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

OTIMIZAÇÃO BI-OBJETIVO PARA O PROBLEMA DE SEQÜENCIAMENTO DE TAREFAS EM UMA MÁQUINA COM TEMPOS DE PREPARAÇÃO DEPENDENTES DA SEQÜÊNCIA

Rodrigo Moreira Carvalho

Orientador: Vinícius Amaral Armentano

Dissertação apresentada à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas – UNICAMP, como parte dos requisitos exigidos para obtenção do título de Mestre em Engenharia Elétrica.
Área de concentração: Automação.

Banca Examinadora:

Vitória Maria Miranda Pureza – UFSCAR / São Carlos
Paulo Augusto Valente Ferreira – FEEC/UNICAMP
Paulo Morelato França – FEEC/UNICAMP
Vinícius Amaral Armentano – FEEC/UNICAMP

maio de 2002

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

BIB ID 295248

C253o

Carvalho, Rodrigo Moreira
Otimização bi-objetivo para o problema de seqüenciamento de tarefas em uma máquina com tempos de preparação dependentes da seqüência / Rodrigo Moreira Carvalho.--Campinas, SP: [s.n.], 2002.

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

1. Otimização combinatória. 2. Escalonamento de produção. 3. Processo decisório por critério multiplo. 4. Programação heurística. 5. Problema do caixeiro viajante. I. Armentano, Vinícius Amaral. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

RESUMO

A área de otimização combinatória multiobjetivo tem despertado crescente interesse pela sua importância prática e pela necessidade de desenvolver métodos eficientes que forneçam uma boa aproximação das soluções ótimas de Pareto. Neste trabalho é abordado o problema de seqüenciamento de tarefas em uma máquina com tempos de processamento dependentes da seqüência, datas de entrega e duas medidas de desempenho: soma do atrasos das tarefas e tempo total para processar todas as tarefas, também chamado de *makespan*. A minimização do *makespan* é equivalente à minimização do comprimento da rota no problema do caixeiro viajante assimétrico. Diversas heurísticas construtivas para cada critério propostas na literatura foram adaptadas para gerar um conjunto inicial de soluções não dominadas. Este conjunto é utilizado como partida em um método de busca em vizinhança com múltiplos recomeços. Vários tipos de vizinhanças foram testados, bem como diferentes estratégias de implementação de uma busca local multiobjetivo. Uma versão do método é testada em problemas pequenos, onde as soluções ótimas de Pareto são obtidas por enumeração completa. Para problemas grandes testa-se o desempenho relativo de diversas versões do método, e compara-se a qualidade das soluções que minimizam cada objetivo individualmente com a qualidade das soluções geradas por algoritmos mono-objetivos propostos recentemente na literatura.

Palavras-chave: Otimização Combinatória Multiobjetivo, Seqüenciamento de Tarefas, Tempos de Preparação, Datas de Entrega, Heurística, Busca Local, Problema do Caixeiro Viajante.

ABSTRACT

The area of multiobjective combinatorial optimization has attracted the attention of researchers due to its practical importance and the need to develop efficient methods that yield a good approximation of the Pareto optimal solutions. This work addresses the sequencing of jobs in a single machine with sequence dependent setup times, due dates and two performance measures: the sum of job tardiness and the total time to process all jobs, also known as makespan. The minimization of the makespan is equivalent to the minimization of the tour length for the asymmetric traveling salesman problem. Several constructive heuristics proposed for each criterion in the literature were adapted to generate an initial set of nondominated solutions. This set is used to start a neighborhood search method with multiple restarts. Various neighborhood types were tested, as well as different strategies of implementing a multiobjective local search. A version of the method is tested for small problems, where the optimal Pareto solutions are obtained by complete enumeration. For larger problems, the relative performance of versions of the method is evaluated, and the quality of the solutions that minimize each objective is compared with the solutions generated by single-objective algorithms recently proposed in the literature.

Keywords: Multiobjective Combinatorial Optimization, Job Sequencing, Setup Times, Due Dates, Heuristics, Local Search, Traveling Salesman Problem.

DEDICATÓRIA

"O Sol e as estrelas entoam a melodia eterna: 'Tempus Fugit'.

E porque temos medo da verdade que só aparece no silêncio solitário da noite, reunimo-nos para espantar o terror, e abafamos o ruído tranqüilo do pêndulo com enormes gritarias. Contra a música suave da nossa verdade, o barulho dos rojões....

Pela manhã, seremos, de novo, o tolo Coelho da Alice:
'Estou atrasado, estou atrasado...'

Mas o relógio não desiste. Continuará a nos chamar a sabedoria:
Quem sabe que o tempo está fugindo descobre, subitamente, a beleza única do momento que nunca mais será..."

O relógio / Rubens Alves

Aos meus pais, de todo coração.

AGRADECIMENTOS

A minha família, minhas irmãs e em especial a Gabi e ao Marquinho: vocês tornam a vida cheia de cores e alegrias.

Ao Prof. Dr. Vinícius Amaral Armentano, pela orientação, paciência, amizade e confiança. Só tenho a agradecer tudo que me ensinou nestes anos de trabalho e descobertas.

Aos colegas Luciana Buriol e Alexandre Mendes, que muito me ajudaram em todo o trabalho com sua experiência na área.

Aos professores de quem fui aluno: aquele que ensina com dedicação constrói um futuro melhor.

Aos colegas do Densis, que me acompanharam durante o curso, em especial ao Eduardo, Aníbal, Luciano, Luís Fernando e Edílson. O bandejão não seria o mesmo sem vocês.

Aos amigos do futebol, e que o segundo tempo no Pantanal continue inspirando muitas dissertações e teses.

À Fernanda, que me ouviu falar muito sobre *scheduling, traveling salesman, etc.* Obrigado por tudo que cresci contigo.

Aos eternos amigos Cristiano, Ítalo e Arthur. Vocês são especiais. A verdadeira amizade não se constrói, se reconhece.

A todos os diretores, ao Marcelo, Áureo, Mônica, Leon Iapap, e todos companheiros de festa. Balada! Seqüência!

Aos funcionários da FEEC, em especial ao pessoal do suporte em informática. Sei o quanto é difícil o trabalho de vocês.

Ao CNPq, cujo apoio financeiro foi fundamental para a realização deste trabalho.

Índice

INTRODUÇÃO

O crescimento e complexidade das organizações que surgiram após a revolução industrial induziram a uma divisão do trabalho e fragmentação das responsabilidades gerenciais. Estas organizações geraram resultados espetaculares, mas o aumento da especialização fez com que em muitos casos diferentes setores tivessem metas e valores próprios, perdendo-se a visão do funcionamento da organização como um todo (Hillier e Lieberman,1988). A organização das operações destas grandes corporações fez com que se popularizassem novas metodologias, e dentre elas destacamos a abordagem científica denominada Pesquisa Operacional (PO), que visa resolver problemas de tomada de decisão através de modelos tipicamente matemáticos, sejam eles determinísticos ou probabilísticos.

Historicamente é atribuído o início da atividade em PO ao início da segunda guerra mundial, quando grupos de cientistas foram formados para auxiliar na resolução dos complexos problemas gerados nas operações militares. Posteriormente foi observada a utilidade das metodologias desenvolvidas na indústria em geral, gerando aumentos na produtividade e melhoria no funcionamento global das organizações. Pesquisas científicas vêm sendo realizadas desde então, impulsionadas pelos avanços na área de computação, na aplicação de modelos científicos a problemas de tomada de decisão na indústria.

A segmentação das atividades e a natureza de vários problemas geram conflitos de objetivos, que em PO resultaram numa nova área de pesquisa, denominada programação multiobjetivo. Inicialmente eram utilizadas adaptações de métodos para um objetivo na resolução destes problemas, como dar pesos a cada objetivo formando uma única medida a ser otimizada. Mais recentemente foram adicionadas a estas técnicas métodos desenvolvidos especificamente para lidar com estes conflitos.

Parte dos problemas abordados em PO corresponde a determinar a seqüência de realização de atividades, denominados problemas de *scheduling*, onde é comum a existência de objetivos conflitantes. Esta classe de problemas tem recebido crescente atenção pela importância prática, estrutura simples e por serem de difícil resolução. O problema de programação (*scheduling*) da produção genérico consiste em alocar recursos a tarefas ao longo do tempo otimizando um ou mais critérios, de forma a melhorar o processo de produção atingindo metas e reduzindo custos. O desenvolvimento de algoritmos

eficientes para a resolução de problemas de *scheduling* é uma área de pesquisa extensa e de vasta literatura, sendo muitos destes problemas de natureza combinatória e pertencentes à classe de problemas *NP-Hard*. Todos os algoritmos exatos conhecidos para esta classe de problema são de complexidade exponencial, ou seja, o esforço computacional cresce exponencialmente com o tamanho da instância.

Tempos de preparação dependentes da seqüência ocorrem freqüentemente na indústria, quando por exemplo é necessário preparar uma máquina para realizar uma nova tarefa, e esta preparação depende da tarefa processada anteriormente. Para simplificar a análise e resolução de problemas de *scheduling*, tempos de preparação (*setup*) tem sido muitas vezes negligenciados, afetando a qualidade das soluções encontradas (Allahverdi et al., 1998). Considerar estes fatores na determinação da seqüência de processamento em uma máquina pode ser um problema de difícil resolução, sendo objeto deste trabalho. Neste problema, diversas medidas podem ser utilizadas para avaliar a qualidade de uma solução. Estas medidas podem ser conflitantes, e freqüentemente o são. Por exemplo, a solução de menor custo para a empresa pode não ser aquela que melhor satisfaz os clientes pois gera atrasos na entrega.

O problema abordado, minimizar o atraso total e o *makespan* no seqüenciamento de tarefas em uma máquina com tempos de preparação dependentes da seqüência, é um problema multiobjetivo onde se pretende gerar aproximações para o conjunto de soluções eficientes (ver seção 1.4). A resolução deste problema é importante por possuir aplicação prática e contribuir para o estudo e resolução de problemas relacionados na área de *scheduling* e de problemas combinatórios multiobjetivo em geral. Além disto, minimizar o *makespan* é equivalente à resolução de um problema do caixeiro viajante assimétrico (PCVA – *Asymmetric Traveling Salesman Problem*), um problema muito estudado principalmente no caso simétrico, sendo considerada a adaptação de métodos para o problema do caixeiro viajante na resolução do problema de *scheduling*. Pretende-se assim avançar no conhecimento de técnicas para resolver problemas de scheduling e para problemas combinatórios multiobjetivos, duas categorias de problemas cuja resolução tem tido crescente atenção por causa de suas aplicações práticas e dificuldade de resolução.

Em muitos dos problemas práticos é necessária a utilização de heurísticas para que o esforço computacional seja compatível com o tempo de processamento (CPU) disponível.

Problemas de *scheduling* geralmente estão relacionados ao chão de fábrica, onde o horizonte de planejamento é curto, o tempo de CPU disponível é pequeno e a precisão dos dados utilizados é por vezes questionável. Heurísticas se caracterizam por buscar soluções de boa qualidade, sem necessariamente alcançar a otimalidade, e são geralmente divididas em heurísticas construtivas e de melhoria. As heurísticas construtivas geralmente encontram soluções de baixa qualidade, adicionando decisões até atingir uma solução completa. Heurísticas de melhoria partem de soluções existentes e buscam melhorar estas soluções alterando parte das decisões. É comum aplicar-se heurísticas de melhoria às soluções geradas por heurísticas construtivas, sendo este procedimento adotado neste trabalho.

Existe ainda o conceito de meta-heurística, que visa guiar heurísticas de melhoria para que explorem soluções promissoras. Exemplos de meta-heurísticas são a Busca Tabu, Algoritmos Genéticos, GRASP e *Simulated Annealing*. A aplicação de meta-heurísticas a problemas combinatórios multiobjetivo é uma tarefa que pode tornar-se complexa e tem sido objeto de estudos recentes. Entretanto, uma boa heurística de melhoria é base para uma implementação bem sucedida quer utilizem-se ou não meta-heurísticas. Assim neste trabalho optou-se por concentrar os esforços na elaboração de uma heurística de melhoria de alta qualidade, sendo utilizados múltiplos recomeços (*multiple start*) e perturbações aleatórias nas melhores soluções encontradas (ver seção 4.5). Este procedimento foi utilizado com sucesso no PCV - Problema do Caixeiro Viajante (TSP - Traveling Salesman Problem) por Johnson e McGeoch (1997), um problema bastante estudado e que possui uma estrutura semelhante ao problema de seqüenciamento abordado.

A proposta deste trabalho é resolver o problema de seqüenciamento utilizando heurísticas construtivas para gerar um conjunto de soluções inicial, um algoritmo de busca local para melhorar estas soluções e um método de recomeço (*multiple start*) para reiniciar a busca local a partir de diferentes conjuntos de soluções. São implementadas e testadas diversas possibilidades para cada uma das partes do algoritmo, buscando uma combinação que melhor aproxime o conjunto de soluções eficientes. Para avaliar a qualidade destas aproximações utilizou-se duas medidas propostas na literatura, as medidas de desvio (Daniels, 1992) e de distância (Czyac e Janszkiewicz, 1998), de forma que se possa também comparar a atuação destas medidas.

O capítulo 1 descreve o problema abordado: "Otimização bi-objetivo para o problema de seqüenciamento de tarefas em uma máquina com tempos de preparação dependentes da seqüência". Apresenta também uma revisão bibliográfica, as metodologias para sua resolução utilizadas e uma introdução à programação combinatória multiobjetivo destacando as principais diferenças com relação à otimização de um único objetivo. Por fim, são descritas as medidas utilizadas neste trabalho para avaliar a qualidade de um conjunto de soluções para o problema multiobjetivo, sendo uma medida relacionada ao desvio e outra à distância com relação a um conjunto de referência.

No capítulo 2 são descritas as heurísticas construtivas implementadas para o problema, que são adaptações de heurísticas para um único objetivo. Estas heurísticas são divididas em quatro categorias: heurísticas baseadas na regra ATCS (*Apparent Tardiness Cost with Setups*), onde é criado um índice de prioridade para cada tarefa; heurísticas de inserção, onde a cada iteração uma tarefa é inserida na seqüência, sendo testadas diferentes regras para ordenar esta inserção; *cheapest insertion*, onde são avaliadas todas as inserções possíveis antes de escolher a próxima tarefa a ser inserida na solução; e heurísticas baseadas no vizinho mais próximo, na qual a próxima tarefa na seqüência é aquela que possui menor tempo de preparação com relação à última seqüenciada.

O resultados dos testes realizados para heurísticas construtivas são apresentados no capítulo 3, sendo feita uma descrição da implementação e das instâncias utilizadas. Os resultados são apresentados separadamente para instâncias de até 14 tarefas, onde foram determinadas as soluções eficientes através de um procedimento de enumeração completa, e instâncias de até 100 tarefas, onde não se conhecem as soluções eficientes.

Descreve-se no capítulo 4 procedimentos de busca local implementados para o problema. Estes procedimentos visam melhorar o conjunto solução até atingir um conjunto localmente eficiente (ver definição 4.2, capítulo 4). São descritas: as estratégias de busca utilizadas, que determinam a forma da realizar a busca; uma proposta de preprocessamento, para que a busca mais refinada seja realizada a partir de soluções de melhor qualidade do que as geradas pelas heurísticas construtivas; as vizinhanças utilizadas; uma estratégia denominada "Estratégia r – grupos com vizinhança 3-opt variável" desenvolvida especificamente para ser utilizada com a vizinhança 3-opt implementada; e o método de múltiplos recomeços utilizado para reiniciar a busca.

No capítulo 5 são testados os procedimentos de busca local descritos no capítulo 4, de forma a comparar diferentes configurações possíveis na resolução do problema. Os testes estão divididos para avaliar isoladamente os componentes do algoritmo, realizando-se, por exemplo, um teste para as diferentes vizinhanças deixando fixa a estratégia de busca. É feita uma comparação com os resultados obtidos por enumeração completa, e também uma análise da qualidade das soluções quando se aumenta o tempo computacional disponível.

No capítulo 6 são feitas análises considerando-se individualmente os objetivos, realizando comparações com trabalhos prévios para minimizar o *makespan* ou o atraso total. As conclusões finais são apresentadas no capítulo 7.

1 Seqüenciamento em uma máquina com dois objetivos

1.1 Descrição do Problema

O problema consiste em determinar a ordem em que n tarefas devem ser processadas em uma máquina. Esta máquina precisa ser preparada antes de processar uma nova tarefa, e o tempo necessário para preparar a máquina depende da tarefa a ser processada e de qual tarefa foi processada antes desta. Além disto para cada tarefa existe uma data de entrega, que é o instante máximo em que a tarefa deveria ter seu processamento finalizado.

Existem diversas medidas que podem ser utilizadas na avaliação da qualidade de uma solução para este tipo de problema, e neste trabalho consideramos duas delas: o *makespan,* tempo necessário para processar todas as tarefas, e o atraso total, que é a soma dos atrasos de cada tarefa com relação à sua data de entrega. A minimização do *makespan* está relacionada a custos envolvidos com o tempo de operação, como por exemplo gastos com mão de obra. Já a minimização do atraso total está relacionada com a satisfação das datas de entrega, algo difícil de se quantificar em termos financeiros. Isto dificulta a utilização de uma única função objetivo que combine estas duas medidas, e portanto propõe-se uma modelagem multiobjetivo, onde cada critério de avaliação é considerado de forma independente. A seguir é apresentada uma descrição formal do problema.

Considere uma única máquina na qual devem ser processadas n tarefas. Cada tarefa i possui um tempo de processamento p_i e uma data de entrega d_i. Temos ainda uma matriz com os tempos de preparação s_{ij} para todo par (i, j) tal que i =0..n, j = 1..n, onde s_{0j} é o tempo de preparação de j dado o estado inicial da máquina.

Define-se:

- S_i : instante de início do processamento da tarefa i *(Start time);*
- C_i : instante de conclusão da tarefa i *(Completion time)*, onde $C_i = S_i + p_i$;
- T_i : atraso da tarefa i *(Tardiness)*, $T_i = max\ \{0, C_i - d_i\}$;
- M : m*akespan*, $M = max_i\{C_i\}$;
- T : soma dos atrasos *(Total Tardiness)*, $T = \Sigma T_i$.

O problema consiste na minimização da soma dos atrasos T e do m*akespan* M simultaneamente, sendo portanto um problema bi-objetivo, onde devem ser geradas as soluções eficientes para posterior escolha do decisor. As medidas de desempenho escolhidas, soma dos atrasos e m*akespan*, são consideradas medidas regulares, ou seja, uma solução não deve ter tempos ociosos entre as tarefas. Assim, qualquer solução válida pode ser descrita como uma permutação das n tarefas.

1.2 Revisão bibliográfica

Dentre os problemas de *scheduling*, os problemas que envolvem tempos de preparação dependentes da seqüência (*sequence dependent setup times*) têm grande importância em processos industriais. Estes tempos de preparação incluem o tempo para ajuste do processo de produção antes do início da execução de uma tarefa, sendo este tempo dependente da tarefa processada anteriormente. Allahverdi et al. (1999) relata que, numa pesquisa entre administradores da área industrial, 75% dos entrevistados possuíam problemas de *scheduling* com tempos de preparação dependentes da seqüência, enquanto que para 15% todas as tarefas incluíam estes tempos de preparação. Entretanto, a maior parte da literatura disponível não trata de problemas com tempos de preparação, sendo este tema pouco explorado com relação à sua relevância.

Allahverdi et al. (1999) descreve várias aplicações de modelos de seqüenciamento com tempos de processamento dependentes da seqüência, como: na fabricação de sacolas

de papel onde é necessário adaptar a máquina quando se troca o tipo de sacola, sendo o tempo de preparação dependente da similaridade entre tipos consecutivos; na indústria gráfica onde o tempo de limpeza das máquinas depende das cores utilizadas na tarefa atual e na seguinte; na indústria têxtil, onde as operações de tecer e tingir dependem da seqüência; na fabricação de caixas e garrafas, onde a preparação depende do tamanho e formato dos recipientes. Aplicações similares são também citadas nas indústrias química, farmacêutica, de alimentos, metalúrgica, de papel, semicondutores e outras.

Além destas aplicações existem diversos problemas relacionados, para os quais os conceitos desenvolvidos para o problema proposto podem ser aplicados. Por exemplo: a utilização de outras medidas de desempenho como o tempo de fluxo médio (que é a média de C_i), o seqüenciamento em máquinas paralelas ou o problema do caixeiro viajante com janelas de tempo (TSPTW – *Traveling Salesman Problem With Time Windows*) como descrito em Solomon e Desrosiers, 1988.

Um caso especial é o problema de *no-wait flowshop scheduling*, onde existem n tarefas a serem processadas em m máquinas. Cada tarefa i deve começar a ser processada na máquina k no instante em que terminar o seu processamento na máquina k-1. Kanellakis e Papadimitriou (1980) demonstram que a minimização do *makespan* é equivalente ao Problema do Caixeiro Viajante Assimétrico (PCVA – *Asymetric Traveling Salesman Problem*), e de forma análoga pode-se demonstrar que o problema é equivalente ao de seqüenciamento em uma máquina com tempos de preparação.

Tradicionalmente, a literatura tem dado maior ênfase a problemas de minimização de um único objetivo em problemas combinatórios (Gupta e Ramnarayanan, 1996, Blazewicz, 1996), sendo mesmo os problemas multiobjetivo modificados com a combinação dos objetivos em uma função escalar ou estabelecendo-se prioridades entre os objetivos, de forma que técnicas mono-objetivo possam ser aplicadas, os chamados métodos "a priori".

Recentemente, os métodos "*a posteriori*", que consistem na geração das soluções eficientes (Pareto ótimas) para posterior escolha do decisor, têm recebido maior atenção (Daniels 1990, Hansen 1997, Ishibuchi e Murata 1998, Czyac et al.1998). Métodos "*a posteriori*" são mais adequados quando não se conhecem as preferências do decisor, tendo aplicação mais abrangente e complexidade maior na resolução e na análise das soluções do

que algoritmos mono-objetivo. Mesmo a comparação de aproximações do conjunto Pareto-ótimo para um problema multiobjetivo não é trivial, tendo recebido atenção de alguns trabalhos da literatura (Daniels 1992, Hansen e Jaszkiewicz 1998), enquanto que para problemas mono-objetivo basta observar o menor valor da função objetivo (problemas de minimização).

A minimização do atraso total em uma máquina com tempos de preparação dependentes da seqüência tem sido objeto de estudo em trabalhos recentes na área de *scheduling* como: Armentano e Mazzini (2000), que propõem um algoritmo genético para o problema; França et al. (2001), que utiliza um algoritmo genético híbrido ou memético, baseado no trabalho de Mendes (1999); Lee et al. (1997), que propõem uma heurística construtiva e uma estratégia simples de busca local e Tan et al. (2000), que comparam a implementação de um método de restart aleatório, um algoritmo *branch and bound*, um algoritmo genético e uma implementação de *simulated annealing* para o problema.

Já a minimização do *makespan* corresponde à resolução de um problema do caixeiro viajante assimétrico (PCVA), onde o tempo de preparação entre as tarefas i e j somados ao tempo de processamento da tarefa j corresponde à distância de i à j. Um trabalho específico para o problema assimétrico é o de Kanellakis e Papadimitriou (1980), onde se adapta a heurística de Lin e Kernigham(1973) para o caso assimétrico. Mais recentemente, Buriol (2000) propõe um algoritmo genético híbrido para o PCVA. Já o caso simétrico, PCV, onde a distância de i a j é idêntica à distância de j a i, é um problema muito estudado na literatura e destacam-se os trabalhos de Lawler et al. (1985) e Johnson e McGeoch (1997), que reúnem diversos métodos e implementações já realizadas para este problema.

1.3 Metodologia proposta

Não se conhece nenhum trabalho que minimize simultaneamente dois objetivos para o problema de seqüenciamento em uma máquina com tempos de preparação dependentes da seqüência, buscando gerar um conjunto de soluções eficientes. Esta classe de problemas combinatórios multiobjetivo (MOCO – *Multiobjective Combinatorial Programming*) é uma área de pesquisa ainda pouco explorada, e portanto este trabalho se baseia nos métodos propostos para cada objetivo isoladamente e em trabalhos desenvolvidos para outros problemas desta classe MOCO.

O problema de seqüenciamento descrito na seção 1.1 possui, além da importância prática, relevância teórica por agregar elementos de linhas de pesquisa distintas das áreas de *scheduling* e do PCV. Algoritmos de busca local para *scheduling* geralmente se baseiam na troca de posição de tarefas, enquanto que para o caixeiro viajante a troca entre arcos é em geral mais apropriada. A metodologia proposta consiste em aplicar técnicas destas duas linhas de pesquisa na resolução do problema multiobjetivo:

- Implementar um algoritmo de enumeração completa para resolução de problemas de pequeno porte.
- Implementar heurísticas construtivas: heurísticas construtivas geram em geral uma solução sem tentar melhorá-la após a construção. A adaptação da regra ATCS (*aparent tardiness cost*, Lee et al. 1997), de algoritmos para o PCV (*farthest insertion*, *nearest insertion*, ver Lawler et al. 1985) e de heurísticas de inserção (Nawaz et al., 1983) ao problema bi-objetivo é considerada. Pretende-se adaptar estas estratégias para o problema e construir conjuntos de soluções não dominadas entre si, gerando um conjunto inicial para a aplicação de uma busca local. Uma descrição mais detalhada destas implementações é feita no capítulo 2.
- Definir a vizinhança na busca local: a definição de vizinhança é um fator crucial para o bom desempenho da busca local, sendo implementadas diferentes vizinhanças como as aplicadas para o problema do caixeiro viajante (mudança em arcos) e as aplicadas a problemas de *scheduling* (troca e inserção de tarefas).
- Definir a estratégia de busca local: propõe-se utilizar estratégias de busca local semelhante à utilizada por Baykasoglu (1999), onde se gera um conjunto de soluções S cujos vizinhos são dominados por S. São avaliadas diferentes forma de realizar a busca, modificando basicamente a ordem em que as vizinhanças são visitadas e o momento de atualizar o conjunto de soluções não-dominadas encontrado.
- Testar o uso de preprocessamento da busca local: propõe-se realizar antes da busca local uma busca de vizinhos que dominem a solução atual, visando avaliar poucos vizinhos em soluções de baixa qualidade. São comparadas versões com e sem preprocessamento da busca local.

- Aplicar o método *multiple start*: recentemente têm sido aplicadas meta-heurísticas como busca tabu, algoritmos genéticos, *simulated annealing* ou *scatter search* para problemas multiobjetivo. Esta aplicação é recente, como em Hansen (1996), Czyac et al. (1998) e Viana et al. (2000), sendo uma área de pesquisa promissora. Não se tem conhecimento da aplicação de métodos de *multiple start*, ou múltiplos recomeços, em programação multiobjetivo. Estes métodos executam a busca local a partir de diversas soluções iniciais, buscando explorar diferentes regiões do espaço de soluções. A escolha desta metodologia foi baseada em dois fatores: a simplicidade da implementação, que possibilita um investimento maior no desenvolvimento e testes da busca local, e os bons resultados obtidos por este tipo de algoritmo para o PCV, como a implementação de *Iterated Lin-Kernigham* descrita por Johnson e McGeoch (1997) onde o recomeço é feito a partir de perturbações aleatórias em ótimos locais já encontrados.

Uma vez que se pretende analisar diferentes possibilidades para implementar os componentes da busca local proposta – solução inicial, preprocessamento, vizinhança, estratégia de busca e método de recomeço – determinar a melhor configuração possível é um dos objetivos deste trabalho. Como a performance do algoritmo depende da combinação destes componentes, o número de possíveis implementações torna-se elevado e adotou-se uma metodologia "gulosa" nos testes realizados: dada uma configuração inicial são testadas diferentes possibilidades para um dos componentes da busca, sendo que o de melhor resultado é fixado nos testes dos próximos componentes.

1.4 Conceitos Básicos de Programação Combinatória Multiobjetivo

1.4.1 Otimização em problemas combinatórios

Uma classe de problemas freqüentemente abordada na área de otimização é a de problemas combinatórios. Estes problemas consistem na seleção, permutação ou designação (*assignment*) de um conjunto discreto de itens, onde itens podem se referir a

máquinas, pessoas, horários, atividades e outros. Um problema de otimização combinatória pode ser definido como:

$$\textbf{minimizar } f(x)$$

$$\textbf{s.a. } x \in X$$

$$\text{onde } f : X \rightarrow \Re.$$

O conjunto X é o espaço das soluções, sendo geralmente um conjunto discreto, finito e com número excessivamente grande de elementos. Uma solução x é dita factível se $x \in X$. Uma solução factível x^* é ótima se $f(x^*) \leq f(x)$, para todo $x \in X$.

A resolução de um problema de otimização combinatória consiste em determinar uma solução ótima x^*, que minimiza a função objetivo f(x) no espaço de soluções X. Entretanto, em muitas das aplicações estes problemas pertencem a classe de problemas NP-Completo (ver Garey e Johnson, 1979), o que significa que não se conhecem algoritmos polinomiais que resolvam estes problemas. Uma alternativa bastante comum é a utilização de heurísticas que visam determinar boas soluções para os problemas sem a garantia de otimalidade.

Tanto algoritmos exatos, que determinam a solução ótima, quanto heurísticas têm como resposta para o problema uma única solução, a de melhor valor encontrado para a função objetivo f(x). Isto é possível porque a função f(x) determina qual solução é a preferida pelo decisor dentre todas as obtidas pelo algoritmo.

1.4.2 Programação Multiobjetivo

É comum existirem diversos critérios a serem utilizados na avaliação de uma solução para um problema combinatório, o que torna o problema multiobjetivo (também conhecido como multicritério). Observe-se que no caso multiobjetivo não se menciona otimização e sim programação, uma vez que não se pode em geral otimizar um vetor de funções objetivo, pois uma solução pode ser a melhor para um objetivo e não ser a melhor para outro. Formula-se um problema com *p* objetivos como:

$$\textbf{minimizar } F(x) = [f_1(x), f_2(x), \ldots f_p(x)]$$

$$\textbf{s.a. } x \in X$$

$$\text{onde } F : X \rightarrow \Re^p.$$

O conjunto X é conhecido como espaço das soluções, enquanto define-se o espaço dos objetivos $Y = \{y \in \Re^p: \exists\, x \in X$ tal que $y = F(x)\}$.

O significado de minimizar um vetor de funções é de que é sempre vantajoso reduzir o valor de um dos objetivos se os demais não aumentarem, gerando o conceito de dominância da definição 1.1.

Definição 1.1 – A solução *x* domina a solução *y* se

$$f_i(x) \leq f_i(y), \text{ para todo } i \in [1, p]$$

$$f_k(x) < f_k(y), \text{ para algum } k \in [1, p]\,.$$

Uma forma de visualizar o conceito de dominância da definição 1.1 é apresentado na figura 1.1, onde está representado no espaço dos objetivos a região dominada por uma solução *x*, a região que domina esta solução e a região que não é comparável a *x*, para um problema com dois objetivos.

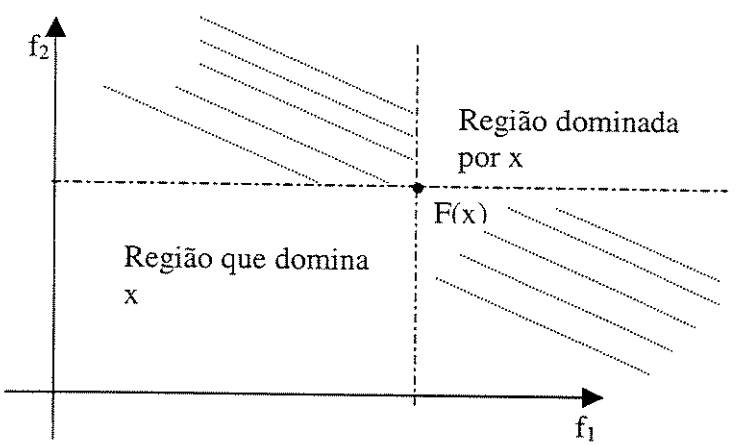

Figura 1.1 – Representação do conceito de dominância no espaço dos objetivos

Este conceito de dominância entre soluções faz com que exista um conjunto de soluções não-dominadas, também conhecido como conjunto de soluções eficientes ou Pareto ótimo, conforme a definição 1.2.

Definição 1.2 - O conjunto E é dito eficiente se

$$E = \{x \in X: \text{não existe } y \in X \text{ tal que y domine x}\}.$$

Toda solução pertencente ao conjunto eficiente é dita solução eficiente, não-dominada ou Pareto-ótima. Todas as soluções que não pertencem ao conjunto eficiente são dominadas por pelo menos uma solução eficiente. O conjunto de soluções não-dominadas é portanto composto de soluções que ou são equivalentes ou não são comparáveis entre si, isto é, se $x \in E$, $y \in E$ então:

$$F(x) = F(y) \text{ ou}$$

$$f_i(x) < f_i(y) \text{ e } f_k(x) > f_k(y), \text{ para algum } i, k \in [1..p] .$$

Os problemas multiobjetivo mais estudados são os problemas de programação linear e não-linear multiobjetivo, onde o espaço das soluções é contínuo. Vários métodos são propostos para estes problemas, sendo uma boa referência o livro de Goicoechea et al. (1982). Existem basicamente duas alternativas na resolução de problemas multiobjetivo:

- métodos com indicação *a priori* – as preferências do decisor são conhecidas antes da resolução do problema
- métodos com indicação *a posteriori* – apenas após a resolução do problema o decisor determina as preferências na escolha da solução a ser adotada.

Os métodos *a priori* são em geral mais simples de serem implementados e também mais eficientes. Entretanto, é necessário o conhecimento das preferências do decisor antes da utilização do método, o que nem sempre é viável. Alguns dos métodos *a priori* mais utilizados são:

- Escalarização dos objetivos – consiste em combinar os objetivos numa única função a ser minimizada, como por exemplo minimizar a soma ponderada dos objetivos.
- Programação por metas (*Goal Programming*) – neste método definem-se metas a serem atingidas para cada objetivo, minimizando uma função que combine os desvios com relação a cada meta.
- Métodos lexicográficos – cria-se uma hierarquia para os objetivos com relação à sua importância. O problema é resolvido minimizando-se o objetivo de maior prioridade $f_1(x)$, depois minimizando o de segunda maior importância $f_2(x)$ sujeito à restrição que o primeiro objetivo não piore de

valor e assim sucessivamente, até o objetivo $f_p(x)$ sujeito a que os objetivos anteriores mantenham o valor ótimo obtido.

É natural que um método *a priori* busque como solução final uma solução eficiente (não-dominada), minimizando o vetor de objetivos a partir do conhecimento das preferências do decisor. Já os métodos *a posteriori* procuram gerar o conjunto de soluções eficientes ou a parte relevante deste conjunto para a escolha posterior do decisor de qual solução adotar para o problema. Esta tarefa é mais complexa do que a dos métodos *a priori*, uma vez que a geração do conjunto eficiente obtém todas as soluções que podem ser escolhidas pelo decisor baseado no conceito de dominância (definição 1.1).

Algumas das vantagens da geração do conjunto eficiente são:

- método geral para a resolução do problema – uma vez desenvolvida uma metodologia para gerar o conjunto de soluções não-dominadas pode-se aplicá-la em diversas situações, independente de mudanças nas preferências do decisor;
- informação sobre soluções alternativas – pode-se avaliar o quanto se ganha ou se perde em cada objetivo quando se escolhe uma solução em detrimento das demais;
- análise do problema – determina-se por exemplo qual o valor mínimo e máximo de cada objetivo numa solução possível para o problema;
- mudança de políticas decisórias – uma vez resolvido o problema pode-se mudar a política decisória, escolhendo uma solução diferente sem a necessidade de resolvê-lo novamente.

Mesmo problemas de otimização combinatórios solúveis em tempo polinomial, como o problema de designação ou de caminho mínimo, podem se tornar NP-Completos para o caso multiobjetivo, conforme demonstrado por Serafini (1987). A geração do conjunto de soluções eficientes é um problema de difícil resolução em diversos casos como o abordado neste trabalho, exigindo a utilização de heurísticas conforme descrito na seção 1.3.

1.5 Métodos de avaliação das heurísticas

A avaliação de aproximações do conjunto Pareto ótimo não é uma tarefa simples, tendo sido objeto específico de estudo em Daniels (1992) e em Hansen e Jaszkiewicz (1998). Dentre as formas de avaliação propostas optou-se por utilizar a medida de desvio proposta por Daniels e a medida de distância descrita em Czyac e Janszkiewicz (1998), por serem razoavelmente intuitivas possibilitando uma comparação entre estes métodos de análise.

1.5.1 Medida de desvio

O método proposto por Daniels define um valor de utilidade de um conjunto solução H para uma dada ponderação linear α dos objetivos, u(H, α), e calcula de forma exata o desvio médio desta utilidade com relação a um conjunto de referência. Adaptamos este método calculando o desvio médio de forma aproximada discretizando o valor da ponderação em mil intervalos, o que simplifica a implementação com um erro extremamente pequeno.

Procedimento para cálculo do desvio

1. Seja R o conjunto de referência (soluções eficientes para instâncias de pequeno porte, e para instâncias maiores o conjunto das soluções não dominadas dentre todas as obtidas pelas heurísticas), H o conjunto solução de uma das heurísticas que será avaliado.

2. Seja M_{max} , M_{min} , T_{max} e T_{min} os valores máximo e mínimo do *Makespan* e do atraso total respectivamente dentre todas as soluções a serem avaliadas e as soluções em R.

3. Seja a utilidade para uma ponderação α dos objetivos

$u(H, \alpha) = \max_{x \in H}\{ V_x(\alpha) \}$, onde

$$V_x(\alpha) = \alpha \cdot \left(\frac{M_{max} - M_x}{M_{max} - M_{min}} \right) + (1-\alpha) \cdot \left(\frac{T_{max} - T_x}{T_{max} - T_{min}} \right) \quad (1.1)$$

4. Seja $$desvio(H, \alpha) = \frac{u(R,\alpha) - u(H,\alpha)}{u(R,\alpha)} \quad (1.2)$$

5. Calcula-se $$desv\_med(H) = \frac{\sum_{i=0}^{r} desvio\left(H, {}^{i}/_{r}\right)}{r} \quad (1.3)$$

Utilizou-se r = 1000. Quanto melhor a heurística H, menor é o valor de desv_med(H). Este valor pode ser interpretado como o desvio médio da utilidade u(H,α) com relação ao conjunto de referência R, para α variando de 0 a 1. Note-se que os dois objetivos são normalizados de forma que $V_x(\alpha)$ esteja no intervalo [0,1], assim como os valores de utilidade.

A função utilidade u(H, α) corresponde ao melhor valor ponderado dos objetivos de uma solução em H. Esta ponderação é linear, o que implica no descarte de algumas soluções na avaliação que, apesar de serem não dominadas, não podem ser melhores que outra solução para uma ponderação linear não negativa dos objetivos. Na figura 1.2 temos um exemplo em que a solução B não seria considerada na avaliação do conjunto solução,

pois apenas A e C podem ser ótimas para uma ponderação linear não negativa dos objetivos.

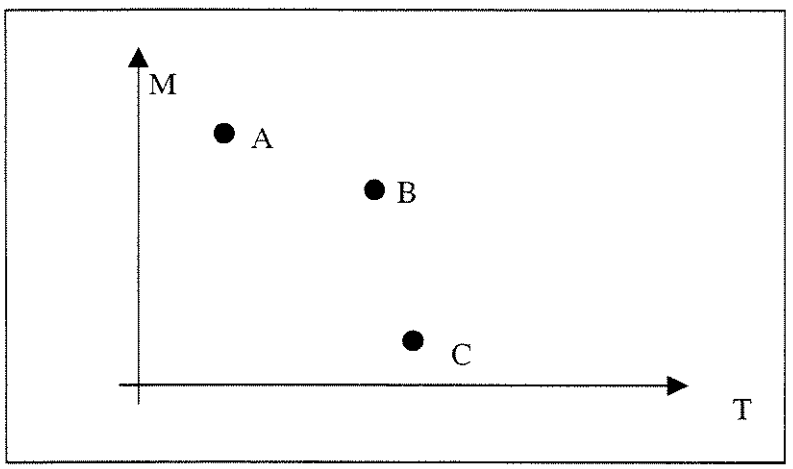

Figura 1.2 – A solução B é eficiente, mas não é considerada numa ponderação linear não negativa dos objetivos

Apesar de não considerar todas as soluções, a medida de desvio apresentada é bastante razoável, sendo a ponderação linear dos objetivos uma abordagem comum na escalarização de problemas multiobjetivos e que descartam as mesmas soluções.

1.5.2 Medida de distância

Apesar de medidas de distância estarem sujeitas a críticas e imperfeições (ver Hansen e Jaszkiewicz, 1998), este tipo de avaliação é usual e foi utilizado por Czyac e Janszkiewicz (1998) e Arroyo e Armentano(2000). Para efeitos de comparação, implementamos uma avaliação da distância média das soluções geradas H ao conjunto de referência R. Esta implementação foi baseada no trabalho de Czyac e Janszkiewicz, sendo diferente apenas a forma de calcular o intervalo de variação de cada objetivo: para Czyac e Janszkiewicz calcula-se M_{max} , M_{min} , T_{max} e T_{min} utilizando-se as soluções no conjunto de referência, enquanto que para manter coerência com a medida de desvio já descrita calculamos estes valores sobre todas as soluções a serem avaliadas juntamente com as soluções em R.

A medida de distância utilizada foi a das equações 1.4 e 1.5, onde |R| é o número de soluções em R (cardinalidade).

$$Dist(H) = \frac{1}{|R|}\sum_{r\in R}\min_{z\in H}\{d(r,z)\} \qquad (1.4)$$

$$d(r,z) = \max\left\{\frac{M_z - M_r}{M_{\max} - M_{\min}}, \frac{T_z - T_r}{T_{\max} - T_{\min}}\right\} \qquad (1.5).$$

2 *Heurísticas Construtivas*

Heurísticas construtivas são procedimentos que constroem uma solução a partir de uma regra, sem tentar melhorar uma solução já estabelecida. Ou seja, num problema de seqüenciamento uma seqüência é construída sucessivamente sem que as partes já estabelecidas sejam modificadas. A utilização destas heurísticas se dá em problemas em que o tempo computacional é muito curto ou como ponto de partida para algoritmos de melhoria.

Trataremos aqui de quatro tipos de heurísticas:

- Heurística baseada na regra ATCS;
- Heurísticas de inserção;
- Cheapest insertion;
- Heurísticas baseadas na regra do vizinho mais próximo (Nearest Neighbor).

Apesar de ser uma heurística de inserção, cheapest insertion será tratada como um caso a parte por ter implementação diferenciada.

2.1 Heurística baseada na regra ATCS

A regra ATCS – Apparent Tardiness Cost with Setups - foi proposta por Lee et al. em 1997, e consiste de três fases: na primeira fase são calculados parâmetros que caracterizam a instância em questão; na segunda, uma seqüência é construída baseada numa regra de despacho cujos parâmetros dependem da primeira fase; na última fase propõe-se um algoritmo de melhoria para a solução da segunda fase. Utilizaremos aqui apenas os conceitos das duas primeiras fases, sendo que algoritmos de melhoria serão tratados posteriormente.

O índice de prioridade da regra ATCS é:

$$I_j(t,l)=\frac{1}{p_j}\exp\left[-\frac{\max(d_j-p_j-t,0)}{k_1\overline{p}}\right]\cdot\exp\left[-\frac{s_{lj}}{k_2\overline{s}}\right] \qquad (2.1)$$

que é calculado para cada tarefa j ainda não seqüenciada sendo t o instante atual, l a última tarefa seqüenciada, $\overline{s}$ a média dos tempos de preparação, $\overline{p}$ a média dos tempos de processamento e k_1 e k_2 são parâmetros de escala. A próxima tarefa a ser processada é a de maior índice. A primeira exponencial na equação 2.1 calcula a folga com relação à data de entrega se o tempo de preparação fosse nulo, dando maior prioridade a tarefas com menor folga. A segunda exponencial prioriza tarefas com tempo de preparação s_{lj} pequenos.

Lee et al. propõem que os parâmetros k_1 e k_2 sejam determinados a partir de estatísticas realizadas nos dados da instância, gerando uma única seqüência. Nossa proposta para o caso multiobjetivo é gerar várias seqüências para diferentes valores de k_1 e k_2. Nos testes realizados por Lee et al. observa-se que valores razoáveis para k_1 estão no intervalo [2,5 , 5,5] e para k_2 em [0,1 , 0,9], pois fora destes intervalos os resultados foram ruins. Subdividimos estes intervalos em r-1 partes e executamos a regra da equação 2.1 para as combinações de valores possíveis, gerando r^2 seqüências das quais se tomam as não dominadas como solução.

Por exemplo, para $r = 3$ executaríamos a regra para os seguintes pares (k_1; k_2):

(2,5 ; 0,1), (2,5 ; 0,5), (2,5 ; 0,9), (4,0 ; 0,1), (4,0 ; 0,5), (4,0 ; 0,9), (5,5 ; 0,1), (5,5 ; 0,5), (5,5 ; 0,9).

Além disto, executamos a regra uma vez com os parâmetros recomendados em Lee et al. (eq 2.2), totalizando $r^2 + 1$ soluções.

Os valores de k_1 e k_2 recomendados em Lee et al.são:

$$k_1 = \begin{cases} 4{,}5 + R & , R \le 0{,}5 \\ 6{,}0 - 2R & , R > 0{,}5 \end{cases} \quad \text{e} \quad k_2 = \frac{\tau}{2\sqrt{\eta}} \qquad (2.2)$$

onde

$$\eta = \frac{\bar{s}}{\bar{p}} \; ; \qquad \tau = 1 - \frac{\bar{d}}{n(\bar{p} + \beta \cdot \bar{s})} \; ; \quad R = \frac{d_{max} - d_{min}}{n(\bar{p} + \beta \cdot \bar{s})} \; (2.3).$$

O parâmetro η é chamado Fator de Severidade dos Tempos de Preparação, indicando a proporção do tempo médio de setup com relação ao tempo médio de processamento. Já τ é denominado Fator de Atraso, sendo que quanto maior τ espera-se um maior número de tarefas entregues com atraso. O Fator de Espalhamento R indica a proporção do intervalo de variação das datas de entrega com relação à estimativa do Makespan.

O valor de $n(\bar{p} + \beta\bar{s})$ é utilizado como uma estimativa do valor do Makespan da solução final. O valor de β influencia R e τ, mas não é definido em Lee et al. sendo apresentado apenas um gráfico do valor de β em função de cv - coeficiente de variação dos tempos de setup. A partir deste gráfico determinou-se β como:

$$\beta = \begin{cases} 0{,}925974 - 2{,}5974 \cdot cv & , cv \le 0{,}1423 \\ 0{,}743125 - 1{,}3125 \cdot cv & , cv > 0{,}1423 \end{cases} ; \qquad (2.4).$$

Procedimento ATCS(r)

1. Calcular $\bar{p}$, $\bar{s}$ e (k_1, k_2) recomendados em Lee et al.;
2. Executar a regra ATCS $r^2 + 1$ vezes;
3. Tomar as soluções não dominadas como aproximação para o conjunto de soluções eficientes.

A regra ATCS tem complexidade $O(n^2)$, pois o cálculo de $\bar{p}$ pode ser feito em O(n), $\bar{s}$ em tempo $O(n^2)$, e é necessário $O(n(n+1)/2 - 1) \equiv O(n^2)$ para gerar a seqüência, já que o índice de prioridade deve ser recalculado para as tarefas ainda não seqüenciadas toda

vez que uma tarefa é incluída na seqüência. O procedimento ATCS(r) tem portanto complexidade $O(r^2n^2)$, pois a regra ATCS é executada r^2 vezes para os diferentes valores de k_1 e k_2.

Nos testes que realizamos, utilizou-se $r = 12$, de forma que o tempo computacional fosse semelhante às heurísticas de inserção (seção 2.2) para $n = 60$, que é o tamanho médio das instâncias testadas. Desta forma, as heurísticas podem ser comparadas de forma mais justa por consumirem aproximadamente o mesmo esforço computacional. A comparação é feita com as heurísticas de inserção porque o tempo destas heurísticas independe de parâmetros utilizados, ou seja, não são ajustáveis.

2.2 Heurísticas de inserção

Heurísticas construtivas baseadas em procedimentos de inserção, onde a cada iteração uma tarefa é inserida na seqüência, são largamente utilizadas no PCV para o qual boas análises podem ser encontradas em Reinelt(1994) e Lawler et al.(1985). Mais recentemente, tem têm sido aplicados a problemas de *scheduling* , a partir do trabalho de Nawaz et al.(1983).

Arroyo e Armentano (2000) adaptam a técnica de inserção de Nawaz para o problema de flowshop multiobjetivo de forma a aproximar o conjunto Pareto – Ótimo, da seguinte forma:

Procedimento de inserção multiobjetivo

1. Ordene as tarefas por alguma regra pré-estabelecida (esta regra diferencia as implementações realizadas).
2. Inicialize o conjunto S_1 (soluções parciais não-dominadas) com uma solução com apenas a primeira tarefa.
3. Faça k = 2.
4. Insira a k-ésima tarefa em todas as posições possíveis em cada solução pertencente a S_{k-1} , formando soluções com k tarefas. Faça S_k ser o conjunto destas novas soluções que não são dominadas por outra solução gerada.
5. Faça k = k + 1
6. Se $k \leq n$ volte ao passo 4. Caso contrário, FIM.

Este procedimento mantém um conjunto de soluções não dominadas entre si que evolui a cada iteração com a inserção de uma nova tarefa. Se considerarmos que $|S_k| < M$, o algoritmo pode ser implementado em $O(M.n^3)$ pois na iteração k são realizadas no máximo M.k inserções, e cada inserção tem complexidade O(k). O total de inserções é portanto:

$$O\left(\sum_{k=1}^{n} M \cdot k^2\right) = O\left(M \frac{n(2n^2+3n+1)}{6}\right) = O\left(n^3\right) \qquad (2.5).$$

Entretanto, no pior caso em que toda solução parcial é não dominada, tem-se uma complexidade exponencial, pois $|S_k|=k!$. É razoável supor que M seja uma constante não muito grande, o que levaria a um comportamento polinomial.

A seguir, apresentamos as regras utilizadas na ordenação inicial, sendo algumas baseadas em implementações para *scheduling* e outras para o PCV.

2.2.1 Inserção com ATCS - I_ATCS

Utilizaram-se os parâmetros das Eq. 2.2, 2.3 e 2.4 para gerar uma única seqüência baseada na regra ATCS (equação 2.1), sendo esta a ordenação inicial das tarefas.

2.2.2 Inserção com EDD – I_EDD

Regra EDD (Earliest Due Date), ou seja, ordenou-se as tarefas pela data de entrega, sendo a tarefa com menor data de entrega a primeira e assim por diante.

2.2.3 Inserção aleatória – I_ALEAT

Toma-se uma ordem aleatória das tarefas.

2.2.4 Farthest Insertion (1) – I_FAR(1)

Suponha, sem perda de generalidade, que k tarefas já ordenadas sejam {0, 1,2,...,k} onde 0 indica o estado inicial da máquina. Define-se a distância da tarefa *j* às tarefas já seqüenciadas como $dist_j = min_{i \leq k} \{s_{ij}\}$. A próxima tarefa na ordem é a de maior distância.

2.2.5 Farthest Insertion (2) – I_FAR(2)

Muda-se a definição de distância do item anterior para

$dist_j = min_{i \leq k} \{min[s_{ij}, s_{ji}]\}$.

2.2.6 Nearest Insertion (1) – I_NEA(1)

A definição de distância é a mesma do item 2.2.4, sendo a próxima tarefa a de menor distância.

2.2.7 Nearest Insertion (2) – I_NEA(2)

A definição de distância é a mesma do item 2.2.5, sendo a próxima tarefa a de menor distância.

2.2.8 Inserção com TLB (Tardiness Lower Bound) – I_TLB

Seja $t_j = min_{\ k} s_{kj}$, o menor tempo de preparação possível para a tarefa j. Ordenam-se as tarefas em ordem crescente de $d_j - p_j - t_j$, onde d_j é a data de entrega e p_j o tempo de processamento da tarefa *j*. Esta regra de ordenação, TLB, foi proposta num procedimento de inserção para um problema equivalente minimizando a soma dos atrasos (Armentano e Ronconi, 1999), com bons resultados.

2.2.9 Inserção com MDD (Modified Due Date) – I_MDD

Seja k a última tarefa seqüenciada (inicialmente k = 0). A próxima tarefa j a ser seqüenciada será a de menor valor de $\max\{d_j;\ C_k + s_{kj} + p_j\}$, onde C_k é o instante de término do processamento da tarefa k se as tarefas forem processadas na ordem atual.

2.2.10 Inserção com vizinho mais próximo (Nearest Neighbor) – I_NN

A primeira tarefa é escolhida aleatoriamente. Seja k a última tarefa seqüenciada. A próxima tarefa j a ser seqüenciada será a de mínimo s_{kj}.

2.3 Cheapest Insertion

Cheapest Insertion (inserção de custo mínimo) é uma estratégia para problemas mono-objetivo, com bons resultados para o PCV (Reinelt, 1994). Dada uma seqüência parcial, testa-se todas as possíveis inserções das tarefas não seqüenciadas, realizando-se a inserção de menor custo.

Para o caso multiobjetivo, propõe-se uma função ponderada de custo

$$f(\alpha) = \alpha M + (1-\alpha)T \quad (2.5)\,,$$

e executa-se r vezes o algoritmo para diferentes valores de α. Observe-se que inicialmente a seqüência é vazia, calculando-se $f(\alpha)$ para o processamento de cada tarefa individualmente. Utilizou-se o seguinte procedimento:

Procedimento cheapest insertion multiobjetivo

1. Seja iter = 1;
2. Faça $\alpha = (\text{iter}-1)/(r-1)$;
3. Enquanto houver uma tarefa não seqüenciada, realize a inserção de menor custo $f(\alpha)$, construindo uma solução seqüencialmente.
4. Faça iter = iter +1; se iter $\leq$ r volte ao passo 2;
5. Tome as soluções não dominadas resultantes do passo 2 como aproximação para o conjunto de soluções eficientes.

Utilizou-se $r = 3$ e $r = 5$ nos teste realizados. O cálculo de $f(\alpha)$ pode ser feito em $O(n)$ para cada posição de inserção de uma tarefa j. Como o número de posições possíveis e o número de tarefas não seqüenciadas são $O(n)$, avaliar a solução de menor custo no passo 3 pode ser feito em $O(n.n.n) \equiv O(n^3)$. Como a inserção de menor custo é realizada $O(n)$ vezes até que todas as tarefas sejam seqüenciadas, para cada valor de α temos uma complexidade $O(n^4)$. Como este procedimento é realizado r vezes, temos uma complexidade final $O(r.n^4)$.

2.4 Heurísticas baseadas no vizinho mais próximo (Nearest Neighbor)

A regra do vizinho mais próximo é muito utilizada no PCV por ser de fácil implementação e por dar bons resultados quando combinada com heurísticas de melhoria baseadas em troca de arestas. Ela consiste em um algoritmo guloso em que dada uma cidade (tarefa) inicial, a próxima cidade é a de menor distância (tempo de preparação) da última visitada (processada).

Propõem-se duas implementações para o problema proposto:

2.4.1 Vizinho Mais Próximo com Rotação - VMPR

Nesta proposta primeiro gera-se uma seqüência baseada na regra do vizinho mais próximo, onde a distância é o tempo de preparação (*Setup*). Esta solução não considera as datas de entrega, buscando reduzir os tempos de preparação. Uma forma de obter soluções melhores para o atraso total a partir desta solução é modificar a tarefa pela qual se inicia o processamento mantendo a seqüência obtida, procedimento denominado rotação. Geram-se novas seqüências através de rotação colocando-se a primeira tarefa em último lugar sucessivamente. Exemplo: suponha $n = 4$, e que a seqüência gerada pelo vizinho mais próximo seja 1234. Geram-se as seguintes seqüências por rotação: 2341, 3412, 4123.

O procedimento descrito foi realizado com todas as tarefas como ponto de partida para a regra do vizinho mais próximo, tomando-se dentre todas as soluções geradas as não dominadas.

Gerar uma seqüência pela regra do vizinho mais próximo pode ser realizado em $O(n^2)$, pois para cada tarefa seqüenciada é avaliado o tempo de preparação das tarefas

restantes, tomando-se a de menor valor como a próxima a ser processada. Gerada a seqüência, é somado o tempo de realizar n - 1 rotações, onde cada rotação implica reavaliar o atraso e o Makespan, o que é feito em O(n). Então as rotações podem ser feitas também em $O(n^2)$. Como o tempo de gerar a seqüência e realizar as rotações é $O(n^2+n^2) \equiv O(n^2)$ e são utilizados n pontos de partida, a complexidade desta implementação é de $O(n.n^2) \equiv O(n^3)$.

2.4.2 Time Oriented Nearest Neighbor - TONN

Solomon (1987) propôs uma adaptação da regra do vizinho mais próximo para o problema de roteamento de veículos com janelas de tempo. A proposta foi mudar o critério de distância para levar em conta as janelas de tempo. Aplicando o mesmo conceito para o problema deste trabalho, propõe-se que a medida de distância da última tarefa k na seqüência às demais seja:

$$D_{kj} = \alpha S_{kj} + (1-\alpha).(d_j - C_k - S_{kj} - p_j) \qquad (2.6)\,,$$

que é a ponderação α entre o tempo de preparação e a folga da tarefa com relação à data de entrega.

A primeira tarefa é a de mínimo D_{0j}. Este procedimento foi realizado para r diferentes valores de α, de forma análoga a *cheapest insertion*. Porém este algoritmo é muito rápido o que possibilitou a utilização de $r = 400$, de forma que o tempo computacional fosse semelhante às heurísticas de inserção para $n = 60$.

Para cada valor de α, gerar uma seqüência pode ser realizado em $O(n^2)$, pois para cada tarefa seqüenciada avalia-se a distância das tarefas restantes. Isto é realizado r vezes, sendo a complexidade $O(r.n^2)$.

3 *Testes para heurísticas construtivas*

Implementaram-se os algoritmos utilizando a linguagem de programação C++, em uma estação SUN Ultra 60 com dois processadores de 360 MHz, e como o código é seqüencial usou-se apenas um dos processadores. O compilador utilizado foi o GNU C Compiler, com a opção de otimização –O2.

A representação de uma solução foi feita através de um vetor com os índices das tarefas na ordem de processamento. Dois vetores auxiliares, um com o instante de conclusão (*Completion Time*) de cada tarefa e outro com a soma dos atrasos das tarefas anteriores a ela, foram utilizados de forma a aumentar a eficiência. Assim, a inserção de uma tarefa em uma posição requer apenas a atualização das tarefas seguintes, sem ter de recalcular a solução desde o início. Um conjunto de soluções não dominadas entre si foi representado através de uma lista ligada de soluções (ver figura 3.1), onde as soluções mais antigas aparecem primeiro na lista. O ponteiro cabeça indica onde começa a lista, sendo que cada solução aponta para a próxima, e a última indica o final da lista.

Figura 3.1 – Exemplo de lista ligada

3.1 Instâncias Teste

Foram geradas instâncias aleatórias dependentes dos valores dos parâmetros η, τ e R, apresentados no capítulo 2, seção 2.1. O esquema de geração é semelhante ao de França et al. (2001):

- Tempo de processamento da tarefa i - p_i : gerado a partir de uma distribuição uniforme no intervalo [1, 100];
- Tempo de preparação para executar uma tarefa j após uma tarefa i - s_{ij} : gerado uniformemente no intervalo [1, 100.η];
- A partir do fator de atraso τ, calcula-se a média das datas de entrega $\mu = (1-\tau)\cdot\sum_{i=1}^{n} p_i$.
- Calcula-se $\delta = R\cdot\sum_{i=1}^{n} p_i$, onde R é o fator de espalhamento
- Data de entrega - d_j : gerada uniformemente entre μ-δ/2 e μ+δ/2. Valores negativos são substituídos por 0.

Todos os três fatores η, τ e R assumiram valores 0.2, 0.6 e 1.0 , totalizando 27 diferentes configurações possíveis. Para cada configuração gerou-se cinco instâncias. Foram realizados dois testes diferentes, um com instâncias até 14 tarefas (n = 10, 12 ou 14) e outro com até 100 tarefas (n= 20, 40, 60, 80, 100), totalizando 405 instâncias de pequeno porte e 675 instâncias maiores. As instâncias de pequeno porte foram resolvidas por enumeração completa.

Uma análise que pode ser realizada é a da correlação entre cada fator e o número de soluções eficientes da instância, nos casos em que se aplica enumeração completa. Na Tabela 3.1 temos esta correlação, demonstrando que o número de soluções não-dominadas cresce com n e R, e decresce com η. Um maior número de soluções indica que os objetivos considerados são mais conflitantes. Outra forma de visualizar esta relação está na média do número de soluções eficientes nas instâncias com um dado parâmetro, apresentadas na Tabela 3.2.

Tabela 3.1 – Correlação estatística entre cada fator e o número de soluções eficientes

	correlação
n	0.274
τ	0.150
η	-0.370
R	0.285

Tabela 3.2 – Média do número de soluções eficientes por parâmetro

n	nsol
10	5.8
12	6.8
14	8.4

τ	nsol
0.2	6.1
0.6	7.4
1.0	7.5

η	nsol
0.2	9.1
0.6	6.2
1.0	5.7

R	nsol
0.2	5.7
0.6	6.9
1.0	8.4

3.2 Instâncias de até 14 tarefas

Separa-se a análise dos resultados entre as 405 instâncias de pequeno porte (n = 10, 12 ou 14) e as 675 de maior porte (n = 20, 40, 60, 80, 100), já que as soluções destas instâncias são comparadas às soluções eficientes obtidas por enumeração completa.

Utilizaram-se as seguintes legendas para identificar os algoritmos:

- ATCS(0), regra ATCS executada apenas uma vez com os parâmetros sugeridos por Lee et al. (1997), gerando portanto apenas um ponto no espaço das soluções, o que permite avaliar a melhoria causada pela execução da regra com diferentes parâmetros;
- ATCS(12), adaptação da regra ATCS para o caso multiobjetivo, com r =12;
- I_X, heurística de inserção com ordenação inicial da regra X, por exemplo I_ATCS;
- CI(3) ou CI(5), Cheapest Insertion para r = 3 ou r =5;
- VMPR, Vizinho Mais Próximo com Rotação;
- TONN, Time Oriented Nearest Neighbor.

Na Tabela 3.3 estão os resultados da média de desv_med (1.3), média da distância (1.4) ao conjunto eficiente, desvio médio para o Makespan e para o Atraso e o número total de soluções eficientes encontradas nos testes. O desvio médio para o Makespan é a média do desvio(H,α), equação (1.2), para α = 1 e o desvio para o atraso calculado com α = 0. A heurística VMPR foi a melhor na média do desvio, com uma vantagem pequena sobre ATCS(12) e I_ATCS. Para a distância. os resultados são similares para as três melhores heurísticas. VMPR foi a melhor heurística para o Makespan enquanto ATCS(12) foi a melhor para o atraso. Entretanto, I_ATCS mostrou-se mais equilibrada, obtendo bons resultados tanto para o Makespan quanto para o atraso. Note-se que ATCS(0) gerou uma solução eficiente em 23 das 405 instâncias.

Tabela 3.3 – Resultados médios para o desvio, distância, desvio para o makespan e para o atraso, e número total de soluções eficientes encontradas para 405 instâncias de 10 a 14 tarefas

Desvio		Distância		Makespan		Atraso		Eficientes	
VMPR	18.6%	VMPR	0.215	VMPR	14.8%	ATCS(12)	11.5%	I_ATCS	131
I_ATCS	19.2%	I_ATCS	0.221	I_NEA(1)	25.7%	I_MDD	13.2%	I_MDD	111
ATCS(12)	19.3%	ATCS(12)	0.238	I_ATCS	26.5%	I_ATCS	13.6%	VMPR	95
I_MDD	23.4%	I_NEA(1)	0.265	I_NN	27.1%	CI(5)	16.9%	ATCS(12)	86
CI(5)	23.7%	I_MDD	0.275	TONN	27.2%	CI(3)	18.6%	CI(5)	69
I_NEA(1)	24.2%	I_NEA(2)	0.279	I_NEA(2)	27.9%	I_EDD	19.7%	I_EDD	66
I_NEA(2)	25.7%	I_NN	0.288	CI(5)	30.1%	VMPR	22.5%	I_NEA(1)	61
CI(3)	25.9%	CI(5)	0.298	ATCS(12)	30.9%	I_NEA(1)	23.8%	I_NEA(2)	61
I_NN	26.2%	I_EDD	0.300	CI(3)	31.1%	I_TLB	24.8%	I_NN	56
I_EDD	26.9%	I_TLB	0.314	I_MDD	32.4%	I_NEA(2)	25.1%	CI(3)	52
I_TLB	28.6%	CI(3)	0.330	I_TLB	32.7%	I_NN	26.2%	I_ALEAT	43
I_FAR(1)	31.0%	I_FAR(1)	0.332	I_ALEAT	33.1%	ATCS(0)	27.0%	I_TLB	43
I_ALEAT	31.4%	I_ALEAT	0.338	I_FAR(1)	33.3%	I_FAR(1)	29.1%	I_FAR(1)	35
TONN	31.7%	I_FAR(2)	0.362	I_EDD	34.4%	I_ALEAT	30.0%	TONN	29
I_FAR(2)	34.0%	TONN	0.373	I_FAR(2)	36.2%	I_FAR(2)	32.1%	ATCS(0)	23
ATCS(0)	42.3%	ATCS(0)	0.531	ATCS(0)	65.2%	TONN	38.5%	I_FAR(2)	22

O número total de soluções eficientes para as 405 instâncias é de 2833, sendo que a melhor heurística neste quesito, I_ATCS, encontrou apenas 131 destas soluções. Isto mostra a dificuldade das heurísticas construtivas em encontrar soluções eficientes, mesmo para instâncias pequenas. Hansen e Jaszkiewicz (1998) argumentam que este tipo de medida cardinal só é representativo quando o número de soluções não dominadas encontradas for alto e se as soluções eficientes estiverem distribuídas de maneira uniforme no espaço das soluções. Portanto, consideramos esta medida como a menos representativa na comparação da eficiência das heurísticas.

Na Figura 3.2 são exibidos os gráficos da média do desvio em função de α para as heurísticas que foram bem em um dos quesitos da Tabela 3.3 (os 4 melhores algoritmos para o desvio, distância, makespan, atraso ou eficientes). Nele pode-se observar a qualidade das soluções dependendo da importância de cada objetivo para o decisor. Os resultados para $\alpha = 0$ correspondem ao desvio médio com relação ao menor atraso total, e para $\alpha = 1$ com relação ao menor *Makespan*.

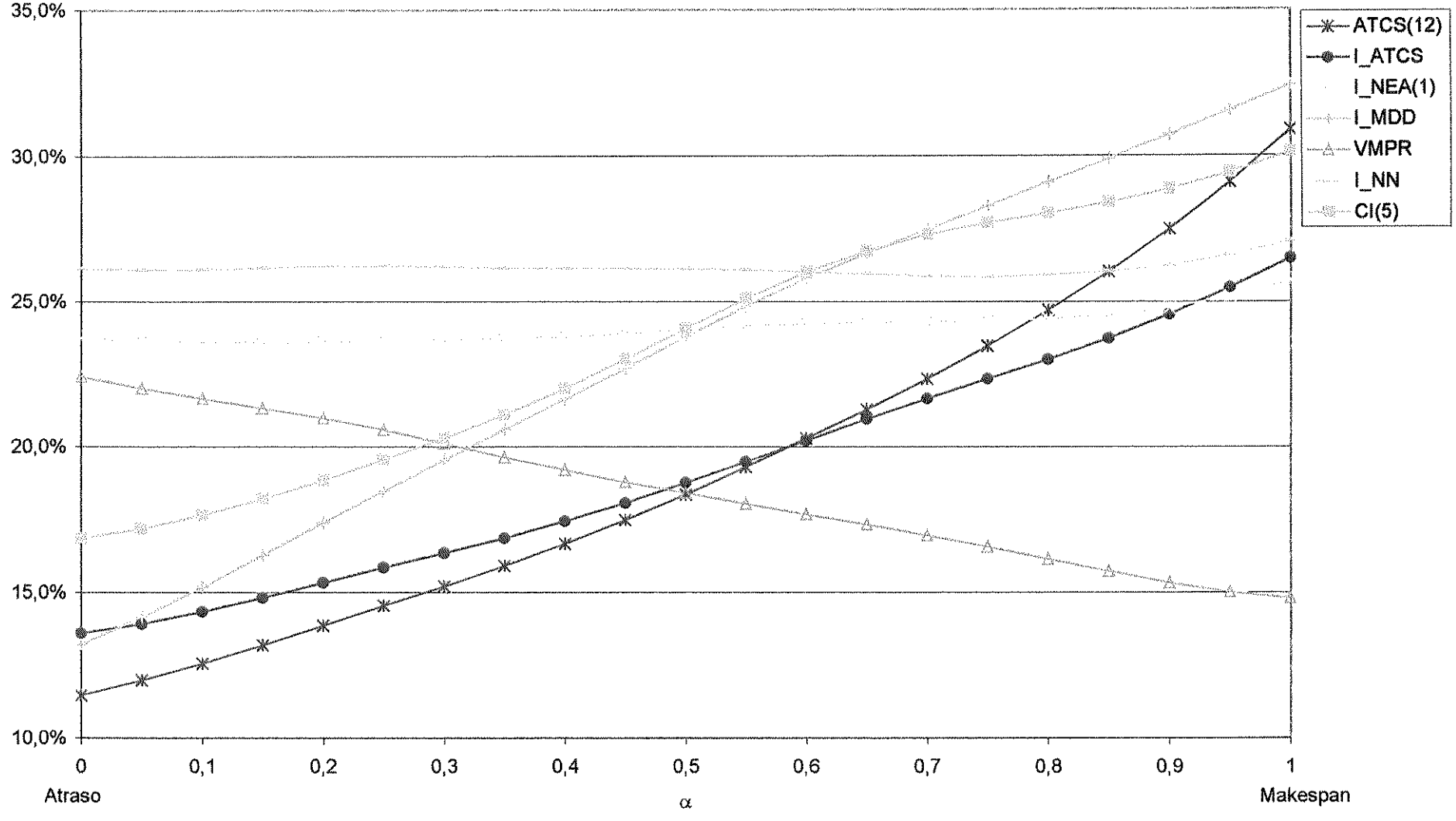

Figura 3.2 - Desvio %

3.3 Instâncias de 20 a 100 tarefas

As heurísticas construtivas foram testadas com o número de tarefas n = 20, 40, 60, 80 e 100, num total de 675 instâncias. Os resultados apresentados são calculados com relação ao conjunto de referência de cada instância formado pelas soluções não dominadas dentre todas as soluções encontradas pelas heurísticas.

Tabela 3.4 - Resultados médios para o desvio, distância, desvio para o makespan e para o atraso, e número de soluções não dominadas encontradas para as 675 instâncias de 20 a 100 tarefas

Desvio		Distância		Makespan		Atraso		Não Dominadas	
ATCS(12)	7.5%	ATCS(12)	0.123	VMPR	1.4%	ATCS(12)	1.4%	VMPR	564
I_ATCS	14.6%	I_ATCS	0.167	TONN	15.3%	I_ATCS	5.0%	I_ATCS	355
VMPR	16.9%	VMPR	0.198	I_NN	18.3%	I_MDD	7.1%	ATCS(12)	318
I_MDD	20.5%	I_NN	0.200	ATCS(12)	27.0%	ATCS(0)	11.2%	I_NN	231
I_NN	21.9%	I_MDD	0.231	I_ATCS	30.7%	I_EDD	15.9%	I_MDD	114
TONN	23.7%	I_EDD	0.245	I_NEA(1)	30.8%	CI(5)	17.7%	I_TLB	96
I_EDD	24.6%	I_TLB	0.268	I_NEA(2)	32.9%	CI(3)	19.2%	TONN	81
I_TLB	27.0%	I_NEA(1)	0.271	CI(5)	35.1%	I_TLB	19.8%	I_EDD	36
CI(5)	28.0%	TONN	0.277	CI(3)	35.2%	I_NN	29.2%	I_NEA(1)	28
I_NEA(1)	28.8%	I_NEA(2)	0.293	I_EDD	36.0%	I_NEA(1)	30.6%	I_ALEAT	14
CI(3)	29.8%	I_ALEAT	0.310	I_ALEAT	36.3%	I_NEA(2)	32.4%	CI(5)	10
I_NEA(2)	31.4%	CI(5)	0.325	I_TLB	36.8%	I_ALEAT	32.8%	I_FAR(1)	6
I_ALEAT	33.1%	I_FAR(1)	0.329	I_MDD	37.4%	I_FAR(1)	34.1%	I_NEA(2)	5
I_FAR(1)	34.9%	I_FAR(2)	0.338	I_FAR(1)	37.9%	I_FAR(2)	34.3%	ATCS(0)	4
I_FAR(2)	36.5%	CI(3)	0.348	I_FAR(2)	39.9%	TONN	40.8%	CI(3)	4
ATCS(0)	39.7%	ATCS(0)	0.578	ATCS(0)	79.8%	VMPR	41.3%	I_FAR(2)	2

Na Tabela 3.4 estão os resultados da média de desv_med (1.3), a distância média ao conjunto de referência (1.4), o desvio médio para o Makespan e para o Atraso e o número total de soluções não dominadas encontradas.

A heurística ATCS(12) foi a melhor na média do desvio, sendo a diferença significativa para I_ATCS e VMPR. As três melhores heurísticas para o desvio também o são para a distância. Observe que a regra ATCS tem bom desempenho para o atraso, mesmo se executada apenas uma vez como em ATCS(0). Já VMPR é a melhor heurística para o Makespan, contudo seu desempenho para o atraso foi pior, mostrando um claro desequilíbrio de desempenho.

Na Figura 3.3 são exibidos os gráficos da média do desvio em função de α para as melhores heurísticas. Na Tabela 3.5 estão os tempos computacionais para cada algoritmo. O crescimento do tempo com o tamanho da instância foi idêntico ao previsto na análise de complexidade, exceto pelas heurísticas de inserção (seção 2.2) que apresentaram comportamento $O(n^4)$. Como a complexidade esta heurística é $O(M.n^3)$ conclui-se que M foi proporcional a *n*, onde M é um limitante superior do número de soluções. Portanto, o número de soluções não dominadas mantidas a cada iteração cresceu linearmente com o número de tarefas.

Tabela 3.5 – Tempo médio(s) para cada tamanho de instância

	n				
	20	**40**	**60**	**80**	**100**
ATCS(0)	0.00	0.00	0.00	0.01	0.01
TONN	0.02	0.05	0.12	0.20	0.31
VMPR	0.01	0.03	0.10	0.23	0.46
ATCS(12)	0.02	0.08	0.18	0.32	0.50
I_ATCS	0.00	0.02	0.07	0.28	0.85
I_MDD	0.00	0.02	0.09	0.34	1.13
I_NEA(2)	0.00	0.02	0.09	0.36	1.17
I_NEA(1)	0.00	0.02	0.09	0.37	1.24
I_NN	0.00	0.02	0.10	0.39	1.26
I_ALEAT	0.00	0.02	0.11	0.41	1.29
I_TLB	0.00	0.02	0.10	0.39	1.30
I_FAR(1)	0.00	0.02	0.11	0.43	1.41
I_EDD	0.00	0.02	0.12	0.45	1.42
I_FAR(2)	0.00	0.02	0.11	0.43	1.45
CI(3)	0.00	0.06	0.27	0.86	2.39
CI(5)	0.01	0.10	0.44	1.42	3.97

Na Tabela 3.6 temos as médias do desvio separadas por fator utilizado na geração da instância. Por exemplo, separam-se as instâncias em três grupos dependendo do valor de τ utilizado, e calcula-se a média do desvio em cada um destes grupos, apresentados na coluna τ da Tabela 3.6. O mesmo é feito para n, R e η. A heurística ATCS(12) mostrou-se robusta, obtendo os melhores resultados exceto para $\tau = 0.2$, onde TONN se apresentou melhor. Este é o caso em que o atraso esperado é pequeno, ou seja, as datas de entrega são folgadas.

Em Arroyo (2000) é sugerida a utilização de duas heurísticas construtivas para problemas biobjetivos conjuntamente, tomando-se as soluções não dominadas da união dos

dois conjuntos de soluções. Na tabela 3.7 estão os resultados dos 40 melhores pares de heurísticas, sendo que a união de boas heurísticas para o Makespan com as de bom desempenho para o Atraso obtiveram os melhores resultados.

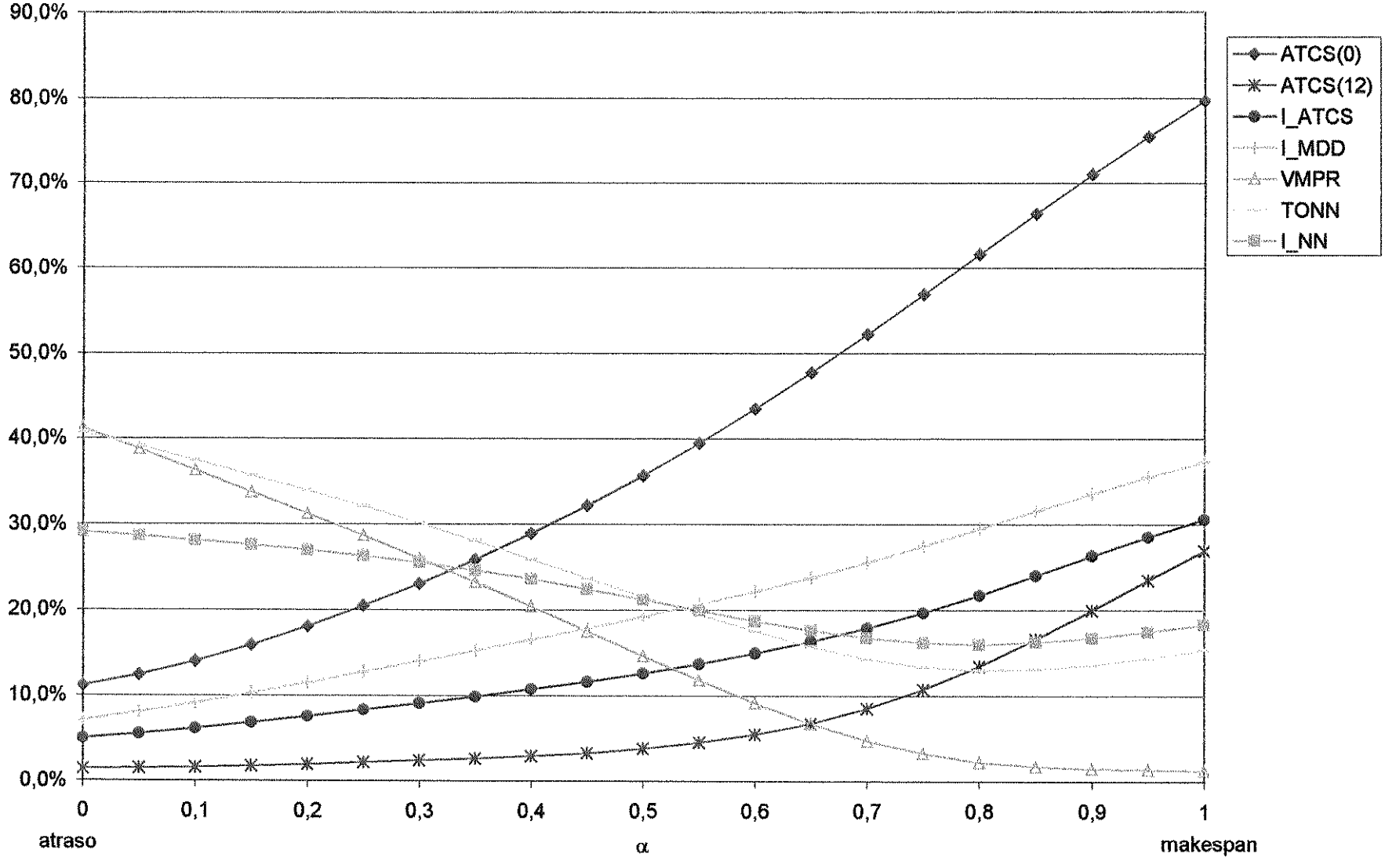

Figura 3.3 - Desvio %

Tabela 3.6 – Média do desvio por fator na geração da instância

	n					η			τ			R		
	20	**40**	**60**	**80**	**100**	**0.2**	**0.6**	**1**	**0.2**	**0.6**	**1**	**0.2**	**0.6**	**1**
ATCS(0)	39.8%	36.8%	38.2%	41.3%	42.2%	37.7%	38.4%	42.9%	31.9%	41.5%	45.6%	39.9%	38.7%	40.4%
ATCS(12)	**10.6%**	**6.0%**	**5.4%**	**8.0%**	**7.4%**	**5.5%**	**7.0%**	**9.9%**	11.0%	**5.6%**	**5.8%**	**6.3%**	**8.0%**	**8.2%**
I_ATCS	11.7%	12.6%	15.4%	16.2%	17.0%	10.8%	15.6%	17.4%	12.9%	16.3%	14.6%	18.1%	14.1%	11.6%
I_EDD	22.4%	24.5%	25.5%	25.9%	24.7%	14.0%	26.6%	33.1%	21.6%	23.8%	28.4%	29.9%	25.5%	18.3%
I_ALEAT	25.4%	34.3%	35.6%	34.9%	35.2%	23.2%	34.7%	41.3%	37.5%	33.3%	28.5%	38.2%	32.2%	28.9%
I_FAR(1)	31.0%	34.3%	36.7%	36.5%	35.9%	25.5%	35.4%	43.8%	38.2%	35.2%	31.3%	37.5%	38.0%	29.1%
I_FAR(2)	31.9%	37.3%	37.1%	39.8%	36.3%	26.2%	36.9%	46.3%	39.0%	36.3%	34.1%	40.6%	37.7%	31.2%
I_NEA(1)	21.0%	28.8%	31.8%	31.0%	31.3%	21.1%	29.5%	35.7%	32.2%	30.4%	23.8%	31.0%	29.1%	26.2%
I_NEA(2)	25.0%	31.5%	33.7%	34.5%	32.5%	22.7%	32.7%	39.0%	35.2%	31.6%	27.6%	34.6%	30.9%	28.9%
I_TLB	24.7%	26.3%	28.4%	29.1%	26.4%	15.2%	27.4%	38.3%	21.5%	25.9%	33.5%	33.8%	26.0%	21.0%
I_MDD	18.9%	19.9%	21.4%	21.1%	21.3%	13.9%	21.0%	26.7%	22.3%	20.5%	18.8%	26.7%	20.0%	15.0%
I_NN	17.8%	19.9%	23.4%	24.9%	23.7%	16.6%	22.8%	26.4%	25.2%	21.3%	19.2%	22.2%	23.5%	20.1%
VMPR	10.7%	15.2%	18.3%	20.2%	20.0%	19.4%	16.5%	14.8%	13.1%	19.1%	18.5%	13.8%	16.4%	20.4%
TONN	23.7%	22.6%	23.1%	24.9%	24.4%	26.5%	23.6%	21.1%	**9.7%**	22.2%	39.3%	28.9%	21.3%	21.0%
CI(3)	23.1%	31.6%	30.7%	31.3%	32.4%	27.6%	30.1%	31.8%	32.0%	29.8%	27.8%	30.0%	29.3%	30.3%
CI(5)	20.4%	28.7%	29.8%	30.0%	30.9%	26.2%	28.0%	29.8%	30.3%	28.3%	25.4%	28.6%	27.2%	28.1%

Tabela 3.7 – Resultados das melhores combinações de heurísticas

Desvio		
ATCS(12)	VMPR	1.74%
ATCS(12)	TONN	4.22%
ATCS(12)	I_ATCS	5.27%
ATCS(12)	I_NN	5.31%
I_ATCS	VMPR	5.45%
ATCS(12)	I_TLB	6.24%
ATCS(12)	I_MDD	6.34%
ATCS(12)	I_NEA(1)	6.47%
ATCS(12)	CI(5)	6.50%
ATCS(12)	I_EDD	6.58%
ATCS(12)	CI(3)	6.58%
ATCS(12)	I_NEA(2)	6.74%
ATCS(12)	I_FAR(1)	6.89%
ATCS(12)	I_ALEAT	6.90%
ATCS(12)	I_LPT(2)	6.98%
ATCS(12)	I_FAR(2)	7.05%
ATCS(12)	I_LPT(1)	7.06%
I_MDD	VMPR	7.21%
ATCS(0)	ATCS(12)	7.51%
I_ATCS	TONN	8.76%
I_EDD	VMPR	9.62%
I_TLB	VMPR	9.94%
ATCS(0)	VMPR	10.03%
I_ATCS	I_NN	10.73%
VMPR	CI(5)	11.11%
I_MDD	TONN	11.16%
VMPR	CI(3)	11.81%
VMPR	TONN	11.97%
VMPR	I_NN	12.37%
I_ATCS	I_MDD	12.96%
I_NEA(1)	VMPR	13.03%
I_ALEAT	VMPR	13.44%
I_ATCS	I_NEA(1)	13.45%
I_ATCS	CI(5)	13.47%
TONN	I_NN	13.60%
I_NEA(2)	VMPR	13.66%
I_ATCS	CI(3)	13.67%
I_ATCS	I_EDD	13.79%
I_ATCS	I_TLB	13.82%
TONN	CI(5)	13.83%

Distância		
ATCS(12)	VMPR	0.032
I_ATCS	VMPR	0.049
I_MDD	VMPR	0.069
ATCS(12)	TONN	0.073
ATCS(12)	I_NN	0.076
I_EDD	VMPR	0.087
ATCS(12)	I_ATCS	0.087
I_TLB	VMPR	0.091
I_ATCS	TONN	0.097
I_ATCS	I_NN	0.100
ATCS(12)	I_TLB	0.101
ATCS(12)	I_MDD	0.101
ATCS(12)	I_NEA(1)	0.103
ATCS(12)	I_EDD	0.103
ATCS(12)	CI(5)	0.107
ATCS(12)	I_NEA(2)	0.107
ATCS(12)	CI(3)	0.108
ATCS(12)	I_ALEAT	0.109
ATCS(12)	I_LPT(2)	0.111
ATCS(12)	I_FAR(1)	0.112
ATCS(12)	I_LPT(1)	0.112
ATCS(12)	I_FAR(2)	0.113
VMPR	I_NN	0.114
I_MDD	TONN	0.123
ATCS(0)	ATCS(12)	0.124
I_NEA(1)	VMPR	0.127
I_MDD	I_NN	0.128
TONN	I_NN	0.130
I_ALEAT	VMPR	0.131
I_NEA(2)	VMPR	0.134
VMPR	TONN	0.135
I_FAR(1)	VMPR	0.136
I_FAR(2)	VMPR	0.137
I_LPT(2)	VMPR	0.138
VMPR	CI(5)	0.139
I_LPT(1)	VMPR	0.140
I_EDD	TONN	0.143
I_EDD	I_NN	0.145
I_ATCS	I_NEA(1)	0.147
I_TLB	I_NN	0.148

4 *Busca Local Multiobjetivo*

Algoritmos de busca local são largamente utilizados em otimização combinatória, merecendo uma publicação exclusiva sobre este assunto (Aarts e Lenstra, 1997, "Local search in Combinatorial Optimization"). Estes algoritmos tentam melhorar uma solução fazendo modificações na mesma, sendo também conhecidos como algoritmos de melhoria. Uma modificação pode ser vista como um movimento para uma solução vizinha, e o desempenho depende da definição da vizinhança, que determina as possíveis modificações. A aplicação de busca local a programação combinatória multiobjetivo é recente, sendo este trabalho baseado principalmente em Arroyo and Armentano (2000) e em Baykasoglu et al. (1999). Outras referencias são Czyac e Janzkiewicz (1998), Hansen (1997) e Viana e Sousa (2000).

Resultados para otimização de um único objetivo no problema de minimização da soma dos atrasos em uma máquina com tempos de preparação dependentes da seqüência são apresentados em Tan et al. (2000), onde um algoritmo de *multiple start* baseado na troca entre pares adjacentes (API – *Adjacent Pairwise Interchange*) obtém resultados competitivos com uma implementação de simulated annealing e é melhor que um algoritmo genético desenvolvido para o problema. Para o mesmo problema, França et al. (2001) obteve bons resultados com um algoritmo genético híbrido, também denominado algoritmo memético, utilizando uma combinação da vizinhança de inserção com a vizinhança de troca entre pares. Armentano e Mazzini (2000) também utilizaram um algoritmo genético para este problema.

Para o problema do caixeiro viajante (PCV) as vizinhanças mais utilizadas são baseadas na troca de arestas, como 2-opt e 3-opt, onde r-opt significa que no máximo r arestas são modificadas num movimento de melhoria. Johnson e McGeoch (1997) descrevem implementações bem sucedidas destas vizinhanças para o caso simétrico aplicadas a instâncias de até um milhão de cidades. Buriol (2000), desenvolveu um algoritmo genético híbrido para o problema assimétrico baseado na troca de arcos.

Para otimização mono-objetivo os movimentos de melhoria são realizados até que se atinja um ótimo local, conforme a definição 4.1. No caso multiobjetivo a busca é executada até que o conjunto P, que busca aproximar o conjunto de soluções eficientes, seja localmente eficiente, conforme a definição 4.2.

Definição 4.1. Uma solução x é um ótimo local se, e somente se:

$$f(x) \leq f(y),\ \forall y \in N(x)$$

$$f: X \rightarrow \Re$$

onde f é a função objetivo, X o espaço de soluções e N(x) designa a vizinhança de x.

Definição 4.2. Um conjunto P é localmente eficiente se, e somente se:

dado $x \in P$, não existe $y \in N(P)$ tal que y domine x, onde $N(P) = \cup_{x \in P} N(x)$.

É considerada a reinicialização da busca a partir de diferentes soluções iniciais, caracterizando um procedimento de *multiple start* (múltiplos recomeços). Uma implementação de busca local é formada por três principais componentes:

- Vizinhança, que é a forma de gerar novas soluções a partir de soluções existentes.
- Estratégia de busca, que define como utilizar a vizinhança na busca de um ótimo local ou conjunto localmente eficiente. Exemplos de estratégias de busca mono-objetivo seriam *Best Improvement*, onde a vizinhança inteira é visitada antes de ser realizado o melhor movimento, e *First Improvement*, onde o movimento é realizado assim que se encontra uma solução melhor que a atual.
- Método de reinicialização da busca, que após a busca local produzir um conjunto localmente eficiente (definição 4.2), gera novas soluções inicias.

Este capítulo descreve as diferentes implementações realizadas de cada um destes componentes para o caso multiobjetivo, e também uma proposta de preprocessamento a ser realizada nas soluções iniciais antes de ser realizada a busca.

Apesar de ser comum para o PCV, a vizinhança 2-opt não é considerada por ser impossível no problema assimétrico modificar apenas 2 arcos numa solução. Este fato é exemplificado na Figura 4.1, onde a tentativa de modificar apenas dois arcos reverte a seqüência de processamento de todas as tarefas entre estes arcos e, portanto, modifica 5 arcos (tempos de preparação) destas tarefas, caracterizando um movimento 5-opt. Se fossem arestas (arcos simétricos, $s_{ij} = s_{ji}$), apenas duas seriam modificadas.

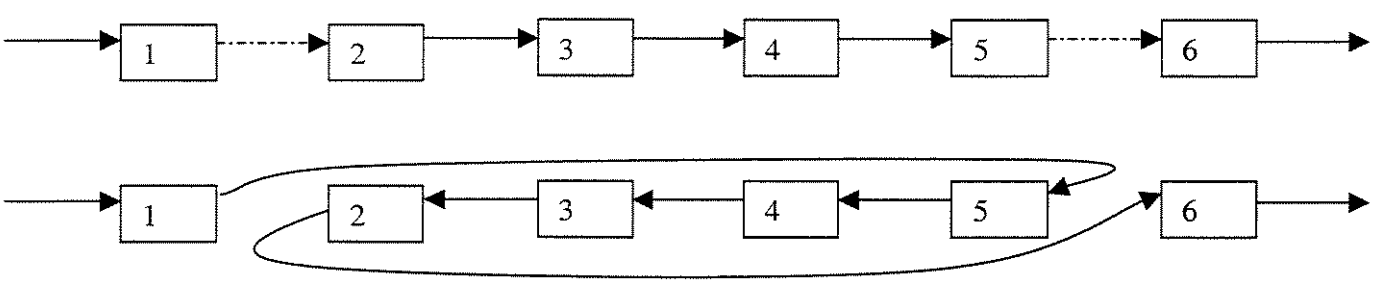

Figura 4.1 – Tentativa de movimento 2-opt que resulta em movimento 5-opt

4.1 Estratégia de busca

Para otimização mono-objetivo existem duas estratégias principais para realizar uma busca local: *Best Improvement* e *First Improvement*. O caso multiobjetivo é muito mais complexo, permitindo diferentes estratégias, uma vez que temos não uma mas um conjunto de vizinhanças aonde se deve realizar a busca. As estratégias implementadas são descritas a seguir.

4.1.1 Estratégia Nfirst (New First)

A estratégia Nfirst é baseada no trabalho de Baykasoglu et al., que propõe uma busca tabu para problemas multiobjetivo. O nome Nfirst – New First - se deve ao fato da busca ser realizada entre as novas soluções encontradas na última vizinhança visitada. Caso uma vizinhança não adicione novas soluções a P (aproximação para o conjunto eficiente), a busca é realizada na mais antiga solução não visitada (uma solução é dita visitada se sua vizinhança já foi analisada pelo algoritmo).

Estratégia NFirst

1. Dado um conjunto de soluções inicial P
2. Selecione aleatoriamente uma solução semente s em P
3. Seja V o conjunto de vizinhos de s que não são dominados por nenhuma solução em P ou em V, ou seja: $V = \{x \in N(s)$: não existe $y \in P \cup N(s)$ onde y domina $x\}$
4. Marque s como uma solução visitada
5. Se $V \neq \varnothing$

   selecione aleatoriamente uma nova semente s em V, atualize P com as soluções em V e volte ao passo 3.
6. Se todas as soluções em P foram visitadas

   PARE, P é um conjunto localmente eficiente (definição 4.2)
7. Selecione a solução não visitada mais antiga em P como solução semente e volte ao passo 3

4.1.2 Estratégia *r* - grupos

Arroyo e Armentano (2000) propõem que as soluções sejam divididas em *r* grupos segundo o valor de um dos objetivos. Por exemplo, o grupo 1 contém as soluções de melhor *makespan*, enquanto o grupo *r* as soluções de pior *makespan*. A busca é feita paralelamente em uma solução escolhida aleatoriamente em cada grupo.

A idéia é realizar a busca em diferentes regiões no espaço das soluções ao mesmo tempo. A Figura 4.2 exemplifica este procedimento para 3 grupos, onde os pontos representam um conjunto de soluções não visitadas no espaço das soluções.

Observe-se que $r = 1$ equivale a escolher aleatoriamente a próxima solução a ser visitada, enquanto $r = \infty$ implica que todas as soluções não visitadas serão analisadas em paralelo.

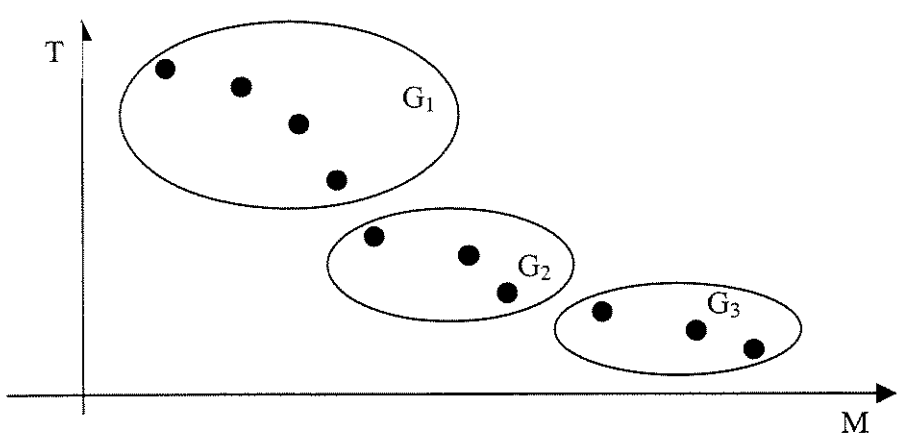

Figura 4.2 – Exemplo de agrupamento para $r = 3$

Procedimento r - grupos

1. Dado um conjunto de soluções inicial P
2. Seja nv = número de soluções não visitadas de P
3. Se $nv = 0$

   PARE, P é um conjunto localmente eficiente (definição 4.2)
4. Se $nv \leq r$,

   faça B = conjunto de soluções não visitadas em P

   Vá para o passo 8
5. Ordene as soluções não visitadas pelo valor do *makespan* e forme r grupos G_i:

   $|G_i| - |G_j| \leq 1$, $\forall$i, j

   $M_x \leq M_y$, $\forall x \in G_i$, $y \in G_{i+1}$ (M_x é o valor do *makespan* da solução x, ver Figura 4.2 exemplificando a formação dos grupos).
6. Seja o conjunto de soluções a serem visitadas $B = \varnothing$
7. Em cada grupo G_i

   selecione aleatoriamente uma solução x_i

   faça $B = B \cup x_i$
8. Seja o conjunto de novas soluções não dominadas $V = \varnothing$
9. Para cada solução s em B

   Marque s como uma solução visitada

   Adicione a V os vizinhos de s não dominados por soluções em P ou V
10. Atualize P com as soluções em V e volte ao passo 2

4.1.3 Estratégias *TFirst e MFirst*

É natural supor que visitando-se primeiro as soluções de menor atraso este objetivo tende a ser privilegiado, o mesmo podendo ser imaginado para o *makespan*. As estratégias aqui propostas visam observar os efeitos de se priorizar as soluções com melhor valor para um dos objetivos, onde TFirst (*Tardiness First*) prioriza o atraso total e MFirst (*Makespan First*) prioriza o *makespan*.

Estratégia TFirst (MFirst)

1. Dado um conjunto de soluções inicial P
2. Selecione s como a solução não visitada de menor atraso total(*makespan*) em P
3. Atualize P com N(s), a vizinhança de s
4. Se todas as soluções em P foram visitadas

   PARE, P é um conjunto localmente eficiente (definição 4.2)
5. volte ao passo 2

4.1.4 Estratégia Mista

Geralmente a busca local é reiniciada várias vezes durante a resolução de uma instância. A estratégia Mista escolhe aleatoriamente entre as estratégias MFirst e TFirst, com igual probabilidade, de forma que se executada diversas vezes aproximadamente metade das buscas serão realizadas com a estratégia TFirst e metade com a estratégia MFirst. Depois de escolhida a estratégia entre TFirst e MFirst, esta é mantida até que se atinja um conjunto localmente eficiente. A estratégia Mista é uma tentativa de unir as qualidades de TFirst e MFirst quando a busca é realizada diversas vezes.

4.2 Preprocessamento

As estratégias de busca analisadas na seção 4.1 sempre visitam todos os vizinhos de uma solução antes de atualizar P, o conjunto de soluções não dominadas encontradas até o momento. Este procedimento equivale à estratégia Best Improvement na otimização de um único objetivo.

Numa analogia com a estratégia First Improvement, propõe-se a utilização de um preprocessamento visando obter soluções que não sejam dominadas por seus vizinhos antes de realizar uma das estratégias de busca. Pretende-se evitar uma

busca profunda na vizinhança de soluções de baixa qualidade, concentrando o esforço computacional próximo à fronteira eficiente.

Preprocessamento

1. Dado um conjunto de soluções inicial P
2. Selecione aleatoriamente uma semente s não visitada em P
3. Enquanto existir $x \in N(s)$ que domine s

   Faça $s = x$
4. Marque s como uma solução visitada
5. Atualize P com s
6. Se todas as soluções em P foram visitadas

   PARE
7. volte ao passo 2

O preprocessamento é realizado antes da busca local ser realizada segundo as estratégias da seção 4.1.

Note-se que no passo 3 do preprocessamento procuram-se apenas soluções que dominem s. Para dominar s, uma solução x precisa ser melhor para o *makespan* e para o atraso. Em nosso problema avaliar o valor do *makespan* de um vizinho pode ser feito em tempo constante enquanto o atraso requer $O(n)$ para as vizinhanças testadas. Assim, somente os movimentos que melhorem o *makespan* serão avaliados para o atraso no passo 3, poupando tempo na avaliação da vizinhança.

4.3 Vizinhanças

A seguir são descritas as diferentes vizinhanças implementadas, assim como estratégias para redução do tamanho de algumas destas vizinhanças.

4.3.1 Troca entre pares adjacentes (Adjacent pairwise interchange - API)

A vizinhança API considera todas as possíveis trocas entre pares de tarefas que sejam consecutivas na solução atual.

Por exemplo: suponha que a solução atual é [1 2 3 4] (isto significa que a primeira tarefa a ser processada é a 1, seguida da tarefa 2, depois a 3 e por fim a tarefa 4). Os vizinhos são [**2 1** 3 4], [1 **3 2** 4] e [1 2 **4 3**]. O número de vizinhos, também conhecido como tamanho da vizinhança, é $n - 1$.

4.3.2 Troca (Pairwise interchange - PI)

Esta vizinhança é formada trocando-se as posições de qualquer par de tarefas na solução atual. O tamanho da vizinhança é $n(n\text{-}1)/2$.

Exemplo: os vizinhos de [1 2 3 4] são [**2 1** 3 4], [**3** 2 **1** 4], [**4** 2 3 **1**] , [1 **3 2** 4] , [1 **4** 3 **2**] e [1 2 **4 3**].

4.3.3 Troca reduzida (PI - RED)

França et al. (2001) propõe algumas reduções de vizinhanças para a vizinhança PI, e a que gerou os melhores resultados, denotada aqui por troca reduzida (PI-RED), é implementada neste trabalho. Suponha uma troca entre as tarefas *i* e *j* como nas Figuras 4.3 e 4.4.

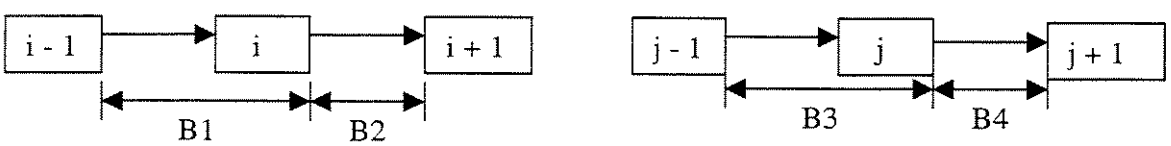

Figura 4.3 – Configuração antes da troca entre *i* e *j*

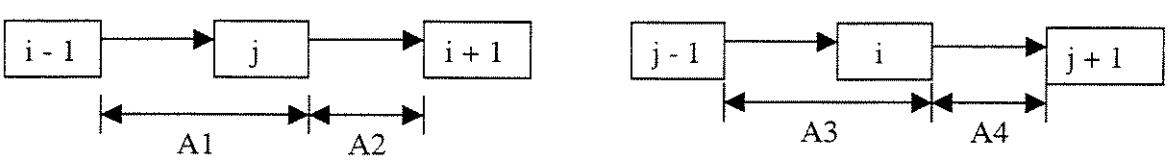

Figura 4.4 – Configuração após a troca entre *i* e *j*

Seja: $B1 = s_{i-1,i} + p_i$; $B2 = s_{i,i+1}$; $B3 = s_{j-1,j} + p_j$; $B4 = s_{j,j+1}$;

$A1 = s_{i-1,j} + p_j$; $A2 = s_{j,i+1}$; $A3 = s_{j-1,i} + p_i$; $A4 = s_{i,j+1}$.

Uma troca só é considerada na vizinhança reduzida se:

[(A1 < B1) ou (A2 < B2)] e [(A3 < B3) ou (A4 < B4)].

O tamanho da vizinhança reduzida foi de apenas 5% da vizinhança original nos testes realizados por França et al. (2001).

4.3.4 Inserção (INS)

Um movimento de inserção consiste em inserir uma tarefa em uma posição diferente da seqüência. As inserções podem ser classificadas como anterior e posterior, conforme a nova posição da tarefa seja antes ou depois da posição atual. Observe-se que França et al. (2001) considera apenas movimentos posteriores na vizinhança de inserção, enquanto este trabalho também considera os movimentos anteriores. Desta forma, o tamanho da vizinhança é de $(n\text{-}1)^2$.

Exemplo: Os vizinhos "posteriores" a partir de [1 2 3 4] são [2 **1** 3 4],

[2 3 **1** 4] , [2 3 4 **1**] , [1 3 **2** 4] , [1 3 4 **2**] e [1 2 4 **3**].

Já os vizinhos "anteriores" são [**2** 1 3 4], [1 **3** 2 4] , [**3** 1 2 4] , [1 **4** 2 3], [1 2 **4** 3] e [**4** 1 2 3]. Os vizinhos que são posteriores e anteriores são gerados apenas uma vez.

4.3.5 Inserção reduzida (INS - RED)

Assim como para PI (*pairwise interchange*), França et al. (2001) propõe reduções para a vizinhança de inserção, e a redução com melhores resultados é implementada aqui. Suponha que um movimento a ser avaliado seja inserir a tarefa i antes de j como nas Figuras 4.5 e 4.6.

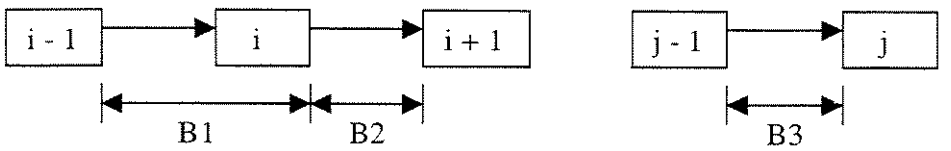

Figura 4.5 –Configuração antes da inserção de *i* antes de *j*

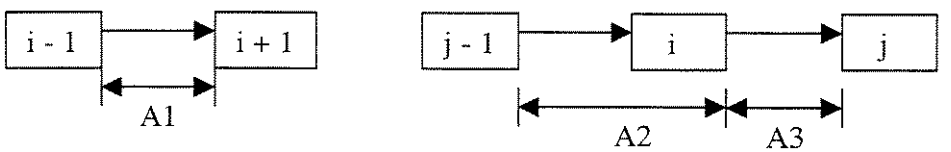

Figura 4.6 – Configuração após a inserção de *i* antes de *j*

Seja: $B1 = s_{i-1,i} + p_i$; $B2 = s_{i,i+1}$; $B3 = s_{j-1,j}$;

$A1 = s_{i-1,i+1}$; $A2 = s_{j-1,i} + p_i$; $A3 = s_{i,j} + p_i$

Um movimento é considerado na vizinhança reduzida somente se:

$(A1 < B1 + B2)$ ou $(A2 + A3 < B3)$.

Esta regra reduziu a vizinhança a aproximadamente 50% do tamanho original em França et al. (2001).

4.3.6 Inserção e troca (PI + INS)

É utilizada a união das vizinhanças de troca e inserção, que em França et al. (2001) gerou os melhores resultados na minimização do atraso, bem como a união das vizinhanças reduzidas (PI+INS RED).

4.3.7 Or-OPT

A vizinhança Or – OPT foi proposta em Or (1976) para o problema do caixeiro viajante. Um vizinho é obtido a partir da inserção de uma, duas ou três tarefas consecutivas em uma posição diferente da seqüência atual, sendo consideradas inserções anteriores e posteriores à posição atual das tarefas na seqüência. O tamanho desta vizinhança é $O(n^2)$.

Exemplos: seja a solução atual [1 2 3 4 5 6 7];

- a inserção da tarefa 1 após a 6 geraria [2 3 4 5 6 **1** 7],

- a inserção de 1 e 2 após 6 geraria [3 4 5 6 **1 2** 7],
- a inserção de 1, 2 e 3 após 6 geraria [4 5 6 **1 2 3** 7].

4.3.8 3-OPT

A vizinhança 3-opt é usualmente aplicada ao problema do caixeiro viajante, e consiste da troca de três arcos na solução corrente. A vizinhança de inserção está contida em Or-opt que está contida em 3-opt, uma vez que um movimento de inserção ou Or-opt modificam apenas 3 arcos.

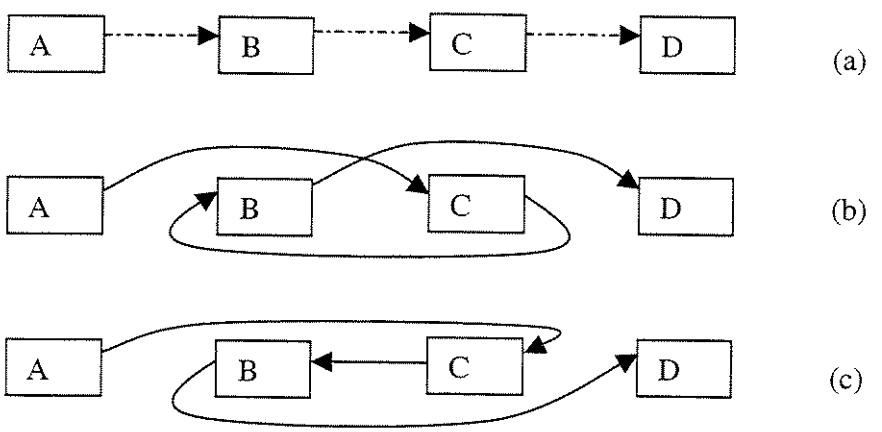

Figura 4.7 – Os blocos representam subseqüências de tarefas. A solução original e os arcos removidos estão em (a). Em (b) encontra-se o único movimento permitido no caso assimétrico, enquanto (b) e (c) são válidos para o caso simétrico, pois os arcos dentro dos blocos B e C no item (c) são invertidos.

Em nosso problema os tempos de setup correspondem aos valores dos arcos, onde estes valores são assimétricos. Uma vez escolhidos três arcos a serem retirados da solução existem 7 maneiras, para o caso simétrico, de reconectar as subseqüências gerando novas soluções. Já para o caso assimétrico existe apenas uma forma de reconectar as subseqüências sem revertê-las, pois reverter uma subseqüência modificaria mais de 3 arcos, sendo apenas um movimento possível para cada três arcos removidos. A Figura 4.7 ilustra este fato, onde os blocos representam subseqüências de tarefas, onde apenas o movimento (b) é permitido dados os arcos escolhidos na solução (a) a serem retirados.

Para o problema de seqüenciamento de tarefas em uma máquina, é necessária uma tarefa artificial indicando a primeira tarefa a ser processada. Seja esta tarefa a de índice 0, com $s_{0j} \neq 0$ e $s_{j0} = 0$. Considere agora o problema assimétrico do caixeiro viajante iniciando e terminando na cidade ou tarefa 0. Ao se retirar os arcos (i, j), (k, l) e (r, s) da solução S, a única solução vizinha S' é obtida pela inserção de (i, l), (r, j) e (k,s). Assim em S e S' temos:

S: 0 - - - - -(i, j) - - - - - (k, l) - - - - -(r, s) - - - - - 0

S': 0 - - - - -(i, l) - - - - - (r, j) - - - - -(k, s) - - - - - 0

Portanto, a troca de exatamente 3 arcos implica que um movimento 3-opt é equivalente a uma inserção de um bloco em que nenhuma subseqüência é invertida.

Seja o bloco B = (j - - - - - k)

S: 0 - - - - - i - B - l - - - - - r – s - - - - - 0

Saem os arcos (i, j), (k, l) e (r, s)

Inserindo B após r, tem-se

S'': 0 - - - - - i – l - - - - - r - B – s - - - - - 0

Entram os arcos (i, l), (r, j) e (k, s) da mesma forma que em S'.

Exemplo:

Seja a solução atual [1 2 3 4 5 6 7 8 9] ; inserindo-se o bloco [6 7] após a tarefa 2 tem-se [1 2 **6 7** 3 4 5 8 9], neste caso os arcos removidos são (5, 6), (7, 8) e (2, 3), sendo inseridos os arcos (2, 6), (7, 3) e (5, 8).

O tamanho da vizinhança 3-opt completa é de $n(n\text{-}1)(n\text{-}2)/6$ para o PCVA, e de $n(n+1)(n\text{-}1)/6$ para o problema de seqüenciamento de tarefas dada a tarefa artificial necessária. A diferença ocorre por que além de seqüenciar as tarefas é necessário determinar qual a primeira tarefa a ser processada. Este tamanho de vizinhança é alto para instâncias com muitas tarefas. Para o PCV, e portanto para o *makespan*, não é necessário avaliar todos os vizinhos possíveis para garantir que a solução é um ótimo local. Johnson e McGeoch (1997) descrevem um procedimento que retira e inclui um arco por vez seqüencialmente até obter um movimento 3-opt. Este procedimento foi adaptado para o problema de *scheduling* da seguinte maneira

(suponha para simplicidade da notação que i+1 é a tarefa processada após i na solução atual, e i-1 a tarefa anterior a i, sendo que após a tarefa n tem-se a tarefa 0):

Procedimento 3-OPT (β)

1. Para cada tarefa i (observe-se a inclusão da tarefa artificial 0) construa uma lista L_i com as demais tarefas j ordenadas por valores crescentes de s_{ij}
2. Seja $folga = \beta.s_{med}$, onde s_{med} é a média de s_{ij} e β um parâmetro dado
3. Para cada tarefa i

   arcmax = $s_{i,i+1} + folga$,

   Para cada tarefa j com $s_{i,j}$ < arcmax (use L_i para este teste)

   arcmax2 = $s_{j-1,j} + s_{i,i+1} - s_{i,j} + 2.folga$

   Para cada tarefa k com $s_{j-1,k}$ < arcmax2 (use L_{j-1} para este teste)

   Verifique a factibilidade de incluir os arcos $s_{i,j}$, $s_{j-1,k}$ e $s_{k-1,i+1}$

   Se factível, avalie o movimento (novo vizinho)

O procedimento 3-opt(β) primeiro escolhe um arco a ser removido (i, i+1), e tenta substitui-lo com um arco (i,j) menor que um dado limite (arcmax). A lista L_i é utilizada para encontrar os arcos que satisfazem esta condição até que uma tarefa da lista não a satisfaça, uma vez que as demais tarefas terão um tempo preparação s_{ij} ainda maior. Uma vez que um arco (i,j) seja incluído, é necessário remover o arco (j-1, j), e procura-se por um arco (j-1,k) menor que arcmax2 para reconectar a solução. Após a escolha de (j-1,k), o arco (k-1, k) precisa ser removido e portanto tem-se a definição de um movimento 3-opt que deve ser verificado quanto a factibilidade (ver figura 4.9) antes de ser considerado como vizinho.

Um movimento é factível se:

- $i > j$ e $k \in [j+1, i]$
- $i < j$ e $k \notin [i+1, j]$

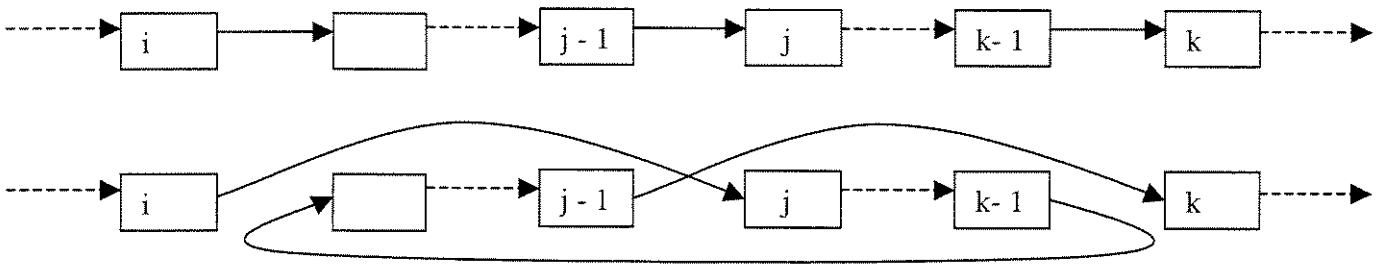

Figura 4.8 – Um movimento 3-OPT factível

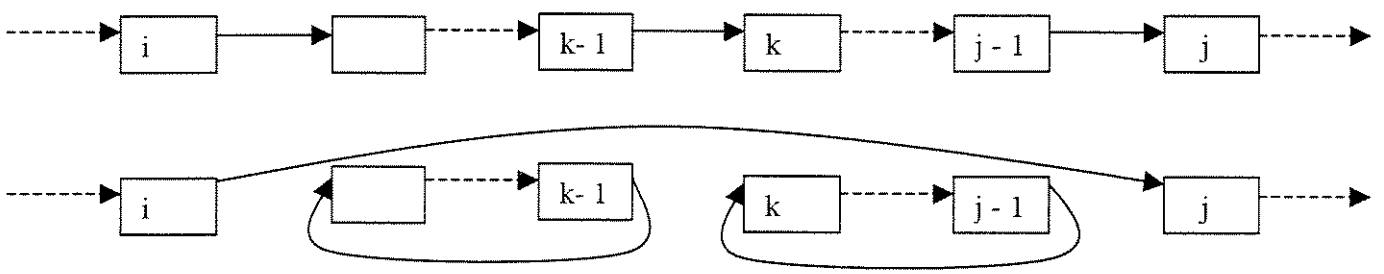

Figura 4.9 – Um movimento 3-OPT infactível

Quando implementado o algoritmo deve-se considerar uma tarefa artificial, no início da seqüência com os arcos correspondentes. Os valores de $s_{0,i}$ são dados e representam o tempo de preparação dado o estado inicial da máquina, enquanto que se define $s_{i,0} = 0$. Esta tarefa artificial é necessária para que se considere todas as possíveis inserções de uma subseqüência em uma posição diferente.

O procedimento com $\beta = 0$ avalia todos os movimentos 3-OPT que reduzem o *makespan* (ver Johnson, 1997). Porém, o mesmo não é verdadeiro para o atraso. O parâmetro β determina a quantidade de folga que será permitida no algoritmo. Esta folga não existe na proposta original de Johnson e aumenta o número de movimentos avaliados. A motivação é que, mesmo aumentando o valor do *makespan*, um movimento pode levar a um valor de atraso melhor, apesar de um aumento muito grande no valor do *makespan* geralmente causar um aumento no atraso total, dado que o instante de término de boa parte das tarefas aumenta. Assim, apenas pequenos aumentos no valor do *makespan* são permitidos.

A folga é o quanto um novo arco pode exceder o arco removido na solução atual. No cálculo de arcmax2 considera-se a inclusão de dois novos arcos, e portanto o valor da folga é multiplicado por 2. O valor da folga é calculado como β vezes o tempo de preparação médio da instância, sendo este parâmetro testado com diferentes valores no capítulo 5.

4.4 Estratégia *r* – grupos com vizinhança 3-opt variável

Espera-se que as melhores soluções para um objetivo sejam encontradas na vizinhança de soluções que já sejam boas para este objetivo. Observando-se o algoritmo 3-opt proposto em 4.3.8, verifica-se que a inclusão da folga é feita para aumentar a vizinhança procurando-se soluções que sejam melhores apenas para o atraso, uma vez que os vizinhos que melhoram o valor do *makespan* são avaliados sem a folga ($\beta = 0$).

Na estratégia com vizinhança variável que propomos aqui diminui-se o valor da folga conforme o valor do atraso da solução aumenta, isto é, soluções com menor atraso tem uma vizinhança maior. A base desta proposta é a estratégia *r*-grupos, seção 4.1.2, onde até *r* vizinhanças são analisadas em paralelo. Além de *r*, outro parâmetro utilizado é β_{max} , que corresponde ao maior valor de β utilizado numa busca 3-opt.

O procedimento é idêntico ao descrito na seção 4.1.2, onde apenas o valor β para a vizinhança 3-opt varia. Este valor , que determina a folga, é calculado da seguinte forma:

- Seja $nb = |B|$, isto é, nb é o número de vizinhanças que serão analisadas em paralelo uma vez que B é o conjunto de soluções a serem visitadas na iteração.
- Se $nb = 1$, utiliza-se $\beta = \beta_{max}$.
- Ordene as soluções em B por valores crescentes de atraso.
- Seja x_i a i-ésima solução em B , ou seja, x_1 a solução de menor atraso e x_{nb} a de maior atraso em B.
- O valor β_i utilizado na vizinhança de x_i é:

$$\beta_i = \beta_{max} \cdot \left(1 - \frac{i-1}{nb-1}\right) \qquad (4.1)$$

A equação 4.1 estabelece que a folga será de β_{max} para a solução de menor atraso (quando i = 1), decrescendo linearmente até atingir valor 0 para a solução de maior atraso (quando i = nb).

4.5 Múltiplos Recomeços (*multiple start*)

Diversos modos de utilizar uma busca local na otimização de um objetivo são propostos na literatura, como busca tabu, algoritmos genéticos híbridos, *simulated annealing* e outros. Recentemente estes métodos tem sido adaptados à otimização multiobjetivo, como em Hansen (1997), Viana e Sousa (2000) e Ishibuchi e Murata(1998).

Optou-se neste trabalho pela utilização do método *multiple start*, ou múltiplos recomeços, que corresponde a reiniciar a busca a partir de uma solução inicial diferente. Esta escolha baseou-se na simplicidade do método e nos bons resultados apresentados para o problema do caixeiro viajante (PCV) apresentados em Johnson e McGeoch (1997), onde o procedimento é denominado *Iterated*, como por exemplo *Iterated Lin-Kernighan*, onde uma iteração corresponde a uma execução do algoritmo de Lin e Kernighan (1973).

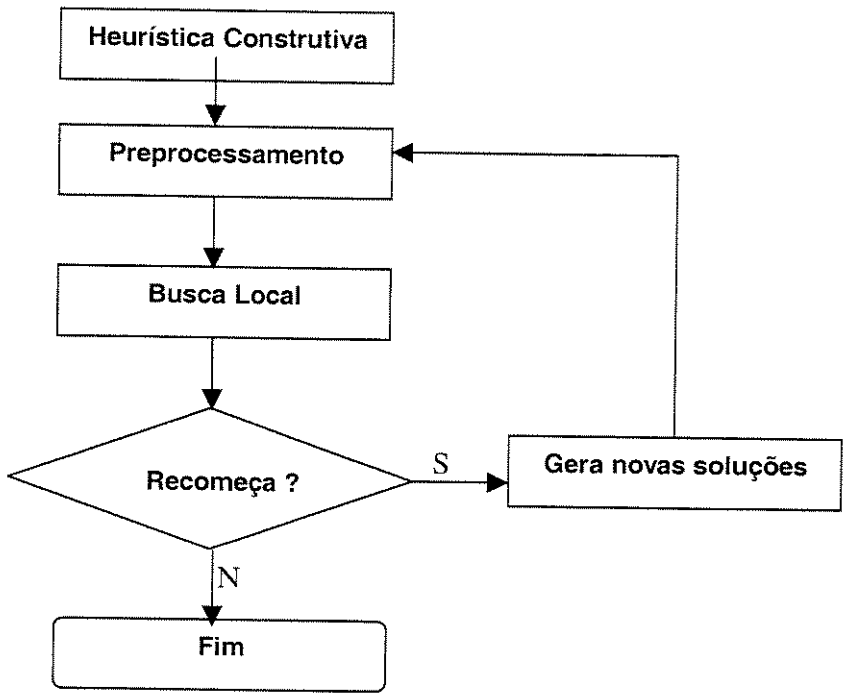

Figura 4.10 – Estrutura do algoritmo com múltiplos recomeços

A Figura 4.10 representa a estrutura com recomeços utilizada na aplicação da busca local. Para o caso multiobjetivo utiliza-se um conjunto inicial de soluções para a busca local, onde este conjunto deve conter uma ou mais soluções não dominadas entre si.

A forma mais simples de gerar novas soluções para a realização da busca é a geração de soluções aleatórias. Nesta forma de recomeço o conjunto de soluções inicial para a busca local contém apenas uma solução, cujo ordenamento das tarefas foi gerado aleatoriamente. Outra forma de gerar este conjunto inicial é realizar uma perturbação aleatória em soluções de boa qualidade. Quanto maior a perturbação, ou seja, quanto maior forem as modificações nas soluções, maior deve ser o esforço da busca local para atingir um conjunto localmente eficiente (definição 4.2). Entretanto, as perturbações não devem ser pequenas demais de forma que a busca local facilmente retorne as soluções originais, pois o objetivo de se reinicializar a busca é encontrar novas soluções de boa qualidade.

Johnson e McGeoch (1997) fazem uma analogia deste procedimento de perturbação com algoritmos genéticos. Nesta analogia uma perturbação corresponde a uma mutação, não existe *crossover* e cada iteração corresponde a uma geração, sendo a população composta de apenas um elemento por se tratar de otimização mono-objetivo.

A cada iteração k gera-se um conjunto localmente eficiente P_k, como resultado da busca local. Atualiza-se então um conjunto E de forma que ele contenha todas as soluções não dominadas encontradas durante as iterações 1, 2, ...k. Portanto, ao final do algoritmo, E contém o conjunto solução encontrado que aproxima a fronteira eficiente. A escolha das soluções de boa qualidade a serem perturbadas foi feita de duas maneiras:

- Perturbar todas as soluções em P_k e tomar as soluções resultantes não dominadas entre si como conjunto inicial para a iteração k + 1.
- O mesmo mas ao invés de P_k, perturbar o conjunto E.

As seções 4.5.1 e 4.5.2 descrevem os tipos de perturbação testados.

4.5.1 Troca entre tarefas

França et al. (2001) utiliza como operador de mutação a troca de posição entre duas tarefas aleatoriamente escolhidas. Generaliza-se este operador como uma troca de posições entre *r* tarefas aleatoriamente selecionadas da seguinte forma:

1. Selecione *r* tarefas t_1 , t_2 ... t_r aleatoriamente tal que $t_i \neq t_j$ para $i \neq j$
2. Faça a tarefa t_i ocupar a posição da tarefa t_{i+1} para $i < r$
3. Faça t_r ocupar a posição da tarefa t_1

4.5.2 Double Bridge

O movimento conhecido como double bridge é utilizado no algoritmo "Iterated Lin-Kernighan" descrito em Johnson e McGeoch (1997) como perturbação antes de uma nova iteração. Ele foi destacado primeiramente em Lin e Kernigham (1973) por ser um movimento 4-opt que não pode ser encontrado pelo algoritmo proposto naquele trabalho. O movimento pode ser interpretado como a troca de posição entre duas subseqüências, como exemplificado na Figura 4.11 onde cada bloco representa uma subseqüência de tarefas.

Este movimento foi implementado selecionando-se aleatoriamente duas subseqüências e trocando as mesmas de posição. As subseqüências devem ser escolhidas de forma que não haja intersecção entre elas, isto é, não exista uma mesma tarefa nas duas subseqüências.

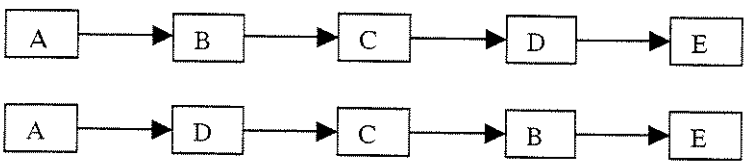

Figura 4.11 – Exemplo de movimento double bridge onde as subseqüências D e B trocam de posição

5 *Testes para busca local*

No capítulo 4 são descritas diferentes possibilidades de escolha para a vizinhança, a estratégia de busca e o método de recomeço. Testar todas as combinações possíveis para a formação de um algoritmo implicaria em um número excessivamente grande de testes. Optou-se por testar um item de cada vez, mantendo-se fixa as demais partes do algoritmo, e utilizando-se as melhores escolhas nos testes posteriormente realizados. Espera-se que este procedimento leve à definição de um bom algoritmo para a resolução do problema, além de possibilitar a análise do impacto de cada escolha no desempenho.

Todos os resultados e estatísticas foram recalculados após serem concluídos todos os testes apresentados neste capítulo. Isto porque as medidas são relativas, isto é, o valor do desvio, por exemplo, depende das melhores e piores soluções que estão sendo comparadas. Desta forma, todos os testes estão padronizados de forma que os valores de diferentes tabelas podem ser comparados entre si para o mesmo conjunto de instâncias.

As soluções não-dominadas encontradas em cada instância serão chamadas de soluções de referência, isto é, o conjunto obtido tomando-se as soluções não

dominadas dentre todas as soluções encontradas para a instância. Este conjunto tem o mesmo papel de um limitante superior num problema de minimização, servindo como parâmetro para avaliar a qualidade de um conjunto de soluções. Dado o número de testes realizados em cada instância, supõe-se que estas soluções de referência sejam bastante próximas das soluções Pareto – ótimas para a instância.

Os testes realizados seguiram a estrutura descrita na seção 4.5, onde a heurística construtiva utilizada foi a união das soluções obtidas por VMPR e por ATCS(12), que obteve os melhores resultados nos testes descritos no capítulo 3.

5.1 Implementação

A implementação dos algoritmos foi feita de forma idêntica à descrita no capítulo 3, onde cada solução é armazenada como um vetor com a seqüência de processamento, um segundo vetor com o instante de término e outro com a soma dos atrasos das tarefas até ela. O computador utilizado também foi o mesmo. Para a vizinhança 3-opt, utilizou-se um vetor adicional, indicando a posição de cada tarefa na seqüência, além de uma lista de vizinhos mais próximos para cada tarefa, onde as tarefas são ordenadas de forma crescente pelo tempo de preparação (ver seção 4.3.8). Esta lista é implementada como um vetor com os índices das tarefas.

Suponha, para simplificar a notação, que a solução x atual seja [1 2 .. n]. Seja um vizinho y tal que antes de uma tarefa i e a partir da tarefa j a seqüência seja a mesma que em x, ou seja, y só difere de x nas posições entre i e j, devido a um movimento realizado. Os códigos implementados para busca local realizam o seguinte procedimento na avaliação de um movimento:

Procedimento para avaliar um movimento

1. Obtenha o instante de término da tarefa *i*, C_i , utilizando o vetor auxiliar, assim como T_i, a soma dos atrasos das tarefas até *i*.
2. A partir da tarefa *i* e utilizando C_i e T_i, calcule os novos valores C'_k e T'_k se o movimento for realizado, para $i<k\leq j$.
3. Seja C'_j e T'_j os novos valores para a tarefa *j* ; $\Delta C_j = C'_j - C_j$ e $\Delta T_j = T'_j - T_j$
4. Se $\Delta C_j \geq 0$ e $\Delta T_j \geq 0$, o movimento pode ser descartado. Caso contrário, avalie os novos valores para o *makespan* e para a soma dos atrasos.

No passo 4 um movimento pode ser descartado porque, para $k > j$:

$\Delta C_k = \Delta C_j \geq 0$

$T'_k = T'_{k-1} + \max(0, C'_j - d_j) \geq T_k$ (5.1)

$\Rightarrow \Delta C_n \geq 0$, e $\Delta T_n \geq 0$

e portanto os valores para a soma dos atrasos e para o *makespan* da nova solução serão piores ou iguais aos valores atuais. Considera-se que uma solução que tenha valores iguais ao da solução atual nos dois objetivos pode ser descartada, apesar de ser não dominada. Desta forma, aumenta-se a eficiência do algoritmo, uma vez que um vizinho pode ser descartado sem que se calcule o impacto do movimento no atraso de todas as tarefas, uma vez que que os valores já calculados são utilizados para calcular o novo *makespan* e a nova soma dos atrasos.

5.2 Instâncias Teste

Foram geradas instâncias teste de forma semelhante às descritas no capítulo 3, porém utilizou-se apenas dois valores, 0.3 e 0.7, para η, τ e R, sendo gerado apenas uma instância para cada combinação dos parâmetros para um dado tamanho de instância *n*. Os tamanhos de instância utilizados foram 20, 40, 60, 80 e 100, e como existem oito diferentes combinações de η, τ e R, 40 instâncias foram geradas. Esta redução no número de instâncias em relação aos dos testes para heurísticas

construtivas foi feita para que os algoritmos de busca local pudessem ser testados com um tempo computacional maior em cada instância.

5.3 Critério de Parada

O critério de parada escolhido para os testes da busca local com recomeços foi o número de vizinhos avaliados, de forma que os testes podem ser facilmente reproduzidos independente do processador, linguagem ou implementação utilizados. Define-se que um vizinho é avaliado quando o impacto do movimento no valor do atraso e no valor do makespan são estimados. Assim, mesmo um vizinho descartado no teste descrito na seção 5.1 é considerado como avaliado. Entretanto, se um movimento é descartado nas vizinhanças reduzidas ou se um movimento aumenta o *makespan* durante o preprocessamento, considera-se que o vizinho não foi avaliado.

Para determinar o número de vizinhos avaliados permitido foram geradas 4 instâncias com $\eta = \tau = R = 0.6$ para cada tamanho n = 20, 40, 60, 80 e 100. Para cada uma das 20 instâncias foi executada uma busca local durante 2 horas com a seguinte configuração:

- vizinhança 3-opt($\beta = 0\%$)
- estratégia NFirst
- não utilizou-se preprocessamento
- recomeços com perturbação de troca entre 3 tarefas nas soluções em P_{k-1} (ver seção 4.5).

Esta configuração foi escolhida porque em testes preliminares a vizinhança 3-opt havia obtido os melhores resultados, sendo utilizada uma estratégia da literatura e uma forma simples de recomeço.

Mediu-se o desvio proposto por Daniels (1992) e descrito no capítulo 1, conforme os resultados evoluíram com o tempo. Para n = 20, o desvio após 1 segundo foi de apenas 0.07%, chegando a 0% após 7 segundos. Os demais tamanhos de instância estão representados no gráfico da Figura 5.1. Analisando-se os resultados, optou-se por limitar o tempo ao necessário para obter um desvio de 1.5%, de forma a obter um balanceamento entre a qualidade das soluções e o

esforço computacional exigido. Para $n = 20$ o tempo foi definido como 1 segundo, o menor tempo medido. Os tempos determinados estão na Tabela 5.1.

Cada vizinhança necessita de um esforço computacional diferente para avaliar o mesmo número de vizinhos. Assim, é necessário utilizar um limite de vizinhos avaliados diferente para cada vizinhança. Para as mesmas 20 instâncias, executou-se o algoritmo durante 3 segundos modificando-se apenas a vizinhança utilizada. Pode-se assim estimar o número de vizinhos avaliados por segundo em cada vizinhança e determinar o número de vizinhos avaliados no tempo estipulado pela Tabela 5.1 para as seguintes vizinhanças:

- API (adjacent pairwise interchange)
- PI (pairwise interchange)
- INS (inserção)
- PI+INS (pairwise interchange e inserção)
- PI RED (PI reduzida)
- INS RED (INS reduzida)
- PI+INS RED (PI+INS reduzida)
- OR - OPT
- 3-OPT(β) (3-opt com parâmetro β)

A Tabela 5.2 apresenta os resultados obtidos.

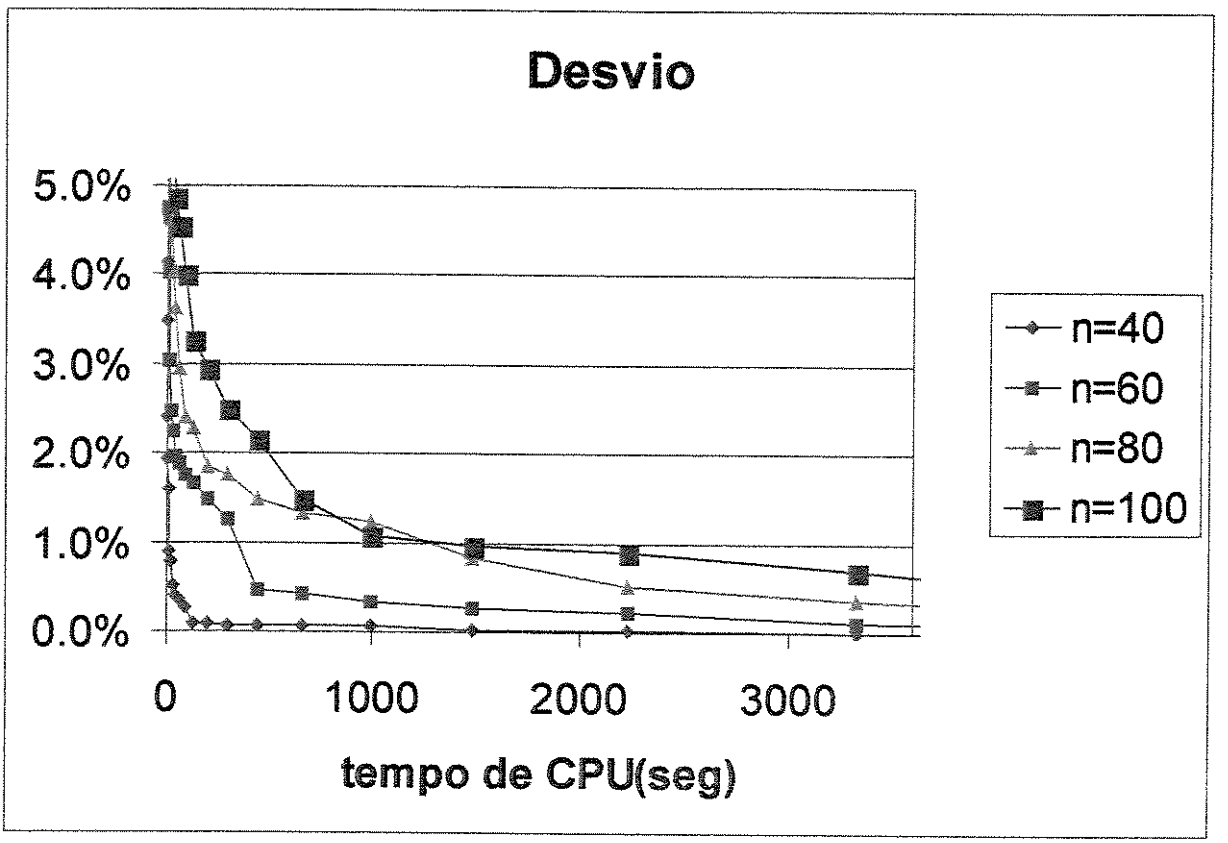

Figura 5.1 – Desvio médio em função do tempo para cada tamanho de instância

Tabela 5.1– Tempo para atingir desvio menor que 1.5% cada tamanho de instância

	n = 20	n = 40	n = 60	n = 80	n = 100
Tempo (s)	1	12	215	460	690

Tabela 5.2 – Limite de vizinhos avaliados para cada vizinhança e cada tamanho de instância (milhões de vizinhos) no tempo estipulado pela Tabela 5.1

	n = 20	n = 40	n = 60	n = 80	n = 100
PI	0.72	10.6	174.06	364.43	499.14
API	0.85	7.00	82.46	139.06	147.57
INS	0.93	7.23	84.84	139.78	147.89
PI+INS	0.93	7.41	85.04	139.85	149.87
PI RED	0.26	2.37	29.90	51.95	59.66
INS RED	0.89	7.05	83.55	134.34	143.07
PI+INS RED	0.79	6.59	78.72	128.97	140.79
OR-OPT	0.75	6.30	76.00	128.97	138.17
3OPT(0%)	0.36	3.44	42.56	77.18	99.26
3OPT(5%)	0.38	3.56	44.32	80.06	102.75
3OPT(7.5%)	0.39	3.67	45.39	81.99	104.73
3OPT(10%)	0.40	3.79	46.45	83.91	106.70
3OPT(12.5%)	0.42	3.83	46.92	84.46	107.46
3OPT(15%)	0.43	3.88	47.40	85.00	108.22
3OPT(20%)	0.44	3.96	48.04	85.91	108.97

Observa-se na Tabela 5.2 que o limite de vizinhos avaliados para 3-opt aumenta com o valor de β, pois quanto menor β, maior é o tempo gasto para encontrar um vizinho que satisfaça as restrições para que ele seja avaliado (ver seção 4.3.8).

5.4 Testes para β em 3-OPT

O primeiro teste realizado foi a determinação do parâmetro β para a vizinhança 3-opt descrita na seção 4.3.8. Testou-se os seguintes valores para este parâmetro: 0%, 5%, 7.5%, 10%, 12.5%, 15% e 20%. Nos testes utilizou-se a seguinte configuração para o algoritmo:

- estratégia de busca NFirst
- não utilizou-se preprocessamento
- recomeços com perturbação de troca entre 3 tarefas nas soluções em P_{k-1} (ver seção 4.5.1).

A Tabela 5.3 apresenta os resultados do desvio médio, distância média, número total de soluções de referência (melhores soluções encontradas para a instância em todos os testes), desvio médio para o atraso ($\alpha = 0$), desvio médio para o *makespan* ($\alpha = 1$). Observa-se que o melhor desvio e a menor distância foram obtidos com β = 10%, apesar da diferença na distância para 7.5% ser de 0.00005. O número de soluções de referência encontradas nos testes é pequeno, dado que o número total de soluções de referência nas instâncias é 1840, obtidas a partir das soluções não dominadas entre todas as soluções geradas em todos os testes realizados na instância. Assim, os testes encontraram apenas aproximadamente 10% das soluções de referência, a maior parte delas para n = 20 ou 40, enquanto que para n = 100 nenhuma delas foi encontrada.

Tabela 5.3 – Valores médios para o desvio, distância, desvio do atraso e desvio para o makespan, e número de soluções de referência encontradas

Desvio		Distância		Sol. De Referência		Atraso		Makespan	
3OPT(10%)	**1.23%**	3OPT(10%)	**0.01429**	3OPT(5%)	**205**	3OPT(15%)	**1.67%**	3OPT(10%)	**0.98%**
3OPT(7.5%)	**1.28%**	3OPT(7.5%)	**0.01436**	3OPT(7.5%)	**205**	3OPT(12.5%)	**1.75%**	3OPT(7.5%)	**0.98%**
3OPT(12.5%)	**1.30%**	3OPT(12.5%)	**0.01457**	3OPT(10%)	**199**	3OPT(20%)	**1.85%**	3OPT(0%)	**1.05%**
3OPT(5%)	**1.34%**	3OPT(5%)	**0.01502**	3OPT(12.5%)	**194**	3OPT(10%)	**1.86%**	3OPT(12.5%)	**1.13%**
3OPT(15%)	**1.41%**	3OPT(15%)	**0.01533**	3OPT(0%)	**183**	3OPT(7.5%)	**1.90%**	3OPT(5%)	**1.14%**
3OPT(0%)	**1.47%**	3OPT(0%)	**0.01666**	3OPT(20%)	**176**	3OPT(5%)	**2.17%**	3OPT(15%)	**1.28%**
3OPT(20%)	**1.55%**	3OPT(20%)	**0.01678**	3OPT(15%)	**175**	3OPT(0%)	**2.36%**	3OPT(20%)	**1.44%**

A Figura 5.2 mostra que β = 10% e 7.5% foram valores que obtiveram bons resultados em toda a fronteira eficiente, enquanto que 0% e 20% apresentam um maior desequilíbrio, dando mais prioridade a um objetivo do que a outro. A partir dos valores para o desvio médio na Tabela 5.3, optou-se por utilizar β = 10% nos testes seguintes.

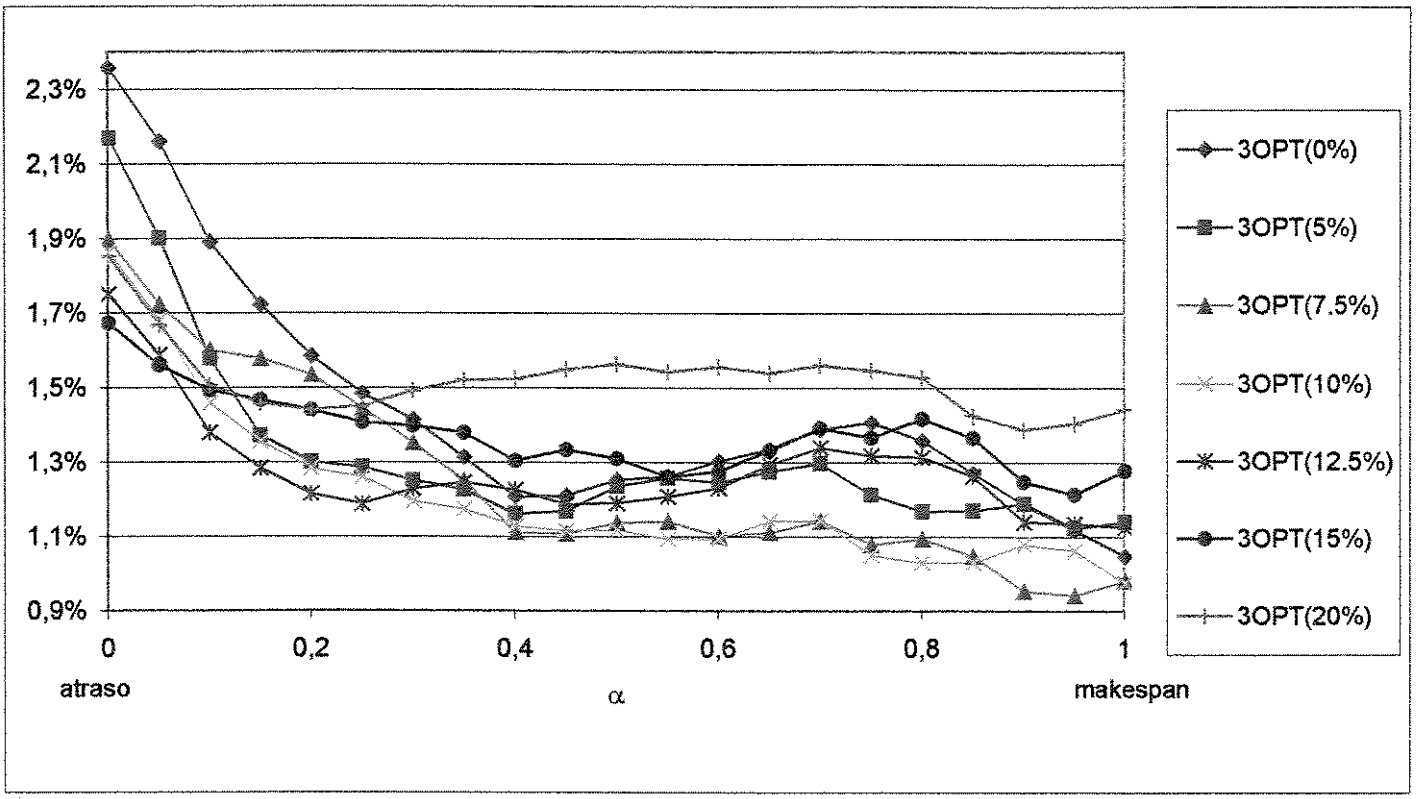

Figura 5.2 – Desvio(α) %

5.5 Testes das vizinhanças

Pretende-se comparar o desempenho das vizinhanças descritas na seção 4.3 para o problema de seqüenciamento bi-objetivo. As siglas utilizadas para identificar a vizinhança são as mesmas utilizadas na determinação do critério de parada na seção 5.3, adicionando-se os resultados obtidos pela heurística construtiva com a sigla CONST, que é a solução inicial. Apenas a vizinhança utilizada diferiu nos testes, sendo que as demais configurações da busca local foram:

- estratégia de busca NFirst
- não utilizou-se preprocessamento
- recomeços com perturbação de troca entre 3 tarefas nas soluções em P_{k-1} (ver seção 4.5).

Tabela 5.4 – Valores médios para o desvio, distância, desvio do atraso e desvio para o makespan, e número de soluções de referência encontradas

Desvio		Distância		Sol. de Referência		Atraso		Makespan	
3OPT(10%)	**1.23%**	3OPT(10%)	**0.0143**	3OPT(10%)	**199**	3OPT(10%)	**1.86%**	3OPT(10%)	**0.98%**
OR OPT	**5.51%**	OR OPT	**0.0553**	OR OPT	**103**	OR OPT	**2.38%**	OR OPT	**10.78%**
TR+INS RED	**11.53%**	TR+INS RED	**0.1256**	TR+INS RED	**51**	TR+INS RED	**4.34%**	TR+INS RED	**18.03%**
TR+INS	**11.72%**	TR+INS	**0.1264**	TR+INS	**50**	TR+INS	**4.48%**	TR+INS	**18.05%**
INS RED	**13.25%**	INS RED	**0.1458**	INS	**49**	INS RED	**4.96%**	INS	**18.43%**
INS	**13.35%**	INS	**0.1473**	INS RED	**44**	INS	**4.98%**	INS RED	**19.21%**
TR	**19.72%**	TR	**0.2270**	TR RED	**3**	TR	**8.28%**	TR RED	**22.39%**
TR RED	**20.08%**	TR RED	**0.2309**	TR	**2**	TR RED	**9.71%**	TR	**22.81%**
API	**25.92%**	API	**0.3264**	API	**0**	API	**12.62%**	API	**26.29%**
CONST	**26.37%**	CONST	**0.3296**	CONST	**0**	CONST	**12.71%**	CONST	**27.10%**

Observa-se na Tabela 5.4 que a vizinhança 3-opt superou claramente as demais em todos os quesitos de avaliação, inclusive para o atraso onde se utilizam com freqüência as vizinhanças de troca e inserção. Or-opt foi a segunda melhor, enquanto que a combinação das vizinhanças de troca e inserção reduzidas ficou em terceiro lugar. É interessante notar que a vizinhança API praticamente não conseguiu melhorar o conjunto de soluções inicial, o que atesta a qualidade das soluções obtidas pelas heurísticas construtivas.

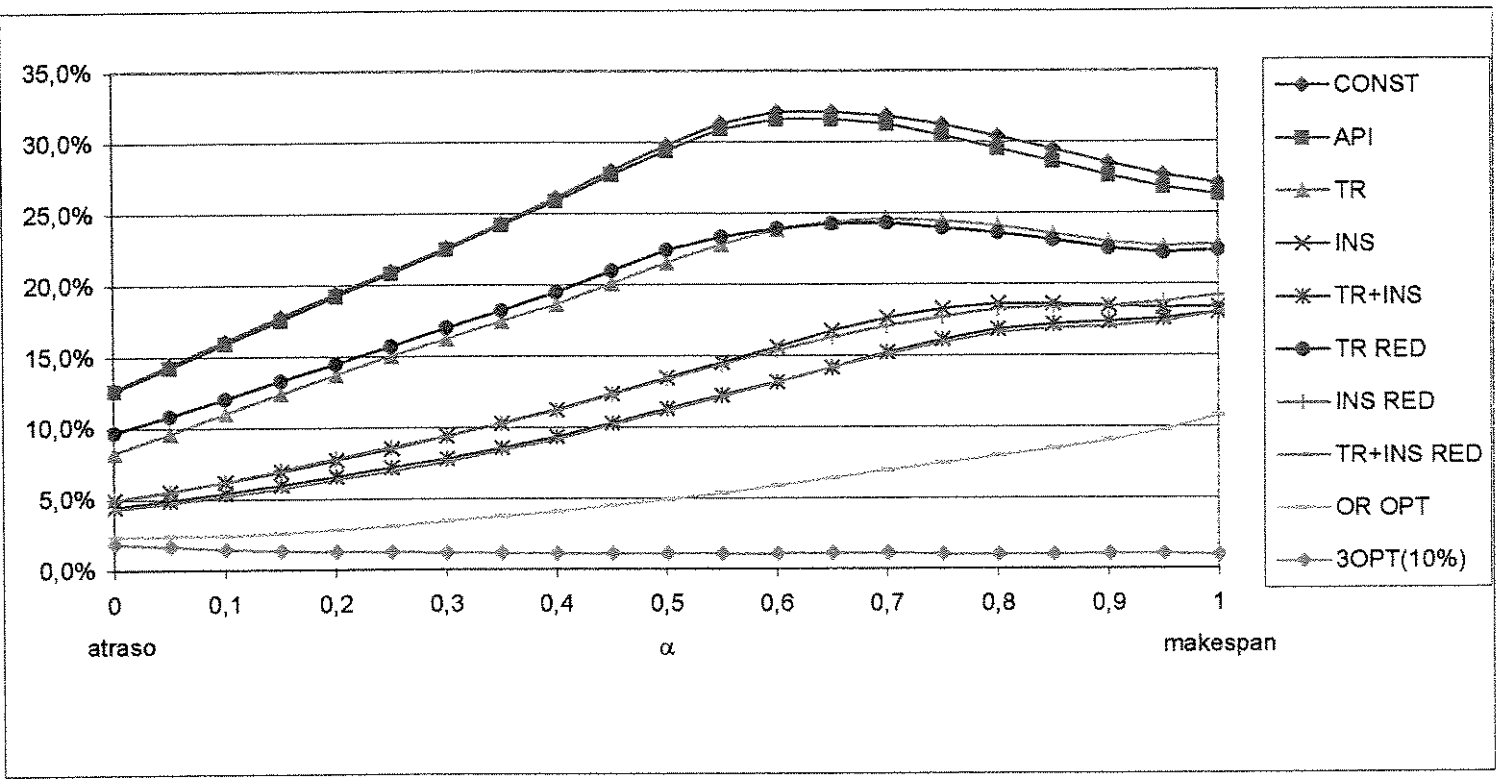

Figura 5.3 – Desvio(α) %

A redução de vizinhança funcionou bem para a vizinhança de troca, aumentando bastante o número de recomeços com relação à versão sem redução. Porém a vizinhança de inserção reduzida aumentou muito pouco o número de iterações, diferentemente dos resultados obtidos por França et al. (2001). Isto se deve principalmente às instâncias testadas, pois aqui η teve valor 0.3 ou 0.7 enquanto que em França et al. (2001) este valor foi de 1.0. O uso de $\eta = 1.0$ diminui drasticamente o número de vizinhos descartados pela redução uma vez que o teste feito depende da proporção entre os tempos de preparação e os tempos de processamento (ver seção 4.3.5, inserção reduzida).

O gráfico na Figura 5.3 mostra claramente a diferença de desempenho entre as vizinhanças utilizadas. Note-se que a diferença de desempenho entre a vizinhança de troca e sua versão reduzida foi muito pequena, apesar do número de recomeços ser muito maior para a segunda. O que se conclui destes testes é que é vantajoso utilizar uma vizinhança mais complexa como Or-opt e 3-opt, mesmo que isto implique em um menor número de recomeços.

5.6 Testes para recomeço

Diferentes maneiras de realizar o recomeço da busca local são descritas na seção 4.5. Estes procedimentos são testados onde:

- ALEAT – gera uma solução aleatória
- TR3 – troca entre 3 tarefas perturbando-se as soluções em P_{k-1}
- TR3* – troca entre 3 tarefas perturbando-se as soluções em E
- TR4 – troca entre 4 tarefas perturbando-se as soluções em P_{k-1}
- TR4* – troca entre 4 tarefas perturbando-se as soluções em E
- DB – Movimento Double Bridge perturbando-se as soluções em P_{k-1}
- DB* – Movimento Double Bridge perturbando-se as soluções em E.

Note que P_{k-1} é o conjunto localmente eficiente encontrado na última iteração, enquanto que E é o conjunto de soluções não-dominadas encontradas pelo algoritmo desde a primeira iteração. A configuração da busca local utilizada foi:

- estratégia de busca NFirst
- não utilizou-se preprocessamento
- vizinhança 3-opt(10%).

A partir da Tabela 5.5, pode-se observar que o reinício aleatório teve o pior desempenho, enquanto que DB* teve a melhor performance. Tem-se ainda que os algoritmos que perturbaram o conjunto E (marcados com *) saíram-se melhor que os que perturbam P_{k-1} , concluindo-se que é melhor realizar a perturbação no conjunto de melhores soluções conhecidas.

Tabela 5.5 – Valores médios para o desvio, distância, desvio do atraso e desvio para o makespan, e número de soluções de referência encontradas

Desvio		Distância		Sol. de Referência		Atraso		Makespan	
DB*	**0.712%**	DB*	**0.00879**	DB*	**293**	TR3*	**1.12%**	DB*	**0.53%**
TR3*	**0.880%**	TR3*	**0.01044**	TR4*	**255**	TR4*	**1.12%**	TR4*	**0.81%**
TR4*	**0.880%**	TR4*	**0.01051**	TR3*	**238**	DB*	**1.14%**	DB	**0.82%**
DB	**1.223%**	DB	**0.01407**	DB	**213**	TR4	**1.77%**	TR3*	**0.92%**
TR3	**1.226%**	TR3	**0.01429**	TR4	**200**	DB	**1.77%**	TR3	**0.98%**
TR4	**1.343%**	TR4	**0.01487**	TR3	**199**	TR3	**1.86%**	TR4	**1.12%**
ALEAT	**2.507%**	ALEAT	**0.02680**	ALEAT	**125**	ALEAT	**2.59%**	ALEAT	**1.79%**

O desempenho de DB* foi superior aos demais para as instâncias maiores, n = 100 e n = 80, onde obteve o melhor desvio em 12 das 16 instâncias. DB* alcançou também o maior número de soluções de referência para todos os tamanhos de instância. Mesmo assim, este número é pequeno se comparado ao total de soluções existentes.

O número de recomeços efetuados com ALEAT é muito menor, dado que a busca local parte sempre de soluções distantes do conjunto eficiente. Já as perturbações geram soluções com valores ruins da função objetivo, mas que com poucas alterações se tornam de boa qualidade. Observa-se ainda que perturbar E aumenta o número de recomeços se comparado à mesma perturbação em P_{k-1}, uma vez que as perturbações são feitas em soluções de melhor qualidade, formando soluções mais próximas da fronteira eficiente em geral.

Um fato importante, que pode ser notado na Figura 5.4, é que os algoritmos que perturbam o conjunto E de melhores soluções tiveram desempenho semelhante para o atraso, não sendo o mesmo observado para o *makespan*. O mesmo ocorreu para os algoritmos que perturbam P_{k-1}, formando dois grupos distintos. O recomeço DB* é claramente superior quando $\alpha > 0.2$, e só não se destaca quando o atraso é o objetivo mais importante.

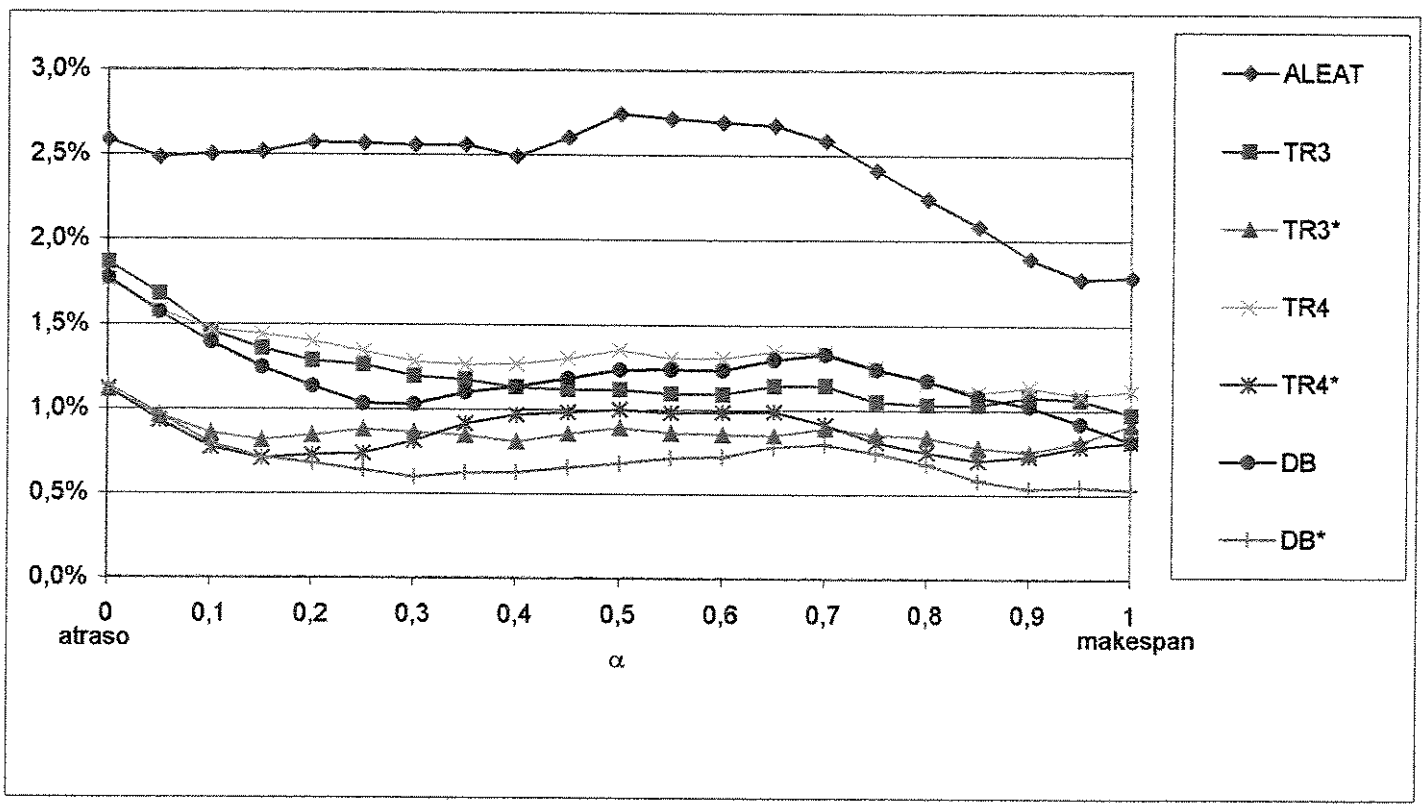

Figura 5.4 – Desvio(α) %

5.7 Testes para r-grupos

Na seção 4.1.2 é descrita a estratégia r-grupos, onde as soluções cuja vizinhança ainda não foi explorada são divididas em r-grupos e explora-se paralelamente uma solução em cada grupo. Pretende-se determinar o valor de r que será utilizado nos próximos testes, sendo testados os valores 1, 3, 5, 7, e 10, onde o valor 1 corresponde a simplesmente tomar uma solução aleatória entre as não visitadas. A configuração da busca local utilizada foi:

- estratégia r-grupos
- não utilizou-se preprocessamento
- vizinhança 3-opt(10%)
- recomeços com perturbação Double Bridge no conjunto E.

Tabela 5.6 – Valores médios para o desvio, distância, desvio do atraso e desvio para o makespan, e número de soluções de referência encontradas

Desvio		Distância		Sol. de Referência		Atraso		Makespan	
r = 5	**0.64%**	r = 3	**0.00862**	r = 1	**291**	r = 3	**1.12%**	r = 10	**0.42%**
r = 3	**0.67%**	r = 1	**0.00864**	r = 5	**289**	r = 7	**1.16%**	r = 5	**0.43%**
r = 1	**0.68%**	r = 5	**0.00866**	r = 3	**265**	r = 5	**1.22%**	r = 7	**0.48%**
r = 7	**0.71%**	r = 10	**0.00917**	r = 7	**256**	r = 1	**1.26%**	r = 3	**0.49%**
r = 10	**0.72%**	r = 7	**0.00923**	r = 10	**243**	r = 10	**1.39%**	r = 1	**0.61%**

O valor médio do desvio indica que 5 seria o melhor valor para *r*, enquanto que 3 foi o melhor para a distância média. O que se conclui é que a diferença no desempenho foi muito pequena entre estes valores de *r* para que se eleja um vencedor, sendo esta proximidade no desempenho também observada na Figura 5.5. Como utiliza-se como medida de referência o desvio, opta-se por $r = 5$ nos próximos testes realizados.

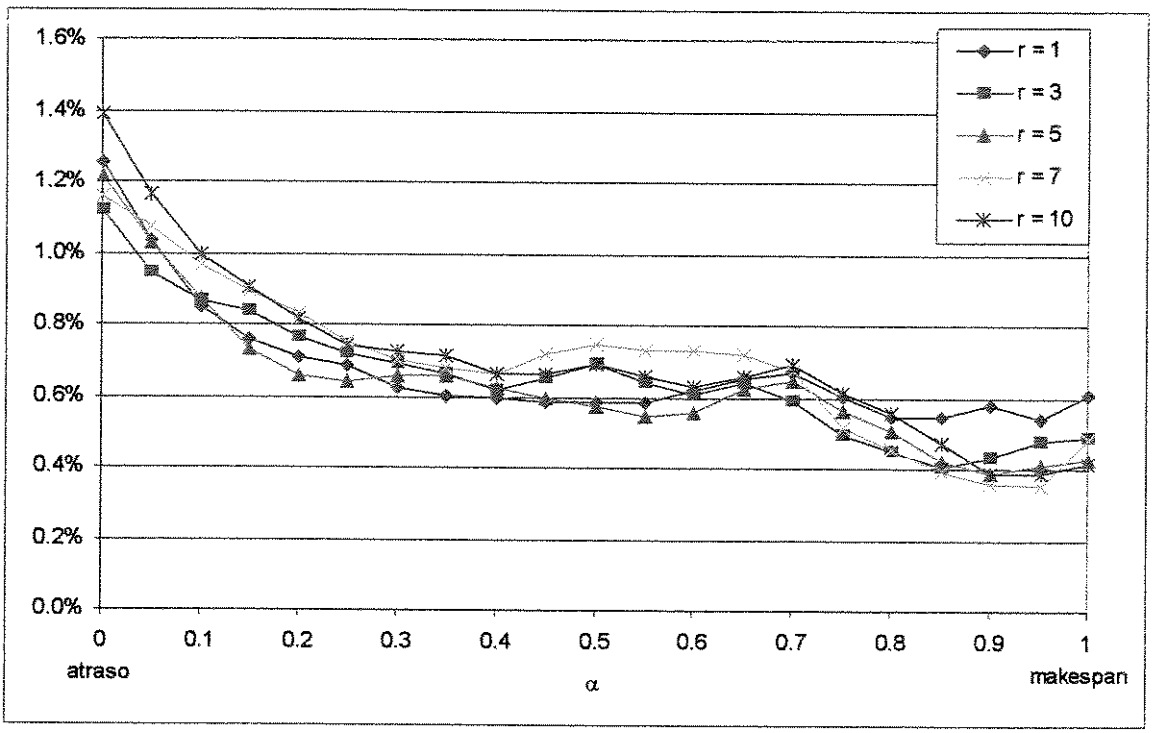

Figura 5.5 - Desvio(α) %

5.8 Testes para estratégia de busca e preprocessamento

A estratégia de busca determina quais vizinhanças serão exploradas e quando atualizar o conjunto solução, e as propostas testadas estão descritas na seção 4.1. Cada estratégia de busca foi testada com e sem preprocessamento, sendo a versão com preprocessamento identificada pelo sufixo –P na sigla. A configuração da busca local utilizada foi:

- vizinhança 3-opt(10%).
- recomeços com perturbação Double Bridge no conjunto E.

A melhor estratégia para o desvio foi a de 5-grupos com preprocessamento, enquanto que para a distância e para o número de soluções de referência a estratégia mista também com preprocessamento foi a melhor, como apresentado na Tabela 5.7. A diferença nos resultados dos melhores algoritmos foi pequena, e por isto as medidas de distância e desvio divergiram. Aliás, cada um dos quesitos apresentados na Tabela 5.7 teve uma estratégia diferente alcançando os melhores resultados, o que mostra o equilíbrio entre as estratégias. Nota-se ainda que todas as estratégias

tiveram uma melhoria no desempenho para o desvio e para a distância com o preprocessamento.

Tabela 5.7 – Valores médios para o desvio, distância, desvio do atraso e desvio para o makespan, e número de soluções de referência encontradas

Desvio		Distância		Sol. de Referência		Atraso		Makespan	
5-GRUPOS-P	**0.626%**	NFIRST-P	**0.00809**	MISTA-P	**315**	TFIRST-P	**0.91%**	MFIRST	**0.41%**
5-GRUPOS	**0.643%**	MISTA-P	**0.00810**	NFIRST	**293**	NFIRST-P	**1.00%**	5-GRUPOS	**0.43%**
NFIRST-P	**0.652%**	5-GRUPOS-P	**0.00813**	5-GRUPOS-P	**292**	5-GRUPOS-P	**1.02%**	MFIRST-P	**0.44%**
MISTA-P	**0.662%**	5-GRUPOS	**0.00866**	MFIRST-P	**291**	TFIRST	**1.03%**	5-GRUPOS-P	**0.52%**
MFIRST-P	**0.666%**	TFIRST-P	**0.00878**	5-GRUPOS	**289**	NFIRST	**1.14%**	NFIRST-P	**0.53%**
TFIRST-P	**0.684%**	NFIRST	**0.00879**	NFIRST-P	**284**	MISTA-P	**1.17%**	NFIRST	**0.53%**
NFIRST	**0.712%**	MFIRST-P	**0.00896**	TFIRST-P	**251**	5-GRUPOS	**1.22%**	MISTA-P	**0.55%**
MISTA	**0.842%**	MISTA	**0.01062**	MFIRST	**249**	MISTA	**1.27%**	MISTA	**0.57%**
MFIRST	**0.907%**	MFIRST	**0.01163**	MISTA	**228**	MFIRST-P	**1.38%**	TFIRST-P	**0.91%**
TFIRST	**1.215%**	TFIRST	**0.01319**	TFIRST	**192**	MFIRST	**1.86%**	TFIRST	**1.60%**

Realizando a busca primeiro nas soluções com menor atraso prioriza este objetivo, como pode ser observado pelo desempenho de TFirst para este objetivo. O mesmo ocorre para o *makespan* com MFirst, sendo que em ambos os casos a busca não apresenta desempenho tão bom para o outro objetivo. A estratégia Mista, que tenta juntar as qualidades de TFirst e MFirst teve um desempenho apenas regular, apesar de alcançar o maior número de soluções de referência.

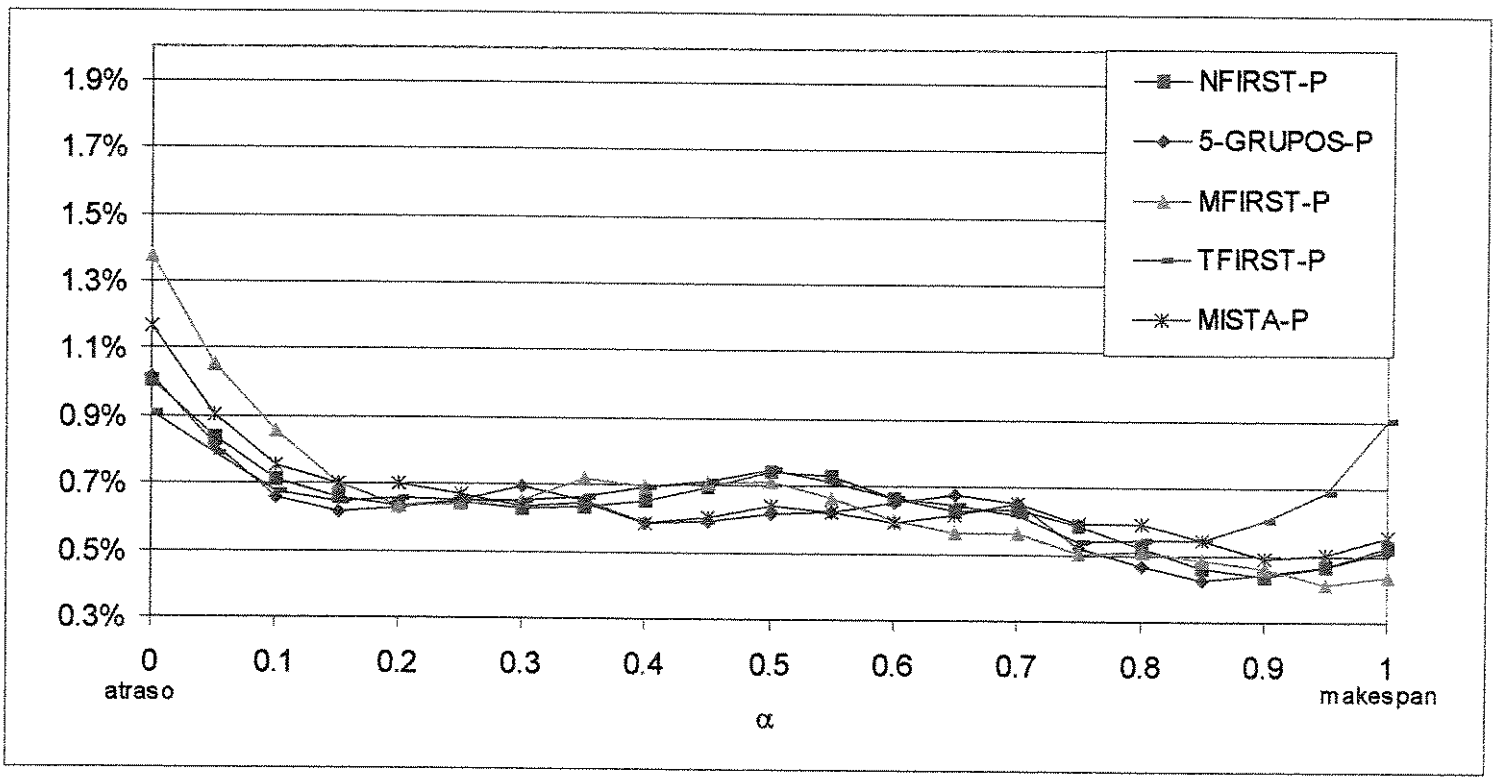

Figura 5.6 - Desvio(α) %

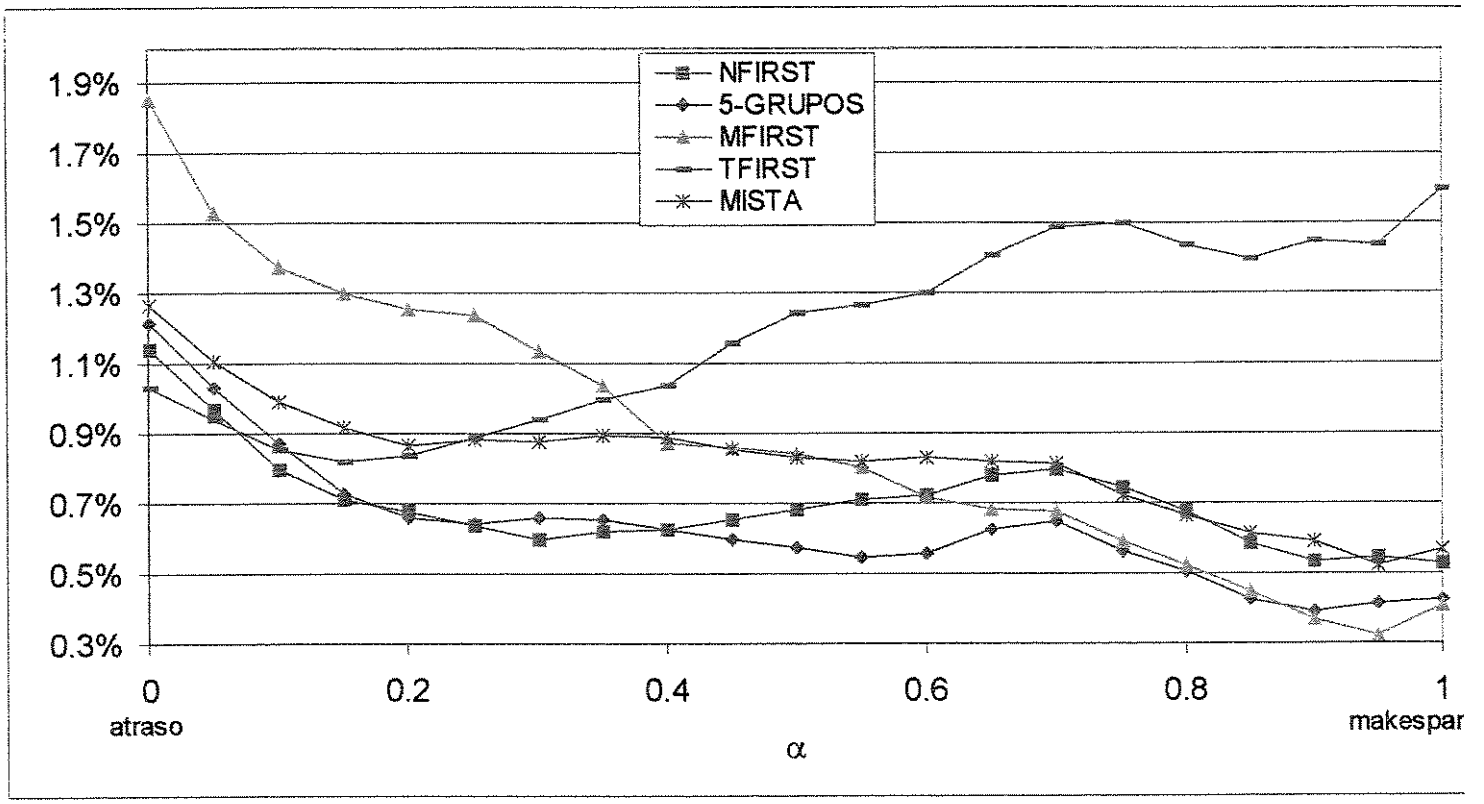

Figura 5.7 - Desvio(α) %

O uso do preprocessamento aumentou o número de recomeços realizados no tempo disponível, exceto para a estratégia 5 –grupos onde, para $n \geq 60$ o número de recomeços diminui com o uso do preprocessamento.

O gráfico da Figura 5.6 apresenta o desvio(α) para as versões com preprocessamento, enquanto que na Figura 5.7 estão os resultados sem o preprocessamento. As buscas MFirst e TFirst claramente tem desempenho melhor para um dos objetivos quando não é utilizado o preprocessamento, sendo que a utilização deste diminui esta diferença.

A utilização do preprocessamento reduz a influência da estratégia nos resultados, dada a proximidade das curvas no gráfico 5.6 comparada aos resultados sem preprocessamento, uma vez que boa parte do processo de busca é realizado durante o preprocessamento. Este resultado é importante pois utilizar o preprocessamento tornou praticamente indiferente a escolha da estratégia, sendo uma escolha a menos a ser feita na formação do algoritmo.

5.9 Testes para o conjunto de soluções inicial

O conjunto de soluções geradas pela heurística construtiva deve influir na qualidade das soluções encontradas nas primeiras iterações. Os testes anteriormente realizados foram feitos iniciando-se a busca a partir do conjunto de soluções geradas pela combinação das heurísticas construtivas VMPR e ATCS(12). É questionável a necessidade de se implementar estas duas heurísticas no caso de se aplicar a busca local com recomeços, especialmente se o número de iterações realizado for alto.

Os testes apresentados nesta seção visam analisar a utilização do conjunto de soluções gerado pelas heurísticas multiobjetivo oposto à utilização de uma heurística que gere apenas uma solução, no caso utilizou-se a heurística ATCS, ou a simples geração de uma solução aleatória. As siglas utilizadas foram:

- COMB – Combinação das heurísticas VMPR e ATCS(12)
- ALEAT – solução inicial gerada aleatoriamente.
- ATCS – Execução da heurística ATCS uma única vez com os parâmetros sugeridos por Lee et al. (1997).

O conjunto inicial gerado em ATCS e em ALEAT é composto por apenas uma solução. A configuração da busca local foi:

- estratégia de busca 5-grupos
- utilizou-se preprocessamento
- vizinhança 3-opt(10%)
- recomeços com perturbação Double Bridge no conjunto E.

Tabela 5.8 – Valores médios para o desvio, distância, desvio do atraso e desvio para o makespan, e número de soluções de referência encontradas

Desvio		Distância		Sol. de Referência		Atraso		Makespan	
COMB	**0.63%**	COMB	**0.00813**	ATCS	**299**	COMB	**1.02%**	ATCS	**0.50%**
ATCS	**0.65%**	ATCS	**0.00844**	COMB	**292**	ATCS	**1.18%**	COMB	**0.52%**
ALEAT	**0.73%**	ALEAT	**0.00901**	ALEAT	**291**	ALEAT	**1.30%**	ALEAT	**0.60%**

Analisando-se a Tabela 5.8 verifica-se que a utilização de uma solução inicial aleatória é a pior das opções testadas, apesar da diferença ser pequena. Já a

utilização da regra ATCS mostrou-se mais competitiva, sendo superada por uma pequena margem nos valores do desvio e da distância médios.

Os resultados obtidos em cada instância mostram que houve um equilíbrio nos testes, onde mesmo ALEAT foi o melhor algoritmo para várias instâncias. Isto indica que o conjunto de soluções inicial tem um impacto reduzido, podendo ser superado por características da instância ou variações aleatórias. Acredita-se que a utilização do preprocessamento com a vizinhança 3-opt é a principal responsável pela reduzida importância do conjunto inicial.

O número médio de recomeços foi: 426 para COMB; 442 para ALEAT; 425 para ATCS. Isto surpreende, pois espera-se que a utilização de um melhor conjunto inicial como em COMB aumente o número de recomeços, pois seria poupado o esforço da busca local para chegar a soluções da mesma qualidade. Entretanto, ALEAT apresentou o maior número de recomeços, enquanto que ATCS só obteve um número menor de recomeços que COMB para $n \geq 60$.

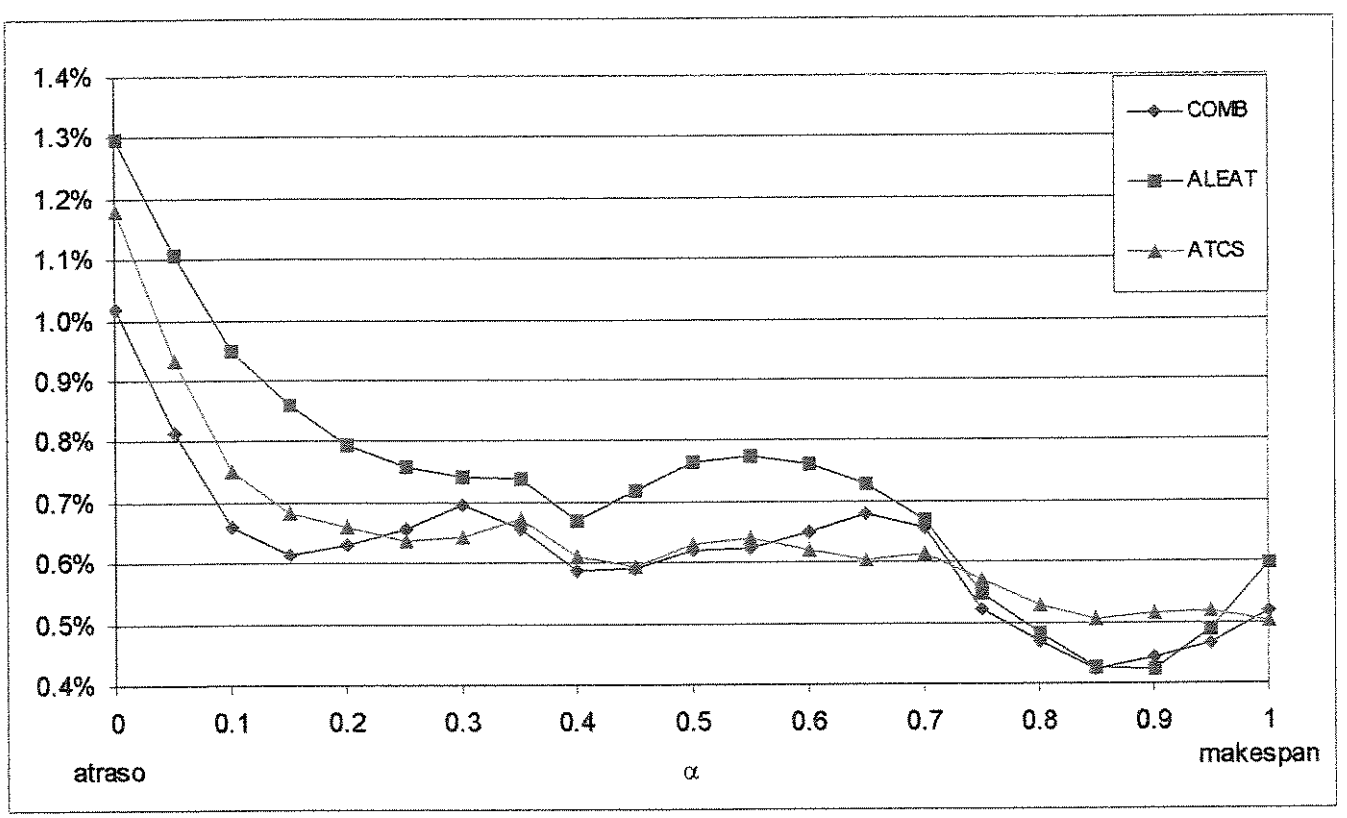

Figura 5.8 - Desvio(α) %

Na Figura 5.8 mostra-se que o impacto do conjunto inicial é mais nítido quando o atraso é o objetivo mais importante, enquanto que no extremo oposto as escolhas se mostraram competitivas entre si. Conclui-se ainda que ALEAT foi pior na aproximação da maior parte da fronteira eficiente, enquanto que ATCS foi superada claramente na região onde o atraso é o objetivo mais importante.

5.10 Testes para estratégia r – grupos com vizinhança 3opt variável

A estratégia de busca apresentada na seção 4.4 utiliza uma folga para a vizinhança 3opt variável, dependendo do grupo onde a vizinhança é obtida. Esta proposta visa utilizar uma folga menor nas soluções de melhor *makespan* e maior nas soluções de melhor atraso total.

Para testar esta estratégia utilizou-se o número de grupos de melhor desvio médio nos testes da seção 5.7, ou seja, r = 5. Variou-se o valor de β_{max} , o valor máximo do parâmetro de folga β numa vizinhança. O critério de parada número de vizinhos avaliados depende da vizinhança utilizada, e nestes testes utilizou-se a média entre o limite para 3opt (0%) e 3opt(β_{max}), conforme valores apresentados na Tabela 5.1. Os algoritmos são identificados por VAR(β_{max}) e a configuração restante da busca local foi:

- utilizou-se preprocessamento
- recomeços com perturbação Double Bridge no conjunto E
- solução inicial COMB (seção 5.9).

Tabela 5.9 – Valores médios para o desvio, distância, desvio do atraso e desvio para o makespan, e número de soluções de referência encontradas

Desvio		Distância		Sol. de Referência		Atraso		Makespan	
VAR(5%)	**0.457%**	VAR(5%)	**0.0063**	VAR(5%)	**380**	VAR(10%)	**0.80%**	**VAR(5%)**	0.41%
VAR(7.5%)	**0.548%**	VAR(7.5%)	**0.0069**	VAR(7.5%)	**328**	VAR(12.5%)	**0.82%**	**VAR(7.5%)**	0.62%
VAR(10%)	**0.607%**	VAR(10%)	**0.0074**	VAR(10%)	**305**	VAR(15%)	**0.87%**	**VAR(12.5%)**	0.79%
VAR(12.5%)	**0.611%**	VAR(12.5%)	**0.0077**	VAR(12.5%)	**293**	VAR(7.5%)	**0.88%**	**VAR(10%)**	0.83%
VAR(15%)	**0.671%**	VAR(15%)	**0.0083**	VAR(15%)	**276**	VAR(5%)	**1.00%**	**VAR(15%)**	0.88%
VAR(20%)	**0.921%**	VAR(20%)	**0.0104**	VAR(20%)	**255**	VAR(20%)	**1.03%**	**VAR(20%)**	1.06%

Os valores apresentados na Tabela 5.9 mostram que para o desvio, distância e número de soluções de referência quanto menor β_{max} melhor o desempenho. Aproximadamente o mesmo ocorre para o *makespan*, com a exceção de 10% ser superado por 12.5%. Já para o atraso os melhores resultados foram obtidos com valores de β_{max} entre 7.5% e 15%.

Valores de β_{max} menores que 5% seriam próximos a executar a busca sem folga, o que implicaria que a vizinhança variável não teria efeito algum. Questiona-se se os resultados obtidos seriam melhores que os de uma combinação entre 5-grupos e 3-opt com pequenos valores para a folga, já que esta estratégia de busca só foi testada com 3-opt(10%). Assim, os próximos testes realizados serão desta combinação.

O tempo utilizado por cada algoritmo diminuiu com o aumento do valor de β_{max} . Isto decorre do critério de parada utilizado, número de vizinhos avaliados, pois para valores de folga altos a Tabela 5.2 estabelece um número maior de vizinhanças a visitar, isto é, quanto maior β_{max} menor o tempo médio para avaliar um vizinho. Como utilizou-se a média entre o número de vizinhos para 0% e para β_{max} como critério de parada, ocorreu uma distorção pois grande parte das vizinhanças foi explorada com o parâmetro β próximo a β_{max}.

É provável que a diferença no desempenho da tabela 5.9 esteja na combinação da estratégia r-grupos com baixos valores de folga na vizinhança 3opt, como testado na seção 5.11. Temos ainda como fator importante o aumento no número de recomeços quando β_{max} = 5% é utilizado, que favorece o desempenho do algoritmo.

A Figura 5.9 mostra que o valor 5% é superior na maior parte da fronteira eficiente, sendo um caso especial quando o atraso é mais importante e valores próximos a 10% encontram melhores resultados. Já para o *makespan* 5% se destaca das demais, o que é esperado dado o maior número de recomeços realizados e a utilização da folga não visar a minimização deste objetivo.

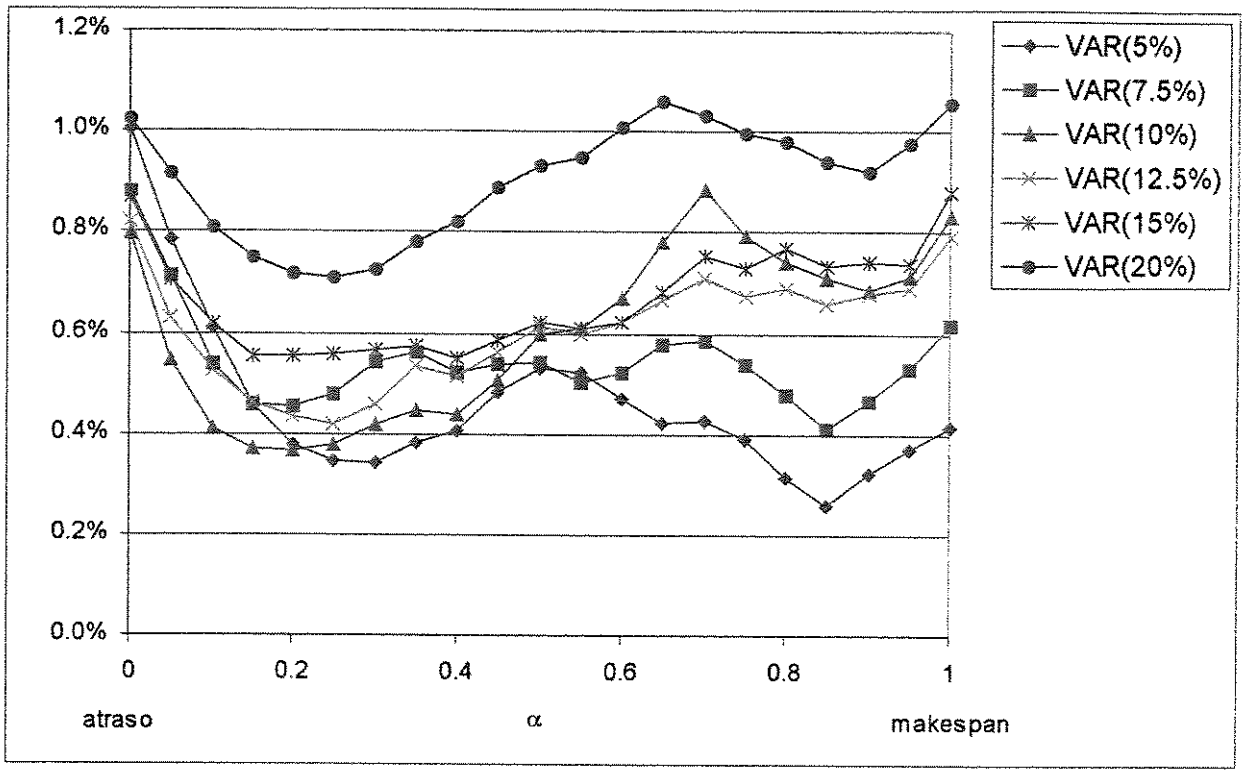

Figura 5.9 - Desvio(α) %

5.11 Testes para β em 3opt com estratégia r-grupos

Na seção 5.4 foram testados diferentes valores para o parâmetro de folga β na vizinhança 3-opt. Estes testes foram realizados com a estratégia de busca NFirst, sem preprocessamento e recomeços com perturbação de troca entre três tarefas, e determinou-se o valor de 10% para β utilizado nos testes subseqüentes.

Os resultados da seção 5.10 indicam que uma combinação de valores de β pequenos com a vizinhança r-grupos deve gerar bons resultados. Pretende-se aqui testar esta combinação, utilizando r = 5, e valores de β menores ou iguais a 10%. A configuração da busca foi:

- utilizou-se preprocessamento
- estratégia 5-grupos
- recomeços com perturbação Double Bridge no conjunto E.

Tabela 5.10 – Valores médios para o desvio, distância, desvio do atraso e desvio para o makespan, e número de soluções de referência encontradas

Desvio		Distância		Sol. de Referência		Atraso		Makespan	
β = 0%	**0.384%**	β = 0%	**0.0055**	β = 0%	**434**	β = 0%	**0.87%**	β = 5%	**0.31%**
β = 5%	**0.507%**	β = 5%	**0.0068**	β = 5%	**365**	β = 10%	**1.02%**	β = 0%	**0.36%**
β = 7.5%	**0.556%**	β = 7.5%	**0.0072**	β = 7.5%	**337**	β = 7.5%	**1.02%**	β = 7.5%	**0.51%**
β = 10%	**0.626%**	β = 10%	**0.0081**	β = 10%	**292**	β = 5%	**1.08%**	β = 10%	**0.52%**

Os valores de desvio, distância e número de soluções de referência encontradas (Tabela 5.10) mostra que para a configuração de busca utilizada não é vantagem utilizar folga na vizinhança 3-opt, contrariando os resultados apresentados na seção 5.4. A principal diferença nestes testes é a estratégia de busca utilizada, e portanto conclui-se que a utilização ou não de folga depende da estratégia utilizada.

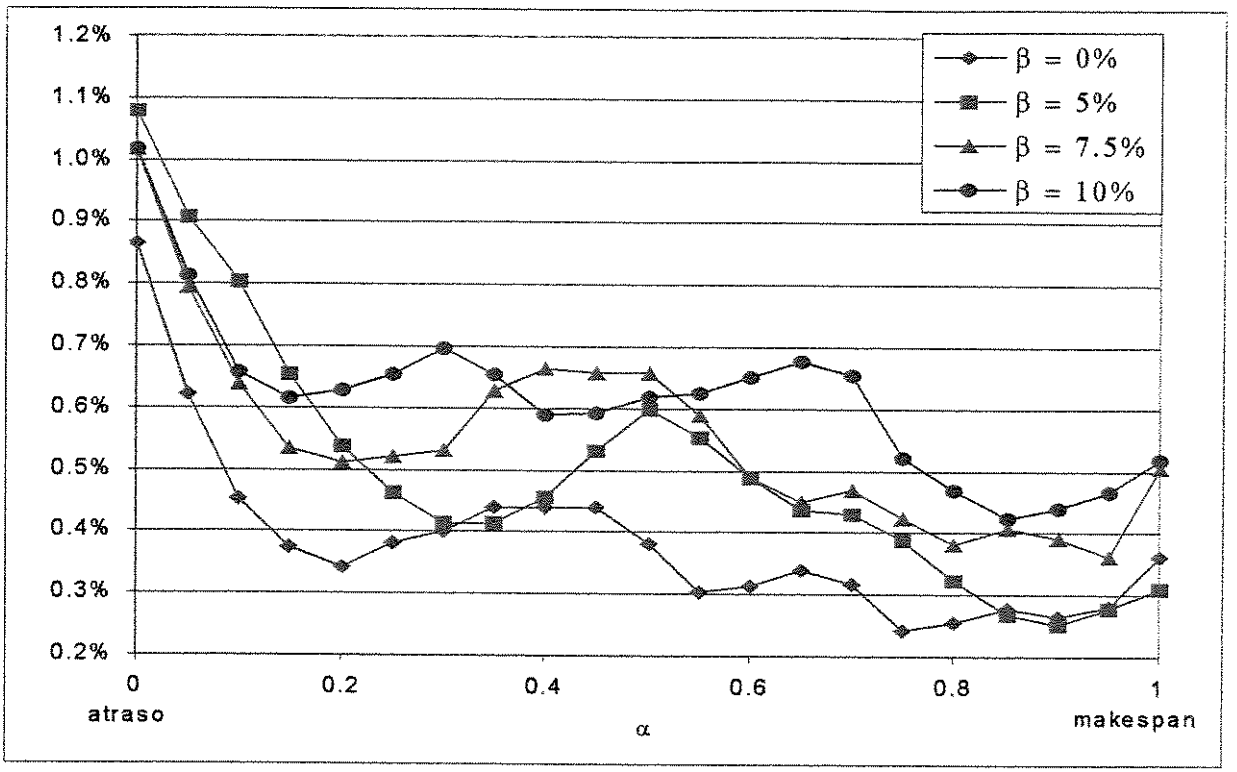

Figura 5.10 - Desvio(α) %

O gráfico da Figura 5.10 mostra que β = 0% é a melhor opção na maior parte da fronteira eficiente. Para α próximo de 1, β = 5% também foi uma boa opção. É interessante notar que 5% obteve os melhores resultados para o *makespan* e os piores para o atraso.

5.12 Testes com enumeração completa

Uma análise que pode ser feita é a da qualidade da solução para instâncias de pequeno porte, que podem ser resolvidas por métodos exatos. Testou-se o algoritmo VAR(10%) apresentado na seção 5.10, aplicado às instâncias resolvidas por enumeração completa utilizadas nos testes para heurísticas construtivas. Quando este teste foi realizado, o algoritmo VAR(10%) era o de melhores resultados, e como este encontrou 99,9% das soluções eficientes em tempo muito curto, julgou-se desnecessário refazer os testes com β = 0%.

O critério de parada utilizado foi o número de vizinhos da Tabela 5.1 para a vizinhança 3-opt(10%) e n = 20. A Tabela 5.11 apresenta a média dos resultados alcançados, onde o tempo é dado em segundos, iter é o número de iterações (recomeços) realizadas, melhor_iter indica a partir de qual iteração não houve melhora nos resultados, dist é a distância média, desv o desvio médio e efi o número total de soluções de referência encontradas.

É notável que o número de soluções de referência nas instâncias, determinado por enumeração completa, seja 2833 e portanto apenas três soluções de referência não foram encontradas pelo algoritmo, sendo as três em instâncias diferentes. Fica comprovada assim a qualidade da implementação para estas instâncias (n =10, 12 e 14), sendo que o tempo é muito menor que o gasto para enumeração completa que é de aproximadamente 4 horas para n = 14. Além disto, apenas 58 iterações em média foram necessárias para resolver as instâncias, o que implica em um tempo médio de aproximadamente 0.03 segundos.

Tabela 5.11 – resultados para instâncias de pequeno porte

tempo	iter	melhor_iter	dist	desv	efi
1.3	2563	58	1.94E-05	2.61E-04%	2830

5.13 Qualidade em função do tempo computacional

É importante analisar a evolução da busca local de forma dinâmica, observando a qualidade das soluções conforme aumenta o tempo computacional utilizado. Esta análise dinâmica possibilita observar como a busca evolui, se ocorre estagnação em um conjunto localmente eficiente ou com que velocidade a qualidade das soluções aumenta. Outro ponto importante é observar a qualidade das soluções após um tempo computacional relativamente elevado, obtendo soluções de alta qualidade para cada instância.

Para realizar estas análises utilizou-se a configuração do algoritmo VAR(10%) apresentado na seção 5.10 que era o algoritmo com melhores resultados, enquanto que os resultados de melhor desempenho apresentados tanto na seção 5.10 como na seção 5.11 só foram obtidos após a realização do experimento aqui descrito.

Utilizou-se como critério de parada um número de vizinhos avaliados 15 vezes superior ao apresentado na Tabela 5.1 para a vizinhança 3opt(10%), sendo apresentados resultados em diferentes instantes de tempo onde, por exemplo, **3x** significa um número de vizinhos avaliado três vezes maior que o da Tabela 5.1.

Tabela 5.12 – Valores médios para o desvio, distância, desvio para o atraso e desvio para o *makespan*, e número de soluções de referência encontradas em função do tempo computacional.

	Desvio	Distância	Sol. de Referência	Atraso	Makespan
1x	0.608%	0.0079	278	0.85%	0.70%
3x	0.306%	0.0043	515	0.49%	0.27%
7x	0.174%	0.0027	741	0.30%	0.16%
15x	0.094%	0.0017	1067	0.25%	0.10%

A Tabela 5.12 mostra os resultados médios para o desvio, distância, desvio para o atraso e para o *makespan* além do número de soluções de referência encontradas conforme o tempo evolui. Observe-se que apesar dos valores para o desvio e a distância serem pequenos para **1x**, o número de soluções de referência cresce consideravelmente conforme o tempo evolui, ilustrando a dificuldade de obter soluções eficientes.

Os resultados para **15x** em cada instância são apresentados na Tabela 5.13 onde são apresentados o desvio, a distância e o número de soluções de referência encontradas em cada instância. É apresentado também o número de soluções de referência existente em cada instância, lembrando-se que se consideram como soluções de referência as melhores soluções conhecidas para cada instância.

Tabela 5.13 – Resultados para tempo **15x** em cada instância

	n	τ	η	R	Desvio	Distância	Referências Encontradas	Referências Existentes
1	20	0.3	0.3	0.3	0.00%	0.0000	11	11
2	20	0.3	0.3	0.7	0.00%	0.0000	15	15
3	20	0.3	0.7	0.3	0.00%	0.0000	12	12
4	20	0.3	0.7	0.7	0.11%	0.0024	17	20
5	20	0.7	0.3	0.3	0.00%	0.0000	25	25
6	20	0.7	0.3	0.7	0.00%	0.0000	29	29
7	20	0.7	0.7	0.3	0.00%	0.0000	18	18
8	20	0.7	0.7	0.7	0.00%	0.0000	11	11
9	40	0.3	0.3	0.3	0.05%	0.0034	12	18
10	40	0.3	0.3	0.7	0.03%	0.0014	21	37
11	40	0.3	0.7	0.3	0.00%	0.0013	16	18
12	40	0.3	0.7	0.7	0.02%	0.0016	32	43
13	40	0.7	0.3	0.3	0.01%	0.0007	32	38
14	40	0.7	0.3	0.7	0.21%	0.0038	21	40
15	40	0.7	0.7	0.3	0.01%	0.0015	32	39
16	40	0.7	0.7	0.7	0.05%	0.0007	34	41
17	60	0.3	0.3	0.3	0.05%	0.0011	28	41
18	60	0.3	0.3	0.7	0.10%	0.0014	37	76
19	60	0.3	0.7	0.3	0.10%	0.0025	18	42
20	60	0.3	0.7	0.7	0.01%	0.0012	31	51
21	60	0.7	0.3	0.3	0.14%	0.0031	19	48
22	60	0.7	0.3	0.7	0.05%	0.0024	30	53
23	60	0.7	0.7	0.3	0.17%	0.0021	34	55
24	60	0.7	0.7	0.7	0.22%	0.0019	40	72
25	80	0.3	0.3	0.3	0.30%	0.0036	5	37
26	80	0.3	0.3	0.7	0.34%	0.0044	10	48
27	80	0.3	0.7	0.3	0.24%	0.0026	22	53
28	80	0.3	0.7	0.7	0.18%	0.0028	18	74
29	80	0.7	0.3	0.3	0.13%	0.0035	15	52
30	80	0.7	0.3	0.7	0.39%	0.0035	22	66
31	80	0.7	0.7	0.3	0.06%	0.0016	38	72
32	80	0.7	0.7	0.7	0.14%	0.0020	45	79
33	100	0.3	0.3	0.3	0.06%	0.0018	21	32
34	100	0.3	0.3	0.7	0.09%	0.0016	18	44
35	100	0.3	0.7	0.3	0.03%	0.0017	33	52
36	100	0.3	0.7	0.7	0.04%	0.0011	41	70
37	100	0.7	0.3	0.3	0.11%	0.0025	28	64
38	100	0.7	0.3	0.7	0.09%	0.0012	67	91
39	100	0.7	0.7	0.3	0.23%	0.0024	39	69
40	100	0.7	0.7	0.7	0.04%	0.0007	70	87

A Tabela 5.14 mostra os resultados médios para cada tamanho de instância para tempo **15x**. É notável a dificuldade que o algoritmo encontrou para n = 80, observado tanto nas medidas de desvio e de distância como no número de soluções de referência.

Outro ponto importante da Tabela 5.14 é destacado na coluna "Melhor Iteração", que mostra qual foi a última iteração que modificou o conjunto solução do algoritmo. Verifica-se que o tempo de 17 segundos é grande para n = 20, pois a busca, em média, não evoluiu após um quarto deste tempo. Já para n = 100 mesmo após 11108 segundos (3 horas e 5 minutos) a busca continua encontrando novas soluções não dominadas, o que ilustra bem a dificuldade em encontrar soluções de referência para instâncias maiores, apesar das medidas utilizadas indicarem que as soluções estão bastante próximas destas (ver Figura 5.15).

Tabela 5.14 – Resultados para tempo **15x** para cada tamanho de instância

n	Tempo (s)	Iterações	Melhor iteração	Distância	Desvio	Referências encontradas	Referências existentes
20	17	3708	797	0.0003	0.01%	138	141
40	204	6189	4032	0.0018	0.05%	200	274
60	3416	21577	19059	0.0020	0.10%	237	438
80	7684	20240	19712	0.0030	0.22%	175	481
100	11108	13803	13691	0.0016	0.09%	317	509

As figuras 5.17 e 5.18 apresentam as soluções encontradas em diferentes instantes conforme a evolução da busca , além das soluções de referência, ou seja, as melhores soluções conhecidas. Para cada instante de tempo são marcadas apenas as novas soluções não dominadas encontradas, facilitando a visualização dos resultados.

A instância 11 foi escolhida para o gráfico da Figura 5.17 por apresentar um desvio de 0.00% apesar da distância medida ser 0.0013. Observe que as duas soluções de referência não encontradas não são consideradas para a medida de desvio, o que explica esta diferença. Após 1.1 segundos as soluções encontradas são bastante próximas das de referência com relação às soluções encontradas após 2 iterações ou 0.2 segundos.

A instância 25 se destaca por ser a instância em que a busca com tempo **15x** encontrou o menor percentual de soluções de referência (apenas 13,5%) e é apresentada no gráfico da Figura 5.18. O conjunto solução se aproxima de forma gradual das soluções de referência. Apesar de apenas 5 soluções de referência serem encontradas, a aproximação para **15x** é muito boa, correspondendo às soluções após 1 hora e 48 minutos.

A Figura 5.11 mostra a evolução do desvio para cada ponderação α dos objetivos, onde é notável a dificuldade maior para valores pequenos de α que correspondem às soluções de menor atraso total.

As figuras 5.12 até 5.16 mostram a evolução do desvio médio para cada tamanho de instância em função do tempo. A forma característica é de uma curva exponencial negativa, pois quanto menor o desvio maior a dificuldade em encontrar novas soluções não dominadas e, portanto, maior a dificuldade em reduzir o desvio.

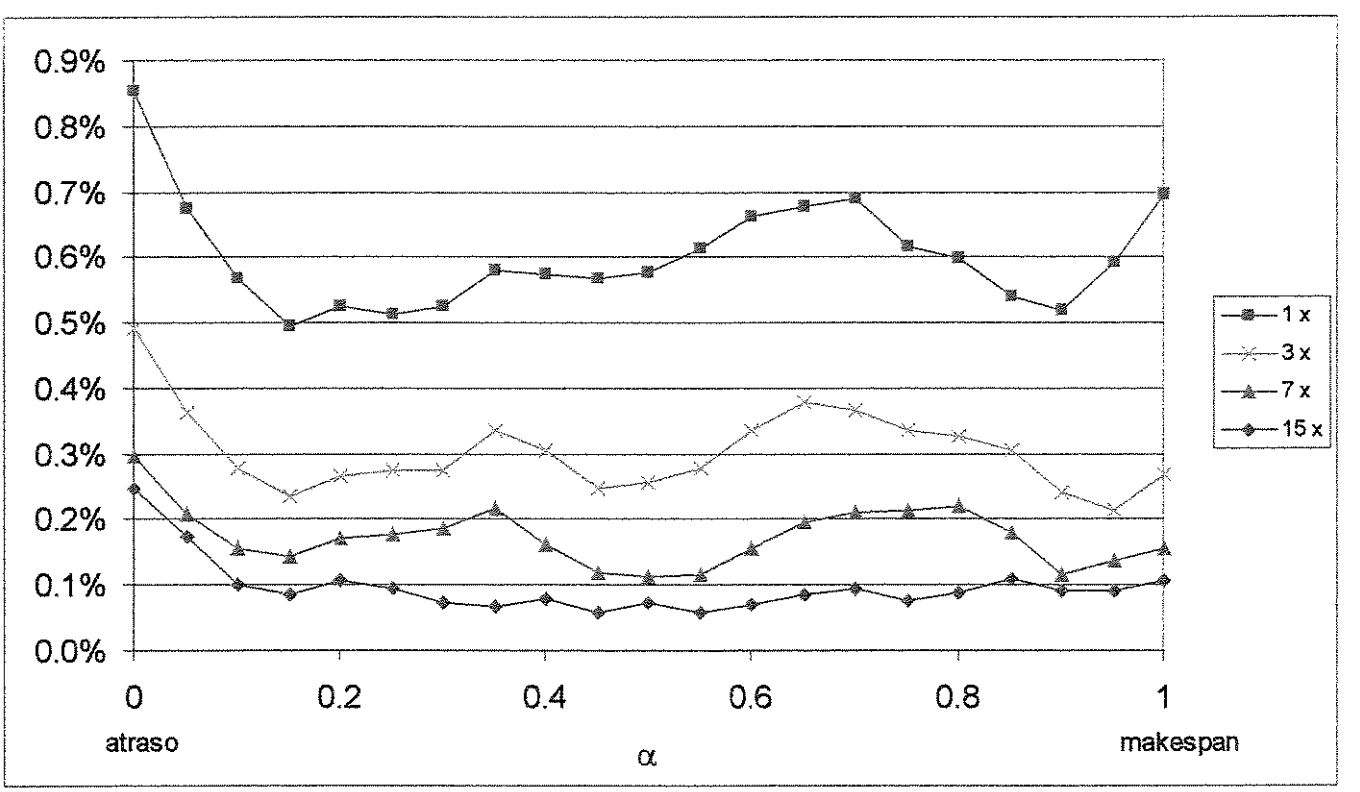

Figura 5.11 - Desvio(α) %

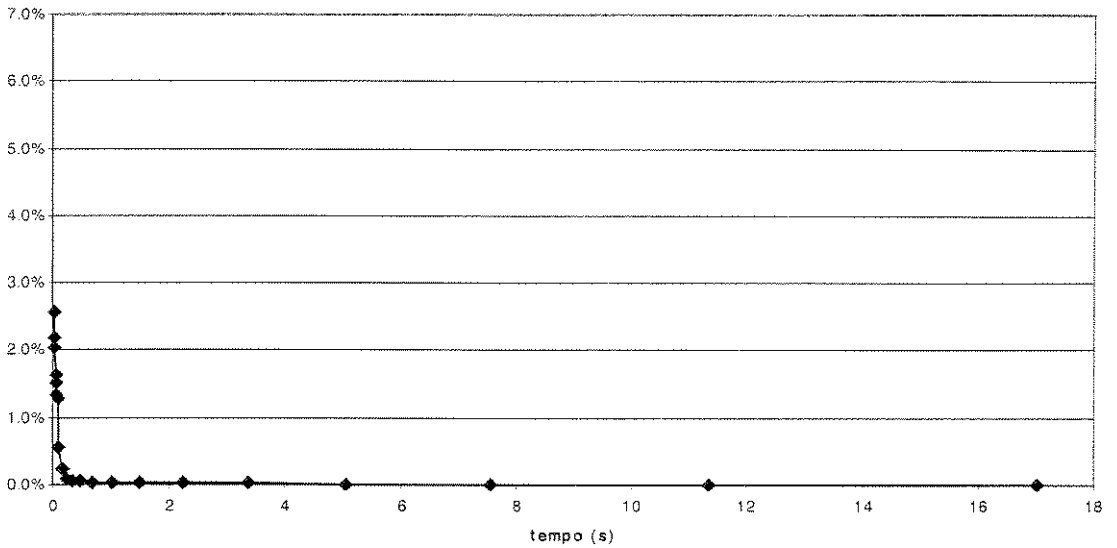

Figura 5.12 - Desvio médio para n = 20

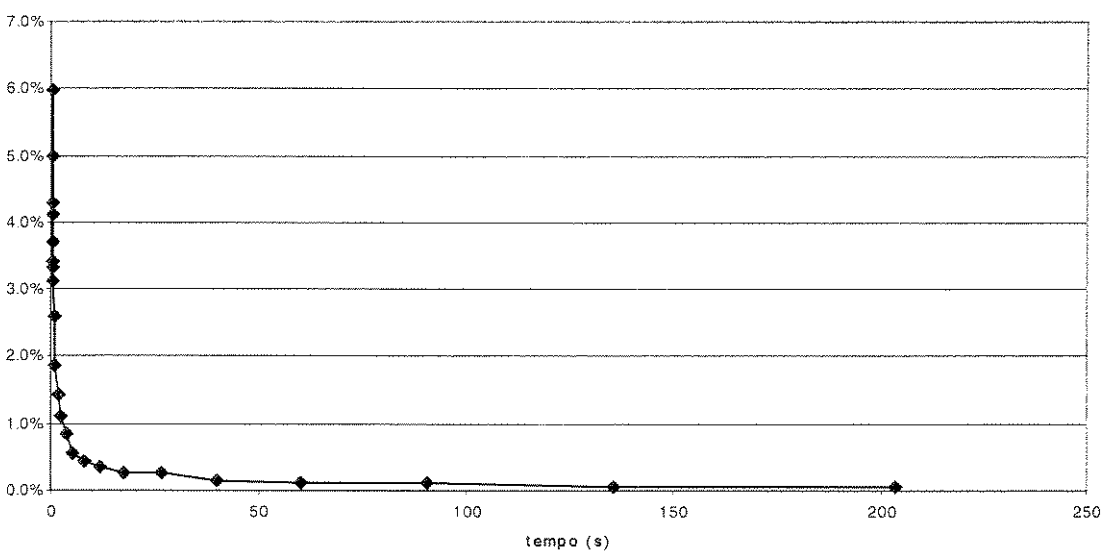

Figura 5.13 - Desvio médio para n = 40

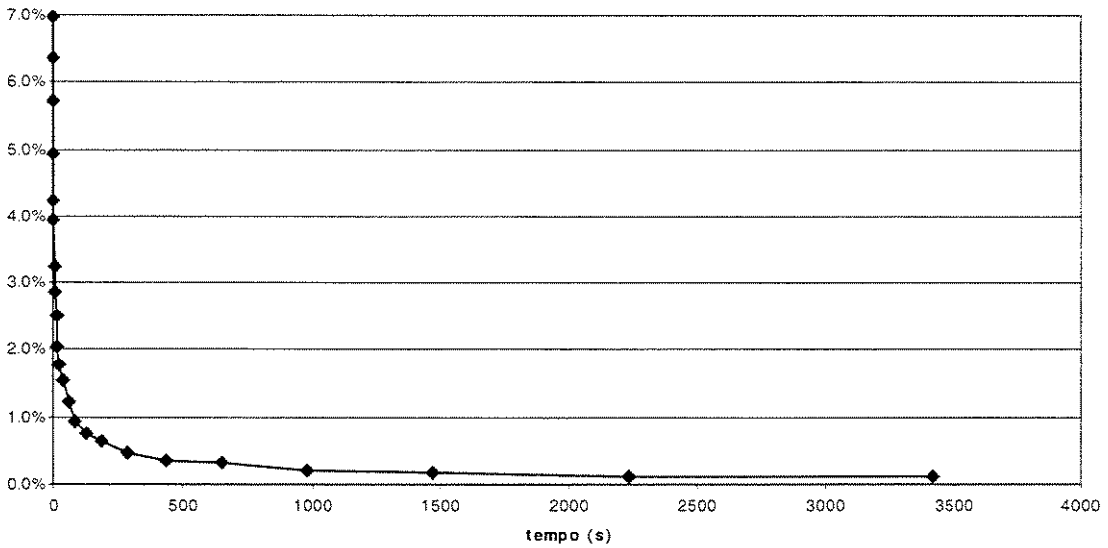

Figura 5.14 - Desvio médio para n = 60

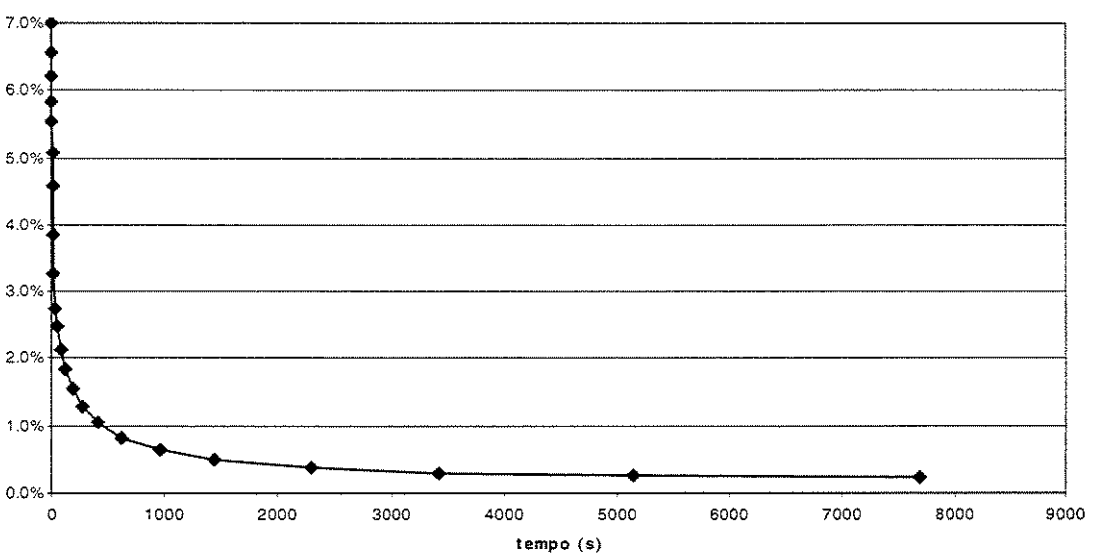

Figura 5.15 - Desvio médio para n = 80

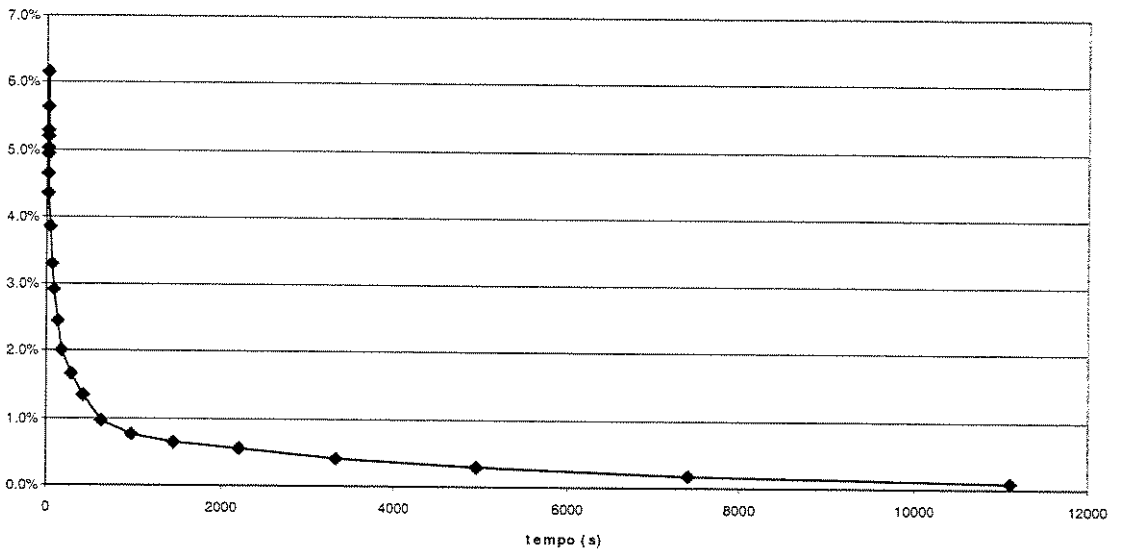

Figura 5.16 - Desvio médio para n = 100

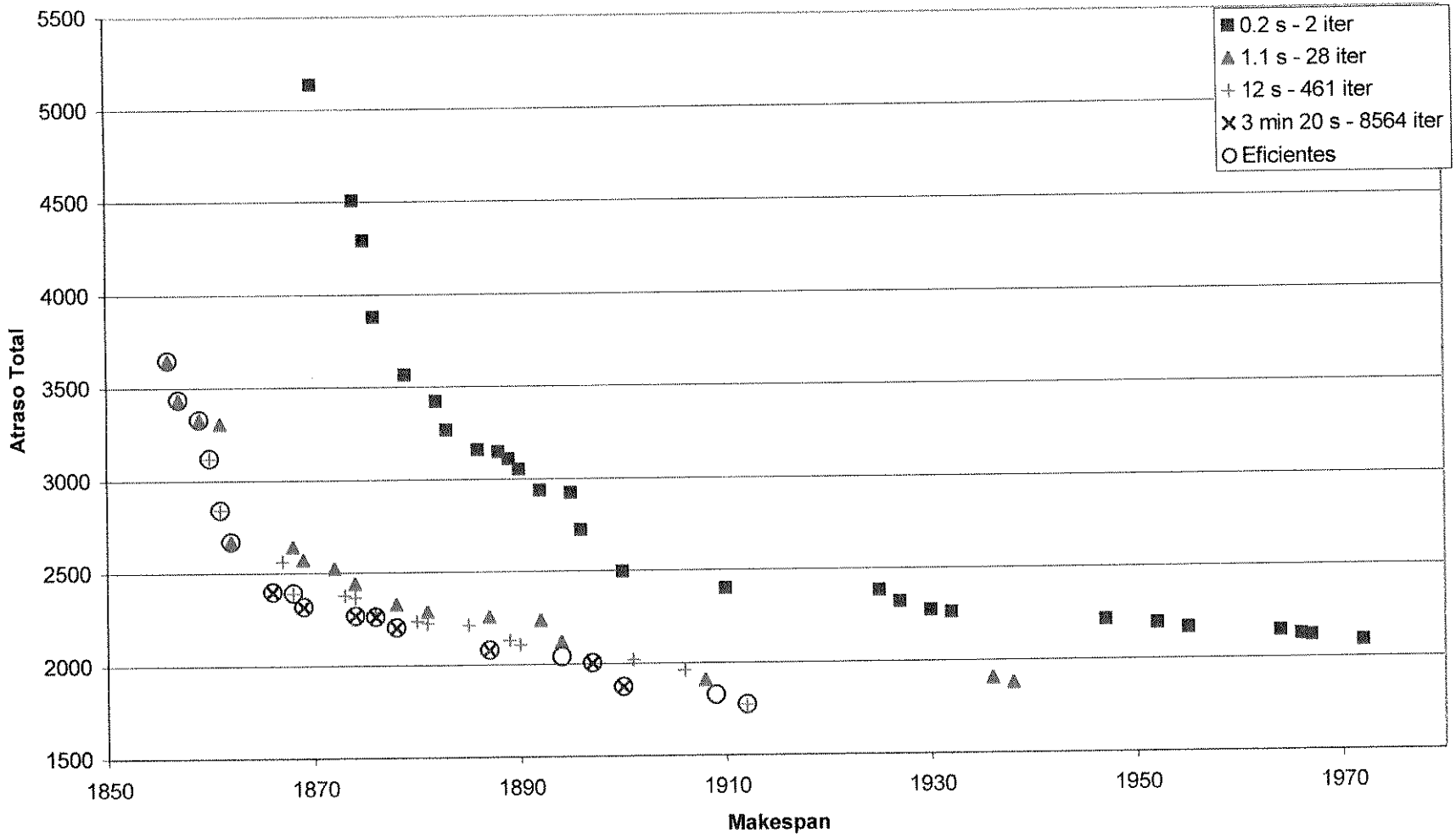

Gráfico 5.17 - Espaço de soluções para a instância 11

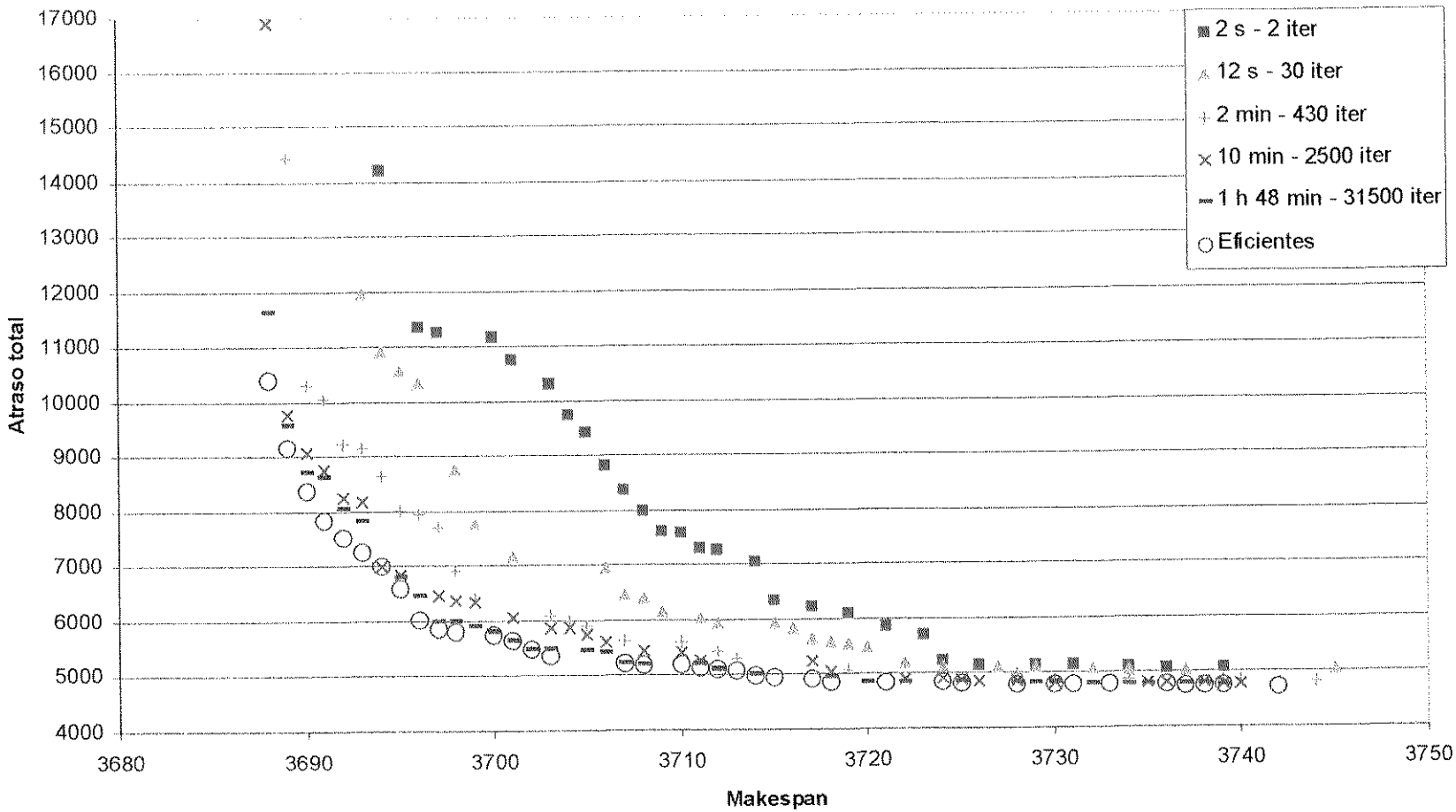

Gráfico 5.18 - Espaço de soluções para a instância 25

6 *Análise individual dos objetivos*

Uma forma de avaliar a qualidade de soluções num problema multiobjetivo é comparar as soluções obtidas através de algoritmos especializados para cada um dos objetivos, propostos na literatura . Esta comparação é importante neste trabalho, uma vez que para o problema aqui abordado não se conhece implementações anteriores que busquem gerar o conjunto de soluções eficientes.

6.1 Análise para o atraso total

6.1.1 Testes nas instâncias geradas neste trabalho

Para analisar a qualidade das soluções para o atraso total aplicou-se o algoritmo memético proposto em França et al. (2001) para as 40 instâncias teste descritas no capítulo 5, e testou-se também uma adaptação mono-objetivo da busca 3-opt(10%) com recomeços. Os resultados para o atraso total foram comparados à melhor solução para o atraso obtida por duas heurísticas multiobjetivo: a melhor testada, 5-grupos com vizinhança 3-opt(0%), e pela melhor heurística para o atraso, 5-grupos com vizinhança 3-opt variável e $\beta_{max} = 10\%$.

A versão mono-objetivo criada para comparação foi feita alterando-se apenas o trecho de código referente à relação de dominância entre duas soluções. Como o objetivo é único, uma solução x domina y se, e somente se, $T(x) < T(y)$, onde $T(x)$ é o atraso total da solução x. O restante da implementação incluindo a heurística construtiva utilizada, combinação de VMPR e ATCS(12), foi mantida inalterada.

Utilizou-se vizinhança 3-opt(10%) e recomeços com perturbação *Double Bridge* realizada na melhor solução encontrada. O critério de parada foi o limite de vizinhos avaliados para a vizinhança 3-opt(10%) apresentado na Tabela 5.1. Foi utilizado preprocessamento, e a estratégia de busca corresponde a Best Improvement uma vez que toda a vizinhança é analisada antes de se realizar um movimento.

Os testes do algoritmo memético foram realizados em um PC com processador AMD K7 de 800 MHz, uma máquina consideravelmente mais veloz que a dos demais testes apresentados neste trabalho, uma estação SUN Ultra com dois processadores de 360 MHz sendo apenas um deles utilizado. A linguagem na qual foi implementado o algoritmo memético foi JAVA, sendo utilizado o compilador JAVA 1.3.

As seguintes siglas são utilizadas na Tabela 6.1:

- MEMÉTICO, algoritmo memético implementado por França et al. (2001)
- MONO-A, versão mono-objetivo da busca 3-opt(10%) com recomeços
- VAR(10%), busca com vizinhança 3-opt variável e β_{max} = 10%, conforme apresentada na seção 5.10
- 5-GRP(0%), estratégia 5-grupos com vizinhança 3-opt(0%), descrita na seção 5.11.

O algoritmo memético utiliza uma vizinhança de troca e inserção reduzida como descrito na seção 4.3.6, e como esta vizinhança apresenta em média o dobro de recomeços que a vizinhança 3-opt(10%) utilizou-se como critério de parada que o número de buscas locais realizadas pelo algoritmo memético seja o dobro do número de buscas locais (recomeços) realizado pela versão mono-objetivo MONO-A. Este critério pode ser observado na Tabela 6.3.

A Tabela 6.1 apresenta os melhores valores obtidos para o atraso total, onde a coluna *Best* contém os melhores valores conhecidos para cada instância. Observa-se que o algoritmo memético obteve bons resultados apenas para n = 20, sendo superado por todos os demais nas instâncias restantes. Apesar da comparação não ser justa por vários motivos como critérios de parada, linguagem, processador, instâncias teste, entre outros, pode-se concluir que os resultados obtidos são de boa

qualidade e competitivos com um algoritmo desenvolvido para minimizar o atraso. Outra observação importante é o melhor desempenho da adaptação mono-objetivo frente às heurísticas multiobjetivo.

Tabela 6.1 – atraso total para cada instância

n	τ	η	R	VAR(10%)	5-GRP(0%)	MONO-A	MEMÉTICO	Best
20	**0.3**	**0.3**	**0.3**	**649**	**649**	**649**	681	~~649~~
20	**0.3**	**0.3**	**0.7**	**592**	**592**	**592**	**592**	~~592~~
20	**0.3**	**0.7**	**0.3**	937	**927**	**927**	973	~~927~~
20	**0.3**	**0.7**	**0.7**	979	959	944	**930**	~~930~~
20	**0.7**	**0.3**	**0.3**	3340	3340	**3317**	**3317**	~~3317~~
20	**0.7**	**0.3**	**0.7**	**5337**	5428	5372	**5337**	~~5337~~
20	**0.7**	**0.7**	**0.3**	5642	**5630**	5642	**5630**	~~5576~~
20	**0.7**	**0.7**	**0.7**	**5763**	**5763**	**5763**	5808	~~5763~~
40	**0.3**	**0.3**	**0.3**	1937	1937	**1873**	1949	~~1873~~
40	**0.3**	**0.3**	**0.7**	55	**47**	49	81	38
40	**0.3**	**0.7**	**0.3**	**1773**	**1773**	1801	2138	~~1773~~
40	**0.3**	**0.7**	**0.7**	839	876	**782**	1049	~~782~~
40	**0.7**	**0.3**	**0.3**	11003	10986	**10972**	11241	10961
40	**0.7**	**0.3**	**0.7**	**16050**	16087	**16050**	16256	15987
40	**0.7**	**0.7**	**0.3**	15245	15454	**15159**	15708	~~15159~~
40	**0.7**	**0.7**	**0.7**	14003	14004	**13912**	14599	~~13912~~
60	**0.3**	**0.3**	**0.3**	3091	**3089**	3091	3329	3082
60	**0.3**	**0.3**	**0.7**	1273	1268	**1168**	1451	~~1168~~
60	**0.3**	**0.7**	**0.3**	3755	3612	**3562**	4148	3526
60	**0.3**	**0.7**	**0.7**	1022	**955**	1032	1505	939
60	**0.7**	**0.3**	**0.3**	23179	23597	**23148**	23715	~~23148~~
60	**0.7**	**0.3**	**0.7**	25052	25106	**24887**	25387	24828
60	**0.7**	**0.7**	**0.3**	**34349**	34738	34677	36248	~~34349~~
60	**0.7**	**0.7**	**0.7**	38310	**37707**	38025	39598	37507
80	**0.3**	**0.3**	**0.3**	4782	4775	**4763**	5313	4734
80	**0.3**	**0.3**	**0.7**	247	240	**237**	412	236
80	**0.3**	**0.7**	**0.3**	9679	**9664**	9771	11503	9570
80	**0.3**	**0.7**	**0.7**	4242	4148	**3965**	6352	~~3965~~
80	**0.7**	**0.3**	**0.3**	45913	46048	**45868**	47367	45470
80	**0.7**	**0.3**	**0.7**	**38663**	39111	38736	39387	38394
80	**0.7**	**0.7**	**0.3**	50666	50686	**50367**	54548	49907
80	**0.7**	**0.7**	**0.7**	59838	**59631**	59658	62894	59013
100	**0.3**	**0.3**	**0.3**	8219	8156	**8067**	8903	8002
100	**0.3**	**0.3**	**0.7**	283	**245**	258	593	231
100	**0.3**	**0.7**	**0.3**	10127	**9844**	10043	13036	~~9844~~
100	**0.3**	**0.7**	**0.7**	357	367	**170**	1426	119
100	**0.7**	**0.3**	**0.3**	85295	84986	**84911**	87385	84805
100	**0.7**	**0.3**	**0.7**	44200	44405	**43705**	45809	43502
100	**0.7**	**0.7**	**0.3**	81597	81975	**81230**	89782	80683
100	**0.7**	**0.7**	**0.7**	62577	63096	**62265**	69539	61455

Tabela 6.2 – desvio percentual médio com relação a Best

VAR(10%)	5-GRP(0%)	MONO-A	MEMÉTICO
8.6%	7.6%	2.9%	46.1%

Tabela 6.3 – Tempo para cada tamanho de instância

	n				
	20	**40**	**60**	**80**	**100**
Var (10%)	1.2	12.9	206.6	459.3	701.1
5-grupos(0%)	1.3	14.4	231.4	496.1	770.3
Mono-A	0.9	11.2	180.7	410.6	662.2
Memético	1.2	10.0	115.0	240.0	340.0

Tabela 6.4 – Número de buscas locais para cada tamanho de instância

	n				
	20	**40**	**60**	**80**	**100**
Mono-A	770	905	2984	2692	1513
Memético	1400	1800	6000	5400	3000

Os tempos apresentados na Tabela 6.3 mostram que o critério de parada para o algoritmo memético tem efeito distinto ao utilizado nos demais, causando um crescimento menor no tempo computacional conforme aumenta o tamanho da instância. Como o processador utilizado é consideravelmente mais veloz, considera-se que nas instâncias menores o algoritmo memético foi favorecido pelo critério de parada com relação aos demais.

A Tabela 6.4 ilustra o critério de parada utilizado para o algoritmo memético, aproximadamente o dobro de buscas locais realizadas por MONO-T.

6.1.2 Testes nas instâncias de Rubin e Ragatz

No trabalho de Rubin e Ragatz (1995) foram geradas 32 instâncias de forma aleatória, com um procedimento parametrizado semelhante ao descrito nas seções 3.1 e 5.2. Essas instâncias foram utilizadas em testes de um algoritmo genético para a minimização do atraso total, comparando-se os resultados com um algoritmo de múltiplos recomeços e uma implementação de Branch & Bound. Essas instâncias também foram utilizadas em Tan et al. (2000), que compara os algoritmos de Rubin e Ragatz com Simulated Annealing (SA), e em França et al. (2001), onde se compara um algoritmo memético com o algoritmo genético de Rubin e Ragatz.

Nesta seção comparamos os resultados obtidos nestes trabalhos com os mesmos algoritmos da seção 6.1.1.

Utilizou-se como critério de parada o tempo máximo utilizado em Tan et al para um PC Intel Pentium 90 MHz., considerando-se a diferença entre os processadores uma vez que nestes testes utilizou-se um PC Intel Pentium III de 750 MHz, sendo os tempos ajustados conforme descritos na Tabela 6.5. Para o algoritmo memético foram utilizados os melhores resultados publicados em França et al. (2001), que correspondem a dois minutos em um PC Intel Pentium II 266. Os resultados são apresentados com as seguintes siglas:

- BEST – atraso total da melhor solução obtida para cada instância, determinado a partir dos resultados obtidos pelos algoritmos comparados. Os desvios nas demais colunas são calculados a partir destes valores conforme a equação 6.1. As soluções destacadas em negrito são ótimas.
- B & B – resultados de algoritmo de *Branch and Bound* publicados em Tan et al. (2000), onde o algoritmo foi limitado a avaliar no máximo 5 milhões de nós.
- GENÉTICO, SA, RANDOM – resultados publicados em Tan et al. (2000) para um algoritmo genético, uma implementação de *simulated annealing* e uma busca local com recomeços aleatórios e vizinhança de troca (*pairwise interchange,* ver seção 4.3.2). Estes algoritmos tiveram como critério de parada um tempo aproximadamente 8,3 vezes superior ao da Tabela 6.4, correspondendo à diferença entre os processadores (750 MHz / 90 MHz). Os resultados apresentados são médias entre 20 execuções.
- MEMÉTICO – resultados descritos para o algoritmo proposto em França et al. (2001), o mesmo utilizado na seção 6.1.1. O tempo foi limitado a dois minutos, que corresponderiam a 42 segundos na Tabela 6.5, considerando-se a diferença entre os processadores (750 MHz / 266 MHz). Os resultados apresentados são médias entre 5 execuções.
- MONO-A, versão mono-objetivo da busca 3-opt(10%) com recomeços limitada aos tempos da Tabela 6.5. Os resultados apresentados são médias entre 20 execuções, mesmo número utilizado em Tan et al. (2000) de forma a reduzir o efeito de variações aleatórias, sem caracterizar uma vantagem de desempenho.

- VAR(10%), busca com vizinhança 3-opt variável e β_{max} = 10%, conforme apresentada na seção 5.10, limitada aos tempos da Tabela 6.5. Os resultados apresentados são médias entre 20 execuções.
- 5-GRP(0%), estratégia 5-grupos com vizinhança 3-opt(0%), descrita na seção 5.11, limitada aos tempos da Tabela 6.5. Os resultados apresentados são médias entre 20 execuções.

$$Desvio\ (\%) = \frac{(\text{Sol.Final}_i - \text{BEST}_i)}{\text{BEST}_i} * 100 \qquad (6.1)$$

Tabela 6.5 – Tempo máximo de execução e tempo médio para obter a melhor solução (segundos)

n	prob	max	MONO-A	VAR(10%)	5-GRP(0%)
15	**1**	43	0.0	0.0	0.0
	2	43	0.0	0.0	0.0
	3	43	0.0	0.0	0.1
	4	43	0.0	0.0	0.1
	5	43	0.0	0.0	0.0
	6	43	0.0	0.0	0.0
	7	43	0.0	0.0	0.0
	8	43	0.0	0.1	0.2
25	**1**	115	2.5	0.0	0.0
	2	115	0.0	0.0	0.0
	3	115	0.7	2.4	7.2
	4	115	0.0	0.0	0.0
	5	115	0.0	0.0	0.0
	6	115	0.0	0.0	0.0
	7	115	0.6	2.5	2.7
	8	115	0.2	1.7	2.0
35	**1**	216	0.4	0.2	0.3
	2	216	0.0	0.0	0.0
	3	216	4.5	24.0	19.9
	4	216	4.0	5.4	5.9
	5	216	83.5	94.2	84.3
	6	216	0.0	0.1	0.0
	7	216	1.2	1.5	1.5
	8	216	3.5	28.1	57.9
45	**1**	432	5.0	62.7	49.3
	2	432	0.1	0.1	0.1
	3	432	35.7	108.4	37.2
	4	432	60.2	162.3	177.9
	5	432	16.9	12.9	64.6
	6	432	0.1	0.1	0.1
	7	432	62.5	251.4	156.5
	8	432	43.4	150.1	187.9
	Total	**6451**	**325.3**	**908.5**	**855.8**

Tabela 6.6 – Desvio percentual acima da melhor solução

n	prob	BEST	B & B	GENÉTICO	SA	RANDOM	MEMÉTICO	MONO-A	VAR(10%)	5-GRP(0%)
15	1	***90***	**0.0%**	4.4%	3.3%	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	2	***0***	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	3	***3418***	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	4	***1067***	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	5	***0***	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	6	***0***	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	7	***1861***	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	8	***5660***	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
25	1	261	1.1%	2.7%	3.1%	2.0%	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	2	***0***	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	3	3146	11.6%	11.8%	**0.6%**	11.2%	11.2%	11.2%	11.2%	11.2%
	4	***0***	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	5	***0***	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	6	***0***	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	7	7225	**0.0%**	6.1%	1.1%	0.1%	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	8	1915	7.9%	1.6%	**0.0%**	**0.0%**	0.0%	**0.0%**	**0.0%**	**0.0%**
35	1	12	150.0%	525.0%	258.3%	229.3%	50.0%	**0.0%**	**0.0%**	**0.0%**
	2	***0***	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	3	17587	1.1%	1.5%	1.9%	1.3%	0.3%	**0.0%**	0.1%	0.1%
	4	19092	1.0%	2.0%	1.7%	0.9%	0.4%	**0.0%**	**0.0%**	**0.0%**
	5	228	27.6%	75.2%	26.1%	22.4%	4.3%	**0.1%**	0.1%	0.1%
	6	***0***	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	7	12969	2.4%	9.1%	2.2%	1.5%	0.6%	**0.0%**	**0.0%**	**0.0%**
	8	4732	41.7%	1.2%	0.6%	0.3%	**0.0%**	**0.0%**	**0.0%**	**0.0%**
45	1	97	19.6%	118.6%	54.6%	46.4%	12.6%	0.7%	**0.5%**	0.7%
	2	***0***	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	3	26506	2.2%	3.8%	2.4%	2.0%	0.7%	0.0%	**0.0%**	0.0%
	4	15206	4.8%	3.2%	3.6%	2.0%	0.2%	**0.0%**	0.0%	0.2%
	5	200	17.0%	121.5%	41.0%	39.0%	12.2%	0.1%	**0.0%**	0.2%
	6	***0***	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	7	23789	5.4%	7.3%	2.8%	2.3%	0.5%	0.0%	0.0%	**0.0%**
	8	22815	5.7%	13.3%	2.3%	1.6%	0.3%	**0.0%**	0.0%	0.0%
	Média		**9.35%**	**28.38%**	**12.67%**	**11.32%**	**2.91%**	**0.38%**	**0.37%**	**0.39%**

Os resultados da Tabela 6.6 comprovam a boa qualidade das soluções obtidas pelos algoritmos desenvolvidos neste trabalho baseados na vizinhança 3-OPT: MONO-A, VAR(10%) e 5-GRP(0%). A diferença entre estes é muito pequena, sendo que o algoritmo MEMÉTICO apresentou bons resultados na maioria das instâncias apesar de ser executado com um tempo menor que os demais. Entretanto, a Tabela 6.6 mostra que na maioria das instâncias os algoritmos baseados em 3-OPT utilizaram um tempo médio pequeno antes da estagnação da busca (a medida de tempo é feita até que a melhor solução do algoritmo naquela execução seja encontrada), não sendo o tempo o principal motivo da vantagem nos resultados e sim a qualidade das soluções encontradas. Destaca-se ainda que a versão mono-objetivo da busca MONO-A encontra soluções de qualidade em tempo menor que as buscas multiobjetivo, dada a complexidade de aproximar a fronteira eficiente ser maior que a de minimizar um único objetivo.

Observe-se que na instância 3, para $n = 25$, os resultados são bastante atípicos pois SA obtém soluções muito melhores enquanto que os melhores algoritmos encontram soluções idênticas. Existe a possibilidade de um erro na publicação deste resultado, apesar dos autores não terem realizado nenhuma correção até o presente.

6.2 Análise para o *makespan*

6.2.1 Testes nas instâncias geradas neste trabalho

De forma análoga à seção 6.1.1, compara-se aqui a qualidade das soluções que minimizam apenas o valor do *makespan*. Neste caso, o problema torna-se equivalente ao problema do caixeiro viajante assimétrico (PCVA), para o qual o tamanho das instâncias aqui abordadas é considerado pequeno. Portanto, é natural que as soluções encontradas sejam próximas à solução ótima, e que a diferença relativa entre as soluções dos diferentes algoritmos seja pequena.

As 40 instâncias teste descritas na seção 5.2 foram resolvidas pelo algoritmo memético desenvolvido para o PCVA por Buriol (2000), que teve bom desempenho nos testes realizados para instâncias clássicas deste problema. Testou-se também uma adaptação mono-objetivo da busca 3-opt(0%) com recomeços, uma vez que a

folga visa obter soluções melhores para o atraso enquanto que $\beta = 0\%$ é suficiente para que os vizinhos que melhorem o *makespan* sejam avaliados. Os resultados foram comparados com a melhor solução obtida por duas heurísticas multiobjetivo: a melhor testada, 5-grupos com vizinhança 3-opt(0%), e pela melhor heurística para o *makespan*, 5-grupos com vizinhança 3-opt(5%).

Para a versão mono-objetivo alterou-se apenas o trecho de código referente à relação de dominância entre duas soluções, assim como a versão mono-objetivo para o atraso: uma solução x domina y se, e somente se, $M(x) < M(y)$, onde $M(x)$ é o valor do *makespan* da solução x. O restante da implementação foi mantido inalterado, inclusive a heurística construtiva utilizada, combinação de VMPR e ATCS(12). Utilizou-se vizinhança 3-opt(0%) e recomeços com perturbação Double Bridge realizada na melhor solução encontrada. O critério de parada foi o limite de vizinhos avaliados para a vizinhança 3-opt(0%) apresentado na Tabela 5.1. Foi utilizado preprocessamento, sendo que ao final deste a solução é um ótimo local para o *makespan*.

Os testes do algoritmo memético foram realizados em uma estação Sun Ultra 1, uma máquina mais lenta que a utilizada nos demais testes apresentados neste trabalho, de 167 MHz. A linguagem utilizada foi JAVA, com o compilador JAVA 1.2.

As seguintes siglas são utilizadas na Tabela 6.7:

- MEMÉTICO, algoritmo memético implementado por Buriol (2000)
- MONO-M, versão mono-objetivo da busca 3-opt(0%) com recomeços para o *makespan*
- 5-GRP(0%), estratégia 5-grupos com vizinhança 3-opt(0%), descrita na seção 9.11
- 5-GRP(5%), estratégia 5-grupos com vizinhança 3-opt(5%).

Testes realizados mostram que na estação Ultra 1 a busca multiobjetivo desenvolvida neste trabalho é aproximadamente 2.15 vezes mais lenta do que na estação Ultra 60 que utilizou-se nos demais testes, que é a proporção entre a velocidade dos processadores (360 MHz / 167 MHz = 2.16). Assim, determinou-se

como critério de parada para o algoritmo memético que o tempo fosse 2.15 vezes o tempo médio utilizado pelos algoritmos multiobjetivos para cada tamanho de instância.

A Tabela 6.7 apresenta os melhores valores para o *makespan* obtidos, onde a coluna Best contém os melhores valores conhecidos para cada instância. Apesar do algoritmo memético obter resultados próximos de Best, seus resultados foram piores ou iguais aos demais algoritmos exceto em duas instâncias nas quais superou 5-GRP(5%) por uma unidade. Destaca-se também o excelente desempenho da versão mono-objetivo MONO-M, que atingiu Best em 36 das 40 instâncias, além de não ter sido superado pelos demais em qualquer instância.

Tabela 6.7 – *makespan* para cada instância

n	τ	η	R	5-GRP(5%)	5-GRP(0%)	MONO -M	MEMÉTICO	Best
20	0.3	0.3	0.3	**1248**	**1248**	**1248**	**1248**	~~1248~~
20	0.3	0.3	0.7	**826**	**826**	**826**	**826**	~~826~~
20	0.3	0.7	0.3	**1130**	**1130**	**1130**	**1130**	~~1130~~
20	0.3	0.7	0.7	**1151**	**1151**	**1151**	**1151**	~~1151~~
20	0.7	0.3	0.3	**826**	**826**	**826**	**826**	~~826~~
20	0.7	0.3	0.7	**1356**	**1356**	**1356**	**1356**	~~1356~~
20	0.7	0.7	0.3	**1354**	**1354**	**1354**	**1354**	~~1354~~
20	0.7	0.7	0.7	**1184**	**1184**	**1184**	**1184**	~~1184~~
40	0.3	0.3	0.3	**2120**	**2120**	**2120**	**2120**	~~2120~~
40	0.3	0.3	0.7	**2228**	**2228**	**2228**	**2228**	~~2228~~
40	0.3	0.7	0.3	**1856**	**1856**	**1856**	**1856**	~~1856~~
40	0.3	0.7	0.7	**2137**	**2137**	**2137**	**2137**	~~2137~~
40	0.7	0.3	0.3	**1832**	**1832**	**1832**	**1832**	~~1832~~
40	0.7	0.3	0.7	**2372**	2373	**2372**	**2372**	~~2372~~
40	0.7	0.7	0.3	**2214**	**2214**	**2214**	**2214**	~~2214~~
40	0.7	0.7	0.7	**2219**	**2219**	**2219**	**2219**	~~2219~~
60	0.3	0.3	0.3	**3017**	**3017**	**3017**	3019	~~3017~~
60	0.3	0.3	0.7	3194	**3193**	**3193**	3195	3192
60	0.3	0.7	0.3	**3189**	3190	**3189**	**3189**	~~3189~~
60	0.3	0.7	0.7	2970	**2969**	**2969**	2971	~~2969~~
60	0.7	0.3	0.3	**2875**	2877	**2875**	2877	2874
60	0.7	0.3	0.7	**3338**	**3338**	**3338**	3341	~~3338~~
60	0.7	0.7	0.3	3249	3249	**3248**	3250	~~3248~~
60	0.7	0.7	0.7	3395	3397	**3394**	3398	~~3394~~
80	0.3	0.3	0.3	3689	**3688**	**3688**	3693	~~3688~~
80	0.3	0.3	0.7	3759	3759	**3758**	3765	~~3758~~
80	0.3	0.7	0.3	4636	**4635**	**4635**	4644	~~4635~~
80	0.3	0.7	0.7	4240	4240	**4238**	4247	~~4238~~
80	0.7	0.3	0.3	3944	3945	**3943**	3947	~~3943~~
80	0.7	0.3	0.7	3748	3748	**3746**	3749	~~3746~~
80	0.7	0.7	0.3	4408	4408	**4407**	4411	4406
80	0.7	0.7	0.7	3912	**3911**	**3911**	3918	~~3911~~
100	0.3	0.3	0.3	4818	4817	**4816**	4823	~~4816~~
100	0.3	0.3	0.7	**4965**	**4965**	**4965**	4968	~~4965~~
100	0.3	0.7	0.3	5050	5050	**5047**	5063	~~5047~~
100	0.3	0.7	0.7	5357	5360	**5355**	5373	~~5355~~
100	0.7	0.3	0.3	5567	5567	**5566**	5571	5565
100	0.7	0.3	0.7	4700	4700	**4698**	4704	~~4698~~
100	0.7	0.7	0.3	5177	5176	**5175**	5190	~~5175~~
100	0.7	0.7	0.7	4796	4798	**4795**	4811	4793

Tabela 6.8 – desvio percentual médio com relação à Best

5-GRP(5%)	5-GRP(0%)	MONO -M	MEMÉTICO
0.020%	0.023%	0.004%	0.089%

Tabela 6.9 – Tempo para cada tamanho de instância

	n				
	20	**40**	**60**	**80**	**100**
5-GRP(5%)	1.3	13.8	227.0	485.3	756.5
5-GRP(0%)	1.3	14.4	231.4	496.1	770.3
MONO-M	1.4	17.1	260.3	573.5	910.7
MEMÉTICO	2.6	30.2	476.0	1037.0	1807.0

Os tempos apresentados na Tabela 6.9 ilustram o critério de parada utilizado para o algoritmo memético, com tempo aproximadamente duas vezes maior que os demais compensando a diferença na velocidade do processador. Já para a adaptação MONO-M o tempo foi maior do que as heurísticas multiobjetivo, pois praticamente todo o tempo computacional é gasto com o preprocessamento uma vez que sua saída já é um ótimo local para o *makespan*. Durante o preprocessamento os vizinhos que piorem o *makespan* não são avaliados e demora-se mais tempo até encontrar um movimento a ser avaliado o que aumenta o tempo médio por vizinho avaliado e, conseqüentemente, o tempo total do algoritmo.

6.2.2 Testes com instâncias da TSPLIB

As instâncias da TSPLIB formam um conjunto de testes comumente utilizados para o problema do caixeiro viajante. Este conjunto foi sugerido por Reinelt (1991), e está disponível na internet em:

http://www.informatik.uni-heidelberg.de/groups/comopt/software/TSPLIB95/index.html. As instâncias ftv90 a ftv160 são geradas a partir de ftv170 retirando-se as últimas cidades. Foram utilizadas as instâncias assimétricas, e comparou-se os resultados com os obtidos por Buriol (2000), que obteve bons resultados para estas instâncias comparados a diversos trabalhos anteriores.

Não é possível aplicar os algoritmos multiobjetivos desenvolvidos para estas instâncias, uma vez que não existem dados sobre data de entrega para cálculo do atraso total. Assim, testou-se apenas a adaptação monobjetivo descrita na seção 6.2.1. Nestes testes utilizou-se um computador Pentium III 750MHz para MONO-M, enquanto que o algoritmo memético utilizou um Pentium II 450Mhz, conforme descrito em Buriol(2001).

O critério de parada utilizado em Buriol(2001) foi de 1000 gerações ou 300 segundos de CPU. Para MONO-M utilizou-se como critério de parada 192 segundos de CPU ou o tempo estimado para 1000 gerações do algoritmo memético. Este tempo estimado foi calculado com base no tempo médio por geração em cada instância e na diferença entre os processadores. Ambos os algoritmos foram interrompidos quando encontram a solução ótima, e foram realizadas 30 execuções em cada instância da TSPLIB com diferentes sementes aleatórias, buscando minimizar a influência das escolhas aleatórias do algoritmo na sua avaliação.

Tabela 6.10 – Resultados para instâncias da TSPLIB

				3-OPT (750Mhz)					Memético (450Mhz)			
	cidades	valor ótimo	limite (s)	achou ótimo	qual	iter	tempo (s)	melhor tempo(s)	achou ótimo	qual	gerações	tempo (s)
br17	17	39	1.0	30	0.00%	1	0.00	0.00	30	0.00%	2	0
ftv33	34	1286	3.2	30	0.00%	90	0.02	0.02	30	0.00%	11	0.06
ftv35	36	1473	2.9	5	0.11%	13,961	2.43	0.04	30	0.00%	81	0.39
ftv38	39	1530	2.9	3	0.12%	12,494	2.65	0.02	13	0.07%	786	3.82
p43	43	5620	4.5	30	0.00%	325	0.86	0.86	30	0.00%	23	0.17
ftv44	45	1613	3.5	26	0.17%	5,391	1.14	0.68	28	0.04%	381	2.22
ftv47	48	1776	4.8	30	0.00%	260	0.09	0.09	30	0.00%	52	0.42
ry48p	48	14422	4.9	30	0.00%	236	0.12	0.12	30	0.00%	112	0.92
ft53	53	6905	5.2	30	0.00%	550	0.40	0.40	30	0.00%	93	0.8
ftv55	56	1608	6.8	30	0.00%	60	0.02	0.02	30	0.00%	21	0.24
ftv64	65	1839	7.6	28	0.05%	1,396	0.56	0.05	30	0.00%	28	0.35
ft70	70	38673	9.1	12	0.05%	7,434	5.60	0.15	30	0.00%	306	4.64
ftv70	71	1950	8.0	30	0.00%	233	0.11	0.11	30	0.00%	67	0.9
ftv90	91	1579	12.3	27	0.04%	5,391	2.67	1.35	30	0.00%	29	0.6
kro124p	100	36230	16.4	24	0.38%	7,637	5.55	2.46	30	0.00%	62	1.71
ftv100	101	1788	12.8	24	0.07%	7,032	3.93	1.42	28	0.02%	95	2.02
ftv110	111	1958	15.8	21	0.18%	7,456	5.84	1.25	29	0.00%	78	2.05
ftv120	121	2166	16.4	24	0.09%	5,765	6.20	3.28	22	0.07%	342	9.35
ftv130	131	2307	19.1	21	0.13%	6,739	7.47	2.21	25	0.04%	271	8.63
ftv140	141	2420	20.9	14	0.13%	10,997	12.04	1.21	1	0.30%	970	33.73
ftv150	151	2611	22.6	29	0.09%	2,031	2.68	1.98	2	0.71%	941	35.51
ftv160	161	2683	27.9	20	0.24%	9,474	12.85	5.02	3	0.64%	912	42.33
ftv170	171	2755	33.7	13	0.27%	12,406	19.56	1.40	28	0.02%	229	12.86
rbg323	323	1326	192.0	29	0.00%	17	38.35	32.34	30	0.00%	33	12.61
rbg358	358	1163	192.0	30	0.00%	15	9.67	9.67	30	0.00%	65	28.78
rbg403	403	2465	192.0	29	0.00%	5	12.66	6.10	30	0.00%	7	6.01
rbg443	443	2720	192.0	30	0.00%	3	12.10	12.10	30	0.00%	6	7.07
Média				**24.0**	**0.08%**	**4348**	**6.13**	**3.12**	**25.5**	**0.07%**	**222**	**8.08**

A Tabela 6.10 apresenta os resultados para cada instância da TSPLIB. Os valores ótimos são conhecidos, e assim na coluna "sucessos" apresenta-se o número de vezes que o ótimo foi obtido nas 30 execuções. Já a coluna "qual" refere-se ao desvio percentual médio da solução com relação ao ótimo, conforme equação 6.2 . Apresenta-se também o número médio de iterações (recomeços) para MONO-M e o número médio de gerações realizadas pelo algoritmo memético, assim como o tempo médio em cada execução. Para MONO-M também é apresentada uma coluna "melhor tempo", que contém o tempo médio até a melhor solução encontrada.

$$Qual\ (\%) = \frac{\sum_{i=1}^{\#exe} \frac{(\text{Sol. Final}_i - \text{Sol. Ótima})}{(\text{Sol. Ótima})} * 100}{\#\,exe} \qquad (6.2)$$

As instâncias ftv35 e ftv38, consideradas de fácil resolução dado o pequeno número de cidades, apresentaram uma dificuldade inesperada. O que ocorreu foi que o algoritmo atingia diversas vezes um mesmo ótimo local e as perturbações realizadas foram facilmente revertidas retornando ao mesmo ótimo local. Esta dificuldade não se apresentou no caso multiobjetivo dado que se trabalha com um conjunto de soluções, o que diversifica as soluções evitando a estagnação precoce. O tempo para obter-se a melhor solução nestas instâncias mostra este fato com clareza, onde em poucos décimos de segundo se obtém a melhor solução e não se conseguem melhorias no restante de tempo disponível. Portanto, para aplicar a metodologia desenvolvida neste trabalho ao PCVA deve-se considerar modificações no processo de recomeço.

Nas instâncias consideradas difíceis, ftv100 a ftv170, MONO-M atingiu bons resultados, o que levou a um resultado global de qualidade bastante próximo do algoritmo memético. Deve-se lembrar que nenhuma adaptação do algoritmo foi feita para este conjunto de instâncias específico nem para o problema do caixeiro viajante, apenas desconsiderou-se o atraso na avaliação de dominância entre soluções. Espera-se que uma implementação da busca local específica para o PCVA e com modificações no processo de recomeço gere resultados de melhor qualidade e em menor tempo.

7 Conclusões

A utilização de programação multiobjetivo na modelagem e resolução de problemas combinatórios é uma evolução natural das pesquisas na área de otimização combinatória, pois auxilia na resolução de conflitos que ocorrem em qualquer processo decisório onde diferentes soluções atendem a objetivos distintos. A escolha de qual solução deve ser adotada é muitas vezes subjetiva, podendo inclusive variar conforme o momento da tomada de decisão, uma vez que mudanças na política decisória são muito freqüentes. A geração do conjunto de soluções eficientes além de possibilitar a escolha entre várias soluções possíveis permite ainda analisar o quanto se ganha ou perde em cada objetivo dadas as soluções disponíveis, possibilitando uma análise semelhante à análise de sensibilidade para programação linear.

Sempre é vantajosa uma modelagem multiobjetivo frente à otimização mono-objetivo, uma vez que esta está contida na primeira, mas a questão é se estas vantagens compensam o aumento da dificuldade na implementação e resolução do problema. Esta questão deve ser respondida caso a caso, dependendo da aplicação e recursos disponíveis como tempo para implementação e tempo para resolução, mas acredita-se que os resultados aqui apresentados demonstram ser possível obter um conjunto solução próximo ao eficiente em tempo razoável para um problema complexo como o de seqüenciamento abordado.

Para problemas combinatórios torna-se muitas vezes inviável a utilização de métodos exatos, sendo necessária a utilização de heurísticas. Uma das dificuldades relacionada à programação combinatória multiobjetivo é a avaliação da qualidade de aproximações ao conjunto de soluções eficientes. A metodologia utilizada baseou-se nas medidas de desvio e distância descritas no capítulo 1 e foi satisfatória na avaliação da qualidade destas aproximações. Em alguns casos onde a diferença entre dois conjuntos era muito pequena houve um conflito entre estas medidas, o que não é um erro uma vez que este tipo de avaliação não é exata e envolve fatores por vezes subjetivos e dependentes da aplicação. A principal conclusão que se chegou sobre este ponto foi que existem casos em que um algoritmo foi claramente superior ao outro e ambas as medidas indicaram isto, mas

por vezes ocorreram testes em que as medidas indicaram pequenas diferenças no desempenho podendo levar a decisões contraditórias onde na realidade não se pode dizer qual algoritmo foi melhor. Não se notou uma diferença significativa entre as medidas para recomendar a utilização de uma ou outra em novos trabalhos, sendo a escolha de uma delas uma questão arbitrária.

O problema de minimizar o atraso total e o *makespan* em uma máquina com tempos de preparação deve ser considerado de difícil resolução, uma vez que apenas a minimização do atraso total tem sido alvo de recentes trabalhos como em Armentano e Mazzini (2000), França et al. (2001), Lee et al. (1997) e Tan et al. (2000). O primeiro passo na resolução deste problema foi a elaboração e teste de heurísticas construtivas multiobjetivo, descritas no capítulo 1. Concluiu-se que a utilização de uma adaptação das heurísticas ATCS e VMP (vizinho mais próximo) em conjunto forma um procedimento que com pouco esforço computacional gera um bom conjunto de soluções não-dominadas, fato comprovado pela dificuldade da vizinhança API em melhorar a qualidade destas soluções, conforme descrito na seção 5.5 .

Optou-se por utilizar uma estrutura de busca local com recomeços como heurística de melhoria para o problema. Esta estrutura foi escolhida por ter obtido bons resultados para o PCV publicados por Johnson e McGeoch (1997) e por ser de simples implementação, possibilitando que uma maior atenção fosse dada aos demais componentes da busca, em especial à escolha da vizinhança.

Os testes da seção 5.5 apresentam um dos principais resultados deste trabalho, pois mostram que a implementação da vizinhança 3-opt apresentada na seção 4.3.8 foi bastante superior à vizinhanças tradicionais na minimização do atraso total, como troca e inserção de tarefas. Esta superioridade se deu em toda a fronteira eficiente, tanto para o atraso como para o *makespan*. Pode-se argumentar que a utilização de meta-heurísticas mais elaboradas, como busca tabu e algoritmos genéticos, pode tirar proveito da maior eficiência (rapidez em obter um ótimo local) das vizinhanças de troca e inserção. Entretanto, os resultados apresentados no capítulo 6 comparando os resultados com meta-heurísticas desenvolvidas especificamente para cada objetivo mostram que os algoritmos com vizinhança 3-opt sempre obtiveram resultados equivalentes ou superiores aos obtidos pelas meta-heurísticas.

Os testes das seções 5.6 e 5.8 mostram que o impacto da escolha da vizinhança foi muito maior que o das formas de realizar o recomeço e das estratégias de busca testadas. Estes resultados indicam que na utilização de busca local para problemas combinatórios a definição da vizinhança é essencial, e deve-se concentrar esforços para desenvolver uma vizinhança eficiente e eficaz em cada problema abordado.

Nos testes para recomeço fica evidente a supremacia de realizar-se perturbações nas melhores soluções encontradas ao invés das encontradas na última iteração. Tem-se ainda que a perturbação *double bridge* obteve pequena vantagem com relação à troca entre tarefas, possivelmente por ter sido utilizada a vizinhança 3-opt nos testes. Outro resultado claro é que o uso de recomeços aleatórios é inferior às versões com perturbação.

Os testes para estratégia de busca e preprocessamento, seção 5.8, mostram que o uso do preprocessamento descrito na seção 4.2 melhora o desempenho da busca. Já a diferença entre as estratégias com preprocessamento foi pequena, não chegando a resultados conclusivos que mostrem que uma seja claramente superior. Observou-se que o uso do preprocessamento torna a busca muito pouco sensível a escolha da estratégia, facilitando a configuração do algoritmo. Entretanto, fica claro que pode-se priorizar durante a busca um objetivo em detrimento do outro, como nas estratégias T-First e M-First, um resultado útil para aplicações onde o decisor esteja mais interessado em um dos extremos da fronteira eficiente.

Apesar de ser vantajoso utilizar heurísticas construtivas multiobjetivo como conjunto de soluções inicial, esta diferença é pequena e não corresponde ao esforço empregado no desenvolvimento e implementação destas heurísticas, conforme apresentado na seção 5.9. Um dos responsáveis por este resultado é a utilização do preprocessamento, que permite à busca concentrar esforços em soluções de melhor qualidade, reduzindo consideravelmente a importância do conjunto inicial.

A vizinhança com tamanho variável, testada na seção 5.10, obteve bons resultados e indicou que uma combinação da estratégia r-grupos com um valor de folga pequeno na vizinhança 3-opt poderia obter bons resultados. Esta combinação foi testada sendo os resultados apresentados na seção 5.11, de onde se conclui que o melhor algoritmo dentre todos os testes realizados foi a estratégia 5-grupos com folga zero na vizinhança 3-opt. Isto mostra que o desempenho da busca depende fortemente da combinação entre a estratégia de

busca e a vizinhança, pois a utilização da folga mostrara-se vantajosa em conjunto com a estratégia NFirst, na seção 5.4.

Para as instâncias resolvidas por enumeração completa, com $n \leq 14$, o desempenho da busca foi excelente, demandando apenas alguns décimos de segundo para atingir o conjunto eficiente em quase todas as instâncias. Das 2833 soluções existentes nas 405 instâncias, apenas 3 não foram encontradas nos 1.3 segundos disponíveis, conforme apresentado na seção 5.12.

As comparações para cada objetivo efetuadas nas seções 6.1 e 6.2 são importantes por serem utilizados algoritmos de qualidade comprovada na minimização de um dos objetivos. Os resultados comprovam a alta qualidade das soluções encontradas pela busca multiobjetivo proposta e ainda que uma adaptação simples do código implementado consegue gerar soluções ainda melhores para estes objetivos. É importante destacar as dificuldades inerentes deste tipo de comparação, como linguagem e máquina distintos, mas deve-se ressaltar que a busca com recomeços, vizinhança 3-opt, perturbação *double bridge* e implementada numa versão mono-objetivo aproveitando as características de cada objetivo deve apresentar, em geral, resultados melhores que os algoritmos meméticos comparados.

A evolução das soluções apresentada na seção 5.13 mostra que é muito mais simples encontrar boas aproximações do que gerar exatamente as soluções eficientes, especialmente nas maiores instâncias. Isto indica que existe ainda espaço a evoluir no desenvolvimento de algoritmos que encontrem soluções melhores e sejam mais eficientes.

Bibliografia

Aarts E., Lenstra J.K., 1997. "*Local Search in Combinatorial Optimization*", John Wiley and Sons, London.

Allahverdi A., Gupta J.N.D., Aldowaisan A., 1999. "A Review of Scheduling Research Involving Setup Considerations", OMEGA, v. 27, pp. 219-239.

Armentano V. A., Mazzini R., "A Genetic Algorithm for Scheduling on a Single Machine with Set-up Times and Due Dates", Production Planning and Control v. 11, 713-720.

Armentano V.A., Ronconi (1999) D.P., "Tabu Search for Total Tardiness Minimization in Flowshop Scheduling Problems", Computers and Operations Research, 26, 219-235.

Arroyo J. E. C., Armentano V. A., 2000. "Uma heurística de aproximação do conjunto ótimo de Pareto em um Problema de Flowshop com dois objetivos", Anais da II Oficina de Planejamento e Controle da Produção em Sistemas de Manufatura, pp 152-169.

Baykasoglu A., Owen S., Gindy N., 1999. "A Taboo Search Based Approach to Find the Pareto Optimal Set in Multiple Objective Optimization", Engineering Optimization, v. 31, pp. 731-748.

Blazewicz J., Ecker K.H., Pesch E., Schmidt G., Werglarz J., 1996. "*Scheduling Computer and Manufacturing Processes*", Springer.

Buriol L.S., 2000. "Algoritmo memético para o problema do caixeiro viajante assimétrico como parte de um framework para algoritmos evolutivos", tese de mestrado, Faculdade de Engenharia Elétrica e de Computação, UNICAMP.

Czyac P., Janszkiewicz C.P., 1998. "A Pareto Simulated Annealing – A Metaheuristic Technique for Multiple Objective Combinatorial Optimization", Journal of Multi-Criteria Decision Analysis, v. 7, pp. 34-47.

Daniels R.L., Chambers R.J., 1990. "Multiobjective Flowshop Scheduling", Naval Research Logistics, v. 37, pp. 981-995.

Daniels R.L., 1992. "Analytical Evaluation of Multi-Criteria Heuristics", Management Science, v. 38, pp. 501-513.

França P. M., Mendes A., Moscato P., 2001, "A Memetic Algorithm for the Total Tardiness Single Machine Scheduling Problem", European Journal of Operational Research, v. 132, No. 1, pp. 224-242

Garey M. R., Johnson D. S., 1979. "*Computers and Intractability: a guide to the theory of NP-Completeness*", W. H. Freeman, New York.

Glover F., Laguna M., 1997. "*Tabu Search*", Kluwer, Boston.

Goicoechea A., Hansen D. R., Duckstein L., 1982. "*Multiobjective Decision Analysis with Engeneering and Business Applications*", John Wiley and Sons.

Gupta J.N.D., Ramnarayanan, 1996. "Single Facility Scheduling with dual criteria: minimizing maximum tardiness subject to minimum number of tardy jobs", Production Planning and Control, v. 7, pp. 190-196.

Hansen M.P., 1997. "Tabu Search for Multiobjective Optimization: MOTS", Proceedings of MCDM'97, Cape Town, South Africa, January 1997.

Hansen M.P., Jaszkiewicz A., 1998. "Evaluating the Quality of Approximations to the Non-Dominated Set", IMM Technical Report IMM-REP-1998-7.

Hillier F. S., Lieberman G. J., 1988. "*Introdução à Pesquisa Operacional*", Editora Campus / São Paulo.

Ishibuchi H., Murata T., 1998. "A Multi-Objective Genetic Local Search algorithm and its Application to Flowshop Scheduling", IEEE Transactions on Systems, Man and Cybernetics - part C: Applications and Reviews, v. 28, pp. 392-403.

Johnson D.S., McGeoch L.A., 1997. "The Traveling Salesman Problem: A Case Study", em *Local Search in Combinatorial Optimization*, Aarts E., Lenstra J.K., John Wiley and Sons, London, pp. 215-310.

Lawler E.L., Lenstra E.L., Rinnooy Kan A.G.H., Shmoys D.B., 1985. "*The Traveling Salesman Problem*", John Wiley.

Lee Y.H., Bhaskaran K., Pinedo M., 1997. "A Heuristic to Minimize the Total Tardiness with Sequence-Dependent Setups", European Journal of Operational Research, v. 100, pp 464-474.

Lin S., Kernigham B.W., 1973. "An Effective Heuristic Algorithm for the Traveling-Salesman Problem", Operations Research, v. 21, pp. 498-516.

Mendes A.S., 1999. "Algoritmos Meméticos Aplicados aos Problemas de Seqüenciamento em Máquinas", tese de mestrado, Faculdade de Engenharia Elétrica e de Computação, UNICAMP.

Nawaz M., Enscore E.E., Ham I., 1983. "A heuristic Algorithm for the m-Machine, n-Job Flowshop Sequencing Problem", OMEGA, v. 28, pp.91-95.

Or I., 1976. "Traveling Salesman Type Combinatorial Problems and their Relation to the Logistics of Regional Blood Banking", Ph. D. Thesis, Dep. of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL.

Reinelt G., 1991, "TSPLIB - A Traveling Salesman Library". ORSA Journal on Computing v. 3, pp. 376-384

Reinelt G., 1994, "*The Traveling Salesman Problem: Computational Solutions for TSP Applications*". Lecture Notes in Computer Science, v. 840, Springer-Verlag."

Rubin P. A., Ragatz G. L., 1995, "Scheduling in a Sequence Dependent Setup Enviroment With Genetic Search", Computers & Operations Research, v. 22, No. 1, pp. 85-99.

Savelsbergh M.W.P., 1990. "An Efficient Implementation of Local Search Algorithms for Constrained Routing Problems", European Journal of Operational Research, v. 47, pp.75-85.

Serafini P., 1987. "Some considerations about computational complexity for multiobjective combinatorial problems", LNEMS, v. 294, pp. 222-232, Springer Verlag.

Solomon M.M., 1987. "Algorithms for the Vehicle Routing and Scheduling Problems with Time Windows Constraints", Operations Research, v.35, No.2, pp. 254-265.

Solomon M.M., Desroisiers J., 1988. "Time Window Constrained Routing and Scheduling Problems", Transportation Science, v.22, No.1, pp. 1-13.

Tan K.C., Narasimham R., Rubin P.A., Ragatz G.L. 2000. "A Comparison of Four Methods for Minimizing Total Tardiness on a Single Machine with Sequence Dependent Setup Times", OMEGA, v. 28, pp.313-326.

Viana A., Sousa J.P., 2000. "Using Metaheuristics in Multiobjective Resource Constrained Project Scheduling", European Journal of Operational Research, v. 120, pp.359-374.