticos desenvolvidos para a resolução de problemas de otimização combinatória e que podem ser aplicados para resolver os problemas de

programação de produção. Pode-se dividir essas heurísticas em três classes que diferem basicamente na forma como exploram o espaço de soluções dos problemas.

A primeira classe de heurísticas engloba os métodos construtivos. Partindo-se de um conjunto de dados que define o problema, constrói-se uma solução escolhendo os elementos a serem inseridos (ou retirados dela) de acordo com a avaliação dessa escolha através da medida de desempenho adequada ao problema. A essa classe pertencem desde as heurísticas mais simples, denominadas regras de despacho, até heurísticas complexas de construção que envolvam, além da inserção de novos elementos, um rearranjo dos elementos previamente inseridos em função da escolha tomada. Heurísticas construtivas, na sua maioria, não são utilizadas isoladamente pois não possuem mecanismos de avaliação geral das decisões tomadas. Poucas heurísticas construtivas consideram o efeito de cada decisão na solução parcial, mas de forma limitada.

A segunda classe de heurísticas vem ao encontro da necessidade de, tendo uma solução completa para o problema, modificá-la sistematicamente avaliando o efeito total de cada modificação possível. Essas heurísticas são chamadas de heurísticas de Busca Local ou Busca em Vizinhança (PAPADIMITRIOU e STEIGLITZ, 1982 e EVANS, 1987). Dada uma solução inicial x, constrói-se uma vizinhança V(x) que contém todas as soluções alcançáveis através de uma regra de movimento aplicada a todos os elementos que compõe x. Dessa vizinhança, escolhe-se uma solução x' que possua uma avaliação melhor que x, substitui-se x por x' e continua-se o processo até que se encontre um ótimo local, ou seja, uma solução que possua uma avaliação melhor que a de qualquer outra solução em sua vizinhança. Claramente, a eficiência das heurísticas de busca local depende da escolha da solução inicial, ou seja, do processo construtivo escolhido para obtenção da primeira solução a ser avaliada. Uma vez tendo chegado ao ótimo local, essas heurísticas param e não são capazes de explorar novas regiões do espaço de busca. Seria interessante poder usar um método capaz de avaliar sistematicamente o espaço de soluções buscando melhorias mas permitindo a escolha estratégica de soluções com avaliação mais pobre que as já encontradas, na tentativa de escapar da otimalidade local. Métodos que possuem esse comportamento pertencem à terceira classe, a das meta-heurísticas. Mesmo não garantindo otimalidade global, esses métodos podem encontrar uma grande quantidade de ótimos locais, explorando o espaço de soluções de forma menos limitada. O número de meta-heurísticas freqüentemente usadas é pequeno pois esses métodos são pouco específicos com relação ao problema em estudo, consistindo de estratégias genéricas de exploração de espaços de busca. Isso faz com que cada meta-heurística possa ser adaptada para o problema que se deseja resolver, resultando em inúmeras aplicações e variações. Ultimamente, alguns conjuntos de estratégias básicas de meta-heurísticas diferentes vêm sendo combinados gerando métodos híbridos que agregam as vantagens que cada método já apresentava isoladamente e tentam superar os pontos fracos de cada um deles.

Cada meta-heurística apresenta características próprias mas pode-se dividir essa classe de métodos em duas grandes subclasses. A primeira delas é a subclasse dos métodos que exploram uma vizinhança a cada iteração, alterando tanto a vizinhança quanto a forma de explorá-la de acordo com suas estratégias e escolhendo apenas um elemento dessa vizinhança a cada passo. Esse tipo de varredura do espaço de busca gera um caminho de soluções, ou uma linha de busca, obtida pela transição de uma solução para outra de acordo com os movimentos permitidos pela meta-heurística. Dentro dessa subclasse de métodos pode-se citar a Busca Tabu (GLOVER, 1995 e DÍAZ et al, 1996) e o "Simulated Annealing" (KIRKPATRICK *et al.*, 1983 e EGLESE, 1990). Mesmo quando implementados em paralelo, cada ramificação da busca continua a determinar seu caminho através da obtenção seqüencial de soluções.

Em contraposição a esses métodos, existem aqueles que exploram uma população de soluções a cada iteração. Esses métodos fazem parte da segunda subclasse de meta-heurísticas e suas estratégias de busca são capazes de explorar várias regiões do espaço de soluções de cada vez. Dessa forma, ao longo das iterações não se constrói um caminho único de busca pois novas soluções sempre são obtidas através de combinações de soluções anteriores. Nessa subclasse estão os Algoritmos Genéticos (GOLDBERG, 1989 e MICHALEWICZ, 1992), a Busca por Espalhamento (do inglês "Scatter Search", GLOVER, 1994a) e os Algoritmos Meméticos (MOSCATO, 1989 e DÍAZ *et al.*, 1996), que mesclam características de busca dos Algoritmos Genéticos com técnicas de busca local.

O objetivo deste trabalho é estudar o comportamento e o desempenho das meta-heurísticas populacionais Algoritmos Genéticos e Busca por Espalhamento na resolução de problemas de programação da produção. O problema escolhido como caso de teste é a programação de tarefas (ou ítens) em máquinas paralelas idênticas minimizando o tempo total de atraso do término dessas tarefas com relação a suas datas de entrega. Considera-se que existem tempos de preparação de máquina entre duas tarefas consecutivas e esse tempo depende da seqüência de processamento estabelecida.

A inclusão de datas de entrega e tempos de preparação de máquina é motivada pela necessidade de considerar condições encontradas na prática no estudo do problema. Mesmo com um alto grau de flexibilidade nas máquinas existem atividades como limpeza, transporte de matéria prima e produto acabado, ajustes de máquina, troca de ferramentas, entre outras, que acrescentam uma quantidade de tempo ao processo produtivo. Esse tempo adicional pode depender da seqüência em que os itens são processados e, neste caso, não pode ser ignorado na programação dos mesmos. As datas de entrega, por outro lado, podem representar restrições internas ou externas ao sistema. Quando essas datas representam compromissos assumidos com clientes, existem custos diretos e indiretos associados ao atraso no processamento de tarefas. Entre eles pode-se citar custos adicionais de transporte de produto, multas contratuais ou mesmo a perda de clientes. Em sistemas de produção de bens perecíveis

ou que utilizem reagentes químicos pode-se perceber o efeito interno do atraso. Quando cada item deve passar por várias etapas de produção as datas de entrega em uma etapa representam um instante de tempo em que os itens produzidos são requeridos em alguma etapa adiante. Nesse caso o atraso pode causar ociosidade em partes do sistema com possível perda de material (por exemplo, reagentes químicos) e, consequentemente, um aumento no custo de produção.

É importante observar que o problema descrito não é abordado de forma analítica. Os métodos estudados são aplicados a ele com a finalidade de responder questões relativas ao desempenho e ao comportamento dos métodos estudados. No Capítulo 1 é apresentada a descrição formal do problema de atraso em máquinas paralelas e de seu modelo matemático. Além disso, são consideradas equivalências entre esse problema e o Problema do Caixeiro Viajante. Ainda no Capítulo 1, apresenta-se o esquema de geração dos problemas de teste utilizados durante todo o trabalho, assim como algumas heurísticas utilizadas para a determinação de limitantes superiores. A última seção desse capítulo contém os resultados da aplicação de Enumeração Completa a problemas de teste de pequeno porte, que possibilitam um estudo das soluções do problema abordado.

No Capítulo 2 apresenta-se o desenvolvimento de duas versões de Algoritmos Genéticos para o problema em estudo. Os testes computacionais realizados com essas duas versões são apresentados no Capítulo 3.

A Busca por Espalhamento é apresentada no Capítulo 4, juntamente com uma versão implementada especialmente para o problema estudado neste trabalho e resultados de testes computacionais realizados com ela. Nos capítulos 5 e 6, apresentam-se duas versões alternativas para a Busca por Espalhamento e os resultados computacionais correspondentes a cada uma delas. A utilização estratégica de busca local na Busca por Espalhamento também é apresentada no Capítulo 6 como um refinamento dessa meta-heurística.

Finalmente, no Capítulo 7, apresentam-se as principais conclusões deste trabalho.

CAPÍTULO 1 - O PROBLEMA ABORDADO

1.1 - DESCRIÇÃO DO PROBLEMA

Considere um conjunto de tarefas $\mathbf{N} = \{J_1, J_2, \ldots, J_n\}$ e um conjunto de máquinas idênticas $\mathbf{M} = \{M_1, M_2, \ldots, M_m\}$, com $m < n$. Deseja-se programar a produção das tarefas de **N** nas máquinas de **M** de forma que a soma ponderada dos atrasos das tarefas em relação a suas datas de entrega seja a menor possível. Por simplicidade, a partir de agora, o problema será referenciado como PST (das palavras-chave em inglês Parallel machines, Setup times, e Tardiness).

São assumidas as seguintes hipóteses:

H1 → Cada tarefa $J_i \in \mathbf{N}$ é constituída de somente uma operação;

H2 → Os tempos de processamento das tarefas são fixos e conhecidos e não variam de máquina para máquina;

H3 → Existem tempos de preparação de máquina fixos e conhecidos antes do início do processamento de cada tarefa. Esses tempos são dependentes da seqüência de produção;

H4 → Qualquer máquina pode processar qualquer uma das tarefas;

H5 → As máquinas não estão sujeitas a quebras e estão disponíveis durante todo o horizonte de planejamento;

H6 → Os tempos de preparação de máquina incluem o tempo de transporte das tarefas até as máquinas;

H7 → Não são considerados os custos de armazenamento de produto acabado;

H8 → Todas as tarefas estão disponíveis para processamento no início do horizonte de planejamento.

H9 → Cada máquina pode processar somente uma tarefa de cada vez;

H10 → Não é permitido interromper o processamento de qualquer das tarefas, uma vez que tenha sido iniciado;

Para cada tarefa $J_i \in \mathbf{N}$ são definidos dois parâmetros: tempo de processamento (p_i) e data de entrega (d_i). Associa-se também a cada tarefa J_i as variáveis instante de início (s_i), instante de término (C_i) e o atraso no processamento ($T_i = \max\{0, C_i - d_i\}$). Os tempos de preparação de máquina são definidos para cada par de tarefas J_i e $J_j \in \mathbf{N}$, sendo denotados por S_{ij}. Note que não é feita nenhuma restrição quanto à simetria dos tempos de

preparação de forma que S_{ij} pode ser diferente de S_{ji} para qualquer par de tarefas J_i e J_j. Com a finalidade de considerar o tempo de preparação inicial em cada máquina, cria-se uma tarefa fictícia J_0 que possui $p_0 = d_0 = 0$. Dessa forma, os valores de S_{0j} indicam o tempo necessário para iniciar a seqüência de produção de uma máquina com a tarefa J_j.

Um programa de produção **S** é caracterizado pela seqüência de produção de cada uma das máquinas de **M**. Esse programa é considerado factível se as hipóteses H1 a H10 forem satisfeitas. O custo de atraso é calculado como:

$$f(\mathbf{S}) = \sum_{i=1}^{n} T_i \qquad (1.1)$$

Um programa de produção **S*** é ótimo se ele minimiza o valor de f(**S**) para todos os programas **S** factíveis.

1.2 - MODELO MATEMÁTICO

O PST pode ser modelado como um problema de Programação Linear Inteira Mista após ser feita a linearização da função objetivo como sugerido em (BLAZEWICZ, DROR e WEGLARZ, 1991). Em (GUINET, 1993) encontra-se o modelo do PST, já linearizado, mostrado a seguir.

Seja x_{ij}^k uma variável que indique precedência imediata entre tarefas, ou seja, $x_{ij}^k = 1$ se J_i é processada diretamente após J_j na máquina M_k e $x_{ij}^k = 0$, caso contrário. O modelo do problema pode ser formulado, então, como:

$$\text{Minimizar} \quad \sum_{j=1}^{n} T_j \qquad (1.2)$$

sujeito a

$$\sum_{k=1}^{m} \sum_{i=0}^{n} x_{ij}^k = 1 \; ; \; j=1,..,n \qquad (1.3)$$

$$\sum_{j=1}^{n} x_{0j}^k \leq 1 \; ; \; k=1,..,m \qquad (1.4)$$

$$\sum_{\substack{i=0 \\ i \neq h}}^{n} x_{ih}^k - \sum_{\substack{j=0 \\ j \neq h}}^{n} x_{hj}^k = 0 \; ; \; k=1,..,m; \; h=1,..,n \qquad (1.5)$$

$$C_j \geq C_i - M + (S_{ij} + p_j + M) \cdot \sum_{k=1}^{m} x_{ij}^k \; ; \; i=0,..,n; \; j=1,..,n \qquad (1.6)$$

$$T_j \geq C_j - d_j; \; j=1,..,n \qquad (1.7)$$

$$C_j \geq 0; \; T_j \geq 0; \; j=1,..,n \qquad (1.8)$$

$$x_{ij}^k \in \{0,1\}; \; i=0,..n; \; j=1,..,n; \; k=1,..,m \qquad (1.9)$$

As restrições (1.3) garantem que cada tarefa será processada somente uma vez, pois estabelece que cada tarefa J_j poderá ter apenas um predecessor imediato e isso ocorrerá em somente uma máquina. As restrições (1.4) fazem com que cada máquina tenha somente uma seqüência de processamento, pois poderá ter somente uma tarefa inicial ou ficar vazia. Dado que cada tarefa tem somente um predecessor imediato, o conjunto de restrições (1.5) garante que terá também somente um sucessor imediato, com exceção da tarefa J_0 que estabelece o início e o final da seqüência de processamento de cada máquina. A relação entre os instantes de término das tarefas é estabelecida por (1.6). Além disso, essas restrições impedem que uma tarefa J_j seja ao mesmo tempo sucessora e predecessora de outra tarefa J_i, ou seja, as restrições (1.6) impedem que se formem ciclos de processamento infactíveis. Em (1.7) são definidos os atrasos das tarefas e, finalmente, (1.8) e (1.9) estabelecem os intervalos em que as variáveis do modelo são definidas. Esse problema é "NP-hard" pois a minimização do atraso total em uma máquina mesmo com $S_{ij} = 0, \ \forall i,j$ é "NP-hard" (DU e LEONG, 1990).

Boas revisões sobre a pesquisa relacionada a máquinas paralelas podem ser encontradas em (CHENG e SIN, 1990) e em (KAWAGUCHI e KYAN, 1988). Em ambos os trabalhos, é salientada a importância de se considerar aspectos como tempos de preparação dependentes da seqüência de processamento e datas de entrega em problemas de programação da produção. Entretanto, o número de trabalhos que consideram a programação em máquinas paralelas considerando esses aspectos é pequena (GUINET, 1991 e 1995). A pesquisa com máquinas paralelas sujeitas a tempos de preparação dependentes da seqüência está concentrada principalmente na minimização do "makespan" (GUINET, 1993; FRANÇA *et al.*, 1995), tempo médio de fluxo e tempo total gasto em preparações de máquina (RAJGOPAL e BIDANDA, 1991).

Dos trabalhos que consideram a minimização do atraso, podem ser destacados os seguintes. BERNARDO e LIN (1994) estudaram o problema de programação bi-critério resultante da minimização do atraso total e dos custos de preparação de máquina, propondo um procedimento heurístico interativo de solução do problema. Não são considerados tempos de preparação de máquina e os autores não estabelecem uma relação explícita entre os custos de preparação e os tempos de preparação correspondentes.

O trabalho apresentado por LEE e KIM (1993) consiste no desenvolvimento de uma rede neural para o cálculo de dois parâmetros necessários à aplicação da regra de despacho proposta por LEE e PINEDO (1992). Essa regra é apresentada para o caso em que se considera $m = 1$ em (LEE, BHASKARAM e PINEDO, 1997) e seu desempenho na resolução do problema PST é investigado na seção 1.4.5.

Em (RAGATZ ,1989) é descrito um algoritmo Branch & Bound para o problema PST com uma única máquina. Esse algoritmo é aplicado a problemas com até 16 tarefas, pois perde eficiência para problemas maiores. Em (RUBIN e RAGATZ ,1995), o Branch & Bound é

utilizado para avaliar uma Busca Genética sobre um total de 32 problemas com 15, 25, 35 e 45 tarefas. Nem o algoritmo Branch & Bound nem a Busca Genética foram implementados neste trabalho. Entretanto, foi possível obter, através do Prof. G. L. Ragatz, os dados dos problemas de teste utilizados para avaliar a Busca Genética e esses problemas foram resolvidos pelas versões implementadas de Algoritmos Genéticos. Os resultados desse experimento são apresentados no Capítulo 3.

Em GUINET (1991) são estudados alguns problemas de programação da produção em máquinas paralelas. É apresentado um modelo geral para o problema PST onde as máquinas podem não ser idênticas e os autores propõem uma heurística construtiva seguida de uma fase de busca local. Essa heurística é apresentada na seção 1.4.3 e seu desempenho é avaliado na seção 1.4.5.

Outros métodos heurísticos propostos para a resolução do problema PST são encontrados na literatura. TAN e NARASIMHAN (1997) propõem uma implementação da meta-heurística "Simulated Annealing" para o caso de uma única máquina e LEE e PINEDO (1997) apresentam uma extensão da heurística de LEE ,BHASKARAM e PINEDO (1997) para o caso de múltiplas máquinas. Infelizmente, por serem trabalhos bastante recentes, não houve tempo de implementá-los e incluir a avaliação de seu desempenho neste trabalho.

1.3 - EQUIVALÊNCIA COM O PROBLEMA DO CAIXEIRO VIAJANTE

Quando $m = 1$, pode-se formular o PST como um Problema de Caixeiro Viajante Assimétrico (PCVA). Cada tarefa corresponde a uma cidade a ser visitada e a cidade J_0 corresponde ao ponto de partida e de chegada do caixeiro viajante. A solução do problema é dada pela ordem em que o caixeiro visita todas as cidades com o menor custo de atraso. As distâncias entre as cidades J_i e J_j são dadas por:

$$D_{ij} = S_{ij} + p_j \quad \forall J_i \text{ e } J_j \in \mathbf{N} \qquad (1.10)$$

e representam o tempo de viagem entre essas cidades.

Note que as distâncias D_{ij} podem não obedecer as condições de simetria (ou seja, D_{ij} não é necessariamente igual a D_{ji}, $\forall i,j$) e não há garantia de que elas obedeçam a desigualdade do triângulo ($D_{ij} \leq D_{ik} + D_{kj}$, $\forall i,j,k$).

A extensão dessa transformação para o caso $m > 1$ leva ao Problema de m Caixeiros Viajantes Assimétrico (m-PCVA) onde cada caixeiro percorre um subconjunto das cidades a serem visitadas. Todos os caixeiros viajantes têm como ponto de partida e chegada a cidade J_0 e essa é a única cidade em comum entre todos os circuitos percorridos.

Nesse ponto pode-se usar a equivalência dada em (LAWLER *et al.*, 1985) entre o m-PCVA com n cidades e o PCVA com $(n + m - 1)$ cidades. Considerando essa representação, pode-se descrever o problema proposto como o de minimização do atraso em

uma permutação de (n + m) nós, onde os nós de 0 a (m - 1) representam as máquinas e os nós de m a (m + n - 1) representam as tarefas. Seja [t] o índice do nó de tarefa ou de máquina que ocupa a posição t de uma determinada solução para o PCVA resultante. Fixa-se $J_{[1]} = J_0$ para forçar a permutação a começar sempre por uma das máquinas. Para calcular o instante de término das tarefas nessa solução basta fazer:

$$C_{[t]} = \begin{cases} C_{[t-1]} + D_{[t-1][t]}, & \text{se } [t] \geq m \\ 0, & \text{se } [t] < m \end{cases} \qquad (1.11)$$

A partir desses instantes de término pode-se calcular o atraso de cada tarefa e o atraso total associado a essa solução.

Tendo sido estabelecida essa equivalência, uma das abordagens possíveis seria a adaptação de bons algoritmos desenvolvidos para o PCVA ao problema PST. Um algoritmo ótimo para o PCVA foi proposto por MILLER e PEKNY (1991) na versão serial e em (PEKNY, MILLER e McRAE, 1990) na versão paralelizada. Esse algoritmo é capaz de obter soluções ótimas para o problema em tempos computacionais bastante razoáveis mas, sua eficiência depende fortemente do limitante inferior utilizado. Visto que o objetivo é minimizar a distância total percorrida pelo caixeiro viajante (equivalente ao "makespan"), o limitante inferior é obtido pela relaxação da formulação do problema permitindo a formação de circuitos parciais, obtendo um problema de designação. Em MILLER e PEKNY (1991) os autores relatam que, para a maioria dos problemas testados, a razão entre o valor da solução ótima do problema de designação e o valor da solução ótima do PCVA fica entre 0,99 e 1. Isso indica que problemas de grande porte podem ser resolvidos em tempos computacionais bastante satisfatórios. Para os problemas em que essa razão fica abaixo de 0,9 são resolvidos problemas com apenas 20 cidades.

A qualidade do limitante inferior de designação para o problema PCVA é estudada por FRIEZE, KARP e REED (1995). Pelos resultados teóricos apresentados nesse trabalho, pode-se concluir que os problemas gerados por Miller e Pekny encontram-se na classe de problemas para os quais a qualidade desse limitante é alta, explicando o sucesso da aplicação do algoritmo ótimo.

A adaptação desse algoritmo ótimo para o PST envolveria a busca por um bom limitante inferior para o PCVA considerando como objetivo a minimização do atraso. Infelizmente, quando são considerados tempos de preparação dependentes da seqüência, a obtenção de bons limitantes inferiores torna-se extremamente difícil. Por esse motivo, para avaliar as versões implementadas das meta-heurísticas estudadas neste trabalho, optou-se por utilizar heurísticas desenvolvidas para o PST para gerar valores de limitantes superiores para os problemas de teste.

1.4 - PROBLEMAS DE TESTE E LIMITANTES SUPERIORES

1.4.1 - Problemas de Teste

Os parâmetros de cada tarefa são todos números inteiros gerados a partir de uma distribuição uniforme nos seguintes intervalos:

- Tempos de processamento: [1,100];
- Tempos de preparação: [0,20];
- Datas de Entrega: A partir dos tempos de processamento, calcula-se $P = \frac{1}{m}\sum_{i=1}^{n} p_i$. O intervalo de geração é, então: [0,αP] com α = 0,8; 1,0 ou 1,2. Dessa forma, podem ser gerados 3 tipos de problemas de acordo com a restritividade dos valores das datas de entrega: tipo 1 com α = 0,8; tipo 2 com α = 1,0 e tipo 3 com α = 1,2.

A seguir são listados os três conjuntos de problemas gerados para testar os procedimentos propostos neste trabalho.

Conjunto A: 900 problemas de grande porte, divididos em 30 tamanhos definidos pela combinação do número de máquinas e tarefas. Foram utilizados m = 1, 2, 3, 4, 5 e 6 e n = 30, 50, 70, 90 e 110. Para cada par de valores (m,n) foram gerados 10 problemas de cada tipo. Esse conjunto de problemas é utilizado para o teste das heurísticas propostas para o PST na literatura e para a avaliação do desempenho das meta-heurísticas estudadas neste trabalho.

Conjunto B: 900 problemas de pequeno porte, sendo 300 problemas com m = 2 e n = 7, 300 problemas com m = 3 e n = 7 e 300 problemas com m = 2 e n = 8. Para cada tamanho, foram gerados 100 problemas de cada tipo. Esse conjunto de problemas é resolvido através de um procedimento de Enumeração Completa para avaliação das características do espaço de soluções do PST. Os resultados desse estudo são mostrados na seção 1.5.

Conjunto C: 360 problemas de pequeno porte, divididos em 18 tamanhos definidos pela combinação do número de máquinas e tarefas. Foram utilizados m = 1, 2, 3, 4, 5 e 6 e n = 6, 8 e 10. Para cada par (m,n) foram gerados 10 problemas por tipo. Esses problemas foram resolvidos utilizando-se o pacote computacional CPLEX. Os resultados obtidos servem como referência para a avaliação inicial das meta-heurísticas implementadas.

Nas seções seguintes, são apresentadas três heurísticas para problema PST. Essas heurísticas são comparadas na seção 1.4.5 utilizando-se o **Conjunto A** de problemas de teste.

1.4.2 - Heurística ATCS (m = 1)

Essa heurística foi proposta por LEE, BHASKARAM e PINEDO (1997) e consiste de

três fases. Na primeira fase, os dados do problema a ser resolvido são utilizados para determinar os parâmetros k_1 e k_2 da seguinte função de prioridade:

$$I_j(t, i) = \frac{1}{p_j} \cdot \exp\left[\frac{-\max(0, d_j - p_j - t)}{k_1 \cdot \overline{p}}\right] \cdot \exp\left[\frac{-S_{ij}}{k_2 \cdot \overline{S}}\right] \qquad (1.12)$$

onde t = instante em que a máquina torna-se livre para processar uma nova tarefa;

i = índice da última tarefa processada na máquina;

$\overline{p}$ = tempo de processamento médio;

$\overline{S}$ = tempo médio de preparação de máquina.

A determinação dos valores de k_1 e k_2 é feita através de equações determinadas empiricamente pelos autores.

Na segunda fase da heurística, a cada vez que a máquina termina o processamento de uma tarefa J_i, o índice $I_j(t, i)$ é calculado para todas as tarefas J_j ainda não programadas. A tarefa com o maior valor de prioridade é seqüenciada após J_i na máquina e o procedimento continua até que todas as tarefas sejam programadas.

A terceira fase da heurística ATCS consiste de uma Busca Local realizada sobre uma vizinhança restrita da solução obtida no final da fase 2. Para o cálculo dos limitantes superiores dos problemas de teste considerados neste trabalho, uma combinação das duas melhores vizinhanças testadas por Lee, Bhaskaram e Pinedo foi utilizada.

Seja $J_{[i]}$ a tarefa que ocupa a posição i na solução atual. Como vizinhos dessa solução consideram-se os movimentos de troca de $J_{[i]}$ com $J_{[j]}$, $i = 1,..,n$ e $|j-i| \leq x$, onde x é a amplitude máxima permitida ao movimento. Também consideram-se os vizinhos construídos pela inserção de $J_{[i]}$ nas posições j tais que $|j-i| \leq x$. A amplitude máxima imposta aos movimentos de troca e inserção pode ser escolhida através de testes computacionais. O valor 10 foi escolhido de acordo com os resultados de LEE, BHASKARAM e PINEDO (1997).

A heurística ATCS pode ser colocada na forma de algoritmo como segue.

- **Algoritmo ATCS**

Parâmetros: $\overline{p}$ = tempo médio de processamento e
$\overline{S}$ = tempo médio de preparação de máquina;
k_1 e k_2 = parâmetros utilizados no cálculo da função de prioridade.

Variáveis: **L** = lista de tarefas não programadas;
S = solução temporária;
S* = melhor solução encontrada pela heurística;
t = instante de término da última tarefa programada na máquina;
i = última tarefa processada na máquina, terminando no instante t.

Passo 1: A partir dos dados do problema, calcule: $\overline{p}$ e $\overline{S}$;
Calcule os parâmetros k_1 e k_2;
Passo 2: Construa a lista **L**, acrescentando a ela todas as tarefas a serem processadas;
Faça t = 0 e **S*** = ∅;
Passo 3: Calcule $I_j(t,i)$ para todas as tarefas J_j em **L**;
Passo 4: Ordene as tarefas em **L** por valores não crescentes de $I_j(t,i)$;
Passo 5: Seja J_{j*} a primeira tarefa de **L**
Faça: **S*** = **S*** + $\{J_{j*}\}$; **L** = **L** - $\{J_{j*}\}$; t = t + S_{ij*} + p_{j*} e i = j*;
Passo 6: Se **L** = ∅, vá para o passo 7;
Caso contrário, volte ao passo 3;
Passo 7: Construa a vizinhança de **S*** composta por todas as soluções alternativas obtidas pela troca de pares de tarefas distantes até 10 posições entre si e pela inserção de tarefas em novas posições que estejam a uma distância menor que 10 da posição atual.
Seja **S** a melhor solução encontrada nessa vizinhança;
Passo 8: Se T(**S**) < T(**S***), faça **S*** = **S** e volte ao passo 7;
Caso contrário, pare.

1.4.3 - Heurística SAA (m > 1)

Para os problemas de teste com m > 1 pode-se usar uma heurística proposta por GUINET (1991), chamada "Sequential Assignment Algorithm"- SAA. Segundo o autor, essa heurística é utilizada em um pacote comercial desenvolvido para a indústria têxtil.

A heurística SAA também é composta por três fases distintas. Na primeira delas, constrói-se uma lista de tarefas ordenadas por valores não decrescentes do parâmetro "Latest Beginning Date", calculado como a diferença entre a data de entrega de cada tarefa e seu tempo de processamento.

Na segunda fase, cada uma das tarefas, tomadas de acordo com a ordenação realizada, é programada em uma das máquinas de forma que seu instante de término seja o menor possível. Finalmente, a terceira fase consiste de uma etapa de melhoria da solução obtida na fase anterior, pela troca de tarefas entre máquinas. A heurística SAA é mostrada na forma de algoritmo a seguir.

- **Algoritmo SAA**

Parâmetros: S(i,j) = equivalente a S_{ij} - tempo de preparação necessário entre o processamento consecutivo das tarefas J_i e J_j.
Variáveis: LB(i) = d_i - p_i → Maior valor do instante de início da tarefa J_i para que ela termine em sua data de entrega;
IL(k) = Instante em que a máquina k torna-se livre para o processamento de uma nova tarefa;
UT(k) = Índice da última tarefa processada na máquina k;
NT(k) = Número de tarefas processadas na máquina k;
MM = Índice da máquina escolhida para processar a próxima tarefa de acordo com o critério de menor

```
                instante de término;
            I(k,p) → Índice da tarefa processada na posição p da
                máquina k;
            L = Lista ordenada de tarefas.
Passo 1: Construa uma lista L de tarefas ordenada por valores não
         decrescentes de LB(i), ∀i;
         Faça i = 1;
Passo 2: Para cada máquina k = 1,..,m faça:
            Instante de liberação igual a zero: IL(k) = 0;
            Última tarefa processada igual a zero: UT(k) = 0;
            Número de tarefas igual a zero: NT(k) = 0;
Passo 3: Retire a i-ésima tarefa da lista L (seja j o índice dessa
            tarefa);
         (Procure pela máquina em que o processamento de Jj
            terminará mais cedo):
            Faça MIN = ∞;
            Para cada máquina k = 1,..,m:
                  Se IL(k) + S(UT(k),j) + pj < MIN
                        MIN = IL(k) + S(UT(k),j) + pj;
                        MM = k;
         (Designe Jj à máquina escolhida e atualize as variáveis da
            máquina):
            UT(MM) = j; (Coloque Jj como a última tarefa)
            IL(MM) = MIN; (Estabeleça o instante de liberação da
                  máquina como o instante de término de Jj)
            NT(MM) = NT(MM) + 1; (Incremente o contador de tarefas)
            I(MM,NT(MM)) = j; (Coloque Jj como na posição correta
                  de processamento)
Passo 4: i = i + 1;
         Se i ≤ n volte ao passo 3.
         Caso contrário, vá para o passo 5.
Passo 5: Enquanto houver melhoria na solução, faça:
         Para cada máquina k = 1,..,m
            Para cada posição i = 1,..,UT(k)
               Para cada máquina h = k+1,..,m
                  Para cada posição j = 1,..,UT(h)
                     Troque entre si as tarefas I(k,i) e I(h,j) se
                     essa troca causar uma diminuição no custo
                     total de atraso.
```

1.4.4 - Heurística EDD-Inserção ($m \geq 1$)

Como uma proposta alternativa para as heurísticas apresentadas nas seções 1.4.2 e 1.4.3, pode-se utilizar uma heurística baseada no procedimento NEH (NAWAZ *et al.*, 1983). Inicialmente, constrói-se uma lista de tarefas ordenadas pela regra EDD ("Earliest Due Date"). Para cada tarefa da lista, tomada de acordo com a ordenação realizada, calcula-se a variação do atraso total da solução parcial existente caso a tarefa seja inserida em cada uma das posições possíveis. A tarefa é inserida na posição em que cause o menor aumento no atraso da solução parcial.

Na Figura 1.1, apresenta-se um exemplo desse procedimento. A tarefa J_5 é a próxima da lista a ser programada na solução parcial composta pelas tarefas J_1, J_2, J_3 e J_4. Calcula-se a variação do atraso total da solução parcial para cada posição em que J_5 pode ser inserida nessa solução. Essas posições são indicadas pelas setas na Figura 1.1.

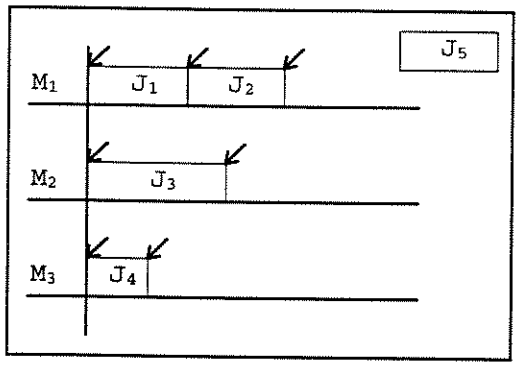

Figura 1.1 - Exemplo do procedimento de inserção da Heurística EDD-Inserção

A heurística EDD-Inserção é mostrada a seguir na forma de algoritmo.

- **Algoritmo EDD-Inserção**

Variáveis: T_k = soma dos atrasos das tarefas processadas na máquina M_k;
n_k = número de tarefas processadas na máquina M_k;
pos(j,k) = posição da tarefa J_j na seqüência de processamento da máquina M_k;
L = lista de tarefas não programadas;
$\Delta T(j,k)$ = aumento em T_k devido à inserção de uma tarefa J_j de **L** na posição pos(j,k) na máquina M_k.

Passo 1: Construa uma lista **L** de tarefas ordenada pela regra EDD;
Faça $n_k = T_k = 0$ para k = 1,..,m; i = 1;

Passo 2: Para k = 1,..,m faça:
Retire a i-ésima tarefa de **L** e insira essa tarefa na primeira posição da máquina M_k;
Faça $n_k = 1$ e calcule T_k; i = i + 1;

Passo 3: Tome a i-ésima tarefa J_j de **L**;
Para cada máquina M_k, k = 1,..,m, encontre a posição pos(j,k) onde a inserção de J_j causa o menor aumento $\Delta T(j,k)$, $1 \leq pos(j,k) \leq n_k + 1$;

Passo 4: Encontre o menor valor de $\Delta T(j,k)$ para J_j e k = 1,..,m;
Seja M_{k^*} a máquina para a qual $\Delta T(j,k^*) = \min\{\Delta T(j,k), \forall k\}$;

Passo 5: Insira a tarefa J_j na máquina M_{k^*}, faça $n_{k^*} = n_{k^*} + 1$ e recalcule T_{k^*};

Passo 6: Se $i \leq n$, faça i = i + 1 e volte ao Passo 3;

Passo 7: Calcule o atraso total da solução completa. Pare.

1.4.5 - Comparação entre as heurísticas apresentadas

As três heurísticas apresentadas nas seções 1.4.2, 1.4.3 e 1.4.4 foram implementadas utilizando-se a linguagem C. Todos os testes foram realizados com o **Conjunto A** de problemas em um PC Pentium 166 MHz.

A Tabela 1.1 mostra a comparação feita entre as heurísticas ATCS e EDD-Inserção para os problemas com m = 1. As colunas "Média de Custo" mostram os valores médios de custo das soluções obtidas por essas heurísticas. Essas colunas têm por finalidade mostrar a ordem de grandeza dos valores de custo obtidos para melhor avaliação

das porcentagens apresentadas. Para cada problema de teste, o desvio entre os custos obtidos pelas heurísticas é calculado como:

$$\text{desvio} = \frac{T(\text{EDD} - \text{Insercao}) - T(\text{ATCS})}{T(\text{ATCS})} \qquad (1.13)$$

onde T(EDD-Inserção) é o atraso total obtido pela heurística EDD-Inserção e T(ATCS) é o atraso total obtido pela heurística ATCS.

Para cada valor de n, é calculado um desvio médio por tipo e um desvio médio total. Na Tabela 1.1 também são apresentados os tempos médios gastos pelas duas heurísticas para cada tamanho de problema.

Pode-se observar que a heurística EDD-Inserção possui um desempenho bastante fraco para todos os problemas testados, principalmente para os problemas do tipo 3. Isso se deve ao fato de que, nesses problemas, as datas de entrega são menos restritivas, o que possibilita que a heurística ATCS encontre soluções de custo bastante baixo (LEE, BHASKARAM e PINEDO, 1997). Nota-se também que há uma deterioração muito rápida no desempenho de EDD-Inserção com o aumento do número de tarefas nos problemas resolvidos. A partir de n = 70, o desvio médio ultrapassa 500%, mostrando que essa heurística, além de não possuir um bom desempenho, não possui robustez. Quanto ao tempo computacional, novamente a heurística ATCS mostra-se superior à EDD-Inserção pois é capaz de resolver mesmo os problemas maiores em menos de 0,2 segundos, em média. Devido a esses fatores, escolheu-se a heurística ATCS como base para a avaliação das meta-heurísticas implementadas neste trabalho para os problemas com m = 1.

Tabela 1.1 - Comparação das heurísticas EDD-Inserção e ATCS para problemas com m=1

		Média de Custo		Desvio(%)		Tempo (s)	
n	tipo	ATCS	EDD-Inserção	por tipo	médio	ATCS	EDD-Inserção
30	1	5174,7	7002,4	36,36			
	2	2898,9	4747,6	82,81	285,06	0,02	0,10
	3	1325,8	3661,3	736,03			
50	1	12046,4	17897,8	50,12			
	2	7237,7	15518,8	151,32	438,91	0,03	0,67
	3	1335,0	9093,7	1115,31			
70	1	25394,2	38352,5	53,51			
	2	12129,2	26532,7	158,46	500,17	0,06	2,47
	3	3371,4	20258,2	1288,55			
90	1	43514,7	63331,5	47,12			
	2	17532,8	45491,4	189,27	2667,83	0,09	6,69
	3	2825,1	30847,5	7767,09			
110	1	64181,4	95926,9	51,73			
	2	27541,3	63399,6	155,76	7886,61	0,14	14,62
	3	2480,6	45228,9	23452,33			

Para os problemas com m > 1, realizou-se uma comparação entre as heurísticas SAA e EDD-Inserção. As tabelas 1.2 a 1.6 apresentam informações análogas às da Tabela

1.1 para cada um dos valores de m considerados. Nessas tabelas, o desvio de custo para cada problema de teste foi calculados como:

$$\text{desvio} = \frac{T(EDD - NEH) - T(SAA)}{T(SAA)} \qquad (1.14)$$

Tabela 1.2 - Comparação das heurísticas EDD-Inserção e SAA para problemas com m=2

		Média de Custo		Desvio(%)		Tempo(s)	
n	tipo	SAA	EDD-Inserção	por tipo	médio	SAA	EDD-Inserção
30	1	3356,0	3704,3	11,20			
	2	1835,7	2518,8	42,84	94,42	0,04	0,05
	3	1129,1	1868,9	229,23			
50	1	8082,7	10039,4	247,73			
	2	4625,3	6514,3	54,89	121,50	0,25	0,29
	3	1802,1	4452,7	284,90			
70	1	14031,0	18109,2	29,87			
	2	8002,0	13787,7	87,64	372,32	0,77	1,02
	3	2440,8	8272,9	999,45			
90	1	23266,1	29246,8	28,26			
	2	11240,0	20131,6	114,94	1084,47	1,62	2,74
	3	2529,5	12306,1	3110,20			
110	1	30967,3	41832,6	35,70			
	2	14823,8	28748,7	111,96	3976,25	3,12	5,95
	3	2532,7	16406,9	11781,10			

Tabela 1.3 - Comparação das heurísticas EDD-Inserção e SAA para problemas com m=3

		Média de Custo		Desvio(%)		Tempo(s)	
n	tipo	SAA	EDD-Inserção	por tipo	médio	SAA	EDD-Inserção
30	1	2456,8	2715,8	1,30			
	2	1574,6	1964,3	30,09	55,40	0,05	0,02
	3	551,2	1022,6	124,83			
50	1	5876,9	7078,8	20,84			
	2	3530,3	4799,0	38,30	140,80	0,28	0,18
	3	1190,9	2590,1	363,27			
70	1	10163,7	12353,5	21,83			
	2	5580,1	8423,0	59,64	196,07	1,00	0,65
	3	1440,0	4864,7	506,74			
90	1	14707,4	19121,6	30,10			
	2	7971,0	13025,5	80,92	937,16	2,06	1,70
	3	1972,1	7496,3	2700,46			
110	1	22043,1	27499,8	25,27			
	2	10051,3	18630,2	98,02	178,77	4,24	3,63
	3	3572,6	11489,0	413,04			

Tabela 1.4 - Comparação das heurísticas EDD-Inserção e SAA para problemas com m=4

		Média de Custo		Desvio(%)		Tempo(s)	
n	tipo	SAA	EDD-Inserção	por tipo	médio	SAA	EDD-Inserção
30	1	1736,9	1945,8	12,43			
	2	1181,6	1476,6	26,65	26,27	0,06	0,02
	3	941,0	1208,7	39,74			
50	1	4476,6	5087,5	13,78			
	2	2466,6	3359,8	41,55	132,38	0,31	0,14
	3	780,7	1876,8	341,82			
70	1	7421,9	9152,3	23,79			
	2	4243,1	5920,3	41,92	112,45	1,12	0,48
	3	1947,7	4250,3	271,65			
90	1	13609,9	16549,0	22,06			
	2	6778,8	10296,2	53,98	179,08	2,34	1,23
	3	1931,4	6090,4	461,21			
110	1	16961,3	21284,1	25,76			
	2	9782,0	15157,7	60,13	192,04	4,33	2,71
	3	1625,3	7365,2	490,24			

Tabela 1.5 - Comparação das heurísticas EDD-Inserção e SAA para problemas com m=5

		Média de Custo		Desvio(%)		Tempo(s)	
n	tipo	SAA	EDD-Inserção	por tipo	médio	SAA	EDD-Inserção
30	1	1854,9	1958,1	5,75			
	2	1146,2	1339,7	17,27	17,53	0,06	0,02
	3	891,2	1071,9	29,58			
50	1	3726,5	4313,1	15,93			
	2	2444,1	3001,6	25,15	45,99	0,38	0,11
	3	1370,4	2116,5	96,87			
70	1	6964,4	8069,6	16,37			
	2	3679,1	5170,6	44,16	136,06	1,04	0,38
	3	1199,2	2934,3	347,64			
90	1	10528,6	12473,7	18,51			
	2	5242,8	7895,8	58,46	182,96	2,45	0,98
	3	1741,1	4694,3	471,90			
110	1	14672,3	17935,3	22,64			
	2	6753,2	10636,1	60,62	182,56	5,24	2,12
	3	2361,1	6786,3	464,41			

Tabela 1.6 - Comparação das heurísticas EDD-Inserção e SAA para problemas com m=6

		Média de Custo		Desvio(%)		Tempo(s)	
n	tipo	SAA	EDD-Inserção	por tipo	médio	SAA	EDD-Inserção
30	1	1688,8	1837,0	9,15			
	2	1412,0	1550,2	9,59	16,51	0,05	0,02
	3	833,0	1050,7	30,80			
50	1	3770,6	4190,8	11,43			
	2	1939,9	2442,4	28,44	44,30	0,27	0,09
	3	936,9	1619,7	93,03			
70	1	5404,8	6368,3	18,11			
	2	3936,0	5125,2	32,88	76,78	0,82	0,32
	3	1127,1	2575,4	179,34			
90	1	9541,5	11245,1	18,27			
	2	4463,6	6543,9	50,56	215,42	1,82	0,81
	3	1737,4	3997,0	577,42			
110	1	11930,9	14628,3	23,09			
	2	6401,1	9508,4	53,59	320,73	3,57	1,78
	3	1495,4	4932,5	885,51			

Observa-se pelas tabelas 1.2 a 1.6 que a heurística SAA é superior, em média, à heurística EDD-Inserção. Novamente, nota-se a tendência de maiores desvios percentuais para problemas do tipo 3. Um estudo dos resultados dessas duas heurísticas para cada problema de teste mostrou que SAA apresenta resultados melhores que EDD-Inserção para praticamente todos os problemas, exceto para 8 deles (1,07% do total de 750 problemas). Por esse motivo, a heurística SAA será utilizada neste trabalho para fornecer limitantes superiores para os problemas de teste com m > 1. Com relação aos tempos computacionais, observa-se que a heurística EDD-Inserção é mais rápida que SAA, embora essa última seja capaz de resolver mesmo os maiores problemas em menos de 6 segundos, em média.

1.5 - ENUMERAÇÃO COMPLETA

Os testes cujos resultados são apresentados nesta seção foram realizados com o **Conjunto B** de problemas em um PC Pentium 166 MHz. Cada um dos 900 problemas desse conjunto teve suas soluções enumeradas duas vezes. Na primeira vez o objetivo foi encontrar a solução ótima e, na segunda vez foram calculados os desvios de cada solução **S** com relação ao ótimo **S*** de acordo com a expressão

$$\text{desvio} = \frac{T(S) - T(S^*)}{T(S^*)} \qquad (1.15)$$

O gráfico da Figura 1.2 mostra o desvio acumulado para todas as soluções dos problemas com m = 2 e n = 7. As linhas correspondentes aos três tipos de problema aparecem praticamente sobrepostas nesse gráfico indicando que há poucas diferenças entre as características associadas à restritividade das datas de entrega. A Figura 1.3 apresenta uma ampliação da área marcada na Figura 1.2. Nessa ampliação pode-se observar que menos de 1,2% das soluções encontram-se a menos de 10% do ótimo para os problemas testados. Dado o tamanho reduzido do problema, isso é uma indicação da dificuldade de resolução do PST.

Na Figura 1.4, apresenta-se um gráfico análogo ao da Figura 1.2 para os problemas com m = 3 e n = 7. A finalidade dos testes com esses problemas é investigar o efeito no aumento do número de máquinas na dificuldade de resolução do problema. Novamente, a área em torno da origem foi ampliada e mostrada na Figura 1.5. Nota-se um ligeiro aumento no número de soluções que se encontram a menos de 10% do ótimo. Entretanto, esse número não chega a 1,4% do número total de soluções. As linhas correspondentes aos tipos 1 e 2 praticamente se sobrepõem mesmo no nível de detalhe do gráfico da Figura 1.5. Isso indica que o aumento no número de máquinas não torna o problema PST mais fácil de ser resolvido.

Para investigar o efeito do aumento do número de tarefa na dificuldade de resolução do problema PST, foi construído o gráfico do desvio acumulado, análogo ao das figuras 1.2 e 1.4, para problemas com m = 2 e n = 8. Esse gráfico é apresentado na Figura 1.6 e a

ampliação da área em torno da origem é mostrada na Figura 1.7. Observa-se que o aumento no número de tarefas pode tornar o problema mais difícil. Como mostra a Figura 1.7, menos de 0,45% das soluções do problemas testados encontram-se a menos de 10% do ótimo.

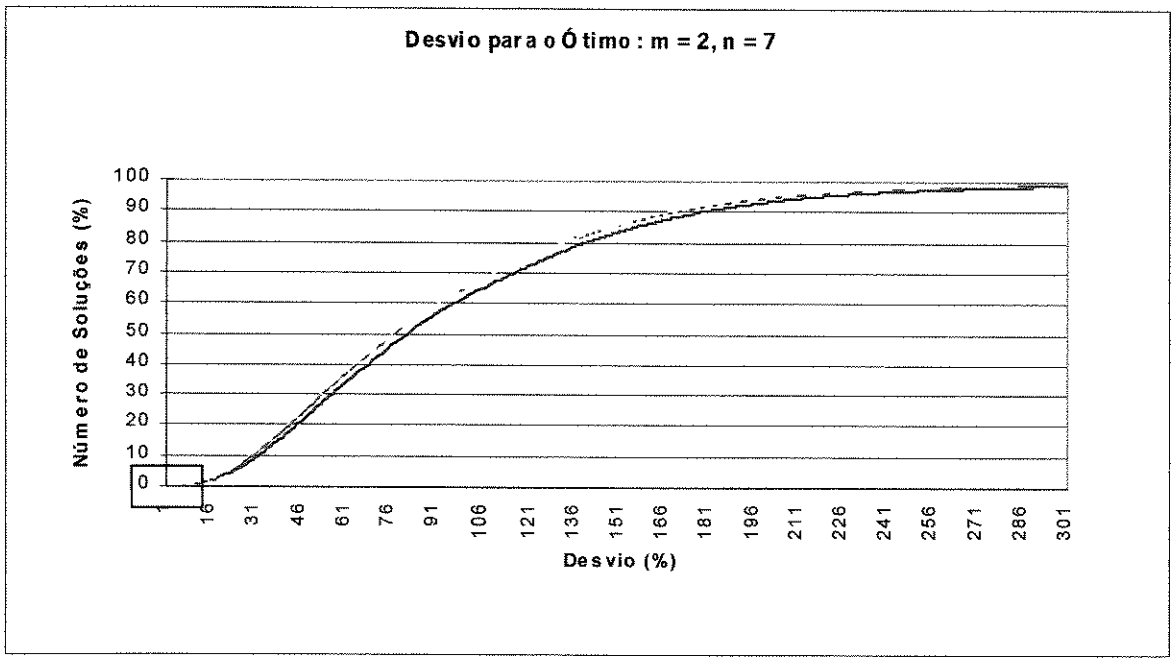

Figura 1.2 - Desvio acumulado das soluções do PST para m = 2 e n = 7

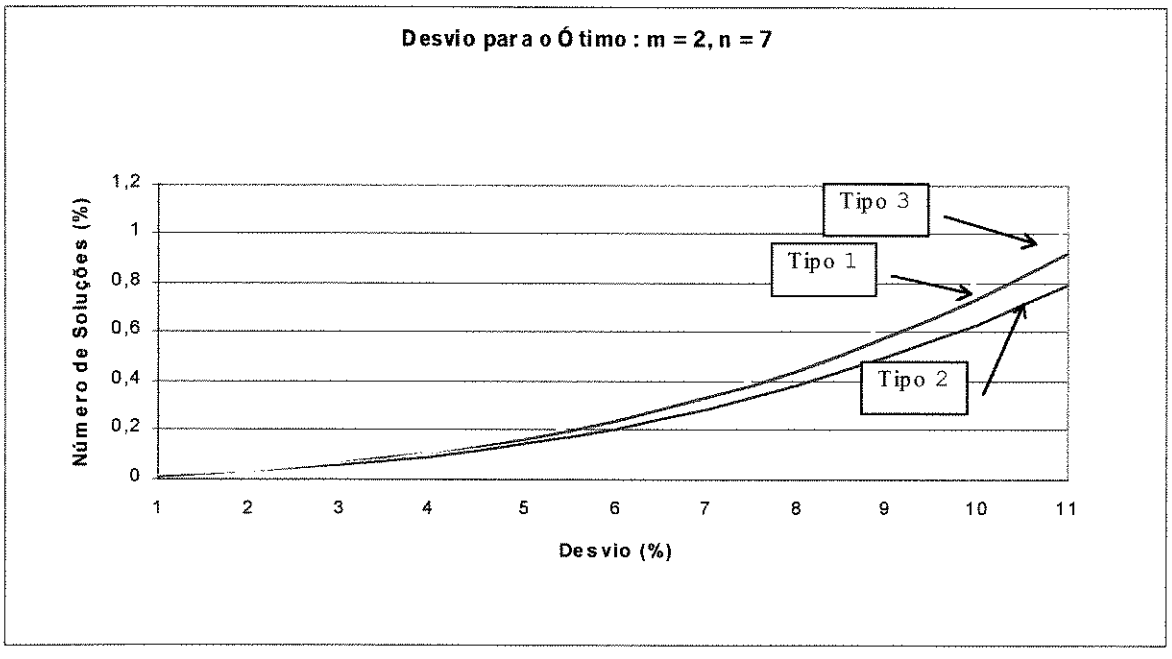

Figura 1.3 - Ampliação da área em torno da origem da Figura 1.2

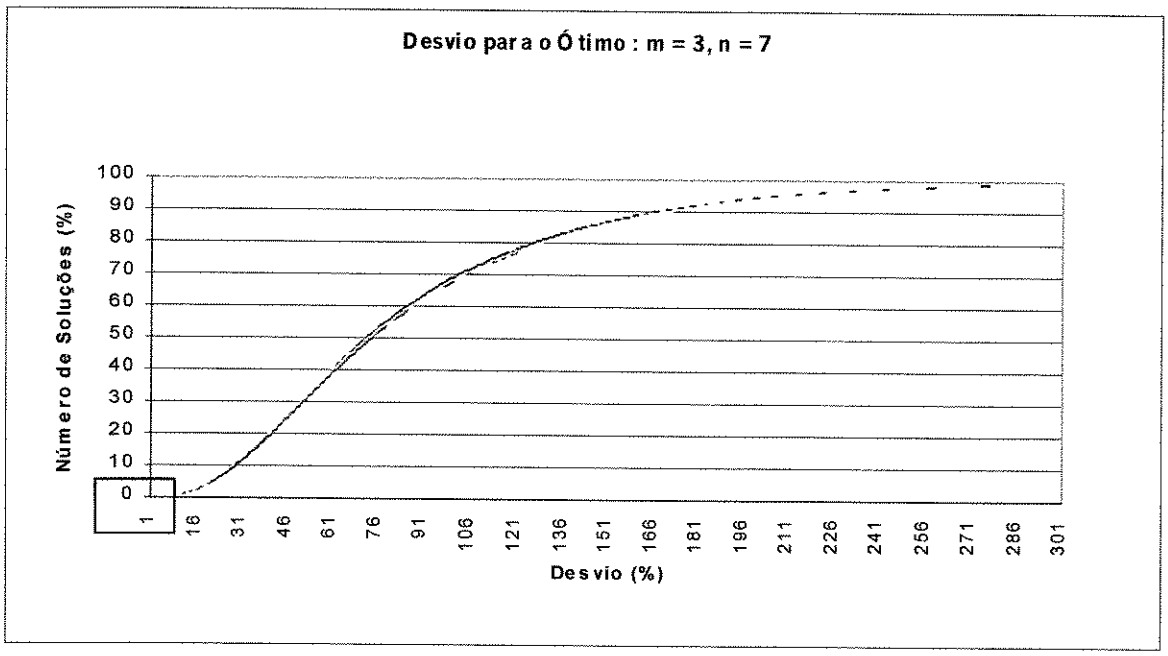

Figura 1.4 - Desvio acumulado das soluções do PST para m = 3 e n = 7

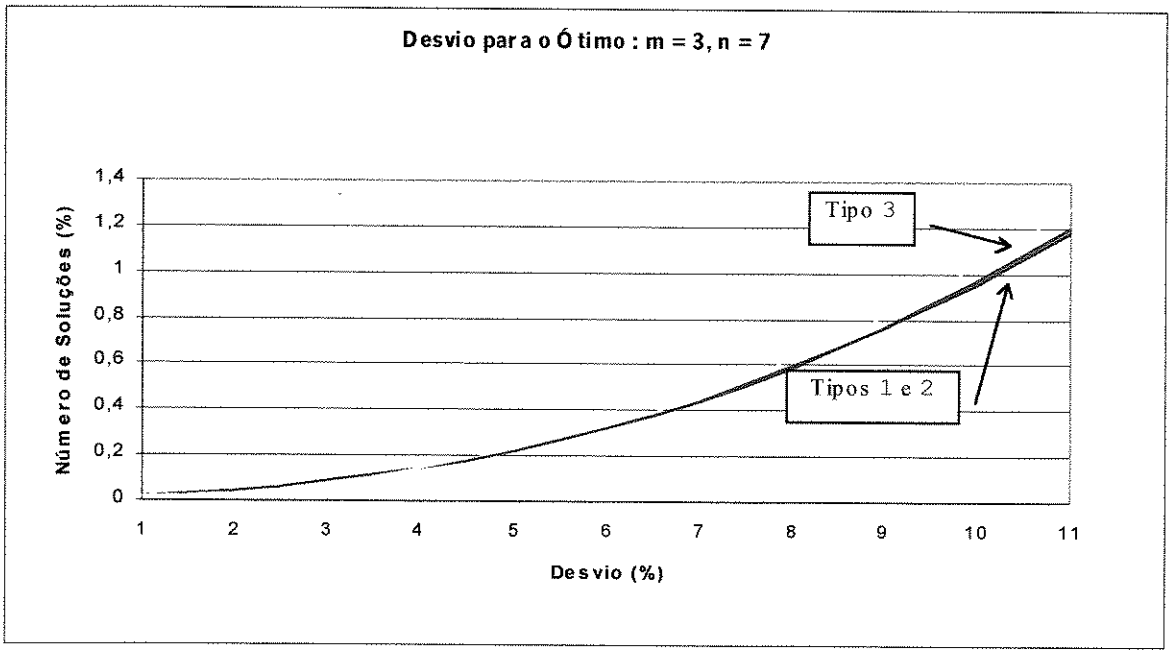

Figura 1.5 - Ampliação da área em torno da origem da Figura 1.4

A conclusão do experimento realizado através de Enumeração completa para problemas de pequeno porte mostra que o problema PST apresenta um alto grau de dificuldade de resolução por ter uma porcentagem muito baixa de soluções com custo próximo ao valor ótimo. Esses resultados também podem ser utilizados como um argumento a favor do investimento de esforço no desenvolvimento de bons algoritmos heurísticos para o problema dado que a configuração do espaço de busca observada, aliada à falta de bons limitantes inferiores, pode significar que algoritmos exatos terão grande dificuldade em

resolvê-lo.

Encerra-se aqui a fase inicial de estudo do problema PST, onde foram estabelecidas suas características e determinadas as heurísticas geradoras de limitantes superiores para o custo dos problemas de teste. A fase seguinte do trabalho consiste da implementação e testes da primeira meta-heurística a ser estudada, os Algoritmos Genéticos.

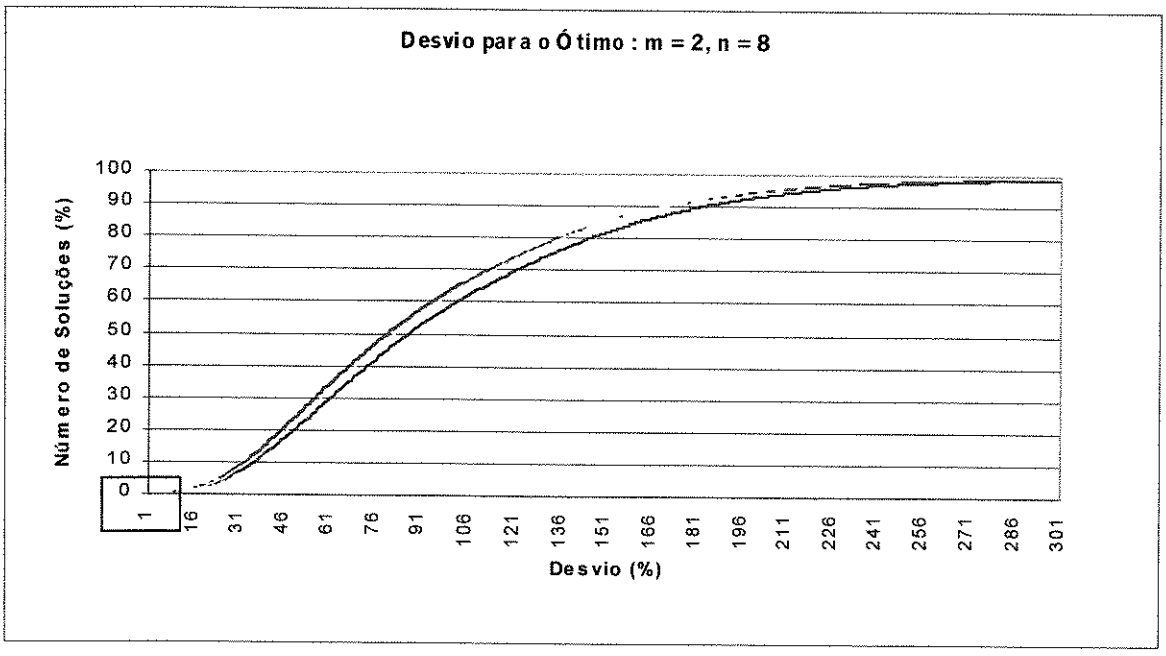

Figura 1.6 - Desvio acumulado das soluções do PST para m = 2 e n = 8

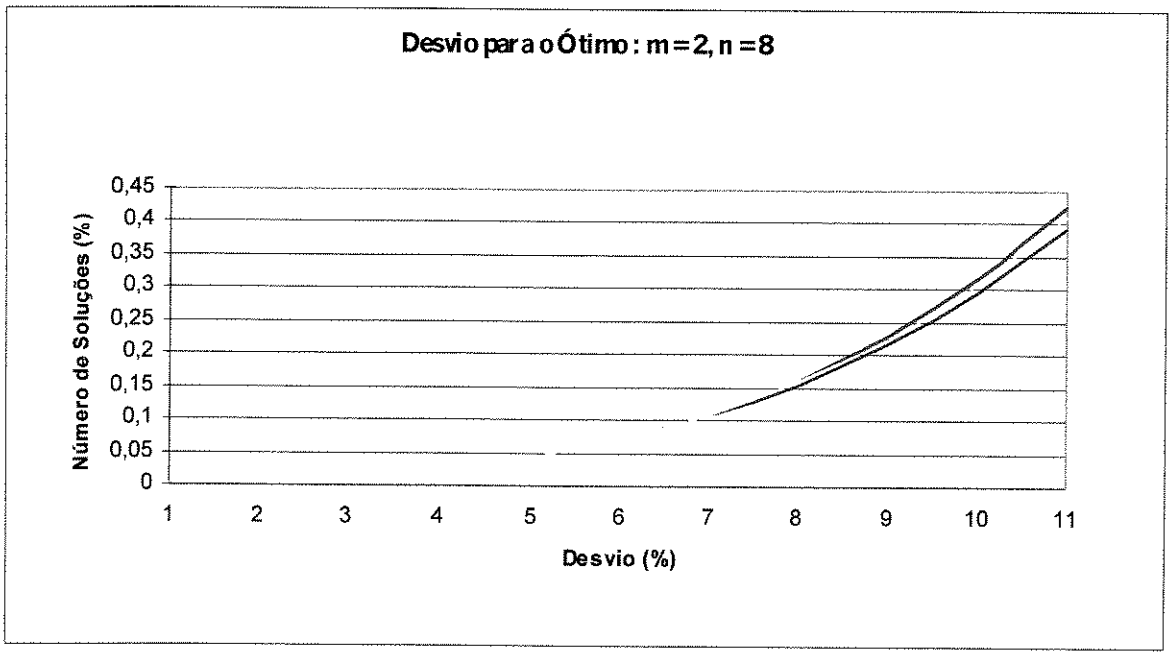

Figura 1.7 - Ampliação da área em torno da origem da Figura 1.6

CAPÍTULO 2 - ALGORITMOS GENÉTICOS IMPLEMENTAÇÃO

2.1 - INTRODUÇÃO

A meta-heurística chamada Algoritmos Genéticos (AGs) foi proposta inicialmente por John Holland em 1972 (HOLLAND, 1995). Em essência, essa meta-heurística é uma técnica de busca populacional, pois avalia um conjunto de soluções a cada iteração, e que se baseia na Teoria da Evolução das Espécies para selecionar e recombinar soluções fazendo com que elas evoluam ao longo das iterações. A analogia não se restringe aos mecanismos utilizados no método mas se estende a todo o vocabulário envolvido na descrição e implementação do mesmo. Por exemplo, usa-se equivalentemente os termos solução, cromossomo e indivíduo. Quaisquer outros termos que apresentem tal equivalência serão explicados ao longo do texto.

O estudo dos AGs pode ser feito de várias formas, mas talvez a maneira mais fácil de compreendê-los seja dividindo a análise em dois níveis. O primeiro deles pode ser chamado de nível externo (ou estrutural) e é composto pelas diretrizes do método. Esse nível pode ser visualizado no diagrama da Figura 2.1 e representa a meta-heurística em si, ou seja, a real analogia com a teoria da evolução. A cada iteração t (também chamada de geração) uma nova população é formada a partir da anterior através de três operadores genéticos básicos: reprodução, recombinação e mutação.

O nível interno (ou operacional) compreende todo o conjunto de estratégias que possam ser implementadas com a finalidade de colocar a estrutura evolutiva em funcionamento. Todo o conjunto de decisões tomadas com relação a essas estratégias como, por exemplo, o tipo de representação a ser usada, a forma de construção de uma nova população a partir da atual e os operadores genéticos a serem aplicados sobre as soluções, constituem uma implementação particular da meta-heurística.

A partir da estrutura externa do método, a implementação mais simples dos AGs é baseada nas seguintes decisões operacionais (GOLDBERG, 1989, MICHALEWICZ, 1996):

- Representação de soluções através de vetores de números binários cuja dimensão depende da aplicação;
- Reprodução feita através de uma roleta ponderada pelas probabilidades calculadas

através da avaliação de cada indivíduo com relação ao restante da população;

- Recombinação aplicada sobre toda a população com uma probabilidade p_{rec} e realizada através da troca de elementos entre dois vetores binários;
- Mutação aplicada sobre toda a população com uma probabilidade p_{mut} e realizada através da complementação de elementos dos vetores.

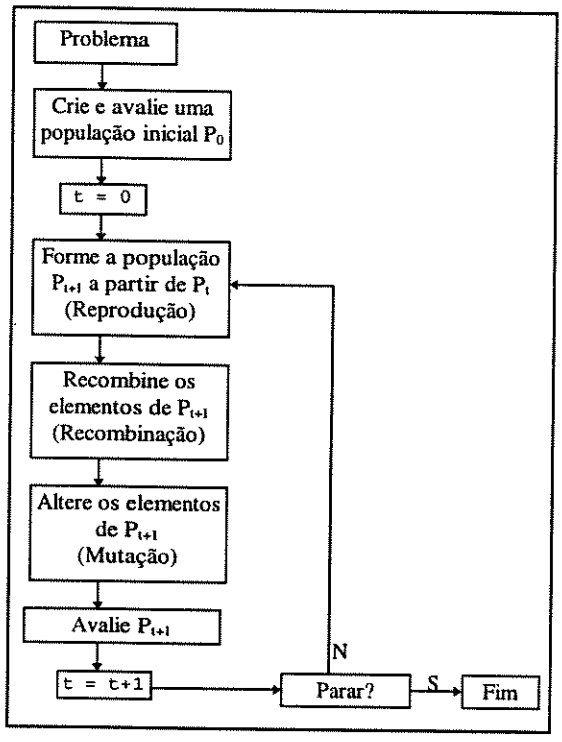

Figura 2 . 1 - Diagrama do nível externo dos AGs.

Apesar de sua simplicidade estrutural e operacional, as aplicações iniciais dessa implementação mostraram que ela pode ser extremamente robusta. Ela se mostrou capaz de realizar uma busca abrangente no espaço de soluções dos problemas propostos usando somente os valores da função objetivo calculados para cada solução, podendo ser aplicada a uma grande variedade de problemas sem necessidade de ser adaptada ou especialmente implementada para cada um deles. Essa independência que as primeiras implementações dos AGs mostraram com relação à complexidade dos problemas fizeram com que eles se tornassem uma ferramenta promissora na resolução de problemas de otimização considerados difíceis. Além de ser simples, o método apresenta grande flexibilidade com relação ao nível operacional, visto que nenhuma restrição é imposta diretamente às decisões possíveis nesse nível. Devido a essa flexibilidade, desde que foi proposto, o método tem sido implementado de várias formas diferentes. Todas as alterações sugeridas tiveram como objetivo a melhoria no desempenho geral dos AGs e incluem estratégias de reprodução, operadores genéticos alternativos, métodos de escalonamento para a função objetivo, entre outros.

Sem prejuízo direto das vantagens apresentadas pelo método, algumas desvantagens

também foram observadas. Em primeiro lugar, o desempenho dos AGs depende da escolha acertada de parâmetros como a taxa de aplicação dos operadores genéticos. Isso faz com que os testes iniciais para determinação de parâmetros correspondam a grande parte do esforço de pesquisa dedicado ao método mas, na verdade, esse fator não chega a ser uma desvantagem quando se compara o desenvolvimento de um algoritmo genético simples ao de uma implementação de Busca Tabu, por exemplo. Entretanto, se a comparação entre Busca Tabu e AGs é levada um pouco mais a fundo, outra desvantagem aparece. A Busca Tabu utiliza estruturas específicas de memória controladas por um conjunto de estratégias de busca para explorar características estruturais de soluções. Essas características são decompostas na forma de atributos de solução que podem ser manipulados através das estratégias de busca de forma a guiar o processo a regiões mais atrativas do espaço de soluções (intensificação) ou escapar de regiões pouco interessantes (diversificação). Os Algoritmos Genéticos não fazem uso de estruturas especiais de memória e não possuem estratégias definidas para a exploração do espaço de soluções, o que poderia ser feito através de características estruturais análogas aos atributos na Busca Tabu. Isso provoca uma grande insensibilidade dos Algoritmos Genéticos com relação ao contexto e à história da busca, podendo levar a uma rápida convergência para soluções localmente satisfatórias mas globalmente pobres. Outro aspecto da falta de tratamento de contexto nos AGs é sentida quando o problema a ser resolvido representa o modelo de um ambiente restrito pois não existe, no método, uma forma prevista de lidar com soluções infactíveis. Quando as restrições são leves, pode-se descartar essas soluções, fazendo com que a busca evite as regiões infactíveis do espaço de soluções. Entretanto, quando o problema envolve restrições muito fortes e encontrar uma solução factível torna-se tão difícil quanto encontrar a solução ótima, essa abordagem pode não funcionar satisfatoriamente.

Uma forma de contornar essas dificuldades é aproveitar a flexibilidade proporcionada pelo nível estrutural e planejar a implementação do nível operacional de uma forma menos presa à analogia genética. Representações alternativas para as soluções dos problemas de interesse, desenvolvimento de novos operadores genéticos, uso de heurísticas auxiliares ao processo de busca, entre outras adaptações, podem ser introduzidas sem descaracterizar o aspecto evolutivo dos AGs, preservando sua estrutura externa. Ao se permitir a introdução de modificações no esquema original para atingir níveis mais altos de qualidade, a meta-heurística Algoritmos Genéticos, composta por todas as implementações possíveis dentro da estrutura apresentada na Figura 2.1, torna-se cada vez mais robusta e pode ser aplicada cada vez a mais problemas diferentes.

Neste trabalho, o problema de otimização estudado possui uma natureza puramente combinatorial. Suas soluções são representadas de forma mais natural e completa através de vetores de números inteiros, correspondendo às possíveis seqüências de processamento das tarefas e suas designações às máquinas. Portanto, a implementação de um AG simples para o

problema PST é feita alterando-se a estrutura interna do algoritmo (representação e operadores genéticos) para melhor acomodar as características do problema. Foram desenvolvidas duas implementações de AG para resolver o problema, chamadas de GEN1 e GEN2. Essas implementações diferem basicamente no operador de reprodução, ou seja, na maneira como é feita a substituição da população P_t pela população P_{t+1}, na iteração t. Algumas outras diferenças menores entre as duas implementações existem mas elas serão melhor explicadas ao longo da descrição de GEN1 e GEN2, feita na seção seguinte.

2.2 - CARACTERÍSTICAS

Antes de descrever as características operacionais de cada uma das implementações de algoritmos genéticos, é apresentada a descrição geral de cada uma delas. As Figuras 2.2 e 2.3 apresentam os esquemas de operação de GEN1 e GEN2, respectivamente.

A principal diferença entre as duas implementações consiste na forma como se substitui a população da geração atual pela próxima população. Em GEN1 cada operador genético atua sobre toda a população e isso provoca a criação de populações intermediárias entre uma geração e outra. Esse comportamento é baseado na descrição dada no Capítulo 1 de GOLDBERG (1989) e no Capítulo 2 de MICHALEWICZ (1996). Já em GEN2, cada indivíduo da próxima população é gerado pela ação conjunta dos três operadores básicos, como sugerido no Capítulo 3 de GOLDBERG (1989).

Passo 1: Crie uma representação para as soluções do problema
Passo 2: Crie e avalie uma população inicial P_0
Passo 3: t = 0
Passo 4: Reproduza a população P_t, criando uma população intermediária P'_t
Passo 5: Realize a recombinação dos indivíduos em P'_t criando uma população P''_t
Passo 6: Realize a mutação dos indivíduos em P''_t criando a população P_{t+1}
Passo 7: Avalie a população P_{t+1} e faça t = t+1
Passo 8: Se nenhum critério de parada for satisfeito, volte ao Passo 4.
Caso contrário, pare e retorne o melhor indivíduo encontrado

Figura 2.2 - Esquema de operação de GEN1

Outras diferenças menores existem entre as duas implementações mas, por motivo de clareza de exposição, essas diferenças serão explicadas ao longo da descrição das características de GEN1 e GEN2, realizada nas seções 2.2.1 a 2.2.8. Em todo o texto é utilizada a equivalência entre os problemas PST e o PCVA, discutida na seção 1.3.

```
Passo 1: Crie uma representação para as soluções do problema
Passo 2: Crie e avalie uma população inicial P0
Passo 3: t = 0
Passo 4: Selecione dois indivíduos de Pt (ind1 e ind2)
Passo 5: Gere um número aleatório p ∈ [0,1]
Passo 6: Se p < pcross, gere um indivíduo de Pt+1 fazendo:
              filho = crossover (ind1,ind2)
              filho = mutação (filho)
Passo 7: Caso contrário, gere um indivíduo de Pt+1 fazendo:
              Se custo(ind1) < custo(ind2): filho = mutação(ind1)
              Caso contrário : filho = mutação(ind2)
Passo 8: Se Pt+1 ainda está incompleta, volte para o Passo 4.
         Caso contrário, vá para o Passo 9
Passo 9: Avalie a população Pt+1 e faça t = t+1
Passo 10: Se nenhum critério de parada for satisfeito, volte ao Passo 4.
         Caso contrário, pare e retorne o melhor indivíduo encontrado
```

Figura 2 . 3 - Esquema de operação de GEN2

2.2.1 - Representação

Logo nas primeiras tentativas de se resolver o PCV através de AGs, reconheceu-se que a representação de soluções através de vetores binários não é adequada. Para GOLDBERG (1989), a maneira mais natural de representar soluções em problemas de ordenamento (como o PCV e o PST) é através de uma permutação de números inteiros. Assim, uma solução de um PCV com 5 nós de cidade, em que os nós sejam visitados na ordem decrescente é representada pela permutação (0 5 4 3 2 1), onde o nó zero representa a cidade base de onde o caixeiro viajante parte e para onde deverá voltar após visitar os outros nós. Essa representação pode ser chamada de representação por caminho, pois a permutação de nós nada mais representa que o caminho percorrido pelo caixeiro viajante. Apesar de ser a mais simples possível para esse problema, essa representação carrega toda a informação necessária para a exploração das características das soluções através dos operadores genéticos: posição absoluta e relativa de cada nó no circuito e os arcos do grafo do PCV utilizados na solução. Como esses operadores devem ser definidos especificamente para a representação escolhida, quando mais informativa for a representação, mais liberdade há na escolha dos operadores. Outro aspecto importante dessa representação é que ela permite isolar e analisar as características das soluções exploradas tanto do ponto de vista de PCV como do ponto de vista de programação da produção. Além disso, o mapeamento do espaço de soluções realizado pela representação por caminho é completo: toda solução factível para o problema possui uma representação e toda permutação corresponde a uma solução factível.

Existem outras representações possíveis para o PCV (ou para o PCVA) mas nenhuma delas parece ser tão apropriada quanto a representação por caminho. Algumas falham no

sentido de fornecer informações relevantes que podem ser exploradas pelos operadores genéticos e outras realizam um mapeamento mais intrincado do espaço de soluções. Um exemplo é a representação por adjacência (MICHALEWICZ, 1996) onde cada solução do PCVA corresponde a uma lista de nós construída de tal forma que o nó `j` ocupa a posição `i` se e somente se o arco `(i,j)` é utilizado na solução. Por exemplo, para um problema com 5 tarefas e uma máquina, a solução `(0 3 5 2 1 4)` seria representada pela lista `(3 4 1 5 0 2)`. Essa representação traz de forma clara as informações sobre as posições relativas dos nós e os arcos utilizados na solução, embora a informação sobre a posição absoluta dos nós necessite de um procedimento adicional de interpretação para ser obtida. Outra limitação dessa representação reside no fato de que nem todas as listas de adjacência correspondem a uma solução factível para o problema. No exemplo acima, se a lista for modificada trocando-se a posição entre os nós 4 e 5 obtém-se o vetor `(3 5 1 4 0 2)` que corresponde a uma solução infactível composta por dois circuitos parciais: `(0 3 4)` e `(1 5 2)`.

Neste trabalho foi utilizada a representação por caminho tanto em GEN1 como em GEN2. Como exemplo do uso dessa representação, considere um problema com 5 tarefas e 3 máquinas. O vetor `(0 1 2 -1 3 4 -2 5)` representa um programa de produção em que a máquina M_1 processa as tarefas J_1 e J_2, nessa ordem, a máquina M_2 processa as tarefas J_3 e J_4, nessa ordem e a máquina M_3 processa a tarefa J_5. Além disso, para facilitar a comparação entre as várias soluções de uma população e para diminuir a complexidade do cálculo do valor de atraso, fixou-se o primeiro elemento de cada vetor de solução como sendo a máquina M_1, ou seja, a cidade de número 0. É preciso observar que a representação por caminho possui redundâncias no sentido de que uma mesma solução pode ser representada por várias permutações distintas. No exemplo acima, o vetor `(0 1 2 -2 5 -1 3 4)` representa a mesma solução dada por `(0 1 2 -1 3 4 -2 5)`. Além disso, para máquinas idênticas, a ordem em que os nós de máquina aparecem na solução é irrelevante e, portanto, o vetor `(0 1 2 -2 3 4 -1 5)` representa a mesma solução dada pelos vetores acima. Alguns autores consideram que a duplicação de uma solução pode prejudicar o desempenho dos Algoritmos Genéticos e proíbem sua ocorrência (RUBIN e RAGATZ ,1995, BEASLEY, 1995). Por outro lado, pode-se esperar que os operadores genéticos, quando aplicados várias vezes a soluções iguais com representações diferentes, podem apresentar resultados distintos, contribuindo para a diversificação das novas soluções obtidas.

2.2.2 - População Inicial

A população inicial P_0 pode ser inicializada de várias formas. Por exemplo, com soluções obtidas por heurísticas mais simples, tanto construtivas como de Busca Local. Entretanto, um dos objetivos da implementação de GEN1 e GEN2 é testar a capacidade dos AGs simples de encontrar boas soluções para o problema em estudo sem fornecer a eles nenhuma indicação do caminho a ser seguido pela busca. Além disso, deseja-se observar suas

características de convergência e a influência dos operadores genéticos sobre a melhoria de qualidade das soluções, ao longo das gerações. Portanto, tanto em GEN1 como em GEN2, a população inicial P_0 é gerada aleatoriamente.

2.2.3 - Avaliação das Soluções

Cada solução de P_t, $\forall t$, é avaliada de acordo com o atraso total. Para calcular esse atraso é preciso determinar, inicialmente, os instantes de término correspondentes aos nós de tarefa. Utiliza-se, para isso, a medida de distância definida pela equação (1.10). Seja [i] o índice do nó que ocupa a i-ésima posição em um dado vetor de solução **S**. Partindo-se do primeiro elemento do vetor e caminhando na ordem da permutação dada:

$$C_{[i]} = \begin{cases} C_{[i-1]} + D_{[i-1][i]}, & [i] > 0 \\ 0, & [i] \leq 0 \end{cases} \qquad (2.1)$$

Finalmente, pode-se calcular o atraso total da solução **S**:

$$T(\mathbf{S}) = \sum_{i=1}^{n} T_i = \sum_{i=1}^{n} \max\{0, C_i - d_i\}. \qquad (2.2)$$

2.2.4 - Reprodução

O primeiro passo para se fazer a reprodução em um AG simples é construir uma roleta de seleção baseada nos valores da avaliação dos indivíduos da população $P_t = \{S_1, S_2, \ldots, S_{ind}\}$.

Seja F a soma dos valores dos atrasos dos indivíduos de P_t:

$$F = \sum_{i=1}^{ind} T(S_i) \qquad (2.3)$$

Supondo que o problema em estudo fosse de maximização a probabilidade de seleção de cada indivíduo seria calculada como:

$$p(S_i) = \frac{T(S_i)}{F}, i = 1, \ldots, ind \qquad (2.4)$$

Entretanto, o PST é um problema de minimização e a expressão (2.4) não é adequada pois atribui maiores probabilidades aos piores indivíduos da população. Para refletir o objetivo de minimização, podem ser utilizadas várias formas de inverter essa atribuição de probabilidades. Neste trabalho, três cálculos distintos de $p(S_i)$ foram testados. O primeiro deles consistiu em se ordenar a população P_t por valores não crescentes de atraso e atribuir a cada solução um valor de probabilidade proporcional à sua classificação na população ordenada. Dessa forma, a solução de menor atraso recebe a maior classificação e a solução de maior atraso recebe classificação 1. Para o cálculo da probabilidade de seleção, pode-se utilizar uma expressão análoga a (2.4) onde $T(S_i)$ é substituído pela classificação de S_i em P_t e F é a soma das classificações de todas as soluções.

O segundo cálculo de $p(S_i)$ testado baseou-se na seguinte expressão, onde o fator

$1/(ind-1)$ foi utilizado para garantir que a soma das probabilidades sobre toda a população P_t fosse igual a 1.

$$p(S_i) = \left[1 - \frac{T(S_i)}{F}\right] \cdot \frac{1}{ind - 1} \qquad (2.5)$$

Entretanto, a expressão para o cálculo de $p(S_i)$ que apresentou melhores resultados foi a proposta por MURATA (1996), e que consiste em se considerar o quadrado da diferença entre o maior atraso encontrado na população P_t e o atraso de cada solução S_i como uma medida relativa da qualidade de S_i. Para tanto, redefine-se F como:

$$F = \sum_{i=1}^{ind} [T_{max} - T(S_i)]^2 \qquad (2.6)$$

onde $T_{max} = \max_{i=1,..,ind} \{T(S_i)\}$, e calculam-se as probabilidades de seleção como:

$$p(S_i) = \frac{[T_{max} - T(S_i)]^2}{F}. \qquad (2.7)$$

A partir das probabilidades de seleção, calculam-se os valores cumulativos:

$$q(S_i) = \sum_{j=1}^{i} p(S_j). \qquad (2.8)$$

Para escolher um indivíduo através da roleta de seleção criada, gera-se um número pseudo aleatório $r \in [0,1]$. Se $r \leq q(S_1)$, escolhe-se o indivíduo S_1. Caso contrário, escolhe-se o indivíduo S_i tal que $q(S_{i-1}) < r \leq q(S_i)$, com $i = 2,..,ind$.

Em GEN1 cria-se uma roleta a partir dos indivíduos de P_t. Essa roleta é utilizada ind vezes escolhendo os indivíduos de P_t que são copiados em P'_t. Em GEN2 a roleta também é criada a partir da população P_t mas é utilizada $2.ind$ vezes pois, para cada indivíduo de P_{t+1} a ser criado é preciso selecionar dois indivíduos de P_t. Além disso, utiliza-se o modelo elitista em GEN2: seja **S*** a melhor solução encontrada entre a primeira geração e a geração t. Quando se termina de criar P_{t+1}, verifica-se se **S*** está presente nessa população. Caso não esteja, **S*** é introduzida em P_{t+1} no lugar de seu pior indivíduo (solução com maior valor de atraso).

Os procedimentos de reprodução utilizados em GEN1 e GEN2 são apresentados a seguir na forma de algoritmo.

- **Procedimento Reprodução_Gen1(P_t)**

Parâmetros: ind = número de soluções nas populações P_t e P'_t
Variáveis: S_i = solução pertencente à população P_t
S'_i = solução pertencente à população P'_t
Passo 1: Calcule $T_{max} = \max_{i=1,..,ind} \{T(S_i)\}$;

$$F = \sum_{i=1}^{ind} [T_{max} - T(S_i)]^2;$$

$$p(S_i) = \frac{[T_{max} - T(S_i)]^2}{F}, \quad \forall S_i \in P_t;$$

$$q(S_i) = \sum_{j=1}^{i} p(S_j), \quad \forall S_i \in P_t;$$

Passo 2: i = 1;
Passo 3: Se i > ind, Pare.
Passo 4: Gere um número aleatório r ∈ [0,1];
Passo 5: Se r ≤ q(S_1), S'_i = S_1;
Caso contrário, encontre S_k ∈ P_t tal que q(S_{k-1}) < r ≤ q(S_k) e faça S'_i = S_k;
Passo 6: i = i + 1; Vá para o passo 6.

- **Procedimento Reprodução_Gen2(P_t,j)**

Parâmetros: ind = número de soluções na populações P_t
Variáveis: S_i = solução pertencente à população P_t
sol_1, sol_2 = soluções selecionadas pelo procedimento
Passo 1: Se j = 1, calcule

$$T_{max} = \max_{i=1,\ldots,ind} \{T(S_i)\};$$

$$F = \sum_{i=1}^{ind} [T_{max} - T(S_i)]^2;$$

$$p(S_i) = \frac{[T_{max} - T(S_i)]^2}{F}, \quad \forall S_i \in P_t;$$

$$q(S_i) = \sum_{j=1}^{i} p(S_j), \quad \forall S_i \in P_t;$$

Passo 2: Gere um número aleatório r_1 ∈ [0,1];
Passo 3: Se r_1 ≤ q(S_1), sol_1 = S_1;
Caso contrário, encontre S_k ∈ P_t tal que q(S_{k-1}) < r_1 ≤ q(S_k) e faça sol_1 = S_k;
Passo 4: Gere um número aleatório r_2 ∈ [0,1];
Passo 5: Se r_2 ≤ q(S_1), sol_2 = S_1;
Caso contrário, encontre S_k ∈ P_t tal que q(S_{k-1}) < r_2 ≤ q(S_k) e faça sol_2 = S_k;
Passo 6: Pare.

2.2.5. Recombinação em GEN1

Após a construção da população P'_t, é realizada a recombinação de seus indivíduos. O parâmetro que controla a atuação do operador de recombinação é a probabilidade p_{rec}. Essa probabilidade é utilizada em GEN1 da seguinte forma: para cada indivíduo em P'_t gera-se um número pseudo aleatório r ∈ [0,1]. Se r ≤ p_{rec} o indivíduo é selecionado para recombinação. Caso contrário, ele é transferido para P''_t sem ser alterado. Após determinar quais indivíduos serão recombinados, agrupam-se aleatoriamente esses indivíduos em pares e as soluções que compõem cada par são chamadas de Pai1 e Pai2. A partir de cada par formado são gerados dois descendentes que são colocados na população P''_t.

A geração de descendentes a partir de um par de indivíduos é feita através da recombinação dos arcos presentes nos pais. Essa operação é uma adaptação do "Edge Recombination Crossover" (MICHALEWICZ, 1996), desenvolvido originalmente para o PCV simétrico. A descrição a seguir corresponde à versão adaptada para o PCVA.

Toma-se o par de indivíduos e constrói-se uma lista dos arcos presentes nos pais. Um exemplo dessa construção é dado na Figura 2.4.

Pai1 =	*(0*	*2*	*3*	*-1*	*1*	*4*	*-2*	*5*	*6)*
Pai2 =	*(0*	*5*	*3*	*2*	*-1*	*-2*	*1*	*4*	*6)*

Lista de arcos dos pais:

nó de partida	*-2*	*-1*	*0*	*1*	*2*	*3*	*4*	*5*	*6*
arcos do *Pai1*	*5*	*1*	*2*	*4*	*3*	*-1*	*-2*	*6*	*0*
arcos do *Pai2*	*1*	*-2*	*5*	*4*	*-1*	*2*	*6*	*3*	*0*

Figura 2.4 - Exemplo de lista de arcos para a recombinação

Nessa lista está toda informação necessária para construção dos descendentes. Por exemplo, na primeira coluna da lista estão indicados os arcos que saem da máquina M_3 (nó -2), na segunda coluna os arcos que saem da máquina M_2 (nó -1) e assim por diante. No primeiro pai existe um arco do nó de máquina M_3 para o nó da tarefa J_5 e no segundo pai, um arco que vai do nó de máquina M_3 para o nó da tarefa J_1. Os descendentes são construídos partindo-se da cidade 0 (máquina M_1) e escolhendo aleatoriamente um dos arcos dos pais. Na lista da Figura 2.4, pode-se ir do nó 0 para o nó 2 ou 5. Estes dois nós tem a mesma probabilidade de serem escolhidos e escolhe-se aleatoriamente um deles. Suponha que seja o nó 5. Passa-se então para a coluna do nó 5 e escolhe-se aleatoriamente um dos arcos que saem dele. Suponha que se escolha o arco que vai para o nó 3. Escolhe-se então um dos arcos possíveis a partir do nó 3 e repete-se o processo até que todos os nós tenham sido visitados. Evita-se a formação de circuitos parciais proibindo qualquer nó de ser escolhido mais de uma vez.

No caso de se chegar a um impasse, ou seja, se a partir do nó atual não houver opção factível, escolhe-se aleatoriamente um nó que ainda não faça parte da solução utilizando um arco que não está presente em nenhuma das soluções-pai. Como exemplo desse procedimento pode-se utilizar a lista da Figura 2.4. Supondo que a solução parcial já tenha 8 nós: (0 2 -1 1 4 -2 5 3) percebe-se que na coluna da lista de arcos correspondente ao nó 3 não há arco que possa ser utilizado pois ambos levam a nós que já estão presentes na solução parcial. Portanto, é necessário acrescentar o arco (3,6) para completar a construção da nova solução e constata-se que esse arco não está presente nem em Pai1 nem em Pai2. Devido ao número reduzido de nós no exemplo, só havia uma opção para a última posição. Entretanto, quando há mais de uma opção, ou seja, quando mais de um nó ainda não faz parte da solução parcial, a escolha é feita aleatoriamente.

A partir da construção da lista de arcos, o processo descrito corresponde à criação da primeira solução filha de Pai1 e Pai2. Para criar o segundo filho, repete-se o processo de forma análoga.

Do ponto de vista de programação da produção, pode-se interpretar o operador de recombinação descrito da seguinte forma: cada solução pai representa uma designação de tarefas a máquinas e uma ordenação das tarefas processadas em cada máquina. Quando duas soluções são recombinadas, dois programas de produção alternativos são construídos. Cada

um deles representa uma mescla entre os dois programas pais. Na verdade, o processo de recombinação age como uma heurística construtiva que parte de um conjunto limitado de duas possibilidades para posicionar cada tarefa ou máquina no programa. Eventualmente, a recombinação insere informação nova pois, quando as duas possibilidades possíveis são infactíveis, a próxima tarefa (ou máquina) a ser inserida no programa de produção é escolhida aleatoriamente dentre aquelas que ainda não foram inseridas. O procedimento de recombinação utilizado em GEN1 é mostrado a seguir na forma de algoritmo.

```
• Procedimento Recombina_Gen1(Pai1, Pai2)
Variáveis:  L1 = lista de nós que ainda não estão presentes na
                 solução filha em construção;
            L2 = lista de nós que estão presentes na solução filha
                 em construção;
            suc(i,j) = sucessor do nó i no pai j;
            filho(1), filho(2) = soluções filhas resultantes do
                 processo de recombinação;
            filho(1,i) = nó que ocupa a posição i de filho(1);
            filho(2,i) = nó que ocupa a posição i de filho(2).
Passo 1: k = 1;
Passo 2: Construa L1 = lista composta de todos os nós de tarefa e
            máquina;
Passo 3: Faça L2 = ∅;
Passo 4: Determine suc(i,1) para todos os nós i de tarefa e de
            máquina em Pai1;
         Determine suc(i,2) para todos os nós i de tarefa e de
            máquina em Pai2;
Passo 5: Faça filho(k,0) = 0; l = 1;
Passo 6: Determine: opção1 = suc(filho(k,l-1),1) e
                    opção2 = suc(filho(k,l-1),2)
Passo 7: Se opção1 ∉ L2 e opção2 ∉ L2, escolha aleatoriamente um nó
            i entre opção1 e opção2 e vá para o passo 11;
Passo 8: Se opção1 ∈ L2 e opção2 ∉ L2, faça i = opção2 e vá para o
            passo 11;
Passo 9: Se opção1 ∉ L2 e opção2 ∈ L2, faça i = opção1 e vá para o
            passo 11;
Passo 10: Se opção1 ∈ L2 e opção2 ∈ L2, escolha aleatoriamente um nó
            i da lista L1 e vá para o passo 11;
Passo 11: Faça:   filho(k,l) = i;
                  L1 = L1 - {i};
                  L2 = L2 ∪ {i};
Passo 12: l = l + 1. Se L1 = ∅, vá para o passo 12;
          Caso contrário, volte ao passo 6;
Passo 13: k = k + 1.
          Se k > 2, Pare. Caso contrário, vá para o passo 2.
```

2.2.6 - Recombinação em GEN2

Para cada novo indivíduo a ser gerado para a população P_{t+1}, seleciona-se um par de indivíduos de P_t utilizando-se a roleta de seleção descrita na seção 2.2.4. O fato de cada par de soluções selecionadas gerar apenas um descendente, ao contrário de dois como ocorre em GEN1, deve-se a uma característica adicional acrescentada ao operador de recombinação.

Após a construção da lista de arcos dos pais, associa-se a esses arcos uma

probabilidade de escolha proporcional ao custo dos pais. Dessa forma, na construção da solução descendente, os arcos de Pai1 serão escolhidos aleatoriamente com uma probabilidade:

$$p = 1 - \frac{T(Pai1)}{T(Pai1) + T(Pai2)} \qquad (2.9)$$

e os arcos de Pai2 com uma probabilidade q = 1 - p. Esse procedimento é baseado no operador "Fusion Crossover" descrito em (BEASLEY, 1995).

A interpretação desse operador do ponto de vista de programação de produção segue a mesma linha da interpretação feita para a recombinação de GEN1. Na verdade, em GEN2, o processo de recombinação atua de forma um pouco mais informada, utilizando o valor de cada solução pai para determinar a probabilidade de cada uma das escolhas possíveis na construção de uma nova solução.

O procedimento de recombinação de GEN2 é mostrado na forma de algoritmo a seguir.

```
• Procedimento Recombina_Gen2(Pai1, Pai2)
Variáveis:  L1 = lista de nós que ainda não estão presentes na
               solução filha em construção;
            L2 = lista de nós que estão presentes na solução filha
               em construção;
            suc(i,j) = sucessor do nó i no pai j;
            filho(1) = solução filha resultante do processo de
                       recombinação;
            filho(1,i) = nó que ocupa a posição i de filho(1).
Passo 1: Construa L1 = lista composta de todos os nós de tarefa e
            máquina;
Passo 2: Faça L2 = ∅;
Passo 3: Determine suc(i,1) para todos os nós i de tarefa e de
            máquina em Pai1;
         Determine suc(i,2) para todos os nós i de tarefa e de
            máquina em Pai2;
Passo 4: Faça filho(1,0) = 0; l = 1;
Passo 5: Determine: opção1 = suc(filho(1,l-1),1) e
                    opção2 = suc(filho(1,l-1),2)
Passo 6: Se opção1 ∉ L2 e opção2 ∉ L2, escolha aleatoriamente um nó
            i entre opção1 e opção2 e vá para o passo 10;
Passo 7: Se opção1 ∈ L2 e opção2 ∉ L2, faça i = opção2 e vá para o
            passo 10;
Passo 8: Se opção1 ∉ L2 e opção2 ∈ L2, faça i = opção1 e vá para o
            passo 10;
Passo 9: Se opção1 ∈ L2 e opção2 ∈ L2, escolha aleatoriamente um nó
            i da lista L1 e vá para o passo 10;
Passo 10: Faça:   filho(1,l) = i;
                  L1 = L1 - {i};
                  L2 = L2 ∪ {i};
Passo 11: l = l + 1. Se L1 = ∅, Pare.
          Caso contrário, vá para o passo 5;
```

2.2.7 - Mutação

O parâmetro que controla a atuação desse operador é a probabilidade p_{mut}. Para determinar se um indivíduo vai ou não sofrer mutação gera-se um número pseudo aleatório r

$\in [0,1]$. Se $r \geq p_{mut}$ a mutação não ocorre. Caso contrário, deve-se escolher qual dos dois tipos de mutação será realizada nesse indivíduo pois ela pode ser feita de duas formas:

- Inserção: um nó é retirado de sua posição atual e inserido em nova posição;
- Troca: dois nós do circuito têm suas posições trocadas.

A probabilidade de escolha entre um e outro tipo de mutação depende da proporção estabelecida entre o número esperado de inserções e trocas a serem feitas na população. Novamente, deve-se gerar um número pseudo aleatório que determinará qual das duas alterações será efetuada no indivíduo selecionado para mutação.

No caso da mutação por inserção, escolhe-se aleatoriamente um nó e a nova posição para ele na solução. Na troca, escolhem-se aleatoriamente dois nós que terão suas posições trocadas. Após a mutação o indivíduo retorna à população e não poderá mais ser escolhido. Isso significa que cada indivíduo sofrerá, no máximo, um dos dois tipos de mutação a cada geração.

A interpretação de programação de produção para esse operador é imediata. Dada uma solução, duas tarefas (ou uma tarefa e uma máquina) terão suas posições trocadas, no caso de mutação por troca. No caso de mutação por inserção, uma tarefa (ou máquina) será deslocada de sua posição atual para uma nova posição na solução.

O procedimento de mutação é mostrado a seguir na forma de algoritmo.

- **Procedimento Mutação(S_i)**

```
Parâmetros: I:T = proporção esperada entre inserções e trocas;
            pmut = probabilidade de mutação.
Variáveis:  r = número aleatório;
Passo 1: Gere um número aleatório r ∈ [0,1];
         Se r ≥ pmut, Pare.
Passo 2: Gere outro número aleatório r ∈ [0,1];
         Se r < I/(I+T), vá para o passo 3 (Mutação por Inserção);
            Caso contrário, vá para o passo 4 (Mutação por Trocas);
Passo 3: Gere aleatoriamente j ∈ [-(m-1),n]e k ∈ [1,(m+n)];
         Construa uma nova solução S'i a partir de Si na qual o nó j
            passa a ocupar a posição k da permutação;
         Pare.
Passo 4: Gere aleatoriamente j e k ∈ [-(m-1),n];
         Construa uma nova solução S'i a partir de Si na qual os nós j
            e k passam a ter suas posições trocadas na permutação;
         Pare.
```

2.2.8 - Critérios de Parada

Três critérios de parada foram implementados em GEN1 e GEN2. O primeiro deles é o número máximo de iterações t_{max}. Os demais critérios dependem do andamento da busca e consistem em parar se for encontrada uma solução S_i com $T(S_i) = 0$ ou se for verificado que não há melhoria na solução incumbente **S*** por mais de $t_{max}/10$ iterações consecutivas.

2.2.9 - Algoritmos

A partir dos esquemas apresentados nas figuras 2.2 e 2.3 e dos procedimentos descritos nesta seção, pode-se colocar as duas implementações de algoritmos genéticos, GEN1 e GEN2 na forma de algoritmo.

- **Algoritmo GEN1**

Parâmetros: ind = número de soluções na população;
t_{max} = número máximo de iterações permitidas;
p_{rec} = probabilidade de recombinação;
p_{mut} = probabilidade de mutação;
I:T = proporção entre o número esperado de trocas e inserções no operador de mutação;

Variáveis: S_i = solução pertencente a P_t;
S'_i = solução pertencente a P'_t;
S''_i = solução pertencente a P_t'' ;
R = conjunto de soluções de P'_t selecionadas para recombinação;
S* → melhor solução encontrada pelo algoritmo.

Passo 1: R = ∅; t = 0;
Passo 2: Crie uma população inicial P_t composta por ind soluções geradas aleatoriamente;
Passo 3: Seja $\overline{S} = \arg\min_{i=1,\ldots,ind}\{T(S_i)\}$; Faça **S*** = $\overline{S}$;
Passo 4: Se algum dos critérios de parada foi satisfeito, Pare. **S*** é a melhor solução encontrada por GEN1.
Passo 5: P'_t = Reprodução_Gen1(P_t);
Passo 6: i = 1; k = 1;
Passo 7: Gere um número aleatório r ∈ [0,1];
Se r < p_{rec}, R = R ∪ {S'_i}, vá para o passo 8;
Caso contrário, $S''_k = S'_i$, k = k + 1;
Passo 8: Se i < ind, faça i = i + 1 e vá para o passo 7.
Caso contrário, vá para o passo 9;
Passo 9: Se |R| é ímpar, retire o último elemento S'_u adicionado a R, faça $S''_k = S'_u$ e k = k + 1;
Passo 10: Se R = ∅, vá para o passo 13;
Passo 11: Escolha aleatoriamente duas soluções Pai1 e Pai2 do conjunto R, faça R = R - {Pai1,Pai2};
Passo 12: Faça (S''_k, S''_{k+1}) = Recombina_Gen1(Pai1,Pai2);
k = k + 2, volte ao passo 10;
Passo 13: Calcule $T(S''_i)$, i = 1,..,ind;
Seja $\overline{S} = \arg\min_{i=1,\ldots,ind}\{T(S''_i)\}$; Se $T(\overline{S}) < T(\mathbf{S^*})$, faça **S*** = $\overline{S}$;
Passo 14: i = 1;
Passo 15: S_i = mutação(S''_i);
Passo 16: Se i < ind, faça i = i + 1 e volte ao passo 15;
Passo 17: Calcule $T(S_i)$, i = 1,..,ind;
Seja $\overline{S} = \arg\min_{i=1,\ldots,ind}\{T(S_i)\}$; Se $T(\overline{S}) < T(\mathbf{S^*})$, faça **S*** = $\overline{S}$;
Passo 18: Faça t = t + 1 e vá para o passo 4

- **Algoritmo GEN2**

Parâmetros: ind = número de soluções na população;
t_{max} = número máximo de iterações permitidas;
p_{rec} = probabilidade de recombinação;
p_{mut} = probabilidade de mutação;
I:T = proporção entre o número esperado de trocas e inserções no operador de mutação;

Variáveis: S_i = solução pertencente a P_t;
S'_i = solução pertencente a P_{t+1};
S* = melhor solução encontrada pelo algoritmo.

Passo 1: t = 0; i = 1;
Passo 2: Crie uma população inicial P_t composta por ind soluções geradas aleatoriamente;
Passo 3: Seja $\overline{S} = \arg\min_{i=1,\ldots,ind}\{T(S_i)\}$; Faça **S*** = $\overline{S}$;
Passo 4: Se algum dos critérios de parada foi satisfeito, Pare. **S*** é a melhor solução encontrada por GEN2.
Passo 5: P'_t = Reprodução_Gen1(P_t);
Passo 6: (Pai1,Pai2) = Reprodução_Gen2(P_t);
Passo 7: S'_i = Recombina_Gen2(Pai1,Pai2);
Se $T(S'_i)<T(\mathbf{S^*})$, faça **S*** = S'_i;
Passo 8: S'_i = Mutação(S'_i); Se $T(S'_i)<T(\mathbf{S^*})$, faça **S*** = S'_i;
Passo 9: Se i < ind, faça i = i + 1 e vá para o passo 6.
Caso contrário, vá para o passo 10;
Passo 10: Se **S*** $\in P_{t+1}$, vá para o passo 11;
Caso contrário, substitua a pior solução de P_{t+1} por **S***;
Passo 11: Faça t = t + 1 e vá para o passo 4

2.2.10 - Parâmetros

Os parâmetros que influenciam o desempenho dos AGs implementados são:

- Tamanho da população: ind;
- Número de gerações a serem executadas: ger;
- Probabilidade de mutação: p_{mut};
- Probabilidade de recombinação : p_{cross};
- Proporção entre as duas formas de mutação (relação entre o número esperado de inserções e trocas): I : T.

Cada um desses parâmetros possui uma influência e um significado diferentes para a busca e a fase de determinação de parâmetros pode ser usada como uma oportunidade de se fazer uma análise do comportamento do algoritmo conforme são variados os parâmetros.

No capítulo seguinte é apresentada a fase de determinação de parâmetros e os testes computacionais realizados com GEN1 e GEN2.

CAPÍTULO 3 - ALGORITMOS GENÉTICOS TESTES COMPUTACIONAIS

No capítulo anterior foram apresentadas as características de duas implementações de Algoritmos Genéticos denominadas GEN1 e GEN2. Neste capítulo são apresentados os resultados dos testes computacionais realizados com essas duas implementações. Inicia-se realizando a determinação dos parâmetros necessários para a execução de GEN1 e GEN2. Logo em seguida, realizam-se testes de desempenho para determinar o ganho dessas implementações com relação às soluções heurísticas obtidas no capítulo 1 e para verificar as características do algoritmo em termos do tempo utilizado. Também são feitos alguns testes de comportamento onde são analisadas as características de convergência dos algoritmos implementados.

As duas implementações foram codificadas utilizando-se a linguagem C e executadas em um microcomputador PC Pentium 166 MHz.

3.1 - DETERMINAÇÃO DE PARÂMETROS

A primeira fase da determinação de parâmetros de GEN1 e GEN2 foi realizada fixando-se ind = 100 e ger = 1000 e variando-se os valores das probabilidades p_{rec} e p_{mut} e a proporção I:T. Após o ajuste inicial dos valores desses parâmetros, testes adicionais foram feitos para determinar valores mais apropriados para ind e ger.

Na determinação dos melhores valores de p_{rec}, p_{mut} e de I:T foram utilizados 30 problemas de teste com 4 máquinas e 70 tarefas. Esse conjunto foi considerado representativo pois possui um número intermediário de máquinas e tarefas. Dessa forma, espera-se que os possíveis desvios no desempenho, decorrentes da variação no tamanho dos demais problemas, sejam pequenos.

A comparação direta entre as várias combinações de valores possíveis para os parâmetros torna-se difícil quando os resultados obtidos por uma delas não dominam completamente os resultados obtidos pelas outras para todos os problemas testados e, como a distribuição dos resultados dos testes é geralmente desconhecida, a comparação de resultados

médios pode levar a conclusões errôneas. Segundo XU, CHIU e GLOVER (1996) pode-se utilizar os testes estatísticos de Friedman e Wilkoxon para analisar os resultados dos testes computacionais com um certo nível de confiança.

Suponha que deseja-se determinar os K parâmetros $(P_1,P_2,..,P_K)$ de um algoritmo genérico G e que cada um desses parâmetros possa assumir I_k valores. Se, para um dado parâmetro P_k, a diferença observada nos resultados dos testes (e gerada pelos valores distintos atribuídos ao parâmetro) puder ser considerada uma variável aleatória independente com uma distribuição contínua, embora desconhecida, então o teste de Friedman pode ser utilizado para determinar a significância dos efeitos dos valores testados no desempenho do algoritmo. A seguir é descrito o mecanismo do teste.

- **Teste de Friedman**

A hipótese nula desse teste é que não há efeito devido à variação dos valores do parâmetro P_k, ou seja, que a diferença nos resultados dos testes é causada por outros fatores aleatórios. Suponha que x_{ij} seja o resultado obtido pelo i-ésimo valor do parâmetro P_k para o problema de teste j $(i = 1,..,I_K$ e $j = 1,..,J)$. Para cada problema j deve-se classificar os resultados x_{ij} $(i = 1,..,I_k)$ de acordo com sua qualidade. Seja R_{ij} a classificação recebida pelo resultado x_{ij}. Se esse resultado for o melhor de todos, $R_{ij} = 1$ e se for o pior, $R_{ij} = I_k$. No caso de empates, os resultados correspondentes devem receber classificação igual à média das classificações que receberiam se houvesse um desempate arbitrário. Para ilustrar esse procedimento, suponha que os quatro resultados (7 5 5 8) devam ser classificados e que considera-se como melhor resultado o menor deles. As classificações recebidas por esse conjunto de resultados seriam (3 1,5 1,5 4).

A partir dos valores R_{ij}, calcula-se a seguinte estatística:

$$F_r = -3.J.(I_k+1) + \frac{1}{J.I_k.(I_k+1)}.12.\sum_{i=1}^{I_k} R_i^2 \qquad (3.1)$$

onde $R_i = \sum_{j=1}^{J} R_{ij}$. Na prática, para valores moderados de J, a estatística F_r tem aproximadamente uma distribuição chi-quadrada com $(I_k - 1)$ graus de liberdade quando a hipótese nula é verdadeira. Portanto, essa hipótese é rejeitada se F_r for maior que o valor crítico χ^2_{α,I_k-1} com um nível de confiança $(1 - \alpha)$.

Se a hipótese nula é rejeitada, isso significa que a variação do parâmetro P_k possui influência no desempenho do algoritmo. Quando isso ocorre, deve-se fazer uma comparação mais detalhada entre todos os pares de valores obtidos para verificar se há alguma relação de dominância entre eles. O valor i_1 é considerado melhor que i_2 a um nível de confiança $(1 - \alpha)$ se:

$$R_2 \geq R_1 + z(\alpha).\sqrt{J.I_k.(I_k+1)/6} \qquad (3.2)$$

onde $z(\alpha)$ é o $100.(1 - \alpha)$ -ésimo percentil da distribuição normal padrão com média

nula e variância igual a 1. Formalmente, $z(\alpha)$ é tal que se x é uma variável aleatória com uma distribuição $N(0,1)$, $Probabilidade(x \geq z(\alpha)) = \alpha$.

Seja R_{imin} o menor valor de R_i encontrado para o conjunto de I_k valores do parâmetro P_k. Para identificar os valores dominantes de P_k comparam-se os valores calculados R_i, $i = 1,..,I_k$ com R_{imin}. Se $R_i \geq R_{imin} + z(\alpha) . \sqrt{J. I_k. (I_k + 1) / 6}$, diz-se que i_{min} domina i. Caso contrário, aceita-se o valor i como um dos melhores valores possíveis para o parâmetro. Os valores dominados de P_k são desconsiderados nos testes subsequentes. Maiores detalhes sobre o Teste de Friedman podem ser encontrados em HETTMANSPERGER (1984) e LEHMANN e D'ABRERA (1975).

Quando duas versões de um algoritmo genérico G diferem apenas pelo valor de um único parâmetro, o teste de Friedman é suficiente para reduzir o número de valores relevantes para esse parâmetro. Entretanto, quando há mais de um parâmetro com valor diferente em duas versões de G, é preciso compará-las com outro teste estatístico. XU, CHIU e GLOVER(1996) sugerem o uso do teste de Wilkoxon pois ele pode comparar satisfatoriamente não só duas versões de G entre si mas qualquer par de algoritmos que tenham sido aplicados sobre o mesmo conjunto de problemas de teste. A seguir descreve-se o teste estatístico de Wilkoxon.

- **Teste de Wilkoxon**

Com esse teste, comparam-se duas versões independentes de G que possuam valores diferentes para mais de um parâmetro. Como as duas versões são testadas sobre o mesmo conjunto de J problemas, assume-se que os resultados obtidos de ambas possuem uma distribuição contínua que difere apenas com respeito a suas médias. Sejam X_A e X_B as séries de resultados obtidos pelas versões A e B do algoritmo G, respectivamente. Seja $D = X_A - X_B$. A hipótese nula desse teste é $H_0 : \mu_D = 0$, onde μ_D é a média de D.

O teste usa a estatística W que pode ser calculada como segue. Primeiramente, classificam-se os componentes de D pelo seu valor absoluto. O componente com menor valor absoluto recebe classificação 1 e o que tiver maior valor absoluto, classificação J e os empates são resolvidos da mesma forma que no teste de Friedman. Calcula-se W como a soma das classificações associadas a componentes positivos de D ($W = 0$ se todos os componentes forem negativos ou nulos). Para valores moderados de J, a estatística W possui aproximadamente uma distribuição normal com média e variância calculadas como:

$$E(W) = \frac{J.(J+1) - n_0.(n_0+1)}{4} \qquad (3.3)$$

$$\sigma^2(W) = \frac{J.(J+1).(2J+1) - n_0.(n_0+1).(2n_0+1)}{24} \qquad (3.4)$$

onde n_0 representa o número de componentes nulos ou negativos da série D.

Pode-se rejeitar, então, H_0 com um nível de confiança $(1 - \alpha)$ quando $W \geq E(W) + z(\alpha) \cdot \sqrt{\sigma^2(W)}$ ou quando $W \leq E(W) - z(\alpha) \cdot \sqrt{\sigma^2(W)}$. O cálculo da média e da variância da distribuição de W, assim como dos valores críticos de rejeição de H_0 podem ser encontrados em LEHMANN e D'ABRERA (1975). Se H_0 é rejeitada e $W \geq E(W) + z(\alpha) \cdot \sqrt{\sigma^2(W)}$, isso significa que as maiores diferenças absolutas $|X_A - X_B|$ calculadas são tais que $X_A > X_B$. Considerando que o algoritmo G está sendo utilizado para resolver um problema de minimização, conclui-se que a versão B é melhor que a versão A. Analogamente, se H_0 for rejeitada e $W \leq E(W) - z(\alpha) \cdot \sqrt{\sigma^2(W)}$, conclui-se que a versão A é melhor que a versão B.

O método de determinação de parâmetros proposto em XU, CHIU e GLOVER (1996) consiste da construção de uma árvore de busca em que cada nível corresponde à determinação de um dos parâmetros. O nó raiz dessa árvore representa uma combinação básica de parâmetros e correspondente ao que se espera ser melhor para o desempenho do algoritmo levando-se em consideração os trabalhos encontrados na literatura e outras hipóteses iniciais. Os nós em níveis mais profundos representam combinações alternativas dos parâmetros. Para se criar um novo nível nessa árvore, cada nó folha é ramificado variando-se o próximo parâmetro a ser determinado por toda a faixa escolhida para ele. Para eliminar algumas combinações de valores de parâmetros são utilizadas as relações de dominância estabelecidas pelos testes estatísticos descritos. Essa estratégia de determinação de parâmetros foi denominada Método de Crescimento e Poda de Árvore (MCPA) e é transcrita da referência citada para maior clareza.

- **Algoritmo MCPA**

Passo 1: Suponha que existem K parâmetros $(P_1, P_2, \ldots, P_K)$ a determinar e que eles estejam numerados em ordem decrescente de importância. Para cada um dos parâmetros P_k, escolha um conjunto de valores $V_k = (V_{k1}, V_{k2}, \ldots, V_{ktk})$.

Passo 2: Inicie a árvore de busca considerando a ramificação de valores do parâmetro P_1 a partir do nó raiz (que é considerado o único nó folha inicial). Defina o nível k = 1.

Passo 3: Para cada nó folha existente no nível k-1, ramifique t_k ramos onde cada ramo representa um dos valores de V_k. Realize os testes sobre todos os problemas para cada ramo e colete os resultados, criando um nó folha no nível k. Os nós do nível k-1 perdem seu estado de folha.

Passo 4: Para cada conjunto de nós folha no nível k que tenham sido ramificados a partir do mesmo nó pai, aplique o teste de Friedman para determinar os melhores valores possíveis. Pode (ou elimine) os nós folha que forem inferiores aos melhores valores determinados pelo teste.

Passo 5: Para cada par de nós folha remanescentes que não tenham sido ramificados a partir do mesmo nó pai e que ainda não tenham sido examinados nesse passo, aplique o teste de Wilkoxon. Se os membros de um par são significativamente diferentes, pode o que for inferior.

Passo 6: k = k + 1. Se k > K, vá para o passo 7. Caso contrário vá para o passo 3.
Passo 7: Cada um dos nós folha remanescentes representa uma das melhores configurações do algoritmo. Pare.

Na Figura 3.1 ilustra-se o processo de determinação de parâmetros para um algoritmo genérico com dois parâmetros a determinar: P_1 e P_2. No nó raiz, P_1 e P_2 assumem valores "default" determinados, de certa forma, arbitrariamente. No primeiro nível, varia-se o parâmetro P_1 mantendo P_2 fixo. Na figura, assume-se que P_1 possa variar entre quatro valores, gerando os nós 1 a 4 no nível 1. Nessa fase, executam-se os testes computacionais para cada nó, comparando os resultados obtidos através do teste de Friedman. Supondo que os nós 2 e 3 sejam eliminados, eles não devem ser considerados nos testes seguintes e são marcados com um X. Os nós 1 e 4, não eliminados no nível 1, são então ramificados, variando P_2 pelos valores possíveis. Novamente, aplica-se o teste de Friedman para cada um dos dois conjuntos de nós criados. Ou seja, aplica-se esse teste primeiramente entre os nós 5, 6 e 7 e depois entre os nós 8, 9 e 10. Supõe-se que os nós 5 e 10 tenham sido eliminados por esse teste e, novamente, eles são marcados com um X. Para poder comparar entre si os nós que tenham sido ramificados a partir de nós pais diferentes, utiliza-se o teste de Wilkoxon entre os pares de nós (6, 8), (6, 9), (7, 8) e (7, 9). A título de exemplo, suponha que apenas os nós 6 e 7 possam ser eliminados dessa forma e sejam marcados por um duplo X. Como há mais de um nó dominante no final da busca, há mais de uma configuração de parâmetros dominante. Pode-se, então, escolher aquela que apresenta a melhor qualidade.

Para aplicar o MCPA na determinação de parâmetros de GEN1 e GEN2, é preciso, inicialmente, estabelecer a ordem em que os parâmetros devem ser considerados para ramificação na árvore de busca, de acordo com sua influência no desempenho das implementações de Algoritmos Genéticos. Para tanto, foram realizados testes computacionais limitados com os 30 problemas de teste, tendo como finalidade determinar qual dos operadores genéticos (recombinação ou mutação) é responsável pela evolução dos indivíduos em direção ao resultado apresentado. Para esses testes foram utilizados os valores p_{rec} = 0,6 e 0,8 ,p_{mut} = 0,05 e 0,2 e I:T = 1:1.

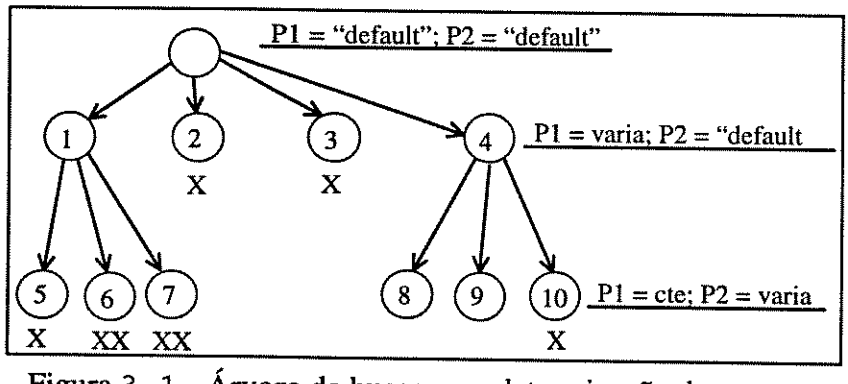

Figura 3.1 - Árvore de busca para determinação de P_1 e P_2.

Observou-se que, em GEN1, a mutação é responsável pelo refinamento do processo de evolução causando praticamente 100% das atualizações de **S*** (definido na seção 2.2.4) ao longo de 1000 gerações e que a recombinação atua como um operador auxiliar, obtendo novas soluções a cada geração mas sem conseguir atualizar **S***. Para testar se o operador de recombinação é realmente necessário para o bom funcionamento de GEN1, novos testes foram realizados com $p_{rec} = 0$. Observou-se que o desempenho cai, significando que apesar de não ser o operador dominante, a recombinação colabora positivamente para o desempenho do algoritmo. Sendo assim, em GEN1, devem-se determinar primeiramente os parâmetros que afetam o desempenho da mutação (p_{mut} e I:T) para depois ajustar o valor de p_{rec}.

Testes equivalentes foram feitos para GEN2 e obteve-se o resultado oposto. A recombinação é responsável por praticamente 100% das atualizações de **S*** e a mutação funciona como um operador secundário. Testou-se a retirada do operador de mutação ($p_{mut} = 0$), notando que o desempenho cai, como no caso de GEN1. Portanto, o ajuste de parâmetros em GEN2 deve começar por p_{rec} e depois passar para p_{mut} e I:T.

Tendo sido estabelecida a ordem de consideração dos parâmetros para GEN1 e GEN2, o MCPA foi aplicado a essas implementações com os seguintes valores de teste para cada parâmetro:

- $p_{rec} = 0,4; 0,6; 0,8$ e $1,0$
- $p_{mut} = 0,05$ a $0,8$ variando em intervalos iguais a $0,05$
- I:T = 2:1, 1:1 e 1:2.

Tradicionalmente são utilizados valores pequenos para p_{mut}, pois esse operador é considerado sempre como secundário (HOLLAND, 1995; GOLDBERG, 1989). Entretanto, o fato de que em GEN1 a mutação desempenha um papel não secundário leva a crer que, possivelmente devido à mudança na representação das soluções e, conseqüentemente, na estrutura dos próprios operadores genéticos, os valores de probabilidades de mutação e recombinação usados na literatura podem não ser os ideais. Como as implicações da inversão na importância dos operadores ainda não está muito clara nessa fase do trabalho, decidiu-se usar uma faixa larga para p_{mut} e intervalos pequenos entre um valor e outro.

O nível de confiança ($1 - \alpha$) para a realização dos testes estatísticos foi fixado em 95% e, portanto, $z(\alpha) = z(0,05) = 1,645$.

Após a construção de toda a árvore de testes, quando se chega ao último nível, a qualidade de cada combinação de parâmetros nesse nível é medida pela seguinte relação de ganho:

$$\text{Ganho} = 100. \frac{\text{Custo(H)} - \text{Custo(G)}}{\text{Custo(H)}} \qquad (3.5)$$

onde Custo(G) é o custo da solução encontrada por GEN1 (ou GEN2) para um determinado problema e Custo(H) é o custo obtido pelas heurísticas apresentadas no Capítulo 1.

Essa medida é ilustrada na Figura 3.2. Como o valor heurístico de custo representa

um limitante superior do custo ótimo de cada problema, a medida dada pela equação (3.5) diz quanto foi a redução no valor desse limitante, obtida com a aplicação dos Algoritmos Genéticos.

A qualidade média relativa a um determinado conjunto de parâmetros, então, é dada pela média dos valores de Ganho calculada sobre os 30 problemas de teste.

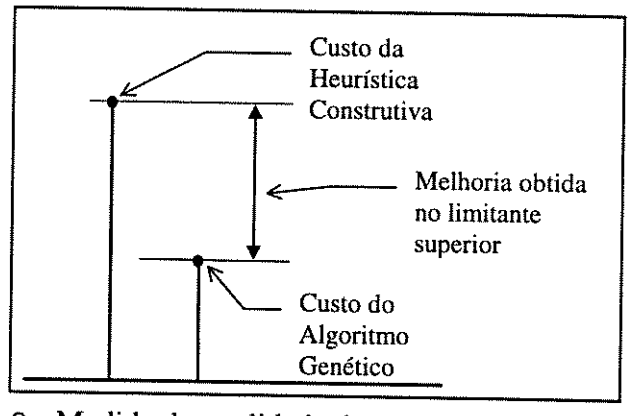

Figura 3.2 - Medida de qualidade dos resultados de GEN1 e GEN2

3.1.1. Parâmetros de GEN1

Nesta seção são apresentados os resultados da determinação de parâmetros de GEN1 utilizando-se o MCPA. Como combinação básica de parâmetros para o nó raiz, escolheram-se os valores: $p_{rec} = 0,6$ e $I:T = 1:1$. O valor de p_{mut} não foi especificado visto que é o primeiro parâmetro a ser determinado. A probabilidade de recombinação foi escolhida de acordo com o observado na literatura. Em geral, $p_{rec} = 0,6$ ou $0,8$, juntamente com valores baixos de p_{mut} geram bons resultados. Já o valor de $I:T$ foi escolhido arbitrariamente como o ponto intermediário dentro de sua faixa de variação.

Na Tabela 3.1 são apresentados os resultados obtidos para R_i variando-se p_{mut}. Nessa tabela, o índice i representa o número atribuído a cada nó da árvore de testes, na ordem de ramificação. Os valores calculados para R_i servem como base para o cálculo da estatística F_r.

$$F_r = 140,196 > \chi^2_{0,05;15} = 24,996 \qquad (3.6)$$

A desigualdade acima indica que a variação de p_{mut} afeta o desempenho de GEN1 a um nível de confiança de 95% e que, portanto, pode-se aplicar a segunda parte do teste de Friedman para determinar quais os valores dominantes para a probabilidade de mutação. Da Tabela 3.1: $i_{min} = 9$ e $R_{imin} = R_9 = 155,5$. Portanto, o nó i que possuir

$$R_i \geq R_9 + z(\alpha) \cdot \sqrt{J.I.(I+1)/6} = 216,165 \qquad (3.7)$$

pode ser desconsiderado dos próximos testes. O nós descartados são $i = 1$ a 6 que correspondem às menores probabilidades de mutação, como era esperado, já que a mutação é o principal operador genético de GEN1. Além disso, foram eliminados os nós de 8 e 13, indicando que os valores mais altos de p_{mut} também não são adequados. Para cada valor de

p_{mut} não eliminado pelo teste de Friedman, realizam-se novos testes para determinar a melhor proporção I:T.

Tabela 3.1 - Variação da probabilidade de mutação
(Nível 1 da árvore de testes)

i	p_{mut}	R_i
1	0,05	480
2	0,10	434
3	0,15	394,5
4	0,20	330
5	0,25	268
6	0,30	226
7	0,35	207,5
8	0,40	228
9	0,45	155,5
10	0,50	176
11	0,55	173,5
12	0,60	171,5
13	0,65	235
14	0,70	195,5
15	0,75	211
16	0,80	184

Os resultados dos testes computacionais variando I:T são apresentados na Tabela 3.2. Os valores de F_r obtidos mostram que esse parâmetro é praticamente irrelevante para o desempenho de GEN1 pois em apenas um caso F_r foi maior que $\chi^2_{0,05;2} = 5,991$. Isso mostra que a qualidade do operador de mutação é determinada unicamente pelo valor de p_{mut} e que as alterações provocadas pelos valores distintos de I:T não afetam o desempenho de GEN1 se p_{mut} for mantida constante. Portanto, o teste de Friedman elimina apenas dois nós no segundo nível da árvore de testes e a única forma de estabelecer alguma relação de dominância entre os demais nós é através do teste de Wilkoxon.

Após a realização desses testes, foram eliminados os nós: 17, 18, 19, 23, 26, 28, 29, 31 a 38 e 40. Cada um desses nós está marcado na Tabela 3.2 com um W na coluna Eliminado. O número que aparece entre parênteses após o W indica o nó dominante responsável pela eliminação. Os outros nós são ramificados para determinar os melhores valores de p_{rec}.

Na Tabela 3.3, observa-se que a maioria dos valores obtidos para F_r variando p_{rec} é bastante a $\chi^2_{0,05;3} = 7,815$, indicando que, para valores fixos de p_{mut} e I:T a variação de p_{rec} não causa variações de desempenho maiores que as causadas por outros fatores aleatórios. Entretanto, ao se aplicar o teste de Wilkoxon entre os pares de nós da Tabela 3.3 que tenham sido ramificados a partir de nós pais distintos, consegue-se eliminar uma grande quantidade de combinações de parâmetros e as combinações não eliminadas são apresentadas na Tabela 3.4. A coluna Ganho (%) é calculada como descrito anteriormente.

Tabela 3 . 2 - Variação dos valores de I : T para as probabilidades de mutação dominantes (Nível 2 da árvore de testes)

p_{mut}	i	I:T	R_i	F_r	Eliminado
	17	2:1	66		W(21)
0,35	18	1:1	53	2,867	W(21)
	19	1:2	61		W(21)
	20	2:1	67		F
0,45	21	1:1	42	16,467	
	22	1:2	71		F
	23	2:1	68		W(21)
0,50	24	1:1	57	3,267	
	25	1:2	55		
	26	2:1	63		W(21)
0,55	27	1:1	56	0,867	
	28	1:2	61		W(21)
	29	2:1	63		W(21)
0,60	30	1:1	58	0,467	
	31	1:2	59		W(21)
	32	2:1	56		W(21)
0,70	33	1:1	61	0,867	W(21)
	34	1:2	63		W(21)
	35	2:1	70		W(21)
0,75	36	1:1	56	5,067	W(21)
	37	1:2	54		W(21)
	38	2:1	69		W(21)
0,80	39	1:1	51	5,4	
	40	1:2	60		W(21)

Tabela 3 . 3 - Variação de p_{rec} para as combinações de p_{mut} e I : T não eliminadas (Nível 3 da árvore de testes)

p_{mut}	I:T	i	p_{rec}	R_i	F_r	Eliminado
		41	0,4	83		F
0,45	1:1	42	0,6	64	128,6	
		43	0,8	67		
		44	1,0	116		F
		45	0,4	89		W(42)
0,50	1:1	46	0,6	69,5	5,81	
		47	0,8	67		
		48	1,0	74,5		
		49	0,4	65		
0,50	1:2	50	0,6	73	3,68	
		51	0,8	79		W(42)
		52	1,0	83		W(42)
		53	0,4	78		W(42)
0,55	1:1	54	0,6	69	7,24	
		55	0,8	64		
		56	1,0	89		W(42)
		57	0,4	79		W(42)
0,60	1:1	58	0,6	68,5	2,33	
		59	0,8	81,5		W(42)
		60	1,0	71		W(42)
		61	0,4	77		W(42)
0,80	1:1	62	0,6	67	1,76	
		63	0,8	77		W(42)
		64	1,0	79		W(42)

Tabela 3 . 4 - Combinações de parâmetros não eliminadas pelo teste de Wilkoxon no último nível da árvore de testes de GEN1

i	p_{mut}	I:T	p_{rec}	Ganho(%)
42	0,45	1:1	0,6	6,37
43	0,45	1:1	0,8	2,24
46	0,50	1:1	0,6	3,95
47	0,50	1:1	0,8	1,43
48	0,50	1:1	1,0	3,95
49	0,50	1:2	0,4	3,01
50	0,50	1:2	0,6	4,41
54	0,55	1:1	0,6	3,78
55	0,55	1:1	0,8	4,20
58	0,60	1:1	0,6	4,49
62	0,80	1:1	0,6	3,73

Como todas as combinações da Tabela 3 . 4 são estatisticamente iguais a um nível de confiança de 95%, escolheu-se como melhor configuração para GEN1 aquela que apresenta o maior valor de Ganho, representada pelo nó 42: $p_{mut} = 0,45$, $I:T = 1:1$ e $p_{rec} = 0,6$.

3.1.2 - Parâmetros de GEN2

A construção da árvore de testes para determinação de parâmetros de GEN2 é basicamente igual à de GEN1, com exceção da ordem em que os parâmetros foram considerados. A partir do nó raiz, com $p_{mut} = 0,05$ e $I:T = 1:1$, foram criados quatro nós correspondentes aos quatro valores previstos para p_{rec}. Na Tabela 3 . 5 são apresentados os valores de R_i obtidos nesse nível. A estatística F_r foi calculada a partir desses valores:

$$F_r = 55,48 > \chi^2_{0,05;3} = 7,815 \qquad (3.8)$$

Tabela 3 . 5 - Variação da probabilidade de recombinação (Nível 1 da árvore de testes)

i	p_{rec}	R_i
1	0,4	63
2	0,6	53
3	0,8	64
4	1,0	120

Deve-se analisar os valores de R_i com relação a $R_{imin} = 53$ para verificar se alguns nós podem ser desconsiderados dos próximos testes. A relação $R_i \geq R_{imin} + z(\alpha) \cdot \sqrt{J.I.(I+1)/6}$ somente é verificada para $i = 4$, que não é ramificado. Os resultados da ramificação dos nós 1, 2 e 3 são mostrados na Tabela 3 . 6.

Tabela 3 . 6 - Variação de p_{mut} para as probabilidades de recombinação dominantes (Nível 2 da árvore de testes)

p_{rec}	i	p_{mut}	R_i	F_r	Eliminado
0,4	5	0,05	346	314,588	F
	6	0,10	173		F
	7	0,15	118		
	8	0,20	68,5		
	9	0,25	66		
	10	0,30	73,5		
	11	0,35	142		F
	12	0,40	205		F
	13	0,45	285		F
	14	0,50	377		F
	15	0,55	425		F
	16	0,60	408		F
	17	0,65	347,5		F
	18	0,70	329		F
	19	0,75	360,5		F
	20	0,80	356		F
0,6	21	0,05	277	337,513	F
	22	0,10	137		F
	23	0,15	70		W(8)
	24	0,20	62		W(8)
	25	0,25	75		W(7)
	26	0,30	144		F
	27	0,35	235		F
	28	0,40	154		F
	29	0,45	382,5		F
	30	0,50	423		F
	31	0,55	412		F
	32	0,60	395		F
	33	0,65	358		F
	34	0,70	331		F
	35	0,75	325,5		F
	36	0,80	299		F
0,8	37	0,05	193	129,374	F
	38	0,10	83,5		W(7)
	39	0,15	46		W(7)
	40	0,20	81,5		W(7)
	41	0,25	143		F
	42	0,30	254,5		F
	43	0,35	305		F
	44	0,40	367		F
	45	0,45	410		F
	46	0,50	417		F
	47	0,55	397		F
	48	0,60	348		F
	49	0,65	327		F
	50	0,70	278		F
	51	0,75	214,5		F
	52	0,80	215		F

Para os três valores de p_{rec}, a estatística F_r é maior que $\chi^2_{0,05;15} = 24,996$. Portanto, existe a possibilidade de eliminação de combinações de probabilidades através do teste de Friedman. Todas as combinações eliminadas dessa forma são marcadas com um F na

coluna Eliminado da Tabela 3.6. Apesar do número de nós restantes nesse nível não ser muito grande, o teste de Wilkoxon foi aplicado entre todos os pares de nós com diferentes valores de p_{rec}. Dessa forma, conseguiu-se eliminar mais algumas combinações de probabilidades, marcadas com um W na coluna Eliminado. Os nós restantes foram, então ramificados, variando-se o parâmetro I:T. Os resultados obtidos são apresentados na Tabela 3.7.

Nenhum dos valores de F_r apresentados na Tabela 3.7 é maior que $\chi^2_{0,05;2} = 5,991$ e, portanto, nenhum dos nós do nível 3 pode ser eliminado pelo teste de Friedman. Aplicando o teste de Wilkoxon entre os pares de nós, foram eliminados aqueles para os quais há um W na coluna Eliminado. As combinações de parâmetros restantes são repetidas na Tabela 3.8, acompanhadas do valor calculado para Ganho.

Tabela 3.7 - Variação do parâmetro I:T para p_{rec} e p_{mut} constantes (Nível 3 da árvore de testes)

i	p_{rec}	p_{mut}	I:T	R_i	F_r	Eliminado
53			2:1	67		W(56)
54	0,4	0,15	1:1	60	3,267	W(56)
55			1:2	53		W(56)
56			2:1	66		W(59)
57	0,4	0,20	1:1	55	2,067	
58			1:2	59		W(60)
59			2:1	61		
60	0,4	0,25	1:1	58	0,200	
61			1:2	61		
62			2:1	68		W(59)
63	0,4	0,30	1:1	51	4,867	
64			1:2	61		W(59)

Tabela 3.8 - Combinações de parâmetros não eliminadas pelo teste de Wilkoxon no último nível da árvore de testes de GEN2

i	p_{rec}	p_{mut}	I:T	Ganho(%)
57	0,4	0,20	1:1	-23,97
59	0,4	0,25	2:1	-20,92
60	0,4	0,25	1:1	-25,00
61	0,4	0,25	1:2	-24,88
63	0,4	0,30	1:1	-22,14

As combinações de parâmetros na Tabela 3.8 são estatisticamente iguais a um nível de confiança de 95%. Portanto, escolhe-se a combinação que apresenta maior porcentagem de ganho, representada pelo nó 59: $p_{rec} = 0,4$, $p_{mut} = 0,25$ e I:T = 2:1.

Com os valores de p_{rec}, p_{mut} e I:T determinados, foram realizados testes adicionais para ajustar os valores de ind e ger. A necessidade de ajuste desses parâmetros fica clara quando se observa os valores de Ganho obtidos para GEN2. Os testes consistiram de variar o número de tarefas de n = 30 a n = 110 e avaliar o desempenho dos algoritmos para ind

= n, 2.n e 3.n combinados com ger = 1000, 2000 e 3000. Os melhores resultados foram obtidos com ind = 3.n e ger = 3000.

Tendo sido determinados os parâmetros das duas implementações de Algoritmos Genéticos, passa-se a fase de avaliação de desempenho. Para tanto, GEN1 e GEN2 são aplicados a todo os problemas de teste dos conjuntos A e C, descritos na seção 1.4.1.

3.2 - DESEMPENHO

O desempenho de GEN1 e GEN2 é avaliado principalmente por dois fatores: qualidade de solução e tempo computacional. A qualidade de solução é medida da seguinte forma. Cada problema de teste é resolvido cinco vezes, variando-se a semente fornecida ao gerador de números aleatórios. Isso é feito com o objetivo de bloquear o efeito da grande quantidade de números aleatórios gerados durante a execução de GEN1 e GEN2. Sendo assim, é possível que, para um mesmo problema, GEN1 (GEN2) convirja para 5 soluções distintas, uma para cada semente fornecida. Portanto, no calculo de Ganho, pela expressão (3.5), Custo(G) é o valor da média aritmética dos atrasos dessas 5 soluções.

Os valores de Ganho calculados para os problemas de teste são então agregados por tamanho de problema (determinado por m e n) e por tipo de problema (determinado pela restritividade das datas de entrega). Finalmente, é calculada uma média geral de ganho por tamanho de problema.

O tempo que GEN1 (GEN2) demora para resolver cada problema de teste é determinado pela média dos tempos medidos nas 5 vezes em que ele é aplicado ao problema. Da mesma forma que é feito para a porcentagem de ganho, calcula-se, para cada conjunto de problemas de mesmo tamanho, uma média de tempo por tipo e uma média geral para os 30 problemas de mesmo tamanho.

A avaliação do desempenho de GEN1 e GEN2 foi feita em duas fases distintas. Na primeira delas, os algoritmos foram aplicados aos problemas de teste do Conjunto C, que são problemas de pequeno porte, para que fosse possível comparar os resultados finais com aqueles obtidos pelo pacote CPLEX. Esse pacote comercial contém rotinas para resolver modelos de Programação Inteira Mista como o modelo do problema PST, dado pelas expressões 1.2 a 1.9. Os resultados da comparação dos resultados de GEN1 e GEN2 com os resultados dados pelo CPLEX são apresentados na seção 3.2.1.

Na segunda fase de avaliação do desempenho das implementações de Algoritmos Genéticos, elas foram aplicadas a todos os problemas de teste do Conjunto A, que são problemas de grande porte. Os resultados obtidos para esses problemas são comparados com os limitantes superiores dados pelas heurísticas ATCS (para m = 1) e SAA (para m > 1). Os resultados dessas comparações são apresentados nas seções 3.2.2 e 3.2.3.

3.2.1 - Desempenho de GEN1 e GEN2 para problemas do Conjunto C

Para a resolução dos problemas de teste do Conjunto C, o pacote CPLEX foi executado em uma estação SUN Sparc 1000 e seus parâmetros foram ajustados para que o processo de solução fosse interrompido após 30 minutos ou 100.000 nós e, dessa forma, alguns dos problemas de teste não puderam ser resolvidos até a otimalidade. Nesses casos, as soluções dadas por GEN1 e GEN2 foram comparadas com a solução incumbente dada pelo CPLEX. Na Tabela 3.9 são apresentados os valores médios de desvio das soluções dos algoritmos genéticos implementados para as soluções ótimas e quase ótimas obtidas pelo CPLEX. Esse desvio é calculado como:

$$\text{Desvio} = 100. \frac{\text{Custo(G)} - \text{Custo(CPLEX)}}{\text{Custo(CPLEX)}} \qquad (3.9)$$

Para os problemas em que a solução do CPLEX apresenta um custo pior que o custo dado por GEN1 ou GEN2 o desvio calculado por (3.9) é negativo e esses problemas não são considerados no cálculo do desvio médio apresentado na Tabela 3.9. O número de problemas para os quais isso ocorre é indicado entre parêntesis, logo após os valores de desvio médio de GEN1 e GEN2. Quando a solução do CPLEX não é melhor que a de GEN1 ou GEN2 para todos os problemas de um determinado tamanho (m,n), o desvio (3.9) é considerado nulo para esse grupo de problemas.

Tabela 3.9 - Desempenho de GEN1 e GEN2 para os problemas do Conjunto C

n	m	Tempo CPLEX (s)	Desvio Médio GEN1 (%)	Tempo GEN1 (s)	Desvio Médio GEN2 (%)	Tempo GEN2 (s)
6	1	6,01	0,27	1,00	0,33	3,40
	2	40,21	1,17	1,08	0,36	3,70
	3	162,53	0,30	1,13	0,25	3,75
	4	616,18	0,01	1,16	0,10	3,77
	5	1174,12	0	1,18	0	3,74
	6	1723,55	0(1)	1,23	0(1)	3,73
Média		620,43	0,29	1,13	0,17	3,68
8	1	1552,38	1,06(5)	1,09	0,45(4)	4,00
	2	1783,73	0,33(29)	1,25	1,85(29)	4,15
	3	1775,90	3,78(28)	1,33	3,99(28)	4,41
	4	1800,00	0(30)	1,37	0(30)	4,50
	5	1800,00	0,33(29)	1,41	1,53(29)	4,50
	6	1800,00	0(30)	1,42	0(30)	4,61
Média		1752,00	0,92	1,31	1,30	4,36
10	1	1791,05	0(30)	1,36	0(30)	5,00
	2	1800,00	0(30)	1,44	0(30)	5,09
	3	1800,00	0(30)	1,53	0(30)	5,13
	4	1800,00	0(30)	1,60	0(30)	5,64
	5	1800,00	0(30)	1,67	0(30)	5,64
	6	1800,00	0(30)	1,68	0(30)	5,57
Média		1798,51	0	1,55	0	5,35

Observa-se que GEN1 e GEN2 são capazes de encontrar boas soluções para os

problemas de teste com n = 6 e n = 8 em um intervalo de tempo computacional bastante pequeno. As médias de desvio para essas duas implementações são bastante parecidas, com GEN1 apresentando desvios ligeiramente menores. Também do ponto de vista do tempo computacional, GEN1 apresenta uma convergência entre 60% e 70% mais rápida que GEN2 para os problemas do Conjunto C. Para aqueles problemas em que a solução do CPLEX não é melhor que a solução dada por GEN1 e GEN2, foi calculado o valor de Ganho para essas implementações através da expressão (3.5), onde se substitui Custo(H) por Custo(CPLEX). Os ganhos médios obtidos por GEN1 são iguais a 0,96% para n = 6, 15,89% para n = 8 e 49,49% para n = 10. Os ganhos médios de GEN2 são: 0,96 para n = 6, 15,95% para n = 8 e 49,22% para n = 10. Com relação a esses ganhos médios, praticamente não há diferença entre GEN1 e GEN2. Entretanto, o resultado geral do experimento mostra que GEN1 tem uma tendência a convergir mais rapidamente que GEN2, alcançando soluções de melhor custo.

Prof. G. L. Ragatz forneceu um outro conjunto de problemas de teste com m = 1 e n = 15, 25, 35 e 45. Além dos dados dos problemas, foram fornecidas as soluções ótimas (ou incumbentes, quando o ótimo não foi alcançado) obtidas pelo Branch & Bound de RAGATZ (1989). Esses problemas foram resolvidos por GEN1 e GEN2 com o objetivo de comparar essas implementações com a Busca Genética de RUBIN e RAGATZ (1995). Os resultados desse experimento mostraram que, para os problemas menores, com 15 tarefas, a Busca Genética é um pouco superior a GEN1 e GEN2. O desvio médio para o ótimo da Busca Genética é 1,59% enquanto que o de GEN1 é 4,23% e o de GEN2 é 4,57%. Para os demais problemas, GEN1 e GEN2 apresentam desempenhos bastante parecidos com o da Busca Genética, com diferenças inferiores a 1% no desvio médio com relação às soluções do Branch & Bound. Em termos do tempo computacional, novamente GEN1 é superior a GEN2, resolvendo os problemas de teste em menos de 1 minuto. A média de tempo para GEN2 fica em torno de 3 minutos.

A seguir são apresentados os resultados da aplicação de GEN1 e GEN2 sobre os problemas de teste de grande porte.

3.2.2 - Desempenho de GEN1 para problemas do Conjunto A

Nas tabelas 3.10 a 3.15 são apresentados os resultados obtidos nos testes com GEN1 sobre os problemas de teste do Conjunto A. Cada tabela contém resultados associados a um número fixo de máquinas. As colunas "Média Heur" e "Média GEN1" apresentam as médias de atraso obtidas pelas heurísticas do Capítulo 1 e por GEN1, respectivamente. Essas colunas tem a finalidade de mostrar a ordem de grandeza dos atrasos das soluções obtidas para melhor avaliação das porcentagens de ganho apresentadas. O gráfico da figura 3.3 mostra os valores da coluna "Ganho Médio" das tabelas 3.10 a 3.15.

Tabela 3.10 - Resultados obtidos para GEN1 com m = 1

n	tipo	Média Heur	Média GEN1	Ganho/ Tipo(%)	Ganho Médio(%)	Tempo/ Tipo(s)	Tempo Médio(s)
30	1	5174,7	4760,24	7,82	19,92	9,98	9,39
	2	2898,9	2398,88	17,94		10,44	
	3	1325,8	1078,18	33,99		7,75	
50	1	12046,4	11055,70	8,80	18,48	45,33	42,47
	2	7237,7	6585,90	8,79		46,86	
	3	1335,0	899,28	37,85		35,21	
70	1	25394,2	24421,20	3,90	13,28	115,14	106,98
	2	12129,2	11004,30	12,77		19,56	
	3	3371,4	2924,44	23,16		86,24	
90	1	43514,7	41887,40	3,87	13,85	242,29	233,17
	2	17532,8	16506,00	6,95		24,94	
	3	2825,1	2252,06	30,73		212,300	
110	1	64181,4	62412,96	2,79	15,30	328,83	292,17
	2	27541,3	25856,48	8,08		331,73	
	3	2480,6	1885,50	35,03		215,96	

Tabela 3.11 - Resultados obtidos para GEN1 com m = 2

n	tipo	Média Heur	Média GEN1	Ganho/ Tipo(%)	Ganho Médio(%)	Tempo/ Tipo(s)	Tempo Médio(s)
30	1	3356,0	2924,12	13,99	22,55	10,55	10,04
	2	1835,7	1519,64	19,17		10,26	
	3	1129,1	818,80	34,48		9,32	
50	1	8082,7	7216,20	10,61	20,53	42,72	42,08
	2	4625,3	3852,62	17,02		43,76	
	3	1802,1	1313,04	33,94		39,74	
70	1	14031,0	12652,46	9,95	24,45	109,47	103,13
	2	8002,7	6782,42	17,42		113,37	
	3	2440,8	1653,78	45,97		86,57	
90	1	23266,1	2054,54	12,10	17,82	245,61	212,50
	2	11240,0	8117,42	22,41		240,77	
	3	2529,5	1889,82	18,96		151,13	
110	1	30967,3	2773,10	10,36	16,60	323,32	293,62
	2	14823,8	11618,26	23,82		318,02	
	3	2532,7	1900,84	15,61		239,51	

Tabela 3.12 - Resultados obtidos para GEN1 com m = 3

n	tipo	Média Heur	Média GEN1	Ganho/ Tipo(%)	Ganho Médio(%)	Tempo/ Tipo(s)	Tempo Médio(s)
30	1	2456,8	2288,12	6,78	12,33	11,31	10,54
	2	1574,6	1413,84	9,98		10,50	
	3	551,2	445,06	20,24		9,82	
50	1	5876,9	5467,20	7,07	12,66	40,74	42,32
	2	3530,3	3109,30	12,97		45,56	
	3	1190,9	941,38	17,92		40,67	
70	1	10163,7	9319,98	8,30	15,04	113,63	111,46
	2	5580,1	4830,14	13,90		120,54	
	3	1440,0	1175,02	22,93		100,19	
90	1	14707,4	1367,98	6,99	16,82	226,15	211,49
	2	7971,0	7020,96	12,52		223,21	
	3	1972,1	1453,96	30,96		185,10	
110	1	22043,1	20231,00	8,24	15,58	323,33	314,46
	2	10051,2	8409,30	16,62		353,25	
	3	3572,6	2861,14	21,88		266,81	

Tabela 3.13 - Resultados obtidos para GEN1 com m = 4

n	tipo	Média Heur	Média GEN1	Ganho/ Tipo(%)	Ganho Médio(%)	Tempo/ Tipo(s)	Tempo Médio(s)
30	1	1736,9	1627,22	6,35	10,77	11,55	11,92
	2	1181,6	1098,74	8,14		12,18	
	3	941,0	785,54	17,83		12,03	
50	1	4476,6	4174,04	6,82	8,79	45,15	43,49
	2	2466,6	2210,36	10,72		46,47	
	3	780,7	670,56	8,84		38,85	
70	1	7421,9	6907,12	6,85	9,12	105,29	109,35
	2	4243,1	3749,20	11,17		121,54	
	3	1947,7	1685,56	9,35		101,22	
90	1	13609,9	12655,44	6,88	12,79	235,65	220,58
	2	6778,8	6019,82	11,04		231,17	
	3	1931,4	1633,90	20,44		194,93	
110	1	16961,3	15798,22	6,98	8,88	297,51	286,98
	2	9782,0	8995,14	8,55		330,69	
	3	1625,3	1439,92	11,12		232,75	

Tabela 3.14 - Resultados obtidos para GEN1 com m = 5

n	tipo	Média Heur	Média GEN1	Ganho/ Tipo(%)	Ganho Médio(%)	Tempo/ Tipo(s)	Tempo Médio(s)
30	1	1854,9	1752,88	5,58	7,94	11,42	11,42
	2	1146,2	1080,34	5,90		10,64	
	3	891,2	797,14	12,33		12,20	
50	1	3726,5	3555,92	4,62	8,41	45,69	44,68
	2	2444,1	2240,28	9,50		45,22	
	3	1370,4	1235,30	11,12		43,13	
70	1	6964,4	6495,86	6,74	8,30	120,28	117,89
	2	3679,1	3348,10	9,09		121,64	
	3	1199,2	1043,26	9,05		111,75	
90	1	10528,6	9835,64	6,58	7,94	255,63	236,51
	2	5242,8	4746,06	10,03		245,09	
	3	1741,1	1556,58	7,21		208,80	
110	1	14672,3	13989,82	4,80	6,40	313,50	307,49
	2	6753,2	6030,42	10,73		325,58	
	3	2361,1	2082,16	3,68		283,39	

Tabela 3.15 - Resultados obtidos para GEN1 com m = 6

n	tipo	Média Heur	Média GEN1	Ganho/ Tipo(%)	Ganho Médio	Tempo/ Tipo(s)	Tempo Médio(s)
30	1	1688,8	1610,40	4,72	6,04	12,12	12,29
	2	1412,0	1312,94	7,42		13,21	
	3	833,0	786,90	6,00		11,85	
50	1	3770,6	3589,18	4,97	7,58	45,07	45,83
	2	1939,9	1806,22	6,91		46,56	
	3	936,9	855,88	10,88		45,86	
70	1	5404,8	5149,32	4,63	7,16	117,00	112,48
	2	3936,0	3705,06	6,11		112,63	
	3	1127,1	1032,80	10,74		107,80	
90	1	9541,5	8993,58	5,75	1,04	242,29	233,17
	2	4463,6	4118,06	7,83		244,94	
	3	1737,4	1575,40	-10,46		212,30	
110	1	11930,9	11329,40	4,96	4,01	321,65	302,46
	2	6401,1	6010,76	5,86		311,65	
	3	1495,4	1331,64	1,19		274,52	

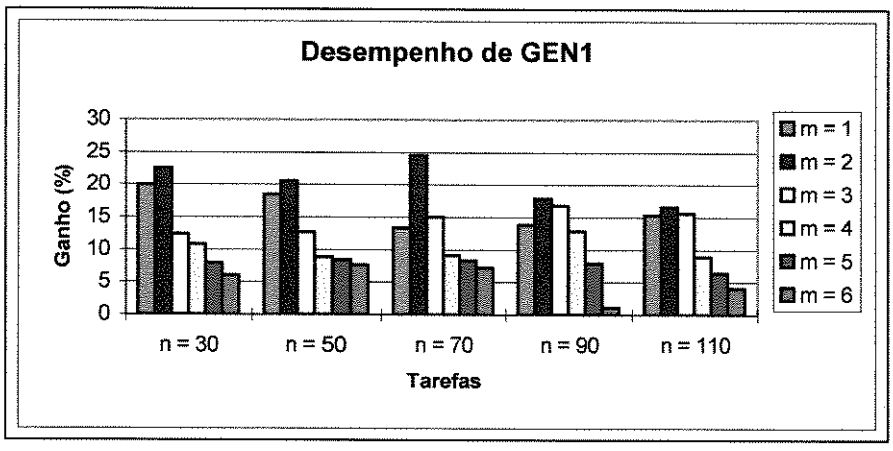

Figura 3.3 - Gráfico do Ganho Médio de GEN1

Os valores médios de Ganho são bastante satisfatórios mas, para fazer uma análise de desempenho de GEN1 é preciso, primeiramente, fazer algumas hipóteses sobre o desempenho da heurística SAA. Pela forma como essa heurística constrói as soluções para os problemas de teste com m > 1 é de se esperar que os erros de designação de tarefas a máquinas sejam mais numerosos quando o número de tarefas é grande. Entretanto, quando o número de máquinas aumenta pode-se esperar que o impacto desses erros no valor final de atraso seja menor pois cada máquina processa um número menor de tarefas. Quando isso ocorre, o passo de melhoria, onde são testadas trocas de tarefas entre máquinas, é capaz de corrigir os erros de designação mais facilmente. Os valores de ganho representados na Figura 3.3 mostram que GEN1 possui melhor desempenho com relação à heurística construtiva justamente para problemas em que ela deve enfrentar mais dificuldade em obter boas soluções. Isso indica que, se os erros médios de GEN1 com relação à solução ótima pudessem ser calculados para todos os problemas testados, os valores desses erros se manteriam, provavelmente dentro de uma faixa estreita, apresentando pouca variação com relação aos diversos tamanhos de problema.

Uma informação complementar ao valor do ganho médio é o número de sucessos de GEN1, ou seja, o número de problemas para os quais obteve-se Custo(G) < Custo(H). A porcentagem de sucessos de GEN1 foi calculada para cada valor de m e os valores obtidos foram: 92,67% para m = 1; 96,67% para m = 2; 95,67% para m = 3; 94,67% para m = 4; 94% para m = 5 e 92% para m = 6. A média geral é 94,28%.

Os tempos computacionais obtidos são satisfatórios, indicando que GEN1 pode ser aplicado a problemas ainda maiores que os testados, sem perda de eficiência computacional. Da Tabela 3.15, observa-se que o maior problema de teste (110 tarefas, 6 máquinas) pode ser resolvido em pouco mais de 5 minutos. O gráfico da Figura 3.4 mostra o crescimento do tempo computacional de GEN1 com relação ao tamanho dos problemas.

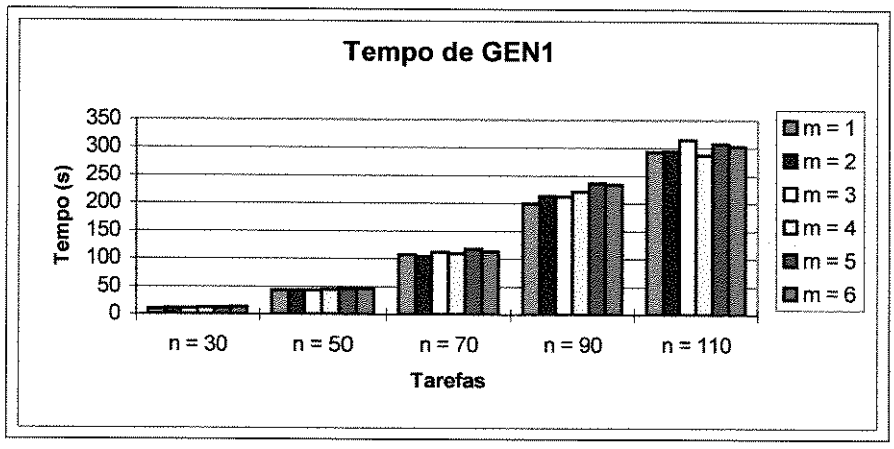

Figura 3.4 - Gráfico do Tempo Computacional de GEN1

3.2.2 - Desempenho de GEN2 para problemas do Conjunto A

Os resultados obtidos nos testes com GEN2 são apresentados nas tabelas 3.16 a 3.21. O gráfico da figura 3.5 mostra os valores da coluna "Ganho Médio" dessas tabelas.

Tabela 3.16 - Resultados obtidos para GEN2 com m = 1

n	tipo	Média Heur	Média GEN2	Ganho/ Tipo(%)	Ganho Médio(%)	Tempo/ Tipo(s)	Tempo Médio(s)
30	1	5174,7	5560,76	-7,18	11,06	24,64	23,20
	2	2898,9	2662,10	10,53		26,16	
	3	1325,8	1140,60	29,84		18,80	
50	1	12046,4	12225,30	-0,57	10,73	106,34	100,77
	2	7237,7	7169,78	-0,08		107,25	
	3	1335,0	982,68	32,85		88,74	
70	1	25394,2	26486,30	-4,09	5,22	234,93	234,88
	2	12129,2	12071,70	2,57		244,75	
	3	3371,4	3180,86	17,18		224,96	
90	1	43514,7	45436,50	-4,17	1,95	413,70	411,10
	2	17532,8	18866,30	-8,69		426,08	
	3	2825,1	2720,34	18,70		393,52	
110	1	64181,4	78595,82	-21,62	-18,14	324,48	350,28
	2	27541,3	31874,32	-16,52		373,71	
	3	2480,6	3368,14	-16,28		352,66	

Tabela 3.17 - Resultados obtidos para GEN2 com m = 2

n	tipo	Média Heur	Média GEN2	Ganho/ Tipo(%)	Ganho Médio(%)	Tempo/ Tipo(s)	Tempo Médio(s)
30	1	3356,0	2943,92	13,24	22,05	22,65	20,59
	2	1835,7	1517,24	19,49		21,15	
	3	1129,1	834,20	33,40		17,98	
50	1	8082,7	7311,62	9,57	18,21	94,95	85,91
	2	4625,3	3965,42	14,32		88,80	
	3	1802,1	1355,20	30,74		73,98	
70	1	14031,0	12910,74	8,22	21,76	240,38	222,64
	2	8002,7	6864,54	16,00		234,94	
	3	2440,8	1728,58	41,07		192,61	
90	1	23266,1	21179,64	9,27	11,28	424,19	404,36
	2	11240,0	9454,38	19,34		433,60	
	3	2529,5	1979,22	5,24		355,30	
110	1	30967,3	30133,92	2,67	-4,49	524,87	516,68
	2	14823,8	13425,02	10,79		526,11	
	3	2532,7	2366,50	-26,93		499,07	

Tabela 3.18 - Resultados obtidos para GEN2 com m = 3

n	tipo	Média Heur	Média GEN2	Ganho/ Tipo(%)	Ganho Médio(%)	Tempo/ Tipo(s)	Tempo Médio(s)
30	1	2456,8	2317,32	5,62	12,47	21,47	20,39
	2	1574,6	1402,34	11,07		20,56	
	3	551,2	445,74	20,73		19,15	
50	1	5876,9	5553,06	5,54	9,98	86,64	83,27
	2	3530,3	3207,58	10,11		85,75	
	3	1190,9	970,86	14,29		77,41	
70	1	10163,7	9412,42	7,45	12,51	234,80	225,70
	2	5580,1	4941,86	11,50		227,58	
	3	1440,0	1208,52	18,56		214,74	
90	1	14707,4	13975,42	4,98	11,90	428,75	422,98
	2	7971,0	7242,08	9,29		427,38	
	3	1972,1	1519,24	21,44		412,81	
110	1	22043,1	21418,7	2,77	4,26	527,72	526,36
	2	10051,2	9450,62	5,23		528,96	
	3	3572,6	3274,26	4,79		522,41	

Tabela 3.19 - Resultados obtidos para GEN2 com m = 4

n	tipo	Média Heur	Média GEN2	Ganho/ Tipo(%)	Ganho Médio(%)	Tempo/ Tipo(s)	Tempo Médio(s)
30	1	1736,9	1634,44	5,75	8,89	21,17	21,71
	2	1181,6	1110,32	7,07		22,56	
	3	941,0	812,30	13,86		21,39	
50	1	4476,6	4238,02	5,45	5,80	91,63	85,71
	2	2466,6	2261,36	8,20		89,31	
	3	780,7	691,06	3,75		76,19	
70	1	7421,9	7047,24	4,94	6,04	235,40	231,55
	2	4243,1	3815,22	9,82		232,79	
	3	1947,7	1719,42	3,37		226,47	
90	1	13609,9	12913,60	4,98	8,85	423,09	422,96
	2	6778,8	6267,74	7,52		429,41	
	3	1931,4	1711,56	14,05		416,31	
110	1	16961,3	16722,24	1,48	-0,87	526,04	525,86
	2	9782,0	9736,68	0,65		528,09	
	3	1625,3	1660,38	-4,74		523,44	

Tabela 3.20 - Resultados obtidos para GEN2 com m = 5

n	tipo	Média Heur	Média GEN2	Ganho/ Tipo(%)	Ganho Médio(%)	Tempo/ Tipo(s)	Tempo Médio(s)
30	1	1854,9	1764,76	5,04	6,83	23,84	23,75
	2	1146,2	1079,08	5,87		25,72	
	3	891,2	818,84	9,60		21,69	
50	1	3726,5	3608,04	3,26	6,45	100,24	91,88
	2	2444,1	2280,38	8,00		91,99	
	3	1370,4	1269,44	8,08		83,41	
70	1	6964,4	6655,24	4,47	5,34	236,37	229,08
	2	3679,1	3416,98	7,27		242,05	
	3	1199,2	1073,44	4,29		208,84	
90	1	10528,6	10137,88	3,70	5,19	422,35	423,06
	2	5242,8	4937,20	6,27		432,17	
	3	1741,1	1635,40	5,60		414,67	
110	1	14672,3	14740,42	-0,35	-6,58	531,55	532,00
	2	6753,2	6625,34	1,79		534,03	
	3	2361,1	2458,20	-21,16		530,43	

Tabela 3.21 - Resultados obtidos para GEN2 com m = 6

n	tipo	Média Heur	Média GEN2	Ganho/ Tipo(%)	Ganho Médio(%)	Tempo/ Tipo(s)	Tempo Médio(s)
30	1	1688,8	1644,38	2,65	4,32	23,11	23,87
	2	1412,0	1322,42	6,810		25,84	
	3	833,0	807,04	3,49		22,66	
50	1	3770,6	3685,12	2,49	4,20	92,17	93,36
	2	1939,9	1864,30	3,50		96,48	
	3	936,9	887,64	6,62		91,43	
70	1	5404,8	5228,36	3,21	5,09	245,94	239,00
	2	3936,0	3762,54	4,69		250,43	
	3	1127,1	1067,80	7,38		220,52	
90	1	9541,5	9267,20	2,84	-3,34	426,18	422,54
	2	4463,6	4304,78	3,34		433,40	
	3	1737,4	1651,78	-16,18		408,05	
110	1	11930,9	12088,18	-1,42	-14,90	530,62	531,02
	2	6401,1	6540,84	-3,18		532,32	
	3	1495,4	1647,74	-40,09		530,11	

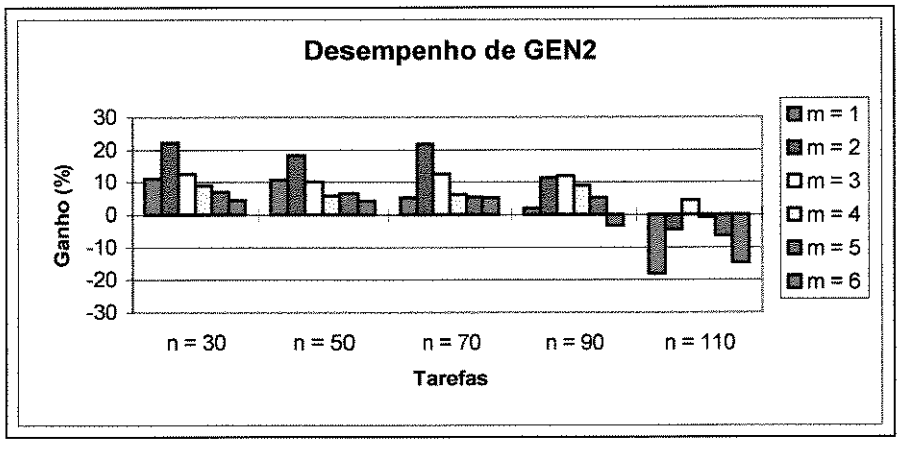

Figura 3.5 - Gráfico do Ganho Médio de GEN2

Observa-se que o ganho obtido com a aplicação de GEN2 aos problemas de teste é

menor que o de GEN1, especialmente para os problemas maiores, confirmando a tendência, observada em 3.2.1, de que a segunda implementação de algoritmos genéticos é inferior à primeira. Além disso, considerando os critérios de parada estabelecidos, GEN2 é bem mais lento que GEN1. Como pode ser visto na Figura 3.6, o tempo computacional utilizado para resolver os maiores problemas fica um pouco abaixo de 9 minutos.

Também foi calculada a taxa de sucessos de GEN2 com relação às heurísticas de comparação, ou seja, a porcentagem de problemas para os quais obteve-se Custo(G) < Custo(H). A porcentagem de sucessos de GEN2 foi calculada para cada valor de m e os valores obtidos foram: 42,67% para m = 1; 94% para m = 2; 94% para m = 3; 84% para m = 4; 82% para m = 5 e 69,33% para m = 6. A média geral é 77,67%, bastante inferior à obtida para GEN1.

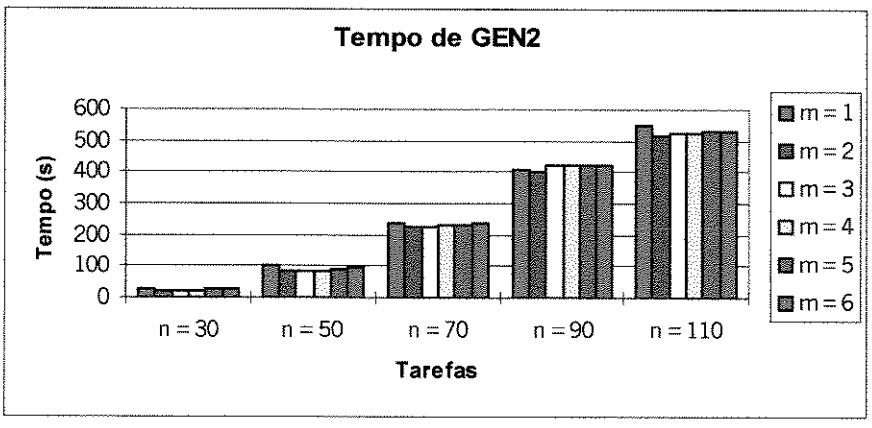

Figura 3.6 - Gráfico do Tempo Computacional de GEN2

3.2.4 - Comparação de Desempenho e Desvios Padrão

Para comparar o desempenho de GEN1 e GEN2 pode-se avaliar a diferença de ganho médio dessas duas implementações e determinar a partir de que valores de m e n essa diferença passa a ser mais significativa. Na Figura 3.7, são representados os valores calculados para (Ganho(GEN1) - Ganho(GEN2)). Observa-se que as diferenças aumentam quando o número de tarefas e de máquinas aumenta, indicando que GEN2, apesar de apresentar um desempenho melhor que as heurísticas ATCS e SAA para problemas com n < 110, não consegue soluções tão boas quanto as de GEN1 para os problemas testados.

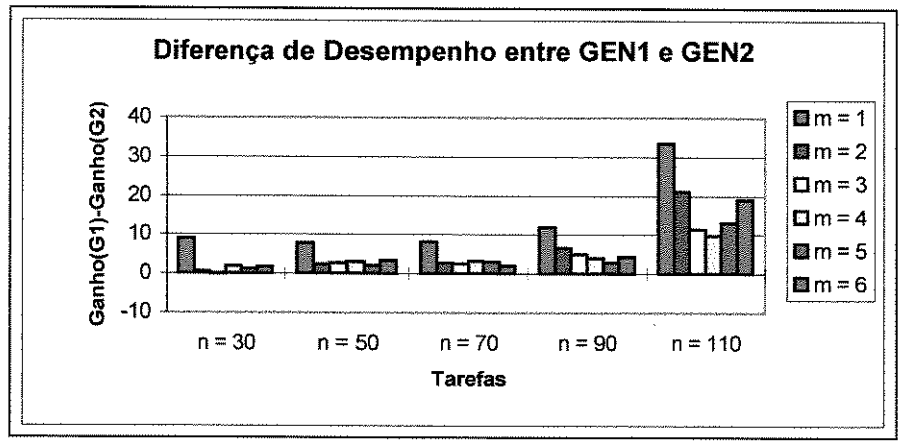

Figura 3.7 - Diferença de ganho entre GEN1 e GEN2

Nesse ponto é interessante comparar os algoritmos no que diz respeito à sua robustez. Isso pode ser feito através do cálculo dos desvios padrão de Ganho para cada grupo de 30 problemas de mesmo tamanho com relação ao "Ganho Médio" apresentado nas tabelas 3.10 a 3.21. Para que os desvios padrão de ganho de todos os problemas de teste possam ser comparados entre si, eles são normalizados com relação ao ganho médio. Portanto, para cada grupo de problemas de mesmo tamanho, calcula-se o coeficiente de variação de ganho como:

$$cv = (\text{desvio padrão})/(\text{ganho médio}) \qquad (3.10)$$

As figuras 3.8 e 3.9 mostram os coeficientes de variação calculados para GEN1 e GEN2, respectivamente.

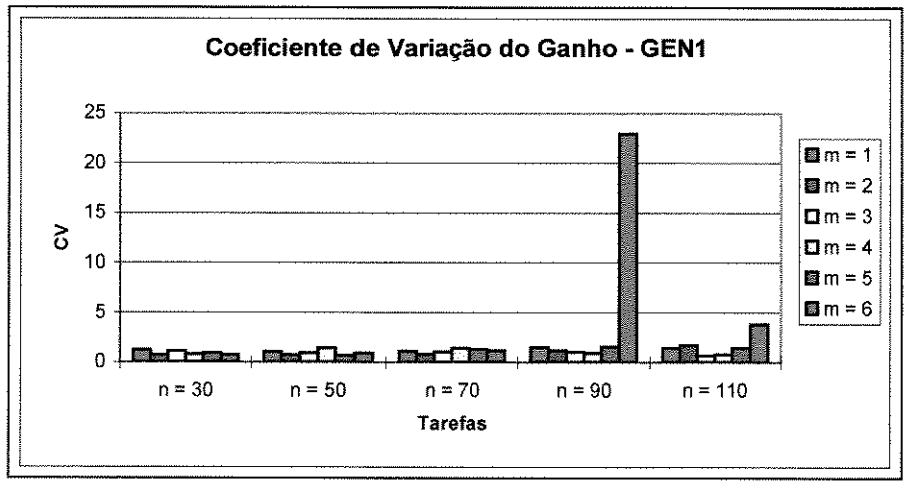

Figura 3.8 - Coeficiente de Variação do Ganho para GEN1

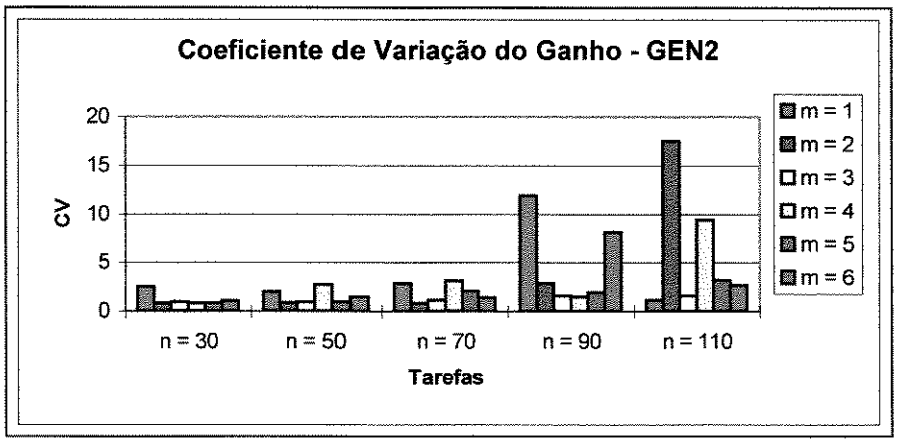

Figura 3.9 - Coeficiente de Variação do Ganho para GEN2

Observa-se que os coeficientes de variação de ganho de GEN1 são consistentemente mais baixos que o de GEN2, principalmente para os problemas com n = 90 e n = 110, indicando que a primeira implementação, além de gerar resultados melhores que a segunda, apresenta um pouco mais de robustez. O único conjunto de problemas que apresenta maior desvio nos valores de ganho para GEN1 é aquele composto pelos problemas com m = 6 e n = 90. O valor elevado do coeficiente de variação é devido a três problemas de teste para os quais a heurística SAA encontra soluções com atrasos muito baixos, iguais a 91, 185 e 157. Para esses mesmos problemas, GEN1 encontra soluções finais com valores de atraso iguais a 186, 264 e 219, respectivamente. Se esses problemas de teste forem desconsiderados no cálculo do coeficiente de variação, o novo valor seria 0,57%, bem abaixo do valor atual de 22,93%.

3.3 - COMPORTAMENTO

A análise de comportamento dos algoritmos genéticos implementados tem como objetivo investigar o processo evolutivo interno desses algoritmos. Essa análise pode ser feita de forma quantitativa e qualitativa. As medidas utilizadas na análise quantitativa são duas: sensibilidade a números aleatórios e número de gerações utilizadas para encontrar a melhor solução. Como suporte à análise qualitativa são utilizados gráficos que mostram a evolução dos custos da melhor solução, da mediana e do custo médio da população de soluções ao longo das gerações.

3.3.1 - Sensibilidade a números aleatórios

Dado que GEN1 e GEN2 são aplicados cinco vezes sobre cada problema de teste e o custo médio das soluções obtidas é usado como medida de desempenho, o desvio padrão dos custos finais com relação ao custo médio de cada problema pode indicar tendências de

estabilidade/instabilidade desses algoritmos com relação ao uso extenso de números aleatórios durante sua execução. Novamente, para que os desvios padrão de custo de todos os problemas de teste possam ser comparados entre si, todos são normalizados com relação ao custo médio do problema correspondente. Portanto, para cada problema calcula-se o coeficiente de variação de custo como:

$$cv = (\text{desvio padrão})/(\text{custo médio}) \qquad (3.11)$$

Os valores assim obtidos são então agrupados por tamanho de problema e podem ser representados de forma gráfica, como é feito nas figuras 3.10 e 3.11 para GEN1 e GEN2, respectivamente.

Esses gráficos mostram que GEN1 é ligeiramente mais estável que GEN2 com relação aos números aleatórios gerados durante o processo de evolução de soluções. Ambos os algoritmos apresentam menos sensibilidade conforme o número de máquinas aumenta, indicando que também se favorecem desse fator, como a heurística SAA.

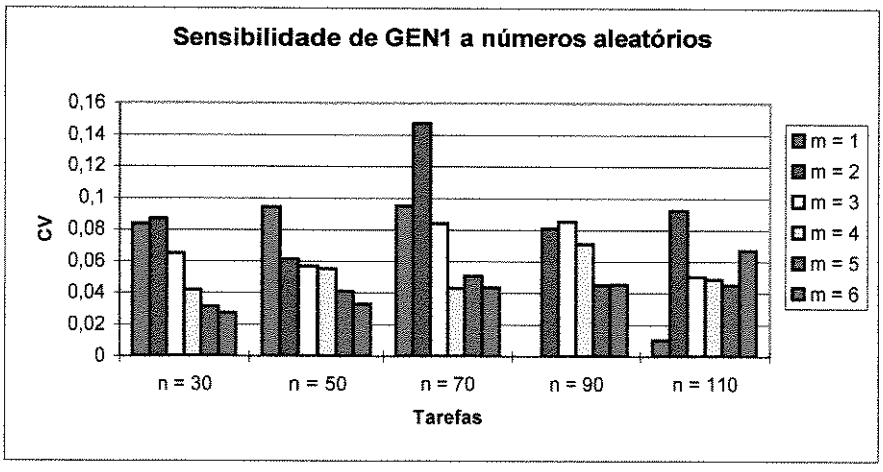

Figura 3.10 - Coeficientes de Variação de Custo para GEN1

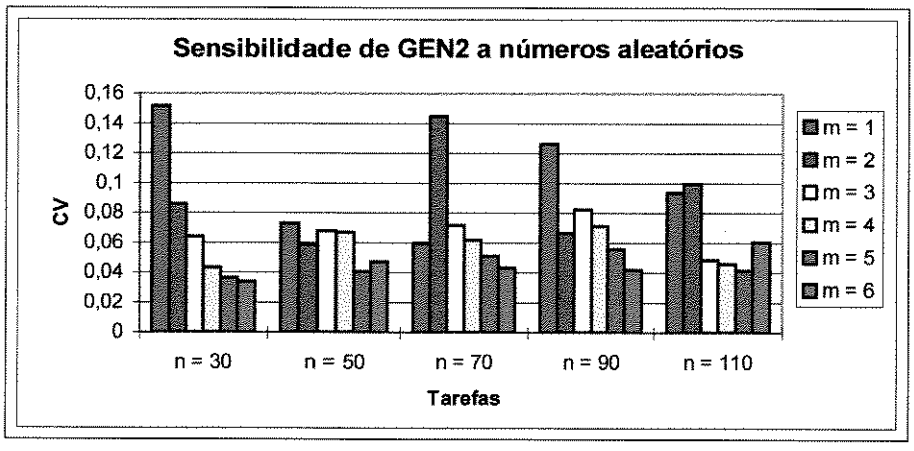

Figura 3.11 - Coeficientes de Variação de Custo para GEN2

3.3.2 - Gerações para o melhor

Os testes computacionais mostram que tanto GEN1 como GEN2 atingem um ponto do processo evolutivo em que não mais é possível atualizar a solução **S***. O número da geração em que isso ocorre mostra com que rapidez os algoritmos são capazes de convergir para a solução final, indicando a porcentagem de tempo realmente utilizada na busca até encontrar essa solução. Nas figuras 3.12 e 3.13 essas porcentagens são mostradas graficamente, agrupadas por tamanho de problema.

Essas figuras mostram que a convergência de GEN2 é mais lenta que a de GEN1. Esse fato, aliado a um desempenho um pouco mais pobre para todo o conjunto de problemas de teste, mostra que as características estruturais e operacionais de GEN2 levam a uma implementação de algoritmos genéticos inferior.

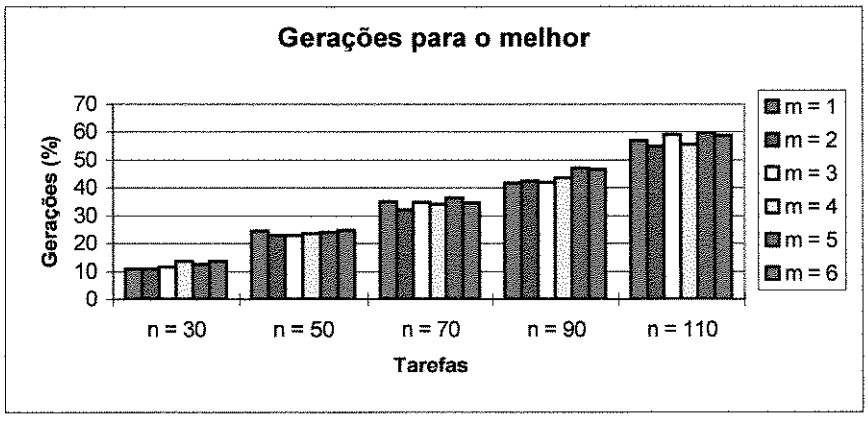

Figura 3.12 - Porcentagem de gerações utilizada por GEN1 para atualizar **S***

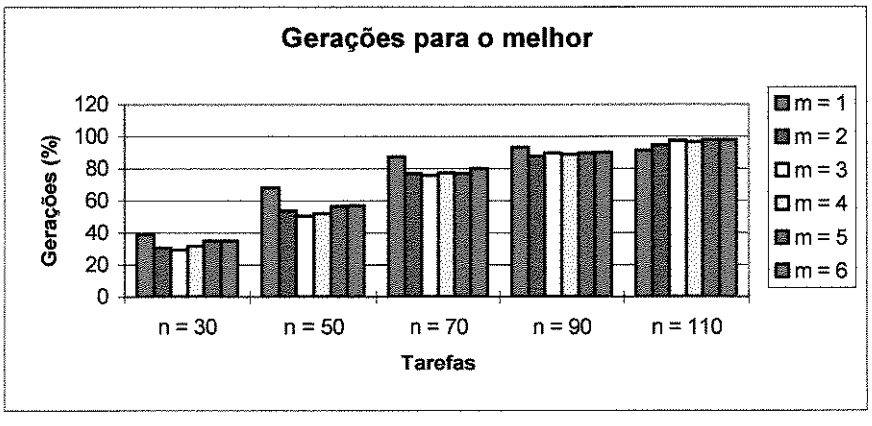

Figura 3.13 - Porcentagem de gerações utilizadas por GEN2 para atualizar **S***

3.3.3 - Análise qualitativa da busca

Um conjunto de testes computacionais limitados foram realizados com o intuito de se obter informações adicionais sobre o processo de busca realizada por GEN1 e GEN2. Esses testes tiveram por objetivo avaliar qualitativamente a busca realizada, visto que os resultados

finais foram bastante explorados.

Foram escolhidos aleatoriamente alguns problemas de teste, para os quais GEN1 e GEN2 foram aplicados somente uma vez. A cada geração t, a população P_t foi ordenada para que as seguintes informações pudessem ser extraídas: custo da melhor solução, custo médio da população e custo da solução mediana. A análise inicial dos resultados mostrou que, para todos os problemas testados, GEN1 e GEN2 possuem um comportamento bastante consistente e, por esse motivo, são apresentadas a seguir as conclusões tiradas para um problema com 4 máquinas e 110 tarefas.

Nas figuras 3.14 e 3.15 pode-se observar a evolução do melhor custo da população de soluções ao longo do processo evolutivo de GEN1 e GEN2, respectivamente. Esses gráficos são bastante típicos para algoritmos genéticos (GOLDBERG, 1989) mas apresentam diferenças entre si. A primeira delas é a velocidade de convergência, bem mais lenta em GEN2, e a outra é o próprio valor da solução final. Nesse problema em particular, GEN1 converge para uma solução com Custo(G) = 14908 e GEN2 converge para uma solução com Custo(G) = 16090, partindo da mesma população inicial aleatória.

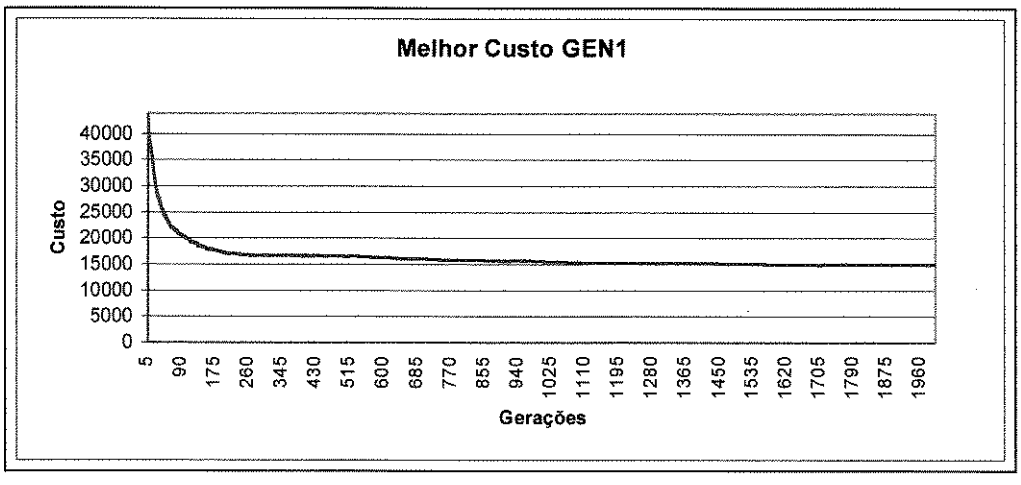

Figura 3.14 - Evolução do melhor custo da população para GEN1

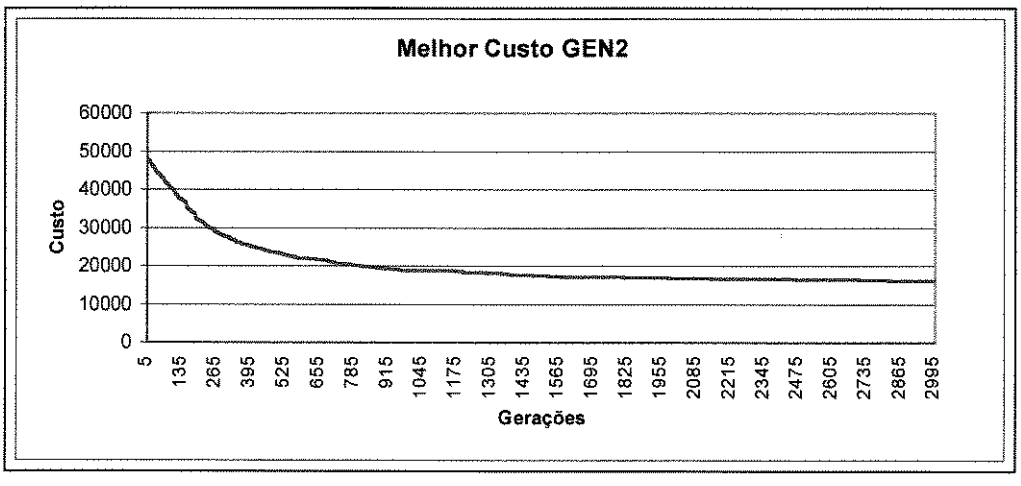

Figura 3.15 - Evolução do melhor custo da População para GEN2

As informações sobre a evolução da mediana e do custo médio de cada população, ao longo das gerações de GEN1 e GEN2, são mostradas nas figuras 3.16 e 3.17. Note que, para as duas implementações, a mediana de cada população fica muito próxima do melhor custo, e as duas linhas se sobrepõem nos gráficos. Isso mostra que pelo menos 50% das soluções possuem valores de custo próximos ou iguais ao da melhor solução.

Quanto ao custo médio, observa-se que ele não converge para um valor fixo mesmo nos estágios finais da busca, mostrando que a população não converge toda para a mesma solução. Isso é um aspecto positivo quando encarado do ponto de vista de diversificação de busca. Apesar de 50% das soluções ficarem sempre próximas da melhor solução, os outros 50% da população incluem soluções com custos um pouco mais altos e variáveis ao longo das gerações, possivelmente contendo informações genéticas bastante diferentes das contidas na melhor solução. Apesar dessa característica ser comum a GEN1 e GEN2, observa-se que a menor variabilidade do custo médio obtida por GEN1 permite que essa implementação continue a percorrer o espaço de soluções, buscando por melhores indivíduos, mesmo quando não consegue mais atualizar **S***. Para GEN2, entretanto, a variação do custo médio possui uma amplitude maior, indicando que a taxa de diversificação das soluções é bastante alta para as soluções com custo maior que o da mediana e que não há um mecanismo capaz de aproveitar a informação estrutural fornecida pelas soluções de custo alto.

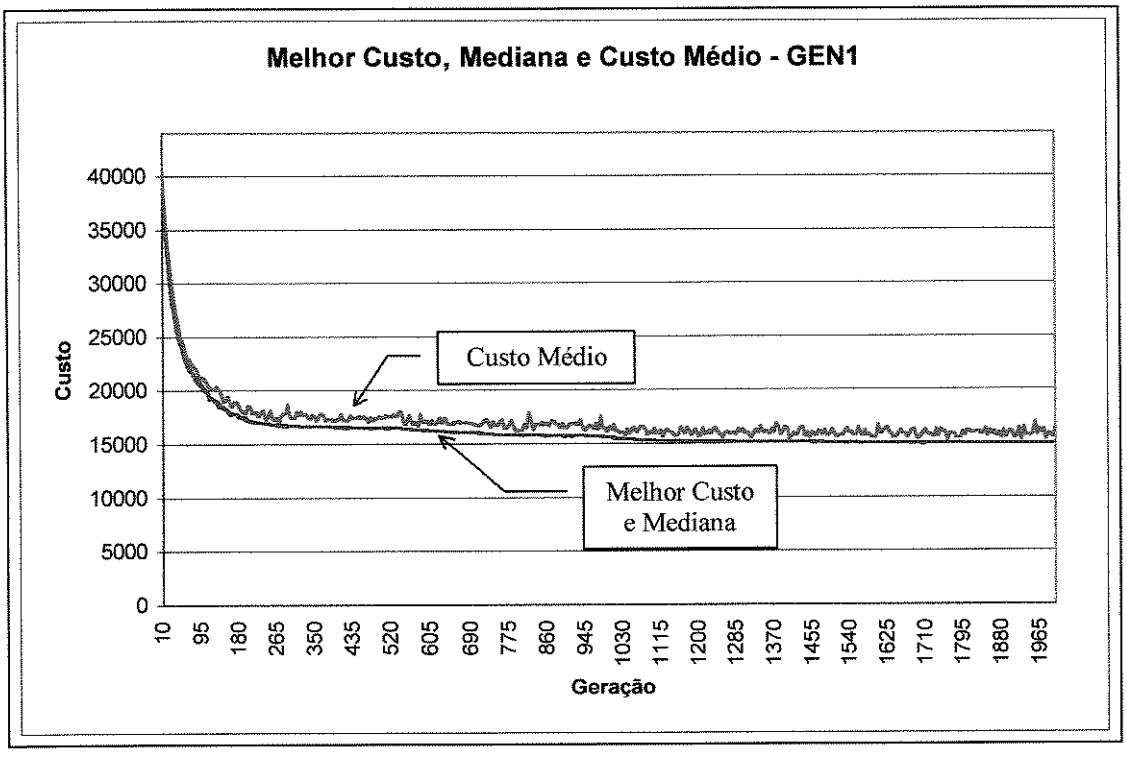

Figura 3.16 - Melhor custo, Custo médio e Mediana a cada geração para GEN1

Essas diferenças de comportamento podem ser parcialmente explicadas pela forma

como os operadores genéticos agem em conjunto, em cada implementação. Tanto em GEN1 como em GEN2, permite-se a repetição de soluções dentro de cada população. Dada a escolha probabilística feita pela roleta de seleção, pode-se esperar que as melhores soluções sejam escolhidas com mais freqüência que as soluções de pior custo, a cada iteração. Em GEN1, a população P'_t possui um número elevado de cópias das melhores soluções de P_t. Quando o operador de recombinação é aplicado, as soluções não selecionadas passam inalteradas para a população P''_t e as que tiverem sido selecionadas são emparelhadas para gerar novas soluções. É de se esperar que alguns pares de soluções, nesse ponto, sejam compostos por soluções pais idênticas, e a combinação não tem nenhum efeito sobre elas, que também passam inalteradas para P''_t. Sendo assim, quando a população P''_t está completa, ela também possui um número elevado de cópias das melhores soluções de P_t, embora esse número possa ser menor que o existente em P'_t. O operador de mutação, quando aplicado a várias cópias da mesma solução, age como uma busca local aleatória na vizinhança dessa solução. Como as soluções com mais cópias devem possuir os melhores custos, essa busca local é bastante frutífera, funcionando como um elemento de intensificação em torno das melhores soluções encontradas até o momento. Isso explica o fato do operador de mutação ser mais eficaz que o de recombinação para GEN1.

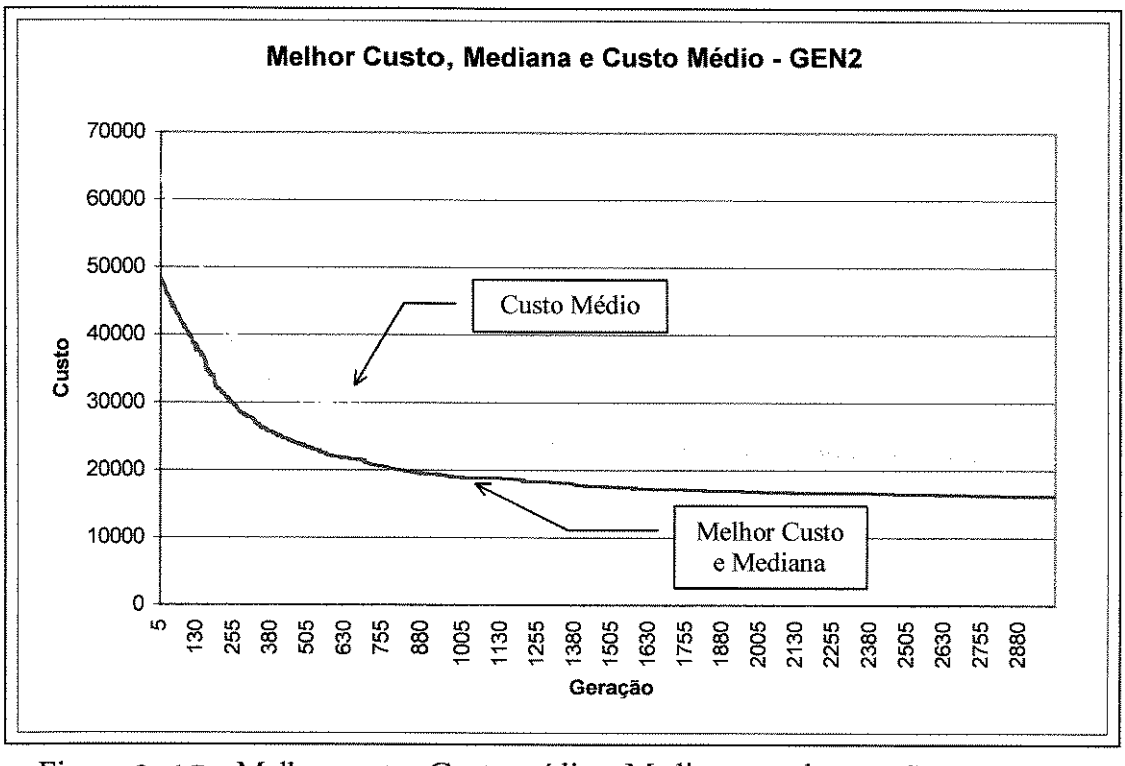

Figura 3.17 - Melhor custo, Custo médio e Mediana a cada geração para GEN2

Em GEN2, a estrutura proposta na Figura 2.3 parece não conseguir aproveitar o fato de existirem cópias das melhores soluções em P_t. Existe um aumento gradativo do número de

cópias do melhor indivíduo, como pode ser observado pelo gráfico da Figura 3.17, onde o custo da solução mediana e o melhor custo se sobrepõem. Entretanto, o fato da população convergir para as melhores soluções não permitiu aos operadores genéticos realizar uma boa intensificação em torno dessas soluções.

Os testes de comportamento apresentados nesta seção encerram a parte do trabalho relacionada à implementação de Algoritmos Genéticos para o problema PST. Nos capítulos seguintes, são apresentadas implementações da meta-heurística Busca por Espalhamento.

CAPÍTULO 4 - BUSCA POR ESPALHAMENTO APRESENTAÇÃO

Neste capítulo é apresentada uma implementação da meta-heurística populacional chamada Busca por Espalhamento (do inglês "scatter search"). Essa meta-heurística foi proposta por Fred Glover (GLOVER, 1977, GLOVER, 1994a e 1994b, GLOVER e LAGUNA, 1997) para a resolução de problemas de otimização não linear.

Apesar de trabalhar com populações de soluções, como os Algoritmos Genéticos, a Busca por Espalhamento possui estratégias bem definidas de exploração de subconjuntos dessas populações e usa estruturas de memória adaptativa para guiar essas estratégias ao longo da busca. Essas características aproximam a Busca por Espalhamento da Busca Tabu, criando uma ponte entre as buscas populacionais aleatórias e as buscas seqüenciais determinísticas.

Por ser um método relativamente novo, poucas aplicações são encontradas na literatura. FLEURENT *et al.* (1996) apresentam uma abordagem simples para a Busca por Espalhamento para a otimização de funções não lineares contínuas e irrestritas. Nessa abordagem, a Busca Tabu é utilizada para melhorar cada nova solução encontrada pela Busca por Espalhamento.

GLOVER *et al.* (1996) descrevem um aplicativo recentemente desenvolvido que integra simulação e otimização. O objetivo é otimizar o conjunto de dados de entrada de um simulador para evitar uma enumeração completa dos cenários possíveis. Nesse aplicativo, a fase de otimização é realizada pela Busca por Espalhamento, complementada por elementos de Busca Tabu. Os parâmetros podem ser variáveis contínuas ou discretas. Entretanto, a otimização é feita no espaço real e os resultados são arredondados quando se requer valores discretos.

CUNG *et al.* (1997) apresentam uma abordagem baseada na Busca por Espalhamento para o problema de designação quadrática. Embora as variáveis de decisão sejam binárias o algoritmo proposto gera novas soluções usando combinações lineares de soluções conhecidas. As soluções obtidas passam por um processo de factibilização antes de serem aceitas como parte da população.

O método de Busca por Espalhamento original, como proposto em GLOVER (1994a) é descrito na seção 4.1. Nas demais seções são discutidas as adaptações propostas para a

solução do PST. O objetivo é comparar essa nova meta-heurística aos Algoritmos Genéticos apresentados nos capítulos 2 e 3, tanto do ponto de vista de desempenho como do ponto de vista de comportamento.

4.1 - INTRODUÇÃO

O método heurístico de Busca por Espalhamento consiste, basicamente, da geração de um conjunto disperso de soluções tentativas para o problema em estudo a partir de um conjunto de soluções de referência previamente conhecidas. Isso é feito através de combinações sistemáticas das soluções selecionadas para compor o conjunto de referência.

A implementação iterativa desse método pode ser feita utilizando-se informações do histórico de progressão da busca para a escolha de soluções de referência adequadas. Podem ser utilizadas estruturas de memória tabu sobrepostas a esse processo, introduzindo restrições tabu e critérios de aspiração para determinar a composição do conjunto de soluções de referência a cada iteração. Estratégias de diversificação e intensificação também podem ser introduzidas, direcionando a criação de soluções tentativas para novas regiões do espaço de busca ou restringindo essa criação a regiões promissoras, respectivamente.

Algumas características gerais da Busca por Espalhamento possuem uma correspondência direta com características dos Algoritmos Genéticos. Uma delas é a necessidade de se definir uma representação (ou mapeamento) das soluções do problema para que o método possa manipulá-las e combiná-las eficientemente. Além disso, ambas meta-heurísticas utilizam parâmetros externos, que devem ser determinados pelo usuário. Entretanto, ao contrário dos Algoritmos Genéticos, a Busca por Espalhamento realiza poucas escolhas aleatórias e, portanto, os parâmetros externos são usados de forma determinística.

Outra característica específica da Busca por Espalhamento é a divisão da população em subconjuntos (ou subpopulações) com finalidades estratégicas específicas. A finalidade de cada conjunto determina seu processo de construção, composto por critérios de aceitação para as soluções geradas no processo de busca. A descrição da Busca por Espalhamento inicia-se pela apresentação dos conjuntos utilizados e suas regras de construção.

4.1.1 - Conjuntos de soluções

O primeiro conjunto a ser definido é P_t, que representa a população total de p soluções consideradas pelo método na iteração t. A partir de P_t são definidos os demais conjuntos:

P^*_t → conjunto que armazena as p^* melhores soluções de P_t;

H_t → conjunto histórico responsável pelo armazenamento das soluções de elite, definidas como as h melhores soluções encontradas pela busca até a iteração t;

H^*_t → conjunto que armazena as h^* melhores soluções de H_t;

T_t → conjunto de soluções tabu. São as soluções de H_t proibidas de compor H^*_t na iteração atual;

R_t → conjunto de soluções de referência na iteração t, definido como $H^*_t \cup P^*_t$. Note que, pela forma como foram definidos os conjuntos P^*_t e H_t, existe a possibilidade de sobreposição entre H^*_t e P^*_t, principalmente nas iterações iniciais. Quando essa sobreposição ocorre, o número de soluções em R_t pode se tornar inconvenientemente pequeno e, portanto, define-se um tamanho mínimo para R_t de forma que esse conjunto tenha sempre um cardinalidade adequada. Dessa forma, quando $|H^*_t \cup P^*_t|$ for menor que um parâmetro r^* mínimo, soluções adicionais de P_t são adicionadas a R_t.

Além desses conjuntos, dois outros são definidos. Entretanto, a regra de escolha de suas soluções é um pouco mais complexa que a dos conjuntos acima, pois baseia-se na localização das soluções no espaço de busca e não em seus custos. Esses conjuntos são chamados Subconjuntos Diversos pois são construídos a partir de conjuntos maiores através da escolha das soluções mais distantes entre si. A regra geral para sua construção é dada a seguir.

Regra Geral para a Construção de Subconjuntos Diversos: "Dado um conjunto X, constrói-se $D(X)$ como um subconjunto diverso de X da seguinte forma. O primeiro elemento de X selecionado para entrar em $D(X)$ é aquele que apresenta a maior distância ao centro de gravidade de X. O segundo elemento de $D(X)$ é selecionado como o elemento de X que apresente a maior distância ao primeiro elemento. O terceiro elemento, e os demais, são escolhidos de forma a maximizarem a menor distância aos elementos que já tenham sido selecionados para compor $D(X)$".

Note que a cardinalidade de $D(X)$ deve ser mantida pequena para que a construção seja feita de forma eficiente. A partir dessa regra, definem-se os dois últimos conjuntos de soluções utilizados pela Busca por Espalhamento.

$D(P^*_t)$ e $D(P_t \setminus P^*_t)$ → subconjuntos diversos contendo soluções dispersas de P^*_t e $P_t \setminus P^*_t$, respectivamente, que devem ser combinados com as soluções de R_t.[1]

4.1.2 - Composição Inicial dos Conjuntos de Soluções

Na primeira iteração ($t = 0$) deve-se escolher uma população inicial de soluções P_0 para construir os demais conjuntos definidos. Essa população inicial pode ser gerada aleatoriamente ou pode ser construída como um subconjunto diverso de um conjunto maior de soluções geradas aleatoriamente. Tendo sido gerada essa população inicial, pode-se construir os demais conjuntos:

P^*_0 = {p^* melhores soluções de P_0};

[1] Notação: $P_t \setminus P^*_t$ é o conjunto formado por todos os elementos pertencentes a P_t com exceção dos elementos pertencentes a P^*_t.

H_0 = {h melhores soluções de P_0};

$T_0 = \emptyset$;

H^*_0 = {h* melhores soluções de H_0}. Nessa primeira iteração, $H^*_0 \subseteq P^*_0$;

R_0 = {r* melhores soluções de P_0}. Note que, nessa iteração, a sobreposição entre H^*_0 e P^*_0 é total e, portanto, para que R_0 tenha no mínimo r* elementos, devem ser adicionadas a ele r* - $|R_0|$ soluções de $P_0 \setminus P^*_0$.

$D(P^*_0)$ e $D(P_0 \setminus P^*_0)$ → subconjuntos diversos de P^*_0 e $P_0 \setminus P^*_0$, respectivamente.

Nas figuras 4.1 a 4.3 são apresentadas três situações possíveis para a sobreposição dos conjuntos definidos. Em 4.1 mostra-se a sobreposição existente na primeira iteração, quando os conjuntos são criados. Na Figura 4.2 mostram-se sobreposições parciais entre os conjuntos e, finalmente, na Figura 4.3, mostra-se uma situação em que os conjuntos H_t e P_t são disjuntos. A freqüência em que cada uma dessas situações ocorre depende das estratégias de atualização dos conjuntos definidos, a cada iteração. Comparando-se essas três situações do ponto de vista da diversidade e do número de soluções exploradas, nota-se que a situação da Figura 4.3 é mais vantajosa e que sua predominância com relação às outras situações deve ser estimulada.

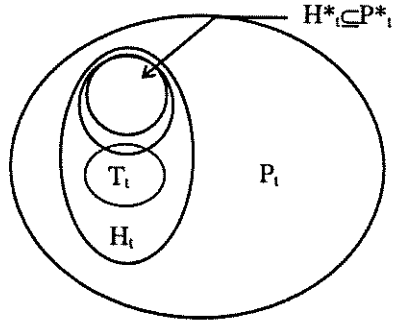

Figura 4.1 - Sobreposição completa entre os conjuntos H_t e P_t

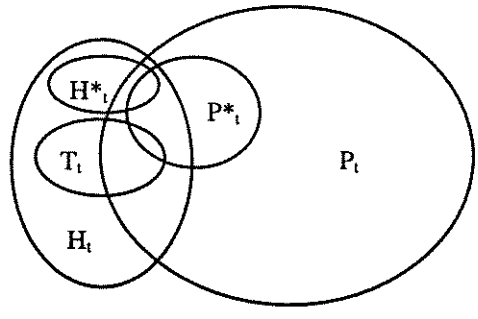

Figura 4.2 - Sobreposição parcial entre os conjuntos H_t e P_t

4.1.3 - Resumo de uma iteração

Basicamente, uma iteração da Busca por Espalhamento consiste de três estágios: seleção do conjunto de soluções de referência, geração de novas soluções tentativas e

atualização dos conjuntos de soluções. Novamente, nota-se uma grande semelhança entre as linhas gerais de uma iteração da Busca por Espalhamento e o nível estrutural dos Algoritmos Genéticos. Existem, entretanto, diferenças profundas entre os dois métodos quanto aos mecanismos previstos para a seleção e combinação de soluções.

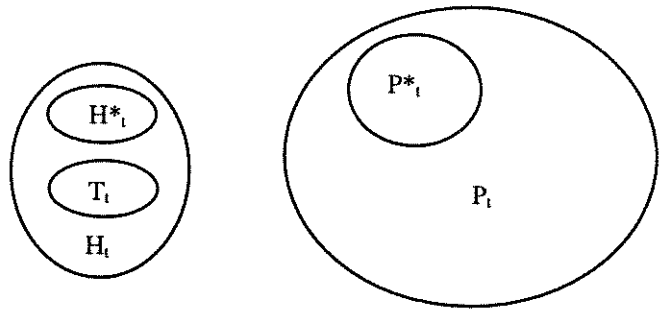

Figura 4.3 - Nenhuma sobreposição entre os conjuntos H_t e P_t

A seleção de soluções de referência pode ser feita automaticamente a partir da definição dos conjuntos utilizados. A cada iteração, deve-se selecionar os elementos não tabu de H_t para formar H^*_t, os elementos de P^*_t ,construir o conjunto R_t com o tamanho adequado e, finalmente, identificar os conjuntos $D(P^*_t)$ e $D(P_t \setminus P^*_t)$.

O próximo estágio consiste da geração de soluções tentativas. As combinações de soluções utilizadas para essa geração em (GLOVER, 1994a) são: (1) Gerar os centros de gravidade de P^*_t e de todos os subconjuntos de P^*_t com $p^* - 1$ elementos; (2) Combinar esses centros de gravidade com as soluções em R_t para criar linhas de busca (retas no $\mathbf{R}^n$) sobre as quais são geradas novas soluções tentativas; (3) Emparelhar e combinar entre si os elementos de P^*_t para gerar novas linhas de busca e novas soluções; e (4) Utilizar os subconjuntos diversos $D(P^*_t)$ e $D(P_t \setminus P^*_t)$ para criar linhas de busca adicionais emparelhando seus elementos com os de H^*_t e R_t, respectivamente. As combinações (1), (2) e (3) podem ser interpretadas como uma forma de intensificação na região do espaço de busca definida pelo conjunto de soluções de referência. Já a combinação (4) utiliza tanto soluções de elite como soluções escolhidas para formar subconjuntos diversos na geração de soluções tentativas. Essas soluções devem ter uma maior diversidade estrutural com relação às soluções de referência.

No terceiro estágio da iteração t, avaliam-se as soluções obtidas e atualizam-se os conjuntos de soluções. As soluções candidatas a compor P_{t+1} são identificadas, assim como aquelas que deverão compor H_{t+1}. Finalmente, T_t é atualizado utilizando-se as restrições tabu e os critérios de aspiração estabelecidos e pode-se passar para a próxima iteração.

4.1.4 - Geração de soluções tentativas

As formas previstas para a geração de soluções tentativas na Busca por Espalhamento são:

(1) (Centros de gravidade em P^*_t) As soluções em P^*_t são denotadas por S_k, $k = 1,...,p^*$. Gera-se o centro de gravidade de P^*_t como (Figura 4.4):

$$y_0 = \frac{1}{p^*} \sum_{k=1}^{p^*} S_k \qquad (4.1)$$

e os centros de gravidade de cada subconjunto de P^*_t com $p^* - 1$ elementos (Figura 4.5) como:

$$y_k = y_0 + \frac{(y_0 - S_k)}{p^* - 1}, k = 1,...,p^*. \qquad (4.2)$$

Esses centros de gravidade são as soluções tentativas iniciais.

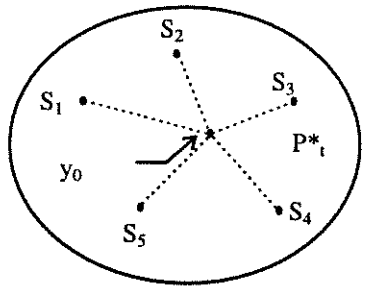

Figura 4.4 - Cálculo do Centro de Gravidade de P^*_t

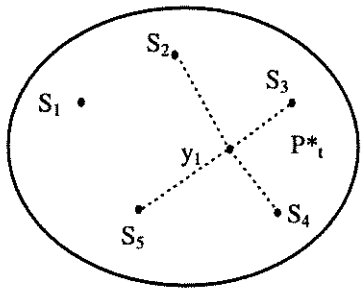

Figura 4.5 - Cálculo do Centro de gravidade dos subconjuntos de P^*_t com p^*-1 elementos

(2) (Combinação dos centros de gravidade com elementos de R_t) Sejam $S_{(p^*+1)},..,S_r$ as soluções de referência que estão em $R_t \setminus P^*_t$ e sejam os pares de soluções (S_k, y_k) para $k = 1,...,p^*$ e (S_k, y_0) para $k = p^*+1,...,r$. Considerando que a linha entre S_k e y_k (ou y_0) é dada pela representação $S_z(w) = S_k + w(y_k - S_k)$, onde w é um peso escalar, inicia-se a busca com as soluções $S_z(1/2)$, $S_z(1/3)$, $S_z(2/3)$, $S_z(-1/3)$ e $S_z(4/3)$ (Figura 4.6). O esforço computacional que deverá ser empreendido na obtenção e avaliação de soluções tentativas determina o número de valores de w utilizados nesse procedimento.

(3) (Combinação de elementos de P^*_t entre si) Aplica-se a regra dada em (2) para criar soluções tentativas a partir dos pares (S_x, S_y), $S_x \neq S_y$, com S_x e S_y pertencentes a

P^*_t (Figura 4 . 7).

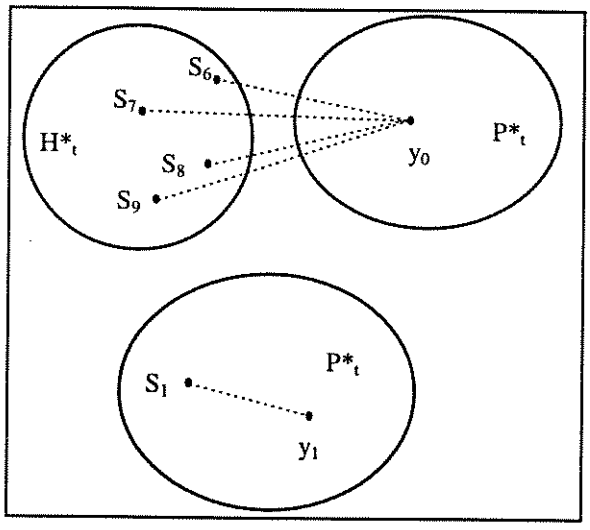

Figura 4 . 6 - Combinação das soluções em R_t com os centros de gravidade obtidos em (1)

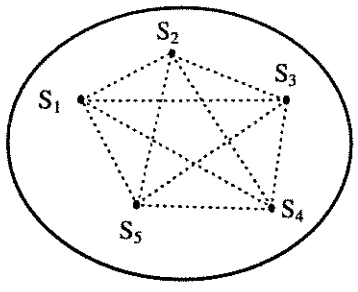

Figura 4 . 7 - Combinação dos elementos de P^*_t entre si

(4) (Combinação das soluções em R_t com subconjuntos diversos $D(P^*_t)$ e $D(P_t \setminus P^*_t)$) A regra de (2) é aplicada aos pares de soluções (S_x, S_y) obtidos pelo produto cruzado dos conjuntos $H^*_t \times D(P^*_t)$ e $R_t \times D(P_t \setminus P^*_t)$, gerando novas linhas de busca e soluções tentativas (Figura 4 . 8).

(5) (Componente de Intensificação) Durante a execução dos passos (2) a (4) identifica-se a solução tentativa S_z com melhor avaliação em cada linha de busca. Para cada uma dessas soluções, se sua avaliação for melhor que a avaliação média das soluções em P^*_t, intensifica-se a busca sobre S_z gerando novas soluções tentativas nos pontos médios dos segmentos que unem S_z aos seus vizinhos mais próximos já examinados. Se alguma das novas soluções criadas tem uma avaliação melhor que a de S_z, o processo é repetido.

4.1.5 - Restrições Tabu e Critérios de Aspiração

As restrições tabu são introduzidas para que haja a transferência de elementos do conjunto H_t para o conjunto T_t, impedindo que essas soluções possam ser selecionadas como soluções de referência por um determinado número de iterações. Essas restrições são impostas

a uma solução como conseqüência de sua seleção para o conjunto de referência na iteração atual (ou durante um número de iterações que inclui a atual).

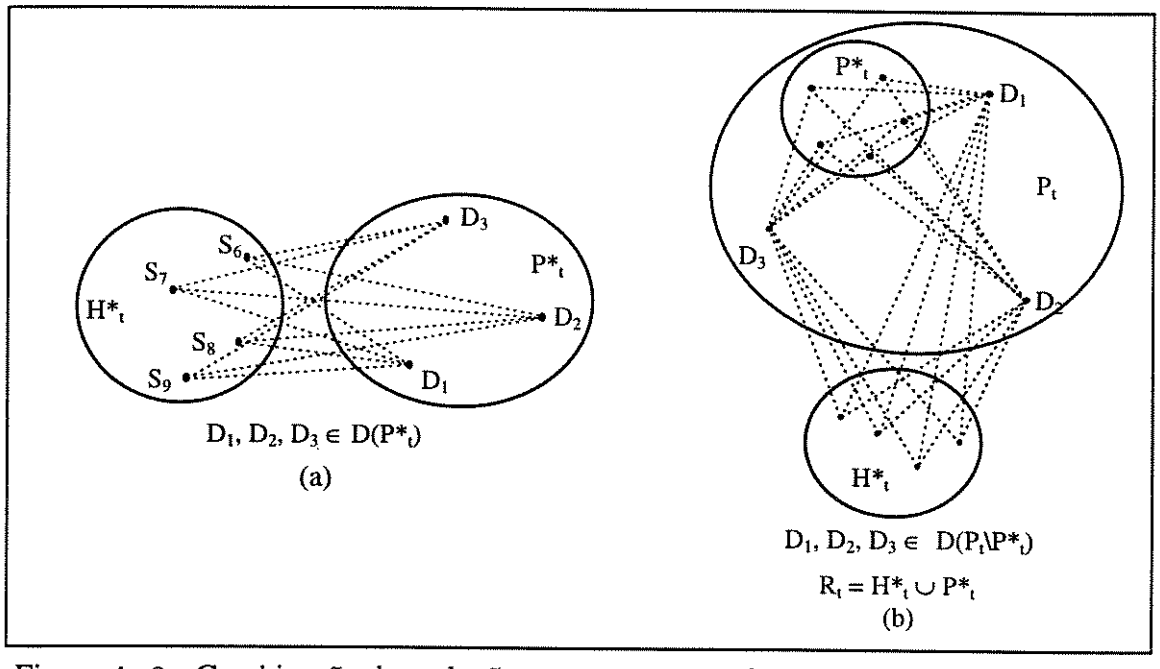

Figura 4 . 8 - Combinação das soluções em R_t com aquelas em $D(P^*_t)$ e $D(P_t \setminus P^*_t)$
(a) $(S_x, S_y) \in H^*_t \times D(P^*_t)$
(b) $(S_x, S_y) \in R_t \times D(P_t \setminus P^*_t)$

A utilização de soluções de H_t como soluções de referência pode ser feita de forma estratégica, aumentando a eficiência da busca. Uma estrutura de memória tabu de curto prazo pode ser utilizada para forçar, a princípio, que uma solução S_x recém introduzida em H^*_t tenha um certo nível de exposição. Ou seja, S_x é forçada a permanecer em H^*_t por um número pré-determinado de iterações, ao final das quais é transferida imediatamente para T_t, a não ser que satisfaça um critério de aspiração. O critério de aspiração definido é o seguinte: se a solução S_x participou da geração de uma solução que foi incorporada ao conjunto P^*_t, é permitido que ela permaneça em H^*_t por um novo período de exposição. Esse critério somente pode ser aplicado um número limitado de vezes a cada solução que o satisfaça para que seja estimulada a diversificação do conjunto H^*_t.

Outra questão relacionada à memória de curto prazo é a da duração tabu, que pode ser determinada de forma progressiva. Ou seja, na primeira vez que uma solução é transferida para T_t, ela permanece nesse conjunto pelo mesmo número de iterações que fica exposta em H^*_t. Essa restrição pode ser progressivamente aumentada incrementando a duração tabu de uma solução S_x a cada vez que ela é transferida para T_t. A duração tabu pode ser diminuída para seu valor inicial quando S_x contribui para a geração de uma solução tentativa que se qualifica para entrar em H_t. Além disso, deve ser estabelecido um valor máximo para a

duração tabu de qualquer solução para que ela não permaneça em T_t indefinidamente.

Para atingir um certo grau de diversificação, pode-se usar estruturas de memória tabu de longo prazo para assegurar que todos os elementos de H_t tenham a oportunidade de participar de H^*_t. Isso pode ser feito levando-se em conta quantas vezes cada uma das soluções de H_t já participou de H^*_t (freqüência) e por quanto tempo (residência). A manipulação adequada dessas informações pode melhorar o desempenho da Busca por Espalhamento.

4.1.6 - Algoritmo

A Busca por Espalhamento pode ser colocada na forma de algoritmo, como segue.

- **Algoritmo Busca_por_Espalhamento**

Parâmetros: h, h*, p, p*;
Passo 0: Crie o conjunto P_0;
Crie o conjunto H_0 = {h melhores elementos encontrados na busca até o momento}
Faça t = 0; T = ∅;
Passo 1: Crie os conjuntos:
H^*_t = {h* melhores elementos de H}
P^*_t = {p* melhores elementos de P}
$R_t = H^*_t \cup P^*_t$; se $|R_t| < r^*$ acrescente os $(r^* - |R_t|)$ melhores elementos de $P_t \backslash P^*_t$ a R_t.
Passo 2: Gere as soluções y_0 (Centro de gravidade de P^*_t) e y_k, k = 1,..,p* (Centros de gravidade dos subconjuntos de P^*_t com p* - 1 elementos.
Passo 3: Sejam $S_{(p^*+1)},\ldots,S_r$ pertencentes a R_t mas que não pertencem a P^*_t. Emparelhe as soluções (S_x,y) tais que:

$$(S_x,y) = \begin{cases}(S_k,y_k), k = 1,..,p^* \\ (S_k,y_0), k = p^*+1,..,r\end{cases}$$

Crie $S_z(w) = S_x + w.(y-S_x)$, para todo (S_x,y) com os valores w = 1/2, 1/3, 2/3, -1/3 e 4/3.
Se valor(S_z) melhor que valor_médio(P^*_t) intensifique a busca sobre a solução S_z.
Passo 4: Emparelhe as soluções (S_k,S_j), j,k = 1,..,p*, j ≠ k.
Crie $S_z(w) = S_k + w.(S_j-S_k)$, para todo (S_k,S_j) com os valores w = 1/2, 1/3, 2/3, -1/3 e 4/3.
Se valor(S_z) melhor que valor_médio(P^*_t) intensifique a busca sobre a solução S_z.
Passo 5: Identifique os subconjuntos diversos $D(P^*_t)$ e $D(P_t \backslash P^*_t)$.
Crie os pares (S_x,S_y) tais que $\begin{cases}S_x \in H^*_t, S_y \in D(P^*_t) \\ S_x \in R_t, S_y \in D(P_t \setminus P^*_t)\end{cases}$
Crie $S_z(w) = S_x + w.(S_y-S_x)$, para todo (S_x,S_y) com os valores w = 1/2, 1/3, 2/3, -1/3 e 4/3.
Se valor(S_z) melhor que valor_médio(P^*_t) intensifique a busca sobre a solução S_z.
Passo 6: Altere os conjuntos P_t, H_t e T_t convenientemente;
Faça t = t + 1;
Se o critério de parada for satisfeito, Pare.
Caso contrário, volte para o passo 1

4.1.7 - Refinamentos possíveis

O método descrito acima pode ser refinado através de algumas modificações que podem ser consideradas isoladamente ou em conjunto.

(1) Os centros de gravidade do Passo 2 podem ser ponderados com referência a avaliações dos elementos em P^*_t.

(2) Podem ser criados caminhos paralelos de busca, cada uma com seus próprios conjuntos P_t e H_t. Periodicamente, os elementos pertencentes à união dos diferentes conjuntos H_t podem ser realocados entre esses conjuntos.

(3) Os valores de w usados para gerar soluções tentativas foram determinados anteriormente como "default" e assume-se que o comportamento da função de avaliação na região examinada é desconhecido ou seu cálculo é muito trabalhoso. Se um conhecimento prévio desse comportamento existe ou pode ser inferido pela busca, ele deve ser usado para determinar outros valores de w.

(4) Durante uma linha de busca intensificada, uma estratégia possível é construir iterativamente uma função quadrática ou cúbica que aproxime a avaliação sucessiva de várias soluções tentativas e calcular a localização aproximada de um mínimo hipotético. Em conjunto com essa abordagem, podem ser estabelecidas linhas de intensificação para soluções que se encontrem na direção de melhoria apontada pelas avaliações sucessivas.

(5) O método pode ser melhorado se for usado reconhecimento de duplicações aproximadas. Quando uma solução é considerada para inclusão em H^*_t ou P^*_t e sua avaliação for muito parecida com a de algum elemento que já pertença a esses conjuntos então a distância entre as duas soluções deve ser examinada. Se elas estiverem muito próximas, a solução que tiver menor avaliação é descartada. (O grau de separação que caracteriza a distância mínima para eliminação de uma das soluções depende do contexto do problema. A escolha do grau de separação pode ser um procedimento heurístico para prevenir que as soluções se agrupem muito).

(6) Quando uma nova solução S_x é adicionada a H_t ela pode ser emparelhada com as demais soluções desse conjunto para assegurar que todas as linhas representativas de busca geradas por membros de H_t sejam exploradas.

(7) Uma forma simples de diversificação pode ser alcançada pela restrição no número de soluções tentativas de uma linha de busca que se permite incluir em P^*_t ou H_t. Por exemplo, uma regra pode ser imposta para prevenir que mais de duas soluções da mesma linha de busca entre em P^*_t ou H_t. Essa regra também pode exigir que as soluções estejam separadas por uma distância mínima, como em (5).

(8) Finalmente, a definição de subconjunto diverso pode ser generalizada. Dado um conjunto pai X e uma solução u (não necessariamente em X), um subconjunto diverso D(X,u) de X pode ser caracterizado como um subconjunto gerado pelas seguintes regras: (a) o primeiro elemento é selecionado em X como sendo o que maximiza sua distância de u, (b) cada elemento subsequente é selecionado de X para maximizar sua distância mínima de todos os elementos escolhidos anteriormente (excluindo-se u), (c) empates podem ser decididos pela maximização da distância mínima às últimas k-1 soluções escolhidas (k-2, k-3,...,

até que se decida o empate). Para diminuir o esforço computacional, pode-se escolher um valor pequeno v e escolher os novos elementos de D(X,u) calculando sua distância aos últimos v elementos adicionados a esse conjunto.

Ao longo da descrição do método de Busca por Espalhamento observa-se que ele foi desenvolvido especificamente para problemas em que as soluções são representadas por vetores no espaço n-dimensional. A maioria dos procedimentos apresentados em GLOVER (1994a) e que fazem parte do método são apenas sugestões do autor e não existem trabalhos na literatura em que todos esses procedimentos tenham sido testados e discutidos, como é o caso da Busca Tabu e mesmo dos Algoritmos Genéticos. Na seção 4.2 é apresentada a implementação da Busca por Espalhamento para o PST, onde foram feitas modificações e adaptações nos procedimentos descritos.

4.2 - IMPLEMENTAÇÃO DA BUSCA POR ESPALHAMENTO

A implementação de um método populacional para resolver o PST baseado na Busca por Espalhamento envolve algumas adaptações no esquema original, mas segue as linhas mestras do método descrito e seu objetivo principal é obter uma busca estratégica que considere tanto aspectos de qualidade como aspectos estruturais das soluções exploradas. A implementação computacional apresentada nessa seção será chamada deste ponto em diante de BE1.

O primeiro passo para a descrição de BE1 é a escolha de um mapeamento (ou representação, equivalentemente) adequado para as soluções do PST. Foi escolhida a mesma representação utilizada em GEN1 e GEN2, dado que essa representação mostrou-se adequada para a manipulação eficiente através dos operadores genéticos. As informações de posição e precedência de nós de tarefas e máquinas presentes nessa representação pode facilitar a escolha das regras a serem utilizadas para combinar soluções.

Dado o mapeamento a ser utilizado, deve-se definir uma métrica para o cálculo de distância entre duas soluções que seja representativa das diferenças estruturais entre elas. Para esse fim foi escolhido o número de arcos distintos entre as soluções, pois cada arco representa uma relação de precedência entre duas tarefas, uma tarefa e uma máquina ou entre duas máquinas. O número de arcos distintos entre duas soluções pode variar entre 0 e (m+n-1), onde m é o número de máquinas e n é o número de tarefas. Note que a distância baseada no número de arcos distintos entre soluções leva em conta somente a informação de precedência contida na representação por caminho. Outras métricas poderiam ter sido definidas levando em consideração, por exemplo, a informação sobre a posição de cada nó no vetor solução.

4.2.1 - Conjuntos de Soluções

Nas figuras 4.1 a 4.3 foram apresentadas três situações possíveis para a sobreposição entre os conjuntos de soluções utilizados na Busca por Espalhamento. Foi observado que a Figura 4.3 apresenta a situação mais vantajosa do ponto de vista do número de soluções exploradas e da diversidade dessas soluções. Em BE1 essa situação é mantida desde a criação dos conjuntos na iteração $t = 0$ e, para tanto, é necessário redefini-los. A seguir são dadas as novas definições utilizadas em BE1.

$P_t \rightarrow$ subpopulação de soluções tentativas aceitas pela sua diversidade estrutural. São soluções que representam um subconjunto diverso das soluções tentativas geradas na iteração $t-1$;

$P^*_t \rightarrow p^*$ melhores soluções de P_t;

$H_t \rightarrow$ subpopulação de elite. São as melhores soluções encontradas pela busca desde seu início até a iteração $t-1$;

$H^*_t \rightarrow h^*$ melhores soluções de H_t;

Não se permite que uma mesma solução seja designada a pertencer simultaneamente a H_t e P_t, numa mesma iteração e, portanto, esses conjuntos representam subpopulações disjuntas. Dessa forma, os conjuntos H^*_t e P^*_t não se sobrepõem em nenhuma circunstância e, portanto, a definição de R_t fica simplificada.

$R_t \rightarrow$ conjunto de soluções de referência dado por $H^*_t \cup P^*_t$.

Entre duas iterações consecutivas, uma transferência de soluções de H_t para P_t está prevista, no momento em que esses conjuntos são atualizados. Essa transferência de soluções é discutida mais adiante, juntamente com os mecanismos de atualização dos conjuntos.

Nas combinações descritas na seção 4.1.4, observa-se que as soluções pertencentes a P_t mas que não pertencem a P^*_t ou $D(P_t \setminus P^*_t)$ não são utilizadas em nenhuma das regras para geração de soluções tentativas. Como P_t foi redefinido nesta seção como um subconjunto diverso das soluções geradas nas iterações anteriores, utiliza-se $P_t \setminus P^*_t$ nas combinações de soluções e não se define $D(P_t \setminus P^*_t)$.

Para que BE1 possa ser aplicada a problemas de grande porte é preciso garantir que a memória necessária ao armazenamento de soluções mantenha-se dentro de limites razoáveis e uma forma de controlar o uso de memória é manter os conjuntos de soluções com cardinalidades pequenas. Sendo assim, para não comprometer a diversidade da busca, proíbe-se a ocorrência de repetições de uma mesma solução nos conjuntos P_t e H_t. A forma como isso é feito é descrita a seguir.

Controle de Repetição de Soluções

Uma opção para se comparar soluções e evitar que elas se repitam numa mesma iteração é utilizar a métrica definida para o cálculo de distâncias. Assim, a cada vez que uma solução for candidata a entrar em P_t ou H_t, calcula-se a distância dessa solução a cada uma

das outras soluções previamente aceitas e inseridas em um desses dois conjuntos. A solução candidata é aceita se todas as distâncias calculadas forem não nulas.

Apesar das cardinalidades de P_t e H_t serem mantidas em valores pequenos essa abordagem pode tornar a busca muito lenta, principalmente se o número de soluções candidatas avaliadas for grande. Seria interessante que cada solução tivesse um valor associado a ela que traduzisse da melhor forma possível suas características estruturais. Esse valor seria calculado somente uma vez para cada solução de P_t e H_t e quando uma solução candidata tivesse que ser avaliada, seu valor característico seria calculado e comparado com os valores das soluções desses conjuntos. A complexidade computacional dessa última abordagem é bastante reduzida quando comparada à primeira.

WOODRUFF e ZEMEL (1993) apresentam um esquema de transformação de uma solução S_x representada por uma permutação de números inteiros em um único valor que pode ser chamado de característico(S_x). A utilização desse esquema para comparar soluções do PST começa pela atribuição a cada nó (de máquina ou de tarefa) de um valor $\psi(i)$ gerado aleatoriamente no intervalo [1, 131072]. Toma-se o cuidado de manter $\psi(i) = \psi(j)$ se i e j forem nós de máquina pois na definição do PST considera-se que as máquinas são idênticas. Ao longo da busca, cada solução S_x tem seu valor característico calculado como:

$$\text{característico}(S_x) = \sum_{i=0}^{m+n-2} \Psi([i]) \cdot \Psi([i+1]) + \Psi([m+n-1]) \cdot \Psi([0]) \quad (4.3)$$

onde [i] = índice do nó que ocupa a posição i da permutação.

Dessa forma, se alguma solução em P_t ou H_t possui o mesmo valor característico e o mesmo custo de S_x, essa solução é rejeitada pois sua inclusão em um desses conjuntos duplicaria uma das soluções aceitas anteriormente. O valor do custo é utilizado em conjunto com o valor característico na comparação entre duas soluções pois, apesar de Woodruff e Zemel terem sugerido o intervalo [1,131072] de forma a minimizar a probabilidade de colisão (ocorrência do mesmo valor característico para soluções distintas), essa probabilidade não é nula.

Tendo sido determinada a forma de se evitar repetições de soluções nos conjuntos P_t e H_t , o próximo aspecto da implementação da Busca por Espalhamento a ser considerado é a construção de subconjuntos diversos.

Construção de Subconjuntos Diversos

A regra de construção estabelecida na seção 4.1.1 implica uma grande quantidade de cálculos de distância entre pares de soluções. Dado o conjunto X, seria necessário calcular, de acordo com alguma medida estabelecida para o problema, seu centro de gravidade para determinar o primeiro elemento de D(X). Para cada novo elemento a ser adicionado a esse conjunto todas as distâncias entre os elementos de X\D(X) e os elementos de D(X) deveriam

ser calculadas. Esse processo é ineficiente do ponto de vista computacional pois exige um grande número de cálculos de distância entre pares de soluções. Neste trabalho, a regra de construção de subconjuntos diversos é alterada com o intuito de diminuir o número de tais cálculos, mantendo-se o objetivo de maximizar a menor distância entre as soluções em $D(X)$.

O esquema proposto é iniciado pela escolha arbitrária de $|D(X)|$ soluções de X como uma composição inicial para o subconjunto diverso. As soluções de $D(X)$ são retiradas de X e todas as soluções S_t remanescentes nesse conjunto tornam-se candidatas a entrar em $D(X)$. A avaliação da inserção de cada uma dessas soluções candidatas no subconjunto diverso é feita medindo-se a distância mínima atual entre as soluções desse conjunto e a nova distância mínima obtida com a substituição de cada uma das soluções de $D(X)$ por S_t. Para tanto, deve-se calcular, inicialmente, a matriz **M** de distâncias entre cada par de soluções de $D(X)$. Essa matriz é simétrica e pode-se armazenar somente os elementos M_{ij} com $i < j$ (ou $i > j$), correspondentes à porção da matriz abaixo (ou acima) da diagonal. Por conveniência, utiliza-se $|D(X)|=a$, $|X \setminus D(X)|=b$ e d_{ij} = distância entre as soluções S_i e S_j.

$$\mathbf{M} = \begin{vmatrix} - & & & & & \\ d_{21} & - & & & & \\ d_{31} & d_{32} & - & & & \\ \vdots & & & & & \\ d_{a-1,1} & d_{a-1,2} & \cdots & \cdots & d_{a-1,a-2} & - \\ d_{a1} & d_{a2} & \cdots & \cdots & \cdots & d_{a,a-1} \quad - \end{vmatrix}$$

A menor distância entre cada solução de $D(X)$ para as demais soluções desse conjunto é denotada por M_i e calculada como: $M_i = \min_{j<i}\{d_{ij}\}$ para $i > 1$. Cada M_i corresponde ao menor valor de distância encontrado na linha i da matriz **M**. O valor de M_1 é calculado como o menor valor da coluna 1 de **M**, pois não se armazena nenhum elemento da linha 1.

Dados os valores de M_i, $i = 1,\ldots,a$, calcula-se a menor distância entre duas soluções quaisquer de $D(X)$ como $M_{min} = \min_{i=1,\ldots,a}\{M_i\}$. Seja S_k a solução de $D(X)$ tal que $M_k = M_{min}$. No caso de haver mais de uma solução com $M_i = M_{min}$, escolhe-se S_k como aquela que apresentar maior custo. Na Figura 4.9 mostra-se um exemplo de um conjunto $D(X)$ composto por quatro soluções. A distância entre cada par de soluções encontra-se representada sobre a aresta entre elas.

Para esse exemplo, os valores mínimos de distância $M_1 = 5, M_2 = 5, M_3 = 4, M_4 = 8$ e $M_{min} = 4$ são calculados a partir da seguinte matriz. Verifica-se, também, que $S_k = S_3$.

$$\mathbf{M} = \begin{vmatrix} - & & & \\ 5 & - & & \\ 7 & 4 & - & \\ 15 & 12 & 8 & - \end{vmatrix}$$

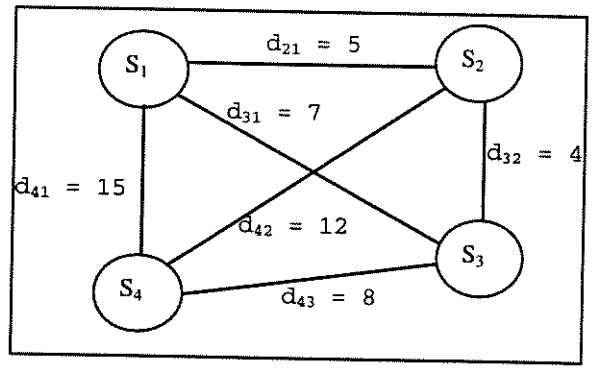

Figura 4.9 - Exemplo da configuração inicial de D(X)

Para determinar se uma dada solução $S_t \in X$, candidata a entrar em D(X), será ou não inserida nesse conjunto, calcula-se a distância entre S_t e todas as soluções de D(X). Então, podem ser calculados os valores $M'_i = \min_{j \neq i}\{d_{tj}\}$. Para cada solução $S_i \in D(X)$, M'_i representa a menor distância entre S_t e as soluções em $D(X) \setminus \{S_i\}$, ou seja, cada valor M'_i corresponde ao novo valor que M_i assumiria se S_i fosse substituída por S_t em D(X). Um exemplo do cálculo desses valores pode ser construído a partir do exemplo da Figura 4.9. Suponha que a solução candidata S_t seja tal que $d_{t1} = 6, d_{t2} = 10, d_{t3} = 8$ e $d_{t4} = 10$. Os valores de M'_i para $i = 1$ a 4 seriam: $M'_1 = 8, M'_2 = 6, M'_3 = 6$ e $M'_4 = 6$. Observe que M'_i pode assumir apenas dois valores. Com exceção da solução S_u para a qual $d_{tu} = \min_i\{d_{ti}\}$, o valor de M'_i é igual a d_{tu}. Note também que $M'_u = \max_i\{M'_i\}$.

Dados os valores de M_i e M'_i para $i = 1, \ldots, a$, avaliam-se três casos em que a inserção de S_t em D(X) pode ser vantajosa.

Caso 1: $M'_k > M_k = M_{min}$. Nesse caso, a substituição de S_k por S_t em D(X) faz com que o valor mínimo da distância entre duas soluções desse conjunto seja aumentado. Para o exemplo da Figura 4.9, $M'_3 > M_3$ e, portanto, a substituição de S_3 por S_t é vantajosa. Na Figura 4.10, mostram-se as novas distâncias calculadas em D(X) após essa substituição.

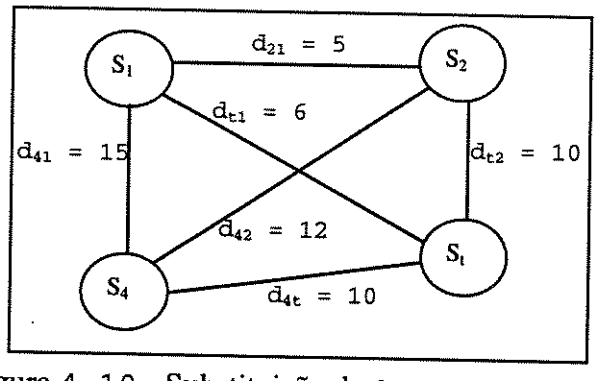

Figura 4.10 - Substituição de S_3 por S_t em D(X)

Os novos valores mínimos de distância são $M_1 = 5, M_2 = 5, M_3 = 6, M_4 = 10$ e

$M_{min} = 5$, calculados a partir da seguinte matriz.

$$\mathbf{M^1} = \begin{vmatrix} - & & & \\ 5 & - & & \\ 6 & 10 & - & \\ 15 & 12 & 10 & - \end{vmatrix}$$

Ainda considerando o Caso 1, poderia ter sido feita, alternativamente, a substituição de S_2 por S_t pois $d_{32} = d_{23} = 4 = M_{min}$. Essa substituição também acarretaria um aumento no valor de M_{min}. Na Figura 4.11, mostra-se o conjunto D(X) que seria obtido nesse caso.

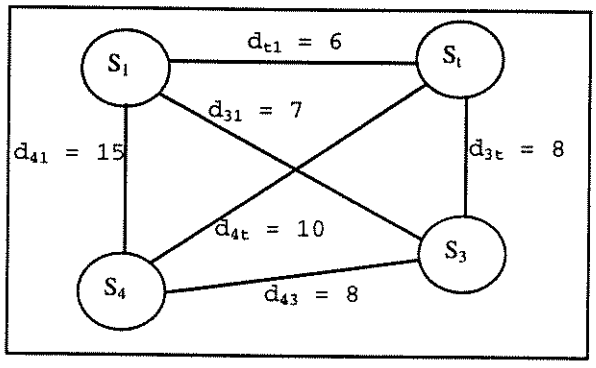

Figura 4.11 - Substituição de S_2 por S_t em D(X)

Com essa substituição, a nova matriz $\mathbf{M^2}$ seria:

$$\mathbf{M^2} = \begin{vmatrix} - & & & \\ 6 & - & & \\ 7 & 10 & - & \\ 15 & 10 & 8 & - \end{vmatrix}$$

e os valores mínimos para cada linha: $M_1 = 6, M_2 = 6, M_3 = 7, M_4 = 8$ e $M_{min} = 6$.

Apesar do valor final de M_{min} ser maior quando se substitui S_2 por S_t, e não S_3 por S_t, no exemplo, a avaliação das duas possibilidades de substituição para cada solução tentativa de X pode reduzir a eficiência do procedimento. A substituição S_2 por S_t foi mais vantajosa que a substituição de S_3 por S_t somente porque o segundo menor elemento de **M** é menor que o novo valor de M'_3 em $\mathbf{M^1}$ e, coincidentemente, seu valor corresponde a uma distância associada à solução S_2. Portanto, por uma questão de velocidade, decidiu-se realizar o teste de substituição do Caso 1 apenas para a solução S_k em cuja linha encontra-se o menor elemento de **M**. No exemplo, essa solução seria S_3.

Caso 2: Se o teste de substituição do Caso 1 falhar, ou seja, se $M'_k \leq M_k$, pode-se tentar utilizar a solução S_t para melhorar o valor mínimo da linha u da matriz **M**, aproveitando o fato de que M'_u é o único valor de M'_i, $i = 1, \ldots, a$, maior que M'_k (se $u \neq k$). Nesse

caso, se $M'_u > M_u$, substitui-se S_u por S_t.

Caso 3: Se os dois testes anteriores falharem, uma última tentativa de inserir S_t em $D(X)$ é realizada. Este último teste consiste em se verificar se é possível manter o valor atual de M_{min}, melhorando o valor médio do atraso das soluções de $D(X)$. Para isso, constrói-se um subconjunto $D'(X)$ de $D(X)$ composto pelas soluções para as quais $M'_i \geq M_i$ e $T(S_i) > T(S_t)$. Se esse subconjunto não for vazio, escolhe-se uma solução S_v tal que $T(S_v) = \max_{S_i \in D'(X)} \{T(S_i)\}$ e substitui-se S_v por S_t em $D(X)$.

Se, após a análise desses três casos, concluir-se que a entrada de S_t em $D(X)$ não é vantajosa, descarta-se S_t e passa-se à analise da próxima solução candidata.

Para que a regra de construção de subconjuntos diversos fique mais clara, ela é colocada na forma de algoritmo, a seguir. Os dados de entrada desse procedimento são: o conjunto $D(X)$ contendo a soluções, a solução candidata S_t e a matriz de distâncias **M**.

- **Procedimento Subconjunto_Diverso (D(X), S_t, M)**

Passo 1: Calcule $M_i = \min_{\substack{j \in D(X) \\ j<i}} \{d_{ij}\}, i = 2, \ldots, a$

$M_1 = \min_{\substack{S_i \in D(X) \\ i \neq 1}} \{d_{i1}\}$

$M'_i = \min_{\substack{S_j \in DX) \\ j \neq i}} \{d_{tj}\}, \forall S_i \in D(X)$;

$M_{min} = \min_{i \in D(X)} \{M_i\}$;

Passo 2: Encontre os índices k e u tais que $M_k = M_{min}$ e $M'_u = \max_{i \in D(X)} \{d_{ti}\}$

Passo 3: Se $M'_k > M_{min}$, substitua S_k por S_t e vá para o passo 8;

Passo 4: Se $M'_u > M_u$, substitua S_u por S_t e vá para o passo 8;

Passo 5: Construa o subconjunto D'(X) contendo as soluções S_i de D(X) tais que $M'_i \geq M_i$ e $T(S_i) > T(S_t)$;

Passo 6: Se $D'(X) \neq \emptyset$:

Escolha $S_v \in D'(X)$ tal que $T(S_v) = \max_{S_i \in D'(X)} \{T(S_i)\}$;

Substitua S_v por St e vá para o passo 8;

Passo 7: Descarte a solução S_t e pare.

Passo 8: Atualize a matriz **M** e pare.

Tendo sido definidas as regras de construção de todos os conjuntos utilizados por BE1, passa-se agora à descrição da composição inicial desses conjuntos na iteração $t = 0$.

4.2.2 - Composição Inicial dos Conjuntos de Soluções

Partindo-se das definições dadas em 4.2.1, devem ser criados os conjuntos H_0 e P_0 para, a partir deles, construir os demais conjuntos. Para tanto, foi escolhida a abordagem sugerida por GLOVER (1994a) em que as soluções de P_0 e H_0 são selecionadas a partir de um conjunto X gerado aleatoriamente. Gera-se, portanto, um conjunto X de soluções, ordena-se esse conjunto por valores não decrescentes de custo e cria-se H_0 selecionando-se as h melhores soluções de X. Das soluções restantes, seleciona-se um subconjunto diverso para compor P_0. Nota-se que, colocado dessa forma, o procedimento requer muita memória pois,

para garantir a geração aleatória de soluções representativas de todo o espaço de busca, o conjunto X deve ter uma cardinalidade alta. A forma encontrada para não ter que armazenar todo o conjunto X de uma vez é usar um procedimento iterativo de geração aleatória de soluções e construção dos conjuntos H_0 e P_0. Esse procedimento é descrito a seguir, na forma de algoritmo. Os parâmetros de entrada são os tamanhos dos conjuntos H_0 e P_0 (h e p, respectivamente) e o número de soluções que devem ser geradas aleatoriamente (q_{max} > h+p).

- **Procedimento Cria_População_Inicial (h, p, q_{max})**

Variáveis: q = contador do número de soluções geradas aleatoriamente;
S_{temp} = solução temporária utilizada em substituições;
d_{ij} = distância entre o par de soluções S_i e S_j.

Passo 1: q = 1;

Passo 2: Gere aleatoriamente uma solução S_q e calcule $T(S_q)$ e característico(S_q);

Passo 3: Se $T(S_q) = T(S_i)$ e característico(S_q) = característico (S_i) para alguma solução $S_i \in H_0$, volte para o passo 2;
Caso contrário:
insira S_q em H_0
faça q = q + 1;
Se q > h, vá para o passo 4;
Caso contrário, vá para o passo 2.

Passo 4: Ordene o conjunto H_0 por valores não decrescentes de custo.
Sejam $H_0(1)$, $H_0(2)$,..., $H_0(h)$ as soluções de H_0 tais que $T(H_0(1)) \leq T(H_0(2)) \leq ... \leq T(H_0(h))$.

Passo 5: Se q = q_{max}, pare: os conjuntos H_0 e P_0 estão completos.

Passo 6: Gere aleatoriamente uma solução S_q e calcule $T(S_q)$ e característico(S_q);

Passo 7: Se $T(S_q) = T(S_i)$ e característico(S_q) = característico(S_i) para alguma solução $S_i \in H_0$ ou P_0, volte para o passo 6;
Caso contrário, vá para o passo 8.

(Tentativa de inserção de S_q em H_0)

Passo 8: Se $T(S_q) < T(H_0(h))$, faça:
$S_{temp} = S_q$; $S_q = H_0(h)$; $H_0(h) = S_{temp}$;
Reordene H_0;
Vá para o passo 9;

(Tentativa de inserção de S_q em P_0)

Passo 9: Se P_0 já possui p soluções, vá para o passo 11
Caso contrário
insira S_q em P_0;
faça q = q + 1;
se q = h + p, vá para o passo 10;
caso contrário, vá para o passo 5;

(Cálculo da matriz de distâncias **M**)

Passo 10: Calcule d_{ij}, para $\forall$ S_i e $S_j \in P_0$, armazenando esses valores na matriz **M**;
Vá para o passo 5;

Passo 11: Subconjuntos_Diversos(P_0, S_q, **M**);
Faça q = q + 1 e vá para o passo 5;

Os passos de 1 a 3 correspondem ao preenchimento de H_0 com as primeiras h soluções distintas geradas aleatoriamente. No passo 4, o conjunto H_0 é ordenado por valores não decrescentes de custo. H_0 é mantido ordenado durante todo o processo de construção para

que a avaliação da entrada de S_q nesse conjunto possa ser feita comparando $T(S_q)$ apenas com o atraso da pior solução de H_0. A partir do momento em que a h-ésima solução é inserida em H_0, cada nova solução S_q é testada, em primeiro lugar, pelo seu custo. Se $T(S_q) < T(H_0(h))$, S_q é inserida em H_0 e tenta-se inserir $H_0(h)$ em P_0. Caso contrário, tenta-se inserir S_q diretamente em P_0 (Passo 8). Em ambos os casos, se P_0 ainda possui menos que p soluções, a nova solução é simplesmente adicionada a ele (Passo 9). No momento em que a p-ésima solução é inserida em P_0, calcula-se a matriz de distâncias **M** (Passo 10) pois, a partir dessa iteração, a inserção de novas soluções nesse conjunto deve ser feita pela regra de construção de subconjuntos diversos (Passo 11). O critério de parada previsto baseia-se no número máximo de soluções geradas aleatoriamente (Passo 5).

A construção dos conjuntos P^*_0, H^*_0 e R_0 segue a definição dada na seção anterior e pode-se, então, iniciar o processo iterativo de geração de novas soluções. BE1 possui, em cada iteração, os mesmos três estágios descritos em 4.1.3: seleção do conjunto de referência, geração de novas soluções tentativas e atualização dos conjuntos. O primeiro estágio corresponde à identificação do conjunto $R_t = P^*_t \cup H^*_t$, como no procedimento de Busca por Espalhamento original. Entretanto, a geração de soluções tentativas não pode ser feita da maneira descrita em 4.1.3 pois a representação através de permutações impede o cálculo de combinações lineares. Também não se pode mais falar em centro de gravidade de um conjunto de soluções. No R^n, o centro de gravidade de um conjunto A é uma solução eqüidistante de todas as soluções de A, obtida pela combinação linear dessas soluções. No espaço de soluções do problema PST, além do cálculo de combinações lineares levar a soluções infactíveis, a noção de distância depende da métrica escolhida, ou seja, das características estruturais selecionadas para diferenciar uma permutação das demais. É preciso, então, definir um tipo de combinação de soluções que considere as características estruturais relacionadas à métrica escolhida para o cálculo de distâncias, fazendo com que as soluções com melhores características possuam maior influência no resultado da combinação. Esse resultado poderia ser chamado de centro de influência, em analogia ao centro de gravidade. Em GLOVER (1994a), são sugeridas duas formas de combinar soluções representadas por permutações de inteiros, apresentadas nas seções 4.2.3 e 4.2.4.

4.2.3 - Combinações Estruturadas Ponderadas

As combinações estruturadas são definidas de forma a tentar preservar as condições de factibilidade associadas à representação específica das soluções de cada problema. Uma dada regra de combinação de soluções somente é considerada uma combinação estruturada ponderada se possuir as seguintes propriedades.

P1: (Representação) Cada vetor deve representar um conjunto de votos (decisões) associados a características particulares da solução que ele representa. Por exemplo, em um circuito de caixeiro viajante, o vetor solução representa em cada uma de suas posições, um

voto que corresponde a uma cidade. O conjunto de votos do vetor diz em que ordem as cidades são visitadas na solução que ele representa. No caso do PST, o vetor solução representa votos consecutivos associados à tarefa ou máquina que deve ser escolhida para ocupar cada posição do vetor.

P2: (Interpretação - Solução tentativa) O conjunto de votos especificado por um vetor traduz-se em uma solução tentativa para o problema de interesse por um processo bem definido. No exemplo do caixeiro viajante, ou no do PST, o vetor já é a própria solução tentativa.

P3: (Atualização) Se uma decisão é tomada de acordo com os votos de um dado vetor, existe uma regra claramente definida para atualizar todos os vetores do problema residual de forma que as propriedades **P1** e **P2** continuem válidas. Como exemplo, suponha que se deseja construir um novo circuito de caixeiro viajante a partir de dois circuitos distintos. A cada vez que se escolhe a próxima cidade a ser visitada na nova solução, uma regra válida de atualização é a eliminação dessa cidade dos dois circuitos que estão sendo combinados. Esses circuitos parciais passam a representar novos conjuntos de votos para o problema parcial restante.

Essas propriedades podem ser usadas como base para o desenvolvimento de um procedimento construtivo que visa obter o centro de influência de um conjunto de soluções (vetores) de referência. A partir de um certo número de vetores, S_1, S_2,.., S_r, uma única solução é criada de acordo com os votos desses vetores, ponderados por pesos designados a eles e denotados por w_1, w_2,.., w_r. Ao final desse procedimento, o centro de influência das soluções S_k, $k = 1,..,r$, representa uma mescla factível das decisões votadas por essas soluções e ponderadas pelos seus pesos.

O processo construtivo pode ser feito de forma seqüencial se for possível transformar os pesos correspondentes a cada solução em avaliações específicas de cada decisão a ser tomada. Tal transformação tem como base a modificação progressiva do impacto dos votos de cada vetor, ao longo do processo construtivo. Isso faz com que, mesmo que um vetor tenha tido pouca influência em decisões passadas, sua influência sobre a decisão atual pode aumentar. Logicamente, esse aumento de influência é feito de forma estratégica com o objetivo de tornar a influência total de um vetor S_k, avaliada sobre todas as decisões, proporcional ao seu peso w_k.

Para implementar esse tipo de combinação, é preciso definir alguns pontos, listados a seguir:

(1) Seqüência de tomada de decisões: dado um conjunto de soluções de referência, representando votos para possíveis decisões no processo construtivo, é preciso escolher uma ordem em que esses votos são considerados e avaliados;

(2) Votos: dependendo da representação utilizada e do problema em estudo, cada vetor solução possui um conjunto de características estruturais que podem ser utilizados como votos

no processo construtivo. Deve-se escolher qual característica será usada como voto;

(3) Avaliação de decisões: a implementação de um processo construtivo seqüencial depende da definição de uma regra de avaliação das possíveis decisões a cada iteração;

(4) Atualização de vetores: é preciso escolher uma regra de atualização para as soluções, de acordo com **P3**, de forma que, a cada tomada de decisão, essas soluções continuem a representar conjuntos de votos válidos para o problema parcial restante;

(5) Pesos das soluções de referência: a determinação dos pesos de cada solução depende do problema a ser resolvido.

A seguir, cada um desses pontos é definido para as soluções do PST para que se possa resolver um exemplo de combinação estruturada ponderada.

- **Seqüência de tomada de decisões**

Para os vetores de permutação utilizados em BE1, permanece a restrição de que o primeiro nó de todas as soluções corresponda à máquina M_1 (nó 0). Dessa forma, o centro de influência de qualquer combinação estruturada já possui esse nó mesmo antes do processo de combinação ser iniciado. Uma seqüência natural para a tomada de decisões é o preenchimento das demais posições do vetor a ser construído, iniciando-se pela segunda posição e caminhando em direção à última. Cada iteração do procedimento construtivo corresponde, então, à escolha do nó que deverá ocupar a próxima posição ainda não preenchida no novo vetor.

- **Votos**

Os vetores de referência podem representar diferentes tipos de votos, e isso é uma decisão que depende do problema a ser resolvido. Por exemplo, pode-se considerar que os vetores representem votos do tipo 0-1 para relações de precedência entre nós. Ou seja, o vetor S_k vota 1 para "i precede j" se e somente se os nós i e j ocupam posições consecutivas nesse vetor. Caso contrário, o voto de S_k para essa relação de precedência é 0. Além disso, como a representação para o PST foi derivada do problema do Caixeiro Viajante, é preciso considerar que a permutação é cíclica, ou seja, ela deve obrigatoriamente iniciar e terminar no nó 0.

Dado que existem outras informações estruturais presentes nos vetores de solução, como as informações sobre posição absoluta dos nós, outros tipos de votos poderiam ter sido definidos.

- **Avaliação de decisões**

De acordo com GLOVER (1994a), é possível quantificar a influência de cada vetor S_k, ao longo do processo de construção de um novo vetor, através de pontuações atribuídas a esses vetores. Seja v(k,j) o voto do vetor S_k para que o próximo nó da nova solução seja o nó j (v(k,j) $\in$ {0,1}, $\forall$k,j). Para medir o grau de contribuição de cada vetor S_k para as possíveis decisões da iteração atual, calcula-se:

$$\texttt{pontuação(k)} = \max\{0,\ v(k, j) - \texttt{Media}_{\substack{h=1,..r \\ h\neq k}} v(h, j)\} \qquad (4.4)$$

Calcula-se a pontuação cumulativa de S_k para todas as decisões passadas:

$$\texttt{cumulativa(k)} = \sum_{\texttt{iteração}=1}^{\texttt{atual}} \texttt{pontuação(k)} \qquad (4.5)$$

e define-se a contribuição efetiva de S_k para o centro de influência como:

$$\texttt{contribuição(k)} = \frac{\texttt{cumulativa(k)}}{\sum_{h=1}^{r} \texttt{cumulativa(h)}}. \qquad (4.6)$$

Usando essa definição, a escolha de uma das decisões possíveis para o estágio atual pode ser feita comparando-se os valores de `contribuição(k)` e w_k. Portanto, para cada uma das decisões candidatas para o estágio atual, pode-se escolher aquela que minimiza

$$\texttt{desvio} = \sum_{h=1}^{r} |\texttt{contribuição(k)} - w_k|. \qquad (4.7)$$

Dessa forma, realiza-se a construção do centro de influência de um conjunto de soluções através de uma combinação ponderada, fazendo com que a diferença entre a contribuição efetiva de cada vetor de referência S_k e seu peso w_k seja a menor possível a cada iteração do processo construtivo.

- **Atualização dos vetores de referência**

A atualização dos vetores de referência pode ser feita como foi sugerido em **P3**. A cada decisão tomada, correspondente ao estabelecimento da relação de precedência "`i` precede `j`" na solução parcial, elimina-se o nó `i` de todas as soluções de referência.

A seguir é dado um exemplo de combinação estruturada ponderada utilizando as regras definidas acima. Os vetores de referência são duas soluções do problema de programação da produção de sete tarefas em uma única máquina. A função de avaliação é a soma do atraso de todas as tarefas com relação a suas datas de entrega. Os dados para o problema utilizado no exemplo são mostrados na Tabela 4.1. Não são considerados tempos de preparação de máquina entre o processamento de duas tarefas consecutivas para facilitar os cálculos, mas isso não prejudica a apresentação dos mecanismos da combinação ponderada.

Tabela 4.1 - Dados para o exemplo de combinações estruturadas

i	1	2	3	4	5	6	7
p_i	18	44	72	68	16	23	64
d_i	124	55	25	210	19	203	191

- **Exemplo de combinação estruturada ponderada**

Soluções de Referência:

S_1 : (0 1 2 3 4 5 6 7); $T(S_1) = 467$; $w_1 = 3/5$
S_2 : (0 4 3 2 5 7 1 6); $T(S_2) = 758$; $w_2 = 2/5$

Iteração 0: (Iniciando o processo)
Solução Parcial (SP): (0);

Iteração 1: Decisões possíveis: $j = 1 \rightarrow v(1,1) = 1$;
$j = 4 \rightarrow v(2,4) = 1$;

As decisões possíveis são definidas pelas relações de precedência existentes nos vetores de referência. A decisão $j = 1$ provém do fato do nó 1 seguir o nó 0 em S_1 e a outra decisão, $j = 4$, decorre do fato do nó 4 seguir o nó 0 em S_2.

Cálculo das contribuições de cada vetor para cada decisão:

- j = 1:
  pontuação(1) = 1; pontuação(2) = 0;
  cumulativa(1) = 1; cumulativa(2) = 0;
  contribuição(1) = 1; contribuição(2) = 0;
  desvio = |1 - 3/5|+|0 - 2/5| = 4/5;
- j = 4:
  pontuação(1) = 0; pontuação(2) = 1;
  cumulativa(1) = 0; cumulativa(2) = 1;
  contribuição(1) = 0; contribuição(2) = 1;
  desvio = |0 - 3/5|+|1 - 2/5| = 6/5;

Decisão tomada: j = 1;
SP: (0 1)
Atualização dos vetores de referência:
S_1: (1 2 3 4 5 6 7)
S_2: (4 3 2 5 7 1 6)

Iteração 2: Decisões possíveis: $j = 2 \rightarrow v(1,2) = 1$;
$j = 6 \rightarrow v(2,6) = 1$;

Cálculo das contribuições de cada vetor para cada decisão:

- j = 2:
  pontuação(1) = 1; pontuação(2) = 0;
  cumulativa(1) = 2; cumulativa(2) = 0;
  contribuição(1) = 1; contribuição(2) = 0;
  desvio = |1 - 3/5|+|0 - 2/5| = 4/5;
- j = 6:
  pontuação(1) = 0; pontuação(2) = 1;
  cumulativa(1) = 1; cumulativa(2) = 1;
  contribuição(1) = 1/2; contribuição(2) = 1/2;
  desvio = |1/2 - 3/5|+|1/2 - 2/5| = 1/5;

Decisão tomada: j = 6;
SP: (0 1 6)
Atualização dos vetores de referência:
S_1: (2 3 4 5 6 7)
S_2: (4 3 2 5 7 6)

Iteração 3: Decisões possíveis: $j = 7 \rightarrow v(1,7) = 1$;
$j = 4 \rightarrow v(2,4) = 1$;

Cálculo das contribuições de cada vetor para cada decisão:

- j = 7:
  pontuação(1) = 1; pontuação(2) = 0;
  cumulativa(1) = 2; cumulativa(2) = 1;
  contribuição(1) = 2/3; contribuição(2) = 1/3;
  desvio = |2/3 - 3/5|+|1/3 - 2/5| = 2/15;
- j = 4:
  pontuação(1) = 0; pontuação(2) = 1;
  cumulativa(1) = 1; cumulativa(2) = 2;
  contribuição(1) = 1/3; contribuição(2) = 2/3;
  desvio = |1/3 - 3/5|+|2/3 - 2/5| = 8/15;

Decisão tomada: j = 7;
SP: (0 1 6 7)

Atualização dos vetores de referência:
S_1: (2 3 4 5 7)
S_2: (4 3 2 5 7)

Iteração 4: Decisões possíveis: j = 2 → v(1,2) = 1;
j = 4 → v(2,4) = 1;
Cálculo das contribuições de cada vetor para cada decisão:
• j = 2:
pontuação(1) = 1; pontuação(2) = 0;
cumulativa(1) = 3; cumulativa(2) = 1;
contribuição(1) = 3/4; contribuição(2) = 1/4;
desvio = |3/4 - 3/5|+|1/4 - 2/5| = 3/10;
• j = 4:
pontuação(1) = 0; pontuação(2) = 1;
cumulativa(1) = 2; cumulativa(2) = 2;
contribuição(1) = 1/2; contribuição(2) = 1/2;
desvio = |1/2 - 3/5|+|1/2 - 2/5| = 2/10;
Decisão tomada: j = 4;
SP: (0 1 6 7 4)
Atualização dos vetores de referência:
S_1: (2 3 4 5)
S_2: (4 3 2 5)

Iteração 5: Decisões possíveis: j = 5 → v(1,5) = 1;
j = 3 → v(2,3) = 1;
Cálculo das contribuições de cada vetor para cada decisão:
• j = 5:
pontuação(1) = 1; pontuação(2) = 0;
cumulativa(1) = 3; cumulativa(2) = 2;
contribuição(1) = 3/5; contribuição(2) = 2/5;
desvio = |3/5 - 3/5|+|2/5 - 2/5| = 0;
• j = 3:
pontuação(1) = 0; pontuação(2) = 1;
cumulativa(1) = 2; cumulativa(2) = 3;
contribuição(1) = 2/5; contribuição(2) = 3/5;
desvio = |2/5 - 3/5|+|3/5 - 2/5| = 2/5;
Decisão tomada: j = 5;
SP: (0 1 6 7 4 5)
Atualização dos vetores de referência:
S_1: (2 3 5)
S_2: (3 2 5)

Iteração 6: Decisões possíveis: j = 2 → v(1,2) = 1;
j = 3 → v(2,3) = 1;
Cálculo das contribuições de cada vetor para cada decisão:
• j = 2:
pontuação(1) = 1; pontuação(2) = 0;
cumulativa(1) = 4; cumulativa(2) = 2;
contribuição(1) = 2/3; contribuição(2) = 1/3;
desvio = |2/3 - 3/5|+|1/3 - 2/5| = 2/15;
• j = 3:
pontuação(1) = 0; pontuação(2) = 1;
cumulativa(1) = 3; cumulativa(2) = 3;
contribuição(1) = 1/2; contribuição(2) = 1/2;
desvio = |1/2 - 3/5|+|1/2 - 2/5| = 3/15;
Decisão tomada: j = 2;
SP: (0 1 6 7 4 5 2)

```
                Atualização dos vetores de referência:
                      S1: (2 3)
                      S2: (3 2)

Iteração 7: Decisões possíveis:     j = 3 → v(1,3) = v(2,3) = 1;
            Centro de influência de S1 e S2:
                  (0 1 6 7 4 5 2 3) com T = 628.
```

A seguir é apresentada a outra abordagem apresentada em GLOVER (1994a) para a combinação de soluções representadas por vetores de permutação. A comparação entre as duas abordagens é feita no final da seção 4.2.4.

4.2.4 - Combinações Estruturadas Adaptativas

Uma combinação estruturada adaptativa segue, em linhas gerais, as mesmas idéias apresentadas para a combinação ponderada. Entretanto, ao contrário de se estabelecerem pesos "a priori" para determinar a influência de cada vetor S_k na solução final, permite-se que essa influência seja determinada dinâmica e adaptativamente, de acordo com a avaliação do efeito das decisões votadas por S_k. Para implementar uma combinação ponderada adaptativa é preciso, então, estabelecer uma forma eficiente de avaliar o impacto de cada decisão votada tanto em relação às decisões passadas como em relação às possíveis decisões futuras.

Outro aspecto importante das combinações adaptativas é que o processo de atualização dos vetores de referência, a cada iteração, deve alterar o mínimo possível o conjunto original de votos desses vetores. Essa condição é chamada de condição de parcimônia e tem um papel importante na construção do centro de influência de um conjunto de vetores. Se as características estruturais dos vetores de referência forem minimamente alteradas durante as atualizações, garante-se que o centro de influência incorpore essas características mais completamente.

Como no caso das combinações ponderadas, a implementação de uma combinação adaptativa de soluções envolve a definição dos seguintes pontos:

(1) Votos;

(2) Avaliação de decisões;

(3) Atualização dos vetores de referência;

(4) Seqüência de tomada de decisões.

A seguir são descritas as escolhas feitas para uma implementação de combinação adaptativa, usada na resolução do mesmo exemplo numérico da seção anterior.

- **Votos**

Ao contrário de se considerar os votos de cada vetor de referência como sendo do tipo 0-1 para relações de precedência, pode-se considerar que cada vetor possua um conjunto de votos correspondentes às relações de precedência presentes na solução que representam e que, a cada iteração do processo de construção do centro de influência, os vetores contribuam com

todos os seus votos. Seja, por exemplo, v(k,i,j) o voto do vetor S_k pela relação "i precede j". Como cada vetor S_k, para o PST, é constituído de (m+n) nós numa permutação cíclica, ele vota, inicialmente, por (m+n) relações de precedência. Por exemplo, o vetor S_1: (0 1 2 3 4 5 6 7) possui os votos v(1,0,1), v(1,1,2), v(1,2,3), v(1,3,4), v(1,4,5), v(1,5,6), v(1,6,7) e v(1,7,0). Conforme as decisões com relação ao centro de influência vão sendo tomadas, o número de votos de cada vetor diminui.

- **Avaliação de decisões**

A avaliação das decisões possíveis a cada iteração envolve duas etapas. A primeira delas consiste da escolha do melhor voto de cada vetor de referência e a segunda corresponde à avaliação dos votos selecionados. Escolher o melhor voto de cada vetor de referência, em vez de usar todos os votos, limita a quantidade de opções e implica na necessidade de definição de uma regra de escolha. Além disso, a qualidade do processo de obtenção do centro de influência pode sofrer uma diminuição decorrente da limitação no número de opções consideradas a cada iteração. Apesar disso, é preciso fazer essa limitação para que a implementação computacional das combinações adaptativas seja mais eficiente.

A regra de escolha do melhor voto de cada vetor S_k depende do problema em estudo e, para um mesmo problema, podem existir várias regras. Para o PST, pode-se atribuir a cada voto v(k,i,j) um valor correspondente, por exemplo, ao comprimento do arco que vai do nó i ao nó j. Esse comprimento foi definido como a soma do tempo de processamento da tarefa j com o tempo de preparação de máquina necessário entre i e j. Outro valor que pode ser atribuído a v(k,i,j) é o atraso da tarefa j ou seu instante de término. A atribuição de valores aos votos faz parte da regra de escolha do melhor voto e influi na qualidade das combinações. Portanto, devem ser realizados testes computacionais para determinar qual aspecto da relação de precedência entre i e j é mais adequado para atribuir valores aos votos. Nas soluções utilizadas no exemplo de combinações ponderadas, se fosse atribuído a cada um dos votos v(k,i,j) um valor correspondente ao tamanho do arco que vai de i para j, esse tamanho seria igual ao tempo de processamento da tarefa j pois não foram considerados tempos de preparação de máquina. Além disso, o nó zero corresponde a um nó de máquina e, conseqüentemente, não é definido um tempo de processamento para ele. Pode-se arbitrar um valor nulo para os votos v(k,i,0), para todo i e todo k.

A partir dos tempos de processamento apresentados na Tabela 4.1, os valores dos votos da solução S_1: (0 1 2 3 4 5 6 7) seriam: v(1,0,1) → 18; v(1,1,2) → 44; v(1,2,3) → 72; v(1,3,4) → 68; v(1,4,5) → 16; v(1,5,6) → 23; v(1,6,7) → 64 e v(1,7,0) → 0. Para esse exemplo, portanto, poderia ser usada a seguinte regra de escolha: "o melhor voto v(k,i*,j*) do vetor S_k corresponde ao arco (i*,j*) de menor comprimento".

A segunda etapa da avaliação de decisões consiste de se construir soluções tentativas, a partir dos vetores de referência, em que cada um dos votos escolhidos na etapa anterior seja

fixado. Dado o voto v(k,i*,j*) escolhido para o vetor S_k, constroem-se soluções tentativas, forçando que a relação "i* precede j*" seja introduzida em cada um dos vetores S_h, h = 1,..,r, h ≠ k. Dados, por exemplo, os vetores S_1: (0 1 2 3 4 5 6 7) e S_2: (0 4 3 2 5 7 1 6) e v(1,i*,j*) = v(1,7,0), a etapa de avaliação do voto v(1,7,0) consiste em forçar a relação "7 precede 0" em S_2. Isso deve ser feito considerando-se a condição de parcimônia e, portanto, causando o menor número possível de mudanças nas demais relações de precedência existentes em S_2.

A introdução da precedência entre os nós 7 e 0 em S_2 pode ser feita de duas formas, cada uma delas gerando uma solução tentativa. A primeira forma consiste em retirar o nó 7 de sua posição atual e inseri-lo na última posição de S_2, obtendo-se S_{tent1}: (0 4 3 2 5 1 6 7) e a segunda consiste em retirar o nó 0 de sua posição atual e inseri-lo imediatamente após o nó 7, obtendo S_{tent2}: (4 3 2 5 7 0 1 6). Lembrando que o nó zero deve sempre estar na primeira posição de qualquer vetor solução, S_{tent2} deve ser rotacionada, tornando-se S_{tent2}: (0 1 6 4 3 2 5 7). Note que foram alteradas em S_2 somente as relações de precedência necessárias para a obtenção de soluções tentativas a partir de v(1,i*,j*). O mesmo processo, então, é aplicado a S_1 para gerar S_{tent3} e S_{tent4} a partir de v(2,i*,j*) = v(2,6,0). Esse é o voto escolhido para S_2 pois possui valor 0. As soluções tentativas construídas a partir de S_1 e v(2,i*,j*) são S_{tent3}: (0 1 2 3 4 5 7 6) e S_{tent4}: (0 7 1 2 3 4 5 6).

A avaliação de v(1,i*,j*) e v(2,i*,j*) é feita calculando-se o custo de cada uma das soluções tentativas de acordo com a função objetivo do problema, ou alguma função de custo auxiliar definida especificamente para esse propósito. A solução tentativa que apresentar o menor custo corresponde à melhor decisão nessa iteração e a relação de precedência correspondente deverá ser fixada em cada um dos vetores de referência. Para o exemplo do parágrafo anterior, o atraso das soluções tentativas é: $T(S_{tent1})$ = 671, $T(S_{tent2})$ = 662, $T(S_{tent3})$ = 508 e $T(S_{tent4})$ = 665. Portanto, a melhor decisão v* é v(2,6,0). A seguir, discute-se como realizar a atualização dos vetores de referência a partir da melhor decisão v*.

- **Atualização dos vetores de referência**

A avaliação de decisões através da construção de soluções tentativas provê uma forma direta de atualizar os vetores de referência a cada iteração. Uma vez escolhida a melhor decisão a ser implementada na iteração atual, cada vetor S_k é substituído pela solução tentativa de menor custo construída a partir dele e da decisão v*.

No caso de S_1 e S_2 acima, após o processo de atualização, obtém-se

S_1: (-0 1 2 3 4 5 7 6-) com $T(S_1)$ = 508 e

S_2: (-0 4 3 2 5 7 1 6-) com $T(S_2)$ = 758.

Há três observações a fazer com relação aos vetores atualizados. A primeira delas é que S_2 não foi alterada pois a relação "6 precede 0" já estava presente nesse vetor. Isso

acontece com todos os vetores S_h, $h = 1,..,r$ que porventura contenham a relação de precedência votada por v^*. O segundo ponto a observar é que, para escolher entre S_{tent3} e S_{tent4} para substituir o vetor S_1 original, escolheu-se S_{tent3} pelo critério $T(S_{tent3}) < T(S_{tent4})$. Finalmente, por uma questão de notação, a precedência fixada na iteração atual é marcada nos vetores atualizados através de um hifem entre os nós i^* e j^*. Essa precedência não poderá ser desfeita na construção de soluções tentativas nas iterações futuras, mesmo que isso cause uma diminuição no número de opções disponíveis.

- **Seqüência da tomada de decisões**

Pelo que foi descrito acima, nota-se que a seqüência em que as decisões são tomadas depende diretamente da regra estabelecida para a escolha do melhor voto de cada vetor de referência, a cada iteração. Portanto, essa seqüência não é uma regra externa ao processo de combinação, mas depende de sua progressão pois os vetores de referência vão sendo alterados de uma iteração para outra, fazendo com que a influência das características estruturais mude dinamicamente.

A seguir, é apresentado um exemplo numérico de combinação estruturada para os mesmos vetores de referência utilizados no exemplo de combinação ponderada. Os dados das tarefas (p_i e d_i, $i = 1,..,7$) são os mesmos do exemplo anterior, apresentados na Tabela 4.1.

- **Exemplo de combinações estruturadas adaptativas**

Regra de seleção do melhor voto de cada vetor: "o melhor voto $v(k,i^*,j^*)$ do vetor S_k corresponde ao arco (i^*,j^*) de menor comprimento".

Vetores de referência:

$S_1 = (0\ 1\ 2\ 3\ 4\ 5\ 6\ 7)$; $T(S_1) = 467$;

$S_2 = (0\ 4\ 3\ 2\ 5\ 7\ 1\ 6)$; $T(S_2) = 758$;

Iteração 1:

Tabela 4.2 - Votos dos vetores S_1 e S_2 na primeira iteração

Voto de S_1	Valor	Voto de S_2	Valor
v(1,0,1)	18	v(2,0,4)	68
v(1,1,2)	44	v(2,4,3)	72
v(1,2,3)	72	v(2,3,2)	44
v(1,3,4)	68	v(2,2,5)	16
v(1,4,5)	16	v(2,5,7)	64
v(1,5,6)	23	v(2,7,1)	18
v(1,6,7)	64	v(2,1,6)	23
v(1,7,0)	0	v(2,6,0)	0

$v(1,i^*,j^*) = v(1,7,0)$ com valor 0;

$v(2,i^*,j^*) = v(2,6,0)$ com valor 0;

- Incorporando "7 precede 0" em S_2:

S_{tent1}: $(0\ 4\ 3\ 2\ 5\ 1\ 6\ 7)$; $T(S_{tent1}) = 671$;

S_{tent2}: $(0\ 1\ 6\ 4\ 3\ 2\ 5\ 7)$; $T(S_{tent2}) = 662$;

- Incorporando "6 precede 0" em S_1:

S_{tent3}: $(0\ 1\ 2\ 3\ 4\ 5\ 7\ 6)$; $T(S_{tent3}) = 508$;

S_{tent4}: $(0\ 7\ 1\ 2\ 3\ 4\ 5\ 6)$; $T(S_{tent4}) = 665$;

- $v^* = v(2,6,0)$

- Atualizando os vetores de referência

S_1 = (-0 1 2 3 4 5 7 6-); $T(S_1)$ = 508;
S_2 = (-0 4 3 2 5 7 1 6-); $T(S_2)$ = 758;

Iteração 2:

Tabela 4.3 - Votos dos vetores S_1 e S_2 na segunda iteração

Voto de S_1	Valor	Voto de S_2	Valor
v(1,0,1)	18	v(2,0,4)	68
v(1,1,2)	44	v(2,4,3)	72
v(1,2,3)	72	v(2,3,2)	44
v(1,3,4)	68	v(2,2,5)	16
v(1,4,5)	16	v(2,5,7)	64
v(1,5,7)	64	v(2,7,1)	18
v(1,7,6)	23	v(2,1,6)	23

v(1,i*,j*) = v(1,4,5) com valor 16;
v(2,i*,j*) = v(2,2,5) com valor 16;

- Incorporando "4 precede 5" em S_2:

S_{tent1}: (-0 4 5 3 2 7 1 6-); $T(S_{tent1})$ = 674;
S_{tent2}: (-0 3 2 4 5 7 1 6-); $T(S_{tent2})$ = 622;

- Incorporando "2 precede 5" em S_1:

S_{tent3}: (-0 1 2 5 3 4 7 6-); $T(S_{tent3})$ = 392;
S_{tent4}: (-0 1 3 4 2 5 7 6-); $T(S_{tent4})$ = 604;

- v* = v(2,2,5)
- Atualizando os vetores de referência

S_1 = (-0 1 2-5 3 4 7 6-); $T(S_1)$ = 392;
S_2 = (-0 4 3 2-5 7 1 6-); $T(S_2)$ = 758;

Iteração 3:

Tabela 4.4 - Votos dos vetores S_1 e S_2 na terceira iteração

Voto de S_1	Valor	Voto de S_2	Valor
v(1,0,1)	18	v(2,0,4)	68
v(1,1,2)	44	v(2,4,3)	72
v(1,5,3)	72	v(2,3,2)	44
v(1,3,4)	68	v(2,5,7)	64
v(1,4,7)	64	v(2,7,1)	18
v(1,7,6)	23	v(2,1,6)	23

v(1,i*,j*) = v(1,0,1) com valor 18;
v(2,i*,j*) = v(2,7,1) com valor 18;

- Incorporando "0 precede 1" em S_2:

S_{tent1}: (-0 1 4 3 2-5 7 6-); $T(S_{tent1})$ = 672;
S_{tent2}: (0 1 6 4 3 2-5 7) Essa solução é inválida pois a relação de precedência entre os nós 6 e 0 foi quebrada. Portanto, ela não é considerada.

- Incorporando "7 precede 1" em S_1:

S_{tent3}: (-0 2 5 3 4 7 1 6-); $T(S_{tent3})$ = 481;
S_{tent4}: (-0 7 1 2-5 3 4 6-); $T(S_{tent4})$ = 557;

- v* = v(2,7,1)
- Atualizando os vetores de referência

S_1 = (-0 2-5 3 4 7-1 6-); $T(S_1)$ = 481;
S_2 = (-0 4 3 2-5 7-1 6-); $T(S_2)$ = 758;

Iteração 4:

Tabela 4.5 - Votos dos vetores S_1 e S_2 na quarta iteração

Voto de S_1	Valor	Voto de S_2	Valor
v(1,0,2)	44	v(2,0,4)	68
v(1,5,3)	72	v(2,4,3)	72
v(1,3,4)	68	v(2,3,2)	44
v(1,4,7)	64	v(2,5,7)	64
v(1,1,6)	23	v(2,1,6)	23

v(1,i*,j*) = v(1,1,6) com valor 23;
v(2,i*,j*) = v(2,1,6) com valor 23;

Como as duas soluções votam pela mesma relação de precedência, é suficiente fixá-la.

- Atualizando os vetores de referência

S_1 = (-0 2-5 3 4 7-1-6-); T(S_1) = 481;
S_2 = (-0 4 3 2-5 7-1-6-); T(S_2) = 758;

Iteração 5:

Tabela 4.6 - Votos dos vetores S_1 e S_2 na quinta iteração

Voto de S_1	Valor	Voto de S_2	Valor
v(1,0,2)	44	v(2,0,4)	68
v(1,5,3)	72	v(2,4,3)	72
v(1,3,4)	68	v(2,3,2)	44
v(1,4,7)	64	v(2,5,7)	64

v(1,i*,j*) = v(1,0,2) com valor 44;
v(2,i*,j*) = v(2,3,2) com valor 44;

- Incorporando "0 precede 2" em S_2:

S_{tent1}: solução inválida;
S_{tent2}: solução inválida;

- Incorporando "3 precede 2" em S_1:

S_{tent3}: (-0 3 2-5 4 7-1-6-); T(S_{tent3}) = 554;
S_{tent4}: solução inválida;

- v* = v(2,3,2)
- Atualizando os vetores de referência

S_1 = (-0 3-2-5 4 7-1-6-); T(S_1) = 554;
S_2 = (-0 4 3-2-5 7-1-6-); T(S_2) = 758;

Iteração 6:

Tabela 4.7 - Votos dos vetores S_1 e S_2 na sexta iteração

Voto de S_1	Valor	Voto de S_2	Valor
v(1,0,3)	72	v(2,0,4)	68
v(1,5,4)	68	v(2,4,3)	72
v(1,4,7)	64	v(2,5,7)	64

v(1,i*,j*) = v(1,4,7) com valor 64;
v(2,i*,j*) = v(2,5,7) com valor 64;

- Incorporando "4 precede 7" em S_2:

S_{tent1}: (-0 3-2-5 4 7-1-6-); T(S_{tent1}) = 554
S_{tent2}: solução inválida;

- Incorporando "5 precede 7" em S_1:

S_{tent3}: solução inválida;
S_{tent4}: solução inválida;

- v* = v(1,4,7)
- Atualizando os vetores de referência

S_1 = (-0 3-2-5 4-7-1-6-); T(S_1) = 554;
S_2 = (-0 3-2-5 4-7-1-6-); T(S_2) = 554;

Como os dois vetores de referência convergiram para a mesma solução, essa solução é o centro de influência e o procedimento pára.

Após esse exemplo, é preciso salientar a grande semelhança entre as combinações estruturadas adaptativas e o procedimento de "Path Relinking". Esse procedimento é apresentado geralmente em conjunto com a Busca Tabu e caracteriza-se pela construção de um caminho alternativo entre soluções visitadas pela busca em iterações distintas. Basicamente, dadas duas soluções S_1 e S_2 visitadas pela Busca Tabu, o "Path Relinking" consiste da incorporação gradativa de atributos(características estruturais) de S_2 em S_1, se o caminho que se deseja construir vai de S_1 para S_2. Esse caminho não necessariamente coincide com a trajetória percorrida pela Busca Tabu, como é ilustrado na Figura 4.12. Nessa figura, a linha cheia representa a trajetória original de busca, passando pelas soluções BT_1 a BT_4, nessa ordem. A linha tracejada representa um caminho possível de ser construído pelo "Path Relinking", passando pelas soluções PR_1 a PR_5.

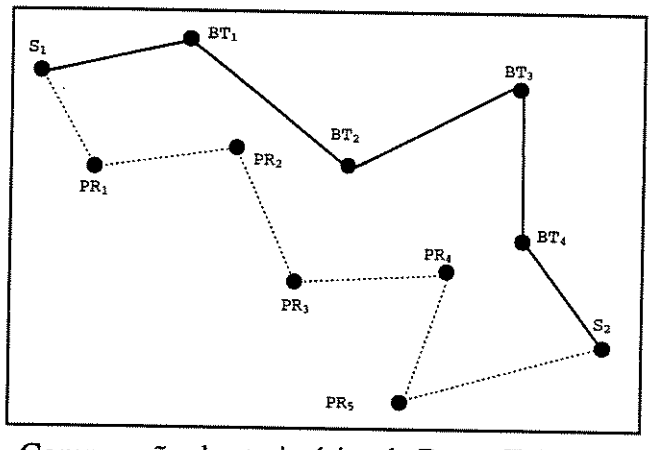

Figura 4.12 - Comparação das trajetórias de Busca Tabu e "Path Relinking"

Essa não é a única forma possível de se realizar o "Path Relinking". Variações são sugeridas na literatura (GLOVER e LAGUNA, 1997) e uma delas se assemelha à construção de um centro de influência de duas soluções. Ao contrário de incorporar sempre os atributos de S_2 em S_1, pode-se fazer a incorporação alternada dos atributos de uma das soluções na outra. No exemplo da Figura 4.12, isso poderia ser feito da seguinte forma: incorpora-se um atributo de S_2 em S_1 para obter PR_1 e, depois disso, incorpora-se um atributo de PR_1 em S_2 obtendo-se PR_5. O processo se repete até que as soluções convirjam para PR_3, por exemplo. Essa solução representaria o centro de influência de S_1 e S_2.

Comparando-se os processos de combinação ponderada e adaptativa, nota-se que esse último cumpre o objetivo de gerar um conjunto disperso a partir do conjunto de soluções de referência mais completamente que o primeiro. A partir de apenas duas soluções, a combinação adaptativa foi capaz de gerar doze soluções tentativas distintas para o PST no exemplo dado, enquanto a combinação ponderada gerou apenas uma. Além disso, nas combinações adaptativas, a influência de cada solução de referência é determinada dinamicamente, evitando a determinação de pesos "a priori". Por esses motivos, as

combinações estruturadas adaptativas foram escolhidas como o processo de geração de soluções tentativas em BE1.

A seguir é formalizado o procedimento de combinação de soluções, na forma de algoritmo, para um conjunto de r soluções de referência S_1, S_2,...,S_r. Esse procedimento é apresentado na sua forma mais geral, independente da regra de atribuição de valores a votos e escolha do melhor voto.

- **Procedimento Combina_Soluções (S_1, S_2, ..., S_r)**

Variáveis: k,l = contadores de soluções de referência;
S_{tent}, S_{tent1} = soluções tentativas;
$T(S_{melhor})$ = custo da melhor solução tentativa encontrada a cada iteração
v(k,i*,j*) = melhor voto da solução S_k;
v*(k*) = melhor decisão a cada iteração, corresponde a v(k*,i*,j*).

Passo 1: Construa uma tabela com os votos de cada solução e seus respectivos valores;
Passo 2: $T(S_{melhor}) = \infty$; k = 1;
Passo 3: Se k > r, vá para o passo 6;
Caso contrário, escolha v(k,i*,j*);
Passo 4: l = 1;
Passo 5: Se l > r, faça k = k + 1 e vá para o passo 3;
Caso contrário, se l ≠ k, faça:
S_{tent} = solução obtida pela inserção de j* imediatamente após i* em S_l;
Se S_{tent} é uma solução válida e $T(S_{tent}) < T(S_{melhor})$:
$T(S_{melhor}) = T(S_{tent})$; k* = k; v*(k*) = v(k,i*,j*);
S_{tent} = solução obtida pela inserção de i* imediatamente antes de j* em S_l;
Se S_{tent} é uma solução válida e $T(S_{tent}) < T(S_{melhor})$:
$T(S_{melhor}) = T(S_{tent})$; k* = k; v*(k*) = v(k,i*,j*);
Faça l = l + 1 e repita o passo 5;
Passo 6: Dado v*(k*) = v(k*,i*,j*), faça l = 1;
Passo 7: Se l > r, vá para o passo 8;
S_{tent} = solução obtida pela inserção de j* imediatamente após i* em S_l;
S_{tent1} = solução obtida pela inserção de i* imediatamente antes de j* em S_l;
Se ambas as soluções forem válidas e $T(S_{tent}) > T(S_{tent1})$ ou se somente S_{tent1} é válida:
$T(S_{tent}) = T(S_{tent1})$; $S_{tent} = S_{tent1}$;
Faça $S_l = S_{tent}$ e $T(S_l) = T(S_{tent})$;
Faça l = l + 1 e repita o passo 7;
Passo 8: Se as soluções atualizadas convergiram para seu centro de influência, pare.
Caso contrário, atualize a tabela de votos de acordo com a decisão tomada no passo 6 e vá para o passo 2.

No Passo 1 a tabela de votos das soluções de referência é construída. A cada iteração do procedimento, essa tabela é atualizada de acordo com a decisão v*(k*) (Passo 8). Nos passos de 2 a 5 é feita a escolha de v*(k*) para a iteração atual. O melhor voto de cada uma das r soluções de referência é avaliado pela construção de soluções tentativas. A solução tentativa de melhor custo determina a decisão a ser implementada no Passo 7.

Como conjunto de soluções a ser utilizado pelo procedimento Combina_Soluções, escolheu-se usar $R_t \cup P_t \backslash P^*_t = H^*_t \cup P_t$. Essa escolha é justificada pelo fato do conjunto possuir as soluções de elite de H^*_t e o subconjunto diverso de todas as soluções geradas até a iteração atual. Espera-se, com isso, reunir características estruturais interessantes presentes nas soluções de menor atraso e nas soluções diversificadas de P_t..

Visto que a forma de combinar soluções a cada iteração de BE1 já está definida, passa-se para a apresentação dos procedimentos de atualização dos conjuntos H_t e P_t.

4.2.5 - Atualização dos Conjuntos de Soluções

Dado que o processo de combinações adaptativas foi escolhido por gerar uma grande quantidade de soluções tentativas, é preciso definir quais dessas soluções serão escolhidas, a cada iteração de BE1 para compor os conjuntos H_{t+1} e P_{t+1}. A forma ideal de realizar essa escolha é armazenar todas as soluções tentativas geradas, ordená-las por seus valores de custo e escolher o novo conjunto H_{t+1}. Feito isso, P_{t+1} seria identificado como um subconjunto diverso das soluções restantes. Infelizmente, o número de soluções tentativas geradas cresce muito quando a dimensão do problema (número de máquinas e tarefas) aumenta. Portanto, o armazenamento de todas essas soluções não é computacionalmente viável. Sendo assim, decidiu-se construir, ao longo de cada iteração t dois conjuntos intermediários H'_t e P'_t que armazenam um subconjunto das soluções tentativas geradas nessa iteração.

Para que se avalie a inserção em H'_t ou P'_t de cada solução gerada pelas combinações estruturadas, o procedimento Combina_Soluções deve ser modificado como segue: cada solução S_{tent}, construída no passo 5 do procedimento Combina_Soluções, é testada para inclusão em um desses conjuntos no momento em que é criada. Se $T(S_{tent})$ for baixo o suficiente para que essa solução seja considerada de elite, ela é inserida em H'_t. Caso contrário, testa-se sua inclusão em P'_t através da regra de construção de subconjuntos diversos, apresentada na seção 4.2.1. O procedimento de construção dos conjuntos intermediários é bastante parecido com o de geração da população inicial e tem como parâmetros de entrada o número de soluções de H'_t e P'_t (h e p, respectivamente) e a solução S_{tent} candidata a entrar em um desses conjuntos.

- **Procedimento Conjuntos_Intermediários(h, p, S_{tent})**

Variáveis : k,l = contadores do número de soluções presentes em H'_t e P'_t, respectivamente;
S_{temp} = solução temporária utilizada em substituições;
d_{ij} = distância entre duas soluções S_i e S_j.
Passo 1: Sejam k e l o número de soluções atualmente em H'_t e P'_t, respectivamente;
Calcule característico(S_{tent});
Passo 2: Se $T(S_{tent}) = T(S_i)$ e característico(S_{tent}) = característico(S_i) para alguma solução $S_i \in H'_t$ ou $S_i \in P''_t$, pare: S_{tent} é uma solução repetida.

```
(Tentativa de inserção de S_tent em H'_t)
Passo 3: Se k < h, faça:
            insira S_tent em H'_t;
            se k + 1 = h, ordene H'_t por valores não decrescentes de
                  custo e sejam H'_t(1), H'_t(2),...,H'_t(h) as soluções
                  de H'_t tais que T(H'_t(1)) ≤ T(H'_t(2))≤...≤T(H'_t(h));
            pare.
Passo 4: Se k = h (H'_t já está ordenado) e T(S_tent) < T(H'_t(h)):
            S_temp = S_tent; S_tent = H'_t(h); H'_t(h) = S_temp;
            Reordene H'_t;
         Vá para o passo 5;
(Tentativa de inserção de S_tent em P'_t)
Passo 5: Se l < p
            Insira S_tent em P'_t
            se l + 1 = p, vá para o passo 6;
         Caso contrário, vá para o passo 7;
Passo 6: Construa a matriz M de distâncias d_ij ∀ S_i, S_j ∈ P'_t; Pare.
Passo 7: Subconjuntos_Diversos (P'_t, S_tent, M);
         Pare.
```

Esse procedimento é utilizado a cada vez que uma solução tentativa é gerada no passo 5 do procedimento Combina_Soluções, que deve ser modificado como segue:

- **Procedimento Combina_Soluções (S_1, S_2,..., S_r)**

```
Passos 1 a 4: como na seção 4.2.4.
Passo 5: Se l > r, faça k = k + 1 e vá para o passo 3;
         Caso contrário, se l ≠ k, faça:
            S_tent = solução obtida pela inserção de j* imediatamente
                     após i* em S_l;
            Se S_tent é uma solução válida:
                  Conjuntos_Intermediários(h,p,S_tent);
                  Se T(S_tent) < T(S_melhor): T(S_melhor) = T(S_tent);
                        k* = k; v*(k*) = v(k,i*,j*);
            S_tent = solução obtida pela inserção de i* imediatamente
                     antes de j* em S_l;
            Se S_tent é uma solução válida:
                  Conjuntos_Intermediários(h,p,S_tent);
                  Se T(S_tent) < T(S_melhor): T(S_melhor) = T(S_tent);
                        k* = k; v*(k*) = v(k,i*,j*);
         Faça l = l + 1 e repita o passo 5;
Passos 6 a 8: como na seção 4.2.4.
```

A modificação realizada faz com que cada solução tentativa seja avaliada para entrar em H'_t ou P'_t. Quando todas as combinações previstas tiverem sido realizadas, os dois conjuntos intermediários estarão prontos e pode-se atualizar H_t e P_t, criando H_{t+1} e P_{t+1}. O procedimento apresentado a seguir realiza essa atualização.

- **Procedimento Atualiza_Conjuntos(h, p)**

```
Variáveis: k = contador de soluções em H_t e P_t;
Passo 1: k = 1;
Passo 2: Se k > h, faça k = 1 e vá para o passo 3;
         Caso contrário:
            Conjuntos_Intermediários(h,p,H_t(k));
            faça k = k + 1 e repita o passo 2;
```

```
Passo 3: Se k > p, faça k = 1 e vá para o passo 4;
         Caso contrário:
            Conjuntos _Intermediários(h,p,P_t(k));
            faça k = k + 1 e repita o passo 3;
Passo 4: Se k > h, faça k = 1 e vá para o passo 5;
         Caso contrário:
            H_{t+1}(k) = H'_t(k);
            faça k = k + 1 e repita o passo 4;
Passo 5: Se k > p, pare: os conjuntos H_{t+1} e P_{t+1} estão completos.
         Caso contrário:
            P_{t+1}(k) = P'_t(k);
            faça k = k + 1 e repita o passo 5;
```

Nos passos 3 e 4, avalia-se se as soluções de H_t e P_t tem características de custo e estrutura que fazem com que seja vantajoso mantê-las em H_{t+1} e P_{t+1}. Quando todas as soluções de H_t e P_t tiverem sido avaliadas, H'_t é copiado para H_{t+1} e P'_t é copiado para P_{t+1} (passos 4 e 5).

Na seção seguinte, apresenta-se BE1 na forma de algoritmo, utilizando os procedimentos descritos nesta seção e nas anteriores.

4.2.6 - Algoritmo

O procedimento de Busca por Espalhamento adaptado para o PST pode ser apresentado da seguinte forma.

- **Algoritmo BE1 (h, h*, p, p*, q_{max})**

```
Passo 1: Dado um problema com m máquinas e n tarefas, gere
            aleatoriamente um valor ψ(i), para cada nó i de tarefa,
            no intervalo [1,131072]. Gere um novo valor ψ no mesmo
            intervalo e faça ψ(i) = ψ para todo nó i de máquina.
Passo 2: t = 0;
         Cria_População_Inicial(h,p,q_max);
Passo 3: Identifique os conjuntos:
            H*_t = {h* melhores soluções de H_t};
            P*_t = {p* melhores soluções de P_t};
            D(P_t\P*_t);
Passo 4: Se o critério de parada já foi satisfeito, pare.
         Caso contrário, vá para o passo 5;
Passo 5: Identifique o conjunto R_t = P*_t ∪ H*_t
Passo 6: Combina_Soluções(R_t ∪ D(P_t\P*_t));
Passo 7: Atualiza_Conjuntos(h,p);
Passo 8: Faça t = t + 1 e vá para o passo 3.
```

No passo 4, dois critérios de parada são previstos. O primeiro deles faz com que o algoritmo pare se encontrar alguma solução com atraso nulo. O segundo critério foi previsto para que o algoritmo BE1 fosse executado por um tempo fixo. Esse tempo é calculado em função do número de máquinas e tarefas, estimando-se o tempo que a melhor versão de Algoritmos Genéticos implementada, GEN1, levaria para resolver um problema de mesma dimensão nas 3000 gerações permitidas. Por exemplo, para os problemas de 4 máquinas e 70 tarefas, GEN1 gasta um tempo médio de 110,214 segundos, utilizando 1033,533

gerações (média para os 30 problemas). Através desses valores, estima-se que, para 3000 gerações, GEN1 gastaria em torno de 320 segundos. Este é o tempo máximo permitido para o algoritmo BE1 na resolução dos mesmos problemas com 4 máquinas e 70 tarefas. Esse critério de parada foi estabelecido tendo em vista o objetivo de comparar o desempenho de BE1 ao de GEN1 e GEN2.

Para que a implementação de BE1 fique completa e os testes de determinação de parâmetros possam ser realizados é preciso definir a regra de atribuição de valor aos votos de cada solução nas combinações adaptativas e a regra de escolha do melhor voto. A princípio, poderia ser usado apenas o comprimento de cada arco (i,j) da solução S_k para determinar o valor de v(k,i,j). Entretanto, deseja-se verificar se a influência do atraso dos nós de tarefa na qualidade estrutural das soluções. Portanto, determinou-se que o valor de cada voto v(k,i,j) seja igual a:

$$\beta.T_j + (1 - \beta).D_{ij}, \text{ com } 0 \leq \beta \leq 1 \qquad (4.8)$$

onde D_{ij} é o comprimento do arco (i,j). Quando j é um nó de máquina, usa-se $T_j = 0$.

A regra de escolha do melhor voto, então, é: "O melhor voto v(k,i*,j*) da solução S_k corresponde ao menor valor de $\beta.T_j + (1 - \beta).D_{ij}$."

4.3 - EXPERIMENTOS COMPUTACIONAIS

4.3.1 - Planejamento

Antes de aplicar BE1 a todo o conjunto de problemas de teste, é preciso determinar os melhores valores dos parâmetros que influenciam seu desempenho. Essa determinação foi feita sobre 15 problemas com 4 máquinas e 70 tarefas, sendo 5 problemas de cada tipo. Os parâmetros a determinar são: h, h*, p, p*, β e q_{max}. Desses seis parâmetros, os mais relevantes são os cinco primeiros pois q_{max} determina somente o tamanho da população inicial. Pode-se manter seu valor fixo durante a fase inicial de testes e, quando os outros parâmetros estiverem ajustados, corrigi-lo adequadamente. Escolheu-se manter $q_{max} = 200$ na fase inicial de determinação dos demais parâmetros.

Como observado em GLOVER (1994a), as cardinalidades dos conjuntos de soluções devem ser mantidas pequenas para que a Busca por Espalhamento possa ser feita de maneira eficiente. Por esse motivo, e também para facilitar o processo de determinação de parâmetros, decidiu-se utilizar p = 2.p*. Os valores de teste para essas cardinalidades foram escolhidos como:

- h = 20, 30, 40;
- h* = 4, 6, 8, 10;
- p* = 4, 6, 8, 10;
- β = 0 a 1,0, variando em intervalos de 0,1.

Na seção seguinte são apresentados os testes realizados com BE1 para determinar os melhores valores dos parâmetros mencionados.

4.3.2 - Determinação de Parâmetros

A primeira fase dos testes computacionais realizados com BE1 consistiu da determinação dos melhores valores para os parâmetros p*, h, h* e β. O parâmetro p não faz parte do conjunto de valores a determinar pois decidiu-se usar p = 2*p.

A determinação de parâmetros de BE1 foi feita pelo Método de Crescimento e Poda de Árvore (MCPA) apresentado na seção 3.1. Para aplicar esse método ao algoritmo, foi preciso, inicialmente, estabelecer a ordem de importância dos parâmetros no desempenho do algoritmo. Testes preliminares foram realizados mas, infelizmente, não foi encontrada nenhuma relação de dominância entre p*, h e h*. Decidiu-se, então, construir a árvore de busca do MCPA em dois níveis. No primeiro deles, fixou-se β = 0,5 e o nó raiz foi ramificado para todas as combinações de valores dos outros três parâmetros, obtendo-se 48 nós. Todos os pares de nós foram comparados através do teste de Wilkoxon visto que as diferenças entre eles correspondem à variação de mais de um parâmetro, impedindo a aplicação do teste de Friedman. Na Tabela 4.8 são apresentados, para cada nó i no primeiro nível da árvore de busca, os valores de h, h* e p*. Os nós eliminados são marcados com um W na coluna Eliminado e o número entre parênteses é o número do nó responsável pela eliminação. Cada nó restante no primeiro nível da árvore de busca foi ramificado variando-se β pelos valores previstos.

Observa-se que os valores mais baixos de p* (e, conseqüentemente, de p) foram eliminados, indicando que a diversidade estrutural das soluções de P_t é importante para o funcionamento de BE1.

Os 14 nós não eliminados no primeiro nível foram ramificados variando o valor do parâmetro β. A cada conjunto de onze nós gerados a partir do mesmo nó pai, aplica-se o teste de Friedman para eliminar os piores nós. Para os nós restantes, que não tenham sido gerados a partir do mesmo pai, aplica-se o teste de Wilkoxon. Na tabela 4.9 são mostrados os nós que restaram ao final desse procedimento. Nessa tabela são apresentados os valores de h, h*, p* e β, assim como o ganho percentual médio calculado por uma expressão equivalente à expressão (3.5). Para cada problema testado, calcula-se:

$$\text{Ganho} = 100 * \frac{T(H) - T(BE)}{T(H)} \qquad (4.9)$$

onde T(BE) é o custo da solução encontrada pela Busca por Espalhamento para um determinado problema e T(H) é o custo obtido pelas heurísticas apresentadas no Capítulo 1.

Tabela 4.8 - Primeiro Nível da Árvore de Busca - BE1

Nó	h	h*	p*	Eliminado	Nó	h	h*	p*	Eliminado
1	20	4	4	W(36)	25	30	8	4	W(36)
2	20	4	6	W(36)	26	30	8	6	W(36)
3	20	4	8		27	30	8	8	W(36)
4	20	4	10	W(36)	28	30	8	10	W(24)
5	20	6	4	W(36)	29	30	10	4	W(36)
6	20	6	6	W(36)	30	30	10	6	W(36)
7	20	6	8		31	30	10	8	
8	20	6	10		32	30	10	10	
9	20	8	4	W(36)	33	40	4	4	W(36)
10	20	8	6	W(36)	34	40	4	6	W(36)
11	20	8	8	W(36)	35	40	4	8	
12	20	8	10		36	40	4	10	
13	20	10	4	W(36)	37	40	6	4	W(36)
14	20	10	6	W(36)	38	40	6	6	W(36)
15	20	10	8	W(36)	39	40	6	8	
16	20	10	10		40	40	6	10	
17	30	4	4	W(36)	41	40	8	4	W(36)
18	30	4	6	W(36)	42	40	8	6	W(36)
19	30	4	8	W(36)	43	40	8	8	W(36)
20	30	4	10		44	40	8	10	
21	30	6	4	W(36)	45	40	10	4	W(36)
22	30	6	6	W(36)	46	40	10	6	W(36)
23	30	6	8	W(36)	47	40	10	8	W(36)
24	30	6	10		48	40	10	10	W(36)

Tabela 4.9 - Resultados da determinação de parâmetros de BE1

Nó	h	h*	p*	β	Ganho(%)
83	20	8	10	0,1	-179,2
93	20	10	10	0	-127,7
94	20	10	10	0,1	-153,4
104	30	4	10	0	-139,7
138	30	10	10	0,1	-156,9
161	40	4	10	0,2	-177,7
181	40	6	10	0	-163,9
192	40	8	10	0	-201,4
193	40	8	10	0,1	-224,3
194	40	8	10	0,2	-191,7

Observa-se que os melhores nós obtidos correspondem a valores altos de p* e h* (com poucas exceções) e valores baixos de β. Isso mostra que a consideração do atraso das tarefas no cálculo dos valores dos votos de cada solução nas combinações adaptativas não contribui favoravelmente para a obtenção de melhores soluções. Um outro fato a ser observado na Tabela 4.9 é que os valores de Ganho são negativos e seus valores absolutos são altos, indicando que as soluções obtidas por BE1 são muito piores que as obtidas pela heurística SAA. Esses valores indicam que BE1, na sua forma atual, não pode ainda ser aplicada aos demais problemas de teste. É preciso realizar modificações no algoritmo para tentar melhorar os valores de Ganho. Essas modificações são descritas na seção seguinte.

4.3.3 - Modificações no Algoritmo

A primeira modificação introduzida em BE1 na tentativa de melhorar seu desempenho

foi a utilização de uma estrutura de memória tabu de curto prazo. Isso afeta a construção do conjunto H^*_t, que deve ser redefinido como:

H^*_t = {h* melhores soluções não tabu de H_t}.

A restrição tabu é imposta a toda solução S_k de H^*_t na iteração atual. Cada solução S_k é proibida de pertencer a H^*_t por um determinado número de iterações se ela permanecer em H_t após suas atualizações consecutivas. A duração tabu é determinada, para cada solução S_k em H^*_t, pela geração de um número aleatório no intervalo [1,max{1,h/h* - 1}]. Os limites desse intervalo foram estabelecidos de forma que S_k permaneça tabu por pelo menos uma iteração e que sempre haja h* soluções não tabu em H_t mesmo que ele permaneça inalterado por um grande número de iterações. Ou seja, mesmo que as combinações adaptativas falhem em gerar soluções boas o bastante para entrar em H_t, haverá sempre h* soluções não tabu disponíveis para formar H^*_t.

Essa modificação foi implementada e testada para os nós da Tabela 4.9. Os desvios médios obtidos são mostrados na Tabela 4.10.

Tabela 4.10 - Novos valores de Ganho usando memória de curto prazo em BE1

Nó	Ganho(%)	Nó	Ganho(%)
83	-149,5	161	-115,4
93	-112,3	181	-140,2
94	-94,8	192	-109,2
104	-127,9	193	-142,8
138	-138,8	194	-89,8

Nota-se uma grande melhora nos valores de Ganho1 com o acréscimo da memória de curto prazo. Isso leva à conclusão de que a diversidade das soluções combinadas é importante tanto para as soluções de P_t como para as soluções de elite em H_t. Mas, como o valor de Ganho ainda é baixo, outras modificações foram inseridas na implementação de BE1, para aumentar esses valores. Como o teste de cada modificação em todos os nós da Tabela 4.10 demanda uma grande quantidade de tempo, escolheram-se os dois melhores nós (94 e 194) para testar as seguintes propostas, mantendo-se a utilização da memória de curto prazo.

Outras modificações testadas em BE1:

(1) Uma das hipóteses levantadas para explicar o baixo desempenho de BE1 foi o fato de que, com muitas soluções a combinar, o algoritmo perfaz poucas iterações. Ou seja, apesar de cada combinação gerar uma grande quantidade de soluções tentativas, são realizadas poucas combinações. A primeira modificação proposta consiste na diminuição do conjunto de soluções a combinar. No Passo 6 do algoritmo BE1, ao contrário de se chamar o procedimento Combina_Soluções para todas as soluções de $R_t \cup P_t \backslash P^*_t$, optou-se por chamar esse procedimento para combinar somente as soluções de R_t entre si.

(2) Outra proposta com o objetivo de aumentar o número de combinações a serem realizadas por BE1 é modificar novamente o Passo 6 do algoritmo de forma que sejam combinados entre si todos os pares de soluções (S_x, S_y) tais que $S_x \in R_t$ e $S_y \in P_t \setminus P^*_t$.

(3) Visando aumentar a diversidade das soluções combinadas, propõe-se a utilização de uma estrutura de memória de médio prazo em BE1. Uma forma de medir a diversidade da busca em médio prazo é construir uma matriz de freqüência de residência para os arcos presentes em todas as soluções de P_t e H_t ao longo da busca. Essa informação é utilizada para alterar os valores dos votos de cada solução nas combinações adaptativas. Cada voto v(k,i,j) recebe um valor calculado como:

$$\beta . T_j + (1 - \beta) . D_{ij} + f_{rel} . D_{ij} \qquad (4.10)$$

onde f_{rel} é a freqüência relativa do arco (i,j) com relação a todos os arcos presentes nas soluções já exploradas pela busca. A regra de escolha do melhor arco ainda é a mesma, favorecendo a escolha do arco de menor valor em cada solução. Essa modificação na determinação dos valores dos votos foi proposta com o objetivo de estimular a utilização dos arcos menos freqüentes nas combinações de soluções.

(4) Finalmente, propõe-se um esquema de atualização de H_t que incorpora alguns aspectos da regra de construção de subconjuntos diversos. Para tanto, na construção do conjunto H'_t, quando uma solução S_{tent} é candidata a entrar nesse conjunto e $T(S_{tent}) < T(H'_t(h))$, S_{tent} não substitui $H'_t(h)$ diretamente, como proposto na seção 4.2.5. A modificação proposta consiste de obter, nessa situação, um novo conjunto H'_t que seja um subconjunto diverso de $H'_t \cup \{S_{tent}\}$. Dessa forma, procura-se aumentar a diversidade das soluções de elite no momento de sua escolha para compor H'_t.

Na Tabela 4.11 são mostrados os valores Ganho após a implementação de cada uma dessas modificações.

Tabela 4.11 - Resultados dos testes das modificações propostas

Modificação	Ganho(%) Nó 94	Ganho(%) Nó 194
(1)	-136,7	-106,7
(2)	-260,5	-264,1
(3)	-127,9	-189,5
(4)	-158,7	-171,5

Infelizmente, todas as modificações causaram uma diminuição no valor de Ganho. Como o melhor valor atual não é satisfatório, decidiu-se tentar algumas modificações mais profundas no algoritmo antes de aplicá-lo a todos os problemas de teste. Duas implementações alternativas de Busca por Espalhamento são apresentadas nos capítulos seguintes. Essas implementações diferem de BE1 na métrica utilizada para medir as distâncias entre soluções e na forma como realizam as combinações estruturadas.

CAPÍTULO 5 - BUSCA POR ESPALHAMENTO IMPLEMENTAÇÃO ALTERNATIVA

Neste capítulo é apresentada uma implementação alternativa para a Busca por Espalhamento descrita no capítulo anterior. Essa implementação alternativa, chamada BE2, utiliza uma nova métrica para o cálculo da distância entre soluções, além de possuir algumas diferenças no procedimento de combinações estruturadas. Os testes computacionais realizados sobre essa implementação são descritos e seus resultados discutidos no final do capítulo.

5.1 - MÉTRICAS DE DISTÂNCIA ENTRE SOLUÇÕES

A métrica proposta para o cálculo de distâncias entre soluções em BE2 considera os dois aspectos principais das soluções do PST do ponto de vista de programação da produção: designação de tarefas a máquinas e seqüenciamento de tarefas em cada máquina. A abordagem seguida neste trabalho consiste de considerar cada um desses aspectos em separado, desenvolvendo dois tipos de medidas de distância. Através de testes computacionais, estabeleceu-se o relacionamento entre essas distâncias de forma que elas realmente traduzissem as diferenças entre quaisquer soluções S_x e S_y da melhor forma possível.

5.1.1 - Distância de Designação

O primeiro passo dado para a definição de uma distância baseada na diferença de designação entre duas soluções foi estabelecer a representação da designação em si. Pode-se visualizar a designação de tarefas a máquinas em uma determinada solução através de um grafo não direcionado com n nós correspondentes às tarefas. Cada aresta (i,j) no grafo de designação de uma solução S_x traduz a seguinte relação: "as tarefas J_i e J_j são processadas pela mesma máquina na solução S_x". Como exemplo, tome a solução S_1 = (0 1 2 3 -1 4 5 -2 6 7). Essa solução representa o programa de produção em que a máquina M_1 processa as tarefas J_1, J_2 e J_3, a máquina M_2 processa J_4 e J_5 e a máquina M_3 processa as tarefas J_6 e J_7. O grafo de designação dessa solução é mostrado na Figura 5.1.

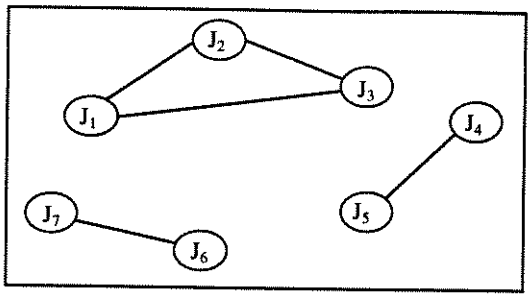

Figura 5.1 - Grafo de designação para a solução S_1

A partir desse grafo de designação pode-se pensar em algumas formas de medir as diferenças entre duas soluções S_x e S_y. Uma proposta consiste em se determinar o número de arestas em comum entre os grafos de S_x e S_y e, a partir desse número, calcular a distância entre elas. Sejam n_x e n_y o número de arestas no grafo de designação de S_x e S_y, respectivamente. Seja n_c o número de arestas em comum entre esses dois grafos. Note que n_x e n_y podem assumir valores diferentes dependendo do número de tarefas processadas em cada máquina, em cada uma das soluções. Dessa forma, o valor n_c possui uma magnitude relativa para cada solução e é melhor expresso como uma porcentagem do número total de arcos em cada uma delas. A diferença percentual entre S_x e S_y, medida a partir de S_x, é $(n_x - n_c)/n_x$ e, quando medida a partir de S_y, é: $(n_y - n_c)/n_y$. A partir desses dois valores, pode-se estabelecer como distância de designação entre S_x e S_y, a medida:

$$dd = \frac{1}{2} \cdot \left[\frac{n_x - n_c}{n_x} + \frac{n_y - n_c}{n_y} \right] 100 \qquad (5.1)$$

Para ilustrar essa medida, sejam: S_1 = (0 1 2 3 -1 4 5 -2 6 7) e S_2 = (0 7 2 6 5 -1 1 -2 4 3). O grafo de designação de S_1 foi apresentado na Figura 5.1. O grafo de S_2 é mostrado na Figura 5.2.

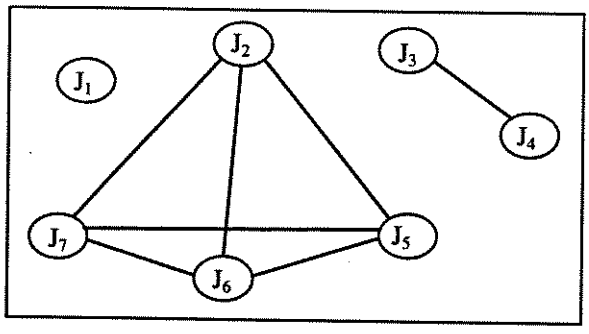

Figura 5.2 - Grafo de designação para a solução S_2

Note que o grafo de S_2 possui mais arestas que o de S_1, pois as tarefas em S_2 estão mais agrupadas em uma das máquinas. No caso limite, em que uma das máquinas processasse todas as tarefas, o grafo de designação seria completo, contendo $n.(n-1)/2$ arestas. Os grafos das soluções S_1 e S_2 possuem apenas uma aresta em comum, ligando as tarefas J_6 e

J_7. Portanto, para essas soluções, $n_c = 1, n_x = n_1 = 5$ e $n_y = n_2 = 7$. Substituindo-se esses valores na equação (5.1), obtém-se $dd = 82,9\%$.

Uma outra medida para a distância de designação entre duas soluções através do grafo, pode ser obtida contando-se o número n_t de tarefas distintas nos extremos das arestas em comum. A medida de distância definida a partir desse número é

$$dd = 100. \frac{n - n_t}{n} \qquad (5.2)$$

Observa-se que, em (5.2), a distância de designação não depende do agrupamento de tarefas em cada solução e é bem mais simples que a distância definida por (5.1). Para os grafos das figuras 5.1 e 5.2, $n_t = 2$ e $n = 7$ tarefas. Portanto,, a distância entre S_1 e S_2, calculada por (5.2), é $dd = 71,4\%$.

Como é impossível prever o impacto de cada uma dessas medidas de distância no desempenho de BE2, a decisão de qual medida usar foi objeto de testes computacionais, descritos mais adiante.

5.1.2 - Distância de Seqüenciamento

As diferenças de seqüenciamento entre duas soluções S_x e S_y podem ser medidas pelas diferenças de posicionamento das tarefas em cada solução. A Tabela 5.1 mostra a posição de cada tarefa na máquina que a processa para as duas soluções S_1 e S_2 utilizadas como exemplo anteriormente.

Tabela 5.1 - Posição das tarefas nas soluções S_1 e S_2

Tarefa	Posição em S_1	Posição em S_2
J_1	1	1
J_2	2	2
J_3	3	2
J_4	1	1
J_5	2	4
J_6	1	3
J_7	2	1

A semelhança entre S_1 e S_2 pode ser medida pelo número de tarefas n_t em posições iguais em ambas as soluções. A diferença entre elas podes ser dada, então, por:

$$ds = 100. \frac{n - n_t}{n} \qquad (5.3)$$

e, para as tarefas na Tabela 5.1, $n_t = 3$ e a distância de seqüenciamento dada por essa expressão é $57,1\%$.

Outra forma de se medir a semelhança entre duas soluções S_x e S_y foi proposta por ARMENTANO e RONCONI (1998) e consiste em atribuir a cada tarefa um número de pontos dependente de seu posicionamento em S_x e S_y. Cada tarefa que estiver na mesma posição nas duas soluções recebe dois pontos. As tarefas que estiverem numa posição em S_x imediatamente anterior ou posterior à posição que ocupam em S_y recebem um ponto. As

demais tarefas não são pontuadas. Calcula-se, então, um valor n_k correspondente à soma das pontuações de todas as tarefas. A diferença entre S_x e S_y é dada, então, por:

$$ds = 100. \frac{2.n - n_k}{2.n}. \qquad (5.4)$$

Para as tarefas na Tabela 5.1, J_1, J_2 e J_4 recebem dois pontos, pois estão nas mesmas posições em ambas soluções. As tarefas J_3 e J_7 recebem um ponto cada e, portanto, $n_k = 8$, $n = 7$ tarefas e, substituindo-se esses valores em (5.4), obtém-se uma diferença de seqüenciamento igual a 42,9%.

Aqui também vale observar que a escolha da melhor forma de se medir a distância de seqüenciamento entre duas soluções foi feita através de testes computacionais, descritos mais adiante.

A seguir, são apresentados todos os elementos de BE2, na mesma ordem em que foram apresentados para BE1, levando-se em consideração as modificações necessárias para acomodar as novas medidas de distância entre soluções e para melhorar o desempenho das combinações de soluções.

5.2 - CONJUNTOS DE SOLUÇÕES

Os conjuntos de soluções utilizados por BE2 são os mesmos de BE1, com exceção de R_t. Esse conjunto é redefinido durante a apresentação do novo procedimento de combinação de soluções. O controle de repetição de soluções permanece inalterado mas ocorre uma modificação na regra de construção de subconjuntos diversos. Isso se deve ao fato de agora existirem duas medidas de distância entre cada par de soluções S_x e S_y. A nova regra é descrita a seguir supondo-se que a distância de seqüenciamento seja mais significativa que a distância de designação e que deva ser considerada em primeiro lugar. A inversão da ordem de consideração das distâncias também foi considerada nos testes computacionais.

Construção de Subconjuntos Diversos:

Como em BE1, a construção de um subconjunto diverso D(X) a partir de um conjunto X de soluções é iniciada pela escolha arbitrária de |D(X)| soluções de X para composição inicial do subconjunto diverso. Essas soluções são retiradas de X e todas as soluções S_t remanescentes nesse conjunto tornam-se candidatas a entrar para D(X).

Para avaliar a entrada de cada uma das soluções candidatas S_t em D(X) é preciso que tenham sido construídas duas matrizes de distância, **MS** para distâncias de seqüenciamento e **MD** para distâncias de designação. Assim como a matriz **M** descrita na seção 4.2.1, **MS** e **MD** são simétricas, tornando-se necessário armazenar apenas os elementos MS_{ij} e MD_{ij} tais que i > j (ou i < j) correspondentes ao elementos abaixo (ou acima) da diagonal. Por

conveniência, utiliza-se $|D(X)|=a$, $|X \backslash D(X)|=b$, ds_{ij} = distância de seqüenciamento entre as soluções S_i e S_j e dd_{ij} = distância de designação entre essas soluções.

$$\mathbf{MS} = \begin{vmatrix} - & & & & & \\ ds_{21} & - & & & & \\ ds_{31} & ds_{32} & - & & & \\ \vdots & & & & & \\ ds_{a-1,1} & ds_{a-1,2} & \cdots & \cdots & ds_{a-1,a-2} & - \\ ds_{a1} & ds_{a2} & \cdots & \cdots & \cdots & ds_{a,a-1} \quad - \end{vmatrix}$$

$$\mathbf{MD} = \begin{vmatrix} - & & & & & \\ dd_{21} & - & & & & \\ dd_{31} & dd_{32} & - & & & \\ \vdots & & & & & \\ dd_{a-1,1} & dd_{a-1,2} & \cdots & \cdots & dd_{a-1,a-2} & - \\ dd_{a1} & dd_{a2} & \cdots & \cdots & \cdots & dd_{a,a-1} \quad - \end{vmatrix}$$

Como a descrição da regra de construção de subconjuntos diversos baseia-se na hipótese de que a distância de seqüenciamento é mais significativa que a de designação, a avaliação da entrada de cada solução candidata S_t em $D(X)$, substituindo uma solução S_i desse conjunto, é feita, inicialmente, medindo-se a distância mínima de seqüenciamento atual entre as soluções de $D(X)$ e a nova distância mínima obtida com a substituição de cada uma de suas soluções por S_t. Nessa fase não se consideram as distâncias de designação.

Analogamente ao que foi feito em BE1, calculam-se os valores MS_i para todas a soluções S_i em $D(X)$ para identificar o valor mínimo MS_{min} e a solução S_k de $D(X)$ tal que $MS_k = MS_{min}$. Dadas as distâncias ds_{ti} entre a solução candidata S_t e as soluções S_i em $D(X)$, calculam-se os valores MS'_i e determina-se a solução S_u tal que $MS'_u = \max_i \{MS'_i\}$. Após o cálculo desses valores, avaliam-se três casos em que a inserção de S_t em $D(X)$ pode ser vantajosa do ponto de vista da distância de seqüenciamento.

Caso 1: $MS'_k > MS_k = MS_{min}$. Nesse caso, a substituição de S_k por S_t em $D(X)$ faz com que o valor mínimo da distância de seqüenciamento entre duas soluções desse conjunto seja aumentado.

Caso 2: Se o teste de substituição do Caso 1 falhar, $MS'_k \leq MS_k$ e pode-se tentar utilizar a solução S_t para melhorar o valor mínimo da linha u da matriz **MS**, aproveitando o fato de que MS'_u é o único valor de MS'_i, $i = 1,..,a$, maior que MS'_k (se $u \neq k$). Nesse caso, se $MS'_u > MS_u$, substitui-se S_u por S_t.

Caso 3: Se os dois testes anteriores falharem, uma última tentativa de inserir S_t em $D(X)$ é realizada. Este último teste consiste em se verificar se é possível manter o valor atual de MS_{min}, melhorando o valor médio do atraso das soluções de $D(X)$. Para isso, constrói-se um subconjunto $D'(X)$ de $D(X)$ composto pelas soluções para as quais $MS'_i \geq MS_i$ e

$T(S_i) > T(S_t)$. Se esse subconjunto não for vazio, escolhe-se uma solução S_v tal que $T(S_v) = \max_{S_i \in D'(X)} \{T(S_i)\}$ e substitui-se S_v por S_t em $D(X)$.

Após a realização desses três testes, se algum deles causar a entrada de S_t em $D(X)$, o procedimento deve passar à avaliação da próxima solução candidata de X. Entretanto, se todos os testes anteriores falharem, antes de descartar S_t, passa-se à avaliação de sua entrada em $D(X)$ através da análise das distâncias de designação. Entretanto, deve-se garantir que, se S_t entrar em $D(X)$ por esse segundo critério, a distância mínima de seqüenciamento MS_{min} não será diminuída. Para isso, novamente constrói-se o subconjunto $D'(X)$ de $D(X)$ composto das soluções para as quais $MS'_i \geq MS_i$. Ou seja, $D'(X)$ é composto pelas soluções de $D(X)$ cuja substituição por S_t não altera o valor de MS_i. A matriz **MD** é reduzida de forma a armazenar somente as distâncias de designação correspondentes às soluções em $D'(X)$. Se esse conjunto não for vazio, realizam-se cálculos análogos aos anteriores para determinar os valores mínimos presentes na matriz **MD** e identificar as soluções S_x tal que $MD_x = MD_{min}$ e S_y tal que $MD'_y = \max_i \{MD'_i\}$. Os testes de substituição realizados são os seguintes.

Caso 1: $MD'_x > MD_x = MD_{min}$. Nesse caso, a substituição de S_x por S_t em $D(X)$ faz com que o valor mínimo da distância de designação entre duas soluções desse conjunto seja aumentado.

Caso 2: Se o teste de substituição do Caso 1 falhar, $MD'_k \leq MD_k$ e pode-se tentar utilizar a solução S_t para melhorar o valor mínimo da linha y da matriz **MD**, aproveitando o fato de que MD'_y é o único valor de MD'_i, $i = 1,..,a$, $S_i \in D'(X)$, maior que MD'_x (se $x \neq y$). Nesse caso, se $MD'_y > MD_y$, substitui-se S_y por S_t.

Caso 3: Se os dois testes anteriores falharem, uma última tentativa de inserir S_t em $D(X)$ é realizada. Verifica-se se é possível manter o valor atual de MD_{min}, melhorando o valor médio do atraso das soluções de $D'(X)$. Para isso, constrói-se um subconjunto $D''(X)$ de $D'(X)$ composto pelas soluções para as quais $MD'_i \geq MD_i$ e $T(S_i) > T(S_t)$. Se esse subconjunto não for vazio, escolhe-se uma solução S_z tal que $T(S_z) = \max_{S_i \in D''(X)} \{T(S_i)\}$ e substitui-se S_z por S_t em $D(X)$.

A nova regra de construção de subconjuntos diversos é colocada na forma de algoritmo a seguir. Os dados de entrada do procedimento são: o conjunto $D(X)$ contendo a soluções, a solução candidata S_t e as matrizes de distâncias **MS** e **MD**.

- **Procedimento Subconjunto_Diverso (D(X), S_t, MS, MD)**

Passo 1: Calcule $MS_i = \min_{\substack{S_j \in D(X) \\ j<i}} \{ds_{ij}\}, i = 2,..,a$;

$MS_1 = \min_{\substack{S_i \in D(X) \\ j \neq i}} \{ds_{i1}\}$;

$MS'_i = \min_{\substack{S_j \in DX) \\ j \neq i}} \{ds_{tj}\}, \forall S_i \in D(X)$;

$MS_{min} = \min_{S_i \in D(X)} \{MS_i\}$;

Passo 2: Encontre os índices k e u tais que $MS_k = MS_{min}$ e $MS'_u = \max_{S_i \in D(X)} \{ds_{ti}\}$;

Passo 3: Se $MS'_k > MS_{min}$ substitua S_k por S_t e vá para o passo 14;
Passo 4: Se $MS'_u > MS_u$ substitua S_u por S_t e vá para o passo 14;
Passo 5: Construa o subconjunto D'(X) contendo as soluções S_i de D(X) tais que $MS'_i \geq MS_i$ e $T(S_i) > T(S_t)$;
Passo 6: Se $D'(X) \neq \emptyset$:
Escolha $S_v \in D'(X)$ tal que $T(S_v) = \max_{S_i \in D'(X)}\{T(S_i)\}$;
Substitua S_v por St e vá para o passo 14;
Passo 7: Construa o subconjunto D'(X) contendo as soluções S_i de D(X) tais que $MS'_i \geq MS_i$;
Se $D'(X) = \emptyset$, vá para o passo 13;
Calcule $MD_i = \min_{\substack{S_j \in D'(X) \\ j<i}}\{dd_{ij}\}, i = 2,..,a; S_i \in D'(X)$;

$MD_1 = \min_{\substack{S_i \in D'(X) \\ j \neq i}}\{dd_{i1}\}$ se $S_1 \in D'(X)$;

$MD_i = \min_{\substack{S_j \in D'(X) \\ j \neq i}}\{dd_{tj}\}, \forall S_i \in D'(X)$;

$MD_{min} = \min_{S_i \in D'(X)}\{MD_i\}$;
Passo 8: Encontre os índices x e y tais que $MD_x = MD_{min}$ e $MD'_y = \max_{S_i \in D'(X)}\{dd_{ti}\}$;
Passo 9: Se $MD'_x > MD_{min}$ substitua S_x por S_t e vá para o passo 14;
Passo 10: Se $MD'_y > MD_y$ substitua S_y por S_t e vá para o passo 14;
Passo 11: Construa o subconjunto D"(X) contendo as soluções S_i de D"(X) tais que $MD'_i \geq MD_i$ e $T(S_i) > T(S_t)$;
Passo 12: Se $D'(X) \neq \emptyset$:
Escolha $S_z \in D''(X)$ tal que $T(S_z) = \max_{S_i \in D''(X)}\{T(S_i)\}$;
Substitua S_z por St e vá para o passo 14;
Passo 13: Descarte a solução S_t e pare.
Passo 14: Atualize as matrizes **MS** e **MD** e pare.

Os procedimentos de construção da população inicial, construção de conjuntos intermediários e atualização dos conjuntos H_t e P_t permanecem inalterados, com exceção dos passos em que são calculadas diferenças entre soluções. Nesses passos, considera-se que os procedimentos calculam as duas medidas de diferença, construindo as matrizes **MS** e **MD** quando necessário. Por se tratar de modificações muito pequenas, esses procedimentos não são reapresentados para BE2. Entretanto, dadas as novas medidas de distância, é preciso redefinir as combinações estruturadas adaptativas para que elas também aproveitem as informações de designação e seqüenciamento presentes nas soluções combinadas. O novo procedimento de combinação de soluções é apresentado a seguir.

5.3 - COMBINAÇÕES ESTRUTURADAS ADAPTATIVAS

A modificação do procedimento de combinação de soluções não diz respeito somente à nova definição dos aspectos relativos aos votos de cada solução e sua avaliação. Analisando o processo de combinação proposto na seção 4.2.4, nota-se que, dado um conjunto de soluções, a busca pelo centro de influência na forma proposta pode fazer com que as soluções tentativas obtidas não sejam muito diversificadas. Tal construção é feita de forma a diminuir a

distância entre as soluções de referência e, de fato, ao final da combinação, todas as soluções convergem para o centro de influência.

Conclui-se que, do ponto de vista da obtenção de soluções dispersas, apenas as primeiras iterações do procedimento `Combina_Soluções` são eficazes, pois nas últimas iterações as soluções de referência apresentam distâncias muito pequenas entre si. Além disso, ao combinar todas as soluções do conjunto de referência ao mesmo tempo, o procedimento `Combina_Soluções` é chamado somente uma vez a cada iteração da Busca por Espalhamento e consome muito tempo computacional na análise das possíveis decisões. Dado que o critério de parada da Busca por Espalhamento é o tempo computacional, poucas iterações são realizadas. Todos esses fatores em conjunto podem constituir uma das causas do mau desempenho apresentado por `BE1`.

Portanto, em `BE2`, o procedimento de combinação de soluções foi modificado de forma a:

- Combinar as soluções de referência duas a duas e não todas ao mesmo tempo, permitindo que o procedimento seja chamado várias vezes por iteração com soluções diferentes. Neste ponto é conveniente redefinir R_t como o conjunto de pares de soluções a serem combinadas. Dessa forma, se a combinação estruturada for realizada entre todos os pares de soluções (S_x,S_y) com $S_x \in H^*_t$ e $S_y \in P^*_t$, R_t é representado como $R_t = H^*_t \times P^*_t$.

- Realizar poucas iterações de combinação estruturada para cada par de soluções, obtendo assim as soluções mais dispersas a partir das soluções de referência e evitando gastar tempo computacional com soluções muito próximas entre si. O número de iterações do procedimento de combinação de soluções a ser realizado é chamado, deste ponto em diante, de profundidade, para que não seja confundido com o número de iterações do algoritmo de busca. A motivação para essa modificação ficará mais clara após a resolução de um exemplo de combinações estruturadas, no final dessa seção.

Antes de apresentar esse exemplo, entretanto, é preciso definir os seguintes pontos:

(1) Votos: qual informação presente nas soluções de referência será utilizada para definir os votos dessas soluções nas combinações;

(2) Valores dos Votos: como avaliar os votos definidos em (1) para determinar os melhores votos de cada solução; e

(3) Construção de Soluções Tentativas: como passar informação de uma solução de referência para outra, tentando obter soluções tentativas dispersas.

Cada um desses pontos é discutido em separado a seguir.

- **Votos**

Considera-se `v(k,i,j)` como o voto da solução S_k para que a tarefa J_i ocupe a posição `j` em uma das máquinas. Para cada solução de referência, existem `n` votos, um para cada tarefa. Por exemplo, a solução S_1 = `(0 1 2 3 -1 4 5 -2 6 7)` possui os

seguintes votos: v(1,1,1), v(1,2,2), v(1,3,3), v(1,4,1), v(1,5,2), v(1,6,1) e v(1,7,2). A diferença básica entre os votos definidos para BE2 e aqueles definidos para BE1 está no tipo de informação que se utiliza para defini-los. No caso de BE1 foi utilizada a informação referente às relações de precedência imediata entre os nós das soluções. Para BE2, considera-se a posição de cada tarefa na máquina que a processa.

- **Avaliação dos Votos**

Para avaliar os votos de cada par de soluções S_1 e S_2, propõe-se um valor baseado na variação do atraso de cada tarefa entre as duas soluções. O voto v(1,1,1) da solução S_1, por exemplo, pode ser avaliado pela diferença de atraso de J_1 nas soluções S_2 e S_1. Essa diferença procura medir a qualidade do voto v(1,1,1) em relação ao voto v(2,1,j), de S_2. Se uma tarefa J_i está mais atrasada em S_1 que em S_2, supõe-se que, quanto maior for a diferença no atraso de J_i nessas duas soluções, maior será a melhoria obtida em S_1 se a posição de J_i votada por S_2 for imposta a S_1. Portanto, para duas soluções de referência S_1 e S_2, cada voto v(1,i,j) recebe um valor

$$\Delta(1,i) = T(2,i) - T(1,i) \qquad (5.5)$$

onde T(1,i) e T(2,i) são os valores de atraso da tarefa J_i nas soluções S_1 e S_2, respectivamente. Cada voto v(2,i,j) recebe um valor $\Delta(2,i) = -\Delta(1,i)$. Escolhe-se como melhor voto de cada solução v(1,i*,j*) e v(2,i*,j*) aquele que apresentar o maior valor $\Delta(1,i)$ e $\Delta(2,i)$. Note que, pela relação entre $\Delta(1,i)$ e $\Delta(2,i)$, o melhor voto da solução S_1 será o pior voto da solução S_2 e vice-versa. Essa característica é boa pois faz com que tarefas diferentes sejam escolhidas no melhor voto de cada solução.

Outros critérios para avaliação dos votos e escolha do melhor voto podem ser utilizados. A escolha do voto de cada solução pode ser feita, inclusive, de forma aleatória para estimular a construção de soluções tentativas mais diversificadas.

- **Construção de Soluções Tentativas**

Tendo sido escolhido o melhor voto v(1,i*,j*) da solução S_1, as soluções tentativas são obtidas forçando-se que a tarefa J_{i*} ocupe a posição j* em todas as máquinas em S_2, através da realização de trocas e inserções de tarefas nessa solução. O processo é então repetido trocando-se os papéis de S_1 e S_2. Cada uma das soluções tentativas obtidas é testada para entrar em um dos conjuntos intermediários H'_t ou P'_t. A seguir é apresentado um pequeno exemplo de como as soluções tentativas são construídas considerando-se que as combinações estruturadas são realizadas até que se encontre o centro de influência entre as soluções de referência. Na Tabela 5.2 encontram-se os dados das tarefas (tempos de processamento e datas de entrega). Assim como no exemplo da seção 4.2.3 e 4.2.4, não são considerados tempos de preparação de máquina para facilitar os cálculos.

Tabela 5 . 2 - Dados para o exemplo de combinações estruturadas

i	1	2	3	4	5	6	7
p_i	18	44	72	68	16	23	64
d_i	124	55	25	210	19	203	191

- **Exemplo da Combinação Estruturada em BE2**

Soluções de Referência: S_1 = (0 1 2 3 -1 4 5 -2 6 7) → $T(S_1)$ = 179
S_2 = (0 7 2 6 5 -1 1 -2 4 3) → $T(S_2)$ = 296

Iteração 1:

Os melhores votos de cada solução são escolhidos pelo critério de maior diferença de atraso de cada tarefa entre as duas soluções de referência:

v(1,i*,j*) = v(1,5,2)
v(2,i*,j*) = v(2,1,1)

Soluções tentativas obtidas por v(1,5,2) sobre S_2:

(1) J_5 na posição 2 da máquina M_1

S_3 = (0 7 5 6 2 -1 1 -2 4 3) → $T(S_3)$ = 268
obtida pela troca de J_5 com J_2

S_4 = (0 7 5 2 6 -1 1 -2 4 3) → $T(S_4)$ = 256
obtida pela inserção de J_5 na segunda posição de M_1

(2) J_5 na posição 2 da máquina M_2:

S_5 = (0 7 2 6 -1 1 5 -2 4 3) → $T(S_5)$ = 183
obtida pela inserção de J_5 na segunda posição de M_2

(3) J_5 na posição 2 da máquina M_3:

S_6 = (0 7 2 6 3 -1 1 -2 4 5) → $T(S_6)$ = 296
obtida pela troca de J5 com J3

S_7 = (0 7 2 6 -1 1 -2 4 5 3) → $T(S_7)$ = 249
obtida pela inserção de J_5 na segunda posição de M_3

Soluções tentativas obtidas por v(2,1,1) sobre S_1:

(1) J_1 na posição 1 da máquina M_2

S_8 = (0 4 2 3 -1 1 5 -2 6 7) → $T(S_8)$ = 231
obtida pela troca de J_1 com J_4

S_9 = (0 2 3 -1 1 4 5 -2 6 7) → $T(S_9)$ = 174
obtida pela inserção de J_1 na primeira posição M_2

(2) J_1 na posição 1 da máquina M_3:

S_{10} = (0 6 2 3 -1 4 5 6 -2 1 7) → $T(S_{10})$ = 191
obtida pela troca de J_1 com J_6

S_{11} = (0 2 3 -1 4 5 6 -2 1 6 7) → $T(S_{11})$ = 156
obtida pela inserção de J_1 na primeira posição de M_3

O melhor voto é escolhido como aquele que possibilitou obter a solução tentativa de menor custo.

Melhor voto: v* = v(2,1,1)

A tarefa J_1 fica fixa na posição 1 e não se permite, a partir desse ponto que nenhuma solução tentativa seja obtida alterando-se essa posição. Permite-se, entretanto, que a tarefa J_1 mude de máquina, desde que seja processada na primeira posição. Nas soluções atualizadas, as tarefas que tiverem suas posições fixadas aparecem em negrito.

Atualização das soluções de referência:

S_1 → S_{11} = (0 2 3 -1 4 5 6 -2 **1** 6 7) → $T(S_1)$ = 156

S_2 = (0 7 2 6 5 -1 **1** -2 4 3) → $T(S_2)$ = 296

Iteração 2:

v(1,i*,j*) = v(1,5,2)
v(2,i*,j*) = v(2,4,1)

Soluções tentativas obtidas por v(1,5,2) sobre S_2:

(Como a solução S_2 não foi alterada e o voto de S_1 é o mesmo da iteração anterior, obtém-se as mesmas soluções tentativas a partir de S_2 e v(1,5,2). Por esse motivo, essas soluções são apresentadas com os mesmos índices utilizados na iteração 1)

(1) J_5 na posição 2 da máquina M_1

S_3 = (0 7 5 6 2 -1 **1** -2 4 3) → $T(S_3)$ = 268

S_4 = (0 7 5 2 6 -1 **1** -2 4 3) → $T(S_4)$ = 256

(2) J_5 na posição 2 da máquina M_2:

S_5 = (0 7 2 6 -1 **1** 5 -2 4 3) → $T(S_5)$ = 183

(3) J_5 na posição 2 da máquina M_3:

S_6 = (0 7 2 6 3 -1 **1** -2 4 5) → $T(S_6)$ = 296

S_7 = (0 7 2 6 -1 **1** -2 4 5 3) → $T(S_7)$ = 249

Soluções tentativas obtidas por v(2,4,1) sobre S_1:

(1) J_4 na posição 1 da máquina M_1

S_{12} = (0 4 3 -1 2 5 -2 **1** 6 7) → $T(S_{12})$ = 156

S_{13} = (0 4 2 3 -1 5 -2 **1** 6 7) → $T(S_{13})$ = 216

(2) J_4 na posição 1 da máquina M_3:

S_{14} = (0 2 3 -1 **1** 5 -2 4 6 7) → $T(S_{14})$ = 106

Melhor voto: v* = v(2,4,1)

Atualização das soluções de referência:

S_1 → S_{14} = (0 2 3 -1 **1** 5 -2 **4** 6 7) → $T(S_1)$ = 106

S_2 = (0 7 2 6 5 -1 **1** -2 **4** 3) → $T(S_2)$ = 296

Iteração 3:

v(1,i*,j*) = v(1,5,2)

v(2,i*,j*) = v(2,6,3)

Soluções tentativas obtidas por v(1,5,2) sobre S_2:

(Novamente, a solução S_2 não foi alterada e o voto de S_1 é o mesmo da iteração anterior, obtém-se as mesmas soluções tentativas a partir de S_2 e v(1,5,2). Por esse motivo, essas soluções são apresentadas com os mesmos índices utilizados na iteração 1)

(1) J_5 na posição 2 da máquina M_1

S_3 = (0 7 5 6 2 -1 **1** -2 **4** 3) → $T(S_3)$ = 268

S_4 = (0 7 5 2 6 -1 **1** -2 **4** 3) → $T(S_4)$ = 256

(2) J_5 na posição 2 da máquina M_2:

S_5 = (0 7 2 6 -1 **1** 5 -2 **4** 3) → $T(S_5)$ = 183

(3) J_5 na posição 2 da máquina M_3:

S_6 = (0 7 2 6 3 -1 **1** -2 **4** 5) → $T(S_6)$ = 296

S_7 = (0 7 2 6 -1 **1** -2 **4** 5 3) → $T(S_7)$ = 249

Soluções tentativas obtidas por v(2,6,3) sobre S_1:

(1) J_6 na posição 3 da máquina M_1

S_{16} = (0 2 3 6 -1 **1** 5 -2 **4** 7) → $T(S_{16})$ = 106

(2) J_6 na posição 1 da máquina M_2:

S_{17} = (0 2 3 -1 **1** 5 6 -2 **4** 7) → $T(S_{17})$ = 106

(3) J_6 na posição 1 da máquina M_3:

S_{18} = (0 2 3 -1 **1** 5 -2 **4** 7 6) → $T(S_{14})$ = 106

Melhor voto: v* = v(2,6,3)

Atualização das soluções de referência:

S_1 → S_{16} = (0 2 3 **6** -1 **1** 5 -2 **4** 7) → $T(S_1)$ = 106

S_2 = (0 7 2 **6** 5 -1 **1** -2 **4** 3) → $T(S_2)$ = 296

Iteração 4:

v(1,i*,j*) = v(1,5,2)
v(2,i*,j*) = v(2,7,1)

Soluções tentativas obtidas por v(1,5,2) sobre S_2:
(Na apresentação das soluções tentativas obtidas a partir de S2 e v(1,5,2), exclui-se a solução S4 pois ela viola uma das posições fixadas)

(1) J_5 na posição 2 da máquina M_1

S_3 = (0 7 5 **6** 2 -1 **1** -2 4 3) → $T(S_3)$ = 268

(2) J_5 na posição 2 da máquina M_2:

S_5 = (0 7 2 **6** -1 **1** 5 -2 4 3) → $T(S_5)$ = 183

(3) J_5 na posição 2 da máquina M_3:

S_6 = (0 7 2 **6** 3 -1 **1** -2 **4** 5) → $T(S_6)$ = 296

S_7 = (0 7 2 **6** -1 **1** -2 **4** 5 3) → $T(S_7)$ = 249

Soluções tentativas obtidas por v(2,7,1) sobre S_1:

(1) J_7 na posição 1 da máquina M_1

S_{19} = (0 7 3 **6** -1 **1** 5 -2 **4** 2) → $T(S_{19})$ = 183

Melhor voto: v* = v(1,5,2)
Atualização das soluções de referência:

S_1 = (0 2 3 **6** -1 **1** **5** -2 **4** 7) → $T(S_1)$ = 106

S_2 → S_5= (0 7 2 **6** -1 **1** **5** -2 **4** 3) → $T(S_2)$ = 183

Iteração 5:

v(1,i*,j*) = v(1,2,1)
v(2,i*,j*) = v(2,7,1)

Soluções tentativas obtidas por v(1,2,1) sobre S_2:

(1) J_2 na posição 1 da máquina M_1

S_{20} = (0 2 7 **6** -1 **1** **5** -2 **4** 3) → $T(S_{20})$ = 130

Soluções tentativas obtidas por v(2,7,1) sobre S_1:

(1) J_7 na posição 1 da máquina M_1

S_{19} = (0 7 3 **6** -1 **1** 5 -2 **4** 2) → $T(S_{19})$ = 183

Melhor voto: v* = v(1,2,1)
Atualização das soluções de referência:

S_1 = (0 **2** 3 **6** -1 **1** **5** -2 **4** 7) → $T(S_1)$ = 106

S_2 → S_{20}= (0 **2** 7 **6** -1 **1** **5** -2 **4** 3) → $T(S_2)$ = 130

Iteração 6:

v(1,i*,j*) = v(1,3,2)
v(2,i*,j*) = v(2,7,2)

Soluções tentativas obtidas por v(1,3,2) sobre S_2:

(1) J_2 na posição 1 da máquina M_1

S_{21} = (0 **2** 3 **6** -1 **1** **5** -2 **4** 7) → $T(S_{21})$ = 106

S_{22} = (0 **2** 7 **6** -1 **1** 3 -2 **4** **5**) → $T(S_{22})$ = 130

Soluções tentativas obtidas por v(2,7,2) sobre S_1:

(1) J_7 na posição 1 da máquina M_1

S_{23} = (0 **2** 7 **6** -1 **1** **5** -2 **4** 3) → $T(S_{23})$ = 130

S_{24} = (0 **2** 3 **6** -1 **1** 7 -2 **4** **5**) → $T(S_{24})$ = 156

Melhor voto: v* = v(1,3,2)
Atualização das soluções de referência:

S_1 = (0 **2** **3** **6** -1 **1** **5** -2 **4** 7) → $T(S_1)$ = 106

S_2 → S_{21}= (0 **2** **3** **6** -1 **1** **5** -2 **4** 7) → $T(S_2)$ = 106

Como as duas soluções chegaram ao seu centro de influência, o procedimento pára.

Pode-se interpretar a combinação estruturada proposta como uma busca bastante restrita na vizinhança das soluções de referência composta por todos os movimentos de troca e inserção de tarefas. A restrição da vizinhança de S_1 é feita pela regra de avaliação e escolha do melhor voto de S_2: $v(2,i^*,j^*)$ pois são consideradas como soluções tentativas apenas as soluções da vizinhança de S_1 em que J_{i^*} ocupe a posição j^* em uma das máquinas. O mesmo é válido para S_2 quando os papéis das soluções de referência são trocados nas combinações. Formalmente, quando modificações em S_1 são consideradas, S_2 funciona como um guia de exploração da vizinhança de S_1, restringindo-a através de $v(2,i^*,j^*)$ e, quando modificações em S_2 são consideradas, S_1 é a solução guia, e a restrição da vizinhança de S_2 é imposta por $v(1,i^*,j^*)$.

No exemplo resolvido acima, observa-se que o número de soluções distintas geradas a partir da segunda iteração é bastante pequeno pois, à medida que as tarefas vão tendo suas posições fixadas, a quantidade de opções para a construção de soluções tentativas diminui. Além disso, observa-se que as soluções tentativas construídas nas últimas iterações do exemplo são muito parecidas entre si, indicando que as soluções mais diversificadas foram obtidas nas primeiras iterações. Por esses motivos, o número de iterações do procedimento de combinações estruturadas a serem realizados para cada par de soluções passa a ser controlado pelo parâmetro `profundidade`. A seguir, é apresentado o procedimento de combinação de soluções na forma de algoritmo. Os dados de entrada são as duas soluções a combinar e a profundidade que se deseja atingir na combinação dessas soluções. O procedimento também usa o parâmetro m = número de máquinas.

- **Procedimento Combina_Soluções (S_1, S_2, profundidade)**

Variáveis: k = contador de máquinas nas soluções de referência;
prof = contador de iterações (profundidade);
S_{tent} = solução tentativa;
(S_{tent^*},i) = (Melhor solução tentativa encontrada, solução de referência que deve ser substituída por S_{tent^*} na atualização), i pode assumir os valores 1 e 2;
[x,k] = índice da tarefa processada na posição x da máquina M_k nas soluções de referência
n_k = número de tarefas processadas pela máquina M_k;

Passo 1: prof = 1;

Passo 2: Se prof > profundidade ou se as soluções de referência convergiram para seu centro de influência, Pare.

Passo 3: Construa uma tabela com os votos de cada solução e seus respectivos valores;
Faça $T(S_{tent^*}) = \infty$;

Passo 4: Escolha o melhor voto da solução S_1: $v(1,i^*,j^*)$

Passo 5: k = 1;

Passo 6: Se k > m, faça k = 1 e vá para o passo 10;

Passo 7: Se J_{i^*} não é processada por M_k em S_2:
Se $n_k > j^* - 1$:
S_{tent} = solução tentativa obtida a partir de S_2 pela inserção de J_{i^*} imediatamente após $J_{[j^*-1,k]}$;

```
                    Conjuntos_Intermediários(h,p,S_tent);
                    Se T(S_tent)<T(S_tent*) faça (S_tent*,i) = (S_tent,2);
                    S_tent = solução tentativa obtida a partir de S2 pela
                          troca de J_i* por J_[j*,k];
                    Conjuntos_Intermediários(h,p,S_tent);
                    Se T(S_tent)<T(S_tent*) faça (S_tent*,i) = (S_tent,2);
              Se n_k = j* - 1:
                    S_tent = solução tentativa obtida a partir de S2 pela
                          inserção de J_i* imediatamente após J_[j*,k];
                    Conjuntos_Intermediários(h,p,S_tent);
                    Se T(S_tent)<T(S_tent*) faça (S_tent*,i) = (S_tent,2);
        Caso contrário, vá para o passo 8;
Passo 8: Se J_i* é processada em M_k em S2 e J_[j*,k] ≠ J_i*:
           Se n_k > j* - 1
                    S_tent = solução tentativa obtida a partir de S2 pela
                          inserção de J_i* imediatamente após J_[j*-1,k];
                    Conjuntos_Intermediários(h,p,S_tent);
                    Se T(S_tent)<T(S_tent*) faça (S_tent*,i) = (S_tent,2);
                    S_tent = solução tentativa obtida a partir de S2 pela
                          troca de J_i* por J_[j*,k];
                    Conjuntos_Intermediários(h,p,S_tent);
                    Se T(S_tent)<T(S_tent*) faça (S_tent*,i) = (S_tent,2);
Passo 9: Faça k = k + 1 e vá para o passo 6;
Passo 10: Escolha o melhor voto da solução S2: v(2,i*,j*)
          Faça k = 1;
Passo 11: Se k > m, vá para o passo 15;
Passo 12: Se J_i* não é processada por M_k em S1:
           Se n_k > j* - 1:
                    S_tent = solução tentativa obtida a partir de S1 pela
                          inserção de J_i* imediatamente após J_[j*-1,k];
                    Conjuntos_Intermediários(h,p,S_tent);
                    Se T(S_tent)<T(S_tent*) faça (S_tent*,i) = (S_tent,1);
                    S_tent = solução tentativa obtida a partir de S1 pela
                          troca de J_i* por J_[j*,k];
                    Conjuntos_Intermediários(h,p,S_tent);
                    Se T(S_tent)<T(S_tent*) faça (S_tent*,i) = (S_tent,1);
              Se n_k = j* - 1:
                    S_tent = solução tentativa obtida a partir de S1 pela
                          inserção de J_i* imediatamente após J_[j*,k];
                    Conjuntos_Intermediários(h,p,S_tent);
                    Se T(S_tent)<T(S_tent*) faça (S_tent*,i) = (S_tent,1);
        Caso contrário, vá para o passo 13;
Passo 13: Se J_i* é processada em M_k em S1 e J_[j*,k] ≠ J_i*:
           Se n_k > j* - 1
                    S_tent = solução tentativa obtida a partir de S1 pela
                          inserção de J_i* imediatamente após J_[j*-1,k];
                    Conjuntos_Intermediários(h,p,S_tent);
                    Se T(S_tent)<T(S_tent*) faça (S_tent*,i) = (S_tent,1);
                    S_tent = solução tentativa obtida a partir de S1 pela
                          troca de J_i* por J_[j*,k];
                    Conjuntos_Intermediários(h,p,S_tent);
                    Se T(S_tent)<T(S_tent*) faça (S_tent*,i) = (S_tent,1);
Passo 14: Faça k = k + 1 e vá para o passo 11.
Passo 15: Dado (S_tent*,i), faça S_i = S_tent*;
          Faça prof = prof + 1 e vá para o passo 2;
```

Nos passos 4 a 9 são construídas todas as soluções tentativas a partir de S_2 e do melhor voto de S_1 e os passos 10 a 14, as soluções tentativas a partir de S_1 e o melhor voto de S_2. A variável (S_{tent*},i) armazena em S_{tent*} a melhor solução tentativa construída a cada iteração (profundidade) e, em i, o índice da solução (S_1 ou S_2) que deve ser substituída por

S_{tent^*} a cada vez que as soluções de referência são atualizadas. Essa atualização é feita no passo 15. O procedimento Conjuntos_Intermediários é chamado para cada solução tentativa gerada para que essa solução possa ser testada para entrar em um dos conjuntos intermediários H'_t e P'_t. Esse procedimento é análogo ao apresentado para BE1.

A utilização do procedimento Combina_Soluções por BE2 é simples. A cada iteração, após a identificação do conjunto R_t de pares de soluções de referência, chama-se o procedimento para cada um desses pares. A seguir, BE2 é apresentada na forma da algoritmo.

- **Algoritmo BE2 (h, h*, p, p*, q_{max}, profundidade)**

Passo 1: Dado um problema com m máquinas e n tarefas, gere aleatoriamente um valor ψ(i), para cada nó i de tarefa, no intervalo [1,131072]. Gere um novo valor ψ no mesmo intervalo e faça ψ(i) = ψ para todo nó i de máquina.
Passo 2: t = 0;
Cria_População_Inicial(h,p,q_{max});
Passo 3: Identifique os conjuntos:
H^*_t = {h* melhores soluções de H_t};
P^*_t = {p* melhores soluções de P_t};
Passo 4: Se o critério de parada já foi satisfeito, pare.
Caso contrário, vá para o passo 5;
Passo 5: Identifique o conjunto R_t = {pares de soluções a combinar};
Passo 6: Para cada par de soluções (S_x,S_y) ∈ R_t:
Combina_Soluções(S_x, S_y, profundidade);
Passo 7: Atualiza_Conjuntos(h,p);
Passo 8: Faça t = t + 1 e vá para o passo 3.

5.4 - TESTES COMPUTACIONAIS

5.4.1 - Planejamento

Os testes apresentados nesta seção possuem dois objetivos distintos. O primeiro deles é determinar a melhor configuração da busca baseando-se nas opções apresentadas na seção 5.1.1 e 5.1.2 no que diz respeito à escolha da melhor medida de distância e a ordem de consideração dessas medidas na regra de construção de subconjuntos diversos. Além disso, faz parte da determinação da melhor configuração a escolha adequada de soluções de referência. O segundo objetivo dos testes é determinar o melhor conjunto de parâmetros h, h* e p*, visto que p foi mantido igual a 2.p*.

O primeiro passo no planejamento dos testes computacionais é a especificação de todas as características que se deseja investigar. Cada uma dessas características é apresentada a seguir, na forma de fatores determinantes do desempenho de BE2.

Fator 1: Medida de Distância de Designação

$$\text{Por arestas: } dd = \frac{1}{2} \cdot \left[\frac{n_x - n_c}{n_x} + \frac{n_y - n_c}{n_y} \right] .100$$

Por tarefas: $dd = 100 . \frac{n - n_t}{n}$

Fator 2: Medida de Distância de Seqüenciamento

Simples: $ds = 100 . \frac{n - n_t}{n}$

Gradual: $ds = 100 . \frac{2.n - n_k}{2.n}$

Fator 3: Escolha do melhor voto de cada solução de referência

Por atraso: $\Delta(1,i) = T(2,i) - T(1,i)$; $\Delta(2,i) = -\Delta(1,i)$

Escolha aleatória

Fator 4: Soluções a Combinar

$R_t = H^*_t \times P^*_t$

$R_t = H^*_t \times P_t$

$R_t =$ {melhor solução de H^*_t} $\times P^*_t$

$R_t =$ {melhor solução de H^*_t} $\times P_t$

Fator 5: Ordem da consideração das distâncias na regra de construção de Subconjuntos Diversos

Distância de Seqüenciamento antes de Distância de Designação

Distância de Designação antes de Distância de Seqüenciamento

Consideração simultânea das duas distâncias através de sua soma

Os valores de teste dos parâmetros de BE2 são:

- $h^* = 4, 6, 8, 10$;
- $p^* = 4, 6, 8, 10$;
- $h = 10, 20, 30, 40$;
- $q_{max} = 200$.

A experiência realizada com BE1 mostrou que, apesar de não haver uma relação de dominância muito clara entre os três parâmetros h, h* e p*, a Busca por Espalhamento é relativamente robusta com respeito ao parâmetro h. Isso pode ser verificado na Tabela 4.9, onde são mostrados os nós restantes após a eliminação feita pelo teste de Wilkoxon. Todos os valores de teste de h estão presentes na coluna 2 dessa tabela, indicando que pode-se construir a árvore de busca do MCPA ramificando-se os valores de h em um nível diferente daquele em que se ramificam os valores de h* e p*. Além disso, a determinação de parâmetros de BE1 mostrou que o MCPA deve ser utilizado com um pouco mais de critério para que não se realize uma quantidade muito grande de testes.

Devido ao grande número de fatores que se deseja investigar e à grande quantidade de possíveis combinações de valores dos parâmetros, os testes computacionais com BE2 foram divididos em 2 fases.

Fase 1 : Eliminação de Configurações

Nessa fase, manteve-se o fator 5 fixo escolhendo-se "Distância de Seqüenciamento antes de Distância de Designação" na regra de construção de subconjuntos diversos. A cada versão de BE2, obtida pela combinação das opções possíveis para os fatores 1 a 4, aplicou-se o MCPA para determinar os melhores valores dos parâmetros h, h* e p*. Apesar de existirem 32 versões possíveis, os testes foram feitos em etapas de forma a eliminar os valores de teste menos eficazes para esses parâmetros, diminuindo o esforço computacional necessário.

Ao final da Fase 1, para cada versão testada, os nós restantes no último nível de cada árvore de busca do MCPA foram comparados aos pares pelo teste de Wilkoxon para obtenção de um conjunto reduzido de configurações dominantes de BE2 com relação aos fatores 1 a 4 e aos parâmetros da busca.

Fase 2: Refinamento

Nessa fase, cada uma das configurações não eliminadas ao final da Fase 1 foi testada variando-se a opção escolhida para a ordem de consideração das distâncias na construção de subconjuntos diversos. Todas as configurações alternativas foram comparadas novamente pelo teste de Wilkoxon, eliminando-se aquelas de pior desempenho.

Além disso, testes adicionais foram incluídos na Fase 2 para investigar pequenas modificações no esquema proposto para BE2.

Ao final dos testes da Fase 2, concluiu-se que todas as configurações restantes são consideradas estatisticamente iguais. Nesse ponto pôde-se, então, escolher a configuração que apresenta o melhor valor de Ganho, calculado de acordo com a expressão (4.9).

Durante todos os testes da seção 5.1.5, apenas uma iteração de combinação estruturada foi realizada para cada par de soluções. Ou seja, o procedimento foi chamado na forma Combina_Soluções(S_1, S_2, 1). A investigação do efeito do uso de profundidades maiores foi feita para cada par de valores de (m,n), após os testes da Fase 2.

5.4.2 - Resultados

A) Fase 1

Os testes realizados na Fase 1 para a determinação da melhor configuração de BE2 iniciaram-se pela determinação de uma versão básica da busca, denominada Versão 1. Essa versão possui as seguintes características:

Fator 1: distância de designação por arestas

Fator 2: distância de seqüenciamento simples

Fator 3: escolha do melhor voto por atraso

Fator 4: $R_t = H^*_t \times P^*_t$.

Aplicou-se o MCPA a essa versão, ramificando-se 16 nós no primeiro nível da árvore de busca, cada nó correspondendo a uma das combinações dos valores de teste de p* e h*. Novamente, o fato de cada nó do primeiro nível da árvore diferir dos demais pelo valor de

mais de um parâmetro, o teste de Friedman não pôde ser utilizado e os nós foram comparados através do teste de Wilkoxon. Os nós não eliminados no primeiro nível, após a aplicação desse teste são mostrados na Tabela 5.3.

Tabela 5.3 - Nós restantes no primeiro nível do MCPA para a Versão 1 de BE2

Nó	h*	p*
1	4	4
2	4	6
5	6	4
9	8	4
13	10	4

Cada um desses nós foi ramificado no segundo nível da árvore de busca, variando-se o valor de h. A tabela 5.4 mostra os nós do segundo nível. Nessa tabela, a coluna Eliminados possui um F se o nó foi eliminado pelo teste de Friedman e um W se o nó foi eliminado pelo testes de Wilkoxon. Neste último caso, o número entre parênteses indica o nó dominante que causou a eliminação.

Tabela 5.4 - Último nível do MCPA para a Versão 1 de BE2

Nó	h	h*	p*	Eliminado
17	10			
18	20	4	4	
19	30			
20	40			
21	10			W(17)
22	20	4	6	W(17)
23	30			W(17)
24	40			W(17)
25	10			F
26	20	6	4	
27	30			
28	40			F
29	10			W(17)
30	20	8	4	W(17)
31	30			W(17)
32	40			W(17)
33	10			W(17)
34	20	10	4	W(17)
35	30			W(17)
36	40			W(17)

Observa-se, pela Tabela 5.4, que os valores mais altos de p* e h* geram resultados mais pobres para BE2 pois todos os nós com p* = 8 e 10 foram eliminados no primeiro nível da árvore de busca e todos os nós com h* = 8 e 10 e p* = 6 foram eliminados no segundo nível. Poderia-se dar continuidade aos testes das demais versões de BE2 apenas com p* = 4 e h* = 4 e 6. Entretanto, concluiu-se que uma eliminação tão extensa dos valores de teste dos parâmetros não deveria ser feita tendo sido testada apenas uma das 32 versões.

Optou-se por fazer uma redução mais gradual desses valores, realizando a próxima etapa de testes com $p^* = 4$ e 6 e $h^* = 4, 6$ e 8. O parâmetro h continuou sendo testado com seus quatro valores iniciais.

A próxima etapa dos testes foi realizada sobre seis versões de BE2, obtidas a partir da versão 1 pela variação de apenas um dos fatores de 1 a 4. Essas versões são descritas na Tabela 5.5.

Tabela 5.5 - Versões de BE2 obtidas a partir da Versão 1

Versão	Fator 1	Fator 2	Fator 3	Fator 4
2	aresta	simples	atraso	$H^*_t \times P_t$
3	aresta	simples	atraso	{melhor de H^*_t} $\times P^*_t$
4	aresta	simples	atraso	{melhor de H^*_t} $\times P_t$
5	tarefa	simples	atraso	$H^*_t \times P^*_t$
6	aresta	gradual	atraso	$H^*_t \times P^*_t$
7	aresta	simples	aleatória	$H^*_t \times P^*_t$

A essas versões foi aplicado o MCPA, observando-se que todos os nós com $h^* = 8$ foram eliminados logo no primeiro nível da árvore de busca, indicando que esse valor de h^* poderia ser desconsiderado nos testes seguintes. Das versões testadas, a Versão 7 foi a que apresentou o melhor desempenho médio, calculado pela expressão (4.9). Portanto, a próxima etapa consistiu de testes realizados sobre 5 versões de BE2, obtidas a partir da Versão 7, variando-se apenas um dos fatores 1 a 4. Essas versões são apresentadas na Tabela 5.6.

A aplicação do MCPA às versões 8 a 12 permitiu uma nova redução nos valores de teste dos parâmetros. Observou-se que todos os nós com $p^* = 6$ foram eliminados pelo teste de Friedman no primeiro nível da árvore de busca. Portanto, nas etapas seguintes, foram utilizados $p^* = 4, h^* = 4$ e 6 e $h = 10, 20, 30$ e 40.

Tabela 5.6 - Versões de BE2 obtidas a partir da Versão 7

Versão	Fator 1	Fator 2	Fator 3	Fator 4
8	aresta	simples	aleatória	$H^*_t \times P_t$
9	aresta	simples	aleatória	{melhor de H^*_t} $\times P^*_t$
10	aresta	simples	aleatória	{melhor de H^*_t} $\times P_t$
11	aresta	gradual	aleatória	$H^*_t \times P^*_t$
12	tarefa	simples	aleatória	$H^*_t \times P^*_t$

As etapas de teste seguintes consistiram da aplicação do MCPA às 20 demais versões de BE2 e nenhuma redução adicional nos valores dos parâmetros foi possível. Entretanto, observou-se uma grande diminuição do esforço computacional requerido para os testes dessas últimas versões, devida à eliminação de valores realizada nas etapas anteriores.

Para finalizar a Fase 1, todos os nós restantes no segundo nível da árvore de busca do

MCPA de cada versão de BE2 foram comparados aos pares pelo teste de Wilkoxon. Essa comparação mostrou que existem duas versões dominantes, a Versão 9 e a Versão 27. Todos os nós das demais versões foram eliminados pelos melhores nós dessas duas versões. Na Tabela 5.7, são apresentadas as características de cada uma delas e os valores dos parâmetros de seus melhores nós.

Tabela 5.7 - Conjunto de Configurações dominantes de BE2 obtido ao final da fase 1 dos testes computacionais

Versão	Fator 1	Fator 2	Fator 3	Fator 4	h	h*	p*
9	aresta	simples	aleatória	{melhor de H^*_t} x P^*_t	10	4	4
27	tarefa	simples	aleatória	{melhor de H^*_t} x P^*_t	20	4	4

Uma observação pode ser feita com relação à composição do conjunto R_t (Fator 4) nas duas versões apresentadas na Tabela 5.7. Apenas a melhor solução de H^*_t é combinada com todas as soluções de P^*_t. Como o conjunto H^*_t não é totalmente utilizado nas combinações estruturadas, a forte influência do parâmetro h* sobre o desempenho da busca poderia parecer, a princípio, uma contradição. Entretanto, deve-se lembrar que a melhor solução de H^*_t é a melhor solução não tabu de H_t. Como h* é utilizado no cálculo do número máximo de iterações que uma solução de H_t pode permanecer tabu, sua influência sobre o desempenho de BE2 fica plenamente justificada, não entrando em contradição com o fato do conjunto H^*_t não ser totalmente utilizado para compor R_t.

B) Fase 2

O primeiro teste realizado na Fase 2 foi a variação da ordem de consideração das distâncias de designação e seqüenciamento na regra de construção de subconjuntos diversos (Fator 5) para as duas versões de BE2 da Tabela 5.7. Na Fase 1, cada uma dessas versões foi testada considerando-se a distância de seqüenciamento antes da distância de designação. Para a Fase 2, testaram-se as duas outras opções possíveis para o fator 5, criando-se as versões 33 e 34, baseadas na Versão 9, e as versões 35 e 36, baseadas na Versão 27. Cada par de versões alternativas foi comparado pelo teste de Wilkoxon. Na Tabela 5.8, mostram-se os resultados desses testes.

Verificou-se que a consideração da distância de designação antes da distância de seqüenciamento ou a soma das distâncias na regra de construção de subconjuntos diversos gera versões dominantes de BE2. Entretanto, como pode ser visto pela quarta coluna da Tabela 5.8, o valor calculado para o ganho com relação à heurística SAA (expressão 4.9) ainda é muito baixo. Decidiu-se, então, propor algumas modificações nas características de BE2, na tentativa de aumentar o ganho. Cada uma das modificações propostas é aplicada às duas versões dominantes de BE2: versões 33 e 34. Essas modificações são apresentadas a

seguir.

Tabela 5.8 - Resultados dos testes realizados variando-se o Fator 5

Versão	Versão Base	Fator 5	Ganho(%)	Eliminado
9	-	Distância de Seqüenciamento antes da Distância de Designação	-187,7	W(33)
33	9	Distância de Designação antes da Distância de Seqüenciamento	-142,5	
34	9	Distância de Seqüenciamento somada à Distância de Designação	-150,0	
27	-	Distância de Seqüenciamento antes da Distância de Designação	-240,0	W(33)
35	27	Distância de Designação antes da Distância de Seqüenciamento	-325,8	W(33)
36	27	Distância de Seqüenciamento somada à Distância de Designação	-154,6	W(33)

Modificação 1: Mudança na Avaliação dos Votos das Soluções de Referência

A avaliação dada pela expressão (5.5) leva em consideração o atraso de cada tarefa para avaliação dos votos das soluções de referência. Pode-se modificar essa expressão de forma que a avaliação seja feita através das diferenças de posição de cada tarefa nessas soluções. Dado que a combinação estruturada visa passar informação de uma solução de referência para outra, a cada voto $v(1,i,j)$ da solução S_1 pode-se atribuir um valor

$$\Delta pos(1,i) = pos(2,i) - pos(1,i) \qquad (5.6)$$

onde $pos(1,i)$ e $pos(2,i)$ são as posições ocupadas por J_i na máquina que a processa em S_1 e S_2, respectivamente. Cada voto $v(2,i,j)$ recebe um valor $\Delta pos(2,i) = -\Delta pos(1,i)$. Escolhe-se o melhor voto de cada solução $v(1,i^*,j^*)$ e $v(2,i^*,j^*)$ como aquele que apresentar o maior valor de $\Delta pos(1,i)$ e $\Delta pos(2,i)$. Essa escolha é motivada pelo fato de que quanto maior a diferença de posicionamento de uma tarefa entre as duas soluções de referência, maior será o impacto das modificações realizadas pelas trocas e inserções nas combinações estruturadas.

A aplicação dessa modificação às versões base 33 e 34 gerou as versões 37 e 38, respectivamente. Os resultados dos testes realizados sobre essas duas versões mostraram que a regra de avaliação é bastante ineficaz. As versões 37 e 38 foram eliminadas pelas suas versões base através do teste de Wilkoxon.

Modificação 2: Escolha Probabilística do Melhor Voto das Soluções de Referência

Até esse momento, todas as versões em que a escolha do melhor voto de cada solução de referência é feita por critérios determinísticos foram dominadas por versões em que essa escolha é feita de forma aleatória. A modificação 2 foi proposta com o objetivo de investigar se uma escolha probabilística não seria mais vantajosa, pois estabelece um meio termo entre as escolhas determinística e aleatória. Para realizar essa investigação, três critérios de

avaliação de votos foram utilizados. Cada voto recebeu uma probabilidade de seleção baseada em seu valor (determinado pelo critério de avaliação). Essa probabilidade foi então utilizada na escolha do melhor voto, construindo-se uma roleta de probabilidades como descrito na seção 2.2.4.

Critério 1: Avaliação dos votos pela expressão (5.5). Nesse caso a probabilidade de seleção de cada voto v(1,i,j) da solução S_1, por exemplo é dada por

$$p(1, i) = \frac{\Delta(1, i)}{\sum_{k=1}^{n} \Delta(1, k)} \qquad (5.7)$$

A probabilidade de seleção dos votos da solução S_2 são calculados de forma análoga.

Critério 2: Avaliação dos votos pela expressão (5.6). A probabilidade de seleção de cada voto v(1,i,j) da solução S_1 é dada por

$$p(1, i) = \frac{\Delta pos(1, i)}{\sum_{k=1}^{n} \Delta pos(1, k)}. \qquad (5.8)$$

Critério 3: Avaliação dos votos v(1,i,j) da solução S_1 pelo atraso da tarefa J_i na solução S_2 (e vice-versa). Nesse caso a probabilidade de seleção dos votos v(1,i,j) é dada por:

$$p(1, i) = \frac{T(2, i)}{\sum_{k=1}^{n} T(2, k)}. \qquad (5.9)$$

Esse critério foi proposto na tentativa de se analisar o impacto de modificações realizadas nas posições das tarefas mais atrasadas de uma solução de referência, tendo como base as posições sugerida pelos votos da outra solução.

A partir das versões base 33 e 34 geram-se as versões 39 e 40 utilizando o Critério 1, as versões 41 e 42 utilizando o Critério 2 e as versões 43 e 44 utilizando o Critério 3. Novamente, após os testes computacionais com as versões modificadas, verificou-se que todas elas são eliminadas pelas suas versões base através do teste de Wilkoxon.

Após a realização dos testes com as Modificações 1 e 2, pôde-se concluir que a escolha aleatória do melhor voto das soluções de referência pode estar funcionando como um fator de diversificação da busca e, por isso, seus resultados são estatisticamente melhores que os obtidos com critérios de escolha determinísticos e probabilísticos. Tentando melhorar os aspectos de diversificação de BE2 foi proposta, então, a modificação seguinte.

Modificação 3: Escolha Probabilística da Solução de H^*_t a ser Combinada com as Soluções de P^*_t

A opção dominante para o Fator 4 em BE2 consiste de combinar todas as soluções de P^*_t com a melhor solução de H^*_t. A modificação proposta visa incluir um aspecto de

diversificação na escolha da solução de H^*_t da seguinte forma. A cada iteração t, cada solução S_i em H^*_t recebe uma probabilidade $p(S_i)$ calculada como:

$$p(S_i) = \frac{[T_{max} - T(S_i)]^2}{\sum_{S_k \in H^*_t} [T_{max} - T(S_k)]^2} \qquad (5.10)$$

onde $T_{max} = \max\{T(S_k) \mid S_k \in H^*_t\}$. Faz-se, então, uma escolha aleatória baseada nas probabilidades calculadas e a solução escolhida é combinada com as soluções de P^*_t.

A aplicação dessa modificação às versões base 33 e 34 gerou as versões 45 e 46. Essas duas versões modificadas também foram eliminadas pelas suas versões base através do teste de Wilkoxon.

A discussão a seguir introduz a próxima modificação sugerida. Tendo sido fixado o tamanho do conjunto R_t, o tempo gasto para cada iteração t de BE2 pode ser dividido entre os três procedimentos Combina_Soluções, Conjuntos_Intermediários e Atualiza_Conjuntos. O tempo computacional utilizado pelos dois últimos procedimentos não depende da composição de R_t nem da forma como as combinações são realizadas. Portanto, conclui-se que, para se obter um desempenho melhor de BE2 num tempo computacional fixo (para cada valor de n), é preciso garantir não só que as combinações estruturadas estejam sendo realizadas de forma eficiente mas que elas sejam eficazes.

Todos os testes realizados até esse ponto consideram que, dados um par de soluções de referência, ambas as soluções possuem igual importância e devem ser tratadas da mesma forma. Todos os procedimentos aplicados a S_1 tendo S_2 como guia de exploração de sua vizinhança são aplicados a S_2 tendo S_1 como guia. Deve-se observar, entretanto, que a regra de construção de R_t faz com que S_1 e S_2 sejam soluções pertencentes a conjuntos diferentes e que, portanto, tenham sido escolhidas para pertencer a esses conjuntos na iteração anterior por critérios diferentes. Baseando-se nisso pode-se propor uma modificação em BE2 em que seja investigado o efeito de se usar apenas uma das soluções de referência de cada par como guia de exploração da vizinhança da outra, sem repetir o processo trocando o papel das duas soluções. O tipo de combinação proposto pode ser chamado de unidirecional em oposição à combinação utilizada até o momento, que é bidirecional. Essa modificação faz com que o tempo gasto em cada combinação seja reduzido à metade e, como conseqüência, faz com que o número de iterações de BE2 aumente.

Modificação 4: Combinações Estruturadas Unidirecionais

Cada par de soluções $(S_x,S_y) \in R_t$ nas versões 33 e 34 de BE2 é tal que $S_x \in H^*_t$ e $S_y \in P^*_t$ e pela forma como esses conjuntos são construídos, sabe-se que $T(S_x) \leq T(S_y)$. Por um lado, S_x é uma solução de elite e possui um custo baixo mas, por outro, as soluções S_y foram escolhidas para pertencer a P^*_t por suas características estruturais. Deseja-se investigar qual das duas características, custo ou estrutura, é mais importante na escolha da solução guia. Por isso, os testes realizados com as combinações unidirecionais foram divididos em duas

etapas. Na primeira delas, utiliza-se S_x como solução guia na exploração da vizinhança das soluções S_y, gerando as versões 47 e 48 a partir das versões base 33 e 34. Na segunda etapa, utilizam-se as soluções S_y como guias da exploração da vizinhança de S_x, gerando as versões 49 e 50. A Tabela 5.9 apresenta os resultados dos testes.

Tabela 5.9 - Resultados dos testes com combinações estruturadas unidirecionais

Versão	Versão Base	Combinação Estruturada	Ganho(%)	Eliminado
33	-	bidirecional	-142,5	W(49)
34	-	bidirecional	-150,0	W(50)
47	33	$S_x \in H^*_t$ é guia	-769,9	W(33)
48	34	$S_x \in H^*_t$ é guia	-1297,8	W(34)
49	33	$S_y \in P^*_t$ é guia	-82,8	
50	34	$S_y \in P^*_t$ é guia	-86,2	

Observa-se que a utilização de S_x como solução guia gera versões bastante ruins de BE2. A princípio, poderia ser concluído que a solução S_x não possui boas características estruturais para guiar a exploração da vizinhança das soluções S_y. Entretanto, isso não é verdade pois S_x é uma solução de elite e o fato de possuir um custo baixo pode e deve ser atribuído a suas características estruturais. O fato é que essas características estão sendo usadas de forma distribuída pois, para cada solução S_y apenas uma iteração de combinação estruturada está sendo realizada e, conseqüentemente, apenas uma característica estrutural de S_x está sendo transmitida para cada solução S_y. Como o valor de custo de S_y é alto, supõe-se que, em sua vizinhança, existam muitas soluções de custo alto e, portanto, a exploração dessa vizinhança feita de forma tão restrita possui uma probabilidade baixa de encontrar boas soluções.

Por outro lado, quando se utilizam todas as soluções S_y como guias para a exploração da vizinhança de S_x, as características estruturais das soluções de P^*_t que, por definição desse conjunto, são bastante diversificadas, estão sendo utilizadas de forma concentrada. Como S_x é uma solução de elite, supõe-se que, em sua vizinhança, existam outras soluções de custo baixo e, portanto, uma exploração mais intensa dessa vizinhança tem alta probabilidade de encontrar boas soluções alternativas.

Como as versões base 33 e 34 foram eliminadas pelas versões 49 e 50 através do teste de Wilkoxon, essas últimas passaram a ser utilizadas como versões base nos testes seguintes.

Modificação 5: Redefinindo P^*_t

As conclusões tiradas a partir dos testes realizados para a modificação 4, levaram a uma reavaliação do papel das soluções de P^*_t nas combinações estruturadas. A partir das versões 49 e 50, quatro outras versões foram geradas, redefinindo-se o conjunto P^*_t com a finalidade de investigar como a modificação do conjunto de soluções utilizadas pelas

combinações unidirecionais influencia os resultados obtidos. Duas definições alternativas foram propostas:

- P^*_t = {p* soluções de P_t mais distantes entre si} e
- $P^*_t = P_t$.

Na Tabela 5.10, mostram-se os resultados dos testes realizados com P^*_t modificado.

Tabela 5.10 - Resultados dos testes com a modificação 5

Versão	Versão Base	P*t	Ganho(%)	Eliminado
49	-	{p* melhores soluções de Pt}	-82,8	W(51)
50	-	{p* melhores soluções de Pt}	-86,2	W(52)
51	49	{p* soluções de Pt mais distantes entre si}	-90,9	
52	50	{p* soluções de Pt mais distantes entre si}	-84,8	
53	49	igual a Pt	-72,9	
54	50	igual a Pt	-56,0	

Observa-se um aumento no valor de Ganho quando se usa $P^*_t = P_t$. A aplicação do teste de Wilkoxon entre todos os pares de versões da Tabela 5.9 indicou que as versões 51, 52, 53 e 54 são estatisticamente iguais. Entretanto, devido ao maior valor de Ganho apresentado pelas versões 53 e 54 com relação às demais elas foram utilizadas como versões base nos próximos testes.

Modificação 6: Acrescentando um Par de Soluções a R_t

Na verdade, essa modificação consiste do acréscimo de um par de soluções (S_x,S_y) a R_t onde S_x é a melhor solução de H^*_t e S_y é uma solução gerada aleatoriamente. Isso é feito a cada 5 iterações da busca para que se possa investigar o efeito da introdução de aleatoriedade na exploração da vizinhança de S_x. A partir das versões base 53 e 54 foram geradas as versões modificadas 55 e 56. Após a realização dos testes, as versões modificadas foram eliminadas pelas suas versões base através do teste de Wilkoxon.

Modificação 7: Busca Aleatória na Atualização da Melhor Solução Encontrada

A cada iteração de BE2, verifica-se se foi encontrada uma solução melhor que a incumbente **S***. Em caso positivo, **S*** é atualizada e passa-se para a próxima iteração. A modificação proposta consiste em se considerar soluções tentativas obtidas pela realização de uma busca aleatória na vizinhança de **S*** a cada vez que essa solução é atualizada. A busca é feita através de algumas iterações em que se escolhe aleatoriamente entre a realização de uma troca ou de uma inserção de tarefas. No caso de troca, duas tarefas são escolhidas aleatoriamente e suas posições são trocadas e, no caso de inserção, escolhem-se aleatoriamente uma tarefa e uma nova posição para ela. Essa busca é bem parecida com o operador de mutação utilizado em GEN3 e GEN4. A aplicação dessa modificação sobre as versões base 53 e 54 leva às versões 57 e 58, que são eliminadas através do teste de Wilkoxon.

Após a realização dos testes com as modificações 6 e 7, conclui-se que a diversificação introduzida pelos elementos aleatórios propostos é muito forte e prejudica o desempenho da busca. Isso não ocorre quando a diversificação é feita de forma direcionada pela utilização de soluções aceitas pelas regras de construção de subconjuntos diversos.

C) Variações de Profundidade nas Combinações de Soluções

Ao final da fase 2 dos testes computacionais, escolheu-se a versão 54 como sendo a versão dominante de BE2 até o momento. Entretanto, as combinações adaptativas nessa versão estão sendo realizadas com profundidade igual a 1. Para investigar como o desempenho de BE2 varia com a profundidade das combinações, foi realizada uma série de testes utilizando-se dois problemas de cada tipo para cada par de valores (m,n). Para cada conjunto de 6 problemas com o mesmo número de máquinas e tarefas, a versão 54 de BE2 foi aplicada variando-se a profundidade das combinações adaptativas entre 1 e 6.

Tabela 5.11 - Valores de Ganho para BE2 variando-se a profundidade das combinações adaptativas

n	Prof	m = 1	m = 2	m = 3	m = 4	m = 5	m = 6	Média
30	1	-57,5	16,4	8,1	12,7	5,0	8,8	-1,1
	2	-77,3	21,0	9,8	5,9	4,3	6,9	-4,9
	3	-65,8	-1,0	2,8	8,2	5,1	9,1	-6,9
	4	-79,2	11,3	-5,2	9,4	4,8	5,0	-9,0
	5	-67,9	-14,8	-8,7	3,0	-3,1	3,9	-14,6
	6	-128,8	-41,5	5,1	7,1	4,1	5,6	-24,7
50	1	-627,3	-15,3	-7,9	-7,3	1,8	0,5	-109,3
	2	-667,2	-39,3	-21,4	-17,7	-3,0	-3,0	-125,3
	3	-779,0	-68,9	-32,2	-29,4	-3,4	-4,1	-152,8
	4	-934,5	-62,7	-38,4	-39,5	-8,2	-11,5	-182,5
	5	-552,0	-64,9	-56,9	-71,3	-7,3	-11,4	-127,3
	6	-1101,2	-72,7	-62,7	-74,5	-17,6	-13,4	-223,7
70	1	-730,6	-173,9	-35,8	-25,0	-10,6	-12,8	-164,8
	2	-835,0	-229,3	-45,1	-35,6	-24,6	-19,6	-198,2
	3	-903,1	-327,3	-59,3	-39,1	-26,2	-28,0	-230,5
	4	-852,9	-355,4	-70,5	-49,2	-37,8	-36,5	-233,7
	5	-954,0	-388,0	-77,6	-55,3	-47,4	-37,7	-260,0
	6	-1037,3	-457,4	-78,9	-60,0	-50,0	-36,8	-286,7
90	1	-1563,5	-1390,5	-228,2	-72,8	-75,5	-218,1	-591,4
	2	-1614,1	-1542,2	-245,9	-93,9	-96,8	-297,0	-648,4
	3	-1578,3	-1635,8	-292,4	-118,9	-128,2	-446,6	-700,0
	4	-1684,2	-1704,6	-314,4	-124,6	-121,9	-465,4	-735,9
	5	-1772,9	-1815,6	-321,2	-132,0	-141,1	-698,0	-813,5
	6	-1825,0	-2006,2	-362,8	-146,5	-148,9	-660,8	-858,4
110	1	-9342,4	-246,3	-173,1	-530,0	-196,9	-1334,0	-1970,5
	2	-10067,9	-270,0	-193,8	-639,4	-239,6	-1836,1	-2207,8
	3	-10481,6	-278,3	-203,8	-668,5	-252,3	-1901,6	-2297,7
	4	-10500,6	-292,5	-213,8	-679,4	-267,7	-2339,0	-2382,2
	5	-9540,7	-278,4	-223,0	-702,0	-269,5	-2248,3	-2210,3
	6	-10704,9	-295,6	-229,0	-724,5	-256,4	-2335,3	-2424,3

O tempo computacional máximo para cada valor de n foi estimado como o tempo total

que GEN1 gastaria para realizar 3000 iterações para os problemas com o mesmo número de tarefas. Os tempos máximos por valor de n são: 100s para n = 30; 200s para n = 50, 320s para n = 70, 520s para n = 90 e 550s para n = 110. Para cada profundidade, o valor médio percentual de Ganho foi calculado pela expressão (4.9). Os valores obtidos são mostrados na Tabela 5.11. Nessa tabela, as células marcadas correspondem ao melhor valor de ganho percentual por (m,n) e à melhor média calculada sobre todos os valores de m, para cada n.

Para se escolher a melhor profundidade para cada valor de n, pode-se utilizar o critério da melhor média de Ganho (9ª coluna da Tabela 5.11). Entretanto, os valores de Ganho ainda são muito baixos e decidiu-se realizar novos testes dobrando o tempo computacional máximo para cada valor de n. Na Tabela 5.12 são apresentados os resultados obtidos.

Tabela 5.12 - Valores de Ganho para BE2 variando-se a profundidade das combinações adaptativas (tempo computacional dobrado)

n	Prof	m = 1	m = 2	m = 3	m = 4	m = 5	m = 6	Média
30	1	-26,6	17,0	9,4	14,0	8,2	9,0	5,2
	2	-56,1	22,7	13,7	8,1	5,5	9,0	0,5
	3	-42,8	7,5	6,2	11,6	6,2	10,0	-0,2
	4	-19,3	13,4	0,4	11,1	6,1	6,9	3,1
	5	-37,0	-13,1	5,9	8,5	3,0	5,6	-4,5
	6	-80,3	0,9	11,3	8,6	7,1	7,4	-7,5
50	1	-307,8	3,7	1,7	6,7	5,5	4,6	-47,6
	2	-246,6	-15,8	0,8	-5,2	2,1	2,8	-43,7
	3	-317,6	-25,3	-11,8	-10,7	6,0	2,5	-59,5
	4	-282,3	-22,5	-8,4	-15,3	1,7	-1,5	-54,7
	5	-271,5	-26,9	-11,2	-12,8	0,8	-2,4	-54,0
	6	-497,8	-34,7	-21,3	-25,7	-1,5	-5,2	-97,7
70	1	-473,0	-75,0	-10,0	-5,9	-1,6	-7,0	-95,4
	2	-472,1	-99,0	-19,0	-13,0	-5,0	-7,7	-102,6
	3	-693,1	-172,8	-28,6	-20,9	-8,1	-10,5	-155,7
	4	-661,9	-199,2	-37,5	-22,9	-12,5	-16,8	-158,5
	5	-624,6	-253,7	-37,4	-31,5	-19,5	-19,5	-164,4
	6	-687,9	-287,6	-45,7	-30,4	-16,6	-18,7	-181,2
90	1	-1054,0	-672,3	-91,9	-36,6	-20,5	-58,4	-322,3
	2	-1085,5	-685,7	-127,0	-45,1	-40,9	-120,1	-350,7
	3	-1238,6	-955,9	-171,1	-61,2	-50,8	-168,4	-441,0
	4	-1337,3	-1004,2	-186,7	-70,3	-53,3	-177,4	-471,5
	5	-1348,9	-1135,1	-214,0	-69,0	-74,1	-284,2	-520,9
	6	-1586,4	-1370,7	-227,6	-83,6	-82,1	-242,0	-598,7
110	1	-7704,5	-161,5	-94,4	-259,1	-91,9	-386,0	-1449.6
	2	-8827,4	-186,0	-120,3	-316,6	-118,4	-639,6	-1701.4
	3	-9514,5	-201,9	-129,0	-404,6	-140,8	-846,7	-1872.9
	4	-9017,7	-222,7	-140,6	-400,6	-149,1	-871,8	-1800.4
	5	-8297,9	-215,2	-155,6	-420,4	-174,1	-900,0	-1693.9
	6	-8882,5	-234,4	-161,9	-457,5	-162,0	-1096,3	-1832.4

Infelizmente, BE2 não foi capaz de gerar boas soluções para o PST. Note que os valores médio de Ganho para cada n são bastante baixos, mesmo para n = 30. Os resultados da Tabela 5.12 mostram que as mudanças realizadas em BE1 para obter BE2

foram significativas pois o valor de Ganho foi aumentado. Entretanto, a melhoria obtida não foi satisfatória.

Analisando em detalhes os resultados obtidos, verificou-se que o cálculo das distâncias de designação consome muito tempo computacional, fazendo com que o número de iterações realizadas por BE2 dentro do tempo máximo permitido ainda seja pequeno, mesmo com as modificações adotadas no procedimento de combinações estruturadas. Concluiu-se que o ganho obtido com a redefinição do conjunto R_t e do procedimento Combina_Soluções foi anulado pela introdução de uma medida de distâncias que requer a construção de dois grafos de designação para cada par de soluções. Por esse motivo, essa implementação não foi considerada adequada para ser aplicada a todo o conjunto de problemas de teste.

O próximo passo desse trabalho consistiu, então, do desenvolvimento de uma terceira implementação de Busca por Espalhamento. Nessa nova implementação, tentou-se reunir algumas boas características de BE1 e BE2 evitando introduzir procedimentos que consumissem muito tempo computacional. No capítulo seguinte, descreve-se a implementação de BE3.

CAPÍTULO 6 - BUSCA POR ESPALHAMENTO IMPLEMENTAÇÃO FINAL

Neste capítulo apresenta-se a última implementação de Busca por Espalhamento desenvolvida neste trabalho, chamada BE3. A partir dos resultados obtidos e do estudo realizado nos dois últimos capítulos, procurou-se reunir nessa implementação as características consideradas interessantes das implementações anteriores, BE1 e BE2, procurando dar maior eficiência e qualidade à busca realizada. Nessa última implementação também foi estudado o efeito da aplicação de busca local a algumas soluções exploradas pela Busca por Espalhamento. Nas seções seguintes, descrevem-se as características de BE3, assim como os testes computacionais realizados sobre ela.

6.1 - INTRODUÇÃO

A implementação BE3 possui duas características principais. A primeira delas consiste na métrica utilizada no cálculo de distâncias entre soluções. Por questões de velocidade, decidiu-se utilizar a distância calculada pelo número de arcos distintos entre duas soluções, como foi feito em BE1. A partir dessa métrica de distâncias, pode-se utilizar o procedimento de construção de subconjuntos diversos descrito na seção 4.2.1. Por outro lado, BE3 é caracterizada por realizar as combinações estruturadas da mesma forma que BE2. Ou seja, as soluções a combinar são armazenadas em pares no conjunto R_t e para cada par de soluções não se realizam todas as combinações até que se chegue a um centro de influência. As combinações são realizadas até uma certa profundidade que deve ser determinada de acordo com a dimensão dos problemas a resolver.

A seguir, cada um dos componentes da Busca por Espalhamento presentes em BE3 são descritos mais detalhadamente. Quando não houver nenhuma diferença entre um procedimento utilizado por BE3 e o procedimento correspondente em BE1 ou BE2 isso é indicado, juntamente com a seção onde se encontra a descrição detalhada desse procedimento.

6.2 - CONJUNTOS DE SOLUÇÕES

Em BE3 são utilizados os mesmos conjuntos de soluções descritos para BE1 e BE2. Em particular, define-se R_t como o conjunto de todos os pares de soluções de referência a combinar, da mesma forma que em BE2.

O controle de repetição de soluções e a construção de subconjuntos diversos são feitos exatamente da maneira descrita para BE1, assim como o procedimento de construção da população inicial (conjuntos P_0 e H_0).

6.3- COMBINAÇÕES ESTRUTURADAS E CONJUNTOS INTERMEDIÁRIOS

As combinações de soluções em BE3 são realizadas pelo procedimento análogo ao apresentado na seção 4.2.4. Entretanto, pelo fato de se considerar que as combinações são realizadas até uma determinada profundidade e não até que se encontre o centro de influência, o procedimento é reapresentado a seguir. O procedimento Conjuntos_Intermediários possui a mesma forma apresentada em 4.2.4.

- **Procedimento Combina_Soluções (S_1, S_2, profundidade)**

Variáveis: prof = contador de iterações (profundidade);
S_{tent} = solução tentativa
(S_{tent*},i) = (melhor solução tentativa encontrada, solução de referência a ser substituída por S_{tent*} na atualização)
v(k,i*,j*) = melhor voto da solução S_k;
v*(k*) = melhor decisão a cada iteração, corresponde a v(k*,i*,j*).

Passo 1: Construa uma tabela com os votos de cada solução e seus respectivos valores;
Passo 2: prof = 1; $T(S_{tent*}) = \infty$;
Passo 3: Se prof > profundidade, ou se foi encontrado o centro de influência das soluções de referência iniciais, Pare.
Passo 4: Escolha v(1,i*,j*);
Passo 5: S_{tent} = solução obtida pela inserção de j* imediatamente após i* em S_2;
Se S_{tent} é uma solução válida:
Conjuntos_Intermediários(h,p,S_{tent});
Se $T(S_{tent}) < T(S_{tent*})$: ($S_{tent*}$,i) = ($S_{tent}$,2);
S_{tent} = solução obtida pela inserção de i* imediatamente antes de j* em S_2;
Se S_{tent} é uma solução válida:
Conjuntos_Intermediários(h,p,S_{tent});
Se $T(S_{tent}) < T(S_{tent*})$: ($S_{tent*}$,i) = ($S_{tent}$,2);
Passo 6: Escolha v(2,i*,j*);
Passo 7: S_{tent} = solução obtida pela inserção de j* imediatamente após i* em S_1;
Se S_{tent} é uma solução válida:
Conjuntos_Intermediários(h,p,S_{tent});
Se $T(S_{tent}) < T(S_{tent*})$: ($S_{tent*}$,i) = ($S_{tent}$,1);

```
                S_tent = solução obtida pela inserção de i* imediatamente
                         antes de j* em S_1;
                Se S_tent é uma solução válida:
                      Conjuntos_Intermediários(h,p,S_tent);
                      Se T(S_tent) < T(S_melhor): (S_tent*,i) = (S_tent,1);
Passo 8: Dado (S_tent*,i) faça S_i = S_tent*;
         Atualize a tabela de votos das soluções S_1 e S_2;
         Faça prof = prof + 1, e volte ao passo 3;
```

A partir dos conjuntos intermediários P'_t e H'_t criados, atualizam-se P_t e H_t como realizado em BE1 (seção 4.2.5).

Para que a implementação de BE3 fique completamente definida, é preciso decidir qual a regra de atribuição de valor aos votos de cada solução nas combinações adaptativas e a regra de escolha do melhor voto. Decidiu-se utilizar a mesma regra de BE1, ou seja, o valor de cada voto v(k,i,j) é igual a:

$$\beta.T_j + (1 - \beta).D_{ij}, \text{ com } 0 \leq \beta \leq 1 \qquad (6.1)$$

onde D_{ij} é o comprimento do arco (i,j).

Entretanto, decidiu-se que, em BE3, não podem ser escolhidos arcos (i,j) tais que j é um nó de máquina. Sendo assim, a regra de escolha do melhor voto é: "O melhor voto v(k,i*,j*), j* > 0,da solução S_k corresponde ao menor valor de $\beta.T_j + (1 - \beta).D_{ij}$."

6.4 - TESTES COMPUTACIONAIS

6.4.1 - Planejamento

Como foi feito para BE2, realizaram-se testes computacionais com o objetivo de determinar a melhor configuração de BE3 com relação às configurações possíveis e de ajustar os melhores valores para os parâmetros h, h* e p*, mantendo-se p = 2.p*.

Separam-se ,a seguir, os fatores determinantes de desempenho de BE3:

Fator 1: Ajuste da Regra de Escolha do Melhor Voto das Soluções de Referência nas Combinações Adaptativas

Arco de menor distância: $\beta = 0$ em (6.1)

Arco de menor atraso: $\beta = 1$ em (6.1)

Escolha aleatória

Fator 2: Soluções a Combinar

$R_t = H^*_t \times P^*_t$

$R_t = H^*_t \times P_t$

$R_t =$ {melhor solução de H^*_t} $\times P^*_t$

$R_t =$ {melhor solução de H^*_t} $\times P_t$

Os valores de teste dos parâmetros de BE3 são.

- $h^* = 4, 6, 8, 10;$
- $p^* = 4, 6, 8, 10;$
- $h = 10, 20, 30, 40;$
- $q_{max} = 200.$

Para determinar a melhor versão de BE3 baseada nos fatores 1 e 2 e nos valores de teste dos parâmetros, os testes foram divididos em duas fases, como para BE2.

Fase 1 : Eliminação de Configurações

A cada versão de BE3, obtida pela combinação das opções possíveis para os fatores 1 e 2, aplicou-se o MCPA para determinar os melhores valores dos parâmetros h, h* e p*. Há apenas 12 versões possíveis, mas os testes foram realizados em etapas, como para BE2, visando a eliminação dos valores de teste menos eficazes para esses parâmetros e a conseqüente diminuição do esforço computacional necessário.

Ao final da Fase 1, para cada versão testada, os nós restantes no último nível da árvore de busca do MCPA foram comparados aos pares pelo teste de Wilkoxon para obtenção de um conjunto reduzido de configurações dominantes de BE3 com relação aos fatores 1 e 2 e aos parâmetros da busca.

Fase 2: Refinamento

Foram realizados na Fase 2 os testes adicionais necessários à investigação de modificações no esquema proposto para BE3, análogas às que foram sugeridas para BE2. Ao final dos testes da Fase 2, todas as configurações restantes puderam ser consideradas estatisticamente iguais. Nesse ponto, então, escolheu-se a configuração que apresentou o melhor valor de Ganho, calculado de acordo com a expressão (4.9).

Para BE3, também combinam-se os pares de soluções de referência com profundidade igual a 1, ou seja, o procedimento de combinação de soluções foi chamado como Combina_Soluções(S_1, S_2, 1). A investigação do efeito do uso de profundidades maiores nesse procedimento foi feita para cada par de valores de (m,n), após os testes da Fase 2.

6.4.2 - Resultados

A) Fase 1

Os testes realizados na Fase 1 da determinação da melhor configuração de BE3 iniciaram-se pela determinação da versão básica da busca, denominada Versão 1. Essa versão possui as seguintes características:

Fator 1: Escolha do menor arco ($\beta = 0$)

Fator 2: $R_t = H^*_t \times P^*_t$.

Aplicou-se o MCPA a essa versão, ramificando-se 16 nós no primeiro nível da árvore de busca, cada nó correspondendo a uma das combinações dos valores de teste de p* e h*. No primeiro nível, h foi mantido constante e igual a 30. Os nós não eliminados no primeiro nível, após a aplicação do teste de Wilkoxon são mostrados na Tabela 6.1.

Cada um desses nós foi ramificado no segundo nível da árvore de busca, variando-se o valor de h. A tabela 6.2 mostra os nós do segundo nível.

Observa-se, pela Tabela 6.2, que os valores mais baixos de p* geraram resultados mais pobres para BE3 pois todos os nós com p* = 4 e 6 foram eliminados no primeiro nível da árvore de busca. Todos os nós com h = 10 foram eliminados no segundo nível. Somente o menor valor de p* e de h foi eliminado dos testes seguintes. O parâmetro h* continuou sendo testado com seus quatro valores iniciais.

Tabela 6.1 - Nós restantes no primeiro nível do MCPA para a Versão 1 de BE3

Nó	h*	p*
3	4	8
4	4	10
7	6	8
8	6	10
11	8	8
12	8	10
15	10	8

Tabela 6.2 - Último nível do MCPA para a Versão 1 de BE3

Nó	h	h*	p*	Eliminado
17	10			F
18	20	4	8	W(24)
19	30			W(24)
20	40			W(24)
21	10			F
22	20	4	10	
23	30			
24	40			
25	10			F
26	20	6	8	W(22)
27	30			W(24)
28	40			F
29	10			F
30	20	6	10	
31	30			
32	40			
33	10			W(22)
34	20	8	8	W(22)
35	30			W(24)
36	40			W(22)
37	10			F
38	20	8	10	F
39	30			F
40	40			
41	10			F
42	20	10	8	W(22)
43	30			
44	40			F

A etapa seguinte dos testes foi realizada sobre cinco versões de BE3, obtidas a partir da Versão 1 pela variação de apenas um dos fatores de 1 e 2. Essas versões são descritas na

Tabela 6.3.

A essas versões foi aplicado o MCPA, observando-se que nenhum dos valores dos parâmetros p*, h* e h foi consistentemente eliminado na aplicação dos testes estatísticos. Portanto, decidiu-se realizar os testes com as versões restantes utilizando-se o conjunto atual de valores. Na Tabela 6.4, apresentam-se as versões derivadas da versão 5 e 6, variando-se apenas um dos fatores 1 e 2.

Tabela 6.3 - Versões de BE3 obtidas a partir da Versão 1

Versão	Fator 1	Fator 2
2	menor arco	$H^*_t \times P_t$
3	menor arco	{melhor de H^*_t} $\times P^*_t$
4	menor arco	{melhor de H^*_t} $\times P_t$
5	menor atraso	$H^*_t \times P^*_t$
6	aleatória	$H^*_t \times P^*_t$

Tabela 6.4 - Versões de BE3 obtidas a partir da Versão 5 e da Versão 6

Versão	Versão Base	Fator 1	Fator 2
7	5	menor atraso	$H^*_t \times P_t$
8	5	menor atraso	{melhor de H^*_t} $\times P^*_t$
9	5	menor atraso	{melhor de H^*_t} $\times P_t$
10	6	aleatória	$H^*_t \times P_t$
11	6	aleatória	{melhor de H^*_t} $\times P^*_t$
12	6	aleatória	{melhor de H^*_t} $\times P_t$

Para finalizar a Fase 1, todos os nós não eliminados no segundo nível da árvore de busca do MCPA de cada versão de BE3 foram comparados aos pares pelo teste de Wilkoxon. Essa comparação mostrou que existem duas versões dominantes, a Versão 6 e a Versão 10. Todos os nós das demais versões foram eliminados pelos melhores nós dessas duas versões. Na Tabela 6.5, são apresentadas as características de cada uma delas e os valores dos parâmetros de seus melhores nós.

Tabela 6.5 - Conjunto de Configurações dominantes de BE3 obtido ao final da fase 1 dos testes computacionais

Versão	Fator 1	Fator 2	h	h*	p*	Ganho(%)
6	aleatória	$H^*_t \times P^*_t$	30	10	10	-20.4
10	aleatória	$H^*_t \times P_t$	30	8	10	0.6

B) Fase 2

Como a Versão 10 apresenta o maior valor de Ganho, é utilizada como base para as versões derivadas das modificações em BE3 propostas a seguir.

Modificação 1: Avaliação dos Votos das Soluções de Referência

Até o momento, a melhor versão de BE3 realiza a escolha de votos das soluções de

referência de forma aleatória. As versões alternativas, nas quais a avaliação dos votos atribui maiores valores aos menores arcos ($\beta = 0$) ou aos arcos de menor atraso ($\beta = 1$) foram eliminadas. A modificação proposta corresponde à investigação do comportamento de BE3 quando o parâmetro β é variado na faixa [0,1] em intervalos de 0,1. Para cada valor de β, obtém-se uma versão alternativa de BE3. As nove versões são indexadas de 13 a 21 e são todas eliminadas pela Versão 10 através do teste de Wilkoxon.

Modificação 2: Escolha Probabilística do Melhor Voto das Soluções de Referência

Uma forma alternativa de escolher o melhor voto nas soluções de referência é a escolha probabilística. Para cada arco (i,j), $j > 0$, da solução S_1, atribui-se uma probabilidade igual a:

$$p(i,j) = \frac{[V_{max} - V_{ij}]^2}{\sum_{(i,j)\in S_1} [V_{max} - V_{ij}]^2}. \qquad (6.2)$$

onde $V_{ij} = \beta.T_j + (1 - \beta).D_{ij}$ e $V_{max} = \max\{V_{ij} \mid (i,j) \in S_1\}$. As probabilidades relacionadas aos votos da solução S_2 são calculadas de forma análoga. Foram testadas 11 versões de BE3 utilizando a escolha probabilística do melhor voto, uma para cada valor de β na faixa [0,1], variando em intervalos de 0,1. Todas essas versões, indexadas de 22 a 32 foram eliminadas pela Versão 10 após a aplicação do teste de Wilkoxon.

Concluiu-se, após os testes com as modificações 1 e 2, que a escolha aleatória de tarefas é um elemento de diversificação de BE3 e pode ser um dos fatores responsáveis pela qualidade de suas soluções.

Modificação 3: Combinações estruturadas unidirecionais

Na seção 5.4.2B foram descritas as combinações estruturadas unidirecionais (Modificação 4 para BE2). Decidiu-se realizar testes com esse tipo de combinação também para BE3. Aqui, duas versões alternativas da busca são definidas a partir da Versão 10 utilizando as soluções de H^*_t como guia (Versão 33) e utilizando as soluções de P_t como guia (Versão 34). Essas duas versões foram eliminadas pela Versão 10 através do teste de Wilkoxon.

Modificação 4: Redefinindo P^*_t

A modificação proposta consiste da redefinição do conjunto P^*_t de forma que ele contenha as p^* soluções de P_t mais distantes entre si. A partir dessa modificação, define-se a Versão 35 de BE3, igual à Versão 10, com exceção da composição do conjunto R_t que passa a ser definido como: $H^*_t \times P^*_t$. Após a realização dos testes, a Versão 35 foi eliminada pela Versão 10.

Modificação 5: Acrescentando Pares de Soluções a R_t

Da mesma forma como foi proposto na Modificação 6 para BE2, propõe-se uma versão de BE3 onde sejam acrescentados h^* pares de soluções (S_x,S_y) a R_t onde $S_x \in H^*_t$ e

S_y é uma solução gerada aleatoriamente. Isso é feito a cada 5 iterações da busca para que se possa investigar o efeito da introdução de aleatoriedade na construção de P_t. A partir da versão 10 foi gerada a versão modificada 36. Após a realização dos testes, essa versão modificada foi eliminada pela versão 10 através do teste de Wilkoxon.

Modificação 6: Busca Aleatória na Atualização da Melhor Solução Encontrada

A cada iteração de BE3, se a solução incumbente **S*** é atualizada, consideram-se soluções tentativas obtidas pela realização de uma busca aleatória na vizinhança de **S***. A descrição dessa busca aleatória foi feita na apresentação da Modificação 7 para BE2. A aplicação dessa modificação sobre a versão 10 leva à versão 37, que não é eliminada pela versão 10 através do teste de Wilkoxon, mas apresenta Ganho = -9,984%. Apesar das versões 10 e 37 serem estatisticamente iguais, escolheu-se como versão dominante de BE3 para os próximos testes aquela que apresenta maior valor de ganho, a Versão 10.

C) Variações de Profundidade nas Combinações de Soluções

Nesta subseção, analisa-se o comportamento de BE3 com relação à variação dos níveis de profundidade nas combinações de soluções. Os testes foram realizados de forma análoga ao descrito para BE2 na seção 5.4.2.C, mas a profundidade foi variada entre 1 e 10. Na Tabela 6.6, mostram-se os valores de Ganho para cada par de valores (m,n). As células marcadas correspondem ao melhor valor de ganho percentual por (m,n) e à melhor média calculada sobre todos os valores de m, para cada n.

Observa-se, pela Tabela 6.6 que os valores de Ganho de BE3 são bem melhores que os de BE2. Apesar disso, realizaram-se também os testes dobrando o tempo computacional máximo de BE3. Os testes com o tempo dobrado tiveram como objetivo avaliar a melhoria no ganho de BE3 para os valores maiores de n, visto que, pela 9ª coluna da Tabela 6.6, o desempenho de BE3 cai quando se aumenta o número de tarefas. Os resultados desses testes são mostrados na Tabela 6.7.

Os resultados dessa tabela mostram que BE3 supera BE2 com respeito ao Ganho em relação às heurísticas de comparação, indicando que o esforço realizado no sentido de acelerar a busca fez com que ela encontrasse soluções melhores. Pode-se avaliar o aumento de velocidade obtido em BE3, analisando-se o número de iterações realizadas. O número total de iterações de BE2 fica em torno de 45% do número total de iterações de BE3, na média, para o mesmo tempo computacional, indicando que essa última é realmente mais rápida. Quanto à porcentagem de iterações efetivamente utilizadas pela busca para encontrar sua solução final para cada problema, calculou-se uma média de 95% para BE2 e 71% para BE3. Esses valores mostram que, quando o tempo computacional permitido se esgota, BE2 ainda não convergiu, ou seja, a busca pára em um ponto onde ainda seria capaz de melhorar sua solução de saída sem ter tempo suficiente para tal. Já para BE3, nota-se que a busca converge mais rapidamente, parando o processo de melhoria de sua solução de saída antes que o tempo

computacional máximo se esgote.

Tabela 6 . 6 - Valores de Ganho para BE3 variando-se a profundidade das combinações adaptativas

n	Prof	m = 1	m = 2	m = 3	m = 4	m = 5	m = 6	Média
30	1	12,9	0,6	8,8	7,1	1,7	3,0	5,7
	2	15,6	16,5	11,8	10,2	4,3	2,4	10,1
	3	16,8	19,3	12,8	6,5	1,1	6,0	10,4
	4	16,0	24,8	11,5	4,6	4,8	3,7	10,9
	5	15,1	22,8	8,3	6,4	-2,3	1,7	8,7
	6	16,1	17,1	9,9	3,5	2,1	1,1	8,3
	7	14,3	20,8	12,5	8,8	0,8	0,6	9,6
	8	14,6	18,4	11,7	9,5	0,9	1,1	9,4
	9	11,2	16,0	10,8	5,8	-3,0	-0,4	6,7
	10	14,4	17,9	9,3	4,9	-4,3	2,7	7,5
50	1	12,0	20,8	4,9	-11,3	1,5	-2,3	4,3
	2	18,2	23,4	10,7	6,6	6,5	1,3	11,1
	3	16,7	21,5	18,0	0,2	2,9	0,7	10,0
	4	18,3	23,3	9,1	1,7	5,2	-2,4	9,2
	5	16,9	20,5	5,6	-9,4	-2,4	-0,9	5,1
	6	14,7	19,2	7,8	-14,2	2,8	-7,9	3,7
	7	13,8	18,0	3,6	-12,1	-0,7	-7,3	2,6
	8	12,3	16,0	2,9	-10,8	-1,4	-9,8	1,5
	9	8,7	13,7	1,1	-33,2	-2,8	-9,2	-3,6
	10	8,1	13,8	-7,1	-28,5	-4,7	-15,5	-5,7
70	1	9,7	21,9	11,6	2,6	1,8	-4,0	7,3
	2	8,3	21,9	10,8	4,0	5,5	-10,2	6,7
	3	15,6	23,9	12,3	0,2	0,3	-1,9	8,4
	4	13,2	19,3	6,1	2,5	0,8	-8,0	5,7
	5	7,0	15,9	2,1	-1,7	-0,5	-8,1	2,5
	6	-4,3	9,0	2,3	-3,8	-12,8	-14,5	-4,0
	7	-2,6	2,3	-2,7	-7,4	-8,0	-16,3	-5,8
	8	-13,2	-9,4	-2,7	-12,8	-12,6	-17,0	-11,3
	9	-10,0	-18,6	-8,5	-13,3	-16,3	-22,1	-14,8
	10	-18,2	-20,3	-9,6	-17,7	-17,7	-27,6	-18,5
90	1	10,7	14,1	-6,0	-0,5	-6,9	-50,0	-6,4
	2	15,0	20,3	1,4	0,6	-12,4	-48,0	-3,9
	3	13,2	14,7	-1,5	-5,7	-12,8	-56,4	-8,1
	4	12,4	1,6	-4,3	-2,1	-11,8	-71,9	-12,7
	5	3,4	-14,0	-9,4	-8,1	-16,8	-158,4	-33,9
	6	-0,6	-17,4	-11,9	-16,5	-19,2	-126,4	-32,0
	7	-8,5	-49,9	-26,1	-18,5	-28,5	-151,7	-47,2
	8	-24,2	-41,3	-35,9	-25,0	-37,6	-182,9	-57,8
	9	-42,1	-90,4	-42,5	-29,4	-47,8	-191,3	-73,9
	10	-63,3	-149,6	-49,8	-35,9	-47,7	-246,8	-98,9
110	1	16,7	5,0	2,1	-9,5	-8,1	-98,9	-15,5
	2	13,0	8,3	2,2	-8,5	-11,5	-206,0	-33,8
	3	1,8	-2,6	-4,4	-20,6	-16,0	-219,4	-43,5
	4	-28,5	-9,1	-10,4	-35,0	-30,3	-213,6	-54,5
	5	-46,6	-12,9	-14,4	-85,2	-37,5	-213,0	-68,3
	6	-131,0	-26,3	-24,2	-83,3	-48,8	-383,1	-116,1
	7	-323,0	-32,6	-28,6	-99,6	-56,8	-436,3	-162,8
	8	-341,3	-39,0	-40,4	-121,6	-65,5	-487,2	-182,5
	9	-413,4	-44,1	-43,0	-166,2	-76,5	-619,0	-227,0
	10	-736,2	-54,4	-48,7	-155,4	-81,9	-672,9	-291,6

Tabela 6 . 7 - Valores de Ganho para BE3 variando-se a profundidade das combinações adaptativas (tempo computacional dobrado)

n	Prof	m = 1	m = 2	m = 3	m = 4	m = 5	m = 6	Média
30	1	12,9	0,7	9,4	7,5	2,0	4,0	6,1
	2	15,7	17,1	13,9	11,1	5,0	4,7	11,3
	3	16,8	19,9	13,2	8,8	4,2	7,0	11,7
	4	17,0	25,6	14,1	10,1	7,2	6,1	13,4
	5	15,9	23,1	14,8	9,8	-0,5	5,4	11,4
	6	16,7	17,9	12,7	6,1	7,2	4,0	10,8
	7	15,4	21,2	16,8	10,7	3,0	2,8	11,7
	8	14,9	18,7	16,2	12,3	4,7	4,4	11,9
	9	12,5	16,7	17,4	8,8	4,9	4,0	10,7
	10	16,0	19,0	17,3	10,8	3,6	6,1	12,1
50	1	12,0	21,0	4,9	-10,7	2,0	-2,3	4,5
	2	18,2	23,9	12,0	7,2	7,8	1,8	11,8
	3	17,4	22,1	20,6	3,2	5,6	1,8	11,9
	4	19,1	24,7	11,6	3,4	8,1	1,8	11,5
	5	19,6	23,6	14,4	4,8	3,0	3,7	11,5
	6	17,4	22,8	17,9	-0,3	10,0	0,3	11,4
	7	18,9	24,4	17,6	-1,6	4,6	-0,1	10,6
	8	19,3	23,2	17,4	-2,3	5,9	-0,4	10,5
	9	19,2	22,8	14,4	-7,3	6,9	-0,3	9,3
	10	18,5	23,1	12,8	-0,8	3,6	-5,7	8,6
70	1	9,9	22,5	12,9	3,3	2,3	-3,5	7,9
	2	8,6	24,1	11,9	4,5	6,4	-7,5	8,0
	3	19,2	29,3	15,2	1,8	2,0	1,2	11,5
	4	15,9	23,7	11,0	6,4	3,7	-3,1	9,6
	5	12,8	24,1	7,6	3,0	5,1	-,4	8,7
	6	18,7	29,0	12,3	3,8	-1,8	-5,0	9,5
	7	15,4	24,8	10,8	4,7	3,5	-4,5	9,1
	8	14,8	22,6	11,5	2,3	1,6	-4,1	8,1
	9	16,4	22,4	9,9	1,3	0,3	-2,1	8,0
	10	10,7	17,3	8,4	3,0	3,0	-7,7	5,8
90	1	11,0	15,1	-6,0	0,1	-6,7	-49,8	-6,1
	2	15,2	21,1	2,0	3,6	-7,2	-34,2	0,1
	3	15,7	20,1	4,4	1,0	-7,0	-44,3	-1,7
	4	17,3	10,6	9,7	3,0	-1,8	-47,7	-1,6
	5	16,3	13,4	7,9	2,9	-1,8	-80,0	-7,0
	6	13,9	12,3	3,5	1,8	-1,0	-39,8	-1,5
	7	12,0	7,2	0,6	0,2	-4,4	-57,4	-6,9
	8	10,9	6,9	2,3	-0,8	-8,2	-56,7	-7,6
	9	11,8	13,6	-0,7	-2,2	-10,1	-69,3	-9,5
	10	8,0	1,6	-4,7	-5,6	-10,0	-84,1	-15,8
110	1	17,1	5,6	2,3	-7,7	-5,3	-68,1	-9,4
	2	16,5	9,7	5,7	3,2	-3,1	-113,3	-13,6
	3	16,4	6,1	9,1	1,3	-7,4	-195,4	-28,3
	4	17,0	8,3	9,0	-5,1	-6,5	-67,4	-7,5
	5	12,7	10,1	4,2	-8,4	-8,0	-62,9	-8,7
	6	13,3	3,6	2,0	-13,2	-12,4	-68,8	-12,6
	7	1,6	3,7	-0,1	-22,9	-14,8	-104,1	-22,8
	8	-13,5	-2,2	-6,2	-25,9	-17,3	-155,3	-36,7
	9	-13,3	-2,1	-7,1	-36,1	-25,3	-169,3	-42,2
	10	-26,3	-8,5	-9,6	-47,5	-28,8	-247,4	-61,4

Nota-se que há uma grande melhoria no desempenho de BE3 quando utiliza-se o tempo máximo dobrado. Mesmo assim não foi possível evitar que a busca se comportasse um

pouco pior para problemas com um número alto de tarefas. Entretanto, observa-se que os valores obtidos para a média de ganho para cada valor de n são mais uniformes na Tabela 6.7 que na Tabela 6.6. Portanto, a Versão 10 de BE3 foi aplicada a todos os problemas de teste do Conjunto A utilizando as seguintes profundidades: profundidade = 4 para n = 30, 3 para n = 50 e 70, 2 para n = 90 e 4 para n = 110.

Os resultados desses testes são apresentados e discutidos na seção seguinte.

6.5 - DESEMPENHO DE BE3

Na figura 6.1 apresenta-se um quadro resumindo as características de BE3. Nesta seção são apresentados os resultados da aplicação de BE3 sobre todos os problemas de teste de grande porte, cuja geração é descrita na seção 1.4.5. Utilizou-se para cada valor de n o dobro do tempo estimado para que GEN1 realizasse 3000 gerações sobre os problemas de mesmo número de tarefas. Portanto, BE3 foi aplicada a cada problema por um tempo igual a 200 segundos para n = 30, 400 segundos para n = 50, 640 segundos para n = 70, 1040 segundos para n = 90 e 1100 segundos para n = 110. Nas tabelas 6.8 a 6.13 apresentam-se os resultados desses testes.

```
Conjuntos de Soluções:
      Pt = subconjunto diverso de todas as soluções geradas até a
           iteração t, com p = 20 elementos
      P*t = conjunto com as p* = 10 melhores soluções de Pt
      Ht = conjunto das h = 30 melhores soluções encontradas pela
           busca até a iteração t
      H*t = conjunto com as h* = 8 melhores soluções não tabu de Ht
      Rt = H*t × Pt
População Inicial:
      construída pelo procedimento apresentado na seção 4.2.1, com
           qmax = 200 soluções
Controle de Repetição de Soluções:
      através da comparação do valor característico e do custo de
           cada solução
Métrica de Distância:
      diferença entre os arcos presentes nas soluções
Construção de Subconjuntos Diversos
      Regra apresentada na seção 4.2.1
Geração de Soluções Tentativas
      Combinações estruturadas entre os pares de soluções em Rt
           conforme o procedimento descrito na seção 6.3
      Escolha aleatória do melhor voto de cada solução
      Profundidade das combinações: 4 para n = 30, 3 para n= 50 e 70,
           2 para n = 90 e 4 para n = 110
Armazenamento de Soluções Tentativas e Atualização dos Conjuntos:
      Procedimento apresentado na seção 4.2.5
```

Figura 6.1 - Resumo das características de BE3

Os gráficos das figuras 6.2 e 6.3 mostram, respectivamente, os valores de ganho da coluna "Ganho Médio(%)" das tabelas 6.8 a 6.13 e o número de iterações que BE3 efetivamente utiliza para encontrar a melhor solução, calculado como uma porcentagem do número total de iterações.

Nota-se que, apesar de BE3 ser a melhor versão de Busca por Espalhamento implementada até o momento neste trabalho, seu desempenho ainda não é satisfatório, principalmente quando comparado ao desempenho apresentado por GEN1. Na Figura 6.2 pode-se observar que BE3 possui ganhos altos para problemas com 30 tarefas mas que, já a partir de 50 tarefas, o ganho cai bastante. O algoritmo também mostra-se bastante sensível à variação do número de máquinas. Para problemas maiores, até aqueles com 2 máquinas, apesar de baixo, o ganho de BE3 é positivo. Entretanto, quando o número de máquinas aumenta, BE3 passa a apresentar soluções piores que as das heurísticas utilizadas para comparação.

Tabela 6.8 - Resultados obtidos para BE3 com m = 1

n	tipo	Média Heur	Média BE3	Ganho/ Tipo(%)	Ganho Médio(%)	Iterações (Média)	Iterações para o melhor (média)
30	1	5174,7	4528,0	12,47	24,34	167,90	76,57
	2	2898,9	2277,7	24,14			
	3	1325,8	1025,3	36,39			
50	1	12046,4	10913,2	9,87	21,57	292,43	124,53
	2	7237,7	6352,1	14,26			
	3	1335,0	888,5	40,57			
70	1	25394,2	24462,3	3,99	14,46	379,60	199,57
	2	12129,2	10679,1	15,45			
	3	3371,4	2979,1	23,20			
90	1	43514,7	42502,9	2,53	11,31	635,47	322,23
	2	17532,8	16693,4	4,71			
	3	2825,1	2428,0	26,69			
110	1	64181,4	63910,0	0,33	11,41	194,40	177,50
	2	27541,3	26576,4	5,44			
	3	2480,6	2088,1	28,45			

Tabela 6.9 - Resultados obtidos para BE3 com m = 2

n	tipo	Média Heur	Média BE3	Ganho/ Tipo(%)	Ganho Médio(%)	Iterações (Média)	Iterações para o melhor (média)
30	1	3356,0	2773,2	18,41	28,77	148,07	69,40
	2	1835,7	1417,5	25,89			
	3	1129,1	764,3	42,01			
50	1	8082,7	7136,0	11,92	21,12	274,30	161,23
	2	4625,3	3754,8	19,32			
	3	1802,1	1329,4	32,12			
70	1	14031,0	12590,7	10,34	23,65	345,30	212,37
	2	8002,7	6572,9	19,79			
	3	2440,8	1640,8	40,82			
90	1	23266,1	20925,1	10,60	-8,08	587,63	368,87
	2	11240,0	9335,8	20,60			
	3	2529,5	2084,8	-55,48			
110	1	30967,3	28853,4	6,77	9,07	193,13	176,07
	2	14823,8	12574,8	17,63			
	3	2532,7	2248,8	2,81			

Tabela 6.10 - Resultados obtidos para BE3 com m = 3

n	tipo	Média Heur	Média BE3	Ganho/ Tipo(%)	Ganho Médio(%)	Iterações (Média)	Iterações para o melhor (média)
30	1	2456,8	2228,9	9,31	15,67	131,40	85,63
	2	1574,6	1380,0	12,55			
	3	551,2	420,8	25,15			
50	1	5876,9	5385,7	8,44	12,60	242,97	156,67
	2	3530,3	3115,9	11,87			
	3	1190,9	954,6	17,49			
70	1	10163,7	9275,7	8,81	8,21	337,83	229,53
	2	5580,1	4967,9	11,30			
	3	1440,0	1316,4	4,51			
90	1	14707,4	14051,9	4,47	9,25	627,13	384,27
	2	7971,0	7117,8	10,94			
	3	1972,1	1835,6	12,34			
110	1	22043,1	20838,4	5,49	3,18	191,07	184,83
	2	10051,2	9308,7	7,05			
	3	3572,6	3430,4	-3,01			

Tabela 6.11 - Resultados obtidos para BE3 com m = 4

n	tipo	Média Heur	Média BE3	Ganho/ Tipo(%)	Ganho Médio(%)	Iterações (Média)	Iterações para o melhor (média)
30	1	1736,9	1576,0	9,41	11,74	124,67	86,23
	2	1181,6	1082,1	9,35			
	3	941,0	796,3	16,46			
50	1	4476,6	4150,8	7,41	4,91	239,60	163,97
	2	2466,6	2246,2	8,69			
	3	780,7	727,7	-1,38			
70	1	7421,9	7088,9	4,35	-5,16	327,17	216,33
	2	4243,1	3933,1	6,71			
	3	1947,7	2022,0	-26,54			
90	1	13609,9	12889,2	5,19	0,63	606,60	375,10
	2	6778,8	6417,3	5,06			
	3	1931,4	2000,8	-8,37			
110	1	16961,3	16525,5	2,72	-6,65	190,17	181,27
	2	9782,0	9578,8	1,95			
	3	1625,3	1924	-24,62			

Tabela 6.12 - Resultados obtidos para BE3 com m = 5

n	tipo	Média Heur	Média BE3	Ganho/ Tipo(%)	Ganho Médio(%)	Iterações (Média)	Iterações para o melhor (média)
30	1	1854,9	1736,0	6,70	8,74	117,10	74,27
	2	1146,2	1061,7	7,35			
	3	891,2	797,2	12,16			
50	1	3726,5	3570,6	4,18	4,15	225,03	151,60
	2	2444,1	2245,5	9,12			
	3	1370,4	1336,3	-0,87			
70	1	6964,4	6608,9	5,34	-1,61	301,93	222,47
	2	3679,1	3617,7	1,20			
	3	1199,2	1188,8	-11,39			
90	1	10528,6	10133,7	3,76	-10,96	590,67	397,17
	2	5242,8	5104,9	2,52			
	3	1741,1	2016,9	-39,14			
110	1	14672,3	14716,1	-0,33	-11,54	188,57	180,43
	2	6753,2	6485,8	3,62			
	3	2361,1	2741,8	-37,91			

Tabela 6.13 - Resultados obtidos para BE3 com m = 6

n	tipo	Média Heur	Média BE3	Ganho/ Tipo(%)	Ganho Médio(%)	Iterações (Média)	Iterações para o melhor (média)
30	1	1688,8	1592,0	5,72	5,85	110,57	83,67
	2	1412,0	1327,2	6,74			
	3	833,0	790,3	5,09			
50	1	3770,6	3617,9	4,35	1,77	219,60	129,70
	2	1939,9	1861,1	3,99			
	3	936,9	970,6	-3,04			
70	1	5404,8	5269,1	2,38	-3,50	307,07	204,43
	2	3936,0	3818,1	2,82			
	3	1127,1	1256,9	-15,70			
90	1	9541,5	9280,9	2,70	-13,82	583,60	347,87
	2	4463,6	4465,1	0,13			
	3	1737,4	1959,1	-44,28			
110	1	11930,9	11961,1	-0,44	-28,97	186,37	181,67
	2	6401,1	6771,8	-6,62			
	3	1495,4	1848,8	-79,85			

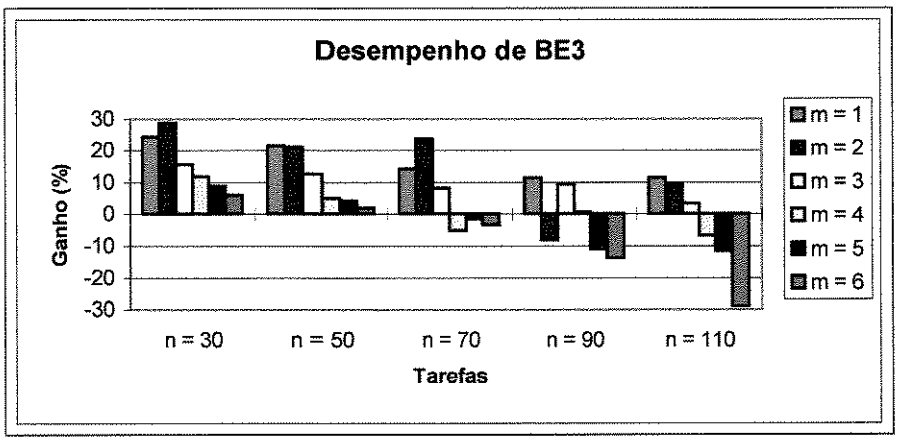

Figura 6.2 - Ganho médio de BE3

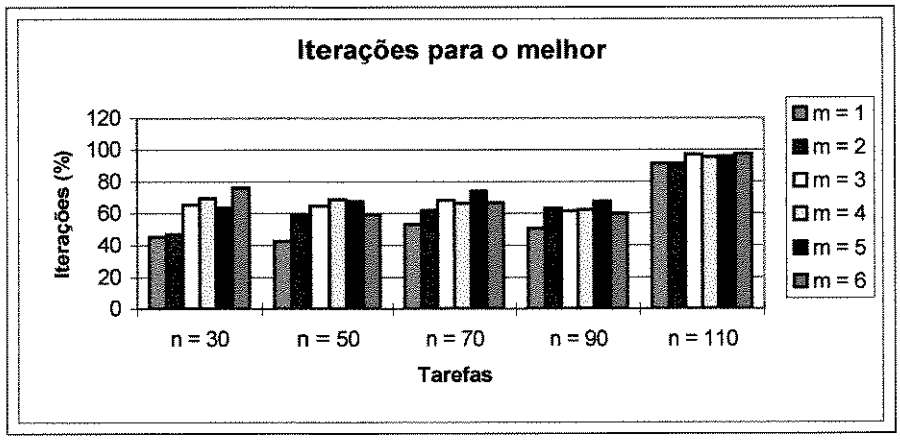

Figura 6.3 - Número de iterações de BE3 utilizadas para encontrar a melhor solução

Antes de alterar mais uma vez o algoritmo implementado, foi feita uma análise da porcentagem de sucessos obtidos por BE3 com relação às heurísticas ATCS e SAA. Observou-se que BE3 gera soluções melhores que ATCS para 89,333% dos problemas com m = 1.

Com relação aos problemas com m > 1, BE3 gera soluções melhores que a heurística SAA para 96,67% dos problemas com m = 2, 87,33% dos problemas com m = 3, 75,33% para m = 4, 70,67% para m = 5 e 56,67% para m = 6. Esses valores estão abaixo dos observados para GEN1 mas, não são tão baixos quanto o gráfico da Figura 6.2 poderia sugerir. Isso indica que BE3 pode ter seu desempenho melhorado se for encontrada uma forma de torná-la mais robusta com relação à variação do número de máquinas.

Uma forma de modificar BE3, indicada em GLOVER (1994a), é introduzir uma busca local a ser realizada em determinados momentos da Busca por Espalhamento, sobre algum dos conjuntos de soluções utilizados por ela. A introdução de uma busca local poderia, a princípio, aumentar o tempo que BE3 demora para resolver os problemas. Entretanto, observa-se na Figura 6.3 que, exceto para os problemas com 110 tarefas, o número de iterações que BE3 necessita para encontrar sua solução final fica em torno de 60% do número total de iterações realizadas no tempo permitido. Pode-se dizer que BE3 passa metade do tempo que é dado a ela para resolver cada problema tentando melhorar, sem sucesso, o valor de sua melhor solução. Esse tempo pode ser aproveitado para a realização da busca local. Na seção seguinte, apresentam-se os detalhes da implementação de busca local sobre as soluções de BE3.

6.6 - BUSCA LOCAL EM BE3

6.6.1 - Implementação

Para a implementação de uma versão de BE3 que utilize busca local foram levados em conta alguns fatores que afetam seu desempenho. O primeiro deles é a definição da vizinhança a ser explorada pela busca local. Três tipos de movimentos foram estudados para determinar a vizinhança que melhor explorasse o espaço de busca do problema PST, aliada a BE3. Esses movimentos são Trocas Simples, Inserções e Trocas Totais.

Dada uma solução S_x, as Trocas Simples definem uma vizinhança NTS(S_x) composta por soluções obtidas a partir de S_x por movimentos de troca de posição entre dois elementos dessa solução. Essa vizinhança foi chamada de simples pois somente são permitidas as trocas realizadas entre elementos de S_x que correspondam a nós de tarefa. Dessa forma, o número de tarefas processadas em cada máquina nas soluções de NTS(S_x) é igual ao de S_x. Portanto, essa vizinhança permite a avaliação de soluções que diferem de S_x nos aspectos de seqüenciamento das tarefas em cada máquina e da designação de tarefas a máquinas, desde que o número de tarefas processadas por máquina não varie.

As Inserções definem uma vizinhança NI(S_x) composta por soluções obtidas a partir de S_x por movimentos de inserção de um de seus elementos em uma nova posição. Novamente, só se permite a escolha de elementos correspondentes a nós de tarefa mas a

posição em que esse elemento pode ser inserido varia de 1 a (m+n). Sendo assim, existem em NI(S_x) soluções com o mesmo número de tarefas por máquina de S_x e soluções em que uma das máquinas apresenta uma tarefa a menos e outra máquina apresenta uma tarefa a mais. Além de permitir a avaliação de soluções com diferenças de seqüenciamento com relação a S_x, essa vizinhança permite que sejam avaliadas algumas soluções que apresentem diferenças de designação das tarefas às máquinas onde o número de tarefas processadas por cada máquina varia de maneira gradual ao longo do processo de busca.

A última vizinhança estudada é chamada NTT(S_x) e é definida pelos movimentos de Trocas Totais. As soluções pertencentes a NTT(S_x) correspondem a todas as soluções de NTS(S_x) e de soluções para as quais é permitido trocar dois elementos de S_x sendo um deles correspondente a um nó de tarefa e outro a um nó de máquina. Essas últimas soluções podem apresentar diferenças de designação de tarefas a máquinas bastante significativas e esse tipo de movimento foi inserido neste estudo para que fosse avaliado o efeito do seu componente diversificador.

Além do movimento que define a vizinhança explorada pela busca local, um outro fator que pode afetar seu desempenho é a forma de transição entre as vizinhanças de duas iterações consecutivas. Na maior parte das aplicações de busca local, toda a vizinhança N(S_x) da solução atual S_x é construída e avaliada em busca da melhor solução S_y dessa vizinhança. Comparam-se, então, os custos dessas duas soluções e se o custo de S_y for melhor que o de S_x, S_y substitui S_x e uma nova iteração é iniciada com a construção da vizinhança N(S_y). Caso contrário, a busca pára e S_x é o ótimo local encontrado com respeito à vizinhança escolhida. Uma outra forma de se realizar a transição de vizinhanças é avaliar gradualmente as soluções de N(S_x), em uma ordem arbitrária e, ao ser encontrada a primeira solução S_z com custo melhor que S_x, S_z substitui S_x e uma nova iteração é iniciada sem que todos os elementos de N(S_x) tenham sido avaliados. O número de soluções de N(S_x) avaliadas com esse último tipo de transição é menor que no primeiro tipo, o que pode acelerar a busca. Entretanto, o efeito do tipo de transição no valor do custo do ótimo local encontrado só pode ser determinado através de testes computacionais. Ambas as transições descritas acima foram testadas para avaliar esse efeito.

Outro aspecto importante relativo à busca local diz respeito aos elementos da solução S_x que são considerados para fazer parte dos movimentos de construção da vizinhança. A princípio, todos os elementos de S_x são candidatos a participar desses movimentos. Entretanto, pode-se reduzir o esforço computacional envolvido na busca se forem considerados apenas os elementos cuja troca ou inserção apresente um maior potencial de gerar uma solução vizinha de custo melhor que o de S_x. Dessa forma, dado um critério para a escolha da lista restrita de candidatos, a vizinhança de S_x pode ser construída por movimentos que incluam somente os elementos dessa lista. No caso da vizinhança NTS(S_x), por exemplo, somente seriam considerados movimentos de troca entre pares de tarefas em que uma delas

estivesse presente na lista de candidatos.

A elaboração de uma lista restrita de elementos candidatos deve considerar os aspectos específicos do problema a ser resolvido e, no caso do problema PST, decidiu-se utilizar as informações relativas ao tempo de processamento das tarefas e dos tempos de preparação de máquina. A cada solução S_x, procura-se pela tarefa J_{j*} que apresente a maior soma de tempo de processamento com o tempo de preparação de máquina associado. Essa medida é descrita como o comprimento do arco de precedência (i,j^*) e seu cálculo é feito através da expressão (1.10). A vizinhança de S_x é construída, então, permitindo-se apenas movimentos de troca e/ou inserção em que a tarefa J_{j*} esteja envolvida. Outra possibilidade é a escolha aleatória de J_{j*}. Ambas as abordagens foram testadas para determinar qual proporciona o melhor desempenho para a busca local.

O quarto aspecto que precisa ser definido para a implementação de busca local em BE3 é o conjunto de soluções sobre as quais será feita essa busca. Para determinar o melhor conjunto de soluções alvo para busca local, foram testadas três opções. A primeira delas consiste do conjunto P^*_t, onde são armazenadas as soluções de melhor custo do subconjunto diverso de todas as soluções encontradas durante a Busca por Espalhamento até a iteração t. Esse conjunto foi considerado por ser composto de soluções bastante diversificadas, o que pode ser aproveitado pela busca local para realizar uma pequena intensificação em regiões distantes entre si no espaço de busca do problema PST. A segunda opção considerada foi o conjunto H^*_t, composto pelas soluções de elite encontradas até a iteração t. Essa escolha foi motivada pelo fato da soluções desse conjunto possuírem baixos valores de custo, o que indica que suas vizinhanças devem conter outras boas soluções.

A terceira opção considerada para as soluções alvo da busca local é o conjunto $D(H_t \cup P_t)$, construído como um subconjunto diverso da união entre H_t e P_t. Procurou-se, dessa forma, proporcionar à busca local um conjunto de soluções iniciais que reunisse tanto soluções de baixo custo como soluções diversificadas. Entretanto, como P_t já foi construído como um subconjunto diverso das soluções encontradas por BE3 ao longo das iterações anteriores, suas soluções são bastante diversificadas com relação à distância calculada pela diferença de arcos entre elas. Se o conjunto $D(H_t \cup P_t)$ for construído pela mesma regra de subconjuntos diversos que P_t, isso pode fazer com que $D(H_t \cup P_t)$ seja composto basicamente por soluções desse conjunto, pois não é feita nenhuma restrição quanto às distâncias entre soluções em H_t. Decidiu-se, então, modificar a regra de construção de subconjuntos diversos para que ela utilize uma métrica de distância entre soluções baseada tanto em aspectos de diferença de arcos quanto no aspecto de designação de tarefas a máquinas pois o aspecto de designação mostrou ter bastante influência no desempenho de BE2. Apesar do ganho médio final dessa implementação ser baixo, a consideração da distância de designação antes da distância de seqüenciamento nos testes apresentados na seção 5.4.2.B gera versões dominantes de BE2. Como foi apontado no final do Capítulo 5, o

baixo valor de ganho médio obtido com BE2 pode ser associado fortemente ao alto tempo computacional gasto com o cálculo da distância de designação. Entretanto, se essa distância for usada como uma medida acessória para a construção de $D(H_t \cup P_t)$, em iterações específicas de BE3, e não como a distância principal entre soluções a ser calculada a cada solução tentativa gerada pelas combinações estruturadas (procedimento Conjuntos_Intermediários), o tempo extra gasto com os cálculos envolvidos pode não ser prejudicial ao andamento da Busca por Espalhamento. O compromisso entre o tempo adicional gasto e a qualidade obtida em decorrência do aumento da diversificação entre as soluções do conjunto $D(H_t \cup P_t)$ foi avaliado computacionalmente.

Portanto, quando o conjunto $D(H_t \cup P_t)$ é construído, utiliza-se a regra de construção de subconjuntos diversos dada na seção 4.2.1, que é a mesma utilizada por BE3 para construir P_t. Entretanto, para $D(H_t \cup P_t)$ a construção é feita considerando-se que a distância entre duas soluções S_x e S_y é calculada como a soma ponderada da distância baseada na diferença entre os arcos dessas soluções e da distância de designação. Como essas duas medidas de distância apresentam intervalos de variação diferentes, foram normalizadas para que ambas variem entre 0 e 100%. Seja d_a a distância baseada em arcos e d_d a distância de designação calculada pela expressão (5.1). Então, a distância d_{xy} entre S_x e S_y, na construção de $D(H_t \cup P_t)$, é dada por

$$d_{xy} = \frac{n}{m} \cdot d_a + \left(n - \frac{n}{m}\right) \cdot d_d \qquad (6.3)$$

e varia entre 0 e n. A ponderação entre as distâncias foi estabelecida de forma que, quando $m = 1$ e as distâncias de designação forem todas nulas, a distância de arcos seja a única considerada. Quando o número de máquinas aumenta, a distância de designação passa a assumir um papel dominante em d_{xy}.

Os últimos aspectos da busca local a serem considerados em sua implementação em BE3 são a freqüência com que ela será aplicada ao conjunto de soluções alvo inicial e o número de vizinhanças que serão exploradas para cada uma das soluções alvo. Duas formas diferentes de aplicação de busca local foram previstas neste trabalho. Na primeira delas, aplica-se a busca local a cada vez que se passar um número fixo de iterações, sem considerar se a Busca por Espalhamento por si só está ou não conseguindo melhorar sua solução incumbente. Na outra abordagem, aplica-se a busca local a cada vez que se passar um certo número de iterações em que não ocorra melhoria da solução incumbente. Quanto ao número de vizinhanças exploradas para cada solução inicial, decidiu-se fixar um número máximo de n iterações para a busca local. Essa decisão foi tomada tendo em vista que a busca local é aplicada sobre um conjunto de soluções e pode-se gastar muito tempo para se chegar a um ótimo local a partir de cada uma delas.

A seguir apresenta-se um resumo dos fatores considerados na implementação da busca local em BE3, juntamente com todas as opções previstas para eles.

Fator 1: Vizinhança

Trocas Simples

Inserções

Trocas Totais

Inserções + Trocas Simples

Inserções + Trocas Totais

Fator 2: Transição da vizinhança

Melhor: passa-se para a próxima iteração após avaliar toda vizinhança da solução atual

Primeiro: avalia-se a vizinhança apenas até encontrar o primeiro vizinho melhor que a solução atual

Fator 3: Lista de Candidatos

Por arco: J_{j*} escolhida pelo maior arco $(i,j*)$ na solução atual

Aleatória: J_{j*} escolhida aleatoriamente

Fator 4: Soluções Alvo

P^*_t

H^*_t

$D(H_t \cup P_t)$

Fator 5: Freqüência de Aplicação

A cada 5 iterações

A cada 10 iterações

A cada 15 iterações

A cada 5 iterações sem melhoria da solução incumbente

A cada 10 iterações sem melhoria da solução incumbente

A cada 15 iterações sem melhoria da solução incumbente

6.6.2 - Testes Computacionais

Há 720 combinações possíveis de todas as opções previstas para os fatores 1 a 5. Como foi feito anteriormente com BE2 e BE3, os testes computacionais realizaram-se em duas fases para que pudesse ser reduzido o esforço computacional necessário para se obter a melhor versão de busca local aplicada a BE3.

O conjunto de problemas de teste escolhido para esta parte do trabalho consiste de 36 problemas com 70 tarefas, sendo 6 problemas para cada valor de m. Essa escolha foi motivada pelo desejo de se obter uma versão final de Busca por Espalhamento que possuísse uma certa robustez com relação à variação do número de máquinas nos problemas de teste. Também foi decidido que a busca local seria aplicada a BE3 com profundidade igual a 1 nas combinações de soluções.

Fase 1: Nessa fase, decidiu-se fixar os fatores 2, 3 e 5 e variar os fatores 1 e 4. Foram

geradas 15 versões, chamadas BE3BL1 a BE3BL15, cada uma delas correspondendo a uma combinação das opções desses fatores. As características que permanecem inalteradas nessas 15 versões são:

Fator 2: Transição da Vizinhança pelo melhor vizinho;

Fator 3: Escolha da tarefa candidata J_{j*} através do maior arco (i,j*);

Fator 5: Aplicação da busca local a cada 10 iterações.

Cada uma das versões de BE3 com busca local foi aplicada a todo o conjunto de 36 problemas de teste e foram calculados seus ganhos médios com relação à heurística SAA pela expressão 4.9 para cada grupo de 6 problemas com mesmo número de máquinas. Também foi calculada a média geral de ganho para todos os problemas e o coeficiente de variação dos valores de ganho por máquina com relação à média geral. Todas as versões testadas foram comparadas entre si, aos pares, pelo teste de Wilkoxon e as versões não eliminadas por esse teste são apresentadas na Tabela 6.14. Nessa tabela também são apresentados os valores de ganho por máquina, média geral de ganho e coeficiente de variação para a versão de BE3 sem busca local.

Tabela 6.14 - Ganhos médios (%) obtidos pelas versões de BE3 com busca local ao final dos testes da Fase 1

	m = 1	m = 2	m = 3	m = 4	m = 5	m = 6	Média	CV
BE3	19,15	29,26	15,24	1,78	1,99	1,19	11,44	1,02
BE3BL1	8,99	27,79	15,52	5,84	9,37	0,50	11,34	0,83
BE3BL3	18,21	26,67	12,41	7,82	10,46	5,03	13,43	0,59
BE3BL6	18,21	27,00	14,19	4,74	8,81	0,99	12,32	0,77
BE3BL10	17,43	23,57	12,06	8,15	7,93	4,99	12,35	0,56
BE3BL12	8,18	25,35	13,17	5,67	8,53	1,58	10,41	0,79
BE3BL13	17,43	25,01	11,78	8,16	6,86	4,84	12,36	0,62
BE3BL15	8,18	18,07	12,82	8,33	7,14	3,54	9,68	0,52

Observa-se que a versão BE3BL3 é a que apresenta a melhor média de ganho, associada ao terceiro menor coeficiente de variação. Essa versão possui as seguintes características com relação aos fatores 1 a 5: vizinhança construída por movimentos de Trocas Simples, transição de vizinhança pelo melhor vizinho, construção de lista de candidatos pela escolha da tarefa J_{j*} através do maior arco (i,j*), aplicação da busca local sobre as soluções alvo em $D(H_t \cup P_t)$ e aplicação da busca local a cada 10 iterações. BE3BL3 foi considerada a melhor versão testada até o momento e serviu como base para os testes realizados na Fase 2.

Fase 2: Nessa fase dos testes computacionais, foram geradas 23 versões da busca, tendo como base a versão BE3BL3 e variando-se os fatores 2, 3 e 5. Novamente, cada uma dessas versões foi aplicada aos 36 problemas de teste e seus resultados comparados através do teste de Wilkoxon. Na Tabela 6.15 são mostrados os valores de ganho médio por máquina,

média geral de ganho e coeficiente de variação para as versões não eliminadas ao final dessa fase. Nota-se que nenhuma das versões alternativas foi capaz de melhorar a média geral de ganho de BE3BL3 nem aumentar a robustez da busca com relação ao número de máquinas.

Duas outras versões foram testadas, BE3BL39 e BE3BL40. A primeira delas consistiu de modificar BE3BL3 de forma que a escolha do melhor voto de cada solução nas combinações estruturadas fosse feita através do maior arco. Com isso, desejava-se investigar se o acréscimo de um procedimento de busca local a BE3 não poderia ter diminuído a necessidade de diversificação proporcionada pela escolha aleatória do melhor voto. Essa versão apresentou uma média geral de ganho, calculada sobre os 36 problemas de testes, de -10,22%, mostrando que, mesmo com a busca local, ainda é necessário manter a escolha aleatória dos votos das soluções na geração de soluções tentativas.

Tabela 6.15 - Ganhos médios (%) obtidos pelas versões de BE3 com busca local ao final dos testes da Fase 2

	m = 1	m = 2	m = 3	m = 4	m = 5	m = 6	Média	CV
BE3BL3	18.21	26.67	12.41	7.82	10.46	5.03	13.43	0.59
BE3BL16	16.61	26.84	13.52	6.91	10.49	4.32	13.10	0.61
BE3BL18	9.49	24.97	13.62	9.77	7.64	4.72	11.70	0.61
BE3BL23	9.65	18.53	13.23	8.44	7.64	-0.13	11.23	0.85
BE3BL24	13.56	13.70	12.53	7.36	8.15	2.56	11.31	0.64
BE3BL29	16.38	16.53	14.59	7.27	7.65	0.93	12.23	0.73
BE3BL30	13.29	26.06	12.77	7.85	8.57	-0.83	11.29	0.78
BE3BL34	19.08	26.86	12.84	6.08	9.00	0.40	12.38	0.77
BE3BL35	15.31	23.59	14.23	4.31	10.74	3.40	11.93	0.63
BE3BL36	14.43	26.12	14.25	8.35	5.48	-1.37	11.21	0.84
BE3BL37	16.03	19.96	12.19	7.04	9.04	-1.28	10.50	0.71

A outra versão, BE3BL40 é idêntica a BE3BL3, com exceção do número de vizinhanças exploradas para cada solução alvo da busca local. O número máximo de vizinhanças permitido foi aumentado para 2.n para saber se a limitação imposta anteriormente não estaria prejudicando a qualidade da busca. Observaram-se poucas diferenças nos valores finais de atraso gerados por BE3BL3 e BE3BL40, sendo que a média geral de ganho da última é igual a 12,73%. Sendo assim, decidiu-se prosseguir com os testes de desempenho utilizando BE3BL3 como melhor implementação de Busca por Espalhamento desenvolvida até o momento.

6.7 - DESEMPENHO DE BE3BL3

Apresentam-se, nesta seção, os resultados da aplicação de BE3BL3 sobre todos os problemas de teste. Da mesma forma como para BE3, utilizou-se, para cada valor de n o dobro do tempo estimado para que GEN1 realizasse 3000 gerações sobre os problemas de

mesmo número de tarefas. Nas tabelas 6.16 a 6.21 apresentam-se os resultados desses testes.

Os gráficos das figuras 6.4 e 6.5 mostram, respectivamente, os valores de ganho da coluna "Ganho Médio(%)" das tabelas 6.16 a 6.21 e o número de iterações que BE3BL3 efetivamente utiliza para encontrar a melhor solução, calculado como uma porcentagem do número total de iterações.

Tabela 6.16 - Resultados obtidos para BE3BL3 com m = 1

n	tipo	Média Heur	Média BE3BL3	Ganho/ Tipo(%)	Ganho Médio(%)	Iterações (Média)	Iterações para o melhor (média)
30	1	5174,7	4622,2	10,62	23,82	633,63	89,30
	2	2898,9	2313,2	22,93			
	3	1325,8	1023,1	37,90			
50	1	12046,4	10981,9	9,29	20,49	598,93	157,53
	2	7237,7	6391,7	11,88			
	3	1335,0	902,1	40,30			
70	1	25394,2	24518,4	3,66	16,22	642,20	240,83
	2	12129,2	10634,1	16,62			
	3	3371,4	2802,5	28,37			
90	1	43514,7	41767,5	4,15	14,89	529,87	306,77
	2	17532,8	16273,8	8,66			
	3	2825,1	2348,4	31,88			
110	1	64181,4	62934,8	2,11	14,50	317,93	263,23
	2	27541,3	25922,1	8,38			
	3	2480,6	1837,4	33,00			

Tabela 6.17 - Resultados obtidos para BE3BL3 com m = 2

n	tipo	Média Heur	Média BE3BL3	Ganho/ Tipo(%)	Ganho Médio(%)	Iterações (Média)	Iterações para o melhor (média)
30	1	3356,0	2830,4	16,53	26,31	658,03	158,43
	2	1835,7	1463,3	22,68			
	3	1129,1	793,5	39,74			
50	1	8082,7	7158,3	11,26	21,46	586,03	140,00
	2	4625,3	3773,7	19,07			
	3	1802,1	1327,9	34,03			
70	1	14031,0	12383,0	11,89	23,47	531,47	178,70
	2	8002,7	6743,0	17,92			
	3	2440,8	1690,0	40,61			
90	1	23266,1	20399,0	12,85	18,78	453,33	273,97
	2	11240,0	9090,7	21,53			
	3	2529,5	1932,4	21,98			
110	1	30967,3	27554,2	11,05	10,62	275,17	227,10
	2	14823,8	11440,0	25,86			
	3	2532,7	2090,2	-5,05			

Tabela 6.18 - Resultados obtidos para BE3BL3 com m = 3

n	tipo	Média Heur	Média BE3BL3	Ganho/ Tipo(%)	Ganho Médio(%)	Iterações (Média)	Iterações para o melhor (média)
30	1	2456,8	2218,4	9,87	16,28	616,17	120,10
	2	1574,6	1339,2	15,20			
	3	551,2	431,3	23,77			
50	1	5876,9	5457,8	7,21	13,92	570,53	139,23
	2	3530,3	3113,7	12,77			
	3	1190,9	911,3	21,78			
70	1	10163,7	9274,8	8,78	12,78	554,97	260,67
	2	5580,1	4883,6	12,49			
	3	1440,0	4583,5	16,06			
90	1	14707,4	13572,2	7,74	5,29	450,97	286,67
	2	7971,0	6888,5	13,67			
	3	1972,1	1609,9	-5,53			
110	1	22043,1	19986,7	9,38	12,77	280,07	240,27
	2	10051,2	8322,0	18,18			
	3	3572,6	3194,9	10,77			

Tabela 6.19 - Resultados obtidos para BE3BL3 com m = 4

n	tipo	Média Heur	Média BE3BL3	Ganho/ Tipo(%)	Ganho Médio(%)	Iterações (Média)	Iterações para o melhor (média)
30	1	1736,9	1583,3	8,69	13,48	642,67	116,47
	2	1181,6	1062,9	11,25			
	3	941,0	761,1	20,50			
50	1	4476,6	4177,4	6,62	11,66	528,83	186,93
	2	2466,6	2175,3	12,09			
	3	780,7	667,6	16,28			
70	1	7421,9	6958,1	6,24	8,11	535,90	193,07
	2	4243,1	3752,0	11,29			
	3	1947,7	1737,5	6,80			
90	1	13609,9	12615,5	7,16	8,50	430,23	245,73
	2	6778,8	6123,4	9,75			
	3	1931,4	1852,4	8,57			
110	1	16961,3	15847,4	6,69	5,97	271,53	230,13
	2	9782,0	9023,6	7,80			
	3	1625,3	1578,6	3,42			

Tabela 6.20 - Resultados obtidos para BE3BL3 com m = 5

n	tipo	Média Heur	Média BE3BL3	Ganho/ Tipo(%)	Ganho Médio(%)	Iterações (Média)	Iterações para o melhor (média)
30	1	1854,9	1730,2	6,79	9,88	577,33	130,83
	2	1146,2	1038,3	9,80			
	3	891,2	792,3	13,05			
50	1	3726,5	3479,3	6,70	9,06	554,47	215,60
	2	2444,1	2209,7	10,99			
	3	1370,4	1253,2	9,48			
70	1	6964,4	6570,0	5,57	3,83	502,97	219,53
	2	3679,1	3412,1	7,52			
	3	1199,2	1137,0	-1,61			
90	1	10528,6	9855,5	6,45	2,85	453,17	232,90
	2	5242,8	4952,8	6,35			
	3	1741,1	1697,5	-4,24			
110	1	14672,3	13958,4	5,00	-4,24	268,40	239,27
	2	6753,2	6108,1	9,90			
	3	2361,1	2407,1	-27,62			

Tabela 6.21 - Resultados obtidos para BE3BL3 com m = 6

n	tipo	Média Heur	Média BE3BL3	Ganho/ Tipo(%)	Ganho Médio(%)	Iterações (Média)	Iterações para o melhor (média)
30	1	1688,8	1603,8	5,17	6,35	572,97	107,80
	2	1412,0	1319,4	7,15			
	3	833,0	777,9	6,72			
50	1	3770,6	3601,7	4,71	6,49	501,13	155,90
	2	1939,9	1857,5	3,50			
	3	936,9	852,9	11,25			
70	1	5404,8	5174,2	4,10	4,07	518,07	213,40
	2	3936,0	3723,5	5,90			
	3	1127,1	1106,1	2,22			
90	1	9541,5	8968,8	6,03	-5,83	408,47	265,73
	2	4463,6	4279,6	4,02			
	3	1737,4	1717,7	-27,52			
110	1	11930,9	11465,9	3,76	-2,16	276,33	242,90
	2	6401,1	6137,6	3,42			
	3	1495,4	1558,5	-13,65			

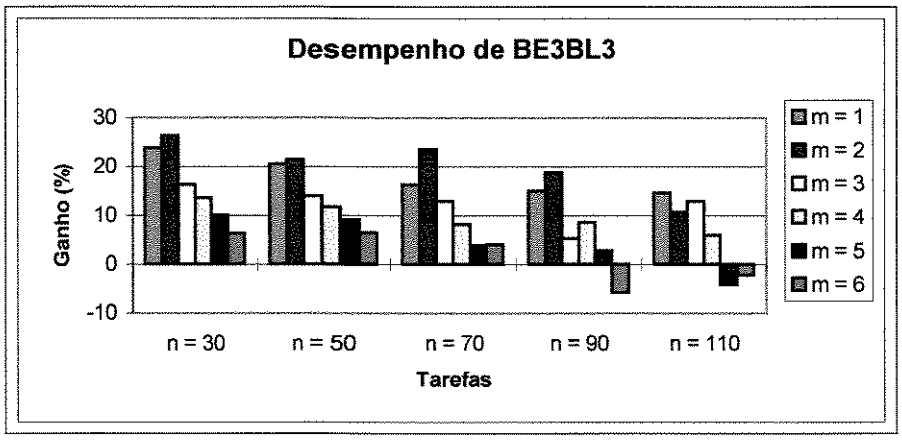

Figura 6.4 - Desempenho médio de BE3BL3

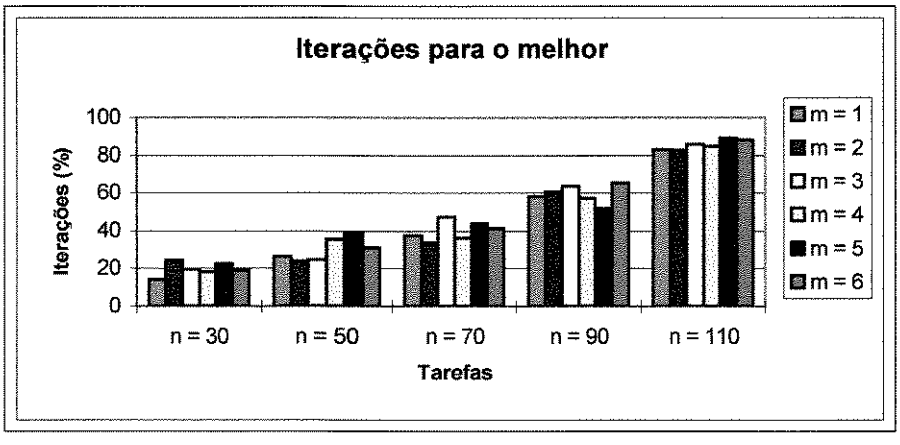

Figura 6.5 - Número de iterações de BE3 utilizadas para encontrar a melhor solução

Nota-se que BE3BL3 apresenta um bom ganho com relação às heurísticas ATCS e SAA para problemas com até 70 tarefas. A fim de complementar a informação do ganho médio, calculou-se a taxa de sucessos de BE3BL3 para cada valor de m. Essa taxa mostra que

BE3BL3 apresenta melhores resultados que a heurística ATCS para 94,67% dos problemas com m = 1 e supera a heurística SAA para 96% dos problemas com m = 2 e m = 3, 92% dos problemas com m = 4, 85,33% dos problemas com m = 5 e 78,67% dos problemas com m = 6. Essas taxas de sucesso mostram que, para a grande maioria dos problemas, BE3BL3 apresenta bons resultados.

Entretanto, apesar do esforço feito para fazer a Busca por Espalhamento um pouco mais robusta com relação à variação do número de máquinas, observa-se que, quando esse número aumenta, o ganho de BE3BL3 é um pouco menor que o de GEN1. As diferenças de ganho entre esses dois métodos foram calculadas e são mostradas na Figura 6.6. As taxas de sucesso de BE3BL3 com relação a GEN1, ou seja a porcentagem de problemas em que BE3BL3 apresenta resultados melhores que os de GEN1, foram calculadas e são iguais a 62% para m = 1, 59,33% para m = 2 e m = 3, 40% para m = 4, 47,33% para m = 5 e 35,33% para m = 6. Essas taxas mostram que BE3BL3 ainda não é tão robusta com relação à variação do número de máquinas e que um esforço de desenvolvimento ainda é necessária para fazê-la conseguir bons resultados para valores maiores de m.

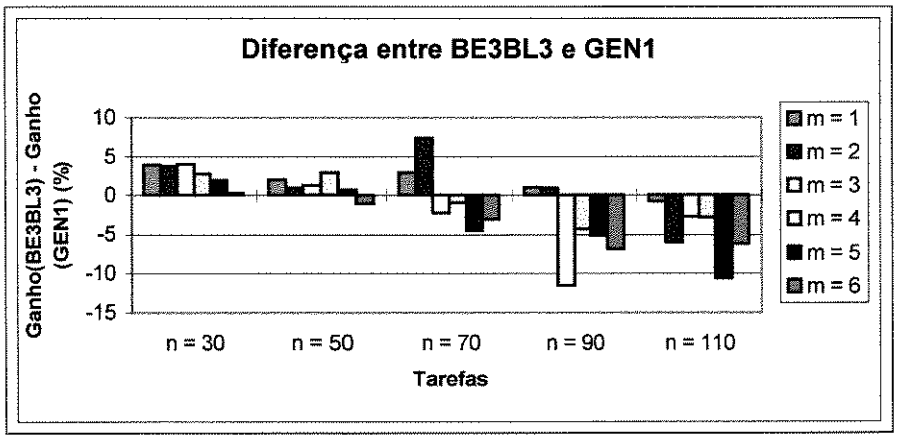

Figura 6.6 - Diferenças de Ganho entre BE3BL3 e GEN1

Quanto à porcentagem do total de iterações que BE3BL3 gasta para encontrar a melhor solução, observa-se que a adição de busca local ao procedimento acelera a convergência da busca. Nota-se que o percentual de iterações necessário para encontrar a solução final diminui de BE3 para BE3BL3, apesar do número total de iterações ter aumentado.

Na seção seguinte é apresentada uma análise do comportamento de BE3BL3, análoga à apresentada na seção 3.3.3.

6.8 - COMPORTAMENTO DE BE3BL3

A primeira característica de comportamento a ser analisada é o coeficiente de variação do ganho de BE3BL3, calculado para cada conjunto de problemas de teste com o mesmo número de máquinas e tarefas. Na Figura 6.7 apresenta-se um gráfico com os coeficientes calculados.

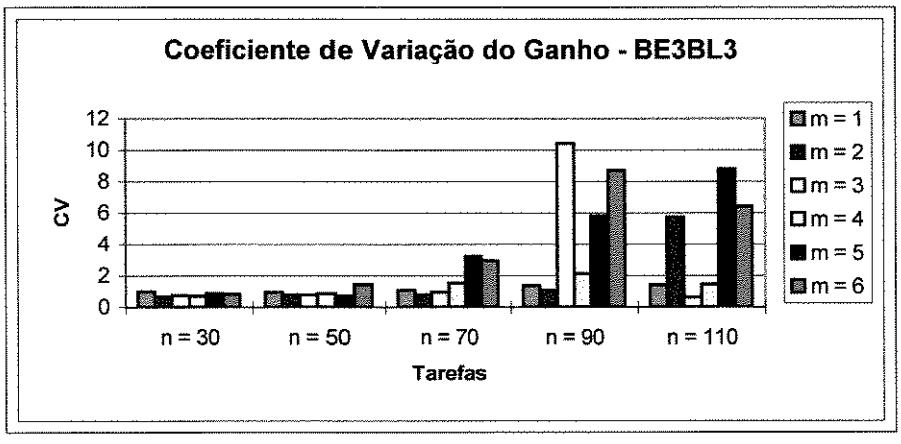

Figura 6.7 - Desvios padrão de Ganho para BE3BL3

Observa-se que os valores mostrados na Figura 6.7 são bastante altos. Isso mostra que existe uma grande variação de desempenho de BE3BL3 mesmo para os problemas em que esse desempenho é bom. Essa variação é devida, em grande parte à ocorrência de problemas para os quais o valor final de atraso encontrado pelas heurísticas ATCS e SAA ou por BE3BL3 é baixo. Para esses problemas, uma pequena variação no valor de atraso causa variações percentuais muito grandes no valor de Ganho e isso faz com que o desvio padrão possua valores altos para todo o conjunto de problemas testados.

Outra análise de comportamento realizada em BE3BL3 consistiu em se resolver alguns problemas com m = 4 e n = 110 para observar a evolução da qualidade das soluções pertencentes aos conjuntos H_t e P_t. Os gráficos das figuras 6.8 e 6.9 mostram o valor do atraso da solução de melhor custo em H_t e P_t, respectivamente, ao longo das iterações. Nas figuras 6.10 e 6.11 mostra-se a evolução do melhor custo, custo médio e mediana para esses conjuntos.

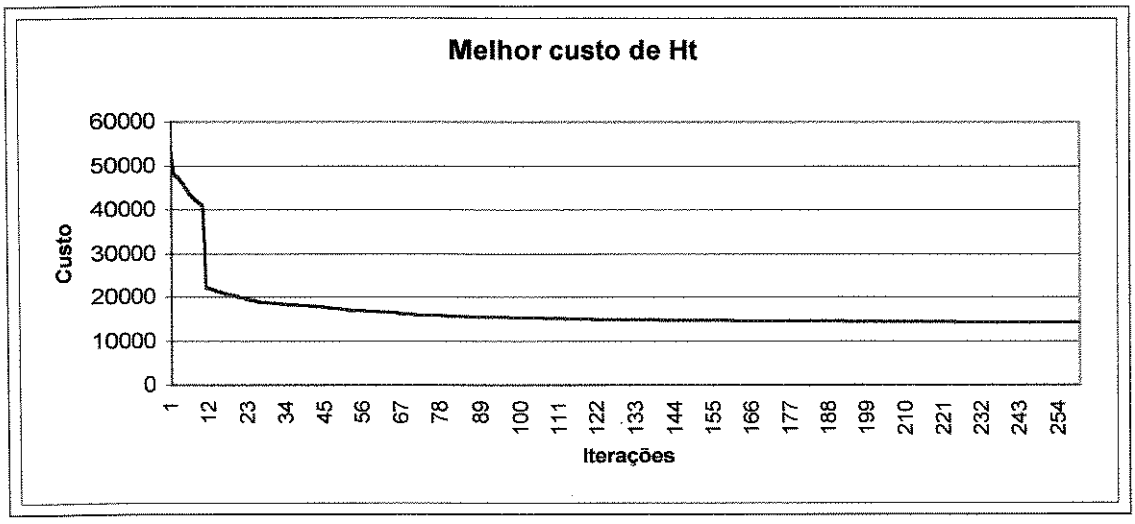

Figura 6.8 - Evolução do melhor custo em H_t para BE3BL3

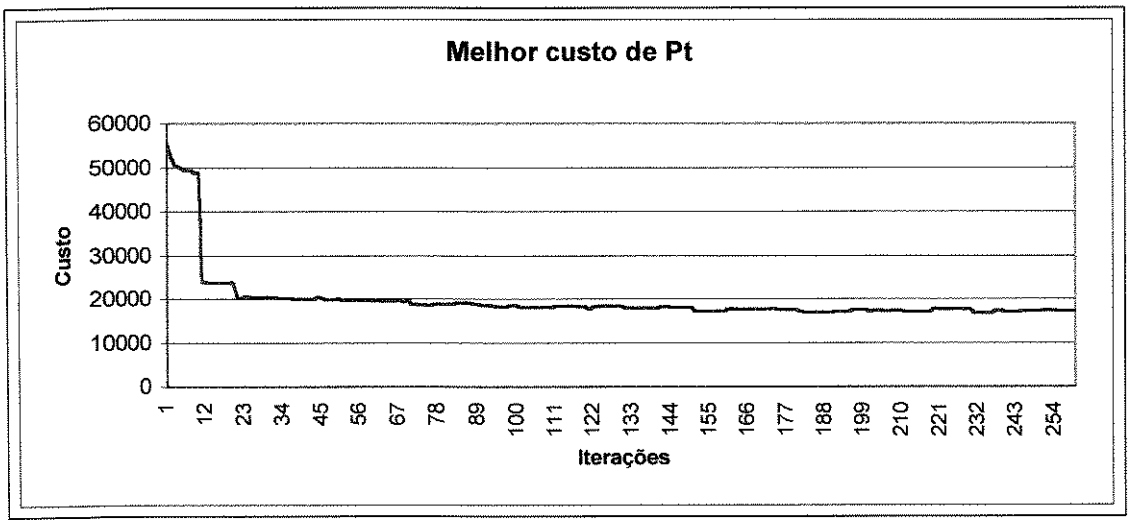

Figura 6.9 - Evolução do melhor custo em P_t para BE3BL3

O melhor custo de H_t apresenta uma rápida evolução nas primeiras iterações, ocorrendo poucas melhorias nas iterações finais. A Figura 6.8 mostra claramente a estagnação da evolução da melhor solução apresentada por BE3BL3. A rapidez da convergência inicial mostra a eficácia da combinação da Busca por Espalhamento com a busca local. Entretanto, pode-se concluir que a inclusão de estratégias de diversificação apropriadas poderiam auxiliar a busca por melhores soluções em regiões do espaço de busca ainda não exploradas. Já para o conjunto P_t, observa-se que o custo de sua melhor solução não é estritamente decrescente ao longo das iterações. Como as soluções desse conjunto são escolhidas por critérios estruturais, esse comportamento já era esperado. Outro comportamento verificado e que também era esperado é a grande proximidade entre o melhor custo, a mediana e o custo médio das soluções do conjunto H_t, mostrada na Figura 6.10. Essa proximidade pode ser decorrência da aplicação da busca local, pois ela trabalha como um

componente de intensificação em torno das soluções em $D(H_t \cup P_t)$.

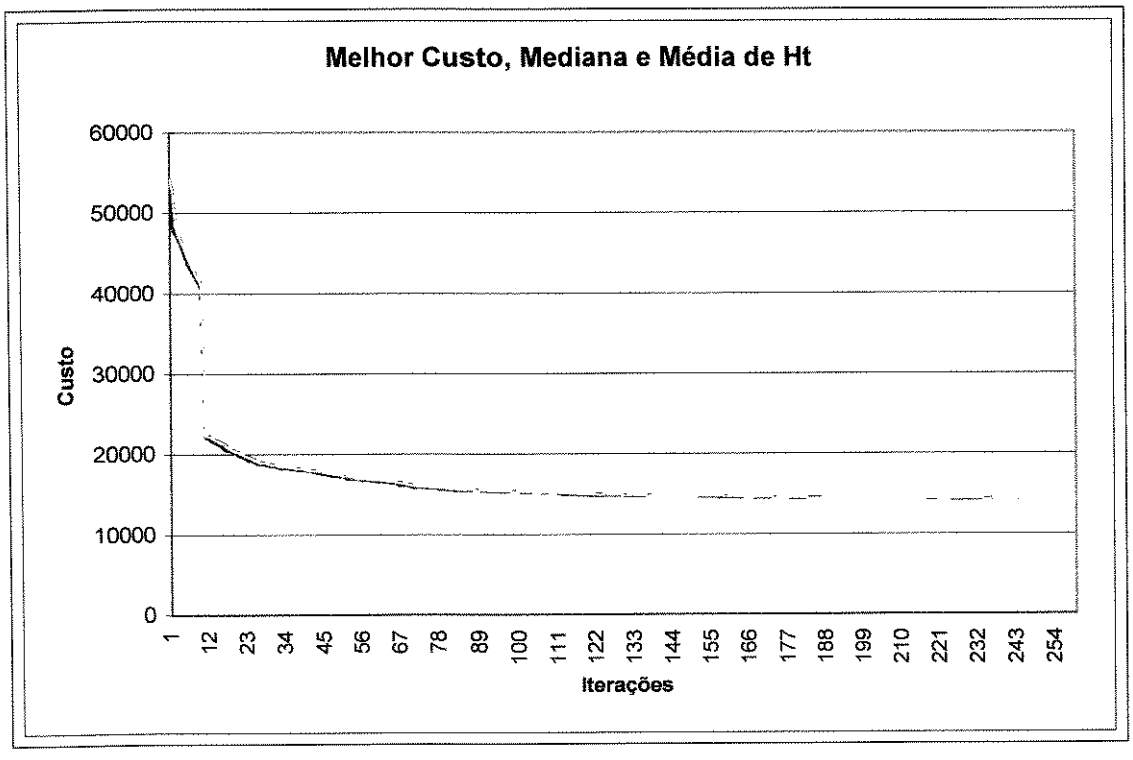

Figura 6.10 - Melhor custo, Custo Médio e Mediana em H_t para BE3BL3

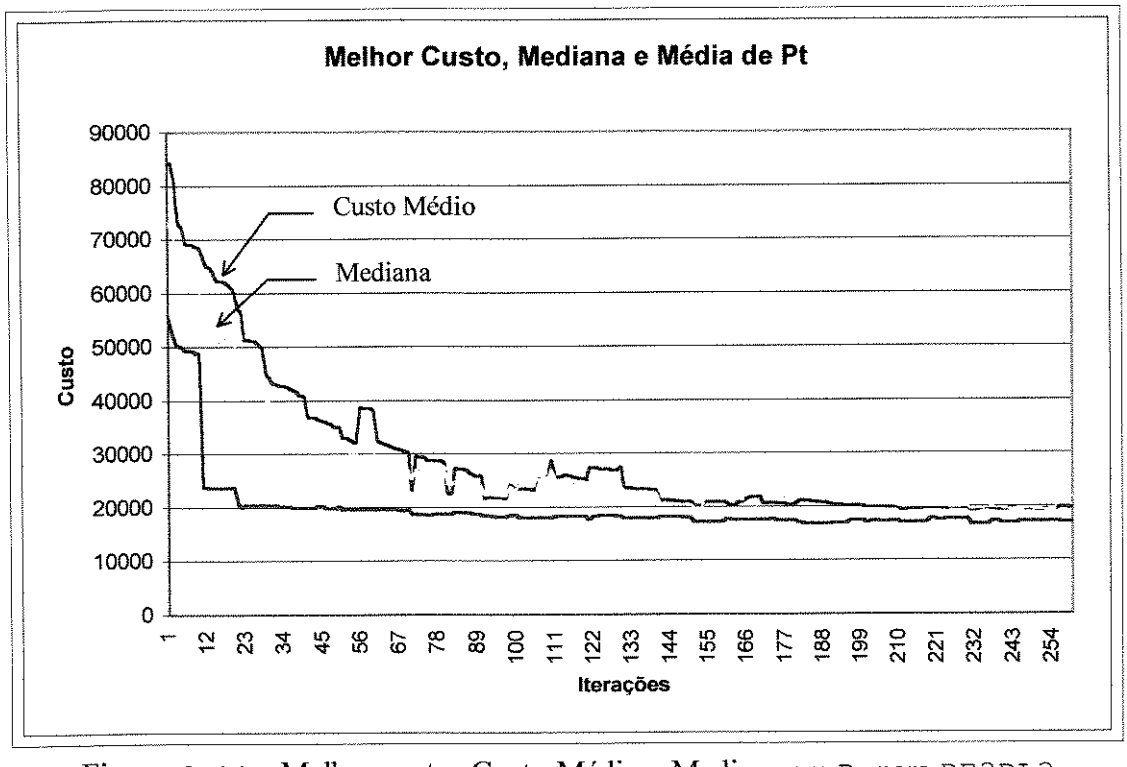

Figura 6.11 - Melhor custo, Custo Médio e Mediana em P_t para BE3BL3

Já para P_t, a mediana e o custo médio possuem a mesma tendência de evolução que o

melhor custo, ficando um pouco acima deste. As variações observadas nesses valores, na Figura 6.11, indicam que a maior parte das soluções desse conjunto são bastante diversificadas, mostrando o efeito da regra de construção de P_t.

CAPÍTULO 7 - CONCLUSÕES

Neste trabalho foram apresentadas implementações das meta-heurísticas Algoritmos Genéticos e Busca por Espalhamento para o problema de programação da produção em máquinas paralelas idênticas para a minimização do atraso total no processamento das tarefas. Considera-se que as máquinas estejam sujeitas a tempos de preparação dependentes da seqüência de processamento e que as datas de entrega das tarefas sejam distintas.

O objetivo principal do trabalho não foi o de desenvolver o melhor algoritmo heurístico para a resolução do problema mas investigar como o desempenho e o comportamento das meta-heurísticas estudadas variam em decorrência das decisões tomadas para sua implementação.

No Capítulo 1, foi feito um breve estudo do problema de programação da produção proposto. Na primeira parte desse capítulo, estudou-se a equivalência do problema PST e o Problema do Caixeiro Viajante (PCV). Entretanto, não foi possível adaptar métodos desenvolvidos para o PCV ao PST pois a grande maioria dos algoritmos encontrados na literatura para o PCV tratam do problema de minimização do comprimento total do circuito percorrido, equivalente ao "makespan". Infelizmente, para o problema de minimização do atraso total, as heurísticas existentes e os limitantes inferiores utilizados atualmente não são adequados.

Dado que a equivalência com o PCV não pôde ser explorada para a obtenção de limitantes superiores (ou inferiores) para os problemas de teste, o próximo passo foi a avaliação de heurísticas específicas para o PST, que pudessem gerar limitantes superiores para esses problemas. Para o PST com uma máquina foi escolhida a heurística ATCS (LEE *et al.*, 1997) e para o PST com múltiplas máquinas, escolheu-se a heurística SAA (GUINET, 1991). Os valores de custo dados por essas heurísticas para os problemas de testes foram utilizados para avaliar o desempenho das implementações de Algoritmos Genéticos e Busca por Espalhamento ao longo de todo o trabalho.

Também no Capítulo 1 foi apresentado um estudo de Enumeração Completa sobre problemas de pequeno porte. Esse estudo teve como objetivo investigar o grau de dificuldade do problema PST. Concluiu-se, através dos testes computacionais, que o problema PST pode apresentar um grande desafio para o desenvolvimento de métodos heurísticos. Para problemas

com 2 máquinas e 7 tarefas, observou-se que menos de 1% das soluções possuem um desvio percentual menor que 10% com relação ao custo ótimo. O aumento do número de máquinas no problema parece torná-lo ligeiramente menos difícil, proporcionando um pequeno aumento no número de soluções com desvio menor que 10% com relação à solução ótima. Para problemas com 3 máquinas e 7 tarefas, esse número passa para 1,2%. Infelizmente, quando o número de tarefas aumenta de 7 para 8, conservando-se o número de máquinas igual a 2, nota-se uma queda significativa na quantidade de soluções próximas da solução ótima. Apenas 0,4% das soluções factíveis possuem um desvio menor que 10% para o custo ótimo.

Os capítulos 2 e 3 foram dedicados à meta-heurística Algoritmos Genéticos. No capítulo 2 foram apresentadas as características das duas implementações desenvolvidas, GEN1 e GEN2. Essas duas implementações são bastante simples e, tanto os operadores genéticos escolhidos como a estrutura externa de cada implementação, foram adaptados a partir de procedimentos propostos na literatura. Apesar disso, a forma como esses operadores são aplicados às soluções do problema PST representam uma das contribuições desse trabalho. Os testes computacionais realizados com essas duas implementações sobre os problemas de teste foram apresentados no capítulo 3 e mostram que ambas apresentam uma qualidade superior às heurísticas escolhidas para comparação. Entretanto, a segunda implementação, GEN2, mostrou-se um pouco inferior, não se mostrando muito robusta quando o número de tarefas aumenta. Ambas as implementações apresentam variações de desempenho bastante características. Quando o número de máquinas e tarefas aumenta, o ganho com relação às heurísticas cai mas, deve-se analisar essa queda considerando-se o aumento do número de máquinas e o de tarefas em separado pois esses dois parâmetros possuem efeitos diferentes no valor final dos custos das soluções dadas pelas heurísticas de comparação.

Essas heurísticas são constituídas, basicamente, de um procedimento construtivo seguido de uma fase de busca local. Quando são resolvidos problemas com poucas máquinas, o impacto de um erro eventual devido à tomada de decisões no processo construtivo possui um impacto maior no valor do custo da solução final pois cada uma das máquinas processa um percentual alto do total de tarefas. Quando o número de máquinas aumenta, o impacto desses erros é um pouco menor e, além disso, eles podem ser corrigidos com mais facilidade pela aplicação de busca local. Dessa forma, pode-se esperar que o desempenho das heurísticas de comparação seja melhor para problemas com menos tarefas e mais máquinas. Isso explica porque as implementações de Algoritmos Genéticos apresentam ganhos maiores com relação às heurísticas para problemas com menos máquinas.

O aumento do número de tarefas também pode prejudicar o desempenho das heurísticas de comparação. Entretanto, quando ocorre esse aumento, o ganho dos Algoritmos Genéticos com relação a essas heurísticas cai. Uma possível explicação para essa queda consiste do seguinte. Quando o número de tarefas é muito grande, o valor de custo das

soluções do problema aumenta, assim como o número de tarefas atrasadas. Isso cria um problema de escala no cálculo do ganho percentual pois a melhoria obtida nos custos das soluções com a aplicação de GEN1 e GEN2 torna-se menos significativa com relação ao custo das soluções obtidas pelas heurísticas, conforme esses custos aumentam em magnitude.

A evolução das soluções das duas implementações de Algoritmos Genéticos foi analisada através de testes de comportamento, descritos no final do capítulo 3. Esses testes mostram que GEN1 possui características evolutivas mais interessantes que as de GEN2, o que pode explicar seu desempenho superior. Observou-se que a convergência de GEN1 é mais rápida e que essa implementação é mais robusta que GEN2 com relação aos números aleatórios gerados durante o processo de busca.

Neste ponto, deve-se ressaltar que muitas outras implementações poderiam ser sugeridas, além de GEN1 e GEN2. Na literatura relativa aos Algoritmos Genéticos existe uma grande quantidade de operadores de recombinação diferentes dos utilizados neste trabalho, assim como representações alternativas para as soluções do problema. Dessa forma, uma das possíveis ramificações desse trabalho, como sugestão de pesquisas futuras, consiste do teste de outras implementações de Algoritmos Genéticos. Além disso, pode-se testar implementações de Algoritmos Meméticos, incluindo procedimentos de Busca Local estrategicamente projetados às implementações sugeridas, para auxiliar o processo de obtenção de boas soluções.

Nos três últimos capítulos deste trabalho foi estudada a meta-heurística Busca por Espalhamento. Essa meta-heurística, ao contrário dos Algoritmos Genéticos, ainda não foi bastante explorada pois existem poucos trabalhos na literatura em que ela tenha sido aplicada a problemas de programação da produção. No capítulo 4, apresentou-se uma interpretação da Busca por Espalhamento baseada nos trabalhos de GLOVER (1994a) e GLOVER e LAGUNA (1997). Basicamente, a Busca por Espalhamento consiste em se manter uma população de soluções de referência, a partir das quais devem ser obtidas novas soluções utilizando-se procedimentos determinísticos de combinação de soluções. As populações de cada iteração são selecionadas segundo critérios de qualidade dados pelos valores de custo e pelas características estruturais das soluções obtidas nas iterações anteriores. As implementações de Busca por Espalhamento apresentadas neste trabalho representaram tentativas de se explorar ao máximo as características propostas para o método e investigar seu potencial de aplicação a problemas de programação da produção como o PST.

A partir da interpretação dada ao método, foi proposta a primeira implementação de Busca por Espalhamento, chamada de BE1. Os testes computacionais realizados com essa implementação mostraram que ela apresentava ganhos muito baixos com relação às heurísticas de comparação e, analisando o processo de busca realizado por BE1, levantou-se a hipótese de que seu baixo desempenho pudesse ser resultante da escolha de uma métrica pobre para a avaliação da qualidade estrutural das soluções e do pequeno número de iterações

realizadas devido ao tempo excessivo gasto com o procedimento de combinação de soluções escolhido.

Na tentativa de sanar ambos os problemas, foi proposta uma segunda implementação, apresentada no capítulo 5. Nessa implementação, chamada de BE2, a métrica de avaliação estrutural foi escolhida de forma a capturar informações relacionadas com os aspectos de designação de tarefas a máquinas e de seqüenciamento de tarefas em cada máquina. Além disso, o procedimento de combinação de soluções foi alterado de forma a gerar menos soluções filhas a partir de cada par de soluções pais, reduzindo o tempo computacional envolvido no processo. Infelizmente, o ganho de tempo proporcionado por essa modificação foi contrabalançado por um consumo excessivo de tempo para o cálculo da métrica baseada em designação de tarefas a máquinas. Sendo assim, BE2 não apresentou um bom desempenho e também foi descartada.

Na terceira implementação de Busca por Espalhamento, BE3, procurou-se reunir as características mais interessantes de BE1 e BE2 de forma que cada iteração da busca fosse realizada rapidamente, permitindo ao algoritmo realizar um número maior de iterações. Utilizou-se a métrica de avaliação estrutural de BE1, mais rápida de ser calculada, e o procedimento de combinação de soluções de BE2. Os testes computacionais feitos com BE3 mostraram que a combinação dessas características foi bastante feliz pois os resultados obtidos com BE3 foram superiores aos das duas primeiras implementações. Entretanto, esses resultados ainda foram inferiores aos obtidos por GEN1, o que motivou a inserção de um procedimento de busca local em BE3, obtendo-se a implementação BE3BL3.

Nessa última implementação de Busca por Espalhamento, a busca local foi aplicada a cada dez iterações sobre um conjunto de soluções selecionado de acordo com a métrica de avaliação estrutural dada pela designação de tarefas a máquinas. Os resultados dos testes computacionais mostraram que essa implementação apresenta resultados bastante satisfatórios, apesar de ainda não superar GEN1 para todos os tamanhos de problema.

Salienta-se, novamente, que as implementações de Busca por Espalhamento apresentadas representam tentativas iniciais de aplicação do método a problemas de programação da produção. Dos resultados obtidos, pode-se concluir que, com algum esforço de pesquisa adicional é possível chegar a soluções de alta qualidade. Uma das características observadas, principalmente quando se comparam os desempenhos de BE3 com BE3BL3 é que a intensificação proporcionada pelo procedimento de busca local foi capaz de aumentar bastante a qualidade dos resultados obtidos. Sugere-se, como continuação desse trabalho a simplificação gradativa de alguns procedimentos que podem estar consumindo muito tempo computacional e o aperfeiçoamento de estratégias de busca local, que poderiam fazer a Busca por Espalhamento mais agressiva do ponto de vista de intensificação da busca em torno de boas soluções.

Dentre as simplificações e modificações possíveis podem-se citar as seguintes. Em

primeiro lugar, a forma como se realizam as combinações de soluções através da introdução gradativa de informação de uma solução pai na outra faz com que as soluções filhas obtidas estejam muito próximas das soluções pais. Seria interessante desenvolver ou adaptar algum procedimento de combinação que possibilitasse a obtenção de uma solução filha com características de ambas as soluções pais mas que, ao mesmo tempo, estivesse a uma distância razoável das mesmas. Uma forma de se estudar se isso ajudaria a melhorar o desempenho da Busca por Espalhamento é adaptar alguns operadores de recombinação desenvolvido para os Algoritmos Genéticos e realizar testes comparativos.

Outra modificação relacionada à combinação de soluções seria considerar mais de um par de soluções como pais. Em BE1 foram utilizadas todas as soluções do conjunto de referência para obter soluções tentativas para a próxima iteração. Em BE2 e BE3, as combinações foram realizadas a partir de pares de soluções para que pudessem ser feitas com mais velocidade. Entretanto, ainda não foi testado o efeito de se considerar três ou mais soluções pais nas combinações, o que poderia fazer com que a busca se tornasse mais eficiente.

No que diz respeito à robustez da Busca por Espalhamento, mesmo a adição de um procedimento de busca local não fez com que ela conseguisse bons resultados para problemas com um número elevado de máquinas. Tanto o procedimento de combinação de soluções atual quanto a métrica de cálculo de distância entre soluções ainda não foram capazes de abordar adequadamente o aspecto de designação de tarefas a máquinas. Portanto, uma sugestão de continuação para esse trabalho seria o desenvolvimento e testes de novas métricas de distância que levassem esse aspecto em consideração mas que não fossem tão caras computacionalmente como a proposta no Capítulo 5.

Além disso, a interpretação dada à Busca por Espalhamento, no Capítulo 4, apesar de estar bem próxima do modelo idealizado por GLOVER (1998), apresenta algumas diferenças com relação a esse modelo, principalmente no que se refere à agressividade da busca. No modelo sugerido, todas as novas soluções obtidas devem ser submetidas a um processo de melhoria antes de se tornarem candidatas a integrar a próxima população. Talvez a aplicação de busca local a todas as soluções obtidas possa representar um gasto de tempo computacional excessivo, mas a idéia de se utilizar procedimentos de melhoria com mais freqüência e de forma estratégica durante a Busca por Espalhamento pode levar a implementações mais eficientes que as apresentadas neste trabalho.

Uma última sugestão com relação à Busca por Espalhamento diz respeito à hibridização desse método com outras meta-heurísticas como, por exemplo, a Busca Tabu. O uso de estruturas de memória para guardar a história da busca e a utilização dessa informação para estabelecer estratégias de combinações ou de melhoramento de soluções pode ter um potencial de aplicação nos problemas de programação da produção muito maior que o da Busca por Espalhamento por si só.

REFERÊNCIAS BIBLIOGRÁFICAS

ARMENTANO, V. A., RONCONI, D. P., 1998. Tabu Search for Total Tardiness Minimization in Flow Shop Scheduling Problems, aceito para publicação no *Computers & Operations Research.*

BEASLEY, J. E., CHU, P. C., 1995. A Genetic Algorithm for the Set Covering Problem, Working Paper, The Management School, Imperial College, Londres, Inglaterra.

BERNARDO, J. J., LIN, K.-S., 1994. An Interactive Procedure for Bi-criteria Production Scheduling, *Computers & Operations Research*, v 21, pp. 677-688.

BLAZEWICZ, Jacek, DROR, Moshe, WEGLATZ, Jan, 1991. Mathematical programming formulations for machine scheduling: a survey, *European Journal of Operational Research*, v 51, pp. 283-300.

CARLTON, William B., BARNES, J. Wesley, 1996. Solving the Traveling-Salesman Problem with Time Windows Using Tabu Search, *IIE Transactions*, v 28, pp. 617-629.

CHENG, T. C. E., SIN, C. C. S., 1990. A State-of-the-Art Review on Parallel Machine Scheduling Research, *European Journal of Operational Research*, v 47, pp. 271-292.

DÍAZ, Adenso, GLOVER, Fred, GHZIRI, Hassan M., GONZALEZ, J. L., LAGUNA, Manuel, MOSCATO, Pablo, TSENG, Fan T., 1996. *Optimización Heurística y Redes Neuronales*, editorial Paraninfo, Madrid.

DU, J., LEONG, J. Y.-T., 1990. Minimizing Total Tardiness on One Machine is NP-hard, *Mathematics of Operations Research*, v 15, pp. 483-495.

EGLESE, R. W., 1990. Simulated Annealing: a tool for operational research, *European Journal of Operational Research*, v 46, pp. 271-281.

EVANS, James R., 1987. Structural analysis of local search heuristics in combinatorial optimization, *Computers and Operations Research*, v 14, n 6, pp. 465-477.

FLEURENT, Charles, GLOVER, Fred, MICHELON, Philippe, VALLI, Zulficar, 1996. A Scatter Search Approach for Unconstrained Continuous Optimization, Working Paper, College of Business, University of Colorado at Boulder.

FRANÇA, P. M., GENDREAU, M., LAPORTE, G., MÜLLER, F. M., 1995. The m-Traveling Salesman Problem with Minmax Objective, *Transportation Science*, v 29, n 3, pp. 267-275.

FRIEZE, A., KARP, R. M., REED, B., 1995. When is the Assignment Bound Tight for the Asymmetric Traveling Salesman Problem?, *SIAM Journal on Computing*, v 24, n 3, pp. 484-.

GELDERS, Ludo F., VAN WASSENHOVE, Luke N., 1981. Production Planning: a review, *European Journal of Operational Research*, v 7, pp. 101-110.

GLOVER, Fred, 1977. Heuristics for Integer Programming using Surrogate Constraints, *Decision Sciences*, v 8, pp. 156-166.

GLOVER, Fred, 1994a. Tabu search for Nonlinear and Parametric Optimization (with links to Genetic Algorithms), *Discrete applied mathematics*, v 49, pp. 231-255.

GLOVER, Fred, 1994b. Genetic Algorithms and Scatter Search: Unsuspected Potentials, *Statistics and Computing*, v 4, pp. 131-140.

GLOVER, Fred, 1998. A Template for Scatter Search and Path Relinking, Working Paper, Graduate School of Business, University of Colorado, USA.

GLOVER, Fred, 1995. Tabu Search: fundamentals and uses, Working paper, Graduate School of Business, University of Colorado, CO, USA.

GLOVER, Fred, KELLY, James P., LAGUNA, Manuel, 1996. New Advances and Applications of Combining Simulation and Optimization, Working Paper, Graduate School of Business, University of Colorado at Boulder.

GLOVER, Fred, LAGUNA, Manuel, 1997. Tabu Search, Kluwer, Boston.

GOLDBERG, David E., 1989. *Genetic algorithms in search, optimization, and machine learning*, Addison-Wesley, Reading, Massachussets.

GUINET, A., 1991. Textile Production Systems: A Succession of Non-Identical Parallel Processor Shops, *Journal of the Operations Research Society*, v 42, n 8, pp. 655-671.

GUINET, A., 1993. Scheduling Sequence-Dependent Jobs on Identical Parallel Machines to Minimize Completion Time Criteria, *International Journal of Production Research*, v 31, n 7, pp. 1579-1594.

GUINET, A., 1995. Scheduling Independent Jobs on Uniform Parallel Machines to Minimize Tardiness Criteria, *Journal of Intelligent Manufacturing*, v 6, pp. 95-103.

HETTMANPERGER, T. P.,1984. *Statistical Inference Based on Ranks*, John Wiley & Sons, New York.

HOLLAND, J. H., 1995. *Adaptation in Natural and Artificial Systems*, MIT Press, Cambridge.

KAWAGUCHI, T., KYAN, S., 1988. Deterministic Scheduling in Computer Systems: a Survey, *Journal of Operations Research Society of Japan*, v 31, pp. 190-216.

KIRKPATRICK, S., GELATT Jr, C. D., VECCHI, M. P., 1983. Optimization by Simulated Annealing, *Science*, v 220, n 4598, pp. 671-680.

LAWLER, E. L., LENSTRA, J. K., RINNOOY KAN, A. G. H., SHMOYS, D. B., 1985. *The Traveling Salesman Problem - A Guided tour of Combinatorial Optimization*, John Wiley & Sons, New York.

LEE, Y. H., KIM, S., 1993. Neural Network Application for Scheduling Jobs on Parallel Machines, *Computers & Industrial Engineering*, v 25, pp. 227-230.

LEE, Y. H., BHASKARAM, K. e PINEDO, M., 1997. A Heuristic to Minimize the Total Weighted Tardiness with Sequence-Dependent Setup Times, *IIE Transactions*, v29, n 1, pp. 45-52.

LEE, Y. H., PINEDO, M., 1997. Scheduling Jobs on Parallel Machines with Sequence-Dependent Setup Times, *European Journal of Operational Research*, v 8, pp. 464-474.

LEHMANN, E. L., D'ABRERA, H. J. M., 1975. *Nonparametrics - Statistical Methods Based on Ranks*, Holden-day Inc., San Francisco, McGraw-Hill International Book Company, New York.

MACCARTHY, B. L., LIU, Jiyin, 1993. Addressing the gap in scheduling research: a review of optimization and heuristic methods in production scheduling, *International Journal of Production Research*, v 31, n 1, pp. 59-79.

MICHALEWICZ, Zbigniew, 1992. *Genetic algorithms + Data structures = Evolution programs*, Springer-Verlag, Berlin.

MICTHELL Jr. F. H., 1991. *CIM Systems: An introduction to computer-integrated manufacturing*, Prentice Hall, Englewood Cliffs, New Jersey.

MILLER, D. L., PEKNY, J. F., 1991. Exact Solution of Large Asymmetric Traveling Salesman Problems, *Science*, v 251, n 4995, pp. 754-761.

MOSCATO, Pablo, 1989. On evolution, search, optimization, genetic algorithms and martial arts - Towards memetic algorithms, Caltech Concurrent Computation Program Report 826, Caltech, Pasadena, CA, USA.

MURATA, T., ISHIBUCHI H., TANAKA H., 1996. Genetic Algorithms for Flowshop Scheduling Problems, *Computers & Industrial Engineering*, v 30, pp. 1061-1071.

NAWAZ, Muhammad, ENSCORE E. Emory, Jr., HAM, Inyong, 1983. A Heuristic for the m-machine, n-problem Flowshop Sequencing Problem, *OMEGA*, v 11, n 1, pp. 91-95.

PAPADIMITRIOU, Christos H., STEIGLITZ, Kenneth, 1982. *Combinatorial Optimization: algorithms and complexity*, Prentice-Hall, Englewood Cliffs, New Jersey, USA.

PEKNY, J. F., MILLER, D. L., McRAE, G. J., 1990. An Exact Parallel Algorithm for Scheduling when Production Costs Depend on Consecutive System States, *Computers & Chemical Engineering*, v 14, n 9, pp. 1009-1023.

RAGATZ, G. L., 1989. Scheduling to Minimize Tardiness on a Single Machine with Sequence Dependent Setup Times, Working Paper, Department of Management, Michigan State University, EUA.

RAJGOPAL, J. BIDANDA, B., 1991. On Scheduling Parallel Machines with Two Setup Classes, *International Journal of Production Research*, v 29, pp. 2443-2458.

RUBIN, P. A., RAGATZ, G. L., 1995. Scheduling in a Sequence Dependent Setup Environment with Genetic Search. *Computers & Operations Research*, v 22, pp. 85-99.

TAN, K. C., NARASIMHAN, R., 1997. Minimizing Tardiness on a Single Processor with Sequence-dependent Setup Times: A Simulated Annealing Approach, *OMEGA*, v 25, n 6, pp. 619-634.

WOODRUFF, D., ZEMEL, E., 1993. Hashing Vectors for Tabu Search, *Annals of Operations Research*, v 41, pp. 123-137.

XU, Jiefeng, CHIU, Steve Y., GLOVER, Fred, 1996. Fine-Tuning a Tabu Search Algorithm with Statistical Tests, Working Paper, University of Colorado at Boulder, Boulder, Colorado, EUA.