Times Assíncronos para a Resolução de Problemas de Otimização Combinatória com Múltiplas Funções Objetivo

Rosiane de Freitas Rodrigues

Times Assíncronos para a Resolução de Problemas de Otimização Combinatória com Múltiplas Funções Objetivo

Este exemplar corresponde à redação final da dissertação devidamente corrigida e defendida pela Srta. Rosiane de Freitas Rodrigues e aprovada pela comissão julgadora.

Campinas, 05 de julho de 1996.

Prof. Dr. Pedro Sérgio de Souza

Dissertação submetida ao Instituto de Computação da Universidade Estadual de Campinas como parte dos requisitos necessários para a obtenção do Título de MESTRE em Ciência da Computação.

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Rodrigues, Rosiane de Freitas

R618 Times Assíncronos para a Resolução de Problemas de Otimização Combinatória com Múltiplas Funções Objetivo / Rosiane de Freitas Rodrigues -- Campinas, [S.P. :s.n.], 1996.

Orientador : Pedro Sérgio de Souza

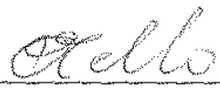
Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Otimização combinatória. 2. Otimização matemática. 3. Problema do caixeiro viajante. 4. Algoritmos. I. Souza, Pedro Sérgio de. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

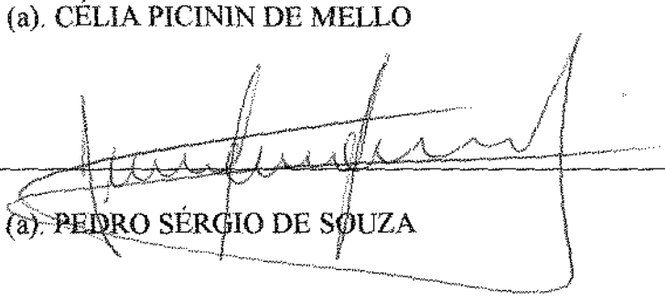
Tese de Mestrado defendida e aprovada em 05 de Julho de 1996

pela Banca Examinadora composta pelos Profs. Drs.

Prof (a). Dr (a). PAULO MORELATO FRANÇA



Prof (a). Dr (a). CÉLIA PICININ DE MELLO



Prof (a). Dr (a). PEDRO SÉRGIO DE SOUZA

Times Assíncronos para a Resolução de Problemas de Otimização Combinatória com Múltiplas Funções Objetivo [1]

Aplicação ao Problema do Caixeiro Viajante com Múltiplas Distâncias

Rosiane de Freitas Rodrigues [2]

Instituto de Computação
Universidade Estadual de Campinas
IC - UNICAMP

Banca Examinadora:

- **Pedro Sérgio de Souza (Orientador)** [3]
- **Paulo Morelato França** [4]
- **Célia Piccinin de Mello** [5]
- **João Carlos Setubal** [5]

---

1 Dissertação apresentada ao Instituto de Computação da UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

2 A autora é Bacharela em Processamento de Dados pela Universidade Federal do Amazonas (UFAM).

3 Professor do Instituto de Computação - IC - UNICAMP / Consultor do *IBM Consulting Group / AMT*.

4 Professor do Departamento de Engenharia de Sistemas - FEE - UNICAMP.

5 Professor(a) do Instituto de Computação - IC - UNICAMP.

A Deus.

Aos meus pais.

Ao meu irmão, e anjo da guarda, Ronaldo.

in memorium

Agradecimentos

Que sintam-se citados todos que de alguma forma contribuíram para criar em mim uma expressão alegre ou agradecida, que dividiram momentos, meus ou seus, de preocupação, de tristeza, de alívio ou de felicidade.

Aos meus pais, Dalva e João, por seus exemplos de vida, me ensinando a ter determinação na busca dos meus ideais e força para superar os problemas. Por suas presenças em minha vida, com muita dedicação, compreensão e amor. Vocês foram essenciais nesta conquista.

Aos meus irmãos e sobrinhos, pelo enorme carinho que nos une, esteja onde estivermos.

Um especial agradecimento ao meu orientador, Prof. Pedro Sérgio de Souza. Sua coerência e extrema dedicação ao que faz, fizeram dele um exemplo de vida para mim. Sua orientação, com críticas e sugestões indispensáveis, sua paciência e compreensão, seu apoio e incentivo em todos os momentos, foram decisivos para o êxito de meus objetivos. Muito aprendi e levo comigo este aprendizado de vida para os próximos caminhos que percorrerei.

Aos amigos que tive a oportunidade de conhecer. Em particular, à Ana Maria Monteiro, por sua amizade, companheirismo e nossas prazerosas conversas. Ao Edson, Victor, Cris (família Melo Barros), Eduardo e Wesley, pelos momentos que pude compartilhar com vocês.

Aos amigos de longa data, Amanda, Gilbert, Herivan, Marcia, Cilene e Edson.

Aos professores Célia Piccinin de Mello, Paulo Morelato França e Tomasz Kowaltowski, por seus relevantes comentários e avaliações, como membros da banca julgadora, no início e/ou término do meu Mestrado.

Aos professores Ademar Teixeira, Crediné Silva e José Alberto, por me apoiarem no início desta jornada.

À Universidade Federal do Amazonas, à CAPES e à FAPESP, pelo apoio recebido.

"A vida da maioria das pessoas é u ma contínua escolha, um compromisso permanente entre o ideal e o possível."

Bertrand Russell.

Resumo

Times Assíncronos (do inglês *Asynchronous Teams* ou *A-Teams*) constituem uma abordagem multi-algorítmica para a resolução aproximada de problemas, cujo princípio básico é a cooperação sinérgica entre um conjunto de algoritmos, que se comunicam assincronamente através de memórias compartilhadas, propiciando soluções de melhor qualidade do que as geradas pelos mesmos algoritmos quando executados isoladamente. Este método tem sido aplicado com sucesso em problemas combinatórios com uma única função objetivo.

O presente trabalho apresenta Times Assíncronos como sendo adequado à resolução de problemas de otimização combinatória com múltiplas funções objetivo. Diferentes estratégias para a manipulação de soluções multidimensionais foram desenvolvidas, tornando possível a rápida obtenção de soluções próximas ou mesmo pertencentes ao Pareto Ótimo do problema. Em especial, foi desenvolvida uma estrutura de manipulação de soluções multidimensionais que possibilita a consideração simultânea de todos os objetivos envolvidos para a geração de soluções.

É proposto um novo problema NP-difícil como uma generalização do clássico Problema do Caixeiro Viajante (do inglês *Traveling Salesman Problema* ou *TSP*), onde ao invés de apenas uma matriz de distância existem múltiplas matrizes, sendo intitulado **Problema do Caixeiro Viajante com Múltiplas Distâncias** (do inglês *Multi-Distance Traveling Salesman Problem* ou *MDTSP)* e ao qual foi aplicado o método de Times Assíncronos.

Os testes computacionais foram realizados de forma concorrente e paralela, obtendo-se conjuntos de soluções não-dominadas bem distribuídas dentro de uma ampla faixa de valores fornecidos pelas funções objetivo envolvidas, para todas as instâncias, mesmo envolvendo várias dimensões. Isto demonstra que os melhores conjuntos de soluções não-dominadas gerados pelos A-Teams foram numerosos e contiveram soluções significativamente distintas entre si, abrangendo todo o espectro desejado. Para as menores instâncias foi possível constatar que o melhor conjunto de soluções não-dominadas obtido fora o próprio Pareto Ótimo. Ainda, foram desenvolvidos algoritmos para o novo problema que incorporam conceitos adequados a problemas multiobjetivos.

Abstract

Asynchronous Teams or **A-Teams** constitute a multi-algorithm approach for approximated problem solving, whose basic principle is the sinergic cooperation among a set of algorithms that communicate asynchronously through shared memories, providing solutions of better quality than those generated by the same algorithms when executed separately. This method has been sucessfully applied to Combinatorial Optimization Problems with a single objective function.

This work presents Asynchronous Teams as an adequated method to solving *Combinatorial Optimization Problems with multiple objective functions*. Different strategies to the manipulation of multi-dimensional solutions were developed, allowing it possible to obtain near-optimal or Pareto Optimal solutions quickly. In special, was developed a structure for multidimensional solution manipulation that allowing it possible the simultanea consideration of all objetives involved to the solution generation.

It is proposed a new NP-hard problem as a generalization of classic Traveling Salesman Problem or TSP, which, instead of only one distance matrix, has various matrices. It has been entitled of **Multi-Distance Traveling Salesman Problem** or **MDTSP** and to which was applied the Asynchronous Teams method.

The implementation tests were accomplished in a concurrent and parallel way, obtaining set of non-dominated solutions well-distributed on a wide range of values provided by the objective functions involved, over the tested instances. This demonstrates that the best sets of non-dominated solutions obtained are numerous and contain solutions significantly distinct among them. To the small instances was possible to show that the best set of non-dominated solutions generated was the Pareto Optimal. Algorithms have been developed for the new problem with the incorporation of compromisse decision and dominance concepts. Still, were developed algorithms to the new problem that incorpore adequated concepts to multiobjective problems.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

"Teus olhos devem olhar à frente, para que a tua vista preceda os teus passos."
Salomão.

Estamos vivenciando o fim do segundo milênio e mais do que nunca existe a necessidade, inerente ao progresso, de gerenciar de maneira eficiente a gama de novos conhecimentos produzida a todo instante. É a era da informação. A contínua busca pelo entendimento e aproveitamento de informações vem estimulando o desenvolvimento de máquinas cada vez mais sofisticadas, onde a qualidade e a eficiência na execução de tarefas revelam-se como premissas básicas para a modernização.

A globalização da economia é agora uma realidade e acirra a competitividade em todos os níveis da sociedade. A otimização de recursos e serviços passa a ser fator primordial nesta disputa, onde a arma mais poderosa é o computador. Eis a era da computação. A nossa visão de mundo pode ser mais refinada e a conseqüência disto é que problemas cada vez mais complexos são descobertos e resolvidos.

Como uma das formas de solucionar alguns destes problemas complexos surgiu a Otimização Combinatória. A partir de então, inúmeros métodos foram sendo propostos visando a resolução destes problemas, onde a busca pela melhor solução possível - a solução ótima - é a tônica da questão. Tais problemas, geralmente, possuem pelo menos uma formulação em Programação Linear Inteira, onde todas as variáveis envolvidas têm valores inteiros. Esta abordagem é particularmente interessante quando se tem um certo conhecimento sobre a estrutura poliedral do problema e, através de sua relaxação, se consegue a geração de soluções factíveis eficientemente até a obtenção do ótimo. Acontece que um grande número de problemas são tão difíceis de se resolver e/ou demandam tanto tempo de processamento, mesmo com os mais rápidos computadores,

que o desenvolvimento de métodos que forneçam soluções de boa qualidade (soluções próximas do ótimo, quando não o ótimo) num período relativamente curto de tempo, se tornou uma necessidade freqüente: são os métodos aproximados para a resolução de problemas ou, simplesmente, heurísticas.

Souza [Sou93] apresentou um novo método para a resolução de problemas, onde a utilização conjunta de um grupo de algoritmos heurísticos, que interagem independentemente uns dos outros, propicia a obtenção de soluções de boa qualidade muito rapidamente. Tal método foi denominado de Times Assíncronos (do inglês *Asynchronous Teams* ou *A-Teams*). Um dos pontos mais destacáveis neste método é que as soluções geradas são de melhor qualidade do que as geradas pelos algoritmos do grupo quando executados isoladamente. A-Teams vêm sendo aplicado com sucesso em Problemas Combinatórios [ST91, TS92, Mur92, Sou93, Pei95, Cav95, Lon95, Cam95].

Paralelamente a todo este contexto, cada vez mais se busca uma modelagem mais fiel dos problemas do mundo real, de tal forma que as distorções entre o problema e seu modelo representativo sejam cada vez menores. Em situações reais, torna-se necessário lidar simultaneamente com a influência de fatores extremamente conflitantes e o ideal é o mapeamento de tais fatores em objetivos distintos. Muitas vezes, os objetivos não são nem mesmo mensuráveis, inexistindo uma relação quantitativa entre os mesmos e apenas qualitativamente se pode compará-los. Acontece também de não ser possível definir quais são os objetivos a serem atingidos, dado o grau de incerteza e imprecisão envolvidos. Tais questões foram por muito tempo ignoradas, visto que a idéia de uma única função objetivo não representa tais problemas. Assim, surgiu a Otimização Multiobjetiva, dando suporte teórico à resolução de problemas que naturalmente possuem mais do que um objetivo a ser atingido. Desta forma, a busca da solução ótima dá lugar à busca de soluções de melhor compromisso possível para todos os objetivos envolvidos - o Pareto Ótimo -, tornando a tarefa ainda mais árdua do que a busca por um ótimo.

O que geralmente é feito para contornar o fato de se lidar com múltiplos objetivos é a encapsulação dos mesmos num só, seja através de uma combinação linear, transformando-os numa função objetivo escalar, seja através da priorização de um deles, transformando os objetivos restantes em restrições do problema; recaindo em ambos os casos, num problema monobjetivo.

Apesar de ser uma área recente onde os conceitos e definições ainda não estão bem definidos, existe um vasto material bibliográfico sobre a resolução de problemas multiobjetivos onde os mesmos são tratados como problemas monobjetivos e a maioria quase absoluta envolve aqueles onde todas as variáveis são contínuas ou, então, os problemas são de pequeno porte [Ste86, KMW92, Kar95].

Mas, e quando se deseja considerar igual e simultaneamente todos os objetivos envolvidos? O que se encontra a respeito na literatura?

Pouco se encontra disponível sobre a resolução de problemas com múltiplos objetivos para o caso em que *todos* os objetivos são igualmente analisados e as soluções obtidas satisfaçam todos eles simultaneamente. No caso do tipo de problema abordado

neste trabalho, onde todas as variáveis envolvidas são inteiras, apenas os trabalhos de Murthy [Mur92] e Karainova *et al* [Kar95] mostraram alguma relevância prática.

A escassez de métodos para Problemas de Otimização Combinatória Multiobjetivos, e dada a importância dos mesmos, motivou o desenvolvimento deste trabalho.

1.1. Principais Objetivos

O tema central desta dissertação é mostrar a adequabilidade do método multi-algorítmico de **Times Assíncronos** (do inglês ***Asynchronous Teams*** ou ***A-Teams***) como sendo apropriado para a resolução de Problemas de Otimização Combinatória Multiobjetivos. A-Teams têm como princípio básico a cooperação assíncrona entre um conjunto de algoritmos, cuja eficiência já pôde ser comprovada para Problemas de Otimização Combinatória Monobjetivos em trabalhos recentemente desenvolvidos [Sou93, Pei95, Cav95, Cam95]. É apresentado, portanto, um projeto de A-Teams para Problemas Multiobjetivos, onde são abordados e analisados todos os parâmetros de interesse envolvidos.

O segundo objetivo é apresentar estruturas de manipulação de soluções multidimensionais para A-Teams mais adequadas a Problemas Multiobjetivos do que as utilizadas por Murthy [Mur92].

Um terceiro objetivo é a proposição de uma generalização de um problema NP-difícil clássico, o Problema do Caixeiro Viajante (do inglês *Traveling Salesman Problem* ou TSP), onde, ao invés de apenas uma matriz de distância, possui múltiplas matrizes. Tal problema recebe a denominação de **Problema do Caixeiro Viajante com Múltiplas Distâncias** (***Multi-Distance Traveling Salesman Problem*** ou **MDTSP**). São apresentados algoritmos para a resolução deste problema (inexistentes até então, dado que o problema está sendo proposto neste trabalho), que compõem os grupos de algoritmos dos A-Teams desenvolvidos.

Como conseqüência do estudo realizado é apresentada uma revisão bibliográfica abrangendo as aplicações de A-Teams desenvolvidas até o momento, explicitando o problema abordado. Na mesma linha, são apresentados os principais métodos para a resolução de Problemas Multiobjetivos e de Problemas Combinatórios.

1.2. Domínio do Trabalho

Este trabalho envolve principalmente as áreas de pesquisa sobre Otimização Combinatória e Otimização Multiobjetiva. Nesta subseção serão comentadas as principais características que delineiam cada uma destas áreas.

Um **Problema de Otimização Monobjetivo** (com uma única função objetivo) pode ser formulado da seguinte forma:

$$\begin{aligned} &min\ f(x) \\ &s.a.\ x \in S \end{aligned} \qquad \text{Eq. 1. 1}$$

onde $f(x)$ é a *única* função objetivo e S é a região de soluções factíveis do problema ($S \subset R^n$). O propósito de um problema de otimização é encontrar o ponto pertencente a S que possui o menor valor possível segundo a função objetivo $f(x)$, ou seja, a **solução ótima** do problema. Eventualmente, podem existir mais do que uma solução ótima para o problema (soluções diferentes, mas com o mesmo valor segundo a função objetivo utilizada).

Dependendo da definição de $f(x)$ e da definição de S os problemas são tratados de maneiras distintas. Desta forma:

- Se $f(x)$ for linear e S for definida por restrições lineares, tem-se um problema de programação linear monobjetivo ou, somente, Programação Linear.
- Se $f(x)$ for linear e S for definida por restrições lineares, com a restrição adicional de que as coordenadas de cada ponto em S devem ser inteiras, tem-se um problema de programação linear inteira monobjetivo ou, somente, Programação Inteira.
- Se $f(x)$ ou qualquer uma das restrições que definem S for não-linear, tem-se um problema de programação não-linear monobjetivo ou, somente, Programação Não-Linear.

Da mesma forma, um **Problema de Otimização Multiobjetivo** (com múltiplas funções objetivo) pode ser modelado da seguinte forma:

$$\begin{aligned} &min \ f_1(x) \\ &min \ f_2(x) \\ &\quad \vdots \\ &min \ f_k(x) \\ &s.a. \ x \in S \end{aligned} \qquad \text{Eq. 1.2}$$

onde $f_1(x), f_2(x), \ldots, f_k(x)$ são as k funções objetivo do problema e S ($S \subset R^n$), a região de soluções factíveis. Tal como para o problema monobjetivo, o multiobjetivo também pode ser linear, inteiro ou não-linear. O propósito, neste caso, é a busca pelos pontos k-dimensionais pertencentes a S que possuem os menores valores possíveis segundo *todas* as k funções objetivo, ou seja, o **Pareto Ótimo** do problema.

1.3. Organização da Dissertação

Esta dissertação está organizada em oito capítulos, distribuídos da seguinte maneira:

Neste primeiro capítulo, foram apresentados os principais objetivos deste trabalho, as principais áreas de pesquisa relacionadas com o mesmo e o que motivou seu desenvolvimento. O método proposto como adequado à resolução de Problemas de Otimização Combinatória Multiobjetivos, o de Times Assíncronos (*Asynchronous Teams* ou A-Teams), bem como o problema proposto e abordado por tal método, o Problema do Caixeiro Viajante com Múltiplas Distâncias (*Multi-Distance Traveling Salesman Problem* ou MDTSP), foram sucintamente descritos.

O segundo e o terceiro capítulo apresentam uma revisão bibliográfica sobre o domínio desta dissertação, onde a teoria envolvida com os problemas combinatórios e

multiobjetivos é apresentada, bem como os principais métodos para a resolução dos mesmos. Assim, no segundo capítulo são abordados os Problemas de Otimização Combinatória e no terceiro capítulo, os Problemas Multiobjetivos.

A partir do quarto capítulo são apresentadas as contribuições desta dissertação. O capítulo quatro apresenta o problema proposto, o MDTSP, bem como algoritmos para a sua resolução. O quinto capítulo aborda o método de Times Assíncronos, onde o projeto de tal método para problemas multiobjetivos é definido. O sexto capítulo apresenta a aplicação de A-Teams ao MDTSP, onde são definidas as configurações de A-Teams e todas as estratégias usadas para a resolução do problema e os demais detalhes de implementação relevantes. O sétimo capítulo apresenta os resultados decorrentes da aplicação sugerida. Alguns limites inferiores foram também calculados.

No oitavo, e último, capítulo são apresentadas as conclusões e as principais contribuições deste trabalho. Algumas sugestões para possíveis trabalhos futuros são, também, citadas.

No Apêndice A podem ser encontrados as matrizes de distâncias geradas randomicamente e seus respectivos limites inferiores. No Apêndice B, encontram-se as matrizes de distância e os limites inferiores relacionados à instância real de 24-cidades.

Capítulo 2

Problemas de Otimização Combinatória

" É difícil dizer o que é impossível, pois a fantasia de ontem é a esperança de hoje e a realidade de amanhã..."

Robert Goddart.

Os **Problemas de Otimização Combinatória** são caracterizados pela presença de um enorme espaço *discreto* de busca, onde, dentre inúmeras alternativas, se deseja maximizar ou minimizar um certo valor, geralmente fornecido pela única função objetivo do problema.

Este capítulo aborda tais problemas e, por conseguinte, seus principais métodos de resolução, divididos em duas grandes classes: **métodos exatos** e **métodos aproximados**. Serão citados tanto os métodos mais tradicionais, quanto algum dos mais recentes na resolução de problemas de otimização combinatória. Serão explicitados, também, a complexidade intrínseca a tais problemas, algumas aplicações práticas, bem como as áreas de pesquisa relacionadas com os mesmos.

O estudo de tal classe de problemas foi de fundamental importância para este trabalho, visto que o problema abordado é, também, um Problema de Otimização Combinatória (ver Cap.4).

2.1. Introdução

Os Problemas de Otimização Combinatória constituem uma das classes de problemas mais estudadas no momento, devido a inerente complexidade que os envolve, atraindo cada vez mais a atenção de inúmeros pesquisadores ávidos em obter melhores e melhores resultados, e aliado ao fato de existir inúmeros problemas práticos que podem ser modelados como sendo de otimização combinatória.

Mas, o que caracteriza um problema combinatório?

Existe um ramo da matemática denominado Combinatória [Liu68]. Esta área desempenha um importante papel na Ciência da Computação. As propriedades de estruturas combinatórias, como grafos, proporcionam o desenvolvimento de eficientes algoritmos para inúmeros modelos de problemas teóricos e, conseqüentemente, para aplicações práticas em computação.

Combinatória pode ser classificada em três grandes classes: enumerativa, existencial e construtiva [PTW83]. **Combinatória Enumerativa** lida com a contagem de objetos combinatórios, ou seja, deseja-se enumerar todas as possíveis configurações que satisfaçam um certo critério. **Combinatória Existencial** estuda a existência ou não de alguma configuração combinatória para um problema onde, neste caso, não se deseja enumerar todas as alternativas e sim, encontrar *qualquer* configuração que satisfaça o critério determinado (provar teoricamente a existência de uma configuração). Por fim, **Combinatória Construtiva** aborda a busca por configurações específicas e, neste caso, não há interesse em saber quantas configurações existem que satisfaçam um certo critério, o que se deseja é encontrar uma certa configuração (encontrar um exemplo, sem necessidade de prova teórica).

E quanto aos problemas de otimização? Onde os mesmos se enquadram?

Otimização é o processo de fazer algo (tal como um projeto, sistema ou uma tomada de decisão) tão eficientemente quanto possível. Em Matemática, significa maximizar ou minimizar o valor de uma função. Tradicionalmente, resolver um problema de otimização consiste em encontrar a **solução ótima** dentre um número finito ou infinitamente contável de soluções alternativas [PS82]. Então:

Definição 2.1: Uma instância de um problema de otimização é um par (S,c), onde S é a região factível e c é a função que se deseja maximizar ou minimizar, tal que:

$$c: S \rightarrow \Re \qquad \text{Eq. 2. 1}$$

Definição 2.2: Um problema de otimização consiste em encontrar um $x \in S$, tal que:

$$c(x) \leq c(y) \quad , \quad \forall \quad y \in S \qquad \text{Eq. 2. 2}$$

Se o conjunto S for discreto, tem-se um problema de otimização combinatória. Sendo assim, Otimização Combinatória pode ser definida como [NW88]:

Definição 2.3: Otimização Combinatória trata problemas de maximização ou minimização de uma função (ou funções) de variáveis, sujeita à restrições de igualdade ou desigualdade e restrições de integralização de algumas ou todas as variáveis.

Um Problema de Otimização Combinatória se classifica no ramo de Combinatória Construtiva onde se deseja *construir* a *melhor* solução possível, ou seja, a solução ótima para o problema.

2.2. Complexidade

Todos os Problemas de Otimização Combinatória possuem sua versão de decisão (Combinatória Existencial) [Cor91, Man89, GJ79], onde suas soluções consistem numa resposta positiva (sim) ou negativa (não). Ao invés da busca pela melhor solução, neste caso se deseja saber primeiro se *existe* alguma solução.

Nesta seção, considera-se que todos os problemas mencionados estejam em suas versões de decisão.

Os Problemas de Otimização Combinatória podem ser classificados segundo a **Teoria da NP-Completude** [GJ79]. Esta teoria faz parte de uma mais genérica, conhecida como **Complexidade Computacional**. Na Teoria da NP-Completude, todo problema é tratado como sendo um conjunto de parâmetros (definição das instâncias) e um conjunto de propriedades (restrições do problema a serem satisfeitas). Sendo assim, entre instâncias de um mesmo problema, as únicas variações estão nos conjuntos de parâmetros. A teoria da complexidade considera o tamanho das instâncias como sendo a quantidade de *bits* necessária para representá-las em computadores digitais. A questão é:

Qual o número de computações necessárias para se obter a solução ótima?

Conforme este número de computações necessárias, os problemas podem ser classificados em três grandes classes (fig. 2.1):

P (***Polinomial time***) → Problemas onde o esforço computacional (número de computações) de pelo menos um algoritmo conhecido cresce polinomialmente em função do tamanho da instância. São conhecidos como problemas ***tratáveis*** e que possuem ***algoritmos eficientes*** para suas resoluções. Exemplos:

- **Problema da Ordenação de Elementos;**
- **Problema da Árvore de Espalhamento Mínimo;**
- **Problema do Fluxo em Redes.**

NP (***Nondeterministic Polinomial time***) → Problemas onde o número de computações do melhor algoritmo conhecido cresce exponencialmente em função do tamanho da instância, não existindo garantia da não-existência de algoritmos melhores e, dada uma solução para o problema, pode-se verificar em tempo polinomial se a solução satisfaz o problema de decisão positivamente ou negativamente. Exemplos:

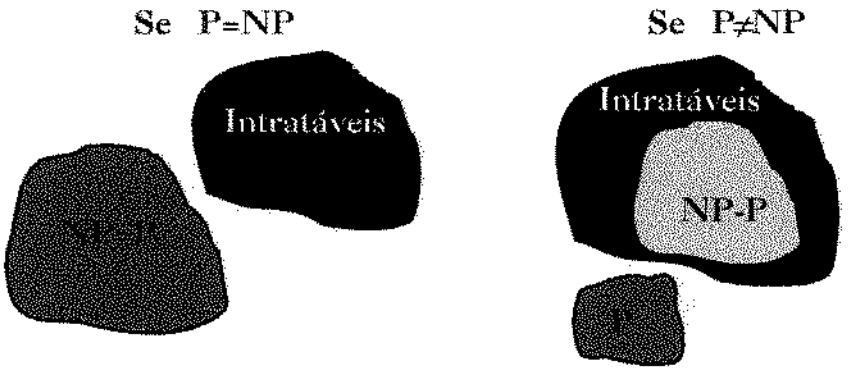
- **Todos os problemas P e NP-completos** (a classe dos problemas NP-completos será descrita a seguir).

Intratáveis → Problemas onde o número de computações do melhor algoritmo conhecido cresce exponencialmente, como os NP, em função do tamanho da instância. Agora, para os problemas intratáveis existe a garantia da não existência de algoritmos melhores. Exemplo:

- **Torre de Hanoi** [Kru84].

Parece ser razoável acreditar que os algoritmos polinomialmente não-deterministas sejam muito mais intrincados que os algoritmos deterministas. *Isto é verdade?* Uma maneira de provar que isto é verdade seria apresentar um problema NP que não fosse P. Até agora isto não foi possível. Em contrapartida, se o objetivo fosse provar que as duas classes são iguais (P=NP), teria que ser demonstrado que todo problema que pertence a NP pode ser resolvido por um algoritmo determinista de tempo polinomial. Também nesta questão, nada foi provado até agora (e muitos acreditam que não seja verdade). O problema de determinar o relacionamento entre as classes P e NP é conhecido como **Problema P=NP** (*P=NP Problem*) [GJ79, Man89, Cor91].

Figura 2. 1: Inter-relacionamento entre as três grandes classes de problemas definidas na Teoria da Complexidade: P, NP e Intratáveis.



Serão definidas, a seguir, duas classes adicionais que contém inúmeros e importantes problemas (todos equivalentes entre si) que, até onde se sabe, não pertencem a P, mas possuem a *dificuldade* característica dos problemas pertencentes a classe NP. Então:

NP-Difíceis → Um problema é dito NP-difícil se todos os problemas da classe NP forem polinomialmente redutíveis a ele. Isto significa que resolvendo um problema NP-difícil em tempo polinomial, todos os problemas da classe NP também poderão ser resolvidos em tempo polinomial. Exemplos:

- **Todos os Problemas que possuem uma versão de Otimização, incluindo todos os NP-Completos.**

NP-Completos → Englobam todos os problemas NP que também são NP-difíceis.

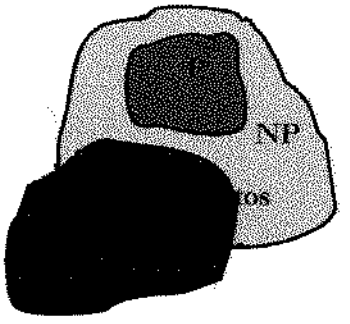
Isto significa que, se for demonstrado que algum problema NP-difícil pertença a P, então P=NP.

Cook, em 1971, deu início a Teoria da NP-Completude através do famoso **teorema de Cook**, onde ele provou que todos os problemas NP são polinomialmente redutíveis ao Problema da Satisfatibilidade (*SAT Problem*), estabelecendo o primeiro problema NP-completo [Coo71]. Desde então, vários problemas foram identificados como sendo NP-completos, como por exemplo:

- **Problema do Caixeiro Viajante** (*Traveling Salesman Problem* - TSP);
- **Caminho e Circuito Hamiltoniano**;
- **Problema da Mochila**;
- **Problema do Conjunto Dominante em Grafos**;
- **Problema da Clique Máxima de um Grafo.**

Vale ressaltar que tais problemas são NP-completos na sua versão de decisão. Na versão de otimização, os problemas acima fazem parte da classe NP-Difícil.

Figura 2. 2: Inter-relacionamento entre as classes de problemas P, NP, NP-difíceis e NP-completos.



2.3. Classificação

Os Problemas de Otimização Combinatória consistem em atribuir, estabelecer seqüências ou selecionar objetos, atingindo um objetivo desejado e, ao mesmo tempo, satisfazendo as restrições. Com base nesta definição, tais problemas podem ser classificados da seguinte maneira [Mül81]:

Problemas de Atribuição

Dois ou mais conjuntos discretos de objetos estão disponíveis e o problema consiste em encontrar *n-uplas* que satisfaçam as restrições, otimizando (maximizando ou minimizando) alguma função objetivo. Exemplos:

- **Problema de Coloração de Grafos;**
- **Problema do Fluxo Máximo em Redes.**

Problemas de Seqüências

Um conjunto discreto de objetos está disponível e deve ser ordenado seqüencialmente de forma a satisfazer as restrições envolvidas, otimizando alguma função objetivo. Exemplos:

- **Problema do Caixeiro Viajante** (*Traveling Salesman Problem* - TSP);
- **Problema do Caminho Mínimo.**

Problemas de Seleção

Um subconjunto deve ser formado a partir de um conjunto discreto de objetos, satisfazendo as restrições e otimizando alguma função objetivo. Exemplos:

- **Problema da Mochila;**
- **Árvore de Espalhamento Mínimo.**

Problemas de Composição

São problemas que não se enquadram em nenhuma das três classes anteriores, mas que, na verdade, é composto por características de duas ou das três classes simultaneamente. Exemplos:

- **Problema da Mochila** (*Knapsack Problem*);
- **Árvore de Espalhamento Mínimo.**

A seguir, serão descritas as principais técnicas para a resolução de Problemas de Otimização Combinatória, classificadas dentro de duas grandes abordagens: a abordagem exata e a abordagem aproximada.

2.4. Métodos para Resolução de Problemas de Otimização Combinatória

Para a maioria dos Problemas de Otimização Combinatória, as formulações mais conhecidas e utilizadas são as feitas em Programação Inteira, que é o caso onde todas as variáveis do problema possuem valores inteiros e, mais especificamente, muitos deles são modelados em Programação Inteira 0-1, onde as variáveis envolvidas possuem os valores discretos 0 ou 1. Existem duas correntes de pensamento dividindo os grandes pesquisadores desta área. Os que seguem Grötschel e Padberg, que acreditam que os Problemas de Programação Inteira sejam tão difíceis que não há nenhuma esperança em resolvê-los genericamente e a única chance estaria no desenvolvimento de métodos específicos para determinada classe de problemas. Os outros pesquisadores, contrariamente, acreditam que num futuro não muito distante será realidade algoritmos genéricos o suficiente para englobar classes distintas de problemas [Cer96].

O importante é que novos e profundos avanços vêm sendo realizados, fazendo com que grandes instâncias reais de problemas clássicos NP-difíceis sejam resolvidas com sucesso. Um exemplo é o caso do TSP, simétrico e assimétrico, onde importantes resultados vêm sendo obtidos através da abordagem de *branch-and-cut*, como a resolução de um TSP com 3000-cidades por Padberg e Rinaldi [PR91] e a obtenção de *boas* trajetórias para instâncias de características especiais de até 10^5-cidades, com a garantia de comprimentos a menos de 2% do ótimo, como reporta Johnson [Joh90].

Será a extinção dos métodos aproximados para a resolução do TSP e de outros problemas NP-difíceis?

Não, a utilização de métodos aproximados ainda é extremamente necessária e uma justificativa para isto é que o tamanho da instância não é tão relevante assim, se considerarmos a existência de instâncias do TSP com menos de 100-cidades e algumas centenas de variáveis, cuja solução ainda não foi encontrada. Além disto, para aplicações práticas o mais importante, no momento, é a obtenção de boas soluções no menor período de tempo possível.

O que se pode concluir disto tudo é quão intrincados são tais problemas e quão interessante é o estudo e o desenvolvimento de novos métodos que se propõe a resolvê-los, seja tentando a garantia da solução ótima, seja gerando boas soluções para os mesmos. Nas subseções seguintes, alguns dos principais algoritmos encontrados na literatura, dentro da abordagem exata e da aproximada, serão brevemente descritos.

2.4.1.Métodos Exatos

Cada vez mais torna-se possível resolver instâncias de problemas que até muito pouco tempo atrás eram consideradas intratáveis na prática. Isto se deve aos grandes avanços tecnológicos, resultando em máquinas mais poderosas e, também, no desenvolvimento de novos algoritmos. Nesta seção, serão descritos os principais métodos exatos, desde os já tradicionais (e, de certa forma, ultrapassados) até os mais recentes em programação inteira.

Antes de abordarmos tais métodos, vale ressaltar que os algoritmos exatos não possuem limite inferior em tempo polinomial para o pior caso, o que significa que eles podem levar um tempo excessivamente longo de execução, dependendo da instância do problema a ser resolvida. Além disso, implementações de métodos exatos resultam, geralmente, numa codificação muito mais complexa do que as implementações de métodos aproximados como, por exemplo, os algoritmos de *branch-and-cut* [JRR94, NW88].

2.4.1.1.Cortes no Plano

Os Algoritmos para Corte no Plano (*Cutting Plane Algorithms*) tradicionais, geralmente, iniciam com a resolução de uma relaxação linear do Problema de Programação Inteira e, em seguida, novas desigualdades (cortes no plano) vão sendo adicionadas ao problema relaxado (com menos restrições), de tal forma que nenhuma solução ótima inteira seja *cortada* (eliminada), ou seja, elimina-se apenas as soluções fracionárias da relaxação inicial.

Gomory [Gom60] propôs vários algoritmos, os quais envolviam a geração automática de cortes no plano (adição de desigualdades) com a garantia de que nenhuma solução inteira seria eliminada. Gomory teve um papel fundamental na teoria da programação inteira, mas, seus cortes nunca foram muito utilizados na prática. Foi o primeiro a utilizar cortes no plano como um procedimento de resolução de problemas de programação inteira e mista.

O ponto crucial de tais algoritmos é desenvolver cortes de tal forma que, depois de um número finito de passos, se garanta que a solução inteira obtida seja a solução ótima do problema.

2.4.1.2.*Branch-and-Bound*

O *branch-and-bound* é um método do tipo enumerativo, pois resolve o problema enumerando um certo número de soluções factíveis, que deseja-se que seja tão pequeno quanto possível [NW88].

A maioria das abordagens de *branch-and-bound* iniciam pela resolução de uma relaxação do problema inteiro: a idéia é encontrar limites inferiores e superiores (através de heurísticas, por exemplo) próximos da solução ótima, para assim reduzir o número de passos enumerativos. Em seguida, é aplicada uma técnica de divisão-e-conquista com o intuito de encontrar a solução ótima do problema, ou seja, a região definida pela relaxação linear é dividida em pequenos subconjuntos, que é chamado de fase enumerativa, pela exploração da árvore de pesquisa formada.

A divisão do problema corresponde ao ***branching*** e a verificação da viabilidade, ou não, de um subproblema corresponde ao ***bounding***.

O *Branch-and-Bound* termina quando todos os subproblemas tiverem sido resolvidos. A escolha de bons algoritmos para o cálculo dos limites inferiores e superiores é crucial, dado que pode-se eliminar grande parte do conjunto de soluções factíveis sem precisar enumerá-las.

Land e Doig [LD60] foram os precursores na aplicação deste método para problemas de programação inteira e Balas [Bal65], especificamente para problemas de programação inteira 0-1.

A abordagem de *branch-and-bound* foi durante muito tempo a preferida pelos interessados na resolução de problemas de programação inteira.

2.4.1.3.Programação Dinâmica

A técnica de Programação Dinâmica (PD) resolve problemas pela combinação das soluções dos sub-problemas derivados do problema original [Cor91]. Tal como o método de divisão e conquista, a PD divide o problema em subconjuntos e os resolve recursivamente. A diferença é que na PD os subproblemas não são independentes uns dos outros, ou seja, os subproblemas compartilham sub-subproblemas. Cada sub-subproblema é resolvido uma única vez e então sua solução é armazenada numa tabela, de forma que se um problema idêntico for encontrado sua solução já está disponível, não sendo necessário resolvê-lo novamente.

A origem da PD se deu no modelagem de processos de decisão seqüenciais. O problema inicial é dividido em estágios onde são efetuadas as decisões e uma solução vai sendo construída ascendentemente (*bottom-up*) e, sempre que uma solução não for ótima, retorna-se a estágios anteriores efetuando-se outras decisões [AHU74].

Geralmente, este método é utilizado para a resolução de problemas de otimização. Qualquer problema de otimização pode ser modelado por PD [GL93]. Sempre existe uma solução recursiva para programas dinâmicos mas, em geral, não é eficiente. O modelo de PD pode ser usado para projetar algoritmos rápidos em casos onde o número de estados pode ser controlado.

2.4.1.4.Cortes Combinatórios no Plano

Os Cortes Combinatórios no Plano (***Combinatorial Cutting Planes***), também conhecidos por ***cortes poliedrais***, foram propostos em oposição aos cortes genéricos no plano, para problemas de programação inteira 0-1, como o TSP. O objetivo é fazer uso da estrutura combinatória (poliedral) para gerar os cortes. Então, o conhecimento da estrutura combinatória é essencial para a aplicação desta abordagem.

Neste tipo de abordagem, os cortes podem eliminar soluções inteiras, desde que não seja a solução ótima. Daí ser necessário o conhecimento da estrutura poliedral do problema.

Crowder, Johnson e Padberg [CJP83] propuseram a seguinte estratégia: para obter os cortes combinatórios, eles consideraram cada restrição do programa inteiro separadamente e, então, usaram a estrutura combinatória do programa inteiro definida por apenas uma restrição (conhecida por problemas da mochila). Depois da aplicação dos cortes combinatórios na relaxação linear inicial, é aplicado um *branch-and-bound*: o objetivo, é que o tamanho da árvore de pesquisa seja significativamente reduzida pelos cortes combinatórios.

2.4.1.5.*Branch-and-Cut*

O termo *branch-and-cut* foi introduzido por Padberg e Rinaldi [PR87], justamente quando lidavam com instâncias do TSP. Antes disso, vários pesquisadores já utilizavam uma abordagem similar [GP79a, GP79b, CP80, GP85].

A título de informação, no parágrafo seguinte serão citados os códigos para o desenvolvimento de *branch-and-cut* mais disseminados no momento:

Dois dos códigos mais disseminados no momento para o desenvolvimento de um *branch-and-cut* é o **CPLEX** [CPLE94], desenvolvido por Bixby da Universidade do Texas em Austin, e o **OSL**, desenvolvido pela IBM. Mas, alguns pesquisadores renomados afirmam que o **MINTO**, desenvolvido na Georgia Tech, é mais eficiente [Cer96]. Eles ainda indicam um terceiro como sendo o mais adequado: o **ABACUS**, desenvolvido na Universidade de Köln. Além destes, podem ser citados: o **ABC-OPT**, desenvolvido por Hoffman e Padberg; e o **MIPO**, desenvolvido por BMAS e Cornuéjols.

No restante desta subseção, serão apresentadas as principais estratégias na utilização do método de *branch-and-cut* para a resolução de problemas.

Branch-and-Cut Tradicionais

A idéia principal é a combinação de cortes combinatórios e *branch-and-bound*. Na abordagem de *branch-and-cut*, a geração automática de cortes é feita não somente na fase anterior ao *branch-and-bound* (como mostrado na seção anterior), mas também entre os estágios de enumeração na árvore de pesquisa. Desta forma, a grande virtude desta abordagem é que uma restrição necessária para um nó de uma determinada sub-árvore pode se manter numa "lista de restrições" e ser válida para os nós de outras sub-árvores da árvore de pesquisa.

O procedimento é iniciado com a relaxação do problema de programação inteira. Resolve-se o problema relaxado e sempre que a solução obtida violar alguma das restrições de integralização do problema original, efetua-se um corte combinatório (adiciona-se mais uma restrição de integralização). A intenção é que a solução ótima seja obtida sem que todas as restrições de integralização do problema original sejam adicionadas.

A abordagem de *branch-and-cut* vem sendo cada vez mais utilizada, pois os resultados de sua aplicação para uma variedade de problemas combinatórios foram muito satisfatórios. O problema desta abordagem, vem justamente do fato de se usar cortes combinatórios e, de que para isto, torna-se necessário um conhecimento profundo, e específico, do problema combinatório em questão.

Cortes Genéricos no Plano num contexto de *Branch-and-Cut*

Esta abordagem foi iniciada por Hoffman e Padberg [HP91], mas somente muito recentemente é que vem sendo mais utilizada [BCC95, Bal95], e ganhando, cada vez mais, novos adeptos.

A principal diferença entre este método e o *branch-and-cut* tradicional, é que os cortes no plano podem ser gerados sem que se tenha um conhecimento específico da estrutura combinatória do problema, ou seja, os cortes efetuados não são combinatórios e sim, cortes gerais no plano.

Seguindo esta linha de pesquisa, cortes no plano disjuntivos ou *lift-and-project* foram usados com sucesso dentro de uma abordagem de *branch-and-cut*, resultando na resolução de uma grande variedade de programas inteiros por Balas, Ceria e Cornuéjols [BCC95]. Ainda dentro de um contexto similar, Balas, Ceria, Cornuéjols e Natraj [Bal95] obtiveram grande sucesso com aplicação dos cortes no plano de Gomory.

Pré-processamento e Heurísticas num contexto de *Branch-and-Cut*

Num projeto de um *branch-and-cut* eficiente, deve-se dedicar um cuidado muito especial à fase de pré-processamento e na utilização de heurísticas.

Técnicas de pré-processamento tentam melhorar a relaxação linear através de simples considerações algébricas. Variáveis e restrições são checadas para evitar redundância, alguns coeficientes da matriz de restrições são ajustados e algumas variáveis são fixadas a certos valores, sem que isto afete o valor da solução ótima do problema inteiro. Desta forma, o número de variáveis e restrições do problema é reduzido significativamente, tornando sua resolução mais fácil.

A importância no uso de heurísticas, vem do fato de que soluções muito boas podem ser rapidamente obtidas, reduzindo significativamente o tamanho da árvore de busca no *branch-and-cut*.

2.4.1.6.Método de Geração de Colunas ou *Branch-and-Price*

Uma nova abordagem que vem sendo utilizada para a resolução de grandes instâncias de problemas de programação inteira é o Método de Geração de Colunas ou *branch-and-price* [Cer96, Bar94].

Nesta abordagem, uma formulação diferente do problema é utilizada, envolvendo um grande número de variáveis. Somente um subconjunto destas variáveis é geralmente utilizado durante o procedimento de resolução do problema e o resto das variáveis são tratadas implicitamente.

O processo se inicia com a geração de cortes no plano, onde são adicionadas desigualdades violadas, eliminadas algumas outras restrições e repetido este processo até que seja encontrado uma solução ótima inteira ou seja parado o processo com uma solução ótima fracionária. Se considera, a partir de então, as variáveis *"esquecidas"* ou descartadas. Para todas as variáveis inicialmente descartadas, é gerada uma coluna correspondente no sistema de restrições lineares e computado seus custos reduzidos através de técnicas de PL. Se todos os custos reduzidos possuírem o sinal correto, a solução encontrada é ótima também para o sistema constituído de todas as variáveis. Se nem todos os custos reduzidos tiverem o mesmo sinal, são adicionadas todas as variáveis (ou algumas, se existirem muitas) com o sinal errado para o PL corrente e repete-se, então, o procedimento de corte no plano. Utilizando o critério de custo reduzido pode ser mostrado que algumas variáveis podem ser eliminadas de início porque provavelmente nunca apareceriam em qualquer solução ótima, ou ainda, que algumas variáveis podem ser fixadas a um certo valor.

Esta metodologia vem obtendo muito sucesso no tratamento de problemas reais de *roteamento* e escalonamento na indústria de transportes [Cer96].

2.4.2.Métodos Aproximados

Na maioria das vezes, é muito difícil encontrar soluções ótimas para instâncias de problemas NP-difíceis. As instâncias podem ser simplesmente muito grandes, extrapolando a capacidade de armazenamento necessária até mesmo pelo melhor algoritmo exato. Por outro lado, pode também ser possível que o tempo permitido para computação não seja suficiente para um algoritmo alcançar a solução ótima. Em todos estes casos existe uma necessidade definitiva por algoritmos de aproximação que determinem soluções de boa qualidade num curto espaço de tempo [Ben90, Tai90, GL85, CP80].

Nesta seção, primeiramente serão considerados os métodos heurísticos e uma segunda abordagem engloba métodos mais recentes, que podem ser chamados de meta-heurísticas.

2.4.2.1.Heurísticas

A palavra ***heurística*** deriva do verbo grego *heuriskein* e significa *descobrir* [ZE81]. Consiste num estudo de métodos e regras para se descobrir ou assistir ao processo de resolução de problemas.

Nicholson [Nic71] define um método heurístico como um procedimento para resolver problemas por uma abordagem intuitiva na qual a estrutura do problema pode ser interpretada e explorada inteligentemente para se obter uma boa solução.

Mas, por que usar um método heurístico?

Tal pergunta já foi respondida na seção sobre complexidade, onde foi mostrada a existência de inúmeros problemas que não possuem algoritmos eficientes para resolvê-los. Então, para Problemas de Otimização Combinatória em particular, os métodos exatos costumam ser computacionalmente inviáveis (em tempo de execução e capacidade de armazenamento) e, por outro lado, as heurísticas possuem uma concepção simples onde são incorporados as partes mais importantes do problema.

Uma boa heurística deve ter as seguintes propriedades [Sou94]:

- Esforço computacional realista para se obter uma solução;
- Na média, a solução gerada deve ser próxima do ótimo;
- As chances da geração de uma solução ruim devem ser baixas;
- A implementação deve ser tão simples quanto possível.

Os métodos heurísticos podem ser divididos em sete grandes grupos, que são:

Heurísticas de Construção → Constroem uma solução completa (válida para o problema), a partir de uma solução parcial (incompleta ou nula) e através da inserção sucessiva de seus componentes individuais. Geralmente, uma solução completa só é obtida no final da execução de tais algoritmos.

Adicionam elemento a elemento, um de cada vez, à uma solução parcial até que ela se torne factível (válida). Exemplo: *Nearest Neighbor* para o TSP.

Heurísticas de Melhoria → Uma heurística de melhoria inicia seu procedimento com uma solução válida e, sucessivamente, a melhora com seqüências de trocas de elementos. Exemplo: algoritmo de Lin-Kernighan para o TSP.

As heurísticas de melhoria são também conhecidas como heurísticas de ***busca local***, isto porque elas realizam uma busca exaustiva na vizinhança da solução inicial, até encontrar a solução que represente o *mínimo local* de tal região. Geralmente, várias soluções completas são geradas durante a execução de tais algoritmos, até que a solução que represente o mínimo local de tal algoritmo, segundo a solução inicial recebida como entrada, seja encontrada.

Heurísticas de Decomposição → São heurísticas que subdividem o problema original em uma seqüência de sub-problemas. Exemplo: Algoritmo heurístico para o Problema de Roteamento de Veículos, onde o mesmo é decomposto num Problema de Particionamento e no TSP (VRP=*Set Partition*+TSP).

Heurísticas de Particionamento → Tais heurísticas dividem o problema em vários problemas menores e, então, unem as soluções.

Heurísticas de Relaxação → *Relaxam* o espaço do problema original para poder ser utilizado um algoritmo mais rápido e, então, ajusta a solução transformando-a em factível para o problema original. *Relaxar* significa retirar algumas das restrições do problema.

Heurísticas baseadas em Formulações Matemáticas → Alteram um algoritmo baseado na formulação matemática do problema, objetivando um melhor desempenho no tempo de processamento. Exemplo: um algoritmo que substitua um parâmetro quadrático por um linear.

2.4.2.2.Meta-Heurísticas

Durante a última década, esforços foram despendidos na tentativa de solucionar grandes problemas de otimização. Vários tipos de heurísticas têm sido sugeridas, muitas das quais projetadas especificamente para determinados problemas, não sendo adaptadas facilmente para tipos diferentes [WiH89].

Um outro tipo de pesquisa por boas soluções considerado mais geral pode ser desenvolvida para problemas de otimização. Cada um destes procedimentos são chamados de **meta-heurísticas**, ou seja, são métodos heurísticos que contém outras heurísticas em seus procedimentos internos, e que tentam escapar de mínimos locais de uma maneira muito mais sistemática do que a sugerida por outros grupos de heurísticas. Um ingrediente básico é o uso de *aleatoriedade* ou *pesquisa estocástica*, ao invés das heurísticas puramente deterministas [EG93].

Algoritmos Genéticos

Algoritmos Genéticos (***Genetic Algorithm* - GA**) são algoritmos de busca baseados em mecanismos de seleção natural e genética [Gol89].

Foi desenvolvido por *John Holland* e seus alunos da Universidade de Michigan, sendo os primeiros resultados divulgados no artigo "*Adaptation in Natural and Artificial Systems*", em 1975.

Um GA recebe uma população de soluções candidatas e tenta melhorá-las dando para cada solução uma certa medida de qualidade. A idéia fundamental do GA é garantir a *sobrevivência da solução mais adequada*, que junto com uma certa informação histórica e aleatoriedade selecionam novos pontos de pesquisa, aumentando a chance de se obter melhores resultados.

As soluções do problema são codificadas numa estrutura denominada *cromossomo*, que por sua vez é constituído por uma cadeia de genes. Um *gen* representa uma certa característica da solução. Os valores que os genes assumem são chamados de *alelos* (geralmente os genes assumem apenas os valores 0 e 1). *Locus* é a posição de um gen no cromossomo. Desta forma, uma solução pode ser representada num GA através da representação de todas as variáveis do problema na forma binária reunidas num único cromossomo.

A cada cromossomo é associado um valor que representa quão bom ele é (representa o nível de qualidade da solução). É a função objetivo do problema, que num GA recebe o nome de ***fitness function*** ou função de adequabilidade.

Um conjunto de cromossomos é denominado de *população* e cada nova população criada corresponde a uma nova *geração*. Para o estabelecimento de uma nova *geração* de soluções, os cromossomos sofrem diferentes tipos de alterações. Tais modificações são efetuadas, basicamente, por dois **métodos de reprodução**:

- **Crossover** → recombina dois ou mais cromossomos para a formação de um novo.
- **Mutação** → utiliza um único cromossomo e através de alterações randômicas gera um outro.

A ***seleção*** dos cromossomos a se reproduzirem é um importante fator na performance de um GA. Os melhores cromossomos da atual população podem ser simplesmente selecionados e inseridos na nova população sem sofrerem nenhuma alteração. Este procedimento é denominado de *elitismo*, garantindo uma melhora monotônica na *fitness function* e acelerando a convergência.

Alguns critérios de parada do GA são: tempo de execução, número de gerações, falta de diversidade (cromossomos da população muito parecidos) ou ainda, convergência (últimas k gerações sem nenhuma melhora na *fitness function* da melhor solução encontrada).

Sendo assim, um GA pode ser descrito da seguinte forma:

Genetic Algorithm

1. Gere um conjunto de soluções aleatoriamente (população inicial);
2. Enquanto o critério de parada não for satisfeito, faça:
   2.1. Gere uma nova população de soluções através dos mecanismos de seleção e reprodução;
   2.2. Avalie os novos cromossomos gerados com a *fitness function*;
3. Retorne o melhor cromossomo encontrado durante todo o processo.

Simulated Annealing

A técnica de ***Simulated Annealing*** **-SA**, é baseada na correspondência entre o processo de pesquisa por uma solução ótima num problema de otimização combinatória e fenômenos ocorridos na física. O nome vem da analogia com os processos físicos de *annealing de sólidos*.

Introduzido por Kirkpatrick, Gelatt e Vecchi [KGV83] e Cerny (independentemente) [Cer85], foi desenvolvido a partir de idéias da mecânica estatística quando realizavam *"The study of the behavior of very large systems of interating components"*.

Mecânica Estatística é o campo central de estudo da física da matéria condensada, consistindo de um conjunto de métodos para analisar propriedades agregadas de um grande número de partículas encontradas em amostras de matéria sólida ou líquida, consistindo em descobrir o que acontece num sistema submetido a limites baixos de temperatura, como por exemplo, se as partículas deste sistema se solidificarão ou se tornarão fluida e, caso se solidifiquem, se elas formam um sólido cristalino ou um vitrificado. A probabilidade de um sistema em equilíbrio térmico a uma certa temperatura T se encontrar no estado s é obtida através da *distribuição de Boltzman* [KGV83].

Metropolis *et al* [Met53] introduziram um algoritmo simples que pode ser usado para fornecer uma eficiente simulação de uma coleção de partículas em equilíbrio a uma dada temperatura. O processo é constituído de dois passos:

1. Aumente a temperatura até um nível que permita a fusão total do sólido;
2. Diminua *lentamente* a temperatura até que todas as partículas atinjam o estado de equilíbrio (este estado de equilíbrio é chamado de *ground state*).

Na fase de fusão, as partículas do sólido se rearranjam aleatoriamente. Na fusão completa (ground state) as partículas estão rearranjadas de uma maneira altamente estruturada. Este equilíbrio é obtido somente se a temperatura máxima for extremamente elevada e se o resfriamento for suficiente lento.

Kirkpatrick *et al* [KGV83] tiveram, então, a idéia de utilizar o *algoritmo de Metropolis* (como ficou conhecido) para a geração de uma seqüência de soluções para Problemas de Otimização Combinatória. Desta forma, se estabeleceu uma analogia entre os sistemas físicos de partículas e tais problemas, como a seguir:

- As soluções de problemas de otimização combinatória equivalem a estados físicos do sistema;
- As variáveis do problema equivalem às moléculas, partículas do sistema;
- O custo da solução equivale à energia do estado.

O algoritmo de *Simulated Annealing* pode agora ser visto como um processo iterativo do algoritmo de Metropolis, onde a redução da temperatura corresponde ao parâmetro de controle a ser otimizado. No SA também se busca o equilíbrio a cada patamar de temperatura. Para isto faz-se um determinado número de transições, cujas condições de paradas são geralmente o número máximo de transições (soluções geradas) ou o número máximo de soluções aceitas. Sendo assim, cada vez que um destes valores for atingido, o teste de equilíbrio é realizado.

Diferentemente dos algoritmos de busca local o SA aceita não somente soluções que são melhores que a solução corrente, mas pode aceitar também soluções que sejam piores e isto é o que garante ao SA escapar de mínimos locais. Uma outra diferença, é que o SA não depende da qualidade da solução inicial gerada.

Uma nova solução é gerada através de perturbações na solução corrente. Perturbar uma solução consiste em gerar uma nova solução partindo-se de uma solução existente.

Um algoritmo básico que descreve os principais passos de um SA é dado a seguir:

Algoritmo *Simulated Annealing*

1. Gere uma solução inicial;
2. Calcule o valor da função objetivo;
3. Faça solução corrente = solução inicial;
4. Enquanto o *critério de parada* não for satisfeito faça:
  4.1. Perturbe a solução corrente e calcule o valor da diferença entre os valores das funções objetivo;
  4.2. Faça *continue*=VERDADEIRO;
  4.3. Se $\Delta c \geq 0$:
    4.3.1. Gere um número aleatório $\alpha \in (0,1)$;
    4.3.2. Calcule a probabilidade de aceitação $P(\Delta c) = e^{-\Delta c / kT}$;
    4.3.3. Se $\alpha \geq P(\Delta c)$:
      4.3.3.1. Rejeite a solução perturbada;
      4.3.3.2. *continua*=FALSO;
  4.4. Se *continua*:

4.4.1. Aceite a nova solução perturbada;
5. Retorne a melhor solução encontrada.

Busca Tabu

A técnica de Busca Tabu (***Tabu Search* - TS**) é uma meta-heurística cuja idéia básica é a realização de movimentos inteligentes e a incorporação dos mesmos numa lista (lista tabu) de forma que não possam ser imediatamente desfeitos.

Foi desenvolvida por Fred Glover [Glo89, Glo90a, Glo90b], baseada nas idéias de Pierre Hansen que desenvolveu o método conhecido por *steepest ascent/mildest descent*.

Os três principais fundamentos de Tabu Search consistem em:

- Uso de estrutura de memória flexível que facilita a implementação de múltiplos critérios de avaliação e de processamento de informação histórica;
- Mecanismo de controle que permita a manipulação de restrições e liberações de condições de busca para períodos de tempo variados;
- Possibilidade de diferentes estratégias de busca para curto, médio e longo prazo (busca agressiva ou diversificada).

As restrições que devem ser satisfeitas num certo período de tempo ficam embutidas na lista tabu, impedindo a realização de movimentos repetidos e, desta forma, aumentando a diversidade da busca e possibilitando a fuga de mínimos locais. A busca tabu também tenta evitar a realização de movimentos cíclicos, induzindo a pesquisa a tomar uma nova direção (revisar algumas soluções já obtidas). As restrições tabu não são manipuladas isoladamente, podendo ser contrabalançadas (de certa forma violadas) pela aplicação de critérios de aspiração.

Critérios de aspiração podem ser definidos, como sendo algumas características tão desejáveis que torna o movimento admissível mesmo se ele pertencer à lista tabu. Por exemplo, se for encontrada a melhor solução dentre todas já obtidas.

A escolha do melhor candidato se constitui num passo crítico. Primeiramente, cada um dos movimentos da lista de movimentos candidatos devem ser avaliados (um movimento pode ser, por exemplo, uma troca de valores entre variáveis). Inicialmente, a avaliação do movimento pode ser baseada a mudança produzida na função objetivo. Existem outras formas de se avaliar os movimentos, porém o mais importante é que esta avaliação seja adaptável, incorporando conceitos de diversificação e intensificação de soluções.

Um algoritmo de TS pode ser descrito através dos seguintes passos:

Algoritmo *Tabu Search*

1. Gere uma solução inicial;
2. Crie uma lista tabu LT;
3. Enquanto o critério de parada não for satisfeito faça:
   3.1. Execute o melhor movimento que não for proibido por LT (movimento que não estiver na lista tabu) ou que satisfaça o critério de aspiração estabelecido;

3.2. Atualize a lista tabu LT;
3.3. Atualize, se desejado, o critério de aspiração;
4. Retorne a melhor solução encontrada.

Grasp

Greedy Randomized Adaptive Search Procedure - **GRASP** - é uma meta-heurística muito simples, consistindo num processo iterativo constituído de duas partes: a fase construtiva e a fase de busca local.

Foi elaborada por Feo e Resende [FR95a] onde, na fase construtiva, um algoritmo heurístico de construção gera uma solução factível que é, então, melhorada na segunda fase por um algoritmo de busca local. A cada iteração a solução corrente é comparada com a nova solução gerada e melhorada e, ao final do processamento, a melhor solução encontrada é devolvida como resultado.

A implementação de um GRASP é trivial, bastando que se tenha disponível um algoritmo de construção e um outro de melhoria (de busca local) para o problema que se deseja resolver.

GRASP pode ser descrito através dos seguintes passos:

Algoritmo GRASP

1. Gere uma solução inicial S;
2. Enquanto o critério de parada não for satisfeito faça:
  2.1. Construa uma solução;
  2.2. Melhore a solução, gerando uma nova solução S';
  2.3. Se S' for melhor do que S:
    2.3.1. Faça $S=S'$;
3. Retorne S.

Redes Neurais

A abordagem conhecida como Redes Neurais é considerada por alguns como uma meta-heurística, classificando-a, tal como GA, SA e TS, como *heurísticas de melhoria aleatória* [JRR94], que tentam escapar de mínimos locais realizando uma busca aleatória em diversos pontos da região factível do problema. De qualquer forma, Redes Neurais vem sendo aplicada em diversas áreas para os mais variados tipos de problemas, principalmente de otimização combinatória [Lau92, IM92, FW91].

Redes Neurais (***Neural Networks***) é uma tentativa de modelar o funcionamento de um cérebro humano baseada em modelos matemáticos que incorporam algumas características de processamento inteligente, tais como: tomada de decisões, adaptação a novas situações, etc.

Desta forma, um conjunto de elementos, no papel de neurônios, são conectados por um determinado tipo de rede de interconexão e, através de vários tipos de modelos

de propagação de sinais entre os "neurônios", realizam algum tipo de aprendizado [JRR94].

2.4.2.3.Times Assíncronos

Times Assíncronos, do inglês ***Asynchronous Teams*** ou **A-Teams**, consiste numa nova técnica para a resolução de problemas introduzida por Talukdar e Souza [TS90, ST91, Sou93]. Esta técnica se baseia na utilização conjunta de diversos algoritmos, de tal forma que os mesmos se ajudem mutuamente, compartilhando soluções e gerando novas soluções resultantes da atuação de vários e, até mesmo, de todos os algoritmos do conjunto. Isto possibilita a obtenção de soluções muito boas, próximas do ótimo ou até mesmo a solução ótima para o problema, que não poderiam ser obtidas por nenhum dos algoritmos do time se executados isoladamente.

O método de Times Assíncronos vem sendo aplicado com sucesso em Problemas de Otimização Combinatória [TS90, ST91, Mur92, Sou93, Pei95, Cav95, Lon95, Cam95].

As vantagens no uso de A-Teams já detectadas para seu uso em problemas monobjetivos, podem ser resumidas como segue:

- Apenas um modelo matemático simples do problema é necessário;
- Durante sua execução, inúmeras soluções factíveis podem ser geradas;
- Fornece como resultado múltiplas soluções;
- Implementação relativamente rápida e fácil;
- Possibilita a geração de uma solução de boa qualidade muito rapidamente;
- Possibilita a geração de soluções com características bem distintas uma das outras;
- Gera melhores soluções do que as geradas por qualquer um dos algoritmos do time quando executados isoladamente.

O capítulo 5 é todo dedicado ao método de Times Assíncronos.

2.5. Resumo

Neste capítulo foi caracterizada uma importante classe de problemas, a classe dos **Problemas de Otimização Combinatória**. A importância de tal classe para este trabalho é devido ao fato do problema abordado pertencer à mesma.

Uma visão geral sobre métodos para a resolução de tais problemas foi apresentada, desde os mais antigos e tradicionais, até os métodos mais recentes, como o de **Times Assíncronos**, que é o utilizado neste trabalho.

Pode ser observado que a ênfase foi dada a problemas de otimização combinatória com uma única função objetivo, isto porque, tradicionalmente, a teoria que envolve problemas multiobjetivos é, ainda, diferenciada dos demais problemas. Talvez porque se constitua numa área relativamente recente onde os conceitos e definições ainda não estão completamente solidificados. No capítulo seguinte, será, então, abordada a classe de

Problemas Multiobjetivos, de forma a completar o entendimento do domínio do problema proposto neste trabalho.

2.6. Notas Bibliográficas

Existe um vasto material bibliográfico envolvendo Problemas de Otimização Combinatória e seus métodos de resolução. O livro de Newhauser e Wolsey foi uma importante fonte de pesquisa [NW88], bem como o de Papadimitriou e Steiglitz [PS82], o de Lawler *et al* [LLKS85] e o texto de Jünger, Reinelt e Rinaldi [JRR94]. Grötchel e Lovász [GL93] apresentam um compêndio sobre o assunto.

A parte sobre complexidade pode ser encontrada nos trabalhos de Cook [Coo71], Karp [Kar72], Trauber [Tra76], Garey e Johson [GJ79], Manber [Man89] e Cormen [Cor91]. Sobre Matemática Combinatória: Pólya, Tarjan e Woods [PTW83]; e, Liu [Liu68].

Em relação aos métodos exatos pode ser citados o texto de Ceria [Cer96], que foi uma fonte preciosa para a descrição das abordagens mais recentes, apresentando, também, um resumo bibliográfico sobre o assunto.

Especificamente para Cortes no Plano: Ceria, Cordier, Marchand e Wolsey [Cer95]; e, Gomory [Gom60].

Para *Branch-and-Bound*: Kumar e Kanal [KK88]; Miller e Peckny [MP91, MP89]; Balas e Toth [BT85]; Aho, Hopcroft e Ullman [AHU74]; Balas [Bal65]; e, Land e Doig [LD60].

Para Programação Dinâmica: Cormen [Cor91]; Grötchel [Gro93]; Karp e Held [KH67]; Bellman e Dreyfus [BD62].

Para Cortes Combinatórios no Plano: Roy e Wolsey [RW87]; e, Crowder, Johnson e Padberg [CJP83].

Para *Branch-and-Cut*: Balas [Bal95]; Padberg e Rinaldi [PR91, PR87]; Hoffman e Padberg [HP93, HP91]; Johnson [Joh90]; Crowder e Padberg [CP80]; e, Grötchel e Padberg [GP79a, GP79b]. Com cortes disjuntivos: Balas, Ceria e Cornuéjols [BCC95, BCC93]; e, Balas, Ceria, Cornuéjols e Natraj [Bal95].

Para *Branch-and-Price* ou Método de Geração de Colunas: Barnhart, Johnson, Nemhauser, Savelsbergh e Vance [Bar94].

Para os métodos aproximados em geral podem ser citados: Souza [Sou94, Sou93]; Bentley [Ben92, Ben90]; e, Syslo, Deo e Kowalik [SDK83].

Especificamente sobre heurísticas: Ceria, Nobili e Sassano [CNS95]; Reinelt [Rei92, Rei94]; Mark e Morton [MM93]; Margot [Mar92]; Papadimitiou [Pap90]; Zanakis, Evans e Vazacopoulos [ZEV89]; Johnson, Papadimitriou e Yannakakis [JPY88]; Golden e Stewart [GS85]; Zanakis e Evans [ZE81]; Silver, Vidal e Werra [SVW80]; Fisher [Fis80]; Karp [Kar76]; Weiner [Wei75]; Rosenkrantz, Stearns e Lewis [RSL77]; Or [Or76]; Magazine, Newhauser e Trotter [MNT75]; Nicholson [Nic71]; e, Pólya [Pol73].

Para Algoritmos Genéticos: Horn e Nafpliotis [HN93]; Gendreau, Hertz e Laporte [GHL92]; Davis [Dav91, Dav85]; Souza e Talukdar [ST91]; Calloway [Cal91]; Mühlenbein, Schomisch e Born [MSV91]; Hurkens [Hur91]; Ulder, Pesch, van Laarhoven, Bandelt e Aarts [Uld90]; Mühlenbein [Müh90]; Austin [Aus90]; Cleveland e Smith [CS89]; Goldberg [Gol89, Gol85]; Mühlenbein, Gorges-Schleuter e Krämer [MGK88]; e, Goldberg e Lingle [GL85]. Uma bibliografia sobre GA é dada por Collins, Eglese e Golden [CEG88]. Cantú-Paz [Can95] apresenta uma revisão bibliográfica sobre GAs paralelos.

Para *Simulated Annealing*: Johnson, Aragon, McGeoch e Scheron [Joh91]; Martin, Oto e Felten [MOF91]; Johnson [Joh90]; Ogbu e Smith [OS90]; Aarts e Korst [AK89]; Rutenbar [Rut89]; [FS88]; [Laa88]; Laarhoven e Aarts [LA87]; Kirkpatrick [Kir84]; [BR84,]; Cerny [Cer85]; Kirkpatrick, Gelatt e Vecchi [KGV83]; Metropolis *et al* [Met53]. Rutenbar [Rut89] apresenta uma revisão geral sobre tal abordagem. Sobre convergência em SA, procurar: Rajasekaran e Reif [RR92]; e, Faigle e Schrader [FS88].

Para *Tabu Search*: Ribeiro [Rib96]; Laguna [Lag95]; Pureza, França e Aragão [PFA95]; Taillard [Tai90]; Glover [Glo90a, Glo90b, Glo89, Glo86]; Werra e Hertz [WH89]; Knox e Glover [KG89]; Widmer e Hertz [WiH89]; Malek, Guruswamy, Owens e Pandya [Mal89a]; Malek, Heap, Kapur e Mourad [Mal89b]; e, Hertz e Werra [HW87].

Para GRASP: Ribeiro [Rib96]; Resende e Ribeiro [RR95]; Pardalos, Pitsoulis e Resende [PPR95]; Resende, Feo e Smith [RFS95]; e, Feo e Resende [FR95a, FR94b].

Maiores detalhes sobre Redes Neurais podem ser encontrados em: Inouchi e McLoughlin [IM92]; Takefuji [Tak92]; Fritzke e Wilke [FW91]; Wilson e Pawley [WP88]; Durban e Willshaw [DW87]; Lippman [Lip87]; Rumelhart, Hinton e McClelland [RHM86]; e, Hopfield e Tank [HT85]. Lau [Lau92] editou um livro específico sobre tal abordagem.

Capítulo
3

Problemas Multiobjetivos

"Meios poderosos, mas objetivos confusos: tal é a característica de nossa época."
Albert Einstein.

Os **Problemas Multiobjetivos** se caracterizam por possuir dois ou mais objetivos a serem atingidos, ao invés de apenas um. Geralmente os objetivos são conflitantes entre si, de tal forma que satisfazê-los simultaneamente é uma tarefa quase sempre impossível, isto é, uma solução que seja a melhor em relação a todos os objetivos é geralmente inexistente e o que se procura são soluções que satisfaçam da melhor maneira possível todos eles. Desta forma, a busca pela *solução ótima* dá lugar à busca pelas *soluções de melhor compromisso* possível para todos os objetivos do problema.

Neste capítulo, será abordada a teoria que envolve os Problemas Multiobjetivos: **Otimização Multiobjetiva**. Será apresentada, também, uma classificação dos principais métodos de resolução encontrados na literatura, juntamente como uma descrição sucinta dos mesmos.

3.1. Introdução

Geralmente, quando se pensa em resolver um problema, a idéia de *um* objetivo claro e único a ser atingido surge em mente. Apesar da maioria dos problemas práticos possuir múltiplos objetivos, a transformação destes objetivos num único ainda é muito utilizada, pois, desta forma, a resolução do problema se torna mais simples. As duas abordagens mais encontradas na literatura para o *encapsulamento* dos múltiplos objetivos do problema num só, são: adoção de uma função objetivo que é uma combinação escalar de todas as funções do problema; e, priorização de um dos objetivos com a transformação dos objetivos restantes em restrições do problema.

A Programação Matemática Multiobjetivo (*Multiple Objective Mathematical Programming - MOMP*) [SR91, Ste86] é a área que envolve a otimização de problemas com múltiplas funções objetivo, onde a presença de objetivos geralmente conflitantes entre si impede a existência de uma *solução ótima* e conduz para a procura das soluções de melhor compromisso.

É apresentada uma classificação geral de Problemas Multiobjetivos. São apresentados, também, os conceitos básicos sobre Otimização Monobjetiva de forma a tornar mais clara as definições posteriores sobre Otimização Multiobjetiva, que pode ser vista como uma extensão do caso de uma só função. Uma classificação dos métodos de resolução, com uma descrição sucinta de cada classe é, por fim, apresentada.

3.2. Histórico

A Teoria que envolve os Problemas Multiobjetivos começou a ser intensamente desenvolvida na década de sessenta. Charnes e Cooper [CC61] desenvolveram a *Goal Programming* e Keeney e Raiffa [KR76] desenvolveram a teoria e alguns métodos para Teoria da Utilidade Multiatributo (*Multiple Attribute Utility Theory* - MAUT) e Análise de Decisão Multicritério (*Multiple Criteria Decision Making* - MCDM).

Na década de setenta [Haw80] foram enfatizados os fundamentos teóricos da Programação Matemática Multiobjetiva e o desenvolvimento de algoritmos e procedimentos para a resolução de tais problemas. Muitas idéias tiveram como base a Teoria da Programação Matemática tradicional. Nesta época, ainda era muito pequeno o material de pesquisa da área disponível na literatura.

Nos anos oitenta [KMW92, Haw80], um grande avanço foi dado tendo como base a aplicação da Otimização Multiobjetiva no Suporte à Decisão Multicritério (*Multiple Criteria Decision Support Systems* - MCDSS), referentes à avaliação de um Agente de Decisão (*Decision Maker* - DM). Cada vez mais, métodos que enfocam as preferências de um Agente de Decisão sobre os critérios em análise, seja antes, durante ou depois do processo de obtenção das soluções vêm sendo desenvolvidos. Durante os últimos anos, o interesse se concentrou no desenvolvimento de Procedimentos Interativos Multiobjetivos [Cli95, KMW92, Wal91].

White [Whi90] publicou um artigo com uma coletânea de modelos de aplicações de Programação Matemática Multiobjetiva.

Corner e Kirkwood [CK91] publicaram um artigo com recentes avanços em aplicações de análise à decisão na Pesquisa Operacional, onde é possível encontrar aplicações na área de Energia, de Manufatura e Serviços, Médica, de Política Pública, entre outras.

O uso de Computação Gráfica na visualização de problemas, critérios e alternativas (soluções) é uma forte tendência também nesta área. Wallenius [Wal91] cita alguns exemplos de sistemas recentes que usam interfaces gráficas, tal como *PARETO RACE* [KW88].

De um modo geral os Problemas de Programação Linear Multiobjetivos são os mais abordados na literatura, embora existam aplicações envolvendo os mais diversos tipos de problemas [LT91].

A classe de Problemas de Programação Inteira Multiobjetiva, entretanto, não têm recebido a devida atenção. Apesar de tais problemas serem muito comuns na prática, existe muito pouco material, teórico e prático, disponível na literatura [RS95a, RS95b, Cli95, Kar95, KMW92].

A seguir, será dada uma classificação dos Problemas Multiobjetivos, divididos em duas grandes classes.

3.3. Classificação de Problemas em Análise de Decisão Multicritério

Os Problemas Multiobjetivos podem ser classificados sob diversos pontos de vista. Muitas vezes os termos "*multiobjetivo*", "*multicritério*", "*multiatributo*" e "*multidimensional*" são usados para descrever situações de decisão e, ainda hoje, não existe uma diferenciação definitiva entre os mesmos [Cli95].

Análise de Decisão Multicritério (*Multiple Criteria Decision Making* - MCDM) vem a ser a designação mais aceita e utilizada para todas as metodologias que envolvem o uso de algum dos termos definidos no parágrafo anterior. Esta grande área pode ser subdividida em duas outras: **Análise de Decisão Multiatributo** (*Multiple Attribute Decision Making* - MADM) e **Análise de Decisão Multiobjetivo** (*Multiobjetivo Decision Making* - MODM) [Haw80].

Os problemas abordados pela Análise de Decisão Multiatributo (MADM) são aqueles que possuem uma pequena e explícita lista de alternativas (soluções), de pequeno porte, muitas vezes denominados de *Problemas Discretos* (tais problemas não tem nenhum relacionamento com os Problemas Combinatórios - Discretos - descritos no capítulo 2, apesar de serem denominados de tal forma), enquanto os problemas abordados pela Análise de Decisão Multiobjetivo (MODM) são aqueles que possuem um grande número de alternativas implicitamente definidas pelas restrições do problema, denominados de Problemas de Programação Matemática (linear, não-linear ou inteira). As duas classes de problemas são muito comuns na prática, motivo pelo qual serão abordadas em sub-seções separadas.

3.3.1. Problemas MADM ou Problemas Discretos

Os Problemas Discretos ou Problemas de Análise de Decisão Multiatributo - MADM englobam problemas que possuem um conjunto de alternativas (soluções) relativamente pequeno e explícito (previamente conhecido). Podem ser destacados os seguintes casos:

- o número de critérios é pequeno vs. grande;
- os valores dos critérios são conhecidos vs. não conhecidos com certeza;
- os valores dos critérios são conhecidos vs. desconhecidos à priori;
- os critérios são explicitamente vs. implicitamente especificados.

Tais casos geram um total de 16 diferentes combinações de problemas, nem todas extensivamente pesquisadas. Os problemas mais abordados na literatura são os seguintes:

1. O número de critérios é pequeno (menor que dez), os valores dos critérios são conhecidos e as alternativas são conhecidas à priori. Exemplo: Escolha de bens de consumo duráveis, como uma televisão, por exemplo, a partir de um conjunto de modelos disponíveis. Pode ser encontrado um exemplo da compra de uma máquina de lavar, no livro de Zeleny [Zel82];
2. Como acima (em 1), mas os valores dos critérios não são conhecidos com certeza. Na maioria dos casos tais critérios são modelados através de alguma distribuição de probabilidade. Exemplo: Escolha da localização de um aeroporto comercial [KR76];
3. O número de alternativas é conhecido à priori, geralmente avaliado usando um grande número de critérios que geralmente possuem uma estrutura hierárquica. Exemplo: Escolha de um *mainframe* (computador de grande porte) a partir de um conjunto pré-selecionado,onde muitos dos critérios são muito difíceis de serem expressados quantitativamente [Kor86];
4. O número de alternativas é muito grande ou a geração de alternativas é custosa. Exemplo: Uma organização recrutando um empregado para um trabalho específico [Kor91];
5. Os critérios necessitam da especificação de um Agente de Decisão. Exemplo: Escolha de uma casa a partir de um conjunto (pequeno) de casas *potenciais*, onde alguns critérios como tipo de jardim e vizinhança não são facilmente especificados [KMW92].

3.3.2. Problemas MODM ou Problemas de Programação Matemática

Os Problemas de Análise de Decisão Multiobjetivo - MODM englobam problemas que possuem um conjunto de alternativas (soluções) implicitamente definidas por um conjunto de restrições. Podem ser destacados os seguintes casos:

- Modelo linear vs. não-linear;
- Variáveis contínuas vs. variáveis inteiras;
- Número de objetivos pequeno vs. grande;

- Problema de grande escala vs. pequena escala (em relação ao número de restrições e/ou variáveis);
- Relacionamentos entre as variáveis quantitativos vs. qualitativos;
- Alternativas de decisão (soluções) são conhecidas vs. não conhecidas à priori.

Isto resulta num total de 64 combinações diferentes de problemas, e também, neste caso, nem todos são extensivamente estudados. Os mais abordados na literatura são:

1. O modelo é linear, as variáveis são contínuas, o número de objetivos é pequeno, o problema é de pequena escala, os relacionamentos entre as variáveis são quantitativos e o espaço de decisão é conhecido à priori. Exemplo: Estabelecimento do preço de bebidas alcoólicas num estado monopolista [KS88];
2. Idem, com exceção ao modelo do problema que passa a ser não-linear. Exemplo: Planejamento de controle num sistema de produção contínuo [RW91];
3. Como em 1, com exceção ao modelo do problema que passa a ser inteiro (todas as variáveis do problema assumem um valor inteiro) e que o problema é de pequeno escala. Exemplo: Seleção do tipo de mídia para a divulgação de uma propaganda [KNW89];
4. Como em 1, com a exceção ao modelo que passa a ser de grande escala. Exemplo: Modelos para o Monitoramento de Grandes Áreas Florestais [Soi88];
5. Como em 1, com exceção ao relacionamento entre algumas ou todas as variáveis que passa a ser qualitativo. Exemplo: Identificação de uma estratégia de marketing para uma *software house* [KW90].

3.4. Otimização Monobjetiva

Um problema de otimização com uma única função objetivo pode ser descrito da seguinte maneira [GHD82]:

$$\begin{aligned} &min\ z = f(x) \\ &s.a.\ \ g_j(x) \geq b_j \quad , \quad j = 1,2,...,m \\ &\qquad x_i \geq 0 \quad , \quad i = 1,2,...,n \end{aligned} \qquad \text{Eq. 3.1}$$

Pode ser observado (Eq.3.1) que tal problema possui três partes: um objetivo, m restrições principais e n restrições de não-negatividade.

As funções $f(x)$ e $g_j(x)$ podem ser funções lineares ou não lineares do vetor de variáveis de decisão x. O vetor de variáveis $x=(x_1, x_2,..., x_n)$ que satisfazem *todas* as restrições é chamado de *ponto factível* (*viável*) e o conjunto de todos os pontos factíveis constituem a **região factível no espaço de decisão** [BJS90]. A *região factível* é denotada por S e pode ser assim definida:

$$S = \{x: x \in R^n,\ g_i(x) \leq 0,\ x_i \geq 0,\ \forall\ i,j\} \qquad \text{Eq. 3.2}$$

Um problema de otimização (minimização) consiste, então, em determinar o ponto factível (ou pontos) x^* da região factível S (Eq.3.2), $x^* \in S$, que possua um valor mínimo para $f(x)$, ou seja, $min\ f(x) = f(x^*)$. Qualquer ponto factível que gere $f(x^*)$ é chamado de **solução ótima**. O principal enfoque da Otimização Monobjetiva, portanto, é a busca pela solução ótima do problema.

Se as funções $f(x)$ e/ou $g_j(x)$ forem lineares, o problema é dito como sendo de Programação Linear. Se $f(x)$ e/ou $g_j(x)$ forem lineares, com a restrição adicional de que as coordenadas de cada ponto em S devem ser inteiras, o problema é dito como sendo de Programação Inteira. Se $f(x)$ e/ou $g_j(x)$ forem não-lineares, então, o problema se classifica como um problema de Programação Não-Linear.

3.5. Otimização Multiobjetiva

Um Problema de Otimização Multiobjetivo pode ser definido da seguinte maneira [Ste86]:

$$\begin{aligned} &min\ f(x) = \{f_1(x),\ f_2(x),...,f_k(x)\} \\ &s.a.\ \ x \in S \end{aligned} \qquad \text{Eq. 3.3}$$

onde $x \in R^n$ é o vetor de variáveis de decisão, f_i, $i=1,2,...,k$, onde $k \geq 2$, são as funções objetivos e S é o conjunto de pontos factíveis. O conjunto de soluções factíveis no espaço de critérios R^k é chamado de **região factível no espaço de objetivos** e denotada por Z, onde:

$$Z = \{z \subset R^k | z = f(x)\ \ \forall\ x \in S\} \qquad \text{Eq. 3.4}$$

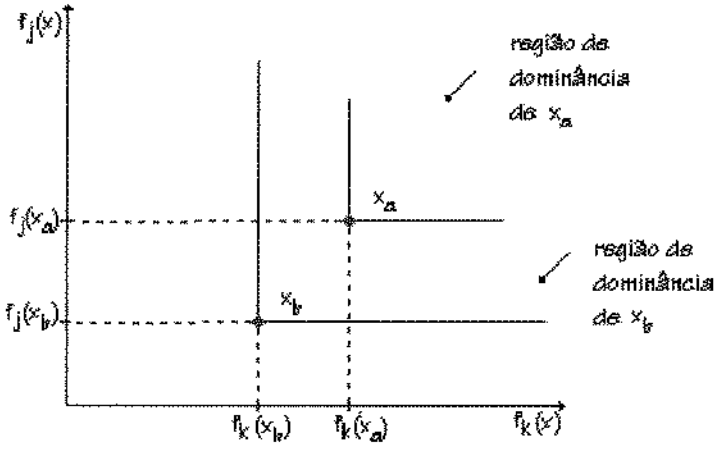
Neste caso, z é um vetor *k-dimensional* de objetivos o qual é um mapeamento de x para cada uma das k funções objetivo do problema. Z é o conjunto de vetores objetivo o qual é a imagem de S sobre todas $f_i(x)$'s e, para cada $z \in Z$, existe pelo menos um $x \in S$ tal que $z=f(x)$.

Neste trabalho, as soluções da região factível no espaço de objetivos serão denominadas de ***soluções k-dimensionais*** ou ***soluções multidimensionais***.

O problema acima é também conhecido como Problema da Minimização de Vetor (*Vector Minimization Problem*). Os múltiplos objetivos do problema podem ser conflitantes entre si, o que faz com que uma solução ótima que possua o valor mínimo de todas as funções objetivo, simultaneamente, seja praticamente impossível de ser obtida. Ao invés disto, existem soluções de melhor compromisso possível para todos os objetivos. Agora, têm-se subsídios suficientes para serem dados alguns conceitos fundamentais em Otimização Multiobjetiva:

Definição 3.1: Dominância. Dada $f_i(\bullet)$, $i=1,2,...,k$ funções objetivo para serem analisadas, se $f_i(x_b) \leq f_i(x_a)$ para todo i e se existe j, $j \in \{1,2,...,k\}$, tal que $f_j(x_b) < f_j(x_a)$, então x_b domina x_a ou, apenas, $x_b \succ x_a$ (fig.3.1).

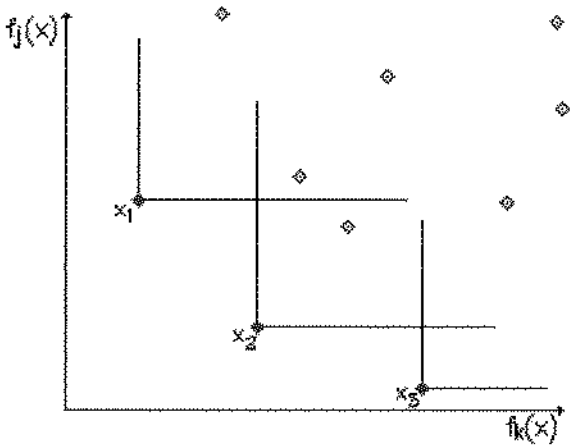
Figura 3.1: Exemplo de dominância entre soluções. Exemplo de dominância entre duas soluções, x_a e x_b, onde x_b domina x_a segundo os objetivos j e k.



Definição 3.2: Solução Eficiente. Um ponto (solução) $x_e \in S$ é eficiente se e somente se não existir um outro $x \in S$ tal que $f_i(x) \leq f_i(x_e)$ para todo $i \in K=\{1,2,...,k\}$ e $f_i(x) < f_i(x_e)$ para pelo menos um $i \in K$ (fig.3.2).

Na literatura, um ponto (ou vetor) no espaço de objetivos Z que corresponde a um ***ponto eficiente*** no espaço de variáveis de decisão S é chamado de ***solução não-dominada***. Em outras palavras, um ponto em S é *eficiente* se e somente se seu vetor de objetivos for *não-dominado*.

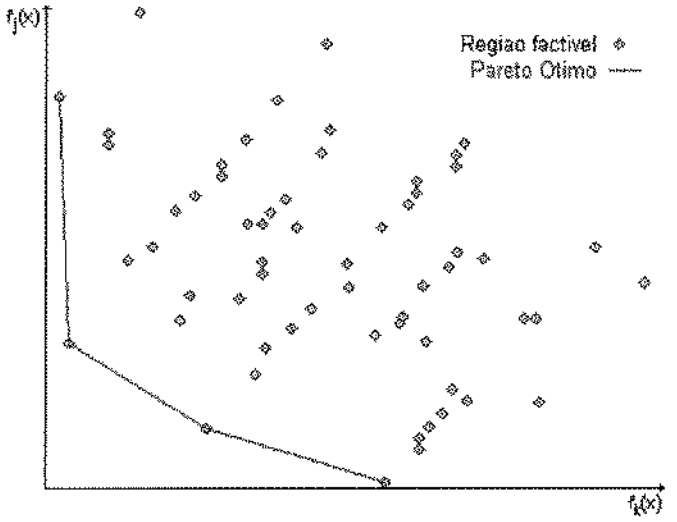
Figura 3.2: Exemplo de soluções não-dominadas. Supondo que o conjunto de soluções apresentado seja a região factível no espaço de objetivos de uma instância de um problema, x_1, x_2 e x_3 são as soluções não-dominadas, considerando os objetivos k e j. Tais soluções não são dominadas por nenhuma outra solução do problema.



O conjunto de soluções não-dominadas é um subconjunto da região de soluções factíveis. A principal característica de um conjunto de soluções não-dominadas é que para cada solução x_a fora deste conjunto (mas ainda dentro da região factível), sempre existe um ponto do conjunto que domine x_a [Wan93].

Uma solução eficiente é também conhecida como solução do **Pareto Ótimo**. As soluções pertencentes ao Pareto Ótimo dominam qualquer outra solução para o problema e não são dominadas por ninguém (fig.3.3).

Figura 3.3: O Pareto Ótimo de uma região factível. Supondo que o conjunto de soluções mostradas seja a região factível de uma instância de um problema, o Pareto Ótimo é o sub-conjunto de soluções factíveis formado por todas as soluções não-dominadas existentes.



O nome Pareto é devido a ***Vilfredo Pareto***, engenheiro de formação que interessou-se pelos estudos econômicos tornando-se um nome consagrado nas áreas de Economia e Sociologia. Pareto desenvolveu a ***Teoria da Utilidade Econômica*** ou ***Ofemilidade*** e em seguida estabeleceu as ***curvas de Pareto***, onde ele afirma que uma análise econômica deve ser feita segundo vários pontos de vista distintos, onde para cada um pode existir uma solução diferente mas de igual importância. Tais estudos foram publicados em 1986 [Par96].

O principal enfoque da Otimização Multiobjetivo, portanto, é a busca pelo conjunto de soluções eficientes - Pareto Ótimo - do problema. Na prática, um sub-conjunto do Pareto Ótimo e soluções próximas do Pareto Ótimo já são satisfatórios.

Neste trabalho, o Pareto Ótimo será representado pelo símbolo θ.

3.6. Métodos para a Resolução de Problemas Multiobjetivos

Os métodos para resolução de problemas com múltiplos critérios se propõem a otimizar o processo de tomada de decisão, dado que mediante um conjunto de critérios há de se considerar um conjunto de soluções possíveis, dentre as quais pode se escolher a melhor sob o ponto de vista de um *agente de decisão*, ou então, selecionar as que possuem os melhores compromissos segundo todos os critérios envolvidos.

O Agente de Decisão (*Decision Making - DM*) pode ser um indivíduo ou um sistema com conhecimentos específicos sobre o problema que está sendo resolvido e que pode decidir se uma determinada alternativa (solução) é a desejada ou não.

Seguindo a classificação de problemas dada na seção anterior, os métodos de resolução serão divididos em dois grandes grupos: os métodos para a resolução de problemas discretos, e, os métodos para a resolução de problemas de programação matemática.

3.6.1. Métodos para Resolução de Problemas Discretos

A questão principal que envolve a resolução de Problemas Discretos é fornecer ao Agente de Decisão mecanismos eficientes para a escolha da alternativa de sua preferência, dentre o conjunto de alternativas não-dominadas, já que eliminar as soluções dominadas em tais problemas não é uma tarefa difícil, pois o conjunto de soluções é pequeno e, geralmente, previamente conhecido.

Nenhum dos métodos descritos nesta seção precisam de um modelo matemático do problema, sendo a entrada de todos eles, um conjunto de alternativas (soluções) para o problema a ser resolvido. A questão principal na resolução de tais métodos é lidar com o alto grau de subjetividade envolvida.

Existem duas grandes escolas no que diz respeito à Análise Multicritério envolvendo Problemas Discretos. A Escola Francesa (ou Européia) e a Escola Americana. A Escola Francesa enfatiza o estudo e desenvolvimento de metodologias onde as preferências pessoais dos agentes de decisão tenham uma menor influência na alternativa escolhida, já a Escola Americana, em contrapartida, visa o desenvolvimento de métodos cuja atuação de um agente de decisão é fundamental na escolha da alternativa final. Pela ***Escola Francesa*** destaca-se a Família **ÉLECTRE** [Roy85] e pela ***Escola Americana*** o método mais conhecido é o **AHP** [Saa80]. Estes e alguns outros métodos são listados a seguir.

No caso de métodos que manipulam grandes conjuntos de alternativas (soluções) e poucos critérios, podem ser citados:

- **SCP** (Sequencial Choice Procedure), desenvolvido por Larichev, Mechitov e Moshkovich [LMM90];
- **AIM**, desenvolvido por Lofti, Stewart e Zionts [LSZ92];
- **VIMDA**, desenvolvido por Korhonen [Kor88].

No caso de métodos que manipulam *critérios com incerteza* (valores dos critérios não definidos precisamente), encontram-se:

- **PCPDA**, desenvolvido por Kirkwood e Van der Felz [KV86].

No caso de métodos que manipulam critérios não especificados explicitamente (critérios não conhecidos):

- Família **ELÉCTRE** - I, II, III e IV - (ELimination Et Choix Traduisant de la REalité) , desenvolvido primeiramente por Roy [Roy85];
- Família **PROMETHEE** (I, II, III, IV e V), desenvolvido por Brans, Marechal e Vincke [BMV84];
- **ORESTE**, desenvolvido por Roubens [Rou82];

- **PRIAM** (PRogramme utilisant l'Intelligence Artificielle en Multicriteria), desenvolvido por Levine e Pomerol [LP86].

No caso de métodos que manipulam *grandes conjuntos de critérios*, citam-se:

- **AHP** (Analytic Hierarchy Process), desenvolvido por Saaty [Saa80];
- **Chernoff Faces**, desenvolvido por Chernoff [Che72].

3.6.2. Métodos para Resolução de Problemas de Programação Matemática

No processo de tomada de decisão algum tipo de preferência pode ser dada pelo agente de decisão. Esta preferência pode ser vista, por exemplo, como o ato de privilegiar um determinado objetivo ou um subconjunto dos objetivos do problema.

A classificação dos métodos apresentada nesta seção refere-se aos pertencentes à Análise de Decisão Multiobjetivo - MODM, onde todos necessitam de um modelo matemático do problema a ser resolvido, iniciando com um conjunto ou apenas um ponto da região factível de decisão.

Dar-se-á a seguir, uma classificação sucinta baseando-se somente em *quando* a intervenção de um Agente de Decisão se torna necessária.

3.6.2.1. Métodos que não necessitam da avaliação de um Agente de Decisão

Estes métodos são aplicados diretamente à definição do problema. O agente de decisão não interfere no processo de resolução, nem antes do processo (manipulando os critérios ou restrições), nem durante o processo (interagindo com o método durante a resolução) e nem depois do processo de resolução, onde neste caso, *uma única solução* é fornecida como resultado, não cabendo ao agente de decisão interferir neste resultado. Exemplo: Métodos de Critério Global.

Métodos do Critério Global

Os Métodos do Critério Global tentam encontrar um vetor ótimo que otimize algum critério global. Podem ser aplicados para Problemas de Programação Linear - PPL e para Problemas de Programação Não-Linear - PPNL (somente objetivos não-lineares) [Haw80].

A seguir, é dado um exemplo de um critério global, consistindo na somatória das p-ésimas potências dos desvios relativos dos critérios em relação aos pontos eficientes (Eq.3.5):

$$\min_{\{x\}} F_p = \left[\sum_{j=1}^{k} \left(\frac{f_j(x^*) - f_j(x)}{f_j(x^*)} \right)^p \right]^{1/p} \qquad \text{Eq. 3.5}$$

Neste caso, se $p=1$ o problema é um PPL e se $p=2$ a função passa a ser quadrática, tornando-se um PPNL.

Tal método possui três fases distintas:

1ª FASE: Obtenção das soluções ótimas para cada critério;

Nesta fase resolve-se *k* PPLs, onde *k* é o número de critérios envolvido, com o método SIMPLEX [BJS90] por exemplo, onde se determina a solução para cada função objetivo envolvida, que são analisadas individualmente (como se fosse uma seqüência de problemas com uma só função objetivo);

2ª FASE: Construção de uma *Tabela de Compensação*;

Nesta fase é construída uma *tabela de compensações* ou *cruzamentos*, comparando a solução obtida para cada um dos objetivos com todas as outras soluções dos outros objetivos (ver eq.3.5);

3ª FASE: Obtenção de uma solução de *preferência*.

Por fim, é resolvido o critério global com base nos valores obtidos nas fases anteriores. É garantido que a solução de preferência resultante é uma solução eficiente (solução pertencente ao Pareto Ótimo).

3.6.2.2. Métodos que necessitam da avaliação do Agente de Decisão *antes* da execução

Estes métodos possibilitam que o agente de decisão forneçam informações de preferência, antes do processo de execução ser iniciado. Os métodos desta classe fazem uso de algum tipo de informação que favorece algum ou alguns dos objetivos envolvidos.

Nesta classe de método encontram-se as duas abordagens mais encontradas na prática. Em ambas, os múltiplos objetivos do problema são transformados num só, seja através do estabelecimento de uma única função como combinação de todas as funções objetivo do problema, seja através da priorização de um do objetivos e a transformação dos objetivos restantes em restrições do problema. Exemplos: Métodos de Função Utilidade; *Goal Programming*; e, Métodos Lexicográficos.

Métodos de Função Utilidade

A chamada ***Teoria da Utilidade Multiatributo*** (Multiatribute Utility Theory - MAUT) é aplicada em quase todos os métodos disponíveis na literatura oMur92, HN93, STo93, ZYW92]. Este é o motivo principal para ser dada uma ênfase maior aos métodos que fazem uso de uma função utilidade.

Pode-se representar matematicamente tais métodos da seguinte forma:

$$\begin{aligned} &min\ U(f(x)) = U(f_1(x), f_2(x), ..., f_k(x)) \\ &t.q.\ \ g_i(x) \geq 0 \quad , \quad i = 1,2,...,m \end{aligned} \qquad \text{Eq. 3.6}$$

onde *U(f)* é a função utilidade de múltiplos objetivos. Estes métodos necessitam da expressão de *U(f)* antes de resolverem o problema multiobjetivo. Esta abordagem é similar ao uso de funções utilidade na *teoria do consumo*, na área econômica. A função utilidade representa as preferências de um agente de decisão. Contudo, a tarefa de achar uma função utilidade satisfatória não é trivial até mesmo para os problemas mais simples [Haw80]. Para problemas muito complexos a definição de uma função utilidade pode ser uma tarefa impossível. A maior vantagem de seu uso é que, garantidamente, a solução encontrada é a que realmente se desejava encontrar antes do processo ser iniciado.

Entretanto, as desvantagens são inúmeras, pois, nada garante que o que se desejava no início é o melhor dentre todas as soluções possíveis.

A função utilidade *U(f)* pode ser definida de várias formas, sendo que a mais usada em métodos para resolução de problemas multiobjetivos é a que faz uso de ponderação dos objetivos, ou seja, são atribuídos pesos w_j para indicar a importância de cada objetivo. Então:

$$\begin{aligned} &min\ U = \sum_{j=1}^{k} w_j f_j(x) \\ &t.q.\ \ g_i(x) \geq 0 \quad , \quad i = 1,2,...,m \end{aligned} \qquad \text{Eq. 3.7}$$

A solução obtida através do uso desta função utilidade pode estar contida ou não no conjunto de soluções eficientes (Pareto Ótimo) do problema [Ste86, Haw80].

Goal Programming

Goal Programming - GP foi proposta por Charnes e Cooper [CC61] para um Problema de Programação Linear (alguns consideram que o primeiro trabalho foi o de Charnes, Cooper e Ferguson). Esta é talvez a abordagem mais antiga dentro do campo de Análise de Decisão Multicritério - MCDM. Tal abordagem requer que o agente de decisão atribua metas para cada um dos objetivos do problema. Uma solução de preferência é então definida como sendo a que minimiza os desvios em relação ao conjunto de metas pré-definidas.

Goal Programming foi originalmente formulada baseada na abordagem de *pesos não-preemptivos* ou abordagem de Arquimedes, sendo em seguida desenvolvida também tendo como base a abordagem lexicográfica, ou de *pesos preemptivos* [Rom86]. A abordagem de Arquimedes é denominada de *Weighted Goal Programming (WGP)* e a abordagem preemptiva tem sido denominada de *Lexicographic Goal Programming* (*LGP*).

Será apresentada uma formulação baseada na abordagem de pesos preemptivos (LGP). O Agente de Decisão, em adição ao conjunto de objetivos do problema, fornece também uma ordem de prioridades aos mesmos. Então:

$$\begin{aligned} &min\,[P_1 h_1(d^-,d^+), P_2 h_2(d^-,d^+),..., P_l h_l(d^-,d^+)] \\ &t.q.\ \ g_i(x) + d_i^- - d_i^+ = b_i \quad , \quad i = 1,2,...,m \\ &\qquad f_j(x) + d_j^- - d_j^+ = b_j \quad , \quad j = 1,2,...,k \\ &\qquad d_i^- - d_i^+ \geq 0 \quad \forall i \\ &\qquad d_j^- - d_j^+ \geq 0 \quad \forall j \end{aligned} \qquad \text{Eq. 3.8}$$

onde b_j, $j=1,2,...,k$ são as metas fornecidas pelo agente de decisão para os objetivos do problema; d_i^- e d_i^+ são, respectivamente, os valores máximo e mínimo da i-ésima meta; $h_i(d^-,d^+)$, $i=1,2,...,l$ são funções lineares das *variáveis de desvio* e são chamadas de *funções de realizações*. Os P_i's são pesos preemptivos, ou seja, $P_i >>> P_{i+1}$. Isto implica que nenhum número W, por maior que seja, pode tornar $WP_{i+1} > P_i$. Se qualquer uma das funções $f_j(x)$ e $g_i(x)$ vier a ser não-linear, então a situação passa a *Non-Linear Goal Programming*.

O primeiro passo deste método é minimizar a primeira função de realização (função com prioridade mais baixa), $h_1(d^-,d^+)$, onde min $h_1=h_1^*$. Em seguida $h_i(d^-,d^+)$ é minimizada, mas em nenhuma circunstância o valor de h_2 poderá ser maior do que o de h_1, lembrando que o problema é de minimização. Isto significa que uma função de realização com uma prioridade mais baixa (h_1) nunca poderá ter o valor menor (minimização) do que uma função de realização com mais alta prioridade (h_2). Este processo continua até que $h_i(d^-,d^+)$ seja minimizada.

3.6.2.3. Métodos que necessitam da avaliação do Agente de Decisão *durante* a execução (Métodos Interativos)

Esta classe de métodos é conhecida como Métodos Interativos, onde um Agente de Decisão interage com o método durante todo o processo de resolução do problema. Tais métodos permitem uma progressiva definição das preferências do agente de decisão na região factível definida pelos objetivos.

Os Métodos Interativos são os mais desenvolvidos atualmente, e a tendência é que isto aumente ainda mais com a exploração de poderosos computadores para um apelo visual maior, onde a interface gráfica será um fator preponderante. Alguns exemplos são: STEM; Método de Zionts e Wallenius; Método de Geoffrion; Método PARETO RACE; e, Método do Ponto de Referência/Direção de Referência.

A seguir são detalhados os métodos STEM e Zionts-Wallenius, por serem dois dos mais difundidos métodos interativos em programação matemática multiobjetiva. São descritos também os métodos PARETO RACE e Ponto de Referência/Direção de Referência por serem recentes e considerados os mais relevantes, principalmente na prática.

STEM

O método ***STEP Method*** ou **STEM** foi desenvolvido por Benayoun, Montgolfier e Tergny [BMT71] e permite ao agente de decisão aprender a reconhecer *boas* soluções e a importância relativa de cada objetivo. Neste método, fases de computação são alternadas com fases de decisão.

O STEM foi o primeiro procedimento interativo de grande repercussão para problemas de programação linear e é conhecido também como um método de redução da região factível [Ste86] porque em cada iteração a região factível é reduzida através da escolha do vetor de critérios que está mais próximo (em relação à métrica L_∞ - *Tchebycheff metric*) ao vetor de objetivos ideal $z^* \in R^k$ (o vetor ideal é formado pelos valores contidos na diagonal principal da tabela de compensações).

O STEM pode ser descrito como segue:

1ª FASE: Construção de uma *tabela de compensação*;

O problema multiobjetivo é dividido em k sub-problemas monobjetivos, um para cada objetivo e estes sub-problemas de programação linear são otimizados individualmente. A tabela de compensações é construída para se obter o ***vetor ideal*** $z^* \in R^k$ As soluções resultantes iniciam a tabela de compensações. Nesta tabela de compensações ou cruzamentos, é feita uma comparação entre a

solução obtida para cada um dos objetivos e todas as outras soluções para os outros objetivos. As linhas da tabela são os vetores de objetivos resultantes da otimização individual de cada um dos objetivos;

2ª FASE: Fase de computação;

Em cada iteração *m*, uma solução factível é obtida de tal forma que seja a mais próxima possível, no sentido MINIMAX [BMT71], da solução ideal.

3ª FASE: Fase de decisão.

A solução de compromisso obtida na fase de computação é apresentada ao agente de decisão, que compara-a com a solução ideal (vetor de objetivos ideal). Se algum dos objetivos possuir valores satisfatórios e outros não, o agente de decisão deve relaxar um dos objetivos satisfatórios de maneira a permitir, na próxima iteração, uma melhoria em um ou alguns dos objetivos cujos valores não são satisfatórios. Volta-se para a fase de computação até todos os objetivos possuírem valores satisfatórios [Haw80].

Método de Zionts-Wallenius

O ***método de Zionts-Wallenius*** ou **Z-W** foi proposto por Zionts e Wallenius [WZ76]. Tal método assume que todas as funções objetivo do problema são côncavas e que as restrições formam um conjunto convexo (funções não-lineares são linearizadas) [Haw80]. Pode ser utilizada uma função composta por uma combinação de todos os objetivos implicitamente definida ou uma base interativa.

Para o caso da abordagem através de uma função utilidade linear tal método pode ser descrito em cinco fases:

1ª FASE: Definir pesos para os objetivos;

O primeiro passo é escolher um conjunto arbitrário de pesos (ou multiplicadores) para os objetivos;

2ª FASE: Gerar uma função utilidade e resolver o problema monobjetivo resultante;.

Gerar uma função objetivo composta (função utilidade) usando os pesos definidos na primeira fase. A função utilidade é então otimizada produzindo uma solução não-dominada para o problema;

3ª FASE: Selecionar um conjunto de variáveis eficientes;

A partir do conjunto de variáveis não-básicas, um sub-conjunto de variáveis eficientes é selecionada (uma variável é eficiente quando sua introdução na base não pode aumentar o valor de um objetivo sem causar um decremento no valor de pelo menos um dos outros);

4ª FASE: Fase de decisão (interação com o agente de decisão);

Para cada variável eficiente um conjunto de compromissos é definido de tal forma que alguns objetivos são incrementados e outros reduzidos. Estes compromissos são apresentados ao agente de decisão que deve decidir se os compromissos estabelecidos são desejáveis ou não;

5ª FASE: Definir um novo conjunto de pesos para os objetivos.

A partir da decisão tomada na fase anterior, um novo conjunto de pesos (multiplicadores) é definido e retorna-se a segunda fase até que uma solução

ótima em relação à função utilidade implicitamente definida pelo agente de decisão ser encontrada.

PARETO RACE

O método ***PARETO RACE*** foi desenvolvido por Korhonen e Wallenius [KW88], sendo baseado na idéia de parametrização de uma função escalar previamente determinada (denominada pelos autores de função de realização), tornando possível projetar a busca na direção da fronteira eficiente (Pareto Ótimo), ao invés de num único ponto. O agente de decisão realiza a procura entre as soluções eficientes do problema.

Esta idéia foi implementada no sistema VIG, recebendo o nome de PARETO RACE [KMW92]

Abordagem de Ponto de Referência/Direção de Referência

Esta idéia foi apresentada muito recentemente por Karainova, Korhonen, Narula, Wallenius e Vassilev [Kar95].

Esta abordagem, juntamente com a de Murthy [Mur92], eram as únicas com relevância prática para Problemas de Programação Inteira.

Na verdade, foram desenvolvidos dois métodos e proposta a utilização conjunta dos mesmos. O primeiro método é baseado na idéia de *ponto de referência*, desenvolvida por Wierzbicki [Wie80] e um método que gera somente soluções inteiras. Tal método busca uma solução inteira que otimize o valor de uma função escalar previamente determinada (função de realização). O segundo método é baseado na *direção de referência* encontrada no método ***PARETO RACE*** (ver descrição acima) para encontrar soluções contínuas e então identificar uma solução inteira que esteja mais próxima da solução contínua corrente em relação à função escalar de realização.

1º MÉTODO: Método Inteiro

1º PASSO: Achar uma solução inteira eficiente através da otimização de um dos objetivos do problema;

2º PASSO: Especificar o vetor de níveis de aspiração $g \in R^k$ para os objetivos do problema (pelo agente de decisão), de tal forma que $g_i - q_i^l < 0$, para pelo menos um $i \in k$. Se o agente de decisão não quiser diminuir o nível de aspiração de pelo menos um objetivo, o método termina. Caso contrário, o processo continua;

3º PASSO: Resolver o problema de programação linear inteira mista. Pode ser usado um algoritmo aproximado ou um branch-and-bound. Se o agente de decisão estiver satisfeito com a solução, o método termina. Caso o agente de decisão não esteja satisfeito, retorna-se ao passo 2.

2º MÉTODO: Método Contínuo/Inteiro

1º PASSO: Estipular limites inferiores e superiores para os valores dos objetivos (pelo agente de decisão), definindo um vetor de pesos para os mesmos;

2º PASSO: Especificar o vetor de níveis de aspiração $g \in R^k$ para os objetivos do problema;

3º PASSO: Resolver um problema de programação linear paramétrico. Neste passo é usado o método PARETO RACE para achar a solução de preferência do problema (segundo o agente de decisão);

4º PASSO: Achar a solução inteira mais próxima da solução ótima, que seja de compromisso para um problema de programação inteira resultante. Se o agente de decisão estiver satisfeito, o procedimento termina e caso contrário, retorna-se ao passo três.

O primeiro método é menos eficiente, porém tem a vantagem de permitir que o agente de decisão manipule somente soluções inteiras, ao contrário do segundo método que é mais eficiente computacionalmente mas faz com que o agente de decisão lide a maior parte do tempo com soluções contínuas.

Os autores afirmam que os dois métodos acima são complementares e propõem uma aplicação conjunta dois a dois através de um Sistema de Suporte à Decisão Multicritério (*Multiple Criteria Decision Support System* - MCDSS). Então:

MÉTODO HÍBRIDO (Ponto de Referência/Direção de Referência)

1º PASSO: Permitir ao agente de decisão *caminhar* pelo conjunto de soluções não-dominadas do problema contínuo usando o método de PARETO RACE (método 2) até uma solução de preferência ser estabelecida;

2º PASSO: Encontrar uma solução inteira que seja a mais próxima possível em relação à solução contínua corrente;

3º PASSO: O agente de decisão decide se a solução inteira encontrada é satisfatória ou não. Se for, fim de execução;

4º PASSO: Se o agente de decisão conseguir perceber que está próximo de uma solução de preferência, vá para o passo cinco. Senão, retornar ao primeiro passo usando a solução contínua corrente como ponto inicial (base inicial);

5º PASSO: O agente de decisão continua com o método 1 até que seja encontrada uma solução inteira de preferência. Pode retornar-se ao passo 2 usando o vetor de níveis de aspiração para a solução inteira corrente.

Deve-se manipular o problema contínuo o máximo de tempo possível, e só passar para uma solução inteira quando uma solução inteira de preferência estiver próxima de ser alcançada.

3.6.2.4. Métodos que necessitam da avaliação do Agente de Decisão *após* serem aplicados

Os métodos desta classe determinam um subconjunto, mediante o conjunto completo de soluções eficientes (não-dominadas ou do pareto ótimo) estipuladas *depois* da resolução do problema de minimização do vetor. Deste subconjunto, o tomador de decisão irá escolher àquela que melhor satisfizer as relações de compensação que o mesmo estabelece. Exemplos: Métodos Paramétricos; Métodos de Pesquisa Adaptativa; Métodos ε-restrição; e, Times Assíncronos.

Método Paramétrico ou Ponderado

No Método Paramétrico ou Ponderado *(Parametric* ou *Weighting Method)* assume-se que a importância relativa dos k objetivos do problema seja conhecida e constante. A solução de preferência, neste caso, é uma solução eficiente (não-dominada) obtida através da resolução do seguinte sistema:

$$\begin{aligned} &min \sum_{i=1}^{k} w_i f_i(x) \\ &t.q.\ x \in S \end{aligned} \qquad \text{Eq. 3.9}$$

onde $w_i \geq 0$ são os coeficientes ponderados que representam a importância relativa dos objetivos do problema. Os w_i geralmente são normalizados tal que:

$$\sum_{i=1}^{k} w_i = 1 \qquad \text{Eq. 3.10}$$

Este método pode ser usado para a geração de soluções não-dominadas através da estipulação de diferentes valores para w, embora tal suposição de utilidades lineares e aditivas não seja facilmente satisfeita.

3.6.2.5. Métodos que utilizam Lógica Fuzzy

A ***Lógica Fuzzy*** é uma técnica para a resolução de problemas com grande aplicabilidade prática, especialmente nas áreas de controle e análise de decisão onde existe a presença de informação imprecisa, qualitativa, incompleta ou ambígua [Rom95]. É considerada como uma reaproximação entre a lógica clássica e a imprecisão do mundo real.

Lógica Fuzzy é um superconjunto (ou generalização) da lógica convencional (Booleana), estendido para suportar o conceito de ***verdade parcial***, que englobam os valores entre o *completamente verdadeiro* e o *completamente falso*.

Foi ***Lofti A. Zadeh*** que em 1965 publicou ***Fuzzy Sets*** (traduzido por alguns como ***Conjuntos Nebulosos***), como uma tentativa de modelar a incerteza das linguagens naturais. Existe também a linha desenvolvida pelos japoneses, onde ***Sugeno*** é o nome de maior expressão. Desta forma, através da incorporação do conceito de ***níveis de verdade***, a lógica fuzzy estende a lógica tradicional de duas formas distintas. Na primeira, segundo Zadeh, os conjuntos são nomeados qualitativamente (usando-se termos lingüísticos como quente, morno, frio, alto, mediano, etc) e são atribuídos ***níveis de pertinência*** aos elementos destes conjuntos, como por exemplo, um homem com 1.75m e outro com 1.85m podem pertencer ao conjunto de *homens altos*, contudo, o homem de 1.85m possuirá um nível de pertinência maior. Na segunda maneira, segundo Sugeno, não são os conjuntos que são fuzzy e sim os relacionamentos ou mapeamentos entre um elemento e tais conjuntos. Por exemplo, dados três tipos de doenças - gripe, bronquite e pneumonia - , cada uma representando um conjunto fuzzy, onde uma mulher pode apresentar sintomas de mais de uma doença (conjuntos fuzzy) simultaneamente e, neste caso, é o diagnóstico ou mapeamento dos sintomas que é fuzzy. Na linha de Zadeh os limites entre os conjuntos fuzzy se confundem (a interseção entre eles não é vazia) e na linha de Sugeno os conjuntos fuzzy não se interceptam mas os elementos podem pertencer a conjuntos fuzzy distintos.

Uma função de pertinência fuzzy atua sobre o intervalo de números reais [0.0,1.0]. A seguir é dado um exemplo de um sistema fuzzy:

FULPAL

O método **FULPAL** (*Fuzzy Linear Programming based on Aspiration Levels*) foi desenvolvido por Rommelfanger [Rom90]. É baseado numa nova interpretação da relação de desigualdade "$\tilde{\leq}$", onde cada restrição fuzzy é trocada por uma desigualdade e uma função objetivo fuzzy (função utilidade). Inclui também o procedimento para resolução de problemas de programação linear com restrições *soft* proposto por Zimmermann, em 1978. É aplicado à problemas de programação linear onde os coeficientes das restrições e/ou das funções objetivo podem ser fuzzy. O procedimento é interativo e controlado por níveis de aspiração para os objetivos.A seguir são apresentados os passos principais de tal método:

1º PASSO: Especificar um índice de tolerância $\varepsilon \in [0,1[$;

2º PASSO: Estipular os valores iniciais dos coeficientes das restrições e funções objetivo do problema, baseando-se no valor de tolerância do passo anterior;

3º PASSO: Transformar as funções objetivo numa função utilidade fuzzy [Rom90] e resolver o programa linear;

4º PASSO: Especificar o valor crítico para a função de pertinência (*membership*) [Rom95] e os limites de tolerância para os objetivos e restrições;

5º PASSO: Fixar os limites inferiores e superiores para os níveis de aspiração e em seguida, especificar os níveis de aspiração;

6º PASSO: Determinar uma função de pertinência (por exemplo, *piecewise linear* [Ste86]) e resolver o sistema LP resultante;

7º PASSO: Se o agente de decisão aceitar a solução resultante, o procedimento termina. Caso contrário, retorna-se ao passo cinco.

3.6.3. Times Assíncronos

O método de **Times Assíncronos** (*Asynchronous Teams* ou A-Teams) é o utilizado neste trabalho e, como abordado no capítulo anterior, ele é um método para resolução de Problemas de Otimização Combinatória.

Neste trabalho, A-Teams é apresentado como sendo também um método para resolução de Problemas Multiobjetivos, como poderá ser observado daqui em diante, onde serão utilizados os conceitos descritos neste capítulo para serem incorporados no A-Team de forma a tornar possível nossa proposição.

Podem ser incorporadas novas vantagens, às já citadas no capítulo 2, para A-Teams, agora canalizado para as características de problemas multiobjetivos:

- Gera como resultado múltiplas soluções;
- Possibilita a geração de soluções de compromisso para todos os objetivos envolvidos (todos os objetivos são considerados igualmente);
- Se desejado, prioriza um ou mais objetivos do problema para a geração de soluções (através de funções utilidade, por exemplo);

- É constituído por múltiplos algoritmos, onde cada algoritmo pode manipular um objetivo específico ou, ainda, um subconjunto de objetivos;
- Possui uma implementação relativamente rápida e fácil;
- Pode ser trivialmente transformado num método interativo, permitindo assim a incorporação das preferências de Agentes de Decisão em tempo de execução.

3.6.3.1. A-Teams x Métodos para Problemas de Programação Matemática

Todos os métodos para problemas de programação matemática citados necessitam da formulação matemática sofisticada do problema para a geração de soluções. No caso de problemas NP-difíceis, isto significa que muito provavelmente a performance dos mesmos não será satisfatória, caso dos Problemas Combinatórios Multiobjetivos. Já A-Team, por ser um método aproximado, não necessita de tal formulação matemática para a geração de soluções e, mesmo assim, possibilita a obtenção de boas soluções para o problema num curto espaço de tempo.

Além disso, dentre os métodos citados, apenas o de Karainova *et al* mostrou ter relevância prática na resolução de Problemas de Programação de Inteira. Isto já elimina todos os outros, restando apenas este último para ser comparado com A-Teams.

O método de ponto/direção de referência, apesar de sempre gerar uma solução do Pareto Ótimo, não tem garantia de tempo de execução e gera apenas uma solução de cada vez. A-Teams possibilita a obtenção de boas soluções para o problema num curto espaço de tempo e gera múltiplas soluções em cada execução, possibilitando um leque de opções. Ainda, na abordagem de Karainova *et al*, os objetivos não são considerados simultaneamente na busca por melhores soluções, se baseando na otimização de objetivos individualmente e, no melhor caso, numa função escalar. A-Teams, como poderá ser comprovado neste trabalho, possibilita a consideração simultânea de todos os objetivos do problema na busca por melhores soluções, fornecendo sempre soluções de compromisso para todos eles. Vale ressaltar, que A-Teams também suporta a resolução de problemas multiobjetivos otimizando-o através de funções escalares.

3.7. Resumo

O objetivo deste capítulo foi apresentar uma visão abrangente do que pode ser encontrado na literatura, em termos teóricos e práticos, sobre **Problemas Multiobjetivos**. Um breve histórico e uma classificação geral de tais problemas foram mostrados.

Os principais conceitos envolvendo **Otimização Multiobjetiva** foram abordados, onde a busca pelo conjunto de soluções eficientes, ou **Pareto Ótimo** do problema, é a principal questão.

Uma classificação dos métodos de resolução foi apresentada, com uma descrição dos principais métodos. Uma breve comparação entre o método utilizado neste trabalho, o de Times Assíncronos, com os principais métodos encontrados na literatura foi dada.

A relevância deste capítulo se deve ao fato de que o problema abordado neste trabalho, além de ser um Problema de Otimização Combinatória (ver Cap.2), é um

Problema Multiobjetivo: um **Problema de Otimização Combinatória Multiobjetivo** (ver Cap.4).

3.8. Notas Bibliográficas

Como principais fontes de consulta envolvendo Problemas Multiobjetivos e seus métodos de resolução podem ser citados o livro de Steuer [Ste86], o de Sarawagi, Nakayama e Tanino [SNT85] e o de Zeleny [Zel82]. Ainda, Goicochea, Hansen e Duckstein [GHD82].

Para Análise de Decisão Multicritério (*Multicriteria Decision Making* - MCDM) em geral, podem ser destacados: Dyer *et al* [Dye92]; Gardiner e Steuer [GS92]; Olson e Courtney [OC91]; Wallenius [Wal91]; Shin e Ravidran [SR91]; White [Whi90]; Aksoy [Aks90]; Vanderpooten e Vincke [VV89]; Lofti e Teich [LT91]; Moskowitz e Bunn [MB87]; Yu [Yu85]; Hwang e Masud [HM79]; e, Ignizio [Ign76]. Korhonen, Moskowitz e Wallenius [KMW92] apresentam um revisão bibliográfica sobre o assunto.

Especificamente para Análise de Decisão Multiatributo (*Multiattribute Decision Making* - MADM), que envolvem problemas de pequeno porte, podem ser destacados: Korhonen, Moskowitz, Salminen e Wallenius [Kor91]; Corner e Kirkwood [CK91]; Korhonen [Kor86]; e, Keeney e Raiffa [KR76].

Para Análise de Decisão Multiobjetivo (*Multiple Objective Decision Making* - MODM) podem ser destacados: Szidarovszky, Gershon e Duckstein [SGD86]; Sengupta [Sen85]; Chankong e Haimes [CH83]; Murthy [Mur92]; Hwang, Paidy e Yoon [Haw80]; e, Zeleny [Zel76].

Entre a literatura envolvendo Programação Matemática Multiobjetivo (*Multiple Objective Mathematical Programming* - MOMP) de um modo geral, podem ser destacados: Armand e Malivert [AM91]; [KR76]. Para Programação Matemática Interativa podem ser citados os trabalhos de: Karainova [Kar91]; Wallenius [Wal91]; Troutt, Clinton e Hemming [TCH91]; Shin e Ravindran [SR91]; White [Whi90]; Korhonen, Siljamäki e Wallenius [KSW90]; e, Saaty [Saa80]. Especificamente para Problemas de Programação Linear Multiobjetivos, podem ser destacados: Steuer e Choo [SC83]; Wierzbicki [Wie80]; Korhonen e Laakso [KL86]; Lewandowski *et al* [Lew89]; e, Korhonen e Wallenius [KW88]. E, para Problemas de Programação Não-Linear Multiobjetivos, podem ser citados: Kreglewski, Paczynski, Granat e Wierzbicki [Kre89]; e, Roy e Wallenius [RW91].

Para os métodos que utilizam a abordagem de Lógica Fuzzy, podem ser citados os trabalhos de: Rommelfanger [Rom95, Rom92, Rom91]; o trabalho de Kim, Moskowitz, Dhingra e Evans [Kim91]; e, o de Hauser e Clausing [HC88].

A resolução de Problemas Multiobjetivos através de Algoritmos Genéticos vem sendo intensamente pesquisada no *Illinois Genetic Algorithms Laboratory*. O trabalho de Horn, Nafpliotis e Goldberg [HN93] pode ser citado como referência.

Existe pouco material, teórico e prático, abordando Problemas de Programação Inteira Multiobjetivos e a maioria envolve problemas com apenas dois objetivos - caso bicritério. Zionts [Zio79] apresentou um esquema de classificação para tais problemas.

Bitran [Bit77, Bit79], Rasmussen [Ras86] desenvolveu alguma teoria e um algoritmo para os problemas inteiros multiobjetivos de variáveis 0-1. Shapiro [Sha76], Bitran e Lawrence [BL80], Clímaco e Martins [CM80], Gabbani e Magazine [GM86], Karwan, Zionts e Vilarreal [KZV81, VK81a, VK81b], Burkard, Krarup e Pruzan [Bur81, BKP82], Bitran e Rivera [BR82], Klein e Hannan [KH82], Decko e Winkofsky [DW83], Kiziltan e Yucaoglu [KY83], Steuer e Choo [SC83], Henig [Hen85], Chalmet, Lemonidis e Elzinga [CLE86], Gabbani e Magazine [GM86], Ramesh, Karwan e Zionts [RKZ86, RKZ89, RKZ90], Katoh [Kat89], Azevedo e Martins [AzM91], Karainova [Kar91], Shin, Ramachandran e Bullington [SRB92], Murthy [Mur92], e, Karainova, Korhonen, Narula, Wallenius e Vassilev [Kar95], são os trabalhos mais significativos envolvendo tais problemas. Uma coletânea de métodos em Programação Inteira Multiobjetiva foi elaborada por Teghem e Kunsch [TK85, TK86].

Podem ser citados, também, os trabalhos desenvolvidos por Rodrigues e Souza para Problemas de Otimização Combinatória Multiobjetivos [RS94, RS95a, RS95b, RS95c, RS96a, RS96b].

Capítulo 4

O Problema do Caixeiro Viajante com Múltiplas Distâncias

"Um homem viaja pelo mundo em busca daquilo que necessita e retorna à sua casa para encontrá-la."
George Moore.

O **Problema do Caixeiro Viajante com Múltiplas Distâncias** (*Multi-Distance Traveling Salesman Problem* - MDTSP [ST93b]), é um problema proposto como generalização do clássico NP-difícil Problema do Caixeiro Viajante (*Traveling Salesman Problem* - TSP [DFJ54, LLKS85]), onde ao invés de apenas uma matriz, existem várias matrizes de custo. Cada matriz de distância pode ser considerada como um objetivo do problema. Trata-se, portanto, de um Problema de Otimização Combinatória com Múltiplas Funções Objetivo.

Este capítulo é todo dedicado a este novo problema, onde, primeiramente, serão abordados os conceitos, definições e algoritmos para resolução do problema original, o TSP, que se tornam necessárias para um melhor entendimento do MDTSP, abordado em seguida.

Algoritmos para o MDTSP, que incorporam conceitos de dominância e decisão de compromisso, foram desenvolvidos e são apresentados neste capítulo.

4.1. O Problema do Caixeiro Viajante

Um caixeiro viajante quer visitar um determinado conjunto de cidades, cada cidade exatamente uma única vez, terminando a visita na mesma cidade que começar. E, além disto, ele quer que sua trajetória seja a menor possível.

Este é o Problema do Caixeiro Viajante (*Traveling Salesman Problem* -TSP) que é um dos problemas mais estudados da área de Otimização Combinatória. Sua enorme importância vem principalmente do fato de servir de *modelo* para inúmeros outros problemas combinatórios.

O problema é de otimização, porque se tenta minimizar (poderia ser maximizar) a distância total, e é combinatório, porque a escolha deve ser feita sobre o conjunto discreto de todas as trajetórias válidas para o problema.

O TSP é o primeiro problema NP-difícil descrito no livro *Computers and Intractability* [GJ79] e continua exercendo enorme influência no desenvolvimento de novos conceitos em otimização, bem como no desenvolvimento de novos algoritmos: novas técnicas de resolução de problemas vem sendo desenvolvidas para o TSP, ou, pelo menos, sendo a ele primeiramente aplicadas para mostrar suas eficácias. A afirmação *"tão difícil quanto o TSP"* é uma das mais utilizadas quando novos problemas NP-difíceis são propostos, fazendo do TSP um padrão de comparação.

4.1.1. Contexto Histórico

Caixeiro viajante era a designação dada aos profissionais liberais que trabalhavam com vendas até o início deste século. Eles compravam mercadorias em grandes centros, e em seguida, percorriam cidades interioranas para venderem seus produtos a pequenos comerciantes. Hoje em dia este termo não é mais usado para denominar profissão semelhante, mas, no meio matemático, ficou em evidência por denominar o Problema do Caixeiro Viajante.

Nesta seção será dada uma visão geral, desde o surgimento deste problema na literatura até suas aplicações mais recentes.

O primeiro indício do registro de um TSP é o encontrado num livro publicado na Alemanha em 1832 e intitulado *"O Caixeiro Viajante, como ele deve ser e o que ele deve fazer para conseguir comissões e sucesso nos negócios. Por um veterano Caixeiro Viajante"* [LLKS85]. Apenas um século mais tarde, em torno da década de 1930, que o termo *traveling salesman problem* foi utilizado no contexto matemático. Tal feito é atribuído a Merrill Flood, que fez uma conexão entre o TSP e um problema de roteamento de uma frota de ônibus escolares [LLKS85]. Flood introduziu o TSP na RAND Corporation, cuja boa imagem e reputação no mercado foram fatores determinantes para a grande popularização do TSP. O outro forte motivo pelo grande interesse pelo *novo* problema vinha do fato de que importantes problemas combinatórios já conhecidos, tais como o *assignment problem* e o *transportation problem*, eram similares, mas, o TSP demonstrava ser mais difícil de se resolver. Tais diferenças introduzidas com o TSP intrigaram e atraíram o interesse de muitos pesquisadores.

O TSP foi formulado como um Problema de Programação Inteira pela primeira vez por Dantzig, Fulkerson e Johnson em 1954 [DFJ54], que também foram os pioneiros na resolução de *grandes* instâncias do TSP (maior instância com 48-cidades), sendo este um dos principais eventos na história da otimização combinatória. A formulação usada por eles foi a seguinte:

Definição 4.1: Variáveis inteiras x_{ij} indicam se uma cidade j é visitada depois de uma cidade i ou não, respectivamente, ($x_{ij} = 1$) ou ($x_{ij} = 0$). A distância da cidade i para cidade j é dado por c_{ij}. Então:

$$min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \qquad \text{Eq. 4. 1}$$

$$t.q. \sum_{i=1}^{n} x_{ij} = 1 \quad , \quad j = 1,2,...,n \qquad \text{Eq. 4. 2}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \quad , \quad i = 1,2,...,n \qquad \text{Eq. 4. 3}$$

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1 \qquad \text{Eq. 4. 4}$$

$$x \in \{0,1\} \qquad \text{Eq. 4. 5}$$

onde n é o número de cidades do problema, S é um subconjunto próprio do conjunto $\{1,2,...,n\}$, e o símbolo "| |" denota a cardinalidade do conjunto. A equação 4.1 representa a formulação geral do problema, ou seja, indica que se deseja minimizar o somatório das distâncias dentre todas as conexões utilizadas entre cidades. Agora, as outras três formulações representam as restrições do problema, ou seja, as equações 4.2 e 4.3 garantem que toda cidade deve ter uma conexão chegando de outra cidade e, uma conexão saindo para outra cidade. A desigualdade 4.4 garante que nenhuma subtrajetória poderá ser criada. Em 4.5 está sendo restringido os valores de todas as variáveis para 0 ou 1.

Dantzig, Fulkerson e Johnson não somente resolveram algumas instâncias práticas do TSP, o que até então era impossível, como também puderam mostrar que os conceitos de cortes no plano eram relevantes para a resolução de problemas de programação inteira em geral. Considera-se que eles também foram os primeiros a usarem conceitos de branch-and-bound.

Vários trabalhos posteriores, como o de Lin e Kernighan [LK73], que desenvolveram a melhor e mais conhecida heurística para o TSP - o algoritmo de Lin-Kernighan, o de Chvátal [Chv73] que deu início ao tratamento do politopo do TSP, os dois artigos de Grötschel e Padberg [GP79a, GP79b], o artigo de Grötschel [Grö80], o de Padberg e Hong [PH80], o artigo de Crowder e Padberg [CP80], como também o de Grötschel e Padberg [GP85], todos eles de extrema importância na definição de novas desigualdades e teoremas para um melhor conhecimento da estrutura poliedral do TSP.

Padberg e Rinaldi [PR87], que introduziram o termo branch-and-cut, obtiveram resultados importantes para o TSP, onde foi apresentada a solução ótima para a famosa instância do TSP simétrico de 532-cidades. Ainda, em 1991, eles encontraram a solução ótima para um TSP com 3000-cidades [PR91].

Johnson [Joh90] obteve trajetórias a menos de 2% do ótimo para instâncias com até 10^5- cidades.

Em 1993, Naddef e Rinaldi [NR93] desenvolveram um importante trabalho onde mostraram que os resultados obtidos para o GTSP (Graphical Traveling Salesman Problem [JRR94]) obtidos anteriormente por Cornuéjols, Fonlupt e Naddef [CFN85], podiam ser extendidos ao TSP, devido ao forte relacionamento entre os politopos destes problemas.

Uma outra abordagem muito recente aplicada a grandes instâncias do TSP é o método de geração de colunas ou *branch-and-price* [Bar94] (ver 2.4.1.5). Inúmeros outros artigos foram publicados envolvendo o TSP, com interessantes resultados, que podem ser encontrados em [JRR94, LLKS85].

Souza, em 1993, aplicou o método multi-algorítmico de Times Assíncronos ao TSP, onde também conseguiu obter a solução ótima para a mesma instância de 532-cidades [Sou93]. Tal trabalho é de extrema importância para esta dissertação, pois o método usado aqui foi o mesmo e o problema a ser tratado é uma generalização do TSP.

4.1.2. Definição

Novas formulações inteiras para o TSP vêm sendo desenvolvidas, fato que já possibilitou a obtenção de resultados significativos. Entretanto, a abordagem utilizada neste trabalho é a aproximada e, neste caso, o tipo de formulação do problema não é crucial para a qualidade das soluções obtidas, como seria se a abordagem fosse a exata.

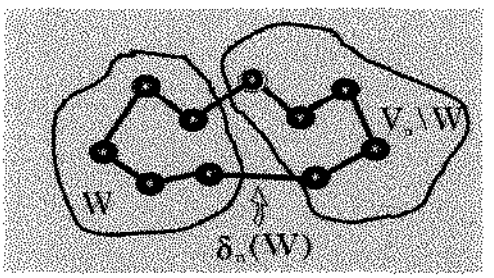
Apenas para um melhor entendimento da estrutura do TSP e, conseqüentemente, do novo problema proposto, será dada a seguir uma das formulações mais comuns em programação inteira do TSP, sendo que, primeiramente, alguns conceitos básicos tornam-se necessários:

Seja $K_n = (V_n E_n)$ um grafo completo não-orientado com $n = |V_n|$ vértices e $m = |E_n|$ arestas. Uma aresta e com extremos u e v é denotada por (u,v).

Denota-se por $\Re^{E_n}$ o espaço de vetores reais cujos componentes são indexados pelos elementos de E_n. Então, para qualquer conjunto de arestas $F \subseteq E_n$ e para qualquer $x \in \Re^{E_n}$, $x(F)$ denota a soma $\Sigma_{e \in F} x_e$.

Para um sub-conjunto próprio de $W \subset V_n$, $E_n(W) \subset E_n$ denota $\{uv \in E_n | u,v \in W\}$ e $\delta_n(W) \subset E_n$ denota $\{uv \in E_n | u \in W, v \in V_n \setminus W\}$. O conjunto $\delta_n(W)$ é o conjunto de arestas que *unem* o conjunto W e o conjunto $V_n \setminus W$ (fig.4.1).

Figura 4. 1: Relacionamento entre os conjuntos V_n, W e $\delta_n(W)$.



O conjunto de soluções para o TSP é o conjunto H_n de todos os ciclos Hamiltonianos de K_n. Um ciclo Hamiltoniano (um ciclo visitando cada vértice exatamente uma vez) é um subgrafo $H=(V_n,E)$ de K_n satisfazendo os seguintes requisitos:

(a) Todos os vértices de H possuem grau 2;

(b) H é conexo.

O conjunto de arestas de um subgrafo de K_n onde todos os vértices têm grau 2, é um *2-matching perfeito*, ou seja, uma coleção de ciclos disjuntos simples de pelo menos 3 vértices, onde cada vértice de K_n pertence a algum destes ciclos. Conseqüentemente, um ciclo Hamiltoniano pode ser definido como um *2-matching perfeito e conexo*.

Todo *2-matching perfeito conexo* é, também, *biconexo*, já que é necessário remover pelo menos 2 arestas para desconectá-lo. Portanto, os requisitos (a) e (b) podem ser trocados por:

(c) Todos os vértices de H possuem grau 2;

(d) H é biconexo.

Para todo $H \in H_n$, é associado um único vetor de incidência $\chi^H \in \Re^{E_n}$, tal que:

$$\chi_e^H \begin{cases} 1 & \text{se } e \in H \\ 0 & \text{c.c.} \end{cases} \qquad \text{Eq. 4. 6}$$

O vetor de incidência de todo ciclo Hamiltoniano satisfaz o seguinte sistema de equações:

$$A_n x = 2 \qquad \text{Eq. 4. 7}$$

onde A_n é a matriz incidência vértice-aresta de K_n e *2* é um *n-vetor* cujos componentes são todos iguais a 2. As equações do tipo $A_n x = 2$ são chamadas de equações de grau (***degree equations***) e transformam os requisitos (c) em termos algébricos. E mais, para qualquer conjunto não-vazio $S \subset V_n$ e para qualquer ciclo Hamiltoniano H de K_n, o número de arestas de H com um extremo em S e o outro em $V_n - S$ é pelo menos 2 (e par). Portanto, a interseção do conjunto de arestas de H com o conjunto $\delta_n(S)$ tem cardinalidade pelo menos 2 (e sempre par) e χ^H deve satisfazer o seguinte conjunto de desigualdades:

$$x(\delta_n(S))2 \qquad \forall\ \varnothing \neq S \subset V_n \qquad \text{Eq. 4. 8}$$

Estas desigualdades são chamadas de desigualdades de eliminação (***subtour elimination inequalities***) de sub-trajetórias porque elas não são satisfeitas pelo vetor de incidências de 2-matching desconexos (a união de duas ou mais trajetórias), e eles transformam os requisitos (d) em termos algébricos.

Então, dada uma função objetivo $c \in \Re^{E_n}$, que associa a cada aresta e de K_n um comprimento c_e, o Problema do Caixeiro Viajante consiste em encontrar um ciclo Hamiltoniano (um ciclo visitando cada vértice exatamente uma única vez) tal que seu c-comprimento (a soma dos comprimentos das arestas) seja o menor possível.

O TSP pode ser resolvido encontrando-se uma solução para o seguinte programa inteiro:

$$\min \; cx \qquad \text{Eq. 4. 9}$$

$$\text{s.a.} \;\; A_n x = 2 \qquad \text{Eq. 4. 10}$$

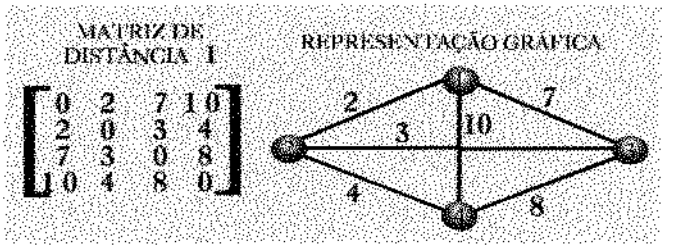
$$x(\delta_n(S)) \geq 2 \qquad \forall \;\; \varnothing \neq S \subset V_n \qquad \text{Eq. 4. 11}$$

$$x \in \{0,1\} \qquad \text{Eq. 4. 12}$$

Pode ser observado que é utilizado um grafo completo na definição do TSP. A razão disto é porque para tais grafos, a existência de uma solução factível é sempre garantida, enquanto para grafos genéricos, verificar a existência de um ciclo Hamiltoniano é NP-completo.

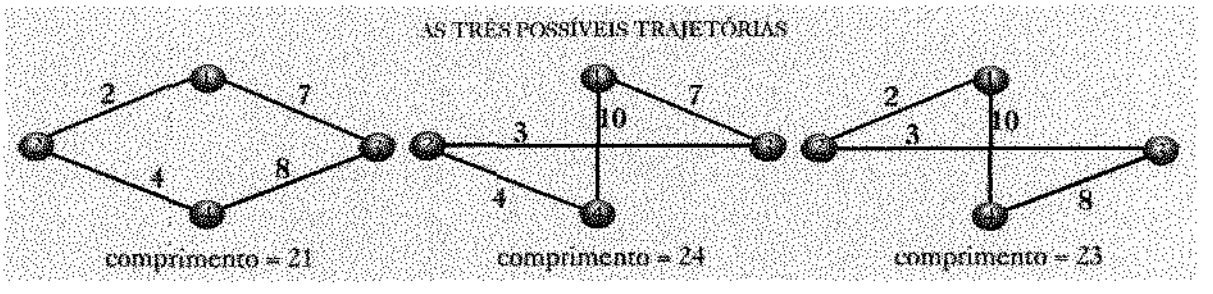
O número de ciclos Hamiltonianos em K_n, ou o tamanho do conjunto de soluções factíveis de um TSP assimétrico, é de $m=(n-1)!$. A figura (fig. 4.2) exemplifica uma instância de um TSP simétrico com 4-cidades (onde, $m=(n-1)!/2$, já que a distância entre duas cidades é a mesma nas duas direções).

Figura 4. 2: Exemplo de uma instância do TSP com 4-cidades.



Três são as trajetórias válidas para um TSP de 4-cidades - (4-1)!/2=3 . As três trajetórias possíveis deste exemplo são apresentadas a seguir, na fig.4.3, com seus respectivos comprimentos. A solução ótima, àquela cujo comprimento é o menor possível, é facilmente detectada, sendo a de comprimento 21 (fig.4.3).

Figura 4. 3: As três possíveis trajetórias para a instância do TSP com 4-cidades.



O TSP simétrico é freqüentemente referido somente por TSP [JRR94, LLKS85]. Neste trabalho, foi considerada a versão simétrica do TSP e, portanto, o termo TSP representará tal versão do problema.

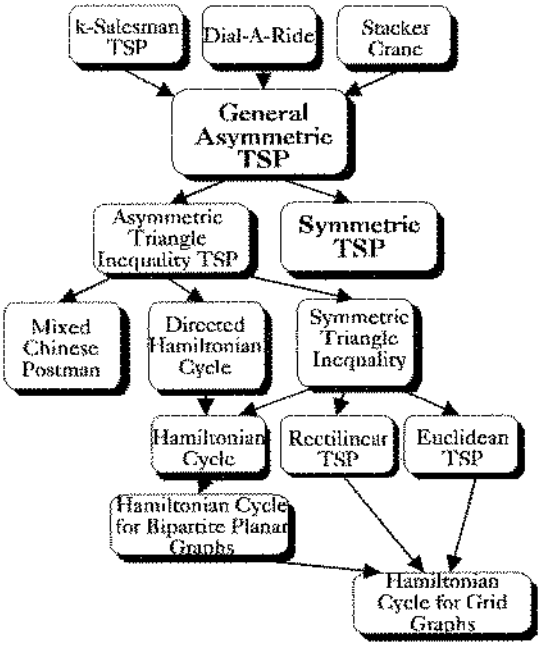
4.1.3. Complexidade

O TSP foi um dos primeiros problemas provado como sendo **NP-difícil** (*Non-deterministic Polinomial time*) por Karp em 1972 [Kar72], no início do desenvolvimento da Teoria da Complexidade Computacional [PS82, GJ79].

A versão de otimização do TSP (versão de interesse deste trabalho) é equivalente em complexidade à versão de decisão (verificar se existe ou não um ciclo Hamiltoniano de custo total menor ou igual que um determinado inteiro w), que é um problema NP-completo.

Abaixo (fig. 4.4), pode ser observada a posição do TSP dentro de uma hierarquia de casos especiais, com níveis crescente de especialização à medida que se dirige para a base da figura. O caso geral do TSP, TSP assimétrico, está próximo do topo e, no nível acima, encontram-se as generalizações mais conhecidas [LLKS85].

Figura 4. 4: Esquema do relacionamento do TSP e com suas principais generalizações e especializações.



4.1.4. Algoritmos para o TSP

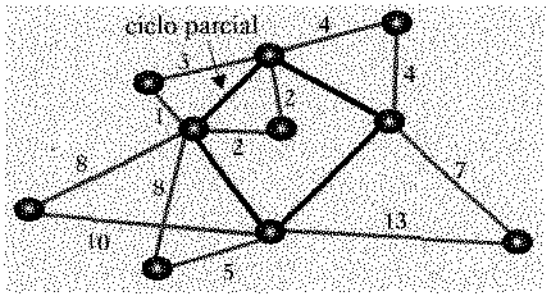
Existe um vasto número de algoritmos para a resolução do TSP, também, dentro de duas grandes classes: a classe dos métodos exatos e a classe dos métodos aproximados. Todos os métodos exatos citados no Cap.2 já foram aplicados para o TSP e tais aplicações podem ser facilmente encontradas na literatura [Cer96, JRR94, LLKS85]. Nesta seção, serão detalhados alguns algoritmos aproximados, dentro de três grandes sub-classes: construção, melhoria (busca local) e relaxação.

4.1.4.1. Heurísticas de construção

No caso do TSP, heurísticas de construção são aquelas que geram um ciclo Hamiltoniano de acordo com alguma regra de construção. Em outras palavras, uma trajetória é construída, cidade a cidade, até formar uma trajetória completa.

A fig. 4.5 apresenta um exemplo de um ciclo em construção (ciclo parcial), com os respectivos vértices que ainda não foram inseridos no ciclo, algumas arestas e seus respectivos pesos. No final desta subseção, é apresentado o resultado da aplicação dos algoritmos de construção a serem descritos, sobre tal exemplo fig. 4.5.

Figura 4. 5: Exemplo de um conjunto de cidades (vértices), onde destaca-se um ciclo parcial em construção e as possíveis ligações (arestas e seus respectivos comprimentos) entre cada cidade fora do ciclo e as duas cidades mais próximas pertencentes ao ciclo parcial.



Dentre as regras de construção, serão abordados alguns algoritmos que utilizam a regra de construção por inserção. Primeiramente, será dado o algoritmo genérico para tal regra:

Algoritmo Geral para Construção por Inserção de uma Trajetória

Selecionar um ciclo inicial com v vértices $v_1, v_2, ..., v_v$ $(v \geq 1)$ e o conjunto $W = V \setminus \{ v_1, v_2, ..., v_v \}$;
Enquanto $W \neq \emptyset$
  Selecionar um vértice $j \in W$ de acordo com algum critério.
  Inserir j em alguma posição no ciclo e faça $W = W \setminus \{j\}$.

Existem várias maneiras diferentes de implementar algoritmos de construção por inserção, e a diferença ocorre na seleção do novo vértice a ser inserido (vértice j). A seguir serão descritos três de tais algoritmos, onde será denominado *vértice do ciclo*

aquele vértice que já está contido na trajetória parcial. Para $j \in W$ é definido $d_{min}(j)=min\{c_{ij}/i \in V \setminus W\}$, ou seja, a distância mínima de um vértice é dada pela distância deste vértice ao vértice mais próximo a ele do ciclo parcial.

Escolhido o vértice, o mesmo é inserido no local que resulte o menor incremento no comprimento do ciclo parcial.

Nearest Insertion - NI

Insere o vértice que possui a menor distância para um vértice do ciclo, ou seja, seleciona $j \in W$ com $d_{min}(j)=min\{d_{min}(l)/l \in W\}$.

Furthest Insertion - FI

Insere o vértice cuja distância mínima para um vértice do ciclo é máxima, isto é, seleciona $j \in W$ com $d_{min}(j)=max\{d_{min}(l)/l \in W\}$.

Arbitrary Insertion - AI

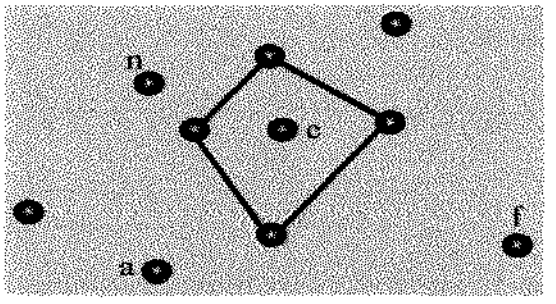
Também conhecido por ***Random Insertion*** seleciona o vértice a ser inserido randomicamente.

Cheapest Insertion - CI

Insere o vértice que causa o menor incremento no custo total do ciclo parcial.

A fig. 4.6 apresenta um possível resultado da aplicação de cada um dos quatro algoritmos de construção descritos nesta subseção - NI, AI, FI e CI -, sobre o grafo do fig. 4.5. O vértice próximo à letra *n* seria o escolhido pelo NI, o próximo à letra *f* seria o vértice escolhido pelo FI, o próximo à letra *c*, pelo CI e, o próximo à letra *a*, um possível vértice escolhido pelo AI.

Figura 4. 6: Comparação entre os algoritmos de construção baseados no esquema de inserção. Numa determinada iteração, dado o ciclo Hamiltoniano parcial (em construção) abaixo e os vértices restantes, o ***NI*** adicionaria o vértice *n* ao ciclo parcial, o ***FI*** adicionaria o vértice ***f***, o ***CI*** adicionaria o vértice ***c*** e o ***AI*** adicionaria o vértice ***a***.



A heurística *cheapest insertion* pode ser executada em tempo $O(n^2 log n)$. Todas as outras podem ser implementadas em tempo $O(n^2)$.

4.1.4.2. Heurísticas de melhoria

Os ciclos Hamiltonianos gerados pelas heurísticas de construção são de baixa qualidade. Embora eles possam ser úteis em algumas aplicações, como em A-Teams, em geral não são satisfatórios. Nesta subseção será abordada a questão de como melhorar estes ciclos gerados. Existem três abordagens de muito sucesso neste caso: *r-opt*, *or-opt* e *Lin-Kernighan*.

R-OPT

Dada uma trajetória válida, este algoritmo elimina ***r*** arestas desta trajetória, produzindo ***r*** segmentos desconectados. Então, *r-opt* reconecta todos os segmentos de todas as maneiras possíveis, gerando novas trajetórias. Se uma trajetória menor for obtida a qualquer momento, o algoritmo reinicia. O algoritmo pára se nenhuma combinação de *r* arestas eliminadas puder gerar uma trajetória menor do que a corrente.

Os dois algoritmos *r-opt* de maior sucesso são [LLKS85, JRR94]:

2-OPT

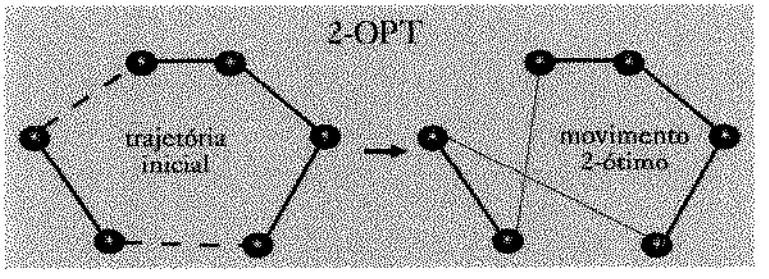
Um *movimento 2-opt* consiste na eliminação de duas arestas de uma trajetória e a adição de outras duas, para todas as combinações de *n*, *2* a *2*. O algoritmo *2-opt* inicia seu procedimento com uma trajetória (solução válida para o TSP) e a deixa com a **propriedade 2-ÓTIMA**, resultando numa nova trajetória *melhorada*.

Algoritmo 2OPT

Entrada: T (trajetória inicial), c (matriz de distância).
Saída: T melhorada.

1. Seja T o ciclo Hamiltoniano corrente;
2. Repita
  2.1. Selecione duas arestas $\in E^T$;
  2.2. Enquanto a troca de duas arestas por duas outras arestas $\in E \backslash E^T$, gerar T' válida, faça:
    2.2.1. Se ($c(T')<c(T)$)
      2.2.1.1. Faça $T=T'$;
até T satisfazer a propriedade *2-ÓTIMA*.
3. Retorne T.

Figura 4. 7: Exemplo de um possível *movimento 2-opt*.



3-OPT

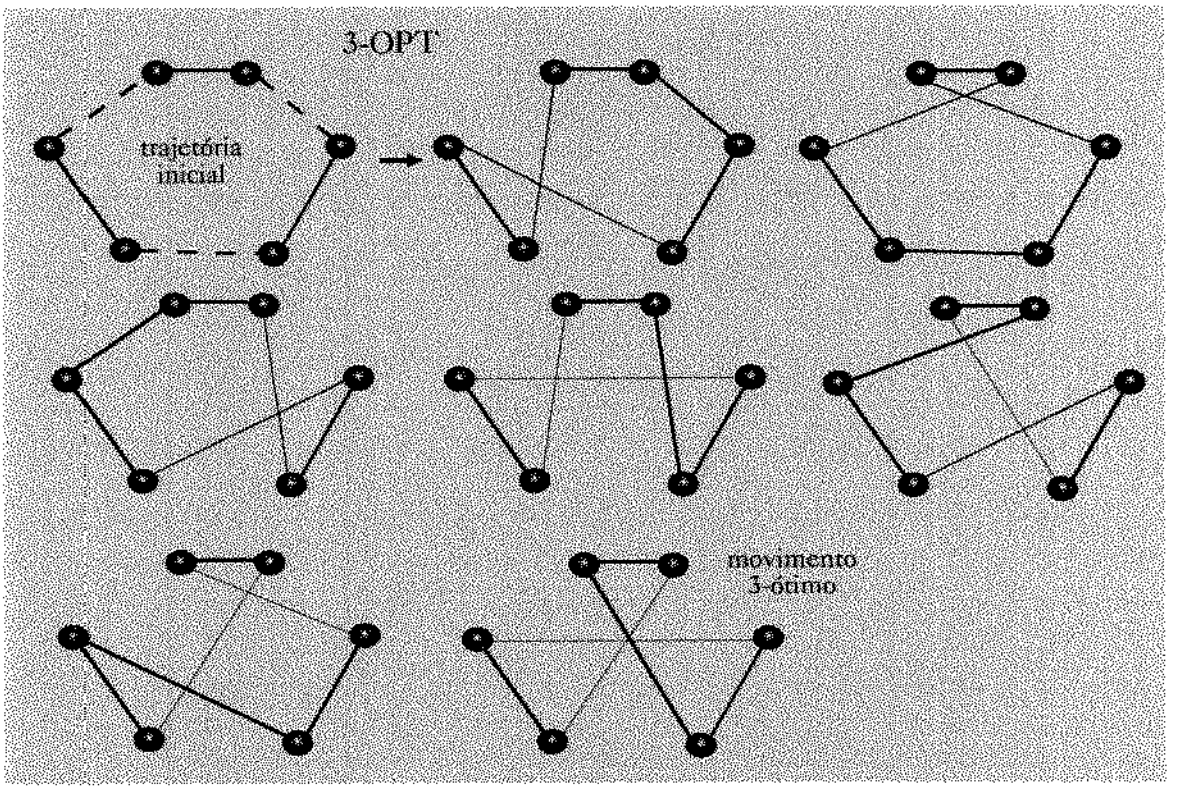
Um *movimento 3-opt*, por sua vez, retira três arestas de uma trajetória e insere outras três, para todas as combinações de *n*, *3* a *3*. O algoritmo *3-opt* também inicia seu procedimento com uma trajetória e a deixa com a **propriedade 3-ÓTIMA**, resultando numa nova trajetória *melhorada*.

Algoritmo 3OPT

Entrada: T (trajetória inicial), c (matriz de distância).
Saída: T melhorada.

1. Seja T o ciclo Hamiltoniano corrente;
2. Repita
  2.1. Selecione três arestas $\in E^T$;
  2.2. Enquanto a troca de três arestas por duas outras arestas $\in E \backslash E^T$, gerar T' válida, faça:
  seja válido, faça:
    2.2.1. Se ($c(T')<c(T)$)
    2.2.1.1. Faça $T=T'$;
até T satisfazer a propriedade *3-ÓTIMA*.
3. Retorne T.

Figura 4. 8: Exemplo de um possível *movimento 3-opt*.



OR-OPT

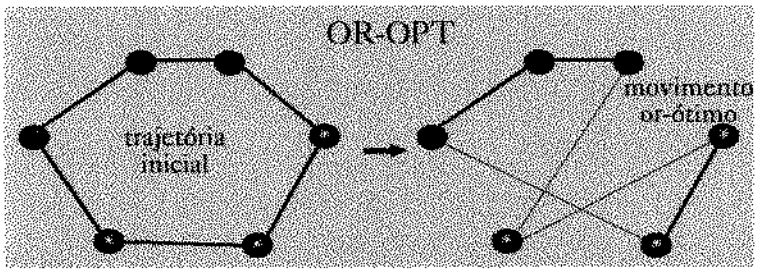
O algoritmo ***or-opt*** é um subconjunto do algoritmo *3-opt*. *Or-opt* pega uma seqüência de três cidades adjacentes e desloca a seqüência entre todas as cidades de uma dada trajetória. Se uma trajetória menor for encontrada, então o algoritmo reinicia. Caso contrário, pega-se uma seqüência de duas cidades e desloca-se estas duas cidades ao longo da trajetória. Novamente, se uma trajetória menor for encontrada o algoritmo reinicia. Se nem desta última forma se encontrar uma trajetória menor, pega-se um conjunto com somente *uma* cidade e a desloca entre todas as demais. O algoritmo termina quando *nenhuma* trajetória menor for encontrada, depois da execução do algoritmo para *todos* os conjuntos de 3, 2 ou 1 cidades adjacentes.

Algoritmo OROPT

Entrada: T (trajetória inicial), c (matriz de distância).
Saída: T melhorada.

1. Seja T o ciclo Hamiltoniano corrente.
2. Faça l=3 e *achou_melhor*=VERDADEIRO.
3. Repita
3.1. Repita
3.1. Selecione l vértices adjacentes em T;
3.2. Enqto a troca de l vértices adjacentes em T por toda seqüência de l vértices não for repetida e desde que T' gerado seja válido, faça:
3.2.1. Se ($c(T')<c(T)$)
3.2.1.1. Faça $T=T'$
3.2.3. Senão
3.2.1.1. Faça l--;
3.2.1.1. Faça *achou_melhor*=FALSO;
enqto *achou_melhor*=VERDADEIRO
enqto $l>0$.
4. Retorne T.

Figura 4. 9: Exemplo de um possível *movimento or-opt*, com a alteração da posição de três cidades adjacentes na trajetória.



Lin-Kernighan - LK

Esta heurística foi originalmente descrita por Lin e Kernighan em 1973 e sua motivação foi baseada em experiências obtidas em computação prática [LLKS85, JRR94].

A idéia é baseada na observação de que algumas vezes uma modificação que gere um pequeno incremento do comprimento da trajetória pode abrir novas possibilidades

para a obtenção de consideráveis melhorias. O princípio básico é construir modificações complicadas que são compostas por movimentos simples, onde nem todos estes movimentos necessariamente têm que diminuir o comprimento da trajetória. Para se obter um tempo de execução razoável, o esforço para se achar as partes dos movimentos compostos tem que ser limitado.

A heurística de *Lin-Kernighan* termina num ótimo local que dependerá da trajetória inicial e dos movimentos que serão executados. Johnson em 1990 sugeriu que depois de se aplicar a heurística de *Lin-Kernighan*, um movimento ***4-opt*** seja realizado e, então, novamente seja aplicada a heurística *Lin-Kernighan* [JRR94]. Usando este método várias soluções ótimas para grandes instâncias de problemas foram encontradas.

Algoritmo LK

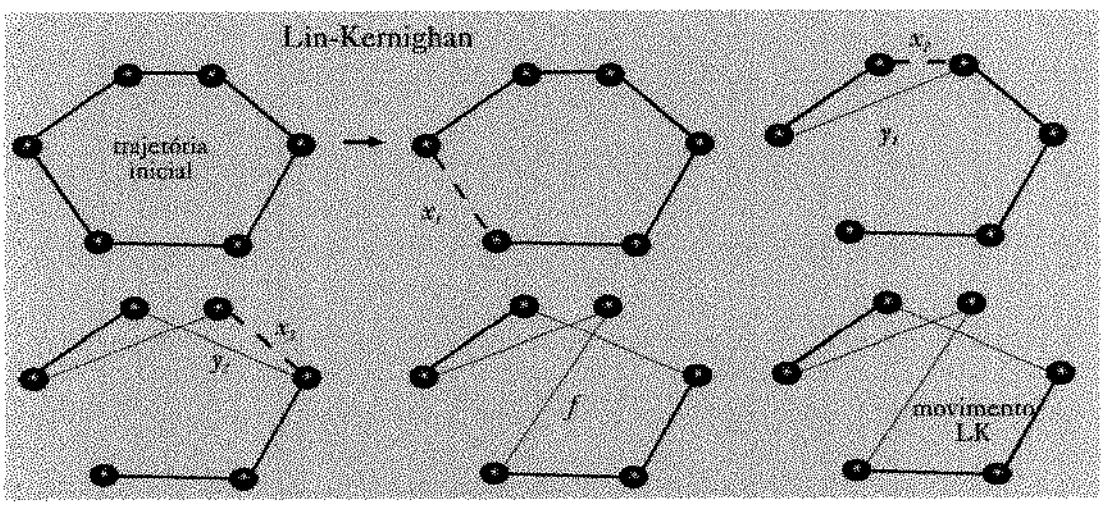
Entrada: T (trajetória inicial), c (matriz de distância).
Saída: T melhorada.

1. **Seja** T o ciclo Hamiltoniano corrente;
2. **Faça** *max_ganho_local*=0;
3. **Repita**
   - 3.1. **Selecione** um vértice $i \in V$;
   - 3.2. **Retire** uma das arestas $x_i \in E^T$;
   - 3.3. **Selecione** a aresta $y_i = y_1 \in E' = E \setminus E^T$;
   - 3.4. **Enqto** $E' \neq \varnothing$ faça:
     - 3.4.1. **Compute** $g_i = x_i - y_i$;
     - 3.4.2. **Se** *max_ganho_local* < g_i
       - 3.4.2.1. **Faça** *max_ganho_local* = g_i;
       - 3.4.2.2. **Faça** *y_escolhido* = y_i;
     - 3.4.3. **Selecione** $y_i \in E' = E \setminus E^T \setminus \{y_1, \ldots, y_{i-1}\}$;

     Fim_Enqto;
   - 3.5. **Troque** x_i por *y_escolhido* em T, gerando T';
   - 3.7. **Compute** $G = c(T) - c(T')$;

   enqto $G > 0$
4. **Retorne** T.

Figura 4. 10: Exemplo de um ***movimento LK***. No *passo 1*, a aresta x_1 é eliminada. No *passo 2* a aresta y_1 é incorporada e x_2 eliminada. No *passo3*, x_3 é eliminada e y_2 é incorporada. No *passo 4* uma aresta *f* é incorporada para gerar uma trajetória factível.



De um modo geral, observa-se que a qualidade das heurísticas diminui com o aumento do tamanho do problema.

4.1.4.3. Algoritmos de relaxação

Uma relaxação de um problema de otimização *M* é um outro problema *R* cujo conjunto de soluções factíveis de *R* contém todas as soluções factíveis de *M*. A função objetivo de *R* é, geralmente, uma extensão arbitrária da função objetivo de *M*. Isto faz com que o valor da solução ótima do problema *R* seja menor (para um problema de minimização) do que o valor da função objetivo de *M*. Se *M* for um problema NP-difícil e *R* pode ser resolvido eficientemente (pertence à classe P), o valor ótimo de *R* pode ser usado como um ***limite inferior*** num esquema de enumeração para resolução de *M*. Quanto mais próximo for o valor ótimo de *R* em relação a solução ótima de *M*, mais eficiente é o algoritmo de enumeração [JRR94].

A seguir são dados alguns algoritmos para a relaxação do TSP, segundo duas estratégias de abordagem: baseadas em árvores e baseadas em cortes no plano.

Baseadas em Árvores

São algoritmos baseados em subconjuntos de vértices (subgrafos) e que tentam satisfazer somente um subconjunto das restrições do problema original. A solução ótima pode ser encontrada em tempo polinomial.

1-TREE

Dado o grafo K_n, cujos *n* vértices representam as cidades do TSP, uma **1-Tree** pode ser obtida a partir de uma **árvore espalhada mínima** dos $K_n \setminus \{1\}$, ou seja, uma árvore espalhada composta de *n-1* vértices, juntamente com o vértice restante e as duas arestas adjacentes a ele. Gerar uma árvore espalhada mínima pode ser feita em tempo polinomial, e portanto, gerar uma *1-Tree* também leva tempo polinomial.

Acontece que esta relaxação não é muito *forte*, ou seja, os limites inferiores fornecidos são muito distantes do valor da solução ótima do problema original.

HELD-KARP

Held e Karp [HK70, HK71] propuseram uma modificação no algoritmo anterior de forma a gerar limites inferiores muito melhores. A modificação foi realizada na função objetivo do problema original, onde eles propuseram a adição de uma constante em todas as arestas do grafo.

O novo problema foi denominado de **Problema Dual Lagrangeano** (*Lagrangean Dual Problem*). Os limites inferiores para o TSP obtidos através da resolução deste problema ficaram conhecidos como **Limites de Held-Karp**.

Os limites inferiores obtidos desta maneira são considerados de alta qualidade e são muito usados nas aplicações atuais.

Baseadas em Cortes

As relaxações baseadas em corte são aquelas que desconsideram uma ou mais classes de restrições envolvidas na definição do problema original e depois inserem, uma a uma, as restrições ao problema, com a esperança de atingir a solução ótima com a inserção de poucas restrições.

Dentro desta classe, existem as relaxações conhecidas por **Relaxações Contínuas**, pois retiram a restrição de integralidade do TSP, onde todas as variáveis passam a ter um valor contínuo entre 0 e 1. As relaxações que mantém a condição de integralização são denominadas de **Relaxações Discretas**.

Relaxação de Sub-Trajetórias

Neste caso, além das restrições de integralidade do TSP, o problema relaxado não contém também as restrições de eliminação de sub-trajetórias. O processo é iniciado com a inserção de um pequeno sub-conjunto das restrições de eliminação de sub-trajetórias, em seguida resolve-se o programa linear e verifica-se se a solução ótima encontrada satisfaz todas as restrições ainda não inseridas. Caso alguma restrição tenha sido violada, adiciona-se as mesmas ao problema relaxado e resolve-se novamente o programa linear resultante. E assim sucessivamente, por um certo número de iterações, até que o limite inferior gerado seja satisfatório.

Os limites inferiores obtidos também são considerados de alta qualidade. Esta técnica permitiu muitos dos bons resultados obtidos para o TSP encontrados na literatura.

Assignment Problem ou 2-matching

A Relaxação *2-matching* desconsidera toda a classe de restrições de eliminação de sub-trajetórias, mas mantém a restrição de integralidade, portanto, é uma relaxação discreta. A resolução do problema relaxado pode ser obtida em tempo polinomial.

Fractional 2-Matching

Neste caso, além das restrições de sub-trajetórias desconsidera-se também a restrição de integralidade. Esta relaxação é freqüentemente a primeira a ser produzida nos algoritmos de cortes no plano, dado que é preciso somente um número muito pequeno de restrições (polinomial n) para resolvê-lo.

4.2. Uma Generalização do Problema do Caixeiro Viajante

Um caixeiro viajante quer visitar todo um determinado conjunto de cidades, cada cidade exatamente uma única vez, terminando a visita na mesma cidade que começar. E, neste caso, ele quer que ***todos*** *os custos envolvidos no percurso sejam os menores possíveis.*

Este é o Problema do Caixeiro Viajante com Múltiplas Distâncias (*Multi-Distance Traveling Salesman Problem* -MDTSP) que é uma generalização do clássico TSP. O MDTSP é um Problema de Otimização Combinatória Multiobjetivo. De otimização, devido à tentativa de se minimizar (poderia ser maximizar) todos os comprimentos da trajetória de acordo com as matrizes de distância envolvidas; é combinatório, devido à escolha ser feita sobre o conjunto discreto de todas as trajetórias válidas para o problema; e é multiobjetivo, devido à cada trajetória válida possuir k custos totais, cada um segundo uma matriz de distância do problema; não existindo necessariamente uma solução ótima e, sim, todo um conjunto de soluções eficientes - o Pareto Ótimo do problema.

4.2.1. Definição

De acordo com os conceitos definidos na subseção 4.1.2, o MDTSP pode ser definido da seguinte maneira:

Dada k funções objetivos $c^k \in \Re^{E_n}$, $k \geq 1$, que associa a cada aresta e de K_n um vetor de k comprimentos c_e^k, o Problema do Caixeiro Viajante com Múltiplas Distâncias consiste em encontrar um conjunto de ciclos Hamiltonianos tal que seus c^k-comprimentos (as k somas totais das arestas segundo as k matrizes de distância) sejam os menores possíveis. Desta forma, o MDTSP pode ser formulado como um problema de programação inteira:

$$\min \quad c^k x \qquad \text{Eq. 4.13}$$

$$\text{s.a.} \quad A_n x = 2 \qquad \text{Eq. 4.14}$$

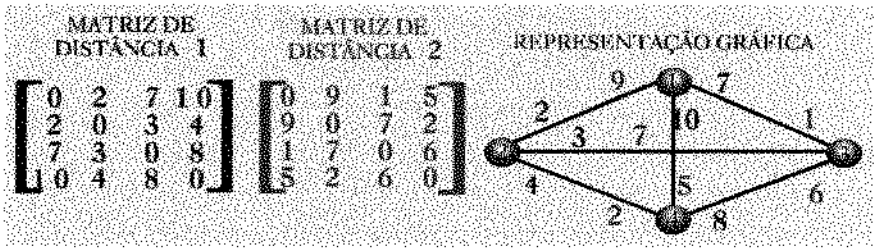
$$x(\delta_n(S)) \geq 2 \qquad \forall \quad \varnothing \neq S \subset V_n \qquad \text{Eq. 4.15}$$

$$x \in \{0,1\} \qquad \text{Eq. 4.16}$$

Também neste caso será considerada a **versão simétrica** do MDTSP, onde o caminho entre duas cidades, nas duas direções possíveis, possui a mesma distância. O número de ciclos Hamiltonianos em K_n de um MDTSP (o tamanho da região factível definida pelas variáveis de decisão), tal qual o TSP, é de $(n-1)!/2$, a diferença é que cada trajetória para o MDTSP possui k custos totais. Isto significa dizer que a *região factível no espaço de decisão* é a mesma, mas *a região factível no espaço de objetivos* passa a ser k-dimensional (ver Cap.3).

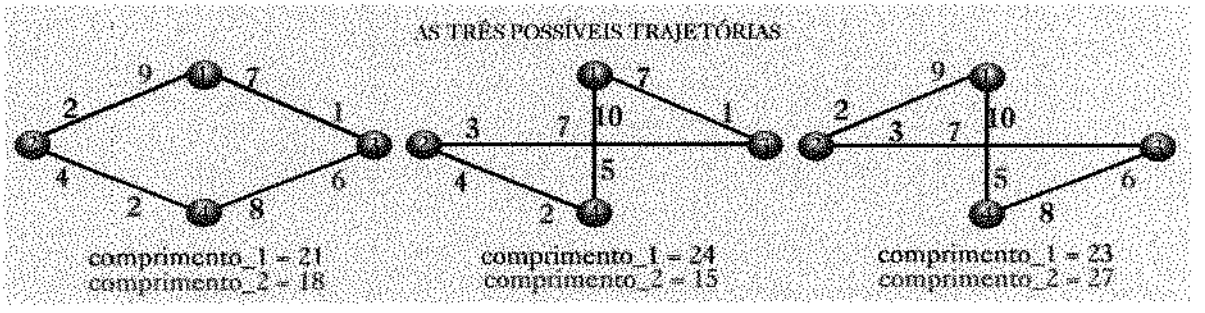
A figura (fig. 4.11) exemplifica uma instância de um MDTSP com 2 matrizes de distância (2-DTSP) e 4-cidades.

Figura 4. 11: Exemplo de uma instância do MDTSP com 4-cidades e 2 matrizes de distância (um 2-DTSP).



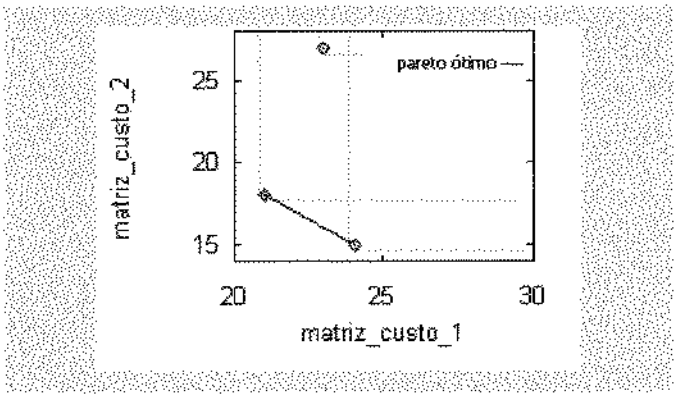
Neste exemplo, um MDTSP simétrico, três ((4-1)!/2=3) são as trajetórias válidas, cada uma com dois comprimentos totais, segundo cada uma das matrizes de distância (fig. 4.12).

Figura 4. 12: As três possíveis trajetórias para a instância do MDTSP com 4-cidades e seus 2 comprimentos segundo as 2 matrizes de distância.



Neste caso, determinar *qual* é a melhor solução dentre as três trajetórias não é tão trivial como no caso monobjetivo, onde a trajetória de menor comprimento é a escolhida. Cada trajetória possui dois comprimentos a serem considerados e a meta é a escolha daquele conjunto de trajetórias cujos comprimentos sejam eficientes entre si. A figura 4.13 apresenta o gráfico contendo as três soluções do problema, em que pode ser observado, claramente, o Pareto Ótimo do problema, representado no gráfico pela semi-reta unindo dois pontos (soluções). As soluções que fazem parte do Pareto Ótimo são as de comprimentos (24,15) e (21,18). A solução que possui os comprimentos (23,27) é dominada pela solução de comprimentos (21,18) e, portanto, não faz parte do conjunto de soluções eficientes.

Figura 4. 13: Pareto Ótimo, com 2 trajetórias, da instância do MDTSP com 4-cidades e 2 matrizes de distância.



No restante deste capítulo, as definições necessárias serão dadas com base nas definições apresentadas nesta subseção 4.2.1 e na subseção 4.1.2.

4.2.2. Complexidade

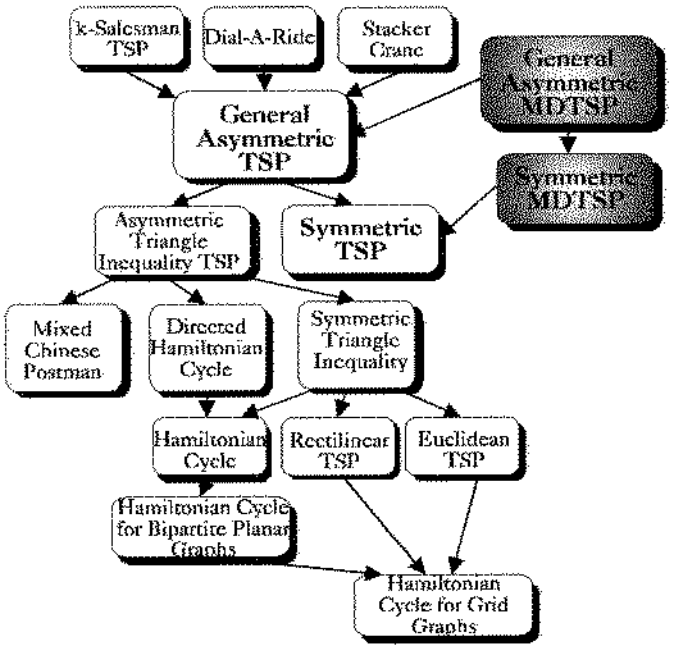
O MDTSP é uma generalização do clássico TSP e, portanto, pelo menos tão difícil de se resolver quanto o mesmo. Na sua versão de otimização, o MDTSP (tal como o TSP [LLKS85]) pertence à classe dos problemas NP-difíceis. A posição do MDTSP dentro do esquema de problemas correlacionados é mostrado a seguir (fig. 4.14).

Conforme pode ser observado no esquema da figura 4.14, o *General Asymmetric Multi-Distance Traveling Salesman Problem* representa o caso assimétrico e está relacionado como uma generalização do *General Asymmetric TSP*. Como um caso especial, encontra-se o *Symmetric MDTSP*, que é a versão utilizada neste trabalho e uma generalização do *Symmetric TSP*.

Podemos reduzir o MDTSP para o tradicional TSP, na versão simétrica ou assimétrica, como segue:

No caso do MDTSP, tem-se um vetor de custos de tamanho k associado a cada aresta e do grafo K_n, onde k é o número de matrizes de custo associadas ao problema. Reduzindo o MDTSP para o TSP, basta fazermos tal vetor de tamanho 1, ou seja, atribuir apenas um custo a cada aresta do grafo.

Figura 4. 14: Esquema de representação da hierarquia de problemas relacionados com o TSP, segundo a Teoria da Complexidade, incluindo o MDTSP assimétrico e simétrico.



4.2.3. Aplicações

Como no caso do TSP, inúmeros problemas podem ser modelados para o Problema do Caixeiro Viajante Multiobjetivo, o MDTSP. Este fato é ainda reforçado pelo ponto de vista dos defensores da modelagem multiobjetiva para problemas reais, os quais afirmam que a maioria dos problemas práticos possuem múltiplos objetivos a serem atingidos e que a modelagem de tais objetivos proporciona resultados ainda mais próximos dos desejados. Como para os Problemas Combinatórios, e suas naturezas intrincadas, soluções exatas não são possíveis de ser garantidas, qualquer melhoria na modelagem que ocasione uma redução das distorções entre o problema real e o modelado é de fundamental importância.

O objetivo desta seção é apresentar algumas aplicações do MDTSP, sendo que algumas, até então, eram modeladas como um TSP tradicional.

4.2.3.1. Problemas de Escalonamento

Problemas de escalonamento podem ser modelados como MDTSPs. Para um TSP, um problema de escalonamento com custo inter-tarefas, onde a ordem com que as tarefas são escalonadas apresentam custos distintos, pode ser modelado da seguinte maneira: as tarefas (*jobs*) se constituem nas cidades do problema e a ordem com que as mesmas são escalonadas se constituem no caminho a ser otimizado, ou seja, deseja-se encontrar a seqüência que envolve todas as tarefas com custo mínimo (se a seqüência não for cíclica, basta criar uma conexão de custo 0 entre a primeira e as demais tarefas da seqüência).

Geralmente tal modelagem é feita ignorando-se os demais custos inter-tarefas envolvidos. Entretanto, para problemas de escalonamento onde existem *vários* custos inter-tarefas, pode-se considerar todos eles para a obtenção da melhor seqüência modelando o problema como um MDTSP: as tarefas são as cidades do problema e a ordem de escalonamento das mesmas, o caminho a ser otimizado; sendo que para ser estabelecido um caminho desta vez, vários custos devem ser otimizados simultaneamente, como por exemplo, minimização do custo de produção, minimização do tempo de escalonamento e maximização da qualidade dos produtos.

4.2.3.2. Recondicionamento de Motor de Turbina à Gás

O *Problema de Recondicionamento de Motor de Turbina à Gás* (do inglês *Overhauling Gas Turbine Engines*) foi reportado pela primeira vez por Plante, Lowe e Chandrasekaran [PLC87] e ocorre quando o motor de turbina à gás de uma aeronave é revisado. Tal problema pode ser modelado como um MDTSP da seguinte maneira:

Durante a (re)montagem das turbinas, um número de palhetas devem ser fixadas sobre a circunferência, sendo que cada palheta possui características próprias e a colocação correta das palhetas pode resultar em benefícios substanciais. A disposição final das palhetas deve garantir, tanto quanto possível, um fluxo de gás uniforme, uma redução do consumo de combustível e uma redução da vibração do motor. Modelando para um MDTSP, as palhetas se constituem nas cidades do problema, onde o caminho é representado pela ordem de colocação das mesmas. Os objetivos do problema são:

maximização da uniformidade do fluxo de gás; minimização da vibração do motor; e, minimização do consumo de combustível.

4.2.3.3. Escalonamento Primário no *Mini-Mill Scheduling Problem*

O *Problema de Escalonamento de Mini-Engenhos* ou *Mini-Mill Scheduling Problem* foi apresentado por Souza e Favilla [SF96] e consiste num problema de escalonamento para o processo de fabricação de aço.

É sugerida a decomposição do problema em três partes: **agrupamento** (agrupamento dos pedidos); **escalonamento primário** (escalonamento de conjuntos de pedidos específicos - denominados de *heats* - em máquinas fortemente acopladas no tempo); e, **escalonamento final** (escalonamento de pedidos em máquinas não acopladas no tempo). Os objetivos a serem otimizados são: maximização da produtividade, minimização do inventário e minimização do atraso de entrega.

A segunda parte do problema, o escalonamento primário, foi modelada como sendo um MDTSP. Foi aplicado A-Teams (ver Cap.5) para a resolução do problema obtendo-se resultados relevantes, dando respaldo para novas aplicações deste método em grandes problemas industriais.

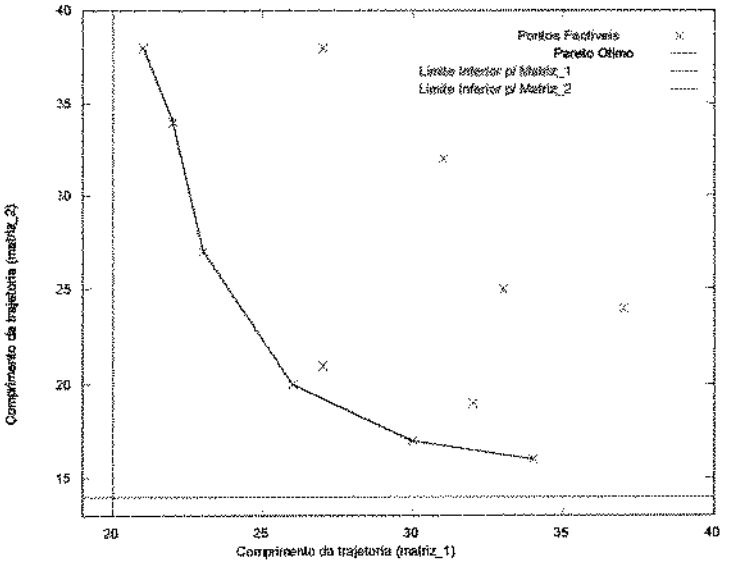
Este problema é bastante complexo e apenas foi apresentado aqui sua essência, de maneira a indicar a aplicação do modelo do MDTSP para um intrincado problema prático.

4.2.4. Limites Inferiores para o MDTSP

Um k-DTSP (MDTSP com k matrizes de distância) pode ser visto como um conjunto de k TSPs, onde a solução ótima de cada um deles fará parte do melhor conjunto de soluções eficientes - o Pareto Ótimo - do k-DTSP. Isto significa dizer que para cada matriz de distância de um MDTSP existe uma solução ótima (eventualmente pode existir mais do que uma solução ótima - trajetórias distintas de igual comprimento *ótimo*), ou seja, existe uma trajetória cujo comprimento é o menor possível segundo a respectiva matriz de distância.

Desta forma, uma maneira de calcular limites inferiores para o MDTSP é abordá-lo como se fosse um conjunto de TSPs e aplicar algum dos métodos para cálculo de limites inferiores disponíveis (ver 4.1.4.3). Cada limite inferior definirá um plano paralelo ao eixo associado à função objetivo correspondente e que passa pelo ponto dado pelo seu próprio valor (o limite inferior calculado). O conjunto dos planos (um plano definido por cada matriz de distância) limita inferiormente a região factível do espaço de objetivos indicando que algumas soluções do Pareto Ótimo (aquelas com o menor valor possível em cada eixo) devem estar $x\%$ próximas a ele (fig.4.15), onde este x dependerá do método empregado para o cálculo dos limites inferiores.

Figura 4. 15: Limites inferiores para um MDTSP com duas matrizes de distância. Cada eixo representa os valores segundo uma das matrizes de distância, sendo que um limite inferior é calculado para cada uma delas isoladamente e utilizados em conjunto para a formação de um semiplano que dista *x%* .



4.2.5. Algoritmos para o MDTSP

Dado que existe um vasto número de algoritmos para o TSP, um objeto de estudo neste trabalho foi buscar alternativas de aproveitamento de tais algoritmos, de maneira a adaptá-los para a geração de soluções válidas para o MDTSP. A seguir, serão apresentadas algumas idéias elaboradas.

4.2.5.1. Utilização das Soluções Intermediárias Geradas pelos Algoritmos

Esta abordagem envolve os algoritmos de melhoria, também conhecidos como algoritmos de busca local (ver 4.1.4.1). Estes algoritmos possuem a característica de gerar um conjunto de soluções válidas em seus procedimentos internos, devolvendo como resultado a melhor solução deste conjunto. Um estudo realizado no início deste trabalho [RS94] foi justamente analisar o comportamento deste conjunto de soluções, gerados segundo uma determinada matriz de distância, em relação às outras matrizes de distância, ou seja, para cada solução intermediária gerada pelo algoritmo de melhoria, calcular seu comprimento segundo as outras matrizes de distância do MDTSP (ver Cap.7). Constatou-se que, via de regra, as soluções intermediárias geradas no processamento interno não eram dominadas pela melhor solução fornecida como resultado final.

A alteração dos algoritmos descritos na seção 4.1.4.1, de melhoria (2-opt, 3-opt, or-opt e Lin-Kernighan), são mostradas a seguir, onde a inclusão da letra I no nome dos algoritmos se refere exatamente às soluções Intermediárias geradas durante a busca de uma solução melhor.

Algoritmo 2OPT-I

Entrada: T (trajetória inicial), c (matriz de distância).
Saída: vetor de trajetórias VT.

1. **Seja** T o ciclo Hamiltoniano corrente;
2. **Repita**
   2.1. **Selecione** duas arestas $\in T$;
   2.2. **Enquanto** a troca de duas arestas $\in E^T$ por duas outras arestas $\in E \backslash E^T$ gerar T' válida, faça:
      2.2.1. **Insira** T' em VT;
      2.2.2. **Se** ($c(T')<c(T)$)
         2.2.2.1. **Faça** $T=T'$;
   até T satisfazer a propriedade *2-ÓTIMA*.
3. **Retorne** VT

Algoritmo 3OPT-I

Entrada: T (trajetória inicial), c (matriz de distância).
Saída: vetor de trajetórias VT.

1. **Seja** T o ciclo Hamiltoniano corrente.
2. **Repita**
   2.1. **Selecione** três arestas $\in T$;
   2.2. **Enquanto** a troca de três arestas $\in E^T$ por duas outras arestas $\in E \backslash E^T$ gerar T' válida, faça:
      2.2.1. **Insira** T' em VT.
      2.2.2. **Se** ($c(T')<c(T)$)
         2.2.2.1. **Faça** T=T'
   até T satisfazer a propriedade *3-ÓTIMA*.
3. **Retorne** VT

Algoritmo OROPT-I

Entrada: T (trajetória inicial), c (matriz de distância).
Saída: vetor de trajetórias VT.

1. **Seja** T o ciclo Hamiltoniano corrente.
2. **Faça** l=3 e *achou_melhor*=VERDADEIRO.
3. **Repita**
   3.1. **Repita**
      3.1.1. **Selecione** l vértices adjacentes em T;
      3.1.2. **Enquanto** a troca de l vértices adjacentes em T por outros l vértices não for repetida e desde que T' gerado seja válido, faça:
         3.1.2.1. **Se** ($c(T')<c(T)$)
            A. **Insira** T' em VT.
            B. **Faça** $T=T'$
         3.1.2.2. **Senão**
            A. **Faça** l--;
            B. **Faça** *achou_melhor*=FALSO;
      enquanto *achou_melhor*=VERDADEIRO
   enquanto l>0.
4. **Retorne** VT

Algoritmo LK-I

Entrada: T (trajetória inicial), c (matriz de distância).
Saída: vetor de trajetórias VT ordenadas decrescentemente.

1. **Seja** T o ciclo Hamiltoniano corrente;
2. **Faça** *max_ganho_local*=0;
3. **Repita**
   - 3.1. **Selecione** um vértice $i \in V$;
   - 3.2. **Retire** uma das arestas $x_i \in E^T$;
   - 3.3. **Selecione** a aresta $y_i = y_i \in E' = E \setminus E^T$;|
   - 3.4. **Enquanto** $E' \neq \emptyset$ faça:
     - 3.4.1. **Compute** $g_i = |x_i - y_i|$;
     - 3.4.2. **Se** *max_ganho_local* < g_i
       - 3.4.2.1. **Faça** *max_ganho_local* = g_i;
       - 3.4.2.2. **Faça** *y_escolhido* = y_i;
     - 3.4.3. **Selecione** $y_i \in E' = E \setminus E^T \setminus \{y_1, ..., y_{i-1}\}$;

     Fim_Enquanto;
   - 3.5. **Troque** x_i por *y_escolhido* em T, gerando T';
   - 3.6. **Insira** T' em VT;
   - 3.7. **Compute** $G = c(T) - c(T')$;

   enquanto $G > 0$
4. **Retorne** VT.

4.2.5.2. Incorporação do Conceito de Dominância e de Decisão de Compromisso

O **conceito de dominância**, como visto no Cap.3, é um importante conceito na manipulação de estruturas k-dimensionais (neste caso, soluções para o MDTSP). Com base neste conceito, surgiu a idéia de modificar os algoritmos existentes para a resolução do TSP, fazendo com que os mesmos sejam capazes de *incorporar* o conceito de dominância entre as soluções que vão sendo geradas.

Por sua vez, uma **decisão de compromisso** é tomada sempre que são considerados simultaneamente todos (ou um subconjunto predefinido) os objetivos envolvidos. A escolha da posição de uma nova solução numa trajetória em construção e a escolha dos vértices a serem permutados numa trajetória de forma a gerar uma trajetória de comprimento menor, são exemplos de decisão de compromisso para problemas multiobjetivos. Desta forma:

Dominância:

- No caso dos algoritmos de melhoria, que possuem a característica de gerar várias soluções factíveis durante uma única execução, uma nova solução gerada só é aceita caso domine em ***d*** objetivos (para o MDTSP, d comprimentos segundo d das k matrizes de distância envolvidas, $1 \leq d \leq k$) a solução corrente, ou seja, seus comprimentos segundo d matrizes de distância devem ser menores do que os comprimentos da solução corrente.
  - Caso $d=k$, tem-se o caso de dominância total, onde uma nova solução gerada só é aceita caso domine em **todos** os objetivos (comprimentos segundo todas as matrizes de distância envolvidas) a solução corrente, ou seja, seus comprimentos segundo todas as matrizes de distância devem ser menores do que os comprimentos da solução corrente. Desta forma, garante-se uma

melhora *monotonicamente crescente* em *todas* as funções objetivo envolvidas, na direção da origem dos eixos formados pelos objetivos. A desvantagem é que isto restringe muito a diversidade de soluções, uma vez que, via de regra, existem poucas soluções intermediárias que dominem completamente uma outra solução intermediária (soluções geradas numa única execução de um algoritmo de melhoria).

- Se $d \neq k$, força-se que em pelo menos um dos d eixos o comprimento deva ser melhor do que os das trajetórias geradas anteriormente. Tal incorporação tem a vantagem de não restringir muito a aceitação de novas soluções, já que na maioria dos casos as soluções intermediárias geradas pelos algoritmos de melhoria não se dominam.

Decisão de Compromisso:

- No caso dos algoritmos que só geram uma solução factível no final de suas execuções, a incorporação do conceito de dominância, tal como descrito anteriormente, não se aplica. Ao invés disto, incorpora-se o conceito de *decisão de compromisso*, onde uma decisão deve ser tomada baseada numa análise de todas as matrizes de distância envolvidas, de tal forma que todas sejam consideradas igualmente. No caso dos algoritmos de construção por inserção, por exemplo, é inserida a cidade que proporcionar o melhor compromisso segundo d matrizes de distância, onde o melhor compromisso pode ser o mais distante, o mais próximo, um aleatório ou o menos custoso.

- Tal estratégia também pode ser incorporada nos algoritmos de melhoria, onde devem ser consideradas todas as matrizes de distância quando for avaliada uma alteração entre os elementos de uma solução. A modificação na estrutura de uma solução será feita com base num compromisso entre todas as matrizes de distância.

Também no caso da incorporação de decisão de compromisso, pode-se ter os casos total e parcial. No caso parcial apenas um subconjunto de matrizes de distância é considerada na tomada da decisão de compromisso.

A seguir são apresentados alguns dos algoritmos existentes para o TSP, adaptados para a manipulação de d matrizes de distância, seja através da incorporação do conceito de *dominância*, seja através da incorporação da *decisão de compromisso* em cada um de seus passos.

Algoritmos de Construção

Os algoritmos de construção desenvolvidos para o Multi-Distance Traveling Salesman Problem são baseados nos algoritmos que seguem o princípio de inserção, conforme descrito na subseção 4.1.4.1 (algoritmos NI, FI, AI e CI). Os novos algoritmos serão descritos com base nas definições apresentadas na subseção 4.1.4.1.

As inserções são sempre baseadas no compromisso entre todas as matrizes de distância envolvidas, onde a melhor opção segundo uma determinada matriz não é, na maioria das vezes, a melhor opção de compromisso para todas as matrizes envolvidas.

Algoritmo Geral para Construção de uma Trajetória *k*-dimensional por Inserção

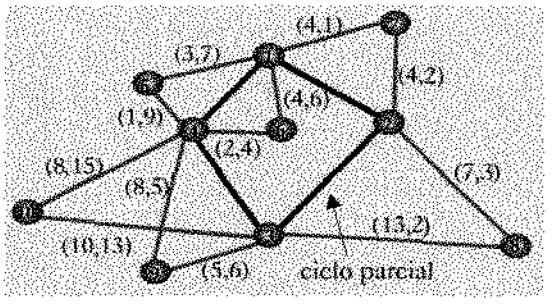
Entrada: $c_1,c_2,...,c_k$ (matrizes de distância), d (grau de dominância).
Saída: trajetória T.

1. **Seja** $c_1,c_2,...,c_k$ as k matrizes de distância envolvidas;
1. **Seja** d o grau de dominância requerido;
2. **Selecione** um ciclo inicial com v vértices $\{v_1,v_2,...v_v\}$, $v>1$, e o conjunto $W=V\setminus\{v_1,v_2,...v_v\}$;
3. **Enquanto** $W\neq 0$
   - **3.1. Selecione** um vértice $j \in W$ de acordo com algum critério;
   - **3.2. Insira** j em alguma posição no ciclo parcial e faça $W=W\setminus\{j\}$;
4. **Retorne** T.

As diferenças estão nos passos 3.1 e 3.2 do algoritmo geral. A seleção de um vértice a ser inserido no ciclo em construção é feita com base em d matrizes de distância, ao invés de apenas uma. A inserção no ciclo parcial, da solução escolhida passo anterior, também pode ser feita com base numa decisão de compromisso entre as matrizes de distância envolvida.

Foram modificados os quatro algoritmos de construção baseados no princípio de inserção, descritos na subseção 4.1.4.1. Agora, os algoritmos estão habilitados para manipular d matrizes de distância simultaneamente. Como as d matrizes de distância envolvidas podem não ser comparáveis quantitativamente (por exemplo, uma matriz representando as distâncias, em *Km*, e outro os custos da viagem, em *R$*), os algoritmos foram desenvolvidos de maneira a realizar uma manipulação *qualitativa* das d matrizes de distância.

Figura 4. 16: Exemplo de um conjunto de cidades (vértices), onde destaca-se um ciclo parcial em construção e as possíveis ligações (arestas e seus respectivos ***vetores de comprimentos***) entre cada cidade fora do ciclo e duas cidades pertencentes ao ciclo parcial.



A distância mínima de compromisso de um vértice ao ciclo em construção é dada pela distância deste vértice em relação ao vértice *mais próximo de compromisso* a ele do ciclo parcial, ou seja, são analisadas as distâncias fornecidas por d matrizes e calculado o vetor de comprimentos que possui os menores valores possíveis (a média total é minimizada). Desta forma:

$$dc_{min}(j) = min\{\sum_{p=1}^{d} rank_{ij}^{p} \,/\, i \in V \setminus W, d \in k\} \quad , \quad j \in W \qquad \text{Eq. 4. 17}$$

Como as d matrizes devem ser comparadas qualitativamente, as trajetórias são construídas com base numa *ordem relativa* (posição relativa dos comprimentos de uma matriz de distância). O comprimento de menor valor será o primeiro no vetor de posições

relativas - *rank* -, e o comprimento de maior valor ocupará a última posição relativa. É estabelecido um vetor de posições relativas para cada matriz de distância e, desta forma, o custo de uma inserção é dado pela somatória das posições relativas que cada comprimento da solução em análise possui no respectivo vetor de posições relativas que ele está relacionado.

Para um melhor entendimento, é dado um exemplo no final desta subseção.

Serão descritos agora, os algoritmos desenvolvidos. Os vértices escolhidos, de acordo com qualquer uma das abordagens descritas a seguir, sempre serão inseridos no ciclo Hamiltoniano parcial de maneira que sua inserção proporcione o menor incremento possível no comprimento total do subciclo Hamiltoniano.

Nearest Insertion of Compromise - NIC

Baseado no algoritmo *Nearest Insertion*, sendo que agora será selecionado o vértice que possuir a menor distância de compromisso. Isto significa dizer que será incluído o vértice cuja somatória das d posições relativas de seus comprimentos (dadas pelo vetor *rank*), $d \in k$ (k matrizes de distância), seja a menor dentre as somatórias de cada vértice candidato a inserção. Então:

$$dc_{min}(j) = min\{dc_{min}(l) / l \in W\} \quad , \quad j \in W \qquad \text{Eq. 4. 18}$$

Furthest Insertion of Compromise - FIC

Neste caso, seguindo o algoritmo *Furthest Insertion*, será selecionado o vértice que possuir a maior distância de compromisso. Isto significa dizer que será incluído o vértice cuja somatória das d posições relativas de seus comprimentos (dadas pelo vetor *rank*), $d \in k$ (k matrizes de distância), seja a maior dentre todas as somatórias de cada vértice candidato a inserção. Então:

$$dc_{min}(j) = max\{dc_{min}(l) / l \in W\} \quad , \quad j \in W \qquad \text{Eq. 4. 19}$$

Arbitrary Insertion of Compromise - AIC

Tal como o algoritmo *Arbitrary Insertion* será selecionado um vértice *aleatoriamente*.

Cheapest Insertion of Compromise - CIC

Neste caso, seguindo o algoritmo *Cheapest Insertion*, será selecionado o vértice que fornecer o menor incremento de compromisso (segundo o vetor de ordem relativa dos comprimentos) ao comprimento total do ciclo Hamiltoniano em construção. Neste caso, a escolha do vértice a ser inserido já está relacionada com a posição de inserção no ciclo Hamiltoniano em construção.

No exemplo abaixo, conforme o conjunto de cidades, arestas e seus respectivos vetores de distância apresentados na fig.4.16, os algoritmos de construção baseados na inserção de compromisso primeiramente classificarão, para cada cidade do ciclo Hamiltoniano parcial, as distâncias de todas as cidades que possuem conexão com a mesma (neste exemplo, o grafo não é completo) numa ordem relativa, ou seja, durante a análise de cada cidade do ciclo parcial, será verificada para cada matriz de distância a

ordem relativa das suas distâncias que, em seguida, podem ser comparadas através da somatória de tais posições relativas. Conforme o esquema abaixo e a fig. 4.17, e seguindo o exemplo da fig.4.16, o NIC escolherá a cidade 7 (o NI escolheria a cidade 5), o FIC escolherá a cidade 10 (o FI escolheria a cidade 8), o AIC escolhe aleatoriamente a cidade 5 e o CIC escolhe a cidade 6 (coincidentemente, o AI também escolheria a cidade 6).

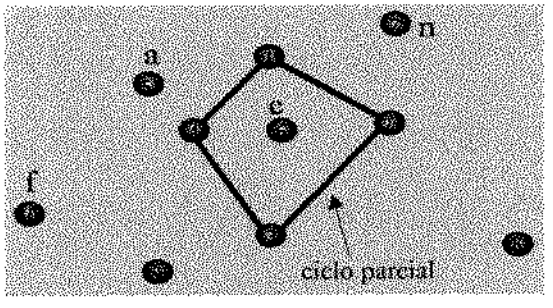
Valores fornecidos pelas matrizes de distância entre a cidade *i* e as cidades a serem inseridas no ciclo parcial.

		1	2	3	4	5	6	7	8	9	10
cidade 1	matriz 1	/	/	/	/	3	4	4	...	...	...
	matriz 2	/	/	/	/	7	6	1	...	...	...
cidade 2	matriz 1	/	/	/	/	...	....	4	7	...	...
	matriz 2	/	/	/	/	...	...	2	3	...	...
cidade 3	matriz 1	/	/	/	/	...	...	...	13	5	10
	matriz 2	/	/	/	/	...	...	...	2	6	13
cidade 4	matriz 1	/	/	/	/	1	4	...	...	8	8
	matriz 2	/	/	/	/	9	6	...	...	5	15

Vetores representando as posições relativas entre a cidade *i* e as demais. O vetor de compromisso (*rank*) contém as *distâncias* a serem consideradas.

		1	2	3	4	5	6	7	8	9	10
vetores de pos. relativa	matriz 1					1	2	3			
	matriz 2					3	2	1			
compromisso (*rank*)						4	4	4			
vetores de pos. relativa	matriz 1							1	2		
	matriz 2							1	2		
compromisso (*rank*)								2	4		
vetores de pos. relativa	matriz 1								3	1	2
	matriz 2								1	2	3
compromisso (*rank*)									4	3	5
vetores de pos. relativa	matriz 1					1	2			3	4
	matriz 2					3	2			1	4
compromisso (*rank*)						4	4			4	8

Figura 4. 17: Comparação entre os algoritmos de construção baseados no esquema de ***inserção de compromisso***. Numa determinada iteração, dado o ciclo parcial (em construção) abaixo e os vértices restantes, o *NIC* adicionaria o vértice *n* ao ciclo parcial, o *FIC* adicionaria o vértice *f*, o *CIC* adicionaria o vértice *c* e o *AIC* poderia adicionar o vértice *a*.



Algoritmos de Melhoria

Os algoritmos descritos na subseção 4.2.5.1, todos de melhoria, devolvem como resultado todo o conjunto de soluções (ciclos Hamiltonianos) geradas durante suas execuções. Baseando-se em tal procedimento, os algoritmos descritos a seguir também devolvem como resultado um conjunto de soluções, entretanto, os ciclos Hamiltonianos resultantes são aqueles não-dominados em pelo menos *d* eixos, segundo os valores das respectivas matrizes de distância.

Algoritmo 2OPT-D

Entrada: T (trajetória inicial), c (matriz de distância), d (grau de dominância).
Saída: vetor de trajetórias d dominadoras - VTd.

1. **Seja** T o ciclo Hamiltoniano corrente;
2. **Seja** $c_1,c_2,...,c_k$ as k matrizes de distância envolvidas;
3. **Seja** d o grau de dominância requerido;
4. **Repita**
   - 4.1. **Selecione** um vértice $v \in V$;
   - 4.2. **Enquanto** a troca das arestas adjacentes a v por duas outras arestas ($E'=E\backslash E^T$) não for repetida e ocasionar um *incremento de compromisso* no comprimento da trajetória resultante T' (T' válido), faça:
     - 4.2.1. **Se** (T' d-domina T)
       - 4.2.1.1. **Insira** T' em VTd;
       - 4.2.1.2. **Faça** $T=T'$;

   até T satisfazer a propriedade 2-ÓTIMA.
5. **Retorne** VTd.

Algoritmo 3OPT-D

Entrada: T (trajetória inicial), c (matriz de distância), d (grau de dominância).
Saída: vetor de trajetórias d dominadoras - VTd.

1. **Seja** T o ciclo Hamiltoniano corrente;
2. **Seja** $c_1,c_2,...,c_k$ as k matrizes de distância envolvidas;
3. **Seja** d o grau de dominância requerido;
4. **Repita**
   - 4.1. **Selecione** um vértice $v \in T$
   - 4.2. **Enquanto** a troca das arestas adjacentes a v por três outras arestas ($E'=E\backslash E^T$) não for repetida e ocasionar um *incremento de compromisso* no comprimento da trajetória resultante T' (T' válido), faça:
     - 4.2.1. **Se** (T' d-domina T)
       - 4.2.1.1. **Insira** T' em VTd;
       - 4.2.1.2. **Faça** $T=T'$;

   até T satisfazer a propriedade 3-ÓTIMA.
5. **Retorne** VTd.

Algoritmo OROPT-D

Entrada: T (trajetória inicial), c (matriz de distância), d (grau de dominância).
Saída: vetor de trajetórias d dominadoras - VTd.

1. **Seja** T o ciclo Hamiltoniano corrente;
2. **Seja** $c_1,c_2,...,c_k$ as k matrizes de distância envolvidas;
3. **Seja** d o grau de dominância requerido;
4. **Faça** l=3 e *achou_melhor*=VERDADEIRO;
5. **Repita**
   - 5.1. **Repita**
     - 5.1.1. **Selecione** l vértices adjacentes em T;
     - 5.1.2. **Enquanto** a troca de l vértices adjacentes em T por outros l vértices não for repetida e ocasionar um *incremento de compromisso* no comprimento da trajetória resultante T' (T' válido), faça:
       - 5.1.2.1. **Se** (T' d-domina T)
         - A. **Insira** T' em VTd;
         - B. **Faça** $T=T'$;
       - 5.1.2.2. **Senão**
         - C. **Faça** l--;
         - D. **Faça** *achou_melhor*=FALSO;

     enquanto *achou_melhor*=VERDADEIRO;

enquanto $l>0$;
6. **Retorne** *VTd*.

Algoritmo LK-D

Entrada: T (trajetória inicial); $c_1,c_2,...,c_k$ (k matrizes de distância); d (grau de dominância, $1 \leq d \leq k$)
Saída: vetor de trajetórias d-dominadoras - *VTd*.

1. **Seja** T o ciclo Hamiltoniano corrente;
2. **Seja** $c_1,c_2,...,c_k$ as k matrizes de distância envolvidas;
3. **Seja** d o grau de dominância requerido;
4. **Faça** *max_ganho_local*=0;
5. **Repita**
   - 5.1. **Selecione** um vértice $i \in V$;
   - 5.2. **Retire** uma das arestas $x_i \in E^T$;
   - 5.3. **Selecione** uma aresta $y_1=y_i \in E'=E\backslash E^T$;|
   - 5.4. **Enquanto** $E' \neq \emptyset$ faça:
     - 5.4.1. **Compute** ganho_de_compromisso $g_i = | x_i - y_i |$;
     - 5.4.2. **Se** *max_ganho_local_de_compromisso* $< g_i$
       - 5.4.2.1. **Faça** *max_ganho_local_de_compromisso* $= g_i$;
       - 5.4.2.2. **Faça** *y_escolhido* $= y_i$;
     - 5.4.3. **Selecione** $y_i \in E'=E\backslash E^T\backslash\{y_1,..., y_{i-1}\}$;

     Fim_Enquanto;
   - 5.5. **Troque** x_i por *y_escolhido* em T, gerando T';
   - 5.6. **Compute** $G = c(T)-c(T')$;
   - 5.7. **Se** (T' d-domina T)
     - 5.7.1. **Insira** T' em *VTd*;

   enquanto $G>0$
6. **Retorne** *VTd*.

Os algoritmos de melhoria ou de busca local 2OPT-D, 3OPT-D, OROPT-D e LK-D são capazes de manipular d matrizes de distância simultaneamente, $1 \leq d \leq k$, onde cada movimento (alteração na trajetória corrente) é feito com base em d matrizes e, uma vez realizado o movimento, a trajetória será inserida no vetor de trajetórias somente se seus valores segundo d matrizes for menor do que o da última trajetória inserida no vetor.

4.3. Resumo

Neste capítulo foi abordado o Problema do Caixeiro Viajante (*Traveling Salesman Problem* - TSP), onde foram apresentadas sua definição, complexidade, um breve histórico e os principais algoritmos encontrados na literatura.

As contribuições do trabalho apresentadas neste capítulo abrange toda a seção 4.2, onde foi apresentado o Problema de Otimização Combinatória Multiobjetivo proposto, o Problema do Caixeiro Viajante com Múltiplas Distâncias (*Multi-Distance Traveling Salesman Problem* - MDTSP), sua definição, complexidade, aplicações práticas e algoritmos de resolução.

4.4. Notas Bibliográficas

Existe um vasto material bibliográfico sobre o TSP, mas, o livro editado por Lawler [LLKS85], o texto de Jünger, Reinelt e Rinaldi [JRR94] e o de Souza [Sou93], foram as principais fontes de consulta.

Dantzig, Fulkerson e Johnson [DFJ54] apresentaram a primeira formulação em programação inteira para o TSP e foram os primeiros a resolver grandes instâncias do problema.

Para os algoritmos existentes para o TSP, de um modo geral, ressalta-se : Mark e Morton [MM93], Laporte [Lap92], Bentley [Ben92, Ben90], Margot [Mar92], Hurkens [Hur91], Johnson [Joh90], Papadimitriou [Pap90], Applegate, Chvátal e Cook [ACC90], Johnson, Papadimitriou e Yannakakis [JPY88], Ohya, Iri e Murota [OIM84], Syslo, Deo e Kowalik [SDK83], Bartholdi e Platzman [BP82]; Rosenkrantz, Stearns e Lewis [RSL77], Or [Or76], Christofides [Chr76], Lin-Kernighan [LK73].

Fredman, Johson, McGeoch e Ostheimer [Fre95] relatam um estudo sobre estruturas de dados para representação de trajetórias do TSP.

Em relação ao MDTSP, foi utilizada toda a referência existente para o TSP e a partir dela elaborado o material descrito sobre o novo problema.

Parte do material referente ao MDTSP pode ser encontrado em [RS95a, RS95b, RS96b].

Capítulo 5

Times Assíncronos

"Um homem pode saber mais do que muitos,
porém nunca tanto como todos."
Marquês de Maricá.

Neste capítulo é apresentado um método multi-algoritmo para a resolução de problemas combinatórios. Tal método, denominado de **Times Assíncronos** (do inglês *Asynchronous Teams* ou A-Teams), se baseia na utilização conjunta de diversos algoritmos, que interagem assincronamente, de maneira a gerar soluções melhores do que as geradas pelos seus componentes isoladamente.

Sua definição e principais características são descritas. Analogias com organizações assíncronas encontradas na natureza são estabelecidas, de maneira a reforçar os conceitos mencionados anteriormente. Um breve histórico é apresentado, onde pode ser encontrado um resumo de várias das aplicações de A-Teams desenvolvidas até o momento.

Por fim, será enfocado o projeto de um A-Team que suporta, também, Problemas de Otimização Combinatória com Múltiplas Funções Objetivo, cujas principais diferenças em relação aos tradicionais A-Teams (A-Teams para problemas com uma única função objetivo) ocorrem na estrutura das memórias compartilhadas, que armazenam agora soluções multidimensionais e nas políticas de destruição e seleção destas soluções. Uma nova estrutura para a organização de soluções na memória é apresentada, sendo denominada de **Memória de Camadas de Conjuntos Não-Dominados**, bem como *novas* políticas de destruição e seleção de soluções adaptadas à nova estrutura de memória compartilhada do A-Team.

5.1. Introdução

Apesar da existência de muitos algoritmos que se propõem a encontrar soluções aproximadas para diversos problemas, estes algoritmos geralmente são adequados somente à circunstâncias particulares, ou seja, dependendo da instância do problema, determinado algoritmo pode ser bem mais eficiente que outro. Com base neste ponto de vista, surge uma indagação:

Dada uma instância de um problema, especificamente de otimização, qual seria a heurística mais adequada a resolvê-lo ?

Uma resposta a esta pergunta pode levar a uma análise bastante exaustiva que depende de uma quantidade muito grande de fatores, tais como: qualidade das soluções geradas, complexidade de implementação, tempo de execução, etc. Além disso, mesmo que tal pergunta pudesse ser respondida com sucesso, ao se escolher uma heurística, certamente se estaria desprezando as qualidades de todas as outras. Daí, então, surge uma outra questão:

Como resolver determinado problema de modo a tirar proveito de mais de uma das heurísticas existentes para resolvê-lo ?

Foi a partir de indagações como esta que surgiu a idéia de se desenvolver o método de Times Assíncronos, onde um conjunto de algoritmos (heurísticos ou não) agem de maneira independente uns dos outros, gerando soluções que podem ser compartilhadas por todos, de tal forma a produzirem resultados cada vez melhores.

5.2. Definição

Times Assíncronos (do inglês ***Asynchronous Teams*** ou **A-Teams** [TS90, ST91, TS92, ST93a, Sou93]) consistem numa abordagem para a resolução aproximada de problemas, baseada em agentes autônomos que se comunicam assincronamente através de memórias compartilhadas, possuindo um fluxo cíclico de dados. Desta forma, os agentes produzem soluções que são compartilhadas por todos, podendo-se chegar a *soluções próximas do ótimo*, ou mesmo, ao ótimo do problema. Um A-Team é definido pelas três características seguintes [TS92]:

Agentes autônomos → Conjunto de algoritmos, que agem independentemente uns dos outros, tal que novos agentes podem ser inseridos ou antigos retirados sem que isto afete o desempenho dos demais.

agente = algoritmo + protocolo de comunicação[1]

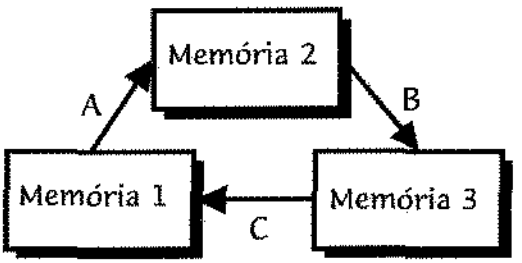
Comunicação assíncrona → Os agentes manipulam dados em memórias compartilhadas sem que haja nenhuma comunicação direta entre eles e sem que haja qualquer sincronismo entre os mesmos, o que permite que sejam executados concorrente ou paralelamente.

[1] *Protocolo de comunicação* consiste num conjunto de regras que definem a política de comunicação entre processos.

Fluxo de dados cíclico → Deve existir pelo menos um ciclo no fluxo de dados, de maneira a permitir uma contínua interação entre os agentes que manipulam - lêem, modificam e armazenam - informações na memória compartilhada.

Os A-Teams podem ser representados graficamente através de *retângulos*, representando as memórias compartilhadas e, de *setas*, representando os agentes. A orientação das setas indica de que memória o agente retirou os dados e em qual ele depositará seus resultados. A seguir (fig.5.1), é dado um exemplo de uma configuração simples de um A-Team, contendo três memórias compartilhadas e três agentes. O *agente A* lê da *Memória 1* e escreve na *Memória 2*, o *agente B* lê da *Memória 2* e escreve na *Memória 3* e o *agente C* lê da *Memória 3* e escreve na *Memória 1*, formando um fluxo cíclico de dados, onde tal ordem pode ser alterada a cada iteração do A-Team, possibilitando uma comunicação completamente assíncrona entre os agentes.

Figura 5.1: Exemplo da representação gráfica de uma configuração de um A-Team. O time é composto por três agentes (A,B,C) e três memórias compartilhadas (1,2,3), onde as setas representam os possíveis fluxos e dados, com suas direções indicando de onde os agentes lêem e onde eles escrevem seus dados, respectivamente.



A ligação entre os algoritmos é realizada pela estrutura de dados usada nas memórias compartilhadas. Não importa o procedimento interno adotado pelo algoritmo para a geração da solução, e sim, que a solução gerada seja apropriadamente formatada para ser armazenada numa memória, servindo de solução inicial para qualquer um dos outros algoritmos do time que lêem da memória. O mesmo raciocínio se aplica para a entrada dos algoritmos: um algoritmo dever ler uma solução da memória num dado formato padronizado e deve ser capaz de transformar para o seu formato interno, se necessário.

Para o caso de problemas com uma única função objetivo, cada solução escrita por um algoritmo numa memória é armazenada numa de suas várias *posições*. Os algoritmos são livres para visitar *qualquer* uma das posições a *qualquer* instante, a única restrição que se aplica, para se garantir a consistência dos dados, é um *mecanismo de semáforo*, ou seja, cada posição é considerada como uma *região crítica da memória* [Tan92]. Isto significa que, se um algoritmo estiver utilizando uma determinada posição, nenhum outro algoritmo poderá visitar tal posição. Nenhuma outra restrição é necessária para a comunicação entre os algoritmos.

Souza [Sou93] mostrou que a aplicação de A-Teams para o TSP resulta num melhor desempenho do que a aplicação de cada algoritmo individualmente. Ele também mostrou que quanto mais algoritmos existirem no A-Team, melhor é o seu desempenho. Tal propriedade foi chamada de ***eficiência em escala***. Além disso, ele realizou

experimentos com várias máquinas e obteve resultados próximos da situação ideal: o tempo de execução para uma máquina foi dividido por 2, 3 e 4 quando usados 2, 3 e 4 processadores, respectivamente. Estes resultados motivaram, ainda mais, a idéia de expandir este trabalho para Problemas Combinatórios com Múltiplas Funções Objetivo.

Um A-Team pode ser composto por uma ou mais memórias compartilhadas, onde os agentes podem ler e escrever dados (soluções). Os dados são lidos conforme a *política de seleção* de cada agente, que é livre para escolher o dado da memória que melhor for conveniente. Como as memórias não podem incorporar dados infinitamente, a cada uma está associado um agente denominado *destroyer* ou destruidor. O destruidor tem a função de julgar e eliminar, baseado em alguma ***política de destruição***, um dado (solução) da memória para abrir espaço a um novo. Tão importante quanto produzir bons dados é a tarefa de eliminar dados não promissores, onde bons dados a serem eliminados não são, necessariamente, os piores dados quanto à função objetivo. A correta eliminação de dados economiza esforços e delineia a evolução dos dados na organização em torno dos resultados desejados. A política de destruição é geralmente relacionada com a qualidade dos dados. Tais itens serão abordados em detalhes posteriormente.

5.3. Contexto Histórico

A idéia de se utilizar várias heurísticas conjuntamente para se resolver determinado problema não é tão recente assim. Em 1975, Weiner [Wei75] afirmou que melhores resultados poderiam ser obtidos para os problemas de otimização se fosse aplicada uma heurística de melhoria sobre a solução gerada por uma heurística de construção. Tal proposta se mostrou bastante inflexível, pois tornava a configuração muito limitada: se a solução construída não fosse muito boa, o algoritmo de melhoria encontraria o mínimo local *daquela* região desfavorável e o processo terminaria mesmo assim.

Uma década depois, Pyo [Pyo85] apresentou um trabalho que inspiraria a criação do método de Times Assíncronos. O autor propôs a aplicação combinada de diversos algoritmos já existentes para a resolução de um dado problema, de forma distribuída e síncrona, que ele denominou de *abordagem de times* (do inglês *Team Approach*). Tal idéia foi aplicada na resolução de equações algébricas utilizando-se os métodos aproximados *Newtow-Raphson* (NR) e *Steepest Descent* (SD), que partem de um ponto (solução) inicial e, sucessivamente, geram soluções cada vez mais próximas da ótima. O método NR transforma as equações em lineares e usa as raízes destas equações lineares como pontos correntes para as iterações seguintes. O método SD usa a direção do gradiente de cada equação como a linha de direção para a minimização do erro entre a solução estimada e uma real.

Em sua abordagem de *Team Approach*, Pyo utilizou o conceito de *BlackBoards* como elo de comunicação entre os dois métodos. Arquitetura *blackboard* é uma técnica distribuída de resolução de problemas introduzida por Newell [New62] e se baseia no uso de uma memória compartilhada, contendo uma solução de cada vez, a qual é atualizada por vários agentes especialistas *sempre* sob a supervisão de um agente de controle (*supervisor*). Inicialmente o *blackboard* fornece um mesmo ponto de partida aos métodos NR e SD. Cada método efetua seus processamentos independentemente um do outro e,

ao término, ambos retornam seus resultados para o *blackboard*. Por sua vez, o blackboard analisa os resultados e permite que o método de menor progresso seja executado novamente, tendo os resultados gerados pelo método de melhor performance como ponto de partida. E assim sucessivamente, até os resultados serem satisfatórios.

5.3.1. O Trabalho Pioneiro

Em **1990**, **Talukdar** e **Souza** [TS90] introduziram o termo *Times Assíncronos* (do inglês *Asynchronous Teams*), através de um trabalho para a *Resolução de Equações Algébricas Não-lineares*. Neste trabalho, os autores, adaptando o modelo proposto por Pyo, usaram num mesmo time o método de *Newton-Raphson* (NR) e um Algoritmo Genético (GA). Estes métodos possuem características bem distintas. Os GAs, como pôde ser observado no capítulo 2, realizam uma busca global no espaço de soluções factíveis para o problema, escapando de mínimos locais, mas, demandando um grande esforço computacional para convergir. O NR, por sua vez, converge muito rapidamente, servindo como um método de melhoria, refinando as soluções geradas por outros métodos, neste caso específico, o GA.

Duas fortes diferenças entre os dois trabalhos, o do Pyo e o de Talukdar e Souza, podem ser notadas: a primeira, é que no *time* de Talukdar e Souza não existia nenhum *supervisor*, ou seja, os dois métodos interagiam, porém, sem nenhum sincronismo entre eles; e, a segunda diferença, é que os métodos que faziam parte do time de Talukdar e Souza possuíam características distintas, cujas virtudes foram adequadamente exploradas através da estrutura de fluxo de dados do time formado, resultando na obtenção de soluções de alta qualidade num curto espaço de tempo, se comparados com a execução dos métodos isoladamente.

5.3.2. Os Trabalhos Conseguintes

A partir do trabalho pioneiro, inúmeros outros foram desenvolvidos. O objetivo desta subseção é dar uma visão geral do que foi desenvolvido e quais problemas foram abordados por A-Teams até os dias de hoje.

Em 1991, Souza e Talukdar [ST91], aprimoraram a idéia precursora e desenvolveram um A-Team mais eficiente, trocando o método NR pelo método *Levenbergh-Marquardt* (LM). Desta vez, duas versões do método LM foram utilizadas. Os autores tiveram a seguinte idéia: utilizar o GA para encontrar os valores das variáveis globais, ou sejam, aquelas variáveis que apareciam em todas as equações; utilizar uma versão do LM (LM com baixa precisão) para encontrar os valores das variáveis locais, ou seja, aquelas variáveis que apareciam somente em algumas equações, desta forma, o LM poderia encontrar soluções mais facilmente. Esta combinação gerava soluções potencialmente boas, que serviam como ponto de partida para uma outra versão do LM (LM com alta precisão), que refinava as soluções iniciais, gerando as soluções finais.

Ainda em 1991, Quadrel [Qua91] desenvolveu um trabalho intitulado *Asynchronous Design Environments: Architeture and Behavior*, onde ele apresenta um sistema denominado de *Anarquia* (do inglês *Anarchy*), cujos conceitos apesar de similares, violam as características fundamentais do A-Team original, tal como a

comunicação assíncrona entre agentes. O problema abordado foi intitulado de: *High Rise Building Design.*

Em 1992, Talukdar e Souza [TS92] desenvolveram o trabalho *Scale Efficienty Organizations*, onde foi dito que organizações eficientes em escala são aquelas *abertas* (crescem facilmente) e cujos componentes, novos e antigos, se cooperam eficientemente de maneira benéfica para o grupo, ou seja, a inclusão de novos componentes proporciona um incremento de benefícios para a organização. O trabalho apresentou Time Assíncrono como uma organização eficiente em escala. Via de regra, os agentes interagem de maneira sinérgica, ou seja, a interação entre eles proporciona melhores resultados do que a "soma" dos resultados produzidos por cada um individualmente.

Também em 1992, Murthy [Mur92] desenvolveu uma abordagem para a resolução de um *Problema de Design de um Robô Manipulador Modular*. O robô manipulador, neste caso, é um dispositivo mecânico motorizado e controlado por computador (similar a um braço humano), que tinha que realizar da melhor maneira possível uma série de tarefas previamente estabelecidas. As especificações de tarefas dadas inicialmente, estipulam restrições dinâmicas e cinemáticas ao comportamento do manipulador. O problema, então, é desenvolver um manipulador capaz de satisfazer estas restrições, atingindo *todos* os objetivos da melhor maneira possível. Os manipuladores tinham que satisfazer os seguintes objetivos: alta precisão, movimentos rápidos, recobrimento de todo o espaço, manobras no espaço determinado. Esta pode ser considerada a primeira vez que um problema multiobjetivo foi abordado através de A-Teams, entretanto, as funções objetivo foram encapsuladas numa só e o problema fora resolvido como sendo monobjetivo.

Chen, também em 1992, utilizou A-Teams para a resolução de um Problema de Diagnóstico de Falhas num Sistema de Transmissão de Energia Elétrica. Este problema pode ser descrito da seguinte maneira: dada uma configuração inicial de falha numa rede de energia elétrica e um conjunto de alarmes, achar a hipótese que forneça explicações para o conjunto de alarmes. *Falhas* são ocorrências anormais, tal como o mal-funcionamento de um dispositivo ou uma perda de mensagem, que pode acontecer numa rede. Tal problema pode ser caracterizado como um problema de busca, ao invés de um problema de otimização, uma vez que o A-Team foi aplicado para encontrar todas as hipóteses que expliquem o conjunto de falhas e não a melhor.

Em 1993, Souza e Talukdar apresentaram o trabalho *Asynchronous Organizations for Multi-Algorithm Problems* [ST93a], onde foi caracterizado o tipo de problema adequado para ser resolvido através de A-Teams, denominados de Problemas Multi-Algoritmos.

No livro *Engineering Design: the creation of products and processes* pode ser encontrado um capítulo intitulado *Objects Organizations and SuperObjects* escrito por ambos [TS93]. Talukdar e Souza publicaram outros artigos envolvendo o método de Times Assíncronos, que podem ser encontrados [Tal93, Tal92, TSM93].

Souza, entre 1991 e 1993 [Sou93], aplicou A-Team para o Problema do Caixeiro Viajante (*Traveling Salesman Problem*-TSP) e reuniu os resultados desta aplicação, bem como todo o material de pesquisa desenvolvido por ele neste período, em sua tese de doutorado intitulada *Asynchronous Organizations for Multi-Algorithm Problems*. Neste

trabalho, fora mostrado que Times Assíncronos resolveu à *otimalidade* grandes instâncias do TSP, inclusive a famosa instância de 532-cidades [PR91, LLKS85], dando respaldo para futuras aplicações de A-Teams para outros problemas NP-difíceis. Esta tese se constitui no principal texto sobre A-Teams e fora essencial para o desenvolvimento deste trabalho.

Ramesh e Talukdar desenvolveram *A-Teams for Real-Time Operations* [TR92a], *Cooperative Methods and Security Planning* [TR92b] e *Contingency Constrained Operations*. Talukdar, Ramesh, Quadrel e Christie, em 1992, desenvolveram o trabalho *Multi-Agent Organizations for Real-Time Operations* [TRQC92b]. Em 1993, Talukdar e Ramesh [TR93] propuseram uma nova abordagem, baseada em A-Teams, para resolver um Sistema de Controle em Tempo-Real para uma Rede de Energia Elétrica, cujo trabalho foi intitulado de *A Multi-Agent Technique for Contingency Constrained Optimal Power Flows*. Em 1994, Ramesh defendeu a tese intitulada *Inertial Search and Asynchronous Decompositions*.

Em 1995, Talukdar, Gove e Souza [TGS95] desenvolveram o trabalho intitulado *Asynchronous Teams: Near-Scale-Effective Organizations for Distributed, Computer-Based Agents*. Também em 1995, Talukdar, Souza, Quadrel e Ramesh apresentaram o trabalho *A-Teams: Multi-Agent Organizations for Distributed Iterations* [Tal95]. Ainda 1995, Kao, Hemmerle e Prinz [KHP95] desenvolveram o trabalho intitulado *Asynchronous Teams Based Collision Avoidance in PAWS*.

Souza, a partir do final de 1993, começou a orientar diversos pesquisadores em trabalhos envolvendo Times Assíncronos.

Em 1994, Peixoto e Souza [PS94a], aplicaram A-Teams para o *Flow Shop Problem - FSP*. E, ainda em 1994, Peixoto e Souza [PS94b], desenvolveram uma *Metodologia de Especificação de Times Assíncronos*. Peixoto, em 1995 [Pei95], terminou sua dissertação de mestrado intitulada *Metodologia de Especificação de Times Assíncronos para Problemas de Otimização Combinatória*, onde foi reportado todos os resultados da aplicação de A-Teams para o FJP, novos limites inferiores para este problema e a metodologia desenvolvida para aplicação de A-Teams. Para várias instâncias do problema se chegou à *otimalidade* e os limites inferiores gerados foram de melhor qualidade do que os conhecidos anteriormente.

Cavalcante e Souza, em 1995 [CS94, CS95], aplicaram A-Teams para o *Job Shop Problem - JSP*. A abordagem adotada envolvia apenas heurísticas de construção. Cavalcante, ainda em 1995 [Cav95], terminou sua dissertação de mestrado intitulada *Times Assíncronos na resolução do Job Shop Problem - heurísticas de construção*, onde foi reportado os resultados de toda a pesquisa desenvolvida. Para várias instâncias conhecidas do JSP se chegou ao valor ótimo.

Em 1995, Camponogara e Souza [CaS95a, CaS95b, CaS95c] aplicaram A-Teams para um *Problema de Distribuição de Derivados de Petróleo*. O problema em questão é dinâmico (os dados do problema sofrem contínua alteração em tempo de execução) e, devido a sua complexidade, a abordagem utilizada envolveu a decomposição do problema em sub-problemas menores. Neste trabalho, foi constatado que A-Teams é apropriado para a resolução de grandes problemas através de sua decomposição, desde que cada sub-problema possa ser resolvido por um ou mais algoritmo específico, bem como os

algoritmos do time possam ser atualizados sem que ocorra qualquer modificação nas versões remanescentes. Camponogara, no final de 1995 [Cam95], terminou sua dissertação de mestrado intitulada *Times Assíncronos para o Problema de Transporte de Derivados de Petróleo.*

Ainda em 1995, Rodrigues e Souza [RS95a, RS95b, RS95c, RS96a, RS96b], aplicaram A-Teams para o *Multi-Distance Traveling Salesman Problem*, um problema proposto como generalização do clássico TSP. Este presente trabalho relata o estudo completo desta aplicação. Vale ressaltar, entretanto, que esta foi a segunda tentativa bem-sucedida de abordagem de problemas multiobjetivos através de A-Teams (a primeira foi por Murthy [Mur92]). Poderá ser verificado no restante desta dissertação, que diversas estratégias de abordagens foram desenvolvidas, sendo que em uma delas, pela primeira vez, otimiza-se *simultaneamente* múltiplas funções objetivo em A-Teams.

Ribeiro e Souza [RiS95], desde 1995, estão estudando a aplicação de A-Teams para o *Prize-Collecting Traveling Salesman Problem.* Gaspareto e Souza estão trabalhando num A-Team para o Job Shop Problem utilizando apenas heurísticas de melhoria [HS94].

Nascimento, Mendonça e Souza desenvolveram, muito recentemente, um trabalho sobre A-Teams para Desenho de Grafos, onde já fazem uso de algumas das idéias apresentadas neste trabalho e cujo artigo foi intitulado *Sinergia em Desenho de Grafos usando Springs e Pequenas Heurísticas* [NMS96]. O problema está sendo considerado como multiobjetivo, onde se tenta encontrar o melhor compromisso entre dois critérios estéticos: maximização da simetria e minimização de cruzamentos entre arestas.

A tabela 5.1 resume todas as aplicações de A-Teams citadas nesta subseção. Na primeira coluna, estão relacionados os nomes dos pesquisadores. Na segunda, o período em que trabalharam com tal problema. E, na terceira coluna, o nome do problema abordado através de A-Teams.

Tabela 5.1: Relação dos problemas abordados através de A-Teams.

Aplicações de A-Teams

Autores	Período	Problema
Talukdar e Souza	1990-1991	*Non-linear Equations*
Souza e Talukdar	1991-1993	*Traveling Salesman Problem*
Murthy e Talukdar	1992-1993	*Multi-Segment Robot Arm*
Chen e Talukdar	1991-1992	*Power System Fault Diagnosis*
Ramesh e Talukdar	1991-1994	*Contingency Constrained Optimal Power Flows*
Peixoto e Souza	1993-1995	*Flow Shop Problem*
Cavalcante e Souza	1993-1995	*Job Shop Problem* (heurísticas de construção)
Longo e Aragão	1993-1995	*Set Cover / Set Partition Problem*
Camponogara e Souza	1994-1995	*Oil Derivative Transportation Problem*
Rodrigues e Souza	1994-1996	*Multi-Distance Traveling Salesman Problem*
Ribeiro e Souza	1994- ...	*Prize-Collection Traveling Salesman Problem*
Glienke e Aragão	1993- ...	*Vehicle Routing Problem*
Haddad e Souza	1993- ...	*Job Shop Problem* (heurísticas de melhoria)
Nascimento, Xavier e Souza	1995- ...	*Desenho de Grafos*

5.4. Organizações Assíncronas

Quando pensamos em *modelo organizacional*, logo vem à mente uma estrutura hierárquica, onde todos seus componentes são organizados de forma a se estabelecer, precisamente, a ordem em que tarefas deverão ser executadas, bem como a ordem em que cada componente do modelo deverá executá-las. Um sincronismo explícito é logo pensado como elemento essencial de um bom modelo organizacional - tal como os modelos organizacionais presentes na sociedade humana [Dun78]. Esta foi a tônica da *Teoria da Organização* até pouco tempo atrás [Fox79]. Mas, basta olharmos a nossa volta, na própria natureza, e encontraremos inúmeros exemplos de modelos organizacionais assíncronos que harmonicamente garantem o equilíbrio e a existência da vida. Por exemplo, existem alguns processos com características marcantes:

O sistema neurológico do corpo humano é um exemplo de um modelo natural autônomo. Cada um dos milhares de neurônios recebem e enviam milhares de sinais elétricos, recobrindo todo o corpo humano e formando uma complexa rede de comunicação. Mas, apesar de toda a complexidade envolvida, não existe nenhum controle central sobre os neurônios. Devido a sua extrema complexidade e eficiência, o sistema neurológico humano tem sido intensivamente estudado e seu comportamento representado em modelos matemáticos. Tais estudos já deram origem, por exemplo, à área de *Redes Neurais* em Ciência da Computação (ver Cap.2).

Os processos cíclicos dos seres vivos, tais como o ciclo da água e a cadeia alimentar, são verdadeiros fluxos cíclicos de informação, permitindo uma completa interação entre seus elementos. Não existe nenhuma ordem específica entre as ações que são executadas: a cada instante um caminho diferente pode ser adotado nestes fluxos cíclicos e interativos. A matéria orgânica é constantemente utilizada e reutilizada no processo de formação dos seres vivos.

Ainda, o nascimento e evolução de uma estrela, dando origem a uma nova galáxia, é caracterizado por processos aleatórios e assíncronos. Novas estrelas e outros elementos cósmicos surgem a todo instante, da mesma maneira que num dado momento de suas existências eles se extinguem. Se extinguem, mas os resíduos resultantes dão início ao processo de formação de novos elementos. Tudo isto sem nenhuma ordem predefinida. Por exemplo, se a nuvem gasosa que dá origem à formação de uma estrela contiver pouco hidrogênio (elemento mais abundante no universo e matéria básica na formação de uma estrela), o núcleo não será muito denso e a estrela resultante será pouco luminosa, se apagando rapidamente. Agora, se o processo de formação estelar contiver hidrogênio em quantidades absurdamente grandes, as reações desencadeadas serão tão violentas que propiciarão a formação de um elemento tão pequeno e compacto que nem a luz escapará dele; e a estrela originária será cercada por uma área que fica completamente isolada do restante do universo, produzindo o que nós conhecemos como *buraco negro*.

Interessantemente, o conceito de Times Assíncronos veio como uma idéia inovadora no sentido de incorporar tais características naturais (assincronismo, aleatoriedade, reutilização, fluxos cíclicos e autonomia) em modelos artificiais, especificamente em organizações de software.

Pode ser encontrado na literatura [Lau92, OW78, Sou93] diversos exemplos de modelos naturais de organizações assíncronas.

5.5. Como Projetar um A-Team?

Desenvolver um A-Team básico que gere soluções válidas para o problema em questão é uma tarefa simples. Para isto, basta que os algoritmos disponíveis para a resolução do problema escolhido sejam reunidos e que pelo menos um fluxo cíclico de dados para conectar tais algoritmos seja estabelecido. Em seguida, deve-se atribuir uma política de seleção para a escolha de soluções pelos algoritmos e uma política de destruição para as soluções contidas na memória. Um passo opcional seria a criação de algoritmos específicos a serem incluídos no A-Team.

Entretanto, apesar desta aparente simplicidade, estipular o melhor conjunto de parâmetros que envolve o projeto de um A-Team é bastante complicado e vem sendo realizado de maneira empírica.

Souza [Sou93] citou alguns passos de como projetar um A-Teams para o TSP, mas, algumas perguntas ficaram no ar no que dizia respeito a aplicabilidade do procedimento usado por ele para outros problemas:

Dado um determinado problema, como saber se ele é adequado para ser resolvido através de A-Teams? E, caso positivo, como projetar um A-Team para tal problema? Poderiam ser preestabelecidas configurações padrões de A-Teams para determinadas classes de problemas?

Tais questões estimularam os estudos realizados por Peixoto [Pei95], resultando no desenvolvimento de uma *Metodologia de Especificação de Times Assíncronos para Problemas de Otimização Combinatória*, onde ele generaliza as idéias descritas anteriormente por Souza [Sou93] e acrescenta outras de fundamental importância.

Os principais *passos* desta metodologia são:

1. **Especificação do problema;**
2. **Representação e qualidade das soluções;**
3. **Verificação da Adequabilidade do Problema à Aplicação de A-Teams;**
4. **Aplicação da configuração geral proposta pela metodologia;**
5. **Especificação das memórias, agentes, fluxos de dados e políticas de seleção e destruição de soluções;**
6. **Análise das soluções obtidas, ou melhor, realização de testes empíricos para ajuste dos parâmetros até os resultados obtidos serem satisfatórios.**

Peixoto [Pei95] afirma que a metodologia foi desenvolvida para Problemas de Otimização Combinatória com ***uma*** única ***função objetivo***, o que não é o caso do Multi-Distance Traveling Salesman Problem - MDTSP, que é um problema multiobjetivo. Acontece que, durante a aplicação de A-Teams para o MDTSP, pôde ser constatado que os passos a serem seguidos são os mesmos, e que a diferença acontece em alguns de seus

procedimentos internos. Especificamente, na estrutura das memórias (organização das soluções na memória) e políticas de seleção e destruição. Tais *diferenças* serão mostradas nas seções seguintes e continuadas no próximo capítulo.

5.5.1. Especificação do Problema

Conhecer o problema é o primeiro passo para a sua resolução, já dizia Polya em *"How to Solve it: A New Aspect of Mathematical Method"* [Pol73], onde são sugeridas quatro grandes fases para a resolução de um problema: 1) Entendimento do Problema; 2) Estabelecimento de um plano de ação; 3) Execução do plano; e, 4) Análise dos resultados. Tais fases estão diluídas nos passos sugeridos para o desenvolvimento de um A-Team, como visto anteriormente.

Especificamente para o caso de Problemas de Otimização Combinatória -POC, é preciso definir o problema corretamente, compreender seu domínio, especificar todas as variáveis envolvidas, as restrições que devem ser respeitadas, o(s) objetivo(s) a ser(em) atingido(s) e o que é uma solução para este problema. Muito importante, também, é saber qual o *tipo* de problema com que será abordado, como por exemplo, se o problema é multiobjetivo (envolve várias funções objetivo) ou dinâmico (os dados do problema sofrem contínuas alterações durante a execução).

Novamente, Polya também sugere que se verifique a existência de um problema similar já resolvido anteriormente e, assim, tirar proveito do estudo realizado. Então, voltando à classe dos POC, é preciso definir um modelo matemático para o mesmo de tal forma que todas as restrições, variáveis e funções objetivo estejam representadas. No caso de uma aplicação prática, pode ser identificado um modelo matemático já existente que o represente, ou pelo menos, qual o modelo que mais se assemelha com o problema que se está querendo resolver e assim, realizar as adaptações necessárias.

Vale ressaltar que para um mesmo problema pode existir inúmeros tipos de formulações matemáticas, basta verificar o caso do TSP onde novas formulações vêm sendo desenvolvidas possibilitando melhores desempenhos dos algoritmos exatos. No caso de abordagem aproximada, a formulação matemática não é crucial: o importante é estabelecer *uma* delas, mesmo que bem simples.

Ao final desta fase, a pergunta *"O que resolver?"* já deve ter uma resposta satisfatória, faltando agora a resposta a uma outra pergunta: *"Como?"*

5.5.2. Representação e Qualidade das Soluções

Um A-Team deve possibilitar a interação entre os agentes (algoritmos) que o compõe. Tais agentes interagem através de memórias compartilhadas, podendo, desta forma, aproveitar as soluções geradas por outros agentes e conseguir resultados melhores.

Já que soluções devem ser compartilhadas por todos, é preciso definir uma representação para as mesmas de tal forma que todos os agentes a ela relacionados possam *entendê-la*. Tal representação depende do problema que está sendo resolvido.

No caso do TSP, por exemplo, a formulação matemática em programação inteira dada por Dantzig, Fulkerson e Johnson [DFJ54] indica que uma solução é representada por uma matriz binária *nxn*, onde *n* é o número de cidades e as posições que possuem o valor 1 são as que representam a existência de um caminho entre as duas cidades, representada pelas respectivas linha e coluna (ver 4.3). Uma outra maneira de representar uma solução factível para o TSP é através de uma seqüência de *n* números, indicando a ordem em que as cidades devem ser visitadas. Tal representação pode ser implementada através de ***arrays*** ou vetores de *n* posições, onde a ordem relativa das cidades pode ser determinada em tempo constante, entretanto, requer tempo $\Omega(n)$ no pior caso para gerar uma ordenação inversa. Para *n* grande, $1.000<n<1.000.000$, são indicadas estruturas mais avançadas como ***AVL trees*** (de *Adelson-Velskii e Landis*, é uma árvore balanceada de busca tal que, para todo nó, a diferença entre a altura de suas sub-árvores à direita e à esquerda é no máximo 1 [Man89, AHU74]) com *n* nós, adaptada para representar uma trajetória do TSP [Mar92], sendo que sua implementação é complexa [Fre95]. No lugar da AVL, para $1.000<n<1.000.000$, são indicadas duas novas estruturas: ***two-level trees***, que são árvores com somente dois níveis para representar uma trajetória, onde as operações de consulta, tal como para os vetores, levam tempo constante, mas a permutação de elementos leva $O(\sqrt{n})$; e, ***segment trees***, que são estruturas compostas por uma árvore e uma lista de segmentos, onde uma lista de segmentos é uma lista duplamente ligada, com cada elemento correspondendo a um dos segmentos, e contendo os índices dos finais do segmento e um *bit reverso*. Além disso, o representante de um segmento na árvore de segmentos contém um ponteiro para o representante na lista de segmentos. Para *n* muito grande, $n \geq 10^6$, caso muito comum em aplicações VLSI, por exemplo, é indicada a estrutura de dados ***splay tree***, que é uma árvore binária de busca onde cada vértice possui um *bit reverso* especial que indica se a raiz (vértice) de uma sub-árvore deve ser trocada *em ordem*, da esquerda para a direita (*bit reverso* ativado), ou invertida *em ordem*, da direita para a esquerda (*bit reverso* desativado), com a característica de que os vértices com *bits* reversos de altura *h* podem desfazer o efeito dos vértices com *bits reversos* de altura *h-1*. A idéia central de uma *splay tree* é que, toda vez que um vértice é visitado, ele é carregado para a raiz (*splayed*) por uma seqüência de rotações (alterações locais na árvore tal que se preserve a busca em ordem transversal). *Splay tree* é assintoticamente a melhor estrutura de dados para a representação de trajetórias, possibilitando operações de consultas e atualizações em tempo $O(\log n)$ [Fre95].

O importante, para este trabalho, é que, para $n<1000$, ***arrays*** (vetores) ainda é a estrutura mais indicada [Fre95].

O tipo de representação de soluções usado causa sensíveis alterações nos resultados computacionais obtidos e por isto, seria natural a opção pela representação que demonstrasse mais eficiência neste ponto. Mas não é tão simples assim. Primeiro porque se formos procurar a estrutura mais eficiente existente, podemos nos deparar com estruturas complexas demais do ponto de vista computacional. E segundo, porque é importante observar o tipo de representação de soluções que cada um dos algoritmos do A-Team possui e, a partir disto, procurar utilizar uma representação o mais semelhante possível, senão a mesma, para evitar constantes conversões de representações e a conseqüente degradação da performance do A-Team.

O ideal seria transformar *todas* as representações de soluções do A-Team (as representações das memórias e as dos algoritmos) numa única representação, mas, novamente, nem sempre fazer isto é uma tarefa trivial e se pode acabar despendendo muito tempo nesta tarefa em detrimento de outras também importantes. Um *compromisso* entre a rapidez para se obter resultados satisfatórios e a rapidez para o desenvolvimento de um A-Team deve ser atingido.

Em relação ao nível de *qualidade das soluções* geradas, embora busca-se sempre a solução ótima, para muitos problemas (geralmente, aplicações práticas) as soluções desejadas não necessariamente incluem a solução ótima. Alguns problemas podem possuir dados imprecisos, por exemplo, onde a busca da solução ótima pode significar esforço desnecessário e o ideal é a obtenção de um conjunto de soluções factíveis próximas do ótimo segundo a formulação feita com base nos dados imprecisos. Para outros problemas, por exemplo, não é claro *qual* a função objetivo a ser otimizada e, portanto, não é possível avaliá-la precisamente.

Para problemas multiobjetivos, como é o caso do MDTSP, a busca da solução ótima cede lugar à busca de soluções de melhor compromisso que satisfaçam todos os objetivos envolvidos. Neste caso, o que se busca é o *Pareto Ótimo* do problema, ou seja, o melhor conjunto de soluções não-dominadas dentre todo o conjunto de soluções factíveis do problema (ver Cap.3). Mais uma vez, e neste caso reforçado pelo acréscimo de dificuldade inserida com o aumento do número de funções objetivo, encontrar o melhor conjunto de soluções não é necessário, o ideal é fornecer um conjunto de soluções de compromisso, tão bom quanto possível, e deixar a análise ser feita por um especialista que possa escolher, dentre as soluções de compromisso fornecidas, qual seria a mais adequada naquele momento.

A possibilidade de escolha por um *tomador de decisão* é proporcionada naturalmente pelo A-Team, já que ele fornece como resultado final um conjunto de soluções de compromisso (e, também, se manipula conjuntos de soluções durante todo o processo de execução), e não apenas uma única solução final.

5.5.3. Tipo de Problemas Adequado à Aplicação de A-Teams

Nesta subseção serão apresentadas as características necessárias que um problema deve possuir para ser considerado adequado à aplicação de A-Teams. Aos problemas que possuem tais características denomina-se *Problemas Multi-Algoritmos*, como veremos a seguir.

Primeiramente, algumas definições tornam-se necessárias:

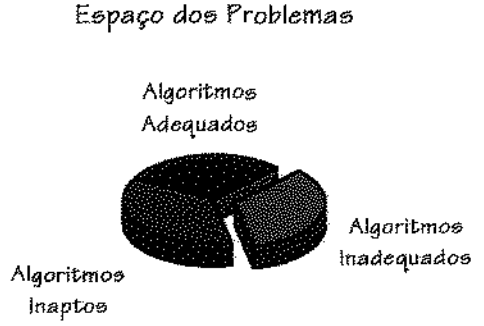
Espaço dos Problemas

Souza [Sou93] classificou os problemas em três grandes categorias, onde cada categoria é definida de acordo com os algoritmos existentes para resolver os problemas pertencentes a ela:

1. Categoria dos Problemas que possuem *Algoritmos Adequados* para resolvê-los;
2. Categoria dos Problemas que possuem *Algoritmos Inadequados* para resolvê-los;
3. Categoria dos Problemas que possuem *Algoritmos Inaptos* para resolvê-los.

Esta classificação **não** é dada pela *complexidade* dos algoritmos existentes, tal como é feito na Teoria da Complexidade [GJ79], e sim, pela capacidade dos algoritmos em fornecer soluções factíveis (soluções válidas, que respeitam todas as restrições envolvidas) em relação ao tempo disponível para resolvê-los.

Figura 5.2: Espaço dos problemas segundo os algoritmos existentes para resolvê-los. Os algoritmos podem ser divididos em três categorias: Algoritmos Adequados; Algoritmos Inadequados; e, Algoritmos Inaptos.



Os problemas pertencentes a categoria dos **Algoritmos Adequados** são aqueles que possuem pelo menos um algoritmo adequado para resolvê-los, ou seja, algoritmos que forneçam soluções válidas no tempo disponível. Assumindo que todos tenham tempo suficiente para encontrar as soluções desejadas. Como por exemplo:

- *Quick Sort* e *Merge Sort* para a ordenação de vetores [AHU74];
- *Simplex* para a resolução de problemas de programação linear [BJS90];
- *Branch-and-Bound* para a resolução de problemas de programação inteira [PR88, NW88].

Vale ressaltar que pela Teoria da Complexidade a *ordenação* é classificada como um problema polinomial (pertence a classe *P*), mas, os *problemas modelados em programação inteira* não possuem algoritmos conhecidos de tempo polinomial para resolvê-los (pertencem a classe *NP*) [GJ79].

A segunda categoria engloba os problemas que possuem **Algoritmos Inadequados** para resolvê-los, ou seja, algoritmos que gerem soluções não satisfatórias para o padrão de qualidade desejado (no caso de alguns algoritmos heurísticos, por exemplo) ou gerem soluções infactíveis que violam alguma restrição do problema (no caso de algoritmos que desconsideram alguma restrição com o intuito de facilitar a busca por melhores soluções). Como por exemplo:

- Algoritmos gulosos (*greedy*) e heurísticas de construção, em geral;
- Algoritmos de relaxação.

Os problemas de *design* (que na sua maioria são problemas multiobjetivos), são classificados como pertencentes a categoria dos Algoritmos Inadequados, dado que, em geral, os algoritmos existentes geram soluções infactíveis que precisam ser continuamente ajustadas até a obtenção de uma solução factível.

Vale ressaltar, entretanto, que um problema pode ter algoritmos pertencentes a mais de uma categoria. Um caso típico acontece com os problemas de otimização. Se o algoritmo pode garantir o ótimo global (algoritmo exato), então o problema pertence a categoria dos Algoritmos Adequados (supondo que se tenha o tempo necessário para encontrar a solução ótima, ou ainda, para instâncias *pequenas* ou com características específicas), agora, a maioria dos algoritmos existentes só podem encontrar soluções aproximadas, embora elas sejam factíveis e, neste caso, o problema de otimização se encaixa na categoria dos Algoritmos Inadequados.

A terceira categoria agrega os problemas que possuem apenas **Algoritmos Inaptos** para resolvê-los, ou seja, os algoritmos existentes não garantem sequer a geração de soluções factíveis, e mais, geralmente eles violam *muito* as restrições do problema, de tal forma que as soluções fornecidas venham a pertencer à uma região muito distante da factível. Os problemas desta categoria são muito difíceis de se resolver, tendo sérias restrições (especialmente em relação ao tempo de execução), que somente um algoritmo randômico pode se aventurar a gerar uma solução. Como por exemplo:

- Algoritmos para problemas de previsão do tempo.

Os problemas de previsão de tempo envolvem equações diferenciais não-lineares com milhões de variáveis que são atualizadas dinamicamente [Bar92].

A categoria dos problemas adequados para a resolução através de ***A-Teams*** é a de ***Algoritmos Inadequados***. Isto se deve aos seguintes fatos:

1. Muitos problemas de aplicação prática se encaixam nesta categoria, especificamente, a maioria dos Problemas de Otimização Combinatória. Isto é devido, principalmente, ao limite de tempo disponível para resolvê-los;

2. Tais problemas já possuem uma estrutura relativamente bem conhecida, devido aos estudos que resultaram no desenvolvimento de algoritmos capazes de encontrar soluções factíveis para os mesmos.

Espaço das Variáveis

Os problemas podem ser decompostos em regiões distintas, que representam o tipo de variável que está sendo abordada num determinado momento do processo de execução de um algoritmo, incluindo as variáveis de entrada e as de saída. Souza [Sou93] definiu três *espaços* de variáveis:

1. Espaço das Variáveis de Entrada;

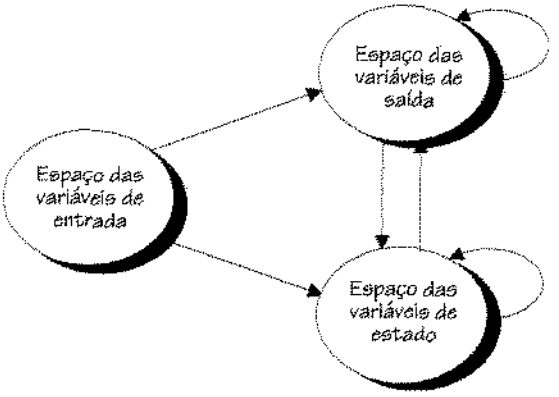
2. Espaço das Variáveis de Estado;

3. Espaço das Variáveis de Saída.

O **Espaço das Variáveis de Entrada** engloba todas as variáveis que definem o problema. O **Espaço das Variáveis de Estado** engloba todas as variáveis relevantes durante o processo de resolução do problema e que podem representar uma solução factível ou não. E, o **Espaço das Variáveis de Saída** engloba todas as variáveis que especificam uma solução para o problema.

Na fig 5.3 pode ser observada uma representação gráfica destes espaços de decomposição de problemas. Os espaços são conectados por setas, que representam a existência de algoritmos que mapeiam as variáveis de um *espaço* para outro. O espaço de variáveis de entrada é fixo, pois representa a definição da instância do problema e, portanto, não pode sofrer alterações.

Figura 5.3: Diagrama de Decomposição de Problemas. As setas representam os algoritmos disponíveis para a resolução de um problema que mapeiam elementos de um espaço para outro.



O objetivo desta decomposição é verificar a existência de algoritmos que atuam nestes espaços. E, a existência de ciclos neste diagrama nos garante que existe a possibilidade de se gerar novas soluções a partir de soluções já existentes ou a partir de alguma representação equivalente.

No caso do MDTSP, por exemplo, o espaço de variáveis de entrada representa o conjunto de cidades a ser visitado e as matrizes de distância relacionadas com este conjunto. O espaço de variáveis de estado representa, por exemplo, subconjuntos das cidades do problema ou o conjunto total de cidades possuindo mais de um ciclo (soluções infactíveis em geral). E, o espaço das variáveis de saída representa seqüências contendo todas as cidades uma única vez (ciclos Hamiltonianos) e os custos dados por cada uma das matrizes de distância do problema.

O fato de existir algoritmos atuando nestes espaços - algoritmos que constroem, modificam e geram novas soluções - é de fundamental importância para A-Teams, como será mostrado na subseção seguinte.

Os problemas que pertencem à categoria dos Algoritmos Inadequados podem ser transformados num MAP desde que se criem algoritmos que gerem ciclos no diagrama de decomposição. Tal possibilidade, faz com que a maioria dos problemas práticos possam ser vistos como um MAP.

O problema multiobjetivo escolhido, o Multi-Distance Traveling Salesman Problem - MDTSP (ver Cap.04), é um Problema Multi-Algoritmo (MAP). É um MAP porque satisfaz as duas características básicas: não existia nenhuma tentativa de resolução exata deste, visto que o mesmo está sendo proposto neste trabalho e existe algoritmos heurísticos (vimos que os algoritmos heurísticos para o TSP podem ser adaptados para o MDTSP - ver 4.2.5) que garantem soluções factíveis próximas do ótimo (heurísticas com

garantia de performance [LLKS85]); e, existe a formação de ciclos no diagrama de decomposição do problema.

5.5.3.1. Problemas Multi-Algoritmos

Um Problema Multi-algoritmo (Multi-Algorithm Problem - MAP [Sou93]) é caracterizado por:

1. Pertencer a categoria dos ***Algoritmos Inadequados***;
2. Possuir pelo menos um ***ciclo no diagrama de decomposição do problema.***

O fato do problema pertencer à categoria dos Algoritmos Inadequados, indica a existência de algoritmos que geram soluções factíveis para o mesmo no tempo disponível para sua resolução, apesar de não se garantir o ótimo global. E, o fato de existir pelo menos um ciclo no diagrama de decomposição de problemas, indica a existência de algoritmos que podem ser executados interativamente para gerarem soluções.

Tais características são fundamentais para o desenvolvimento de um A-Team e serão abordadas em detalhes a seguir.

5.5.4. Classificação dos Algoritmos Existentes

Dado que se sabe *qual o problema* a ser resolvido e que se tenha constatado a adequabilidade do problema para a resolução através de A-Teams, resta identificar quais os algoritmos heurísticos existentes que se propõem a resolvê-lo, ou que, pelo menos, podem ser adaptados para isto.

Os algoritmos são classificados de acordo com a abordagem utilizada para a geração de soluções e também de acordo com o tipo de solução gerada (factível ou não). A classificação utilizada foi a adotada por Peixoto [Pei95], sendo que foram incluídas duas novas classes: algoritmos de *Busca Global* e algoritmos *Randômicos*.

Algoritmos de Construção

São algoritmos que constroem uma solução factível através da inserção de cada um dos elementos do problema individualmente. O processo pode ser iniciado a partir de uma solução parcial. Geralmente, uma solução completa só é obtida no final do procedimento.Exemplos: **algoritmos gulosos** (*greedy*) em geral [Cor91]; ***Nearest Insertion*** - **NN** para o TSP [JRR94].

Algoritmos de Melhoria

São algoritmos que iniciam com uma solução factível e tentam melhorá-las através de sucessivas trocas e rearranjos de seus elementos, fornecendo uma solução completa e melhorada como resultado. Tais algoritmos geram inúmeras soluções durante cada execução, que são de qualidade monotonicamente crescente. São também conhecidas por *algoritmos de busca local*, por realizarem uma busca exaustiva na vizinhança da solução inicial até alcançar o ótimo local. Exemplo: heurísticas que utilizam o princípio *r-opt*, tal como ***2-opt*** e ***3opt*** para o TSP [JRR94, Ben90].

Algoritmos de Relaxação

São algoritmos que desconsideram algumas das restrições do problema (aumentam o espaço de soluções do problema original), gerando um novo problema menos restrito e portanto mais fácil de ser resolvido. A solução resultante geralmente é infactível, pertencendo à região de soluções adicionada à região factível do problema original, como resultado da relaxação. Os algoritmos de relaxação geram Limites Inferiores para o problema original. Tal solução infactível pode ser denominada de solução relaxada. Exemplo: ***assignment problem*** (uma relaxação do TSP, sem as restrições de eliminação de sub-trajetórias e a restrição de integralização das variáveis) [LLKS85]; algoritmo de ***Held-Karp*** para o TSP (outro algoritmo muito usado para cálculos de limites inferiores do TSP, que utiliza o método de subgradiente) [JRR94].

Algoritmos de Factibilidade

Os algoritmos de factibilidade transformam uma solução infactível (fornecida pelos algoritmos de relaxação, por exemplo) numa solução factível para o problema. Tais algoritmos fornecem uma solução completa como resultado final.

Algoritmos Exatos

Tal classe, para o TSP, já foi abordada na subseção 2.4.1. Os algoritmos exatos fornecem como resultado final, a solução ótima para o problema, portanto, uma solução completa. Para problemas NP-difíceis, tais algoritmos levam tempo exponencial em relação ao tamanho da entrada para encontrar o ótimo, no pior caso. Isto restringe o uso destes algoritmos em alguns problemas e sob certas circunstâncias, como por exemplo, quando estão disponíveis: soluções muito próximas do ótimo; bons limites inferiores; quando a instância não é muito grande; ou quando a instância possui características que tornem sua resolução mais fácil. Exemplo: ***branch-and-bound*** para problemas formulados em programação inteira [JRR94].

Algoritmos Primais-Duais

Tais algoritmos convertem o problema original (problema primal) no seu problema dual. Dado que o problema a ser resolvido tenha uma formulação em programação inteira, uma estratégia para a obtenção de limites inferiores é a transformação de tal formulação numa formulação em programação linear (relaxação do problema) e, em seguida, a transformação de tal problema primal no seu problema dual [NW88]. Desta forma, uma solução para o problema original é convertida heuristicamente numa solução para o problema relaxado, sendo finalmente convertida na correspondente dual. Estas soluções duais podem ser utilizadas pelos algoritmos (de melhoria, por exemplo) e, assim, melhorar os limites inferiores, que por sua vez podem ser utilizados por outros algoritmos. Tal estratégia, utilização de limites superiores e inferiores, foi utilizada por Longo [Lon95] na resolução através de A-Teams do Problema de Recobrimento de Conjuntos.

Algoritmos Duais-Primais

Tais algoritmos convertem o problema dual em seu problema primal, atuando de maneira inversa ao procedimento dos algoritmos Primais-Duais. O resultado final de sua execução é uma solução completa ou factível para o problema.

Algoritmos de Partibilidade

Os algoritmos de partibilidade dividem o problema original em sub-problemas, onde cada um é resolvido separadamente um do outro. O resultado deste algoritmo, então, é um conjunto de sub-soluções que devem ser combinadas de alguma forma (por outros algoritmos) para se obter a solução do problema original. Exemplos: algoritmos que utilizam a técnica de **divisão e conquista** (neste caso, a parte de *divisão* seria um algoritmo de partibilidade), como os algoritmos recursivos; e, algumas heurísticas para o Problema de Roteamento de Veículos - VRP, que tratam o problema como sendo uma combinação do problema de alocação de recursos com o TSP [GM74].

Algoritmos de Composição

Os algoritmos de composição têm como dados de entrada sub-soluções (geradas pelos algoritmos de partibilidade, por exemplo) que são combinadas por eles, de tal forma a produzirem uma solução completa (factível) para o problema. Exemplo: a parte de **conquista** dos algoritmos que utilizam a técnica de divisão e conquista; e, novamente, algumas heurísticas para o Problema de Roteamento de Veículos - VRP que combinam as soluções para o TSP e para o problema de alocação de recursos, de forma a gerar uma solução factível para o VRP.

Algoritmos de Consenso

Os algoritmos de consenso tentam obter um *consenso de qualidade* entre duas ou mais soluções factíveis para o problema, com o intuito de gerar soluções potencialmente boas. O resultado de tal algoritmo pode ser tanto uma solução completa quanto uma solução parcial.

Tais classes de algoritmos podem ser subdivididas em duas outras, dependendo do tipo de consenso realizado (que informações são extraídas das soluções de entrada):

Algoritmos de Consenso por União

Os algoritmos de consenso por união geram uma nova solução a partir dos elementos do problema constantes na união das duas ou mais soluções de entrada, ou seja, uma *super-solução* é construída contendo todas as soluções de entrada para em seguida ser transformada (por algum critério específico) numa solução factível para o problema. Exemplo: o algoritmo ***Mixer*** para o TSP, desenvolvido por Souza especialmente para ser aplicado num A-Team [Sou93].

Algoritmos de Consenso por Interseção

Os algoritmos de consenso por interseção extraem as características presentes na interseção de *todas* as soluções fornecidas como entrada, ou seja, as partes *em comum* de

duas ou mais soluções são aproveitadas para a geração de uma nova solução. Exemplo: o algoritmo de deconstrução (DEC) para o TSP, desenvolvido por Souza especialmente para ser aplicado num A-Team [Sou93].

Algoritmos Interativos

Os algoritmos interativos são aqueles que permitem a intervenção humana (ou de qualquer agente externo, como algum *sistema de inteligência artificial*) durante a sua execução, que auxiliam no processo de geração e análise das soluções. O resultado de tais algoritmos é uma, ou mais, solução completa. Tal tipo de algoritmo é freqüentemente aplicado a problemas multiobjetivos. Exemplo: algoritmo ***STEM***; e, algoritmo de ***Zionts-Wallenius*** [Ste86].

No capítulo 3 podem ser encontrados estes e outros exemplos de algoritmos interativos para problemas multiobjetivos disponíveis na literatura (ver 3.6.2.3).

Algoritmos de *Busca Global*

Os algoritmos de busca global, nada mais são do que algoritmos que fazem uso de estratégias para escaparem de ótimos locais e realizarem uma busca em diversos pontos da região factível de soluções do problema. Esta definição, como observado no capítulo 2, corresponde aos algoritmos heurísticos que incorporam em seus procedimentos outras heurísticas, ou seja, são as conhecidas *meta-heurísticas*. Exemplo: ***Tabu Search***, ***Simulated Annealing***, ***Genetic Algorithms*** [JRR94] e ***GRASP*** [FR95a].

Algoritmos Randômicos

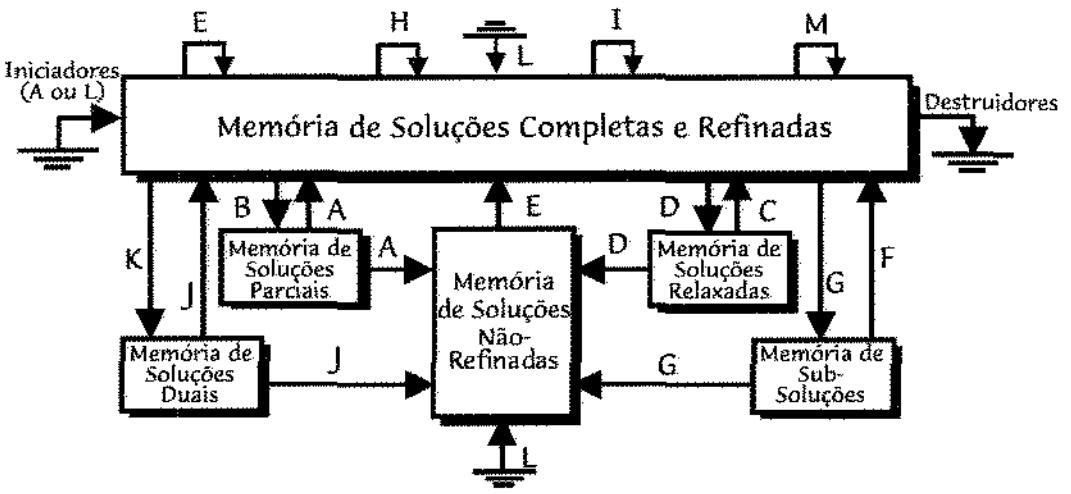
Os algoritmos randômicos são aqueles que geram soluções de maneira completamente aleatória e, portanto, sem nenhuma garantia de qualidade. No geral, as soluções obtidas são muito ruins. Tais algoritmos são interessantes para elevar o índice de diversidade de um A-Team, mas, seu uso deve ser moderado, como será visto nas subseções seguintes.

5.5.5. Estrutura Geral

Peixoto [Pei95] definiu, na sua metodologia de especificação, uma estrutura geral de A-Teams para Problemas de Otimização Combinatória. Tal estrutura é apenas um exemplo de como todas as diferentes classes de algoritmos podem ser combinadas e deve ser usada como *ponto de partida* para o desenvolvimento de um A-Team, até porque nem todos os problemas possuem algoritmos em todas as classes especificadas.

A estrutura geral (fig. 5.4) ilustra como os diversos tipos de agentes podem interagir através das memórias compartilhadas. As duas novas classes de algoritmos apresentadas, de Busca Global e Randômicos, estão incorporadas nesta estrutura.

Figura 5.4: Estrutura Geral de A-Teams. São mostrados possíveis fluxos de dados envolvendo todas as classes de algoritmos abordadas e os tipos de memórias adequados para suportar todos os tipos de soluções geradas.



A	Agentes de Construção	H	Agentes Interativos
B	Agentes de Consenso	I	Agentes Exatos
C	Agentes de Relaxamento	J	Agentes Primal-Dual
D	Agentes de Factibilidade	K	Agentes Dual-Primal
E	Agentes de Melhoria	L	Agentes Randômicos
F	Agentes de Partibilidade	M	Agentes de Busca Global
G	Agentes de Composição		

Nota-se que esta estrutura possui seis distintos tipos de memória. Tais memórias se diferenciam pelo *tipo das soluções armazenadas*. Estes tipos são os seguintes:

Soluções Completa - são soluções factíveis para o problema que está sendo resolvido;

Soluções Parciais - são soluções infactíveis que não possuem o número necessário de elementos (arestas, vértices, etc), portanto, são consideradas soluções incompletas;

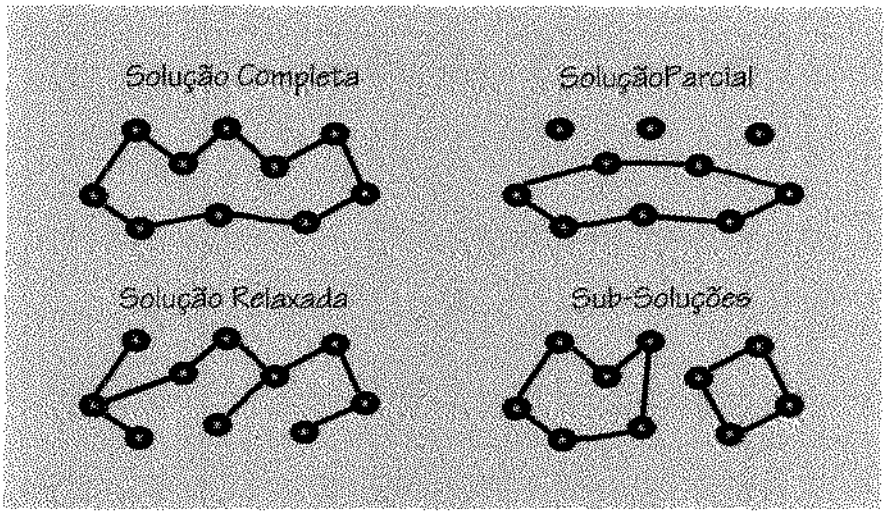
Soluções Relaxadas - são soluções infactíveis que violam alguma restrição do problema;

Soluções Duais - são soluções para o problema na sua verão dual, portanto, infactíveis para o problema original;

Subsoluções - são soluções para sub-problemas do problema original, portanto, infactíveis.

Para exemplificar os tipos de soluções que podem ser armazenadas nas memórias de um A-Team, consideremos o TSP, ressaltando que tais tipos de representações também são válidas para o MDTSP, uma vez que as funções objetivo não são envolvidas nesta representação.

Figura 5.5: Tipos de soluções para o TSP e MDTSP.



5.5.6. Especificação das Memórias

As memórias compartilhadas de soluções constituem um ponto crucial no desenvolvimento de um A-Team. Nelas são depositadas todas as soluções geradas pelos agentes; e, dependendo da qualidade das soluções armazenadas inicialmente, do tamanho da memória e da estrutura de organização das soluções durante o processo interativo, se consegue uma melhora significativa nos resultados.

As memórias devem ser especificadas com base no *tipo de problema* que se deseja solucionar e no *tipo de abordagem* que se utilizará para resolvê-lo. Se o problema for monobjetivo ou for um problema multiobjetivo abordado como um problema monobjetivo (redução para uma única função objetivo), a estrutura da memória é a tradicional, uma simples lista seqüencial de soluções ordenadas segundo algum critério de qualidade, que geralmente é o valor fornecido pela função objetivo. Agora, se o problema for multiobjetivo onde se considere *todas* as funções objetivo para serem simultaneamente otimizadas, então a memória adquire uma estrutura completamente diferente, onde o critério de qualidade é fornecido pela propriedade de dominância entre conjuntos de soluções.

Nesta seção, serão fornecidos os diferentes tipos de memórias, conforme observado na estrutura geral de A-Teams (fig. 5.4), bem como estratégias para o preenchimento inicial da memória compartilhada de soluções, escolha do tamanho da memória e estratégias para a organização das soluções na memória.

Na seção 5.6 será abordada a estrutura para organização das soluções na memória, no caso específico de problemas multiobjetivos.

5.5.6.1. Tipos de Memória

As memórias de um A-Team se diferenciam pelo *tipo* e pela *qualidade* das soluções que elas armazenam. A seguir, são descritas as funções que cada tipo de

memória pode desempenhar, bem como quais classes de algoritmos podem utilizá-las (fig. 5.4).

A *Memória de Soluções Completas e Refinadas* é a mais importante de um A-Team, pois, nela são armazenadas as melhores soluções obtidas. Para tal memória, especificações mais cuidadosas serão feitas. Eventualmente esta memória poderá ser referida no texto por *memória principal* do A-Team.

Todos os outros tipos de memória descritos a seguir, funcionam como simples *buffers* de soluções e, exceto a *Memória de Soluções Completas e Não-Refinadas* (armazenam soluções factíveis mas de baixa qualidade), todas as outras armazenam soluções infactíveis para o problema.

Memória de Soluções Completas e Refinadas

É a memória principal do A-Team, contendo soluções completas que são constantemente refinadas (melhoradas) por um ou mais algoritmos. As soluções desta memória servem de entrada para os algoritmos de consenso, os de relaxamento, os de melhoria, os exatos, os interativos, os de partibilidade e os dual-primal. E, podem armazenar as soluções geradas pelos algoritmos de construção, de factibilidade, de melhoria, de composição, primal-dual, interativos, exatos, de busca global e os randômicos.

Esta é a única memória que necessita ser preenchida com soluções antes do processo interativo e iterativo começar (ver seções seguintes) e tal preenchimento inicial pode ser feito pelos algoritmos de construção e pelos algoritmos randômicos (ver 5.5.6.3).

Este tipo de memória é fundamental num A-Team e *deve* estar presente em qualquer estrutura projetada. Isto porque é nesta memória que estão contidas as melhores soluções para o problema. Portanto, as soluções contidas nesta memória no final da execução, são as melhores soluções obtidas durante *todo* o processo de execução do A-Team.

Memória de Soluções Completas e Não-Refinadas

São memórias onde estão armazenadas soluções completas que não passam por nenhum processo de refinamento. A idéia para a manutenção desta memória é que seja garantida a exploração de soluções consideradas ruins pelos algoritmos, visto que as soluções contidas na *memória de soluções completas e refinadas* são melhoradas rapidamente (mesmo com a adoção de distintas políticas de destruição) e torna-se impossível a inserção de soluções factíveis de baixa qualidade nesta memória. Já na memória de soluções completas e não-refinadas, é possível a exploração da vizinhança de todas as soluções, de baixa qualidade, contidas nela, possibilitando, principalmente, se escapar de mínimos locais e introduzir soluções diversas na memória principal. Os agentes de melhoria, os exatos e os interativos não depositam suas soluções (de alta qualidade) nesta memória, sendo que suas soluções servem de entrada para os algoritmos de melhoria e armazenam as soluções geradas pelos algoritmos de construção, de factibilidade, de composição e primal-dual. Eventualmente, os algoritmos randômicos também podem atuar sobre elas, caso se queira um índice de diversidade muito elevado.

Memória de Soluções Parciais

São memórias que possuem soluções infactíveis (incompletas) para o problema a ser resolvido. Servem de depósito para as soluções geradas pelos algoritmos de consenso e fornecem soluções de entrada para alguns algoritmos de construção.

Memória de Soluções Relaxadas

Tais memórias armazenam as soluções infactíveis geradas pelos algoritmos de relaxação. Elas fornecem as soluções de entrada para os algoritmos de factibilidade.

Memória de Soluções Duais

São memórias que contém soluções completas para o problema na sua versão dual, que são geradas pelos algoritmos primais-duais. Tal memória pode ser utilizada pelos algoritmos duais-primais, que transformam as soluções duais em primais.

Memória de Subsoluções

Estas memórias guardam sub-soluções, ou seja, soluções para sub-problemas do problema original. Os algoritmos de partibilidade armazenam suas sub-soluções nesta memória, podendo ser utilizada pelos agentes de composição, que utilizam tais sub-soluções como entrada em seus procedimentos.

5.5.6.2. Tamanho da Memória

Por *tamanho* da memória entende-se o número máximo de soluções que uma determinada memória do A-Team pode conter, ou seja, é a capacidade máxima de uma memória compartilhada. Este tamanho pode variar entre uma solução até o limite máximo imposto pelo ambiente computacional no qual o A-Team será implementado.

No caso da *Memória de Soluções Completas e Refinadas* (memória principal do A-Team), o tamanho é um importante fator no desempenho de um A-Team. No caso dos outros tipos de memórias, elas funcionam como simples *buffers* de soluções e, portanto, para a maioria dos problemas abordados seus tamanhos não causam influência nenhuma. Para estas memórias, geralmente se define um tamanho igual ao tamanho da instância que será abordada, ou ainda, atribui-se o mesmo tamanho dado à memória principal (de soluções completas e refinadas) [Pei95, Sou93].

Para a Memória de Soluções Completas e Refinadas, torna-se necessário alguns esclarecimentos. As soluções contidas nesta memória devem possibilitar tanta *diversidade* quanto possível (soluções com características bem distintas umas da outras) e tanta *qualidade* quanto possível (soluções relativamente boas - como, por exemplo, diversos mínimos locais). Estas duas características *ao mesmo tempo*. As justificativas podem ser explicadas da seguinte maneira: se as soluções contidas na memória forem bem distintas umas das outras (e isto significa que elas pertencem à regiões distintas do espaço de soluções factíveis do problema), é garantida a busca em diferentes pontos da região factível; e, se existirem soluções potencialmente boas, é garantida uma exploração de suas vizinhanças, com a conseqüente obtenção de mínimos locais, quando não, a obtenção do ótimo global (ou soluções do Pareto Ótimo, para o caso de problemas multiobjetivos).

Sabendo-se as características desejáveis do conjunto de soluções da memória de um A-Team, uma análise do tamanho que a mesma deve possuir torna-se mais fácil: se a memória possuir um tamanho muito pequeno (por exemplo, menos de 50% do tamanho da instância), como as soluções são ordenadas qualitativamente, rapidamente a memória conterá somente soluções boas e com grandes chances de pertencerem à regiões muito próximas, com a conseqüente impossibilidade de se escapar do possível mínimo local; por outro lado, se a memória possuir um tamanho muito grande, a possibilidade da existência de muitas soluções ruins é alta e, portanto, mesmo depois de um longo tempo de execução a diversidade tenderá a permanecer alta.

Foi afirmado nos parágrafos anteriores, e também com base em estudos anteriores [Sou93, Pei95], que um tamanho *pequeno* de memória seria algo em torno de 50% ou menos do tamanho da instância do problema. Agora, em relação a um tamanho *grande* de memória, se recai numa questão subjetiva. Nos A-Teams desenvolvidos anteriormente, o tamanho da memória ficou estipulado em n (tamanho da instância) ou $2n$ (o dobro do tamanho da instância) [Sou93, PS95b, CS95]. Mais do que isto, era considerado uma memória *grande* e portanto seu uso acarretaria todos os problemas já relacionados. No caso de *problemas multiobjetivos*, os resultados empíricos das experiências anteriores não se aplicam mais, pois não se procura por uma única solução ótima, e sim, por todo um conjunto de soluções não-dominadas por nenhuma outra solução, o Pareto Ótimo do problema. Este conjunto pode ser muito grande e para instâncias pequenas, chega a ser maior do que $2n$ (ver Cap. 7), o que já dá respaldo para a adoção de um tamanho maior do que este.

Portanto, com base na análise empírica realizada com o MDTSP (ver Cap. 6 e 7), o tamanho da memória de um A-Team para problemas multiobjetivos deve ficar entre $3n$ (o triplo do tamanho da instância) e $5n$ (cinco vezes o tamanho da instância), pelo menos, isto com base nas restrições de memória e de processamento impostas pelo ambiente computacional onde se implementará o A-Team. Mais uma vez, tais afirmações não passam de sugestões a serem adotadas na configuração básica inicial, onde os testes empíricos deverão ser aplicados e ajustes realizados.

5.5.6.3. Estratégias para o Preenchimento Inicial

Num A-Team, os agentes se comunicam através de memórias compartilhadas num processo totalmente assíncrono. *E como iniciar este processo?* Dado que nenhuma ordem para a execução dos agentes é preestabelecida, algumas soluções *já* devem estar disponíveis no início do processo interativo (e iterativo). A qualidade das primeiras soluções armazenadas na memória influenciam na qualidade das outras soluções obtidas durante o processo. Por isto, algumas estratégias para o preenchimento inicial da memória tornam-se necessárias.

Vale ressaltar, antes de serem abordadas as estratégias de preenchimento inicial, que somente a *Memória de Soluções Completas e Refinadas* precisa ser preenchida inicialmente (ver figura 5.5). Todos os outros tipos de memórias podem ser preenchidas *durante* o processo iterativo pelos próprios agentes que manipulam as soluções das mesmas.

Dois tipos de critérios devem ser, então, discutidos:

- **Quanto à *quantidade* de soluções iniciais;**
- **Quanto à *qualidade* das soluções iniciais.**

Quanto à quantidade de soluções iniciais, Souza [Sou93] observou empiricamente que não se percebia significativas alterações no número de soluções geradas para iniciar o processo iterativo, a menos que este número fosse menor do que 10% do tamanho total da memória. O preenchimento completo da memória também pode ser desnecessário, pois, quando os agentes começarem a gerar outras soluções, as piores soluções geradas inicialmente podem ser imediatamente trocadas (para dar lugar a outras melhores, já que a memória está totalmente preenchida) sem que nenhum agente tenha oportunidade de selecioná-la e explorar sua vizinhança. O ideal seria um número inicial de soluções relativo ao intervalo entre *50% e 90% do tamanho total da memória.*

Quanto à qualidade das soluções iniciais, dependerá do tipo dos agentes utilizados para a geração de tais soluções. Dessa forma, três diferentes estratégias podem ser observadas:

Preenchimento por Agentes Randômicos - a memória é preenchida por soluções geradas por agentes randômicos. Esta estratégia tem a vantagem de possibilitar um rápido preenchimento da memória com *alto índice de diversidade* (as soluções geradas aleatoriamente tendem a ser completamente diferentes umas das outras). A desvantagem é que as *soluções* geradas randomicamente são muito *ruins* e, desta forma, os agentes que as selecionarem terão mais dificuldades em obter boas soluções.

Preenchimento por Agentes de Construção - a memória é preenchida por soluções geradas por algoritmos de construção. Esta estratégia tem a vantagem de produzir soluções que não são tão ruins quanto as soluções geradas por agentes randômicos e, desta forma, os agentes que as selecionarem podem alcançar regiões mais interessantes do espaço de busca em menos tempo. A desvantagem fica por conta da falta de diversidade, se for usado somente um algoritmo de construção.

O ideal seria o uso de pelo menos dois algoritmos de construção que usem abordagens distintas e que, portanto, gerem soluções com características bem diferentes, de tal forma a permitir uma *maior diversidade de boas soluções* na memória inicialmente.

Preenchimento por Agentes de Construção e Randômicos - a memória é preenchida uma parte por soluções geradas por agentes randômicos e o restante por agentes de construção. Neste caso, é garantida a presença de soluções boas e ruins, garantindo-se um *mínimo de diversidade* e, ao mesmo tempo, um *mínimo de qualidade*. Esta abordagem é indicada somente quando não estiverem disponíveis dois (ou mais) algoritmos de construção com abordagens distintas para a geração de soluções.

5.5.6.4. Estratégias para a Organização das Soluções

A estrutura interna de uma memória pode facilitar o processo de seleção e eliminação de soluções pelos agentes. Cada agente do time é independente para *enxergar* as soluções na memória da maneira que melhor lhe convier, mas, se as soluções puderem ser organizadas na memória segundo algum critério de qualidade que favoreça todos os agentes de uma vez, a concepção e implementação do A-Team é facilitada. No caso de

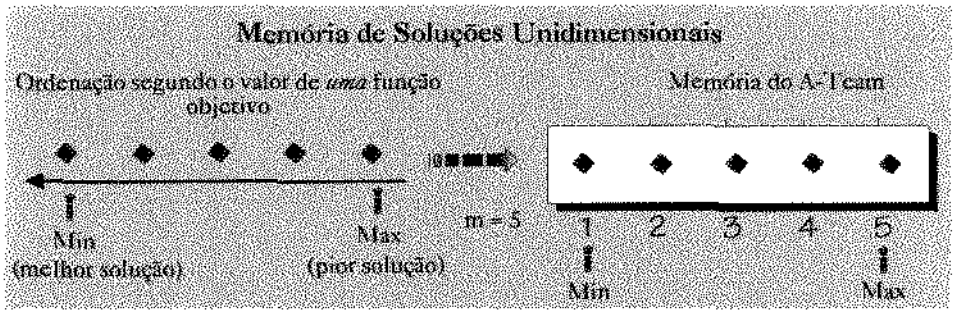
problemas monobjetivos, tal critério é, geralmente, o valor fornecido pela única função objetivo

No caso de problemas com *várias funções objetivo*, a estrutura interna da memória dependerá do tipo de abordagem utilizada para a resolução do problema. Se a abordagem utilizada for uma das duas mais utilizadas na prática, ou seja, se as várias funções objetivo do problema forem transformadas numa única função objetivo (seja pela priorização de um deles, seja pela combinação de todos num só), a estrutura interna da memória adotada é igual a estrutura descrita para problemas monobjetivos, dado que o problema é visto como tendo uma única função. Agora, neste trabalho, está sendo proposto um novo tipo de abordagem para a resolução de problemas multiobjetivos através de A-Team e, como conseqüência, uma estrutura interna completamente diferente foi elaborada. Tais questões são abordadas a seguir.

Memória de Soluções Unidimensionais: para problemas monobjetivos ou multiobjetivos alterados

No caso de problemas com uma *única função objetivo* ou no caso de problemas multiobjetivos *alterados* para um problema monobjetivo, a memória é dividida em *m posições* (*m* é o tamanho da memória), onde cada posição contém *uma* única solução e as informações associadas a ela (fig.5.6). Por exemplo: valor da função objetivo; estrutura de representação da solução; agente gerador; etc. As soluções são, neste caso, organizadas em ordem decrescente de qualidade (na primeira posição da memória está a melhor solução e, na última, a pior). A *qualidade* das soluções é geralmente relacionada ao valor da *única* função objetivo, ou seja, as soluções são ordenadas em ordem decrescente segundo os valores fornecidos pela função objetivo (fig.5.6).

Figura 5.6: Esquema da organização de soluções num A-Team para Problemas Monobjetivos e Multiobjetivos alterados. Esta é a estrutura tradicional de memória de A-Teams, onde as soluções são organizadas na memória em ordem seqüencialmente crescente de qualidade, segundo o valor da *única* função objetivo do problema.



Memória de Soluções Multidimensionais: para problemas multiobjetivos

Como já comentado (ver Cap.3), o problema das abordagens para a resolução de problemas multiobjetivos encontradas na literatura é que a maioria delas *transformam* os objetivos do problema num único objetivo. Isto causa *distorções* na definição do

problema e, conseqüentemente, nas soluções obtidas. O ideal seria manipular todos os objetivos simultaneamente, de maneira que nenhum deles fosse favorecido ou que nenhum fosse prejudicado. Desta forma, a organização de soluções na memória usada no caso de problemas monobjetivos não faz sentido, uma vez que as soluções são ordenadas segundo o valor de uma única função objetivo.

Como fazer, então, para se estabelecer uma ordem de qualidade entre as soluções, no caso de se ter k valores fornecidos por k funções objetivo, considerando todas as k funções simultaneamente?

Deve-se, primeiramente, analisar como avaliar a qualidade das soluções:

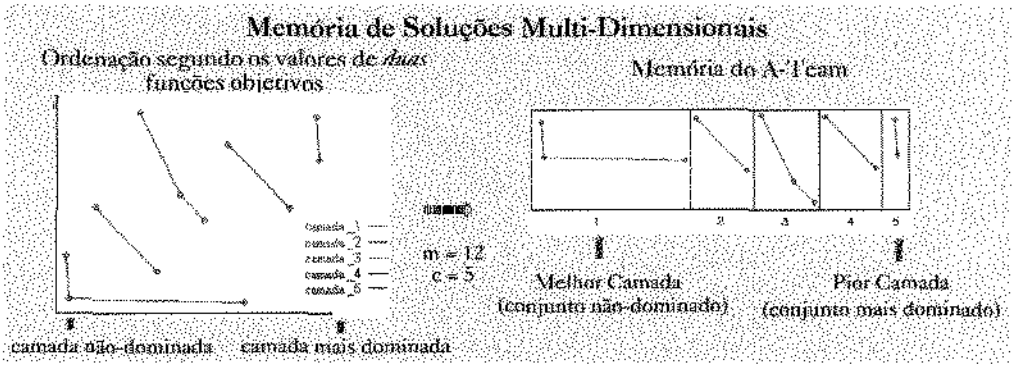
Dado um conjunto de soluções k-dimensionais, com k valores a serem analisados, como saber que uma solução é melhor do que outra? Como organizá-las qualitativamente?

As soluções *k*-dimensionais possuem a propriedade de serem dominadas ou não por outras soluções (ver Cap.3). Com base nisto, se pensou numa estrutura onde as soluções pudessem ser dispostas numa certa ordem de dominância: em primeiro lugar, as soluções não-dominadas por nenhuma outra solução gerada; e, por último, as soluções que não dominavam nenhuma outra solução e eram dominadas por pelo menos uma outra. E, como as soluções que não se dominam são consideradas equivalentes (de mesma qualidade), o novo critério de qualidade deveria considerar como elemento de comparação um *conjunto* de soluções, ao invés de uma única solução. Desta forma, se estabeleceu o critério de qualidade para soluções multidimensionais: segundo o **grau de dominância** que elas possuem.

Sendo assim, a estrutura interna da memória compartilhada de soluções *k*-dimensionais pode ser projetada da seguinte forma: a memória é dividida dinamicamente em *c* posições, onde *c* é o número de camadas (ou conjuntos) de soluções equivalentes (que não se dominam) em determinado momento da execução do A-Team. Cada posição, por sua vez, é subdividida em *s* sub-posições, onde cada sub-posição pode armazenar somente uma única solução (fig. 5.7), $1 \leq s \leq m$.

O número de camadas é dinâmico (varia em tempo de execução). O número máximo (*m*) de soluções é fixo, mas, o conjunto de soluções na memória também é dinâmico (soluções são inseridas e eliminadas a todo instante, durante a execução).

Figura 5.7: Esquema da organização de soluções num A-Team para Problemas Multiobjetivos. Esta é a estrutura de memória proposta, onde soluções *k*-dimensionais são organizadas em camadas de soluções não-dominadas, segundo os valores fornecidos pelas *k* funções objetivo do problema multiobjetivo.



Na primeira posição da memória fica contido o melhor conjunto não-dominado obtido (conjunto de soluções que domina todos os outros conjuntos). Na segunda posição, fica contido o conjunto de soluções que é dominado pelo melhor conjunto obtido, mas que domina todos os outros conjuntos restantes, e assim por diante, até chegarmos à última posição, onde fica contido o pior conjunto de soluções, que não domina nenhum outro conjunto e que é dominado por todos os outros conjuntos de soluções contidos na memória. Num conjunto, nenhum elemento domina ou é dominado por outro elemento.

As soluções de uma mesma *posição* não se dominam entre si e, portanto, não existe nenhuma ordem de prioridade entre elas: elas são eficientes entre si e podem ser organizadas segundo a política FIFO.

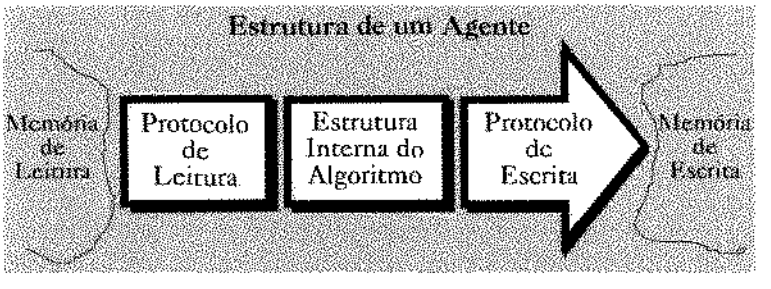
A capacidade máxima da memória é igual a m soluções, dispostas em c posições que variam dinamicamente durante o processo de execução do A-Team. O número máximo de soluções em cada conjunto, número máximo de sub-posições, pode ser no máximo m, já que a memória pode conter somente uma camada de soluções não dominadas no *melhor caso* possível. O número máximo de camadas de conjuntos não-dominadas, número máximo de posições, também pode ser no máximo m, já que pode acontecer de cada solução dominar a solução subseqüente contida na memória, que é o *pior caso* possível.

A adoção de tal organização de soluções é inovadora e possibilitou a obtenção de resultados significativos, como pode ser observado nos capítulos 6 e 7. Na seção 5.6 tal estrutura será mostrada em detalhes, com os algoritmos para manipulação de soluções multidimensionais.

5.5.7. Especificação dos Agentes

Os Agentes de um A-Team são os responsáveis pela geração de todas as soluções, pelo armazenamento das mesmas nas memórias por eles compartilhadas e pelo critério de seleção das soluções usadas por eles para a geração de outras. Cada agente é composto por um algoritmo e por um protocolo que possibilite a comunicação com outros agentes, através das memórias (fig.5.8).

Figura 5.8: Estrutura Geral dos Agentes de um A-Team. Os protocolos de leitura e escrita contém a *interface* necessária para a comunicação com as memórias compartilhadas e embutido no agente encontra-se a estrutura interna do algoritmo.



Cada agente possui uma estratégia para a escolha da solução a ser utilizada por ele. Assim, cabe a uma única classe de agentes, os destruidores, a garantia ou não de permanência de determinada solução na memória.

Dessa forma, os agentes podem ser enxergados como tendo três partes distintas:

Protocolo de leitura - onde está embutida a interface com as memórias em que ele pode retirar soluções, que contém uma estratégia de seleção das soluções que ele utilizará como entrada para o seu processamento interno. Aqui encontra-se embutido também, quando necessário, a transformação do tipo de representação da solução usado pela memória para o tipo de representação usado pelo algoritmo;

Estrutura do algoritmo - onde estão embutidos os comandos e ações para a geração de uma solução (factível ou não) para o problema;

Protocolo de escrita - onde está embutida a interface com as memórias compartilhadas em que o agente pode depositar suas soluções. Neste caso, a decisão sobre a possibilidade ou não da solução gerada permanecer armazenada é tomada unicamente pelos agentes destruidores. Novamente, caso necessário, aqui conterá o procedimento de transformação do tipo de representação usado pelo algoritmo para o tipo usado pela memória onde as soluções podem ser depositadas.

Como existem diferentes estratégias para a seleção e para a destruição de soluções e como tais estratégias são extremamente importantes para o bom desempenho de um A-Team, elas são detalhadas em subseções separadas (ver subseções 5.8 e 5.9).

5.5.7.1. Estratégias para o Desenvolvimento de Agentes

No desenvolvimento de um A-Team para Problemas Multiobjetivos, no que diz respeito aos agentes, levantou-se duas possibilidades:

- **Aplicação Direta dos Algoritmos Existentes;**
- **Elaboração de Novos Algoritmos.**

No caso do problema multiobjetivo a ser resolvido ser uma generalização de algum outro monobjetivo, existe a possibilidade do aproveitamento *direto* dos algoritmos existentes para o caso monobjetivo. Uma versão do algoritmo pode ser aplicada a cada uma das funções e, desta forma, manipula-se através do A-Team todos os objetivos do problema sem alterações dos algoritmos existentes para o problema monobjetivo.

Para uma aplicação mais sofisticada, pode-se adaptar os algoritmos existentes de tal forma que eles sejam capazes de manipular todos os objetivos simultaneamente. Isto pode ser uma tarefa nada trivial, entretanto, pelo menos para os algoritmos mais simples é possível a sua realização.

Caso o problema multiobjetivo a ser resolvido seja muito intrincado e não exista nem mesmo algoritmos que manipulem um dos objetivos separadamente, se poderia dedicar esforços em desenvolver algoritmos para cada um dos objetivos (que é uma tarefa menos árdua do que a de desenvolver um algoritmo que manipule todos os objetivos) e deixar para o A-Team a tarefa de manipular todos os objetivos em conjunto, com o tratamento das soluções multidimensionais, de forma a obter soluções de compromisso para todos os objetivos envolvidos.

5.5.8. Fluxos dos Dados

O *fluxo de dados* de um A-Team também causa significativa influência na geração dos resultados. Como foi observado na subseção 5.2, uma das três características de um A-Team é a necessidade da existência de pelo menos um fluxo cíclico de dados, de maneira a permitir *feedback* e realimentação entre os algoritmos.

Torna-se necessário, desta forma, selecionar todos os algoritmos disponíveis para a resolução do problema e verificar os possíveis ciclos que podem ser formados, mesmo que para isto seja necessário a criação de algum tipo de algoritmo. Por exemplo, algoritmos de construção do tipo guloso (*greedy*) existem para praticamente todo o tipo de problema, então, dado que se tenha este algoritmo disponível, poderia ser elaborado um algoritmo de consenso (que como vimos, também é um tipo de algoritmo bem flexível) que gerasse uma solução parcial para ser completada pelo algoritmo de construção guloso, possibilitando a formação de um ciclo.

Os agentes devem ser dispostos de maneira que as informações possam fluir sem nenhum problema, ou melhor, o fluxo de dados formado não pode ter nenhum *gargalo* (o fluxo das soluções que são depositadas nas memórias deve ser proporcional ao fluxo das soluções retiradas das mesmas). Por exemplo, um algoritmo que demore muito para gerar uma solução (algoritmos que possuem granulosidade grossa, com o baixo fluxo de comunicação com as memórias de soluções) não deve ser colocado numa posição onde impeça outros algoritmos mais rápidos de atuarem. Geralmente, os algoritmos de granulosidade grossa atuam na Memória de Soluções Completas e Refinadas (alguns de melhoria, por exemplo), lendo e escrevendo soluções somente nela. Deve-se ter o cuidado, também, de não colocar muitos algoritmos de granulosidade fina (aqueles que geram soluções muito rapidamente e, desta forma, o fluxo de comunicação com as memórias de soluções é maior) atuando sobre a mesma memória, pois desta forma a memória pode ser preenchida rapidamente de soluções que não poderão ser exploradas, pois rapidamente serão trocadas por outras, sem a oportunidade de serem selecionadas por outros agentes.

Outros algoritmos, apesar de gerarem soluções para o problema, podem possuir uma granulosidade tão grossa que tornam-se, na maioria dos casos, inviáveis para serem usados em A-Teams: é o caso dos algoritmos adequados (ver 5.5.3), já que eles podem ser usados para selecionarem uma solução *muito* boa e continuarem seu processamento independentemente do A-Team; e, também, é o caso dos algoritmos de busca global (meta-heurísticas). Tais agentes não ajudam no trabalho em grupo, pois iniciam o processo selecionando uma determinada solução e sem mais interagirem com os demais agentes, passam a maioria do tempo de execução do A-Team tentando gerar uma solução. Uma maneira de amenizar este problema, seria estabelecer um critério de parada que satisfizesse um determinado tempo de processamento, caso contrário a melhor solução corrente seria depositada no A-Team. Uma outra maneira, poderia ser o aproveitamento das soluções que vão sendo geradas pelas meta-heurísticas, de forma que fossem depositadas nas memórias do A-Team não apenas a solução final, mas também as intermediárias.

Um estudo sobre um melhor aproveitamento dos agentes de granulosidade grossa em A-Teams poderia ser realizado.

5.5.9. Desenvolvendo Algoritmos para a Formação de Fluxos Cíclicos de Dados: agentes característicos de A-Teams

Os agentes descritos no capítulo 4 (ver subseções 4.1.4 e 4.2.5), para o TSP e MDTSP, geram apenas soluções completas ou relaxadas. Tais tipos de soluções não garantem a formação de um fluxo de dados cíclico no A-Team, um dos três fatores que o caracterizam. Por exemplo: como estabelecer uma interação entre os algoritmos de relaxação e os de melhoria, já que os primeiros geram uma solução infactível e os últimos, contrariamente, recebem como entrada uma solução factível? Ou ainda, como aproveitar melhor a interação entre os algoritmos de construção e os de melhoria, de tal forma que além do aproveitamento de cada uma das soluções geradas pelos algoritmos de construção se possa também aproveitar características em comum de mais de uma destas soluções embutidas numa única solução factível de entrada para os algoritmos de melhoria? Tais questões não seriam viabilizadas sem o desenvolvimento de novos algoritmos que atendam as novas necessidades surgidas com os A-Teams.

Souza [Sou93] apresentou alguns algoritmos característicos para A-Teams, agrupando em duas classes denominadas: de consenso por união e de consenso por interseção. A seguir, serão apresentados alguns dos algoritmos elaborados por Souza, bem como outros elaborados durante o desenvolvimento deste trabalho.

5.5.9.1. Agentes de Consenso por União

Os algoritmos de consenso por união constroem uma solução factível a partir da união das características de duas ou mais soluções completas ou não.

São descritos dois algoritmos de consenso por união: Mixer (MI) e Tree Mixer (TM).

***Mixer* - MI**

O algoritmo *Mixer* foi primeiramente desenvolvido por Souza [Sou93] e se baseia no algoritmo apresentado por Whitley *et al* [WSF89].

Ele recebe como entrada duas ou mais soluções completas (trajetórias) e faz uma união de todas as arestas presentes nelas. Este conjunto de arestas é usado para gerar, cidade a cidade, uma nova trajetória T. Então:

Algoritmo MI

Entrada: T_1 (primeira trajetória inicial), T_2 (segunda trajetória inicial).
Saída: trajetória T.

1. **Seja** T_1 e T_2 duas trajetórias iniciais e sejam E^{T1} e E^{T2} os conjuntos de arestas relativos às trajetórias iniciais.
2. **Faça** $E^T = E^{T1} \cup E^{T2}$;
3. **Seja** g o grau do vértice i (número de vizinhos) no conjunto V;
4. **Inicie** T inserindo o vértice $i \in V$;
5. **Repita**
   5.1. **Seja** $C=\{g_i$ tal que $\exists\, e_{il} \in E$ e $l \in V \setminus V^T\}$;
   5.2. **Se** $C \neq \varnothing$
      5.2.1. **Selecione** um vértice j tal que $g_j = \min_i C$ e insira-o em T;

5.3. Senão
5.3.1. Selecione q.q. $j \in V \setminus V^T$;
5.4. Faça $i=j$;
até $V^T=V$
6. Retorne T

Em outras palavras, o algoritmo MI realiza uma busca por melhores soluções a partir de pontos de vista diferentes sobre o problema (que geram, na maioria das vezes, soluções conflitantes) e que são igualmente considerados.

Pode-se perceber que tal algoritmo apesar de ter sido desenvolvido para apenas uma matriz de distância se encaixa para o caso de múltiplas matrizes, pois trajetórias de entrada que possuam características favoráveis à distintas matrizes de distância são consideradas igualmente e a nova trajetória gerada tenderá a possuir uma mistura das características de tais soluções iniciais. Este fato pode ser verificado no passo 5.3.1 do algoritmo MI, onde qualquer vértice que ainda não pertença a trajetória em construção é selecionado aleatoriamente. Entretanto, o vértice pode ser selecionado de acordo com a melhor distância de compromisso (ver 4.2.5), como por exemplo:

5.3.1. Selecione $j \in V \setminus V^T$ t.q. $dc_{min}(j) = min\{\sum_{p=1}^{d} rank_{ij}^{p} / i \in V \setminus V^T, d \in k\}$

$$dc_{min}(j) = min\{dc_{min}(l) / l \in W\}$$

Neste caso, tal como para os algoritmos de construção apresentados no capítulo 4, se incorpora no algoritmo *Mixer* a capacidade de decisão de compromisso, transformando- o num *Mixer of Compromise* (MIC). Neste exemplo, o vértice escolhido no passo 5.3.1. é aquele de *menor* distância de compromisso.

Tree Mixer - TM

O algoritmo *Tree Mixer* -TM, também primeiramente desenvolvido por Souza [Sou93], é semelhante ao anterior, sendo que ao invés de receber dois ciclos Hamiltonianos de entrada, recebe uma solução completa (ciclo Hamiltoniano) e uma solução relaxada.

Por receber como entrada uma solução infactível, este algoritmo pode ser classificado também como sendo de ***factibilidade.***

Neste caso, o conjunto S é formado pela união de um ciclo Hamiltoniano e de uma 1-Tree, gerada pelo algoritmo Held-Karp (ver Cap.5).

Algoritmo TM

Entrada: T_1 (trajetória inicial), TR_1 (1-Tree).
Saída: trajetória T.

1. **Seja** T_1 e TR_1 duas trajetórias iniciais e sejam E^{T1} e E^{TR1} os conjuntos de arestas relativos às trajetórias iniciais.
2. **Faça** $E^T = E^{T1} \cup E^{TR1}$;
3. **Seja** g o grau do vértice i (número de vizinhos) no conjunto V.
4. **Inicie** T inserindo um vértice $i \in V$.
5. **Repita**
   5.1. **Seja** $C=\{g_l$, tal que $\exists\ e_{il} \in E$ e $l \in V \setminus V^T\}$;
   5.2. **Se** $C \neq \emptyset$

5.2.1. Selecione um vértice j tal que $g_j = \min_i C$ e insira-o em T;

5.3. Senão

➤ **5.3.1. Selecione** q.q. $j \in V \setminus V^T$;

5.4. Faça $i=j$;

até $V^T = V$

6. Retorne T.

Da mesma forma que o algoritmo *Mixer* pode ser transformado no *Mixer of Compromise*, o algoritmo *Tree Mixer* (TM) também pode ser transformado *no Tree Mixer of Compromise* (TMC) com a modificação do passo 5.3.1 como mostrado para o MI.

5.5.9.2. Agentes de Consenso por Interseção

Os algoritmos de consenso por interseção constroem uma solução factível a partir das características em comum de duas ou mais soluções.

As arestas coincidentes em todas as soluções iniciais são reunidas para formarem uma nova solução. Desta forma, se distintas soluções forem construídas por diferentes algoritmos, ou sob diferentes condições iniciais, então as chances de que as partes em comum de tais trajetórias sejam características de uma boa trajetória são grandes.

Na verdade, tais algoritmos podem ser compostos por dois tipos distintos de algoritmos: os *deconstrutores* e os algoritmos de construção. Como os algoritmos de construção já foram abordados, resta serem apresentados os algoritmos de deconstrução.

Algoritmos de Deconstrução

Estes algoritmos podem ser divididos em duas partes:

1.Extração de características comuns de duas ou mais trajetórias;

2.Geração de um ciclo parcial.

A primeira parte consiste na extração de características interessantes (boas) das trajetórias iniciais. A segunda parte consiste em, a partir das boas características obtidas no passo 1 que geralmente envolve um subconjunto do conjunto de cidades do problema, construir um ciclo parcial.

Foram implementados três *deconstrutores*: Deconstructor1 (DEC), Deconstructor2 (DEC-NN) e Deconstructor3 (DEC-NNC).

Deconstructor 1 -DEC

Este algoritmo de deconstrução foi desenvolvido por Souza [Sou93] e pode ser descrito da seguinte maneira:

Na primeira parte são extraídas as *arestas em comum* de duas ou mais trajetórias para a construção de uma trajetória parcial. Na segunda parte, esta trajetória parcial é construída seguindo a seqüência de arestas de uma das trajetórias iniciais. Para fechar o ciclo parcial, geralmente é necessário pelo menos uma aresta extra, que liga a cidade inicial à ultima cidade do ciclo parcial. Então:

Algoritmo Deconstructor 1 - DEC

Entrada: T_1 (primeira trajetória inicial), T_2 (segunda trajetória inicial).
Saída: trajetória parcial TP.

1. **Seja** T_1 e T_2 duas trajetórias iniciais e sejam E^{T1} e E^{T2} seus respectivos conjuntos de arestas.
2. **Faça** $E^{TP} = E^{T1} \cap E^{T2}$;
3. **Inicie** TP inserindo um vértice $i \in V$.
4. **Escolha** T_1 ou T_2 para servir de modelo para a trajetória parcial TP em construção.
5. **Repita**
   5.1. **Selecione** um vértice $j \in V \setminus V^{TP}$ e insira-o em TP segundo a ordem das arestas contidas na trajetória modelo (passo 4);
   até $V^{TP} = V$.
6. **Insira** arestas extras, se necessário, para fechar o ciclo parcial.
7. **Retorne** TP.

Deconstructor 2 - DEC-NN (Inserção do mais próximo)

O algoritmo *Deconstructor-Nearest Neighbor* (DEC-NN) foi desenvolvido por Rodrigues em [RS94] e recebeu esta denominação devido ao procedimento usado em cada um de seus dois passos. A primeira parte é exatamente igual ao algoritmo anterior, onde arestas em comum são extraídas das duas ou mais trajetórias iniciais. Na segunda parte, entretanto, ao invés de se tomar como modelo uma das trajetórias iniciais para a construção do ciclo parcial, são conectadas as cidades mais próximas uma das outras, como será mostrado a seguir.

A idéia é fazer com que o ciclo parcial tenha o menor comprimento possível e não que tenha a forma parecida com uma das trajetórias de entrada. Pelo contrário, o formato do ciclo parcial gerado pode ser completamente diferente do formato das trajetórias iniciais.

Algoritmo Deconstructor 2 (DEC-NN)

Entrada: T_1 (primeira trajetória inicial), T_2 (segunda trajetória inicial), c (matriz de distância do problema).
Saída: trajetória parcial TP.

1. **Seja** T_1 e T_2 duas trajetórias iniciais e sejam E^{T1} e E^{T2} os conjuntos de arestas relativos às trajetórias iniciais.
2. **Faça** $E^{TP} = E^{T1} \cap E^{T2}$;
3. **Construa** uma estrutura com as arestas de E^{TP}, ordenando-as em ordem crescente segundo os valores fornecidos pela matriz de distância;
4. **Para** cada *cidade-extremo i* faça:
   4.1. **Calcule** a distância c_{ij} desta cidade-extremo i a todas as outras cidades-extremos j;
   4.2. **Repita**
      4.2.1. **Selecione** a aresta de menor distância possível;
      até que a aresta selecionada satisfaça as condições necessárias (cada cidade pode ter somente dois vizinhos).
   4.3. **Conecte** a cidade-extremo i com a cidade-extremo j de menor distância a i e insira em TP.
   4.4. **Atualize** a estrutura de arestas, eliminando a aresta inserida em TP.
5. **Retorne** TP.

Deconstructor 3 - DEC-NNC (Inserção do mais próximo de compromisso)

O algoritmo *Deconstructor - Nearest Neighbor of Compromise* (DEC-NNC) foi desenvolvido durante este trabalho e consiste numa adaptação feita no algoritmo anterior de forma a manipular mais de uma matriz de distância.

Na primeira parte também são extraídas as arestas em comum das trajetórias iniciais. Na segunda parte, são conectadas as cidades mais próximas de compromisso uma das outras, ou seja, são consideradas todas as matrizes de distância envolvidas e é inserida a cada passo, aquela aresta que possuir a menor distância de compromisso (seguindo a idéia da *inserção de compromisso* descrita no capítulo 4).

Algoritmo Deconstructor 3 (DEC-NNC)

Entrada: T_1 (primeira trajetória inicial), T_2 (segunda trajetória inicial), c_1, ..., c_k (matrizes de distância do problema).
Saída: trajetória parcial *TP*.

1. **Seja** T_1 e T_2 duas trajetórias iniciais e sejam E^{T_1} e E^{T_2} os conjuntos de arestas relativos às trajetórias iniciais.
2. **Faça** $E^{TP} = E^{T_1} \cap E^{T_2}$;
3. **Construa** uma estrutura com as arestas de E^{TP} para cada matriz de distância do problema (como por exemplo, utilizando a técnica *union find* [Cor91]), ordenando-as em ordem crescente segundo o valor fornecido pela respectiva matriz;
4. **Para** cada *cidade-extremo i* faça:
   - **4.1. Calcule** a distância desta cidade-extremo a todos as outras cidades-extremos;
   - **4.2. Repita**
     - **4.2.1. Selecione** a aresta de menor distância de compromisso possível e conecte a cidade-extremo *i* com a cidade-extremo de menor distância a *i*.

     até que a aresta selecionada satisfaça as condições (cada cidade pode ter somente dois vizinhos).
   - **4.3. Conecte** a cidade-extremo *i* com a cidade-extremo de menor distância de compromisso a *i* e insira em *TP*.
   - **4.4. Atualize** a estrutura de arestas, eliminado a aresta inserida em *TP*.
5. **Retorne** *TP*.

5.5.10. Sugestões de Implementação

Um A-Team tem a forte característica de possuir elementos totalmente independentes uns dos outros, estrutural e funcionalmente. Tal característica possibilita uma implementação modular de maneira direta. Cada agente é implementado independentemente dos demais, porém, com a mesma interface de todos os outros, de tal forma que para o grupo não importa como tal agente gera soluções, e sim, em qual(is) memória(s) ele irá depositar soluções e em qual(is) ele irá retirar (a estrutura interna do algoritmo deve ser levada em conta na definição dos fluxos de dados). As memórias também devem ser implementadas cada uma num módulo separado. Geralmente, como já abordado, a memória principal é implementada com uma estrutura que possibilite uma ordenação das soluções segundo algum critério de qualidade e todas as outras memórias, como simples *buffers* de soluções.

Cada posição da memória deve ser visitada somente por um agente de cada vez. É necessário, então, adotar um *mecanismo de semáforo*, onde cada *posição* é considerada

uma *região crítica* da memória [Tan92]. Isto para garantir a consistência dos dados gerados.

A implementação em módulos funcionalmente distintos facilita a identificação de erros e a realização dos testes para a definição da estrutura final do A-Team, onde a retirada e a inserção de um agente no time pode ser amplamente analisada. Além disto, a paralelização torna-se uma tarefa muito mais fácil, pois todos os agentes podem ser executados simultaneamente. Cada agente e memória do time podem ser considerados como tarefas, no processo de paralelização.

5.5.11. Critério de Parada

Para os métodos exatos, o critério de parada é um só: a solução ótima. Isto quer dizer que um método exato só chega ao fim quando ele fornece como resposta a solução ótima do problema, enquanto isto não for alcançado ele ficará em contínuo processo de execução (alguns podem ser interrompidos e a solução corrente pode ser fornecida como resposta: passa a ser um método aproximado).

Por ser um método aproximado, o critério de parada de um A-Team é um tanto *subjetivo*. Depende da qualidade desejada dos resultados finais e do tempo disponível para a resolução do problema. Geralmente, aplica-se A-Teams com o intuito de se encontrar *boas* soluções, que são soluções próximas do ótimo e até mesmo a solução ótima para o problema (soluções do Pareto Ótimo, para problemas multiobjetivos). E, os critérios de parada utilizados são:

- **Tempo de processamento;**
- **Número de iterações;**
- **Convergência.**

Na fase de projeto de um A-Team, geralmente é estipulado um *tempo de processamento* suficientemente grande para que se possa perceber *quando* obteve-se convergência, ou seja, a partir de que instante de tempo os agentes não mais conseguiram produzir melhores soluções do que as já presentes na memória. Este é um trabalho totalmente empírico e requer várias execuções para ser ajustado. O *número de iterações*, é também utilizado como critério de parada e o procedimento adotado é o mesmo, dando para estimar o tempo de uma iteração pelo tempo de execução do algoritmo mais *lento* do time (de granulosidade mais grossa).

O mais importante nisto tudo é a *convergência dos dados*. O tempo de processamento e o número de iterações devem ser estipulados depois de uma bateria de testes e, mesmo assim, podem variar de instância para instância (mesmo para instâncias de tamanhos iguais). Isto pode ser contornado através da garantia de performance dos algoritmos, de onde se pode saber quão distante do ótimo são as soluções geradas por tal algoritmo e, portanto, pode-se estimar a qualidade das soluções que vão sendo geradas pelo time, bem como através da geração de limites inferiores para o problema.

Se os resultados obtidos não atingirem o nível de qualidade desejado, então, uma reavaliação de todos os parâmetros de projeto deve ser realizada, até que os resultados sejam satisfatórios.

5.6. A Memória de Camadas de Conjuntos Não-Dominados

É na estrutura da memória principal, a Memória de Soluções Completas e Refinadas, que reside a maior diferença entre um A-Team aplicado à Problemas Monobjetivos e um A-Team aplicado à Problemas Multiobjetivos.

O critério de qualidade para ordenação das soluções na memória, no caso de problemas multiobjetivos, é feito com base nos k objetivos do problema. Uma solução k-dimensional tem a propriedade de dominar ou não outras soluções (ver Cap.3). Se ela domina uma outra solução é porque seus valores segundo *todos* os objetivos envolvidos são iguais aos valores da solução dominada e num deles, pelo menos, menor. Agora, se ela não domina a outra solução, então, ou ela é dominada pela outra solução ou as duas soluções são equivalentes entre si (uma possui melhores valores segundo alguns dos objetivos - basta um -, e a outra possui melhores valores segundo os objetivos restantes).

Foi com base neste critério de qualidade inerente às soluções k-dimensionais, que foi desenvolvida a estrutura de organização das soluções na memória em camadas de soluções equivalentes.

5.6.1. Manipulando Conjuntos Não-Dominados

Um conjunto não-dominado é composto por soluções equivalentes entre si, ou seja, nenhuma solução do conjunto domina uma outra solução do mesmo conjunto e vice-versa (soluções que não se dominam). Então:

Definição 5.1: Conjunto Não-Dominado. Dado um conjunto M de soluções k-dimensionais, onde $k=1,2,...,K$, não existe um conjunto de soluções *não-dominadas* definido por $ND(M) = \{s, s \in M \,/\, \nexists\, s_1 \in M \,/\, s_1 \succ s\}$.

Definição 5.2: Dominância entre Conjuntos Não-Dominados. Dados dois conjuntos não-dominados $ND_1=\{s_{11}, s_{12}, ..., s_{1h}, ..., s_{1p}\}$ e $ND_2=\{s_{21}, s_{22}, ..., s_{2l}, ..., s_{2q}\}$, $1 \le h \le p$ e $1 \le l \le q$, onde $p,q \le m$, $p \ne q$, e seja $f_i(\bullet)$, $i=1,2,..k$ funções objetivo a serem consideradas, se $f_i(s_{1h}) \le f_i(s_{2l})$ para todo i, h e l, e se existe um j, $j=\{1, ..., k\}$ tal que $f_j(s_{1h}) < f_j(s_{2l})$, então ND_1 domina ND_2 ($ND_1 \succ\succ ND_2$).

Os termos *Conjunto Não-Dominado* e *Conjunto de Soluções Não-Dominadas* serão usados intercaladamente. A relação "$\succ$" é chamada de *dominância* (para conjuntos, definiu-se "$\succ\succ$").

Desta forma, a memória é organizada em camadas de conjuntos não-dominados, onde o melhor conjunto ocupa a primeira posição da memória (conjunto que domina todos os outros e não é dominado por nenhum outro da memória). E assim, o conjunto de soluções não-dominadas da camada da posição p, $1 \le p \le c$, da memória domina o conjunto de soluções não-dominadas da camada da posição $p+1$, até a última posição c, ocupada pelo pior conjunto de soluções não-dominadas (conjunto que não domina nenhum outro e é dominado por todos).

Tal estrutura é dinâmica, variando em tempo de execução, onde novas soluções são inseridas ou retiradas a todo instante e, conseqüentemente, novas camadas são construídas ou desfeitas.

A capacidade máxima da memória, então, é de m soluções dispostas em c camadas (posições da memória).

O número máximo de camadas possíveis é $c=m$, onde cada uma das soluções contidas na memória domina a solução subseqüente, que é o melhor caso, já que na inserção de uma nova solução, a comparação é feita até se encontrar a primeira solução que domina ou é dominada pela solução inserida, que pode estar na primeira posição da memória. Este caso recai na estrutura seqüencial utilizada para problemas monobjetivos.

O número máximo de soluções por camada é $s=m$, caso onde todas as soluções pertencem a uma só camada, que é o *pior caso*, já que na inserção de uma nova solução terá sempre que compará-la com todas as m soluções contidas na memória.

Preparata e Shamos [PS85] afirmam que dado um conjunto fixo de soluções em *duas* dimensões (k=2), encontrar o melhor conjunto não-dominado leva O($mlogm$), onde m é o número total de elementos (número de soluções na memória do A-Team). Tal problema foi chamado por eles de ***Maxima Problem*** e a prova da complexidade foi obtida através de sua redução ao problema de ordenação. Generalizando, para $k\geq 2$ [PS85]:

O máximo de um conjunto de m pontos no espaço E^k, $k\geq 2$, pode ser obtido em tempo $O(m(\log m)^{k-2}) + O(m\log m)$.

Portanto, dado um conjunto dinâmico de soluções, dividí-lo em camadas de soluções não-dominadas a cada nova inserção ou a cada nova eliminação de uma solução, não é possível de ser feito em tempo linear. *Não existe algoritmo em tempo linear para a inserção ou eliminação de pontos num espaço k-dimensional, com $k\geq 2$* [PS85].

Dado este contexto, foram desenvolvidos dois algoritmos para a manipulação de conjuntos de soluções k-dimensionais: um para a inserção e outro para eliminação de uma solução nestes conjuntos não-dominados.

5.6.1.1. Algoritmo para Inserção de uma Solução k-dimensional na Memória

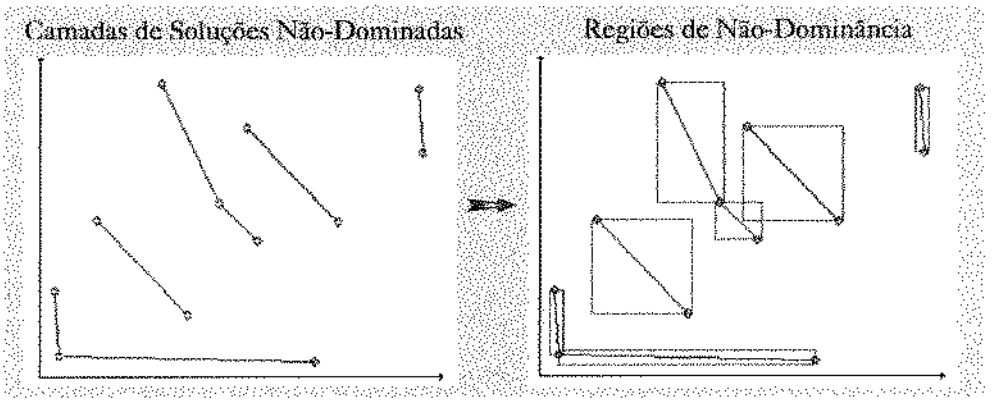
O efeito da inserção de uma nova solução numa determinada camada não-dominada pode se propagar por muitas outras camadas, requerendo o desenvolvimento de um algoritmo que *trate* todas as possibilidades possíveis desta propagação.

Foi pensando nas possibilidades de propagação de alterações nas camadas de soluções não-dominadas que se descobriu uma propriedade existente entre cada duas soluções eficientes entre si: a *região de não-dominância* (fig.5.9).

Definição: Região de Não-Dominância. Seja $s_e \in R^k$ e $s_f \in R^k$ duas soluções equivalentes entre si, $f_i^{min} = min\{f_i(s_e), f_i(s_f)\}$, $f_i^{max} = max\{f_i(s_e), f_i(s_f)\}$, então,

$r_{nd} = \{s \in R^k, f_i^{min} \leq f_i(s) \leq f_i^{max}\}$, $\forall i$.

Figura 5.9: Regiões de Não-Dominância. Definindo as regiões de não-dominância num *meta*-conjunto de conjuntos de soluções não-dominadas.



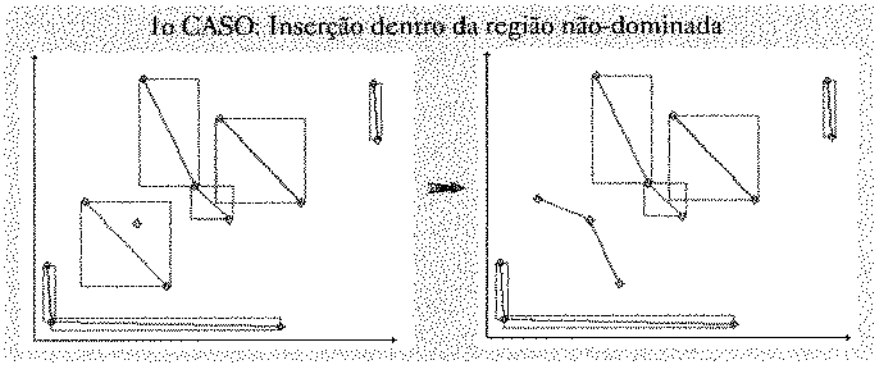
Pode ser observado, que basta definirmos a região de dominância de cada solução (ver Cap.3), tanto para maximização quanto para minimização e a região de não dominância assume uma forma cúbica (fig.5.9).

A região de não-dominância possibilitou a detecção de três possíveis casos que podem ocorrer quando uma nova solução for inserida na memória:

Primeiro Caso: A nova solução é inserida dentro da região de não-dominância.

A nova solução inserida no meta-conjunto de conjuntos de soluções não-dominadas *está contida* numa região de não-dominância definida por duas soluções (fig.5.10). Neste caso, a solução não é dominada pela camada na qual as duas soluções pertencem e nem domina nenhuma solução desta camada. Como entre as soluções desta camada e a nova solução ninguém domina ninguém, elas são equivalentes entre si. A nova solução é, portanto, inserida nesta camada.

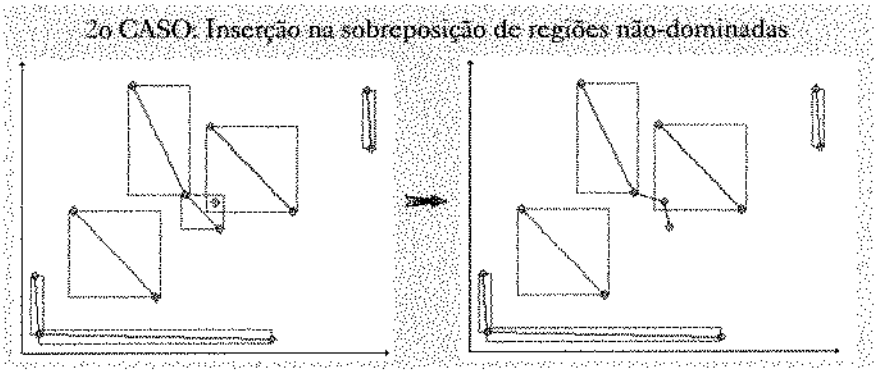
Figura 5.10: Primeiro Caso: A nova solução é inserida *dentro* da região de não-dominância. No primeiro quadro é mostrado a configuração das camadas *antes* da propagação de alterações e a nova solução *dentro* de uma região de não-dominância. No segundo quadro é mostrado a configuração das camadas *depois* da propagação de alterações entre as camadas de soluções não-dominadas.



Segundo Caso: A nova solução é inserida *dentro da sobreposição* de regiões de não-dominância.

A nova solução inserida no meta-conjunto de conjuntos de soluções não-dominadas *está contida* dentro de uma *sobreposição de regiões* de não-dominância formada por soluções de diferentes camadas (fig.5.11). Neste caso, a solução não é dominada por nenhuma solução das camadas nas quais as soluções que compõe a sobreposição de regiões de não-dominância pertencem e não domina nenhuma delas também. Então, a nova solução poderá ser incluída em qualquer uma das camadas envolvidas. Foi adotada a inserção na *melhor* das camadas envolvidas (a camada que domina todas as outras envolvidas).

Figura 5.11: Segundo Caso: A nova solução é inserida *dentro da sobreposição* de regiões de não-dominância. No primeiro quadro é mostrado a configuração das camadas *antes* da propagação de alterações e a nova solução *dentro da sobreposição* de regiões de não-dominância. No segundo quadro é mostrado a configuração das camadas *depois* da propagação de alterações entre as camadas de soluções não-dominadas.

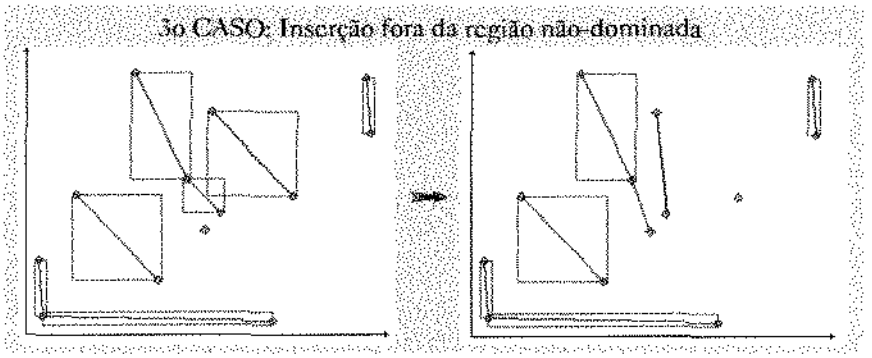


Terceiro Caso: A nova solução é inserida *fora* da região de não-dominância.

A nova solução inserida no meta-conjunto de conjuntos de soluções não-dominadas *não está contida* em nenhuma região de não-dominância definida por duas soluções (fig.5.12).

Neste caso, não se pode afirmar onde a solução será inserida, dado que não se tem nenhuma região de não-dominância como referência da camada que ela deve pertencer. Desta forma, deve-se verificar desde o início da memória em qual camada a nova solução deve ser inserida.

Figura 5.12: Terceiro Caso: A nova solução é inserida *fora* da região de não-dominância. No primeiro quadro é mostrado a configuração das camadas *antes* da propagação de alterações e a nova solução *fora* de uma região de não-dominância. No segundo quadro é mostrado a configuração das camadas *depois* da propagação de alterações entre as camadas de soluções não-dominadas.



Dado que o terceiro caso inviabiliza os dois primeiros, o que só confirma a teoria que diz não existir nenhum algoritmo em tempo linear para a *manipulação* - inserção e deleção - de pontos num espaço *k*-dimensional ($k \geq 2$), foi elaborado um algoritmo de complexidade $O(m^2)$, onde m é o número de soluções na memória. O algoritmo de inserção é mostrado a seguir

O Algoritmo de Inserção

Neste algoritmo, verifica-se desde a primeira camada se a nova solução domina ou é dominada por alguma solução. Quando ela for dominada por alguma solução de determinada camada, pula-se imediatamente para a próxima camada da memória e caso ela domine alguma solução, ela é inserida na camada corrente e as soluções daquela camada que a nova solução domina são *transferidas* para as devidas camadas subseqüentes.

O algoritmo é mostrado numa linguagem *pseudo-c*, onde as variáveis, estruturas e procedimentos adicionais são os seguintes:

Variáveis

Sol → solução que está sendo inserida na memória;
Sol_da_Camada → solução da camada corrente da memória que está sendo comparada com a nova solução;
Sol_Dominada → variável que indica se a solução foi dominada pela solução corrente da camada (Sol_da_Camada);
c → índice que indica qual a camada corrente.

Estruturas

Lst_Sol_Deslocadas → Lista de soluções que foram retiradas de uma determinada camada e devem ser recolocadas em alguma camada posterior;
Lst_Sol_da_Camada → Lista de soluções da camada corrente (camada *candidata* a conter a nova solução).

Rotinas Adicionais

RetiraPrimeiraDaLstSolDeslocadas → Retira a primeira solução da lista de soluções deslocadas (soluções que foram retiradas de uma camada e devem ser inseridas nas posteriores);

RetiraPrimeiraDaLstSolDaCamada → Retira a primeira solução da lista de soluções da camada corrente da memória;

RetiraCamadaDeSol → Seleciona todas as soluções da camada candidata a receber a nova solução;

RetiraSolDaCamada → Retira uma solução da camada candidata *dominada* pela nova solução a ser inserida;

InsereNaLstSolDeslocadas → Insere a solução da camada candidata, dominada pela nova solução, na lista de soluções deslocadas;

InsereSolNaMemória → Insere a nova solução na camada corrente da memória.

```
InserirSoluçãoNaMemória (Sol)

Lst_Sol_Deslocadas ← Sol;
Repetir
   Sol_Dominada ← FALSO;
   Sol ← RetiraPrimeiraDaLstSolDeslocadas (Lst_Sol_Deslocadas);
   c ← Sol.camada + 1;
   Lst_Sol_Camada ← RetiraCamadaDeSol (c);
   Enquanto (Lst_Sol_Camada≠0) e (Sol_Dominada==FALSO) )
        Sol_da_Camada ← RetiraPrimeiraDaLstSolDaCamada (Lst_Sol_Camada);
        Se Sol domina Sol_da_Camada
             RetiraSolDaCamada (Sol_da_Camada);
             InsereNaLstSolDeslocadas (Sol_da_Camada);
        Senão
             Se Sol_da_Camada domina Sol
                  Sol.camada ← c;
                  InsereNaLstSolDeslocadas (Sol);
                  Sol_Dominada ← VERDADEIRO;
   Se (Sol_Dominada == FALSO)
        InsereSolNaMemória (c, Sol);
Até (Lst_Sol_Deslocadas = 0);

FIM
```

5.6.1.2. Algoritmo para Eliminação de uma Solução k-dimensional na Memória

O algoritmo para a eliminação de uma solução *k*-dimensional da memória do A-Teams organizada em camadas de soluções não-dominadas é similar ao processo de inserção, entretanto, é menos intrincado. Não existe casos especiais no caso da eliminação, pois se sabe de qual camada da memória será retirada uma solução, e não existe diferenciação entre as soluções de uma mesma camada.

Vale ressaltar que, o efeito causado pela retirada de uma solução da memória se propaga a partir da camada seguinte à camada na qual a solução eliminada estava contida e, obviamente, a camada da solução retirada *é* conhecida. Diferentemente do caso da inserção, onde primeiro tem que ser descoberto em *qual* camada a nova solução deve ser inserida e este fato força uma busca desde a primeira camada da memória.

As estruturas que compõem o algoritmo de eliminação de uma solução da memória são as seguintes:

Variáveis

Sol → solução que está sendo eliminada na memória;

Sol_da_Camada → solução da camada corrente da memória que está sendo comparada com a nova solução;

Sol_Dominada → variável que indica se a solução foi dominada pela solução corrente da camada (Sol_da_Camada);

c → índice que indica qual a camada corrente.

Estruturas

Lst_Sol_Deslocadas → Lista de soluções que foram retiradas de uma determinada camada e devem ser recolocadas em alguma camada posterior;

Lst_Sol_da_Camada → Lista de soluções da camada corrente (camada *candidata* a conter a nova solução).

Rotinas Adicionais

RetiraPrimeiraDaLstSolDeslocadas → Retira a primeira solução da lista de soluções deslocadas (soluções que foram retiradas de uma camada e devem ser inseridas nas posteriores);

RetiraPrimeiraDaLstSolDaCamada → Retira a primeira solução da lista de soluções da camada corrente da memória;

RetiraCamadaDeSol → Seleciona todas as soluções da camada candidata a receber a nova solução;

RetiraSolDaCamada → Retira uma solução da camada candidata *dominada* pela nova solução a ser inserida;

AtualizaNumCamadas → Reorganiza numeração das camadas posteriores à camada da solução a ser eliminada;

InsereNaLstSolDeslocadas → Insere a solução da camada candidata, dominada pela nova solução, na lista de soluções deslocadas;

InsereSolNaCamada → Insere a solução corrente na camada indicada;

EliminaSolDaMemória → Elimina a solução selecionada da memória.

```
EliminarSoluçãoNaMemória (Sol)

c ← Sol.camada + 1;
EliminaSolDaMemória (Sol);
Lst_Sol_Deslocadas ← RetiraCamadaDeSol (c);
AtualizaNumCamadas (c);
Repetir
  Sol_Dominada ← FALSO;
  Sol ← RetiraPrimeiraDaLstSolDeslocadas (Lst_Sol_Deslocadas);
  c ← Sol.camada+1;
  Lst_Sol_Camada ← RetiraCamadaDeSol (c);
  Enquanto ( (Lst_Sol_Camada≠0) e (Sol_Dominada==FALSO) )
      Sol_da_Camada ← RetiraPrimeiraDaLstSolCamada (Lst_Sol_Camada);
      Se Sol domina Sol_da_Camada
          InsereNaLstSolDeslocadas (Sol_da_Camada);
          RetiraSolDaCamada (Sol_da_Camada);
      Senão
          Se Sol_da_Camada domina Sol
              Sol.camada ← c;
              InsereNaLstSolDeslocadas (Sol);
              Sol_Dominada ← VERDADEIRO;
  Se (Sol_Dominada == FALSO)
      InsereSolNaCamada(c, Sol);
Até (Lst_Sol_Deslocadas == 0);

FIM
```

5.7. Políticas de Destruição

Considerando que o tamanho da memória é limitado, torna-se necessário a definição de alguns critérios para garantir que este tamanho permaneça constante, dado que novas soluções são inseridas na memória a todo momento.

Esta tarefa cabe aos agentes destruidores, que contém em seus procedimentos internos determinada política de distribuição de probabilidade. Eles decidem se uma nova solução gerada pode ou não ser inserida na memória e, em caso positivo, aplicam a distribuição de probabilidade para escolher *qual* solução deve ser eliminada.

Para as memórias que funcionam como *buffers* a eliminação de soluções é automática, ou seja, quando um agente seleciona uma solução ela imediatamente é retirada da memória. Sendo assim, é desnecessária a atuação dos agentes Destruidores sobre estas memórias (fig.5.4).

No caso da *Memória de Soluções Completas e Refinadas*, a atuação dos agentes Destruidores é imprescindível, já que as soluções são organizadas na memória segundo algum critério de qualidade e, portanto, faz sentido priorizar a eliminação de certas soluções em prol da manutenção de diversidade e obtenção de convergência do sistema. Desta forma, tão importante quanto gerar boas soluções é saber quais delas favorecer e quais eliminar da memória.

A seguir, serão apresentadas algumas políticas de destruição, segundo o tipo de organização de memória: se organizadas tradicionalmente (m posições seqüenciais e segundo apenas um critério de qualidade), manipulando-se soluções unidimensionais; e, se organizadas em camadas de soluções não-dominadas (c posições seqüenciais, onde cada posição contém s sub-posições), manipulando-se soluções multidimensionais:

Definições Preliminares para Memória de Soluções Unidimensionais

Neste caso, as soluções são organizadas na memória em ordem seqüencial (no total de m posições), da melhor para a pior segundo o valor da *única* função objetivo do problema. Tal organização de memória é aplicada para problemas monobjetivos ou multiobjetivos *alterados* (todas as funções objetivo do problema são *transformadas* numa só). A seguir, são definidos alguns termos necessários para um melhor entendimento das políticas de destruição adotadas para a memória de soluções unidimensionais.

Definições:

m → número máximo de soluções na memória (capacidade máxima da memória);

s → solução s;

$P_s^{(t)}$ → probabilidade de escolha da solução s, no instante t;

T → tempo máximo de processamento (*CPU time*).

$$\sum_{s=1}^{m} P_s^{(t)} = 1 \quad , \quad \forall \quad P_s^{(t)} \geq 0$$

onde,

$$t \leq T \qquad t, T \in Z^+$$

Definições Preliminares para a Memória de Soluções Multidimensionais

Neste caso, as soluções são organizadas na memória em camadas crescente de dominância, ou seja, *camadas de conjuntos não-dominados*, em ordem seqüencial de camadas (no total de c posições), da melhor camada (que domina todas as outras camadas) para a pior camada (que não domina ninguém e é dominada por todas). Em cada camada, as soluções são organizadas seqüencialmente (no total de s sub-posições) pela ordem de inserção na camada. Tal organização de memória é aplicada para problemas multiobjetivos, onde se deseja otimizar todas as funções objetivo simultaneamente.

Neste caso, a seleção de uma solução ocorre em dois níveis:

1º nível: Escolha de uma camada;

2º nível: Escolha de uma solução da camada selecionada no primeiro nível.

Isto faz com que as políticas relacionadas para o caso de um único objetivo sejam reformuladas e uma descrição teórica das *novas* políticas de destruição de soluções se torna necessária, com a definição dos seguintes termos:

Definições:

m → número máximo de soluções na memória (capacidade máxima da memória);

$C_{(t)}$ → número de camadas de soluções não-dominadas na memória, no tempo t;
$C_{(t)} \in \{1,2,...m\}$

$S_{c_{(t)}}$ → número de soluções na camada c, no instante t;
$S_{c_{(t)}} \in \{1,2,...,m\}$

c → camada c;

s → solução s;

T → tempo máximo de processamento (*CPU time*).

$P_c^{(t)}$ → probabilidade de escolha da camada c, no instante t;

$P_{s_c}^{(t)}$ → probabilidade de escolha de uma solução da camada c, no instante t.

onde,

$$t \le T \qquad t,\ T \in Z^+$$

$$\sum_{c=1}^{C_{(t)}} S_{c_{(t)}} P_{s_c}^{(t)} = 1 \quad , \quad \forall \quad P_{s_c}^{(t)} \ge 0$$

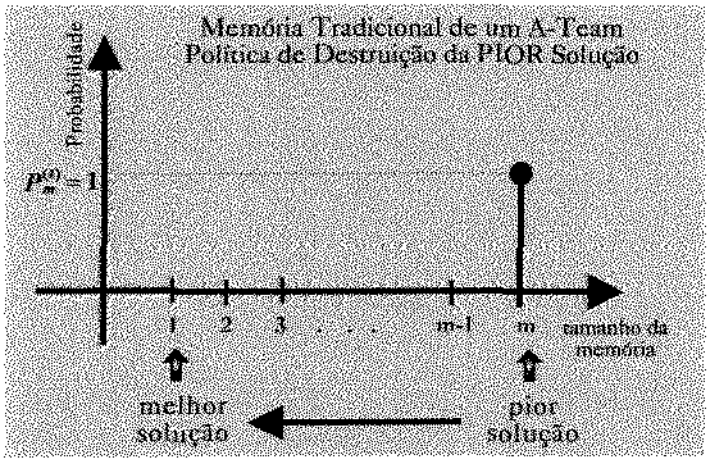
$$S_c = 1 \Leftrightarrow C = m$$
$$S_c = m \Leftrightarrow C = 1$$

Dada as definições necessárias, tanto para o caso da memória de soluções unidimensionais quanto para a memória de soluções multidimensionais, podem ser descritas agora as políticas de destruição de soluções.

5.7.1.Destruição de uma Solução de Pior Qualidade

No caso da **memória de soluções unidimensionais**, a *solução de pior qualidade* presente na memória é *eliminada com probabilidade 1*. A figura 5.13 ilustra tal política, onde a solução contida na *m*-ésima posição da memória é sempre eliminada.

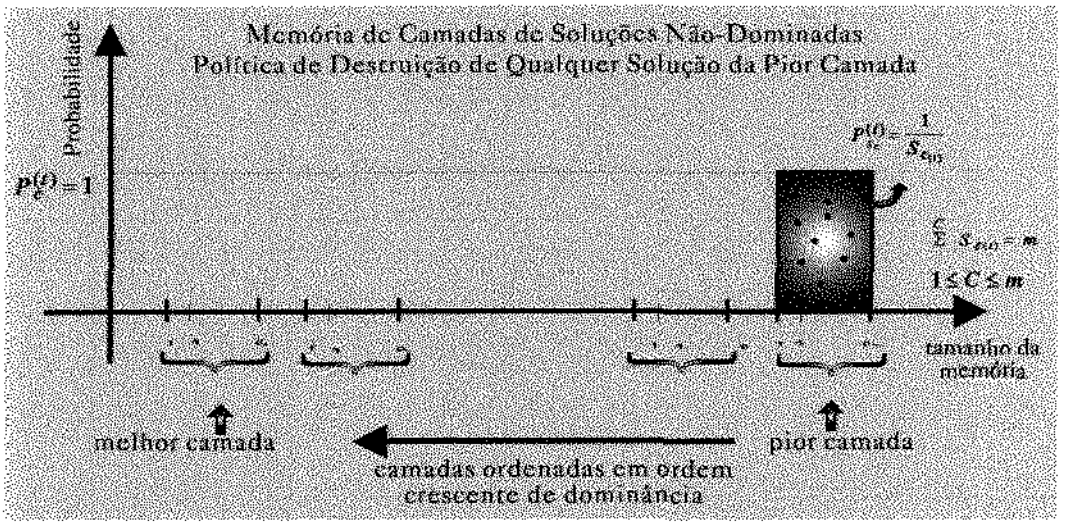
Figura 5.13: Política de Destruição da *Pior Solução*.



No caso da **memória de soluções multidimensionais**, a política de destruição se baseia na eliminação de *qualquer* solução da *pior* camada de soluções não-dominadas existente na memória, no instante *t* (fig. 5.14). A *camada de pior qualidade* presente na memória é *selecionada com probabilidade 1* e, em seguida, uma solução desta camada é escolhida com probabilidade de distribuição uniforme.

A partir do momento em que a memória atingir sua capacidade máxima de soluções (*m*), o uso desta política fará com que uma nova solução gerada seja inserida na memória. Sendo assim, uma solução da pior camada ($P_c^{(t)}=1$, $c=C$) é eliminada com probabilidade $P_{S_c}^{(t)}=\frac{1}{S_{c(t)}}$.

Figura 5.14: Política de Destruição de *Qualquer Solução* da *Pior Camada*. Sempre que uma nova solução é inserida na memória, se ela não pertencer à pior camada existente, uma das soluções da pior camada de soluções não-dominadas é eliminada.



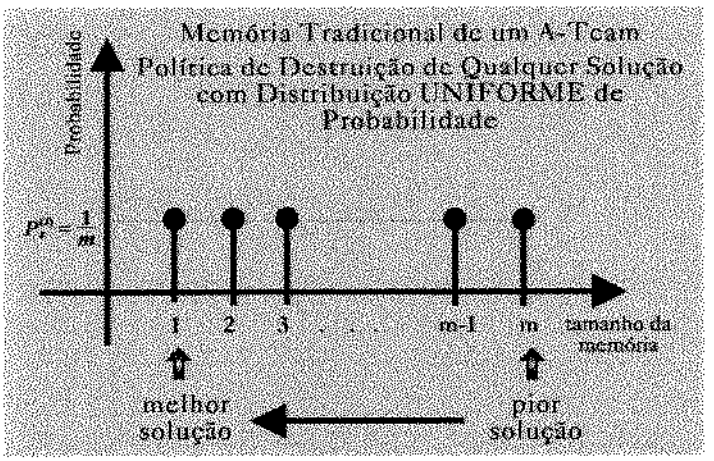
O uso da política de destruição de uma solução de pior qualidade, para os dois tipos de organização de soluções, induz a uma *rápida convergência*, pois a diversidade de soluções da memória é rapidamente reduzida. Isto acontece porque sempre é eliminada uma solução de pior qualidade (seja a pior solução unidimensional ou uma das soluções multidimensionais da pior camada) e, ao longo do tempo, com a ação dos agentes que *refinam* as soluções da memória (agentes de melhoria, por exemplo) e a ação dos agentes que *extraem características em comum* de *boas* soluções, a qualidade média das soluções aumenta. O problema é que como a convergência é rapidamente alcançada, as soluções da memória ficam em torno de um mínimo local, e os agentes ficam impossibilitados, por falta de diversidade, de explorarem outras regiões e alcançarem a região do ótimo global.

5.7.2.Destruição com Distribuição Uniforme de Probabilidade

Com o uso desta política, para os dois tipos de organização de soluções, *todas as soluções da memória possuem a mesma probabilidade de serem eliminadas* (com exceção da melhor solução no caso da memória de soluções unidimensionais e das soluções da melhor camada no caso da memória de soluções multidimensionais).

As soluções de melhor qualidade (melhor solução unidimensional da memória ou soluções multidimensionais da melhor camada da memória) não são eliminadas porque, desta forma, se evita o risco de eliminar as melhores soluções encontradas ao longo da execução do A-Team e que poderiam não ser encontradas novamente. A figura 5.15 ilustra esta política para o caso da memória de soluções unidimensionais.

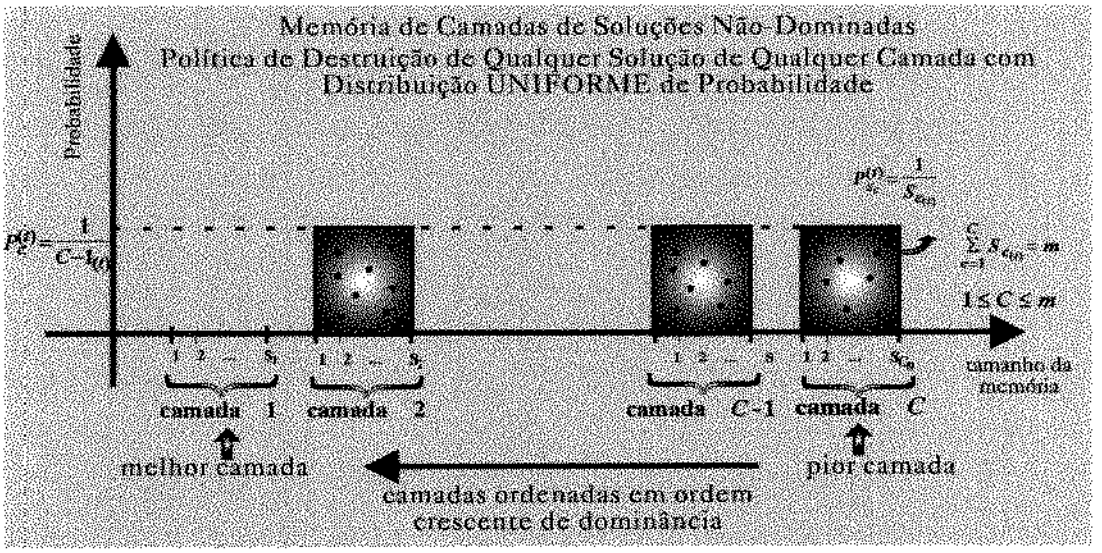
Figura 5.15: Política de Destruição de *Qualquer Solução com Distribuição Uniforme de Probabilidade.*



No caso da memória de soluções multidimensionais, esta política de destruição se baseia na eliminação de *qualquer solução* de *qualquer camada* de soluções não-dominadas (*exceto a melhor camada*) existente na memória, com distribuição *uniforme* de probabilidade, no instante *t* (fig. 5.16). Qualquer solução de qualquer camada (com

exceção da melhor) é eliminada com probabilidade de distribuição uniforme, ($P_{s_c}^{(t)} = \frac{1}{S_{c_{(t)}}}$) e ($P_c^{(t)} = \frac{1}{C_{(t)} - 1}$).

Figura 5.16: Política de Destruição de *Qualquer Solução* de *Qualquer Camada* com *Distribuição Uniforme de Probabilidade*. Sempre que uma nova solução é inserida na memória, se ela não pertencer à pior camada existente, uma das soluções de qualquer camada (exceto a melhor) é eliminada com distribuição uniforme de probabilidade.

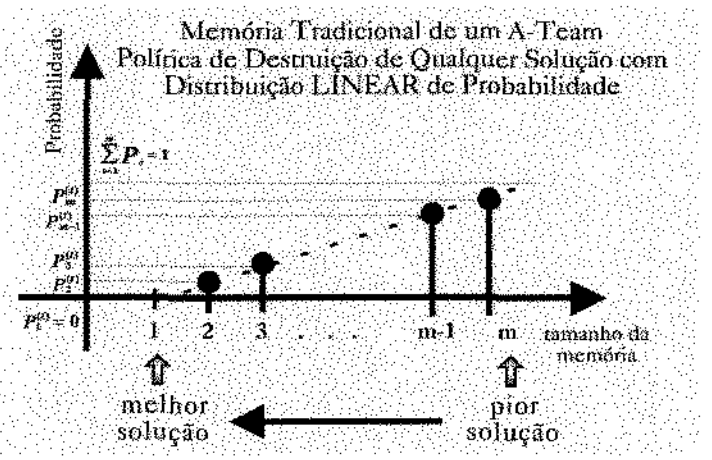


O uso desta política torna o processo de convergência mais lento, pois é garantida a diversidade de soluções na memória por mais tempo. Isto acontece porque qualquer solução, boa ou ruim, tem a mesma chance de ser eliminada e, portanto, as soluções ruins desaparecerem muito lentamente, com a geração cada vez maior de soluções melhores. O problema é que muitas soluções de alta qualidade podem ser eliminadas antes mesmo de serem utilizadas por outros agentes e, desta forma, boas regiões são desfavorecidas prejudicando a busca por melhores soluções.

5.7.3.Destruição com Distribuição Linear de Probabilidade

Para o caso da **memória de soluções unidimensionais**, esta política atribui uma probabilidade 0 à melhor solução contida na memória (nunca se elimina a melhor) e atribui uma probabilidade linearmente crescente ao restante das soluções (da segunda melhor solução até a pior). A figura 5.17 exemplifica a atuação desta política para a memória de soluções unidimensionais.

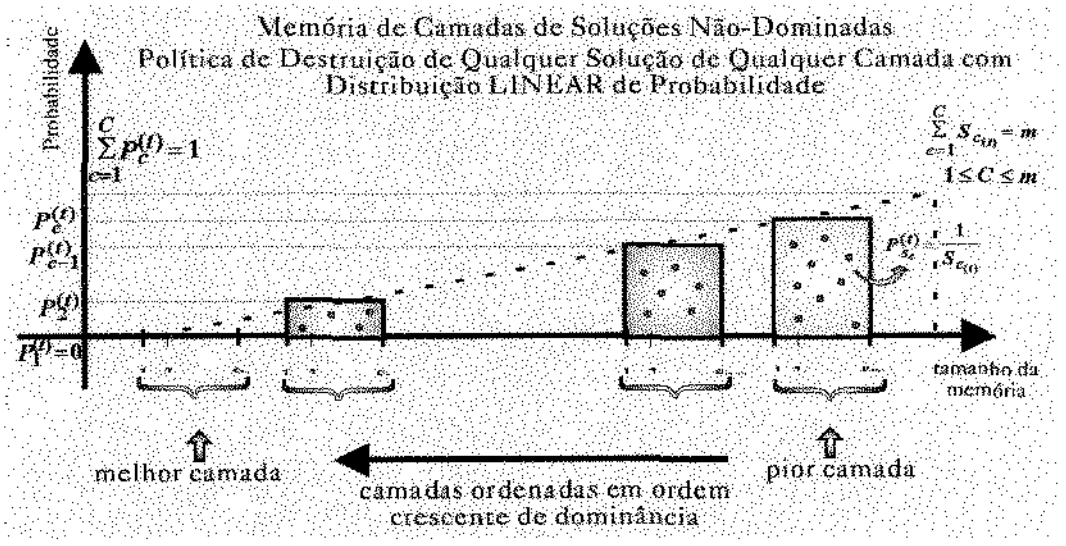
Figura 5.17: Política de Destruição de *Qualquer Solução com Distribuição Linear de Probabilidade*.



No caso da **memória de soluções multidimensionais**, esta política de destruição se baseia na eliminação de *qualquer solução* de *qualquer camada* de soluções não-dominadas (*exceto a melhor camada*) existente na memória, com distribuição *linear* de probabilidade, no instante *t* (fig. 5.18).

Se uma nova solução a ser inserida pertencer a pior camada de soluções não-dominadas ela não é inserida, caso contrário ela é inserida e qualquer solução de qualquer camada (com exceção da melhor) é selecionada com probabilidade de distribuição linear.

Figura 5.18: Política de Destruição de *Qualquer Solução* de *Qualquer Camada* com *Distribuição Linear de Probabilidade*. Sempre que uma nova solução é inserida na memória, se ela não pertencer à pior camada existente, uma das soluções de qualquer camada (exceto a melhor) é eliminada com distribuição linear de probabilidade.



Esta política deixa a memória com uma diversidade mais alta do que a *política de destruição do pior* e possibilita uma convergência mais rápida do que a *política de destruição com distribuição uniforme de probabilidade*. Isto acontece porque como

entre as melhores soluções unidimensionais (ou melhores camadas de soluções multidimensionais) existem muitas soluções similares e, com a baixa probabilidade delas serem eliminadas, o que acontece é que muitas soluções parecidas são eliminadas (aumentando a diversidade entre as boas soluções), por outro lado, muitas outras boas soluções são mantidas na memória (garantindo convergência), já que a probabilidade delas serem eliminadas é pequena.

O problema advindo do uso desta política de destruição é que, depois de um certo tempo de processamento, as soluções presentes na memória se tornam muito parecidas entre si, provavelmente sendo de uma mesma região. Como esta região pode não ser a região do ótimo global e devido a semelhança entre elas os agentes de consenso ficam impossibilitados de gerarem soluções diferentes. Caso elas forem de uma mesma região ou de regiões de mínimos locais, os agentes de melhoria se tornam incapazes de escapar do mínimo local e acaba se obtendo uma convergência não desejada, onde as soluções da memória não são as soluções de melhor compromisso que se desejava encontrar.

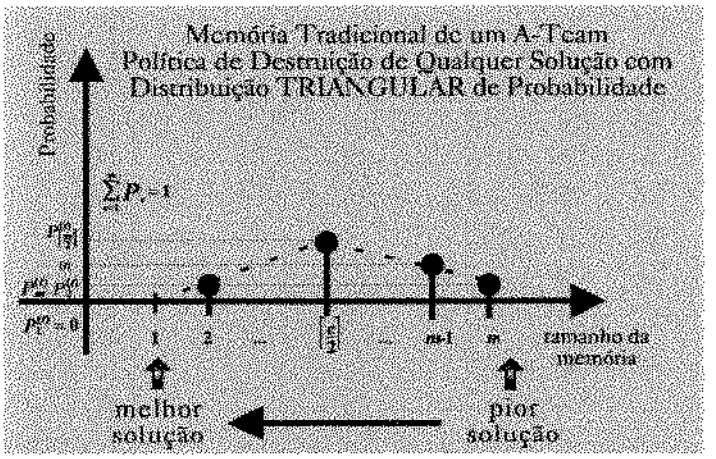
5.7.4.Destruição com Distribuição Triangular de Probabilidade

Esta política foi desenvolvida depois de um estudo sobre o comportamento dos algoritmos para o MDTSP usados nos A-Teams projetados. Seu desempenho foi comparado com os da demais políticas de destruição e os resultados obtidos foram melhores para a maioria dos testes realizados.

Neste caso, a distribuição de probabilidade de eliminação das soluções assume uma forma triangular, onde os extremos da memória (melhores e piores soluções) têm uma probabilidade menor de serem eliminados.

A fig. 5.19 exemplifica esta política para o caso **da memória de soluções unidimensionais**. Pode ser notado que a melhor solução possui probabilidade 0 de ser eliminada.

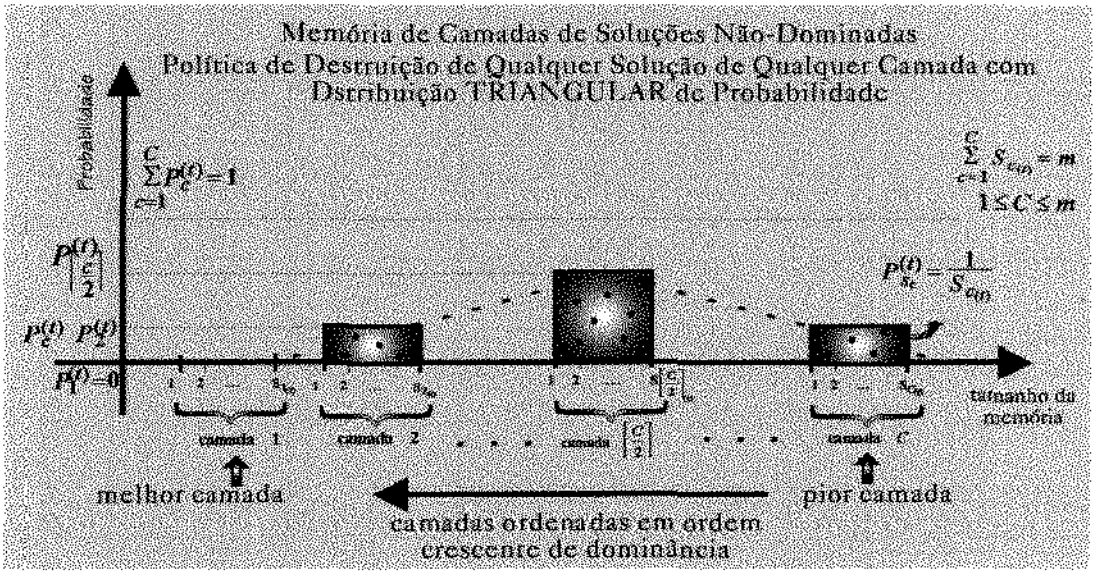
Figura 5.19: Política de Destruição de *Qualquer Solução com Distribuição Triangular de Probabilidade.*



No caso da **memória de soluções multidimensionais**, esta política de destruição é baseada na eliminação de *qualquer solução* de *qualquer camada* de soluções não-dominadas (*exceto a melhor camada*) existente na memória, com distribuição *triangular* de probabilidade, no instante *t* (fig. 5.20).

Se uma nova solução a ser inserida não pertencer a pior camada de soluções não-dominadas (e neste caso, ela não é inserida), então qualquer solução e qualquer camada (com exceção da melhor) é selecionada com probabilidade de distribuição triangular.

Figura 5.20: Política de Destruição de *Qualquer Solução* de *Qualquer Camada* com *Distribuição Triangular de Probabilidade*. Sempre que uma nova solução é inserida na memória, se ela não pertencer à pior camada existente, uma das soluções de qualquer camada (exceto a melhor) é eliminada com distribuição *triangular* de probabilidade.



O uso desta política traz um certo equilíbrio entre a convergência e a diversidade. Esta política surgiu devido a uma análise do comportamento das três políticas anteriores e se percebeu que mesmo no caso da melhor delas (a política de destruição de soluções com distribuição linear de probabilidade), que prioriza as melhores soluções, a convergência era obtida muito rapidamente e a tendência era a memória ficar com baixa diversidade.

Com a política triangular o que acontece é que tanto as soluções muito boas quanto as soluções muito ruins têm mais chances de permanecer na memória e as soluções intermediárias, nem muito boas e nem muito ruins, tendem a serem eliminadas.

Percebeu-se, empiricamente, que depois de um certo tempo, a memória continha soluções muito boas e uma grande quantidade de soluções muito parecidas entre si. Isto dificultava a ação dos agentes, tanto de melhoria quanto de consenso, já que as soluções tendiam a ser de uma mesma região. Quanto às melhores soluções obtidas, nunca devem ser desprezadas (sob pena de não se voltar a encontrá-las), agora, quanto às soluções intermediárias e às piores, qual delas eliminar com maior probabilidade? Quais tem mais chance de serem parecidas com as melhores soluções? Quais deixarão a memória com uma diversidade mais elevada? É natural afirmar que depois de um certo tempo de execução as soluções intermediárias são mais parecidas com as melhores, já que o processo tende a convergir. E, seguindo a mesma linha de raciocínio, as piores tendem a

ser mais diferentes. Desta forma, por que não priorizar a eliminação das soluções intermediárias?

Foi por esta razão que se pensou na política triangular, cujo comportamento é o seguinte:

No início, as soluções contidas na memória possuem baixa qualidade, sendo consideradas ruins, já que são produzidas pelos agentes de construção e/ou randômicos. Nesta fase, o que importa é a priorização das melhores soluções (melhores camadas de soluções multidimensionais) e quanto ao restante da memória, não é muito relevante se a priorização é das piores soluções ou das intermediárias contidas na memória, ou seja, na fase inicial a política linear e a política triangular exercem a mesma influência: a média de qualidade das soluções na memória tendem a aumentar.

5.8. Políticas de Seleção

Para as memórias que funcionam como *buffers* de soluções, a posição da memória a ser escolhida não é relevante, uma vez que não existe uma ordem de qualidade entre as soluções armazenadas neste tipo de memória.

Agora, para as outras (qualquer uma que armazene soluções factíveis), especificamente para a *Memória de Soluções Completas e Refinadas*, é possível o estabelecimento de uma ordem de qualidade entre as soluções e, portanto, *qual* solução o agente deve escolher pode ser significativo para um melhor desempenho do A-Team. E, dependendo do tipo de problema abordado (devido as diferentes possibilidades de organização das soluções na memória), diversas estratégias podem ser aplicadas.

A seguir, são apresentadas algumas estratégias para a seleção de soluções, onde para cada uma delas é explicado o processo de seleção numa memória de soluções unidimensionais e numa memória de soluções multidimensionais.

5.8.1. Priorização da Solução de Melhor Qualidade

Para o caso da memória de soluções unidimensionais, o agente prioriza a escolha da melhor solução (segundo o valor da função objetivo) contida na memória. Para o caso da memória de soluções multidimensionais, o agente prioriza a escolha da melhor camada (camada menos dominada) e seleciona *qualquer* solução desta camada.

Se o agente for determinista (de melhoria, por exemplo), ele inicia a tentativa de seleção no início da memória e percorre-a seqüencialmente até encontrar a primeira solução que não tenha sido gerada ou visitada por ele. Isto porque um agente determinista *sempre* gera a mesma saída para entradas iguais.

Na verdade, esta política é mais apropriada para agentes de melhoria deterministas, pois possibilita a exploração da vizinhança de soluções potencialmente *boas*, garantindo a possibilidade de chegar a soluções melhores ainda. Vale ressaltar, entretanto, que depois de um longo tempo de processamento o uso desta política pode não ser indicado, já que no caso dos algoritmos de melhoria deterministas, se as soluções escolhidas forem *muito* boas, elas terão alta probabilidade de serem mínimos locais, de onde os algoritmos de melhoria *não* conseguem escapar. No geral, esta política não é

indicada para agentes não-deterministas, já que cada execução destes algoritmos, para uma mesma solução inicial, pode levar a geração de soluções completamente diferentes e, portanto, características de uma *boa* solução podem ser desperdiçadas.

O uso desta política tende a *diminuir a diversidade* de soluções na memória e possibilita uma *convergência muito rápida.*

Uma diferente abordagem, seguindo a linha de priorização da melhor solução, pode ser obtida:

• *Priorização da melhor solução com Probabilidade Linear* - neste caso, o agente seleciona uma solução da memória com uma distribuição linear de probabilidade. A pior solução tem uma probabilidade muito baixa de ser escolhida (quase nula) e esta probabilidade cresce na direção das melhores soluções, onde a melhor, tem uma probabilidade muito alta de ser escolhida.

5.8.2.Priorização da Solução de Pior Qualidade

Para o caso da memória de soluções unidimensionais, o agente prioriza a escolha da pior solução (segundo o valor da função objetivo) contida na memória, que não tenha sido gerada por ele. Para o caso da memória de soluções multidimensionais, o agente prioriza a escolha da pior camada contida na memória e seleciona qualquer solução contida nela.

No caso dos agentes deterministas, ele inicia a tentativa de seleção no final da memória e percorre-a seqüencialmente até encontrar a primeira que não tenha sido gerada por ele.

Esta política possibilita a exploração mais intensiva da vizinhança de soluções ruins, possibilitando a melhora das soluções iniciais, ao mesmo tempo que garante uma *alta diversidade* de soluções. O problema é que o processo de *convergência* torna-se muito *lento*, uma vez que encontrar soluções ótimas ou próximas do ótimo torna-se mais difícil, já que soluções muito ruins tendem a prejudicar a performance dos agentes no encontro de soluções muito boas.

Uma abordagem diferente, seguindo a linha de priorização da pior solução, também pode ser obtida:

• *Priorização da pior solução com Probabilidade Linear* - neste caso, o agente seleciona uma solução da memória com uma distribuição linear de probabilidade. A melhor solução tem uma probabilidade muito baixa de ser escolhida (quase nula) e esta probabilidade cresce na direção das piores soluções, onde a pior, tem uma probabilidade muito alta de ser escolhida;

5.8.3.Priorização das Soluções de Qualidade Intermediária

Neste caso é aplicada uma distribuição triangular de probabilidade para a seleção de uma solução (ou uma camada), onde os extremos são desfavorecidos em prol das soluções de qualidade intermediária entre o melhor e o pior da memória.

5.8.4.Escolha Uniforme de uma Solução

O agente *não* prioriza a escolha de nenhuma solução. *Qualquer* solução (de qualquer camada, para o caso de soluções multidimensionais) contida na memória pode ser selecionada por ele. Exceto para o caso dos agentes deterministas, onde *qualquer* solução pode ser escolhida desde que *não* tenha sido gerada ou visitada previamente por ele.

5.8.5.Priorização Dinâmica de uma Solução

Neste caso, é proposto uma utilização conjunta de mais de uma política de seleção durante a execução do A-Team.

5.9. Critérios de qualidade para a diferenciação de soluções equivalentes

A princípio, como as soluções de uma mesma camada são equivalentes entre si, ou seja, soluções da mesma camada não se dominam, elas são consideradas de mesma qualidade.

Na seleção de uma solução por um agente, por exemplo, o natural é a adoção de uma busca seqüencial até que a primeira que não tenha sido gerada ou visitada pelo agente seja encontrada.

Mas, será que existe alguma característica nas soluções eficientes k-dimensionais *que possibilite suas diferenciações?*

Um agente pode priorizar a escolha de soluções de determinada camada. Dados k objetivos, basta uma solução possuir o melhor valor num deles para se tornar uma solução equivalente às outras soluções da camada, mas, nos $k-1$ objetivos restantes ela pode possuir um valor inferior. O agente poderia priorizar, então, as soluções que não sejam dominadas em pelo menos metade dos objetivos, por exemplo.

Mais genericamente, um agente pode determinar o ***grau de dominância*** que uma solução de determinada camada deve possuir para ser escolhida. Alguns critérios para a diferenciação das soluções de uma mesma camada podem ser adotados:

• *Priorização da solução menos dominada* - neste caso, o agente seleciona uma solução de uma certa camada que não seja dominada em pelo menos p objetivos. O número p (grau de não-dominância) pode variar entre $1 \leq p \leq k$.

5.10. Resumo

Neste capítulo foi detalhado o método de **Times Assíncronos** (do inglês *Asynchronous Teams* ou A-Teams), onde foram apresentados os motivos pelos quais se afirmou no início ser ele apropriado para a resolução de Problemas Combinatórios com Múltiplas Funções Objetivo. Os parâmetros mais importantes foram analisados, tais

como: tipos de problemas, classes de agentes, fluxos de dados, estratégias de iniciação e tamanho da memória, políticas de destruição e seleção de soluções.

Um compêndio sobre aplicações de A-Teams desenvolvidas até o momento foi, também, apresentado.

As inovações apresentadas neste capítulo englobam toda a parte de projeto de A-Teams para Problemas Combinatórios Multiobjetivos. Esta é a primeira vez que se adapta tal método atendendo as características de tais problemas. Um novo tipo de organização de soluções na memória foi apresentado, onde as soluções são dispostas em camadas cada vez mais crescente de não-dominância: a Memória de Camadas de Conjuntos Não-Dominados. Algoritmos de inserção e eliminação de soluções num espaço k-dimensional (onde k é o número de objetivos do problema) foram desenvolvidos, de maneira a suportar a estrutura de memória proposta. Novas políticas de destruição e seleção de soluções foram necessárias, também visando atender a nova situação. Ainda, sugestões para o aproveitamento e para o desenvolvimento de algoritmos foram comentadas, bem como sugestões em todos os parâmetros de projeto, de tal forma que este capítulo pode servir de base de consulta para futuras aplicações de A-Teams para Problemas Combinatórios Multiobjetivos.

5.11. Notas Bibliográficas

A principal fonte de consulta sobre Times Assíncronos foi a tese de Doutorado de Souza [Sou93].

Parte do material descrito na seção 5.5, onde foram citados os passos necessários do projeto de um A-Team, pode ser encontrado em [Pei95], que desenvolveu uma *Metodologia de Especificação de Times Assíncronos* onde foi abordada a questão dos Problemas Monobjetivos, servindo de base para as adaptações feitas para Problemas Multiobjetivos.

Parte da classificação sobre algoritmos, apresentadas na subseção 5.5.4, pode ser encontrada nos trabalhos de Zanakis *et al* [ZEV89, ZE81], Müller-Merbach [Mül81], Ball e Magazine [BM81] e Weiner [Wei75].

Sobre Teoria de Probabilidade e Processos Estocásticos, podem ser citados: Taylor e Karlin [TK84] e Grimmett e Stirzaker [GS82].

Sobre a manipulação de pontos num espaço k-dimensional, o livro de Preparata e Shamos é a bibliografia recomendada [PS85].

Dentre os trabalhos envolvendo A-Teams, podem ser particularmente citados: Talukdar e Souza [TS90, ST91, TS92, Tal92, Sou93, TS93, ST93a, TSM93, TBGM96]; Talukdar e Murthy [Mur92]; Peixoto e Souza [PS94a, PS94b, Pei95]; Cavalcante e Souza [CS94, CS95, Cav95]; Camponogara e Souza [CaS95a, CaS95b, CaS95c, Cam95]; Rodrigues e Souza [RS94, RS95a, RS95b, RS95c, RS96a, RS96b]; Ribeiro e Souza [RiS95]; e, Nascimento, Xavier e Souza [NXS96].

Capítulo 6

A-Teams para o MDTSP

"Não basta saber, é preciso aplicar. Não basta querer, é preciso fazer."

Goethe.

Neste capítulo, é descrita a aplicação do método de Times Assíncronos ao Problema do Caixeiro Viajante com Múltiplas Distâncias. Os resultados computacionais são apresentados em separado, no capítulo 7.

São analisadas diferentes estratégias de projeto segundo as duas abordagens de resolução discutidas anteriormente, onde o MDTSP é resolvido como sendo vários problemas monobjetivos e como sendo realmente um problema multiobjetivo. As estratégias envolvem a aplicação de funções utilidades baseadas em métricas-Lp (L_1 até L_{10}, e L_∞) e na equação que descreve uma hipérbole, além da estratégia de manipulação de Camadas de Soluções Não-Dominadas. Tais estratégias proporcionam a organização de soluções na memória de um A-Team com o intuito de facilitar a atuação dos agentes no aproveitamento de soluções com características mais favoráveis às suas estruturas internas, aumentando as chances de melhores soluções serem geradas. Contudo, o ponto mais importante de tais estratégias é tornar a eliminação de soluções mais eficiente, de maneira a facilitar a eliminação de soluções ruins e garantir diversidade, sem prejudicar a convergência do A-Team. No caso da estratégia de manipulação de Camadas Não-Dominadas proporciona, ainda, uma aplicação mais refinada do conceito de dominância.

Primeiramente, é mostrado o porquê da escolha de A-Teams para a resolução do MDTSP. Em seguida, conforme estipulado no projeto de um A-Team, especifica-se o problema em questão, onde são determinadas a representação de uma solução válida e a qualidade desejada das soluções geradas. Os algoritmos para compor as diferentes configurações de A-Teams são, então, selecionados e parte-se para o estudo dos possíveis fluxos de dados, com a análise dos demais parâmetros de projeto de um A-Team.

6.1. Por que A-Team ?

Um A-Team gera múltiplas soluções alternativas, próximas do ótimo, ou mesmo, soluções ótimas para um dado problema combinatório de uma única função objetivo. No caso de problemas combinatórios multiobjetivos, onde não mais se busca uma única solução como resposta e sim, um conjunto de soluções de melhor compromisso para *todos* os objetivos envolvidos, natural esperar que dentre as múltiplas soluções geradas pelo A-Team ao longo do processo de execução, possa ser obtido tais soluções de melhor compromisso. Além disso, para cada uma das múltiplas soluções geradas, o A-Team possibilita a exploração de sua vizinhança, resultando, desta forma, na exploração de diferentes vizinhanças em torno de cada solução disponível.

Outro fator não menos importante é a simplicidade com que se pode aplicar, através de A-Teams, os algoritmos existentes para o tradicional TSP na resolução do MDTSP. Cada algoritmo pode ser aplicado para cada um dos vários objetivos do problema individualmente, ou seja, pode-se colocar no mesmo time várias versões dos algoritmos, onde cada versão é aplicada a uma das matrizes de distância do problema. Desta forma, nenhuma modificação na estrutura interna destes algoritmos se torna necessária para a elaboração de um Time Assíncrono para o MDTSP. Em outras palavras, um k-DTSP pode ser resolvido através de A-Teams como se fosse k TSPs, um para cada uma das k matrizes de distância do MDTSP. Tal experimento foi realizado por Rodrigues e Souza [RS95a].

Contudo, apesar da aplicação direta dos algoritmos existentes, para a obtenção de melhores resultados torna-se necessário a realização de algumas modificações significativas nos parâmetros de projeto de um A-Team para problemas multiobjetivos, tomando como base os trabalhos de Souza [Sou93] e Peixoto [Pei95], e conforme já mostrado no Cap.5. Além disso, adaptações nos algoritmos existentes para o TSP de tal forma que eles possam manipular simultaneamente várias matrizes de custo, resultam na geração de soluções de melhor compromisso para os mesmos. Tais assuntos serão abordados nas seções subseqüentes.

6.2. Adequabilidade do MDTSP à Aplicação de A-Teams

Os problemas apropriados à aplicação de A-Teams devem possuir algoritmos inadequados (ver Cap.5, subseção 5.5.3) que formem pelo menos um fluxo cíclico de dados no diagrama de decomposição de problemas (Cap.5, fig.5.3).

O MDTSP é uma generalização do clássico NP-difícil TSP e, portanto, *é pelo menos de mesma complexidade e dificuldade de resolução*. Todos os problemas NP-difíceis, como o MDTSP, se enquadram na classe de problemas que possuem apenas algoritmos inadequados para a sua resolução.

Como já afirmado, um k-DTSP (MDTSP com k matrizes de distância) pode ser resolvido através de Times Assíncronos como se fosse k TSPs. Isto significa que as inúmeras heurísticas existentes para a resolução do TSP podem ser utilizadas, sem nenhuma modificação em suas estruturas internas, para a resolução de um MDTSP

através de A-Teams. Como o TSP possui algoritmos suficientes para a formação de mais de um ciclo no diagrama de decomposição de problemas, tal como demonstrado por Souza [Sou93], e como os algoritmos existentes para o TSP são válidos para o MDTSP (através de A-Teams), este último também possui mais de um ciclo no diagrama de decomposição de problemas. Desta forma, conclui-se que o *Multi-Distance Traveling Salesman Problem* é um PMA, sendo adequado para ser resolvido através de A-Teams.

Observa-se que não foi preciso considerar os novos algoritmos desenvolvidos especialmente para o MDTSP nesta análise, ou seja, o MDTSP é um problema adequado à aplicação do método de Times Assíncronos e para isto foi preciso considerar apenas os algoritmos já existentes para o TSP padrão.

6.3. Especificação dos Agentes

No quarto capítulo, onde foi abordado o MDTSP, foram propostos vários algoritmos para a sua resolução. Tais algoritmos podem ser combinados de diferentes maneiras na composição de um A-Team, gerando diferentes fluxos de dados.

A incorporação de um algoritmo num Time Assíncrono o transforma num *agente* do time, já que se torna necessário o estabelecimento de um protocolo de comunicação para se tornar possível, e consistente, a interação entre todos os algoritmos e o acesso às memórias compartilhadas.

Os algoritmos para o MDTSP utilizados durante todos os experimentos, e que já foram detalhados no capítulo 4, são apresentados nas tabelas seguintes, onde os algoritmos pertencentes às colunas nomeadas de FO são os que manipulam apenas uma função objetivo (a maioria já disponíveis na literatura) e os algoritmos pertencentes às colunas nomeadas de MFO são os que manipulam múltiplas funções objetivos (todos desenvolvidos neste trabalho). Sendo assim:

Tabela 6. 1: Algoritmos de construção e de melhoria utilizado nos testes computacionais.

	FO	MFO
Algoritmos de Construção	Furthest Insertion - FI	Furthest Insertion of Compromise - FIC
	Arbitrary Insertion - AI	Arbitrary Insertion of Compromise - AIC
	Nearest Insertion - NI	Nearest Insertion of Compromise - NIC
	Random - R	_____
Algoritmos de Melhoria	2-ótimo - 2OPT	2-ótimo c/ dominância - 2OPT-D
	3-ótimo - 3OPT	3-ótimo c/ dominância - 3OPT-D
	Or-ótimo - OROPT	Or-ótimo c/ dominância - OROPT-D
	Lin-Kernighan - LK	Lin-Kernighan c/ dominância - LK-D
	2-ótimo c/ soluções intermediárias- 2OPT-I	_____
	3-ótimo c/ soluções intermediárias- 3OPT-I	_____
	Or-ótimo c/ soluções intermediárias- OROPT-I	_____
	LinKernighan c/ soluções intermediárias- LK-I	_____
Algoritmos de Relaxação	Held-and-Karp - HK	_____
	2-Matching - 2M	_____
	Fractional 2-Matching - F2M	_____

Na tabela seguinte (tab.6.2) são apresentados os agentes característicos de A-Teams, ou seja, aqueles algoritmos que possibilitam a formação de ciclos nos fluxos de dados dos A-Teams. Foram desenvolvidos os seguintes algoritmos - de consenso por união e por intersecção (ver Cap.5, subseção 5.5.9):

Tabela 6. 2: Algoritmos de consenso - por união e interseção - implementados.

	FO	MFO
Consenso por Intersecção	Deconstructor 1 - DEC	_____
	Deconstructor 2 - DEC-NN	Deconstructor 3 - DEC-NNC
Consenso por União	Mixer - MI	_____
	Tree Mixer - MI	_____

Em seguida são apresentados os algoritmos de destruição implementados e as políticas de seleção embutidas nos agentes.

6.3.1. Algoritmos de Destruição

Os algoritmos de destruição não geram nenhum tipo de solução e, como visto no Cap.5, a função de tais algoritmos é selecionar uma determinada solução (segundo as características de cada agente de destruição) e eliminá-la da memória.

Vale ressaltar, entretanto, que o algoritmo de destruição também decide se a nova solução deve, sob algum critério, permanecer na memória. Todos os algoritmos de destruição utilizados nos testes computacionais, por exemplo, somente aceitam que a solução inserida mais recentemente na memória permaneça inserida, se ela for de melhor qualidade do que a pior solução contida na memória e, desta forma, seleciona uma outra para ser eliminada. No caso da memória de Camadas de Soluções Não-Dominadas, uma nova solução somente permanece inserida se pertencer às camadas melhores do que à pior presente na memória.

Os algoritmos de destruição desenvolvidos foram os seguintes:

DP - Destruição da Pior solução contida na memória;

DU - Destruição com distribuição Uniforme de probabilidade de soluções contida na memória;

DL - Destruição com distribuição Linear de probabilidade de soluções contida na memória;

DT - Destruição com distribuição Triangular de probabilidade de soluções contida na memória;

DPQ - Destruição de Qualquer solução da Pior camada contida na memória;

DUQ - Destruição de Qualquer solução contida numa determinada camada selecionada com distribuição Uniforme de probabilidade;

DLQ - Destruição de Qualquer solução contida numa determinada camada selecionada com distribuição Linear de probabilidade;

DTQ - Destruição de Qualquer solução contida numa determinada camada selecionada com distribuição Triangular de probabilidade.

Tabela 6. 3: Algoritmos de destruição desenvolvidos.

	FO	MFO
Algoritmos de Destruição	Destruidor da Pior Solução - DP	Destruidor na Pior Camada - Qquer Solução - DPQ
	Destruidor Uniforme de Solução - DU	Destruidor Uniforme na Camada - qquer Solução - DUQ
	Destruidor Linear de Solução - DL	Destruidor Linear na Camada - qquer Solução - DLQ
	Destruidor Triangular de Solução - DT	Destruidor Triangular na Camada - qquer Solução -

6.3.2. Políticas de Seleção

Manteve-se o nome política de seleção porque tais procedimentos são incorporados em todos os demais algoritmos como parte do protocolo de comunicação entre os mesmos e as memórias compartilhadas. As políticas de seleção definem como o acesso às memórias será feito, ou melhor, como será escolhida a solução (ou soluções) que servirá de entrada para o agente em questão.

Para os agentes que lêem informações das Memórias de Soluções Não-Refinadas, Soluções Parciais e Soluções Relaxadas, a política de seleção utilizada foi a **FIFO** (*First In First Out*), já que as soluções não são organizadas sob nenhum critério e, portanto, não é possível avaliá-las qualitativamente.

Para os agentes que lêem da Memória de Soluções Completas e Não-Refinadas a política de seleção é um fator importante, visto que os mesmos podem ter um desempenho melhor dependendo do tipo da solução selecionada e visto que nesta memória as soluções são organizadas por um ou mais critérios de qualidade.

Os algoritmos de melhoria LK, LK-I, LK-D, 3OPT, 3OPT-I e 3OPT-D, por possuírem uma granulosidade grossa (ver 5.5.8), devem selecionar boas soluções para que tenham oportunidade de encontrar melhores soluções com menor esforço, possibilitando ao restante dos algoritmos aproveitar rapidamente tais soluções boas e fazendo com que o A-Team tenha um melhor desempenho em menos tempo de execução. Por este motivo, tanto o LK quanto o 3OPT, e suas variantes, utilizam a política de seleção que ***prioriza a melhor***, ou seja, são priorizadas as melhores soluções contidas na memória, começando a busca pelo início da memória (melhores soluções ou melhores camadas) e selecionando a primeira solução que não tenha sido gerada ou anteriormente selecionada pelos mesmos.

Todos os demais agentes que podem atuar sobre a Memória de Soluções Completas e Refinadas (2OPT, 2OPT-I, 2OPT-D, OROPT, OROPT-I, OROPT-D, NI, NIC, FI, FIC, AI, AIC, DEC, DEC-NN e DEC-NNC), selecionam uma solução aleatoriamente, visto que possuem uma granulosidade mais fina e, desta forma, podem melhorar um número maior de soluções, num mesmo intervalo de tempo, do que o LK e o 3OPT. Isto possibilita um aproveitamento de todas as soluções, dando chances iguais para as melhores e para as piores soluções contidas na memória.

6.4. Especificação das Memórias

A especificação das memórias compartilhadas constitui um ponto importante do projeto de um A-Team, pois o tipo de organização das soluções, seu tamanho e a priorização de soluções na memória influem profundamente na performance do A-Team (ver 5.5).

Os tipos de memórias dependem dos algoritmos disponíveis. Pode ser observado que dentre os algoritmos listados, há aqueles que geram soluções completas, parciais ou ainda relaxadas. Dado este contexto, onde três tipos de soluções podem ser geradas e lembrando que um A-Team deve sempre possuir uma memória que contenha as melhores soluções válidas, foram utilizadas os seguintes tipos de memórias de soluções:

- **Memória de Soluções Completas e Refinadas.**
- **Memória de Soluções Completas e Não-Refinadas.**
- **Memória de Soluções Parciais.**
- **Memória de Soluções Relaxadas.**

Nos primeiros experimentos, apenas os três primeiros tipos de memória foram utilizados. O objetivo da inclusão da memória de soluções relaxadas, foi mostrar a aplicabilidade de soluções relaxadas (usando agentes de relaxação) para a obtenção de soluções de melhor compromisso na resolução de Problemas de Otimização Combinatória Multiobjetivos através de A-Teams.

O tamanho da memória principal (Memória de Soluções Completas e Refinadas) foi determinado empiricamente.

Os algoritmos utilizados para iniciar a memória principal do A-Team foram os de construção, usados intercaladamente nos A-Teams desenvolvidos.

A configuração resultante da adoção dos quatro tipos de memória em conjunto, possibilitou a geração dos melhores resultados. Poderá ser observado no capítulo seguinte (Cap.7) os principais testes realizados.

6.5. Estrutura Geral de A-Teams para o MDTSP

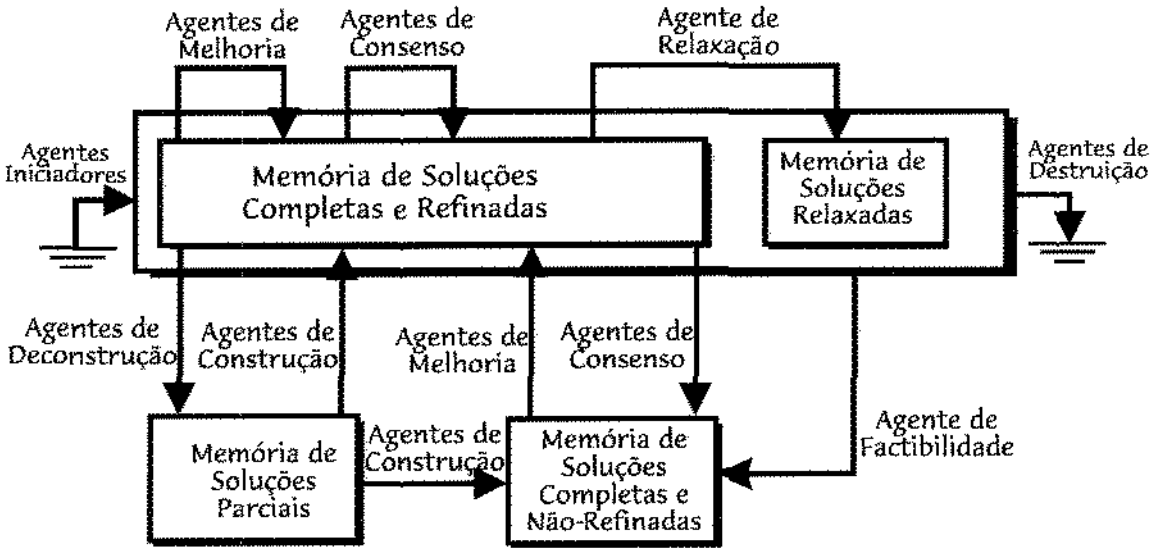
As estruturas de A-Teams devem ser compostas de maneira a proporcionarem o melhor desempenho possível. Isto envolve a maneira com que os algoritmos são dispostos e sobre quais memórias eles atuam. Diversas combinações de tais algoritmos podem ser estabelecidas e o conjunto completo de algoritmos disponíveis é o seguinte:

- Doze agentes de melhoria (2OPT, 2OPTI, 2OPTD, 3OPT, 3OPTI, 3OPTD, LK, LKI, LKD, OROPT, OROPTI, OROPTD);
- Sete agentes de construção (NI, NIC, AI, AIC, FI, FIC, R);
- Três algoritmos de consenso por interseção - (DEC, DEC-NN, DEC-NNC);
- Dois algoritmos de consenso por união (MI, TM - este último é também de factibilidade);

- Um algoritmo de relaxação (HK);
- Oito algoritmos de destruição (DP, DU, DL, DT, DPQ, DUQ, DLQ, DTQ).

Com base neste conjunto de algoritmos e também na estrutura geral apresentada no capítulo 5, pode ser estabelecida uma estrutura geral para o MDTSP como a ilustrada na figura 6.1.

Figura 6. 1: Estrutura de um A-Team para o MDTSP. Os agentes de melhoria lêem e escrevem na memória de soluções completas e refinadas e podem ler também da memória de soluções completas e não-refinadas. Os algoritmos de construção além de iniciarem o preenchimento da memória, lêem da memória de soluções parciais e escrevem na memória de soluções completas e não-refinadas. O agente de relaxação lê da memória de soluções completas e escreve na memória de soluções relaxadas. Os agentes de consenso por união lêem da memória de soluções refinadas e escrevem na memória de soluções não-refinadas.



A seguir serão discutidos os possíveis fluxos de dados derivados deste fluxo geral.

6.5.1. Estipulando os Fluxos de Dados

A definição do fluxo de dados é um fator importante no desempenho de um A-Team, visto que pequenas alterações na disposição dos algoritmos podem resultar tanto numa melhora quanto numa degradação considerável.

O algoritmo de melhoria que vier a ocupar a posição de ligação entre a memória de soluções não-refinadas e a de soluções refinadas, nesta ordem, deve ser rápido o suficiente para permitir um contínuo fluxo de soluções e evitar um possível gargalo neste fluxo. Isto pode ser solucionado com a inclusão do algoritmo 2OPT, OROPT ou do 3OPT, sendo este último com pequenas alterações de modo a deixá-lo mais rápido.

Para a ligação entre a memória de soluções completas e a de soluções relaxadas, apenas um algoritmo está disponível, que é o de Held-Karp.

No caso do algoritmo de factibilidade disponível, o Tree Mixer, como ele precisa de uma solução completa e de uma relaxada como entrada, ele atua sobre as memórias de

solução completa e de soluções relaxadas, escrevendo na memória de soluções não-refinadas.

Os algoritmos de construção lêem da memória de soluções parciais e escrevem na memória de soluções não-refinadas. Eles poderiam escrever também na memória de soluções completas e refinadas diretamente.

Os algoritmos de consenso por interseção, os *Deconstructors*, lêem da memória de soluções completas e escrevem na memória de soluções relaxadas.

Os algoritmos de destruição atuam sempre sobre a memória de soluções completas, selecionando uma solução conforme a política neles embutida e eliminando-a da memória.

Podem ser estipuladas estruturas que envolvam apenas uma ou algumas das memórias e agentes apresentadas na fig. 6.1. Vários testes foram realizados neste sentido, seja para comprovar a eficiência em escala seja para tentar encontrar a melhor configuração possível. Tais testes podem ser encontrados no capítulo de resultados (Cap.7).

6.6. Estratégias para a Organização de Soluções na Memória

Cada agente de um Time Assíncrono possui total autonomia para selecionar qualquer uma das soluções contidas na memória e, até mesmo, para guiar e organizar as soluções na memória da maneira que melhor lhe convier. Desta forma, as soluções poderiam estar dispostas aleatoriamente na memória (sem nenhuma ordem de qualidade entre as mesmas). Entretanto, manter as soluções organizadas na memória, com o conhecimento da ordem relativa de qualidade entre as mesmas, facilita o desenvolvimento de um A-Team, bem como o acompanhamento da evolução do nível de qualidade das soluções que vão sendo geradas.

De qualquer maneira, a organização de soluções na memória não impõe nenhuma restrição de acesso aos agentes do time.

Para problemas combinatórios com uma única função objetivo, uma das maneiras de organização das soluções (trajetórias) na memória é ordenando-as segundo seus valores em relação à função objetivo. As soluções podem ser dispostas seqüencialmente em ordem crescente, por exemplo, onde na primeira posição tem-se a solução de melhor valor (no caso de um problema de minimização, o melhor valor seria o menor), e, na última posição a solução de pior valor. Desta forma, a seleção de soluções para utilização dos agentes, seja para a geração de outras soluções ou seja para suas eliminações da memória, pode ser feita de maneira rápida e eficiente.

Estabelecer uma organização de soluções na memória quando vários objetivos devem ser otimizados simultaneamente, como é o caso do MDTSP, requer maiores cuidados. Quando abordamos os problemas multiobjetivos e as estratégias de resolução encontradas na prática (ver Cap.3, subseção 3.3), vimos que a estratégia mais adotada para contornar o fato de se lidar com múltiplos objetivos, é a transformação dos mesmos num só, seja através de uma combinação linear dos objetivos, seja pela priorização de um

dos objetivos, transformando-se os outros em restrições do problema, restando, nos dois casos, uma única função objetivo para ser otimizada.

Vimos, também, que são vários os problemas ocasionados pela aplicação de uma função utilidade, onde, por exemplo, a melhor solução segundo a função utilidade estabelecida pode nem mesmo ser uma solução pertence ao melhor conjunto de soluções eficientes (fig.6.3). Agora, como o objetivo deste trabalho é mostrar a adequabilidade do método de Times Assíncronos, que é um método *aproximado* de resolução de problemas combinatórios e não gerar exatamente todo o conjunto de soluções eficientes (Pareto Ótimo), o que até então é inviável [Kar95, Ste86], procuramos desenvolver funções cujos comportamentos resultassem num conjunto de soluções com distribuição, no espaço de soluções factíveis, similar ao encontrado num conjunto de soluções não-dominadas.

O uso de funções utilidade torna possível a adoção do mesmo tipo de organização de soluções utilizada para problemas monobjetivos, já que o objetivo passa a ser otimizar uma única função: a função utilidade.

Nas sub-seções seguintes, serão apresentadas as funções utilidades elaboradas. Em seguida, será abordada também a estrutura para organização das soluções baseada em conjuntos de soluções não-dominadas. Tal estrutura permite uma organização das soluções bem distinta, de tal forma que a evolução das soluções na memória compartilhada seja feita sem a priorização de nenhum objetivo especificamente e, sim, com a consideração de todos os objetivos simultaneamente.

6.6.1. Funções Utilidade Propostas

A idéia inicial da utilização de uma função utilidade partiu da análise do trabalho desenvolvido por Murthy [Mur92], que usou A-Teams para a resolução de um Problema de Projeto de um Robô Manipulador (ver Cap.5). Como estratégia de resolução, Murthy desenvolveu um algoritmo de destruição onde foi embutida uma função baseada na distância fornecida pela métrica-L_2 (distância Euclidiana), entre a solução gerada e o melhor conjunto de soluções não-dominadas que iam sendo obtidas durante a execução. Para isto, foi necessário manter o melhor conjunto de soluções não-dominadas na memória durante a execução.

A métrica-L_2 foi utilizada por Murthy somente para que as soluções pudessem ser destruídas de maneira mais eficiente. Os demais agentes selecionam qualquer solução contida na memória.

Murthy sugeriu também, o uso das métricas L_1 e L_∞, cujos experimentos ele não chegou a realizar.

Neste trabalho, foram aplicadas as métricas L_1 à L_{10}, e a métrica L_∞ [Ste86], além da equação que descreve uma hipérbole. Poderá ser observado no capítulo seguinte que algumas das estratégias adotadas neste trabalho mostraram um desempenho melhor do que a proposta por Murthy, na resolução do MDTSP.

O restante desta subseção aborda com maiores detalhes as funções utilidades desenvolvidas.

6.6.1.1. Métricas Lp

Uma métrica em R^k é uma função distância que atribui a cada par de vetores $x,y \in R^k$ um escalar $||x-y|| \in R^k$ se e somente se a função satisfaz, para todo $z \in R^k$, os quatro axiomas seguintes [Ste86]:

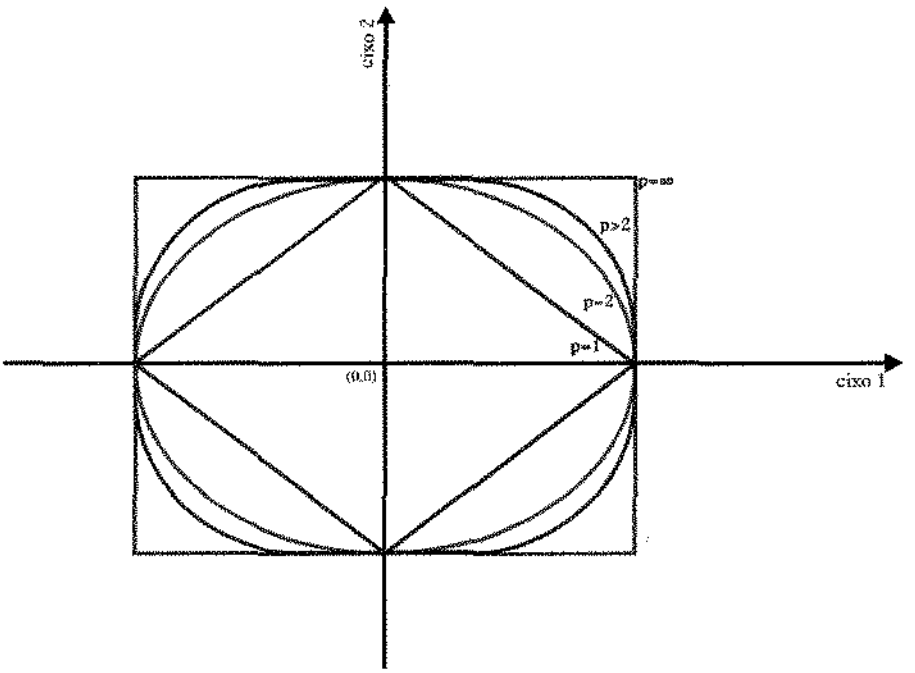
1. $||x-y|| \geq 0$ e $||x-x||=0$.
2. $||x-y||=||y-x||$.
3. $||x-y|| \leq ||x-z||+||z-y||$.
4. Se $x \neq y$, então $||x-y||>0$.

Para uma métrica-L_p a distância entre dois vetores $x,y \in R^k$ é dada por:

$$\|x-y\|_p = \left[\sum |x_i - y_i|^P\right]^{1/p} \quad , \quad p \in \{1,2,3,...\} \cup \{\infty\} \qquad \text{Eq. 6 1}$$

A fig.6.2 apresenta o contorno gráfico destas métricas, de L_1 à L_∞.

Figura 6. 2: Pontos que definem o local geométrico das métricas-L_p. Analisando o primeiro quadrante somente, observa-se que a métrica L_1 define o segmento de reta tangente à origem dos eixos. A métrica-L_2 define uma semicircunferência cujo centro é a origem dos eixos, sendo seguida pelas demais métricas-L_p, com $p>2$. As distâncias entre os contornos definidos pelas métricas diminui cada vez mais à medida que p cresce. Quando $p=\infty$, caso da métrica-L_∞, o contorno assume uma formato quadrado, definido por segmentos de retas paralelos aos eixos.



Métrica-L_1

A métrica-L_1 é conhecida como ***métrica de Manhathan***.

Na métrica-L_1, a distância de uma solução ao início dos eixos consiste na somatória de seus valores em cada eixo.

$$\|x-y\|_1 = \{|x_1-y_1|+...+|x_k-y_k|\} \quad \text{Eq. 6. 2}$$

Métrica-L_2

A métrica-L_2 é denominada de ***distância Euclidiana*** e sua equação pode ser descrita como segue:

$$\|x-y\|_2 = \sqrt[2]{|x_1-y_1|^2+...+|x_k-y_k|^2} \quad \text{Eq. 6. 3}$$

Métrica L_3 à L_{10}

Para as métricas L_3 à L_{10}, basta substituir o número 2 da equação anterior pelo seu correspondente *p* em cada métrica. Então:

$$\|x-y\|_p = \sqrt[p]{|x_1-y_1|^p+...+|x_k-y_k|^p} \quad \text{Eq. 6. 4}$$

Métrica-L_∞

A métrica-L_∞ é conhecida por ***métrica de Tchebycheff***, ou ainda, ***métrica do Máximo*** (*Maximum metric*).

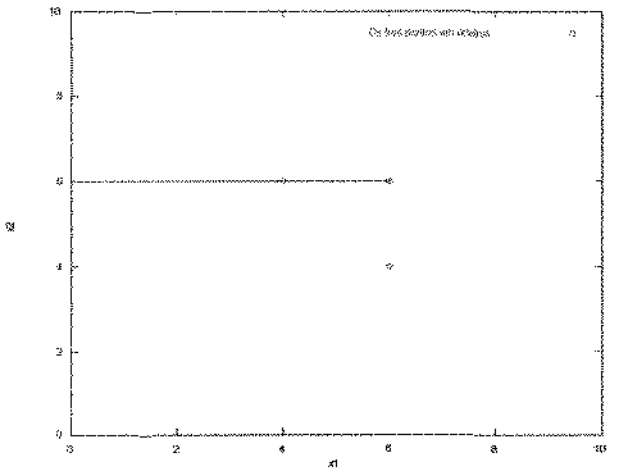
Neste caso, à distância de uma solução *k*-dimensional à uma outra solução se resume à escolha do valor em relação ao eixo, dentre os *k* eixos, de maior valor. Então:

$$\|x-y\|_\infty = max\{|x_1-y_1|,|x_2-y_2|,...,|x_k-y_k|\} \quad \text{Eq. 6. 5}$$

Tal expressão para a métrica-L_∞ é devido ao fato de que, quando $p=\infty$, o valor de maior magnitude domina totalmente os demais valores.

Na fig. 6.3 é mostrada uma análise de três pontos num espaço bidimensional segundo à métrica-L_∞. Neste exemplo, faz-se uma suposição de que apenas três pontos constituem a região factível do problema em análise. Os três pontos (4,6), (6,4) e (6,6) possuem a mesma distância em relação à origem dos eixos (0,0), segundo à métrica-L_∞. Entretanto, o ponto (6,6) é dominado tanto pelo ponto (4,6) quanto pelo ponto (6,4). O resultado final da aplicação desta métrica poderia indicar que o ponto (6,6) é tão bom quanto os pontos (4,6) e (6,4), o que não é verdade.

Figura 6. 3: Análise de três pontos num espaço bidimensional segundo a métrica-L_∞. Supondo que estes três pontos com coordenadas (4,6), (6,4) e (6,6) fossem os únicos de uma determinada região factível de um problema, apenas os dois primeiros fariam parte do Pareto Ótimo, pois o ponto (6,6) é dominado tanto por (4,6) quanto por (6,4). No entanto, segundo a métrica L_∞, os três pontos possuem a mesma distância em relação à origem dos eixos.



Observa-se que as soluções que possuem a mesma distância em relação à origem dos eixos, segundo a métrica L_∞, nem sempre pertencem ao mesmo conjunto de soluções não-dominadas. Entretanto, em relação às demais métricas L_p, as soluções com distâncias iguais pertencem, necessariamente, ao mesmo conjunto de soluções não-dominadas. As soluções com mesma distância (que descrevem um contorno gráfico) segundo tais métricas são dispostas de tal forma a serem sempre monotonicamente crescente em pelo menos um dos eixos envolvidos, garantindo a não-dominância entre as mesmas. Isto significa dizer que, dado qualquer conjunto de soluções, a solução com menor distância resultante da aplicação de qualquer das métricas L_p (com exceção de L_∞) pertencerá sempre ao melhor conjunto de soluções não-dominadas.

O que podemos supor é que o desempenho de tais métricas, quando utilizadas como critério de avaliação de soluções em A-Teams, depende das características do conjunto de soluções inicialmente gerado, bem como das demais soluções geradas durante o processo de execução. E, principalmente, as métricas proporcionam uma distinção significativa dentre as melhores e piores soluções, ou seja, as piores soluções estão longe tanto do centro quanto dos eixos envolvidos, simultaneamente. Isto significa dizer que o uso das métricas tende a eliminar soluções muito ruins.

O capítulo 7 apresenta uma análise comparativa do desempenho de tais métricas em A-Teams.

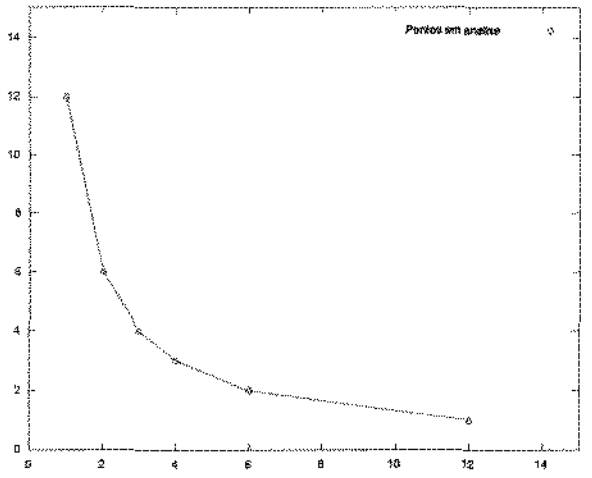
6.6.1.2. Equação que descreve uma Hipérbole

A equação que descreve uma hipérbole fornece como resultado o produto dos valores dos k eixos das soluções em análise. Então:

$$F(x_1,...,x_k) = x_1 \times x_2 \times ... \times x_k \qquad \text{Eq. 6. 6}$$

Observa-se na figura 6.4 o contorno gráfico para um conjunto de pontos com mesmo valor segundo a equação da hipérbole.

Figura 6. 4: Análise de pontos num espaço bidimensional segundo a equação que descreve uma hipérbole. Todos os pontos do gráfico - (1,12), (2,6), (3,4), (4,3), (6,2), (12,1) - fazem parte do melhor conjunto de soluções não-dominadas deste conjunto de soluções e apresentam o mesmo valor segundo a equação que descreve a hipérbole: o melhor conjunto de soluções não-dominadas é a própria hipérbole deste exemplo.



Nota-se que o contorno gráfico da equação que descreve uma hipérbole se assemelha mais ao que esperamos ser um bom conjunto de soluções não-dominadas. Todos os pontos do gráfico - (1,12), (2,6), (3,4), (4,3), (6,2), (12,1) - fazem parte do melhor conjunto de soluções não-dominadas.

Segundo a métrica-L_2, por exemplo, este mesmo conjunto de pontos teria como soluções de menor distância apenas os pontos (3,4) e (4,3), quando todos os pontos envolvidos são eficientes entre si (fazem parte do mesmo conjunto de soluções não-dominadas).

Para este pequeno conjunto fixo de soluções, a equação que descreve uma hipérbole tem um desempenho melhor do que o obtido pela métrica-L_2. Mais uma vez, contudo, ressalta-se o fato de que a aplicação consecutiva de tais funções num conjunto dinâmico de pontos (durante a execução de um A-Team), apresenta peculiaridades que sobrepõe tais conclusões, como poderá ser observado no próximo capítulo (Cap.8).

6.6.2. Estratégias para a utilização das Métricas-L_p

Como abordado no capítulo 5, as funções utilidade são utilizadas num Time Assíncrono para proporcionar uma organização das soluções na memória, ou seja, servem como critério de avaliação (função objetivo) das soluções.

Para todas a métricas utilizadas (menos para a métrica-L_1, pois seria redundante), duas estratégias distintas foram estipuladas:

1. A primeira, consiste na ordenação das soluções na memória segundo suas *distâncias* (conforme a métrica em questão) *em relação à origem dos eixos* definidos pelos objetivos do problema;

2. A segunda, consiste na ordenação das soluções a memória segundo suas *distâncias* (conforme a métrica em questão) *em relação a um limite superior* estabelecido no início da execução do A-Team.

A função que descreve uma hipérbole também não foi considerada nestas duas estratégias, porque depende apenas da solução em análise, sem necessitar de uma segunda solução para fornecer seu resultado.

6.6.2.1. Distância em relação à origem dos eixos

Neste caso, basta considerar um dos dois pontos como sendo a origem dos eixos (vetor nulo *k*-dimensional). Então, aplicando-se à fórmula geral das métricas- Lp:

$$\|x\|_p = \left[\Sigma |x_i|^p\right]^{1/p} \quad , \quad p \in \{1,2,3,...\} \cup \{\infty\} \qquad \text{Eq. 6.7}$$

A figura 6.2 representa esta idéia, se considerarmos apenas o primeiro quadrante do gráfico. Pode ser observado que todos os pontos que definem o local geométrico das métricas são definidos em relação à origem dos eixos. E, como para este trabalho são considerados somente valores inteiros e positivos, apenas o comportamento das métricas no primeiro quadrante interessa.

6.6.2.2. Distância em relação a um Limite Superior

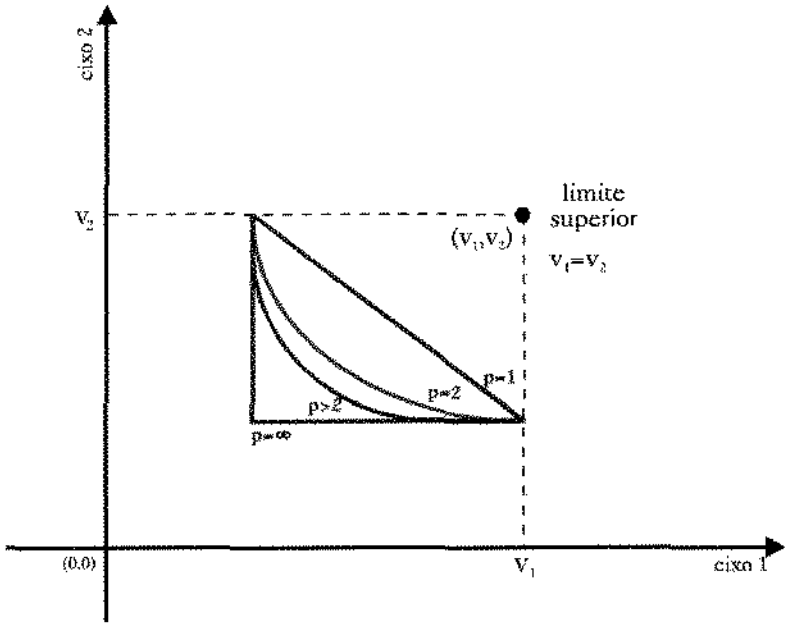
Neste caso, um limite superior é estipulado logo no início da execução do A-Team. Para se estabelecer um limite superior, uma seqüência de soluções são geradas aleatoriamente (na maioria dos experimentos foram geradas 10 soluções) e, a partir deste conjunto, escolhe-se o maior valor alcançado por qualquer uma das soluções em qualquer um dos eixos. É produzido, então, um ponto com tal valor em todos os eixos (todos os eixos do ponto fictício com o pior valor dentre todos os valores das soluções geradas). A distância passa a ser calculada entre uma solução e este ponto fictício (limite superior) e não mais em relação à origem dos eixos dos objetivos.

Também, o que se deseja agora são àqueles pontos com os **maiores valores** segundo as métricas em questão, em relação ao Limite Superior. Ou seja, no cálculo da distância serão utilizados dois pontos: um é o Limite Superior estabelecido (fixo para todos os pontos em análise) e o outro é cada um dos pontos que se está analisando, um por vez.

Quanto mais distante do Limite Superior, melhor é o ponto em análise.

Pode ser observado na figura 6.5 que o contorno gráfico das métricas é invertido, se considerarmos a origem dos eixos, mas, se considerarmos o Limite Superior como o ponto de referência, o contorno gráfico apresentado na fig. 6.5 é similar ao quarto quadrante da figura 6.2, que apresenta o local geométrico das métricas.

Figura 6. 5: Contorno gráfico das métricas Lp em relação a um Limite Superior. Observa-se que os contornos assumem um formato inverso ao obtido em relação à origem dos eixos.



Para o exemplo da subseção 6.6.1.2, cujos pontos (1,12), (2,6), (3,4), (4,3), (6,2), (12,1) pertencem ao contorno gráfico que representa uma hipérbole, a métrica L_2 em relação à origem dos eixos daria como resultado (pontos com o menor valor segundo tal métrica) os pontos (3,4) e (4,3). Neste caso, o resultado da aplicação da métrica L_2 em relação a um Limite Superior seriam os pontos (3,4), (4,3) e também, os pontos (2,6), (6,2), considerando valores inteiros.

A aplicação de tal estratégia, portanto, parece amenizar o problema ocorrido com a aplicação das métricas L_p em relação à origem dos eixos, onde os pontos mais próximos de um dos eixos, e longe dos outros, são desprezados em prol dos pontos centrais. Em relação a um Limite Superior, o comportamento de tais métricas parece melhorar, ou seja, tendem a estender a região de pontos favorecidos por elas.

Os resultados obtidos favorecem esta suposição, conforme poderá pode ser observado no capítulo 7.

Para diferenciar as métricas aplicadas em relação a um Limite Superior, das métricas em relação à origem dos eixos, será adicionado ao nome das mesmas a letra **I**. Portanto, métrica-L_1I à métrica-L_{10}I e métrica-L_∞I, indicam a aplicação das métricas em relação a um Limite Superior.

6.6.3. Manipulando Camadas de Soluções Não-Dominadas

Tal estratégia foi detalhada no Cap.5 (seção 5.6) e tem por objetivo organizar as soluções na memória segundo o grau de dominância de umas com as outras.

O uso desta estratégia, diferentemente das funções utilidades, permite uma avaliação das soluções segundo o proposto teoricamente, com os conceitos de

dominância, eficiência e Pareto Ótimo. A evolução das soluções na memória, durante a execução do A-Team é feita sempre com base nestes conceitos, onde soluções cada vez menos dominadas vão sendo geradas.

A única ressalva no uso desta aplicação é devido ao tempo de execução. A complexidade computacional do algoritmo para a manipulação de soluções não-dominadas (pontos num espaço *k*-dimensional) é pelo menos quadrática em relação ao número de soluções (m) e à dimensão do problema (k), ao passo que com função utilidade isto fica em tempo linear - $O(m)$.

Os resultados desta aplicação foram melhores dos que os obtidos com qualquer uma das funções utilidades descritas anteriormente (ver Cap.7).

6.7. Método para Avaliação de Conjuntos de Soluções Não-Dominadas

Durante o desenvolvimento deste trabalho, nos deparamos com uma dificuldade:

Como avaliar os melhores conjuntos de soluções não-dominadas obtidos pelas diferentes estratégias?

Por exemplo, aplicando-se a métrica-L_2 é obtido um conjunto de soluções não-dominadas e aplicando-se a métrica-L_3 é obtido um outro conjunto. Mas, o que fazer em seguida? O que pode nos ocorrer primeiramente é verificar o número de soluções não-dominadas que cada estratégia gerou, entretanto, isto não nos diz muita coisa: um conjunto não-dominado com menos soluções, encontrado como solução de um problema, pode dominar a maioria das soluções não-dominadas de outro conjunto não-dominado maior.

Se pensou, então, numa comparação relativa entre conjuntos de soluções não-dominadas. O método, denominado de **índice de pertinência**, reúne todas as soluções dos conjuntos não-dominados em avaliação e extrai desta união de soluções o melhor conjunto de soluções não-dominadas (*Maxima Problem* [PS85], como citado no Cap.5). Desta forma, pode-se avaliar o **índice de pertinência** de cada subconjunto desta união formada. Então:

$$IP_{ND_i} = \frac{|ND_i|}{\hat{\Theta}[\bigcup_{i=1}^{Tot} ND_i]} \qquad \text{Eq. 6. 8}$$

onde TP_{ND_i} corresponde ao índice de pertinência do conjunto i na união de conjuntos não-dominados resultantes. $|NDi|$ é a cardinalidade do conjunto não-dominado ND_i, e Tot é o número total de conjuntos não-dominados em avaliação.

$\hat{\Theta}$ representa o melhor conjunto não-dominado possível da união de todos Nd_i.

Pode ser aplicada na avaliação de vários conjuntos não-dominados de uma única vez. Por exemplo, dado um conjunto não-dominado com 7 soluções e outro conjunto não-dominado com 6 soluções, o primeiro passo é reunir as 13 soluções num só conjunto e, em seguida, encontrar o conjunto não-dominado desta união. Como pode ser

observado na figura 6.6, o conjunto não-dominado resultante, em vermelho, possui 3 soluções do primeiro conjunto (pontos no formato de losango, em verde) e 4 soluções do segundo conjunto (pontos no formato de cruz, em azul). Então:

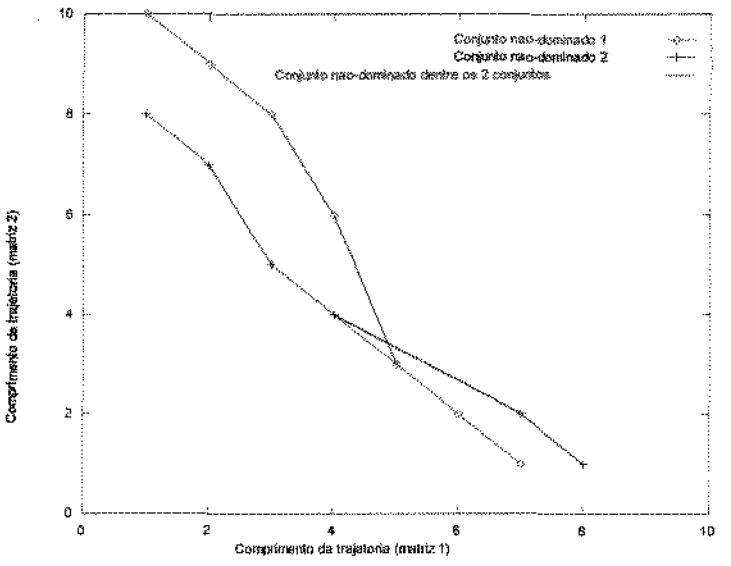
Para o exemplo da figura 6.6, portanto, temos:

$$IP_{ND_1} = \frac{|ND_1|}{\Theta[\bigcup_{i=1}^{2} ND_i]} = \frac{3}{7} = 0{,}428 \cong 0{,}43 \text{ ou } 43\%$$

$$IP_{ND_2} = \frac{|ND_2|}{\Theta[\bigcup_{i=1}^{2} ND_i]} = \frac{4}{7} = 0{,}571 \cong 0{,}57 \text{ ou } 57\%$$

Apesar do primeiro conjunto (losangos) possuir mais soluções do que o segundo conjunto (cruzes), 7 e 6 respectivamente, a taxa de pertinência é de 43%, contra 57% do segundo conjunto. Portanto, dentre estes dois conjuntos, o melhor é o segundo conjunto (pontos no formato de losangos, em azul).

Figura 6. 6: Exemplo gráfico de um super-conjunto não-dominado dentre 2 conjuntos não-dominados gerados por diferentes estratégias.



Quando a diferença entre as taxas de pertinência é relativamente pequena, os conjuntos podem ser considerados equivalentes. Desta forma, adotou-se 5 faixas de valores:

0% ——— 20%
20% ——— 40%
40% ——— 60%
60% ——— 80%
80% ——— 100%

Dentro de cada uma destas 5 faixas de taxas de pertinência, os conjuntos não-dominados são considerados equivalentes.

O índice de pertinência servirá para avaliar as diferentes estratégias desenvolvidas, como poderá ser observado no Capítulo 7.

6.8. Detalhes de Implementação

Primeiramente, foram desenvolvidas versões concorrentes de A-Teams, para depois as mesmas serem paralelizadas.

Os testes computacionais concorrentes foram realizados no Instituto de Computação (IC) da UNICAMP, composto por estações de trabalho SUN SPARCstation conectadas em rede, sendo três multiprocessadas, com 2, 4 e 8 processadores respectivamente.

Os testes computacionais em paralelo foram realizados em dois ambientes distintos:

O primeiro, no próprio IC da UNICAMP. O segundo ambiente, proporcionado pelo Centro Nacional de Processamento de Alto Desempenho - CENAPAD, composto por 8 máquinas disponíveis para o processamento paralelo: 1 CLUSTER RISC/6000 com 8 processadores e 1 IBM/SP1 com 8 processadores. Os 8 processadores do SP1 são os dedicados ao processamento pesado (muitas horas de CPU). Além destas, 6 máquinas interativas com 1 processador cada, podem ser utilizadas durante os ajustes dos programas.

6.8.1. Processamento Paralelo

Um A-Team possui uma estrutura modular que o torna muito suscetível à aplicação de processamento paralelo. Os agentes são independentes uns dos outros e a comunicação entre os mesmos é assíncrona. Isto faz com que cada agente possa ser diretamente associado a um processo. As memórias compartilhadas devem ser implementadas individualmente, de forma a permitir também que cada uma seja vista como um processo.

O fato da maioria dos agentes de um A-Team possuir uma granulosidade grossa (ver 5.5.8) também contribui para a aplicação de processamento paralelo.

Devido aos motivos apresentados acima, um A-Team é adequado à aplicação de processamento paralelo numa estrutura cliente/servidor, onde os agentes são os clientes e as memórias os servidores de dados. A ferramenta utilizada para a implementação paralela do A-Team é descrita em seguida.

6.8.1.1. *Parallel Virtual Machine* -PVM

Parallel Virtual Machine - PVM foi a ferramenta utilizada para a implementação de versões paralelas dos A-Teams. É um software de domínio público desenvolvido pelo *Oak Ridge National Laboratory*, em 1989 [MS94, Gei94a, Gei94b].

PVM é, na verdade, um conjunto de rotinas que efetuam a paralelização de processos, operando em ambientes heterogêneos baseados em UNIX, conectados por diversos tipos de rede.

A comunicação entre os processos é feita por ***message-passing***, ou seja, os processos realizam trocas de mensagens para estabelecer uma interação. Isto força que, no ambiente multiprocessado, cada processador tenha sua própria memória, ou pelo menos o PVM irá enxergá-los desta forma.

O PVM possui dois componentes principais que definem cada operação básica:

- processo PVM;
- bibliotecas (no caso de programação em C, é a libpvm3.a).

As vantagens na utilização do PVM podem ser resumidas da seguinte maneira:

- Reduz o tempo de execução de um programa (*wall clock*);
- Possibilita uma paralelização escalar;
- Relativamente fácil de usar e implementar;
- Software de domínio público;
- Grande aceitação e utilização (é hoje em dia a ferramenta mais usada para a paralelização de programas);
- Flexibilidade:
  - Variedade de arquiteturas;
  - Variedade de redes de trabalho;
  - Programação em C (ou ainda em Fortran);
  - De fácil atualização;
  - Decisão da aplicação, quando e que partes do programa deverão ser executadas em paralelo.

As versões do PVM utilizada foram 3.3.5 e 3.3.7.

Cada agente e memória de soluções foram implementados como processos independentes e executados paralelamente segundo as regras descritas no capítulo 5.

6.9. Instâncias de Testes

Os testes envolveram instâncias do MDTSP com o número de cidades variando de 2 a 10, 50, 100, 200 e 500 cidades. Uma instância real com 24-cidades também foi utilizada.

Para o 2DTSP - MDTSP com 2 matrizes de distância - foram testadas todas as instâncias.

Para o 3DTSP - MDTSP com 3 matrizes de distância - foram testadas as instâncias de 10, 50 e 100-cidades.

6.9.1. Instâncias de Pontos Distribuídos Uniformemente

As instâncias de 2 a 10, 50, 100, 200 e 500-cidades foram produzidas através da distribuição uniforme de *n pontos* (cidades do problema) com as distâncias entre elas estipuladas entre 1 e 100 unidades randomicamente, para todas as matrizes de distância utilizadas.

No Apêndice A pode ser encontrado os Limites Inferiores das matrizes de distância utilizadas nos testes com os Times Assíncronos, bem como as próprias matrizes de 9, 10, 50 e 100 cidades.

6.9.2. Uma Instância Real: O Problema das Principais Cidades Brasileiras

A instância real de 24-cidades foi estruturada através dos dados adquiridos junto às empresas de aviação VARIG - RIO SUL - NORDESTE e consiste na distância entre as 24 principais cidades brasileiras, dentro do contexto de demanda de vôos.

Este é um problema relativamente simples, onde os objetivos que estão envolvidos são custo, qualidade e duração do vôo. O porquê da tentativa de resolução de tal problema é justamente pelo fato de ser uma instância real, onde os objetivos envolvidos não são comparáveis quantitativamente. Deseja-se saber como o A-Team desenvolvido se comporta diante de um caso real do MDTSP.

O problema consiste em determinar as trajetórias de melhor compromisso possível que passam pelas 24 principais cidades brasileiras (em demanda de vôos), onde o melhor compromisso deve ser estabelecido em relação ao custo da viagem e a qualidade do vôo, critérios que serão explicados ainda nesta subseção.

As 24 cidades do problema são apresentadas na tabela 6.4:

Tabela 6. 4: Relação das 24 cidades brasileiras que constituem a instância real do 2-DTSP com 24-cidades.

CÓDIGO	NOME DA CIDADE
AJU	Aracaju
BEL	Belém
BHZ	Belo Horizonte
BSB	Brasília
CGB	Cuiaba
CGR	Campo Grande
CWB	Curitiba
FOR	Fortaleza
FLN	Florianópolis
GYN	Goiânia
IGU	Foz do Iguaçu
JPA	João Pessoa
MAO	Manaus
MCP	Macapá
MCZ	Maceió
NAT	Natal
POA	Porto Alegre
PVH	Porto Velho
REC	Recife
RIO	Rio de Janeiro

SAO	São Paulo
SLZ	São Luiz
SSA	Salvador
VIX	Vitória

Os objetivos estipulados, envolvendo **custo da viagem** (em relação ao passageiro) e **qualidade de vôo** (função envolvendo tipo e capacidade das aeronaves, bem como freqüência de vôos). Um terceiro e quarto objetivo seriam a duração do vôo ou, ainda, a distância da viagem, mas ambos podem ser considerados como sendo diretamente proporcionais ao custo da viagem e, por este motivo, não foram considerados.

1ª Matriz de Distância: Custo da Viagem

Neste caso, é estipulado o custo do vôo entre cada par de cada cidade em relação a todas as outras, em Reais (R$), ressaltando que no momento em que estes valores foram obtidos a moeda brasileira (Real) estava equiparada ao Dólar (US$).

2ª Matriz de Distância: Qualidade de Vôo

Esta matriz de distância foi estipulada com base na frota de aviões existente - segundo a tecnologia envolvida, a capacidade e o tipo de serviço de bordo. Como tais critérios são subjetivos, ou seja, não podem ser medidos quantitativamente, foi necessário a definição de níveis de qualidade. Foram estipulados 12 níveis, com valores pertencentes ao intervalo discreto de 10 a 100, e que tiveram como base as informações apresentadas a seguir.

A tabela 6.5 apresenta os tipos de serviços considerados, onde *F* representa o melhor tipo de serviço num vôo, *Y* representa um serviço normal e *B* àqueles serviços existentes em aviões pequenos, que realizam viagem entre localidades próximas.

Tabela 6. 5: Tabela de tipos de serviços, considerados na definição de uma das matrizes de distância da instância real de 24-cidades.

Doméstico (D)	**Internacional (I)**
F - First Class	F - First Class
Y - Executive Class	Y - Executive Class
B - Serviço Regional	Y - Economy Class

A tabela abaixo apresenta os dados completos, que possibilitaram a definição dos de níveis de qualidade.

Na primeira coluna são listados os modelos das aeronaves e na terceira, seus respectivo códigos. Na segunda coluna, intitulada de *assentos*, são estipulados os serviços disponíveis e a capacidade de passageiros suportada por cada aeronave. Por exemplo, para o modelo BOEING 747-300, o mais utilizado, os dados (D) F16 Y383 (I) F16 C65 Y318 significam que neste modelo estão disponíveis os serviços doméstico (onde a primeira classe possui 16 e a classe econômica possui 383 lugares) e internacional

(onde a primeira classe possui 16, a classe executiva 65 e a classe econômica possui 318 lugares).

Na última coluna encontra-se o nível de qualidade estipulado, que foi baseado nos dados apresentados nas colunas anteriores.

Tabela 6. 6: Tabela de qualidade de serviços, considerados na definição de uma das matrizes de distância da instância real de 24-cidades.

Assentos			Nível de Qualidade
	(I) F18 C35 Y218		10
(D) Y283	(I) F20 C35 Y177		15
(D) F16 Y383	(I) F16 C65 Y318		20
(D) F18 Y176	(I) C18 Y176		30
	(I) F10 C18 Y185		35
(D) Y132	(I) Y132		45
(D) Y109	(I) Y109		50
(D) B111			60
(D) B52			75
(D) B50			80
(D) B30			95
(D) B30			100

Pode ser observado que os níveis de qualidade foram definidos com valores variando de no máximo 10 unidades. Também, quanto melhor a qualidade do vôo, menor é o nível de qualidade associado. Isto para que todas as matrizes pudessem ser *minimizadas*, já que para a matriz que define o custo da viagem, quanto menor o valor, melhor é o resultado.

O que se deseja na resolução deste problema, então, são soluções que possuam os menores valores possíveis considerando a qualidade de vôo (níveis de qualidade) e o custo da viagem.

No Apêndice B encontram-se as matrizes de distância deste problema e seus limites inferiores. No próximo capítulo são apresentados os resultados desta aplicação.

6.10. Resumo

Neste capítulo foi abordado a especificação dos A-Teams desenvolvidos para o *Multi-Distance Traveling Salesman Problem - MDTSP*, bem como os detalhes relevantes de implementação.

A aplicação foi descrita segundo os passos sugeridos na metodologia proposta por Peixoto e Souza [PS94b], com as adaptações necessárias à problemas multiobjetivos, apresentadas no capítulo 5 desta dissertação.

A maioria dos agentes utilizados envolvem os algoritmos disponíveis na literatura sobre TSP, seja através da aplicação direta dos mesmos, seja através de suas adaptações para a manipulação de k matrizes de distância, com a incorporação de conceitos de dominância e decisão de compromisso.

Estratégias envolvendo métricas L_p, a equação que descreve uma hipérbole e a manipulação de camadas de soluções não-dominadas foram elaboradas, para, em conjunto com A-Teams, possibilitar a geração de boas soluções de compromisso para todos os objetivos do problema.

Foram desenvolvidas versões concorrentes e paralelas de A-Teams para o MDTSP. O processamento paralelo foi obtido através da ferramenta PVM (*Parallel Virtual Machine*).

As instâncias de teste envolvem matrizes geradas randomicamente e uma instância real com 24-cidades, denominada de Problema das Principais Cidades Brasileiras.

6.11. Notas Bibliográficas

Sobre estudos de instâncias produzidas artificialmente, pode ser citado Steele [Ste85]. Entre outras coisas, encontram-se técnicas para provar que inúmeras estruturas geométricas apresentam um comportamento similar ao obtido com as instâncias geradas randomicamente com valores pertencentes a uma unidade quadrada.

Sobre o estudo com os algoritmos de melhoria e o aproveitamento de suas soluções intermediárias pode ser encontrado em [RS94].

A parte sobre a utilização da métrica L_2 (distância Euclidiana) em A-Teams, pode ser encontrada em [RS95a, RS95b], além do trabalho desenvolvido por Murthy [Mur92]. E, a parte sobre estratégias para a manipulação de soluções multidimensionais em A-Teams, de um modo geral, pode ser encontrada em [RS96a, RS96b].

Capítulo 7

Resultados Computacionais

" Se - no final das contas - você não puder dizer o que esteve fazendo, é porque tudo o que fez foi inútil. "

E. Schrödinger.

Neste capítulo, são apresentados todos os resultados provenientes dos testes computacionais realizados durante o desenvolvimento deste trabalho, com base nas estratégias de resolução descritas no capítulo anterior.

Foram utilizadas instâncias do *Multi-Distance Traveling Salesman Problem* - MDTSP - geradas aleatoriamente num intervalo entre 1 e 100 (com o número de cidades variando de 2 a 10, 50, 100, 200 e 500) e uma instância real (com 24-cidades), isto envolvendo 2 e 3 matrizes de distância em versões concorrente e paralela.

Uma análise a respeito da variação do número de soluções do Pareto Ótimo e de alguns parâmetros de projeto dos A-Teams em função do aumento do número de cidades e da dimensão do problema é apresentada.

Especificamente sobre os A-Teams desenvolvidos, foram realizadas análises envolvendo tamanho da memória, políticas de destruição, diferentes configurações, desempenho dos algoritmos e tempo de execução, além da análise comparativa entre as diversas estratégias descritas nos Capítulos 5 e 6 desta dissertação.

Para cada uma das instâncias, foram realizadas pelo menos três execuções. E, em todos os testes foi considerada a **versão simétrica** do MDTSP.

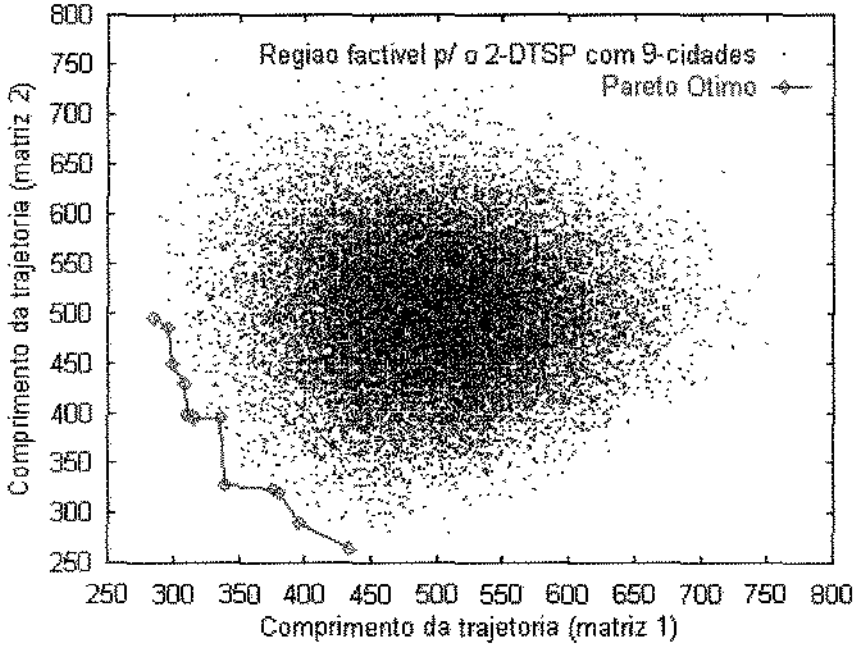
7.1. Estabelecendo o Pareto Ótimo para Pequenas Instâncias do MDTSP

Um algoritmo de enumeração foi utilizado como ferramenta auxiliar para um melhor entendimento das instâncias do MDTSP e para auxílio na análise dos resultados gerados pelo A-Team.

Desta forma, o conjunto exato de soluções pertencentes ao Pareto Ótimo pode ser detectado para instâncias pequenas do MDTSP com até 10-cidades e até 10 dimensões. Para tais instâncias - pequenas - pode-se constatar a proporção entre o número de soluções eficientes e o número total de soluções para o problema (soluções factíveis). O objetivo foi ter uma idéia do que deveria ser esperado em instâncias maiores, considerando que o contorno do Pareto Ótimo seja similar para todas as instâncias, pequenas ou grandes, do MDTSP.

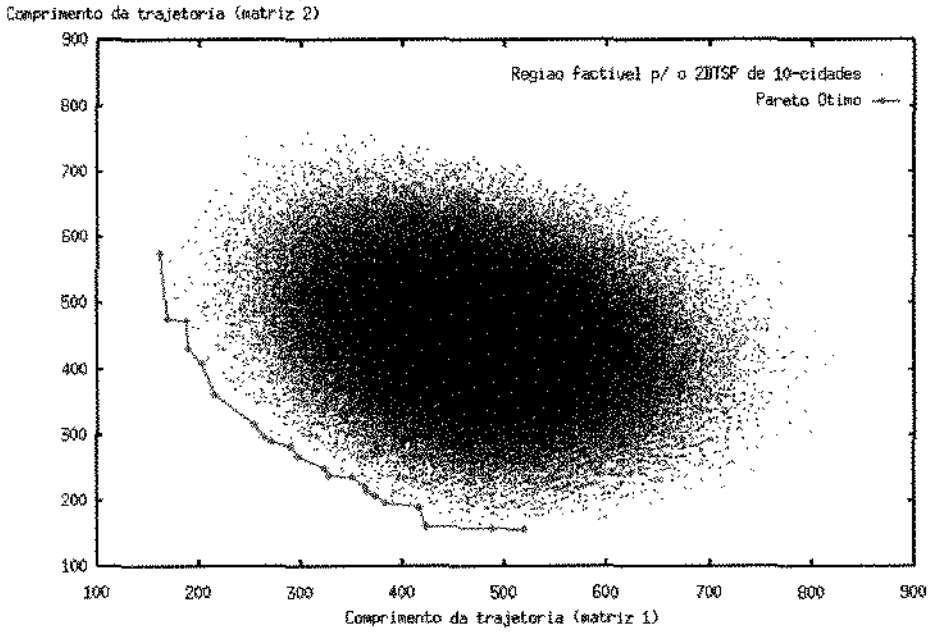
A visualização do número de soluções eficientes em relação ao total de soluções pode ser observado nas figuras 7.1 à 7.3. A figura 7.1 mostra o gráfico de uma instância do 2DTSP (MDTSP com 2 matrizes de distância) com 9-cidades, simétrico, onde estão representadas todas as trajetórias possíveis ((9-1)!/2), perfazendo um total de 20.160 trajetórias (20.160 pontos plotados no gráfico). Apesar da grande quantidade de trajetórias existentes, pode ser observado que o número de trajetórias do Pareto Ótimo é apenas 13 (treze). As sete trajetórias possuem comprimentos diferentes para cada matriz de distância, porém, são equivalentes entre si: nenhuma domina a outra e todas juntas dominam todo o restante de soluções da região factível (20.147 trajetórias restantes).

Figura 7.1: Região Factível e Pareto Ótimo para uma instância do 2DTSP simétrico com 9-cidades. Num total de 20.160 trajetórias válidas, apenas 13 (treze) fazem parte do Pareto Ótimo do problema.



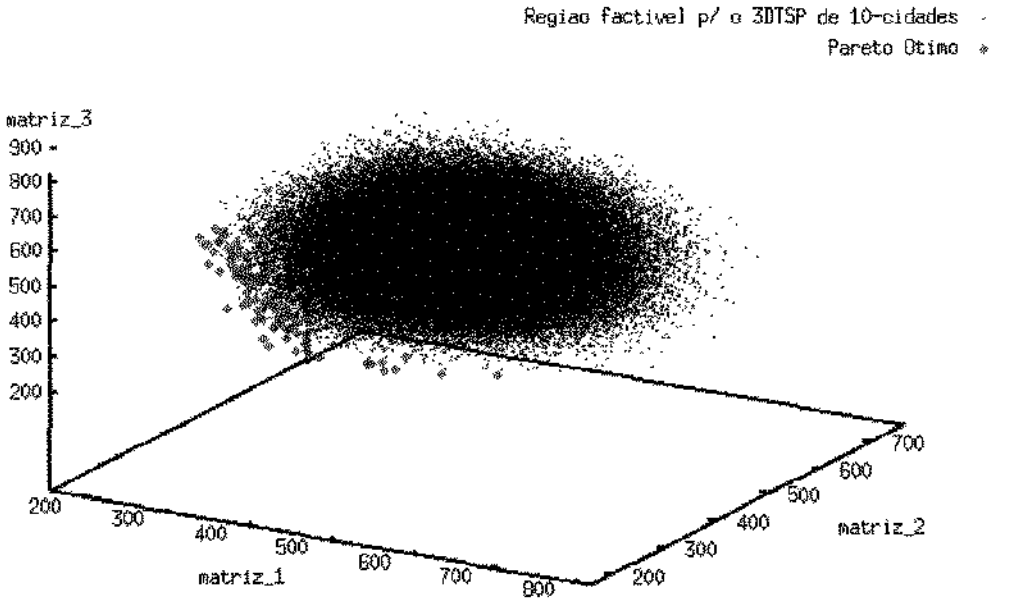
A figura 7.2 apresenta o gráfico de uma instância do 2DTSP com 10-cidades, onde também é mostrada sua região factível completa, com 181.440 trajetórias. Neste caso, o Pareto Ótimo é constituído por 22 trajetórias.

Figura 7. 2: Região factível e o Pareto Ótimo para o 2DTSP simétrico com 10-cidades. Num total de 181.440 trajetórias válidas, apenas 22 constituem o Pareto Ótimo do problema.



A figura 7.3 apresenta o gráfico de uma instância do 3DTSP com 10-cidades, onde pode se observar a mesma região factível, completa, com 181.440 trajetórias. Entretanto, neste caso, o Pareto Ótimo é constituído por 162 trajetórias. A região factível definida pelas variáveis de decisão do problema é a mesma, em qualquer dimensão. A diferença está na região definida pelos objetivos do problema (região factível no espaço de objetivos - ver Cap.3).

Figura 7. 3: Região factível e o Pareto Ótimo para uma instância do 3DTSP com 10-cidades. O Pareto Ótimo é constituído por 162 soluções.



A tabela 7.1 apresenta o número de soluções pertencentes à região factível do MDTSP simétrico, de 2 a 10 cidades, dado por $(n-1!)/2$, onde n é o número de cidades do problema. Vale ressaltar, novamente, que o número de soluções factíveis não varia em função da dimensão do problema.

Tabela 7. 1: Número de soluções da região factível para o MDTSP simétrico, de 2 a 10 cidades.

# de cidades	# de soluções na REGIÃO FACTÍVEL
2	1
3	1
4	3
5	12
6	60
7	360
8	2520
9	20160
10	181440

A tabela 7.2 apresenta o número de soluções pertencentes ao Pareto Ótimo de instâncias do MDTSP entre 2 e 10 cidades (uma única cidade não gera uma trajetória válida para o MDTSP simétrico) variando a dimensão (número de objetivos ou de matrizes de distância do problema) de 1 a 10.

Tabela 7. 2: Número de soluções do Pareto Ótimo variando o tamanho da instância e a dimensão do MDTSP. As colunas representam as dimensões do problema - entre 1D a 10D - e as linhas representam o tamanho da instância - entre 2 e 10-cidades.

# de cidades \ # de dimensões	1D	2D	3D	4D	5D	6D	7D	8D	9D	10D
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1	1	1
4	1	2	3	3	3	3	3	3	3	3
5	1	3	4	5	5	7	8	8	10	10
6	1	6	15	27	47	52	57	59	59	60
7	1	8	19	43	59	123	140	221	223	320
8	1	9	33	82	174	361	569	740	981	1239
9	1	11	50	187	454	862	1917	2216	2672	4509
10	1	17	164	736	1637	3254	8862	15453	18556	24665

Para 2 dimensões foram utilizadas 4 matrizes de distância, combinadas 2 a 2, possibilitando a geração de 6 instâncias para cada tamanho de cidade e a conseqüente obtenção de dados mais precisos. Portanto, o número de soluções pertencentes ao Pareto Ótimo apresentado na tabela 7.2, é resultante da média entre o número de soluções do Pareto Ótimo de cada uma das 6 instâncias. Pode ser observado na tab.7.2, por exemplo, que a média do número de soluções pertencentes ao Pareto Ótimo para o caso de 9-cidades, em 2-dimensões, é de 11 cidades, e não 13, como visto no gráfico da fig.7.1, que é o resultado de apenas uma instância. Para 10-cidades, a média é 17 soluções no Pareto Ótimo, enquanto que no exemplo mostrado na fig.7.2, o Pareto Ótimo é de 22 soluções.

Para 3 dimensões foram utilizadas 4 matrizes de distância, combinadas 3 a 3, possibilitando a geração de 4 instâncias para cada tamanho de cidade. No caso de 10-cidades, a média entre o número de soluções do Pareto Ótimo das quatro instâncias testadas forneceu quase o mesmo número de soluções do exemplo apresentado na figura 7.3. A média foi de 164 soluções, enquanto que o número de soluções da instância apresentada era de 162 soluções pertencentes ao Pareto Ótimo.

Para as demais dimensões, com até 5 cidades, foram utilizadas duas instâncias para cada número de cidade. De 6 a 10 cidades, para dimensões maiores do que três, foi utilizada uma única instância.

As figuras 7.4 e 7.5 apresentam os dados da tabela 7.2 na forma gráfica. Estes dados possibilitam a verificação do rápido crescimento do número de soluções do Pareto Ótimo tanto em função do tamanho da instância do MDTSP (fig.7.4), quanto em função do número de dimensões - matrizes de distância - do problema (fig.7.5).

Apesar do Pareto Ótimo crescer rapidamente, tal crescimento não é proporcional ao crescimento da região factível do problema em relação ao aumento do tamanho da instância, como pode ser observado se compararmos o crescimento dos valores numa linha da tabela 7.2, com o número de soluções válidas para estas respectivas instâncias.

O crescimento do Pareto Ótimo também não é tão rápido quando se varia o número de dimensões do problema, em relação ao crescimento da região factível, que pode ser observado analisando o crescimento dos valores das colunas da tabela em comparação com o número de soluções válidas para as instâncias.

Figura 7. 4: Número de soluções do Pareto Ótimo *versus* o número de dimensões. No intervalo entre 1 e 10 dimensões o crescimento é exponencial no número de dimensões do MDTSP.

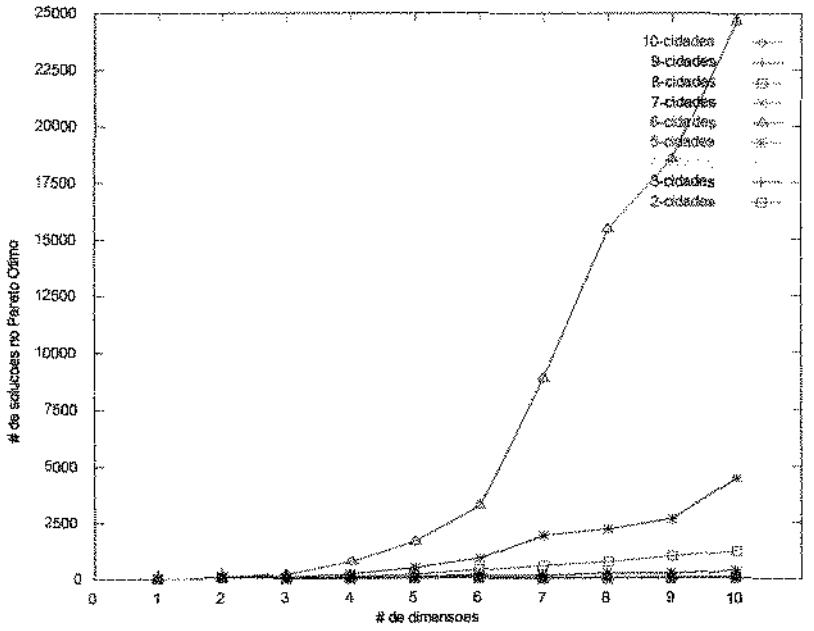
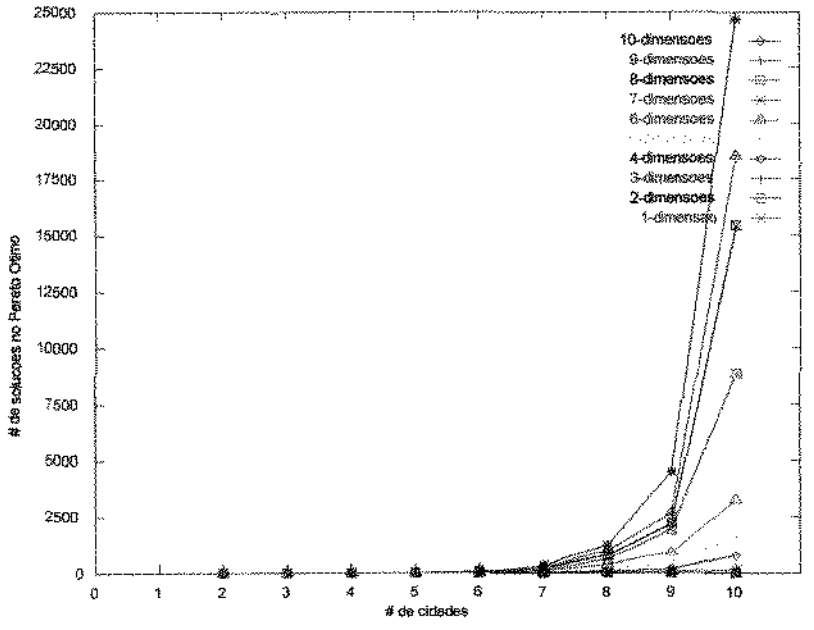


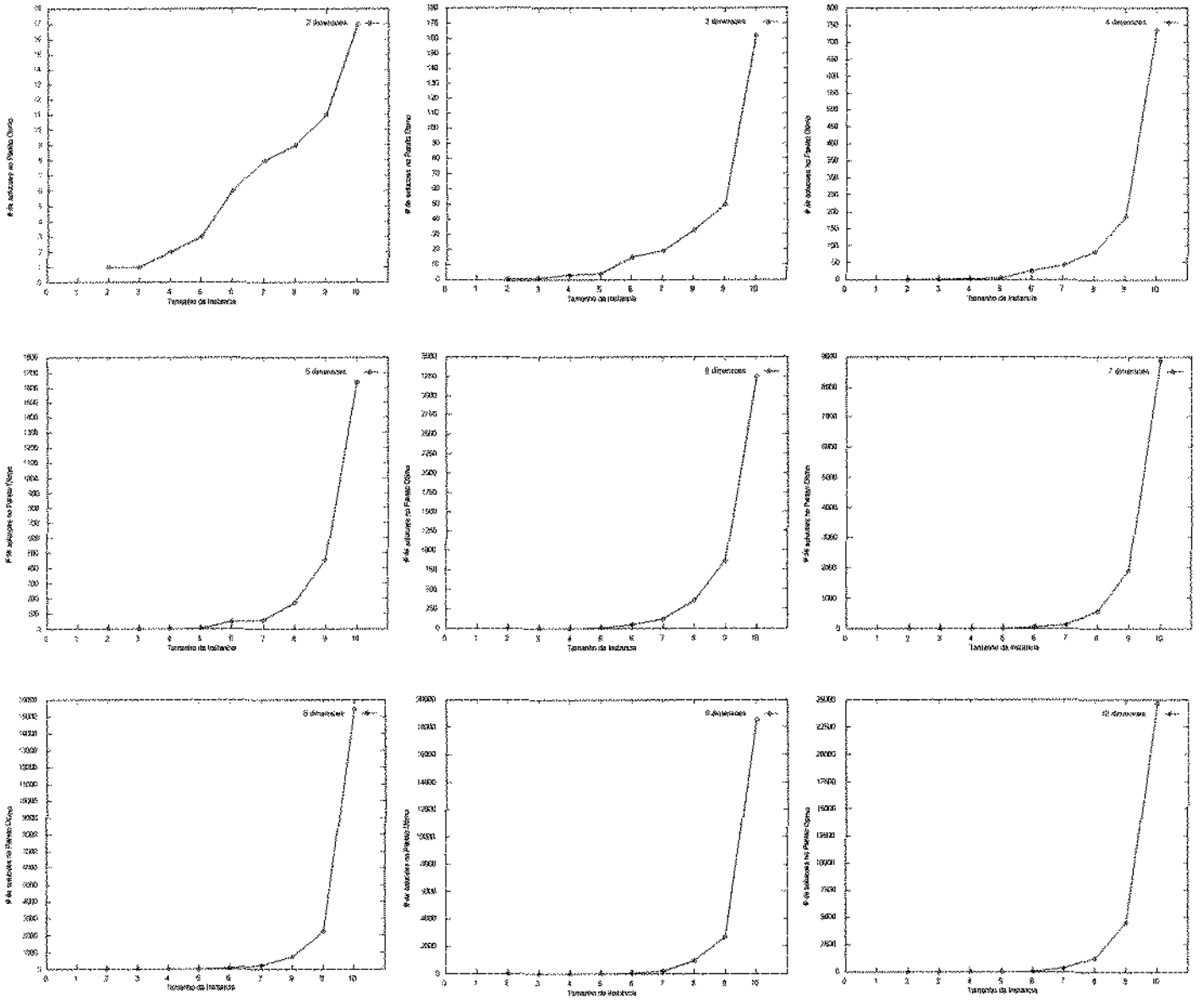
Figura 7. 5: Número de soluções do Pareto Ótimo *versus* o tamanho da instância. No intervalo entre 2 e 10-cidades, o crescimento do Pareto Ótimo é exponencial no número de cidades do MDTSP.



A fig 7.6 apresenta as mesmas curvas apresentadas no gráfico da figura 7.5, mas, agora, cada uma em sua escala natural. Verificamos que os comportamentos das curvas ao se variar apenas a dimensão do problema são idênticos, dando até mesmo a impressão que os gráficos foram repetidos.

A idéia, com base nestes resultados, é conseguir determinar uma função que forneça o mesmo comportamento apresentado para instâncias com até 10-cidades e, em seguida, extrapolar isto para instâncias maiores. Logicamente que nenhuma extrapolação garante que tal comportamento irá continuar para instâncias maiores, mas, fornece uma possível direção para onde o crescimento do Pareto Ótimo poderá seguir.

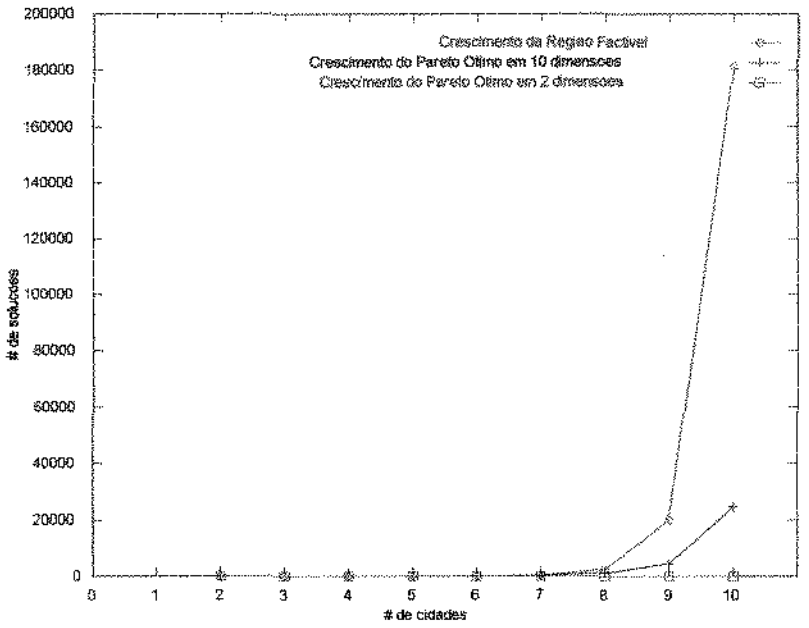
Figura 7. 6: Comportamento do Pareto Ótimo ao se variar o número de dimensões. As curvas apresentadas no gráfico da figura 7.5 são apresentadas individualmente em escala natural, onde percebe-se que o comportamento de todas elas é muito similar.



Para uma análise envolvendo instâncias com mais de 10-cidades, considerou-se que o crescimento do Pareto Ótimo seja pelo menos exponencial, com base nos resultados apresentados anteriormente nesta subseção (pode ser que o crescimento seja representado por um polinômio de alto grau, mas, também, pode ser uma função exponencial do tipo fatorial, por exemplo).

Pode ser observado graficamente na fig. 7.7 a diferença de comportamento entre o crescimento da região factível e o crescimento do Pareto Ótimo de um MDTSP.

Figura 7. 7: Crescimento da região factível *versus* Crescimento do Pareto Ótimo, variando a dimensão do problema.



Observa-se que apesar do crescimento do número de soluções do Pareto Ótimo ser exponencial em função do aumento do número de cidades, este crescimento é encoberto pelo crescimento da região factível do problema, isto para até 10 dimensões e 10 cidades.

Será feita, desta forma, uma extrapolação do crescimento do Pareto Ótimo em função da dimensão e do número de cidades, para grandes instâncias do MDTSP, na ordem de centenas de cidades, que foram as maiores instâncias abordadas pelos A-Teams desenvolvidos.

Assumindo um crescimento que se comporte segundo a equação abaixo (Eq.7.1):

$$\#\Theta_{n,d} = \alpha k_1^n k_2^d \qquad \text{Eq.7. 1}$$

onde:

n - # de cidades;

d - # de dimensões;

α, k_1 *e* k_2 variáveis a serem determinadas.

Selecionando três pontos (verificar tabela 7.1):

P_1 $(\#\Theta_1, n_1, d_1) \rightarrow (1,2,2)$;

P_2 $(\#\Theta_2, n_2, d_2) \rightarrow (5,5,5)$;

P_3 $(\#\Theta_3, n_3, d_3) \rightarrow (24665,10,10)$.

Temos:

$\alpha = 5{,}25$
$k_1 = 1{,}35$
$k_2 = 1{,}70$

Extrapolando para n=500, temos:

d=2: $\#\Theta_{500,2} = \mathbf{2.23e + 066}$

d=10: $\#\Theta_{500,10} = \mathbf{1.55e + 068}$

Pode-se perceber que o número de soluções do Pareto Ótimo é grande o suficiente para se tornar inviável computacionalmente determiná-lo com exatidão. O ideal, ao invés de um sistema que tente calcular exatamente o Pareto Ótimo do problema, seria um sistema que fornecesse soluções uniformemente distribuídas por todo o espectro de soluções do Pareto Ótimo.

Conforme poderá ser verificado no restante deste capítulo, o método de Times Assíncronos fornece soluções abrangendo todo o espectro do provável Pareto Ótimo de instâncias do MDTSP.

7.2. Analisando as Soluções Intermediárias Geradas pelos Algoritmos de Melhoria

No estudo dos algoritmos de melhoria disponíveis para o TSP, na tentativa de adaptá-los para o MDTSP, teve-se a idéia de verificar o comportamento das soluções intermediárias geradas durante cada execução dos mesmos e verificar se elas seriam de melhor compromisso ou não para o MDTSP.

No teste, utilizou-se duas matrizes de distância, isto é, duas dimensões de 200x200 (200-cidades), geradas aleatoriamente num intervalo entre 1 e 100.

Os três gráficos seguintes (fig. 7.8, 7.9 e 7.10), mostram os resultados da aplicação do algoritmo Lin-Kernighan (LK) e do algoritmo 2OPT, com alternância das duas matrizes de distância, tal como descrito a seguir.

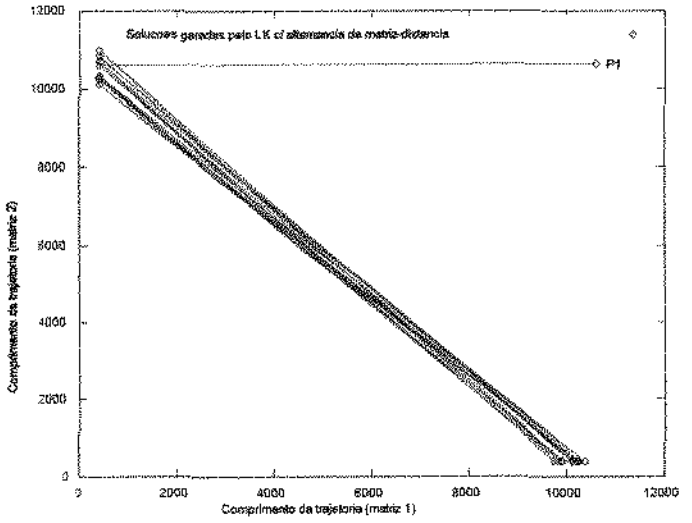
A primeira trajetória foi gerada aleatoriamente e o seu comprimento calculado segundo a matriz de distância_1 e segundo a matriz de distância_2. Este foi, então, o primeiro ponto P_1 a ser plotado (fig.7.8), cujos comprimentos são considerados ruins em ambos os eixos (valores altos nos dois comprimentos da trajetória).

A partir daí, o algoritmo de melhoria Lin-Kernighan foi sucessivamente aplicado, sempre com alternância da matriz de distância, melhorando bruscamente o comprimento

da trajetória inicial que lhe era fornecido. Este fato pode ser melhor entendido através do gráfico contido na fig.7.8, onde as semi-retas diagonais aos eixos x e y foram formadas justamente pela melhora brusca dos valores, a cada execução do LK (quando melhorava bruscamente segundo determinada matriz de distância, seu valor segundo a outra matriz piorava muito, gerando-se pontos diagonalmente opostos).

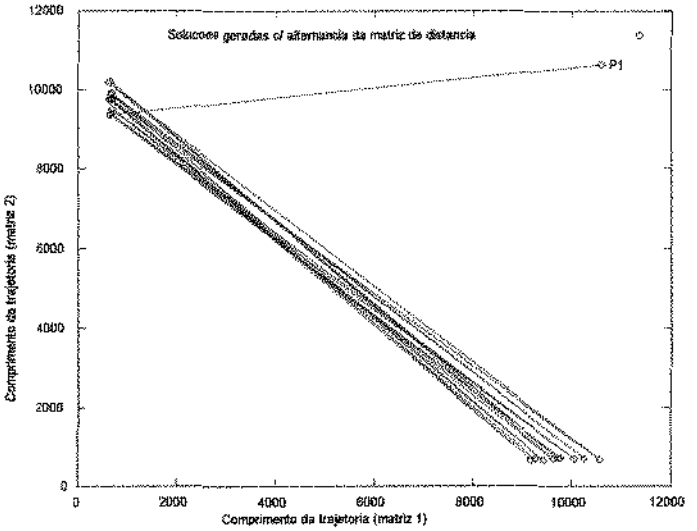
Observa-se que quando o algoritmo LK é aplicado à trajetória inicial P_1 (cujos dois comprimentos segundo as duas matrizes de distância são muito altos - 10591 e 10639, respectivamente), ele melhora significativamente o valor segundo uma das matrizes (de 10591 diminui para 397), em compensação, o valor segundo a outra matriz de distância permanece muito alto (de 10639 foi para 10603). Quando o LK é aplicado novamente, desta vez segundo a outra matriz de distância (valor que permaneceu elevado), o valor segundo a mesma sofre uma brusca diminuição (de 10603 diminui para 403) e o valor segundo a primeira matriz, que havia sido diminuído volta a se tornar elevado (de 397 aumenta para 10104). E assim sucessivamente, formando a estrutura apresentada na figura 7.8.

Figura 7. 8: Aplicação do algoritmo de melhoria Lin-Kernighan para uma instância do 2DTSP de 200-cidades, com alternância das 2 matrizes de distância envolvidas.



A mesma experiência foi realizada com o algoritmo 2OPT e o resultado pode ser observado na fig.7.9. A trajetória inicial foi a mesma fornecida para o LK, com comprimentos iniciais 10591 e 10639, segundo a matriz de distância 1 e 2, respectivamente. Após a aplicação do 2OPT, segundo a matriz de distância 1, o comprimento 10591 diminuiu para 656 e o comprimento 10639 foi para 9363. Aplicou-se novamente o 2OPT na trajetória com comprimentos 656 e 9363, agora segundo a matriz de distância 2, e o resultado foi uma trajetória com comprimentos 9774 e 731. E assim sucessivamente, como apresentado no gráfico da figura 7.9.

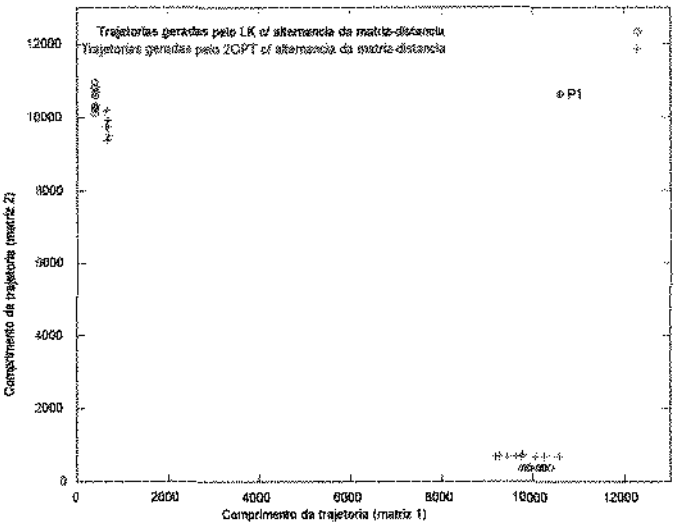
Figura 7. 9: Aplicação do algoritmo de melhoria 2OPT para uma instância do 2DTSP de 200-cidades, com alternância das 2 matrizes de distância envolvidas.



Nota-se que ele também melhora bruscamente a solução gerada aleatoriamente, mas, esta melhora não é tão boa quanto a fornecida pelo Lin-Kernighan.

A figura 7.10 reúne as trajetórias geradas pelo LK e 2OPT, onde pode ser observado mais claramente que o LK conseguiu trajetórias finais com valores mais baixos, ressaltando que ambos partiram do mesmo ponto inicial (a trajetória inicial de entrada era a mesma). O menor comprimento segundo a matriz de distância 1 foi obtido pelo LK que gerou a trajetória com comprimentos 398 e 10830, ao passo que o 2OPT gerou a trajetória com comprimentos 640 e 9756. Segundo a matriz de distância 2, o LK também gerou a trajetória de menor comprimento 400 (9921 para a matriz de distância 1), ao passo que o 2OPT gerou a trajetória de comprimento 689 (9172 para a matriz de distância 1).

Figura 7. 10: Trajetórias geradas pelo LK e 2OPT com alternância de duas matrizes de distância.



Depois de serem obtidos tais dados, onde a diferença entre os comprimentos de uma *mesma* trajetória são *completamente* diferentes quando se considera diferentes matrizes de distância, se pensou no que acontece no processamento interno de tais algoritmos:

Será que as soluções intermediárias que os algoritmos de melhoria geravam não seriam eficientes em relação à solução final?

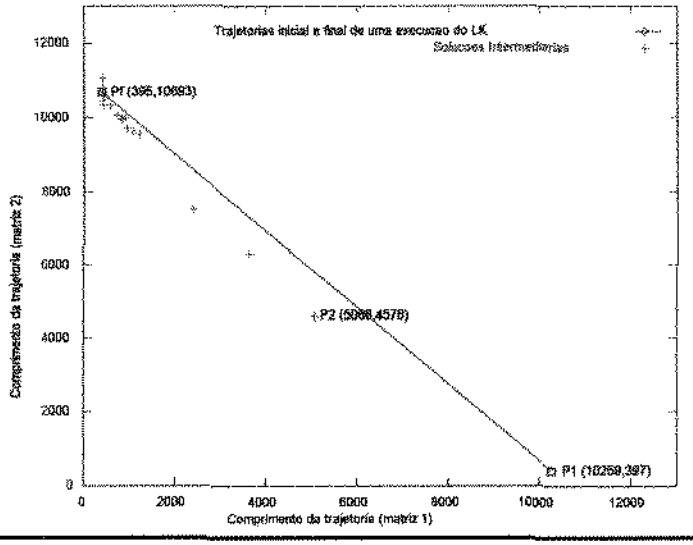
Desta forma, decidiu-se implementar a idéia, já descrita anteriormente (ver Cap.4), de modificar os algoritmos de melhoria e aproveitar as soluções intermediárias que os algoritmos originais simplesmente desprezavam.

O resultado prático demonstrou o que se presumia intuitivamente. As soluções intermediárias, como pode ser comprovado no gráfico da figura 7.11, são em sua maioria não-dominadas em relação à solução final do algoritmo.

Estas *soluções intermediárias* ***tendem*** para a direção do **Pareto Ótimo** do problema, deixando a convicção de que muito ainda poderia ser explorado deste fato.

Na fig.7.11 é apresentado um gráfico com uma execução do Lin-Kernighan, onde o mesmo já está modificado para gerar *todas* as soluções intermediárias, com base na matriz de distância 1. A instância e todos os demais dados são os mesmos dos gráficos anteriores desta subseção. Observa-se que as soluções intermediárias geradas *tendem* para a direção do Pareto Ótimo. À medida que o comprimento das trajetórias segundo a matriz 1 vai diminuindo, o comprimento segundo a matriz 2 vai aumentando. Os comprimentos segundo a matriz base (matriz de distância 1) são sempre monotonicamente decrescentes, entretanto, apesar do comprimento segundo a outra matriz geralmente aumentar, não é garantido um aumento monotônico de seus valores. Observa-se, ainda, que as primeiras soluções intermediárias geradas pelo LK possuem comprimentos consideravelmente distantes uns dos outros, mas, logo em seguida o LK, como melhorou muito os valores logo de início, gera trajetórias com comprimentos muito próximos.

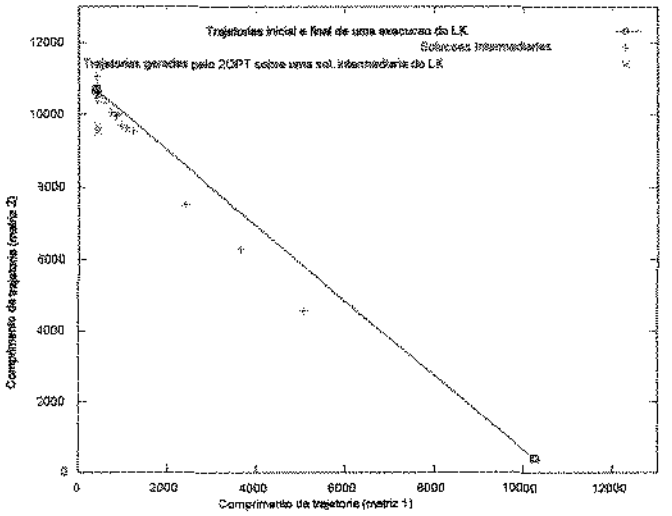
Figura 7.11: Soluções intermediárias geradas durante uma execução do algoritmo Lin-Kernighan. O LK recebe como trajetória inicial P1 a solução de comprimentos (10259,397) e, em seguida, gera a trajetória intermediária P2 de comprimentos (5068,4578) e assim sucessivamente até gerar a trajetória final Pf de comprimentos (395,10693).



A partir disto, os estudos se encaminharam para a exploração da vizinhança (soluções similares) de cada solução intermediária, com a alternância das matrizes de distância e dos algoritmos a serem aplicados. O objetivo foi simular a evolução de tais soluções, tal como se fosse um A-Team, mas apenas com algumas interações entre os algoritmos utilizados.

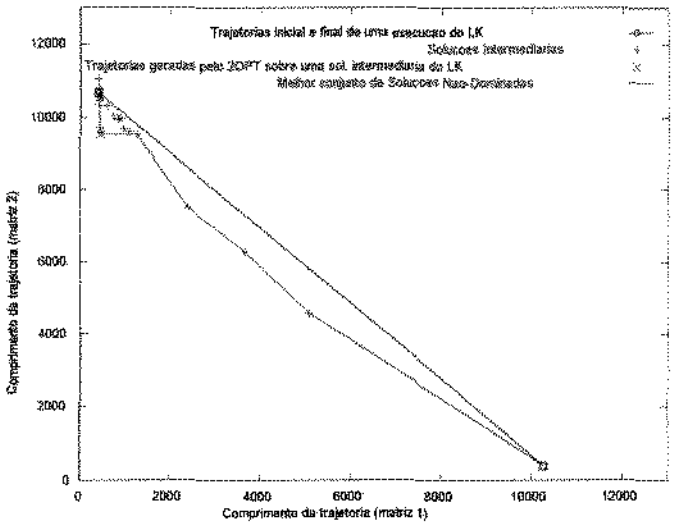
A figura 7.12 apresenta o resultado da aplicação do algoritmo 2OPT, que recebeu como trajetória inicial uma das soluções intermediárias geradas pelo LK, tal como mostrado na figura anterior (fig.7.12). Pode ser observado que o 2OPT não consegue melhorar muito o comprimento da trajetória inicial (segundo a matriz de distância 1), tanto que a trajetória final do 2OPT possui os comprimentos 433 e 9667, enquanto que os comprimentos da trajetória final do LK foram 395 e 10693.

Figura 7. 12: Aplicação do algoritmo 2OPT sobre uma das trajetórias intermediárias geradas durante uma execução do LK.



Apesar do 2OPT não conseguir melhorar muito os comprimentos da trajetória inicial, quando lidamos com mais de uma matriz de distância, caso do MDTSP, percebemos que o desempenho do 2OPT sofre uma melhora. A figura 7.13 apresenta o melhor conjunto de soluções não-dominadas obtido da união dos conjuntos de soluções intermediárias geradas pelo LK e 2OPT. O 2OPT, tendo como trajetória inicial uma das soluções intermediárias do LK, gerou apenas duas outras trajetórias. Em compensação, as duas trajetórias geradas por ele dominaram várias trajetórias intermediárias geradas pelo LK e fazem parte do melhor conjunto de soluções não-dominadas deste exemplo.

Figura 7.13: Melhor conjunto de soluções não-dominadas para a união dos conjuntos de soluções intermediárias do LK e do 2OPT. Apesar do 2OPT gerar somente duas trajetórias, as duas fazem parte do melhor conjunto não-dominado.



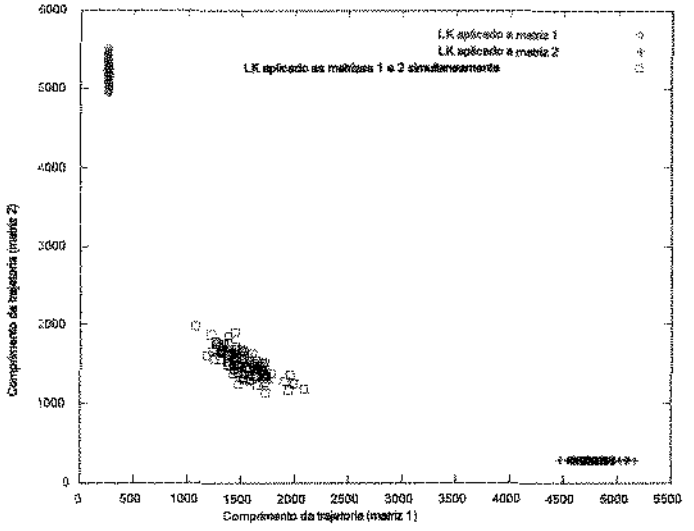
A seguir, serão descritos os resultados da análise dos novos algoritmos desenvolvidos.

7.3. Analisando o Efeito da Incorporação de Dominância e de Decisão de Compromisso nos Algoritmos Desenvolvidos

Os testes realizados com os algoritmos desenvolvidos apresentaram resultados muito favoráveis à resolução de Problemas Multiobjetivos.

A incorporação do conceito de dominância e decisão de compromisso nos algoritmos fez com que os mesmos gerassem soluções de melhor compromisso para o problema multiobjetivo. Tal feito pode ser constatado através do gráfico apresentado na figura 7.14, onde é apresentado o caso do algoritmo LK aplicados à duas matrizes de distância de 100-cidades. Foi aplicado o LK original à matriz 1 e à matriz 2 separadamente, tal como nos primeiros testes realizados, e em seguida aplicou-se o algoritmo LK-D (LK com a incorporação do conceito de dominância) que manipulou as matrizes 1 e 2 simultaneamente para a geração de soluções. Pode-se perceber claramente que as soluções geradas pelo LK-D ocupam a parte central do espectro esperado de melhores soluções, apesar de algumas soluções do LK original pertencerem também ao melhor conjunto de soluções não-dominado que pode ser inferido deste exemplo.

Figura 7. 14: Análise do algoritmo de Lin-Kernighan (LK). Resultado da aplicação do algoritmo LK com aproveitamento das soluções intermediárias e com ou sem a incorporação do conceito de dominância (LK-I x LK-D).



Tal feito aconteceu, também, nos outros algoritmos de melhoria desenvolvidos: variantes do 2OPT, 3OPT e OROPT.

7.4. Testes Computacionais com os A-Teams Desenvolvidos

Nesta seção serão descritos os diversos testes realizados com os A-Teams desenvolvidos.

Os resultados foram obtidos a partir de 6 instâncias do MDTSP geradas randomicamente com 9, 10, 50, 100, 200 e 500 cidades e de 1 instância real com 24 cidades, isto para 2 e 3 dimensões.

Para o 2DTSP - MDTSP com duas matrizes de distância - são reportados os resultados envolvendo todas as 7 instâncias.

Para o 3DTSP - MDTSP com três matrizes de distância - são reportados os resultados envolvendo instâncias com 10 e 100-cidades.

7.4.1. Configurações Utilizadas

Nesta subseção, serão apresentadas todas as configurações utilizadas nos vários testes computacionais com os Times Assíncronos desenvolvidos.

Na figura 7.15 são apresentadas 5 configurações. Todas elas foram aplicadas à instâncias do 2DTSP e somente com algoritmos já existentes para o TSP. Os índices 1 e 2 indicam que os algoritmos possuem uma versão para cada uma das matrizes de distância envolvida.

A **Configuração 1** contém apenas os algoritmos de construção FI (Furthest Insertion) e o de consenso por interseção DEC (Deconstructor), além de duas memórias

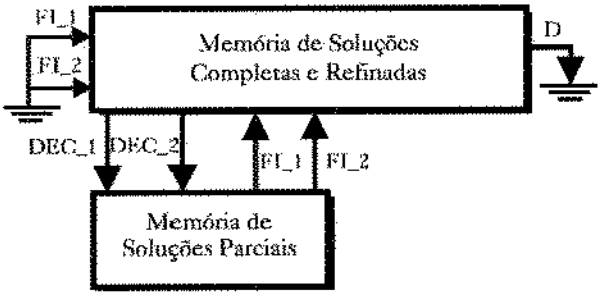
de soluções: Memória de Soluções Completas e Refinadas e Memória de Soluções Parciais. Apenas um fluxo cíclico pode ser estabelecido.

Na **Configuração 2** foi inserido o algoritmo de melhoria 2OPT à configuração anterior. Na **Configuração 3** foi inserido o algoritmo LK. E, na **Configuração 4** foram inseridos o algoritmo 3OPT e uma memória de soluções (Memória de Soluções Completas e Não-Refinadas).

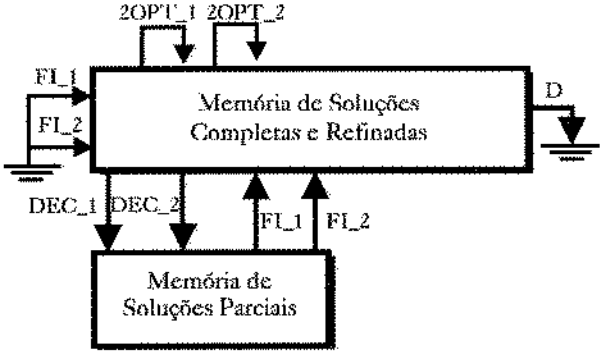
A **Configuração 5** contém apenas os algoritmos de melhoria 2OPT e LK e a Memória de Soluções Completas Refinadas.

Figura 7. 15: Configurações de A-Teams utilizando apenas algoritmos já existem para o TSP. Na Configuração 1 apenas um ciclo no fluxo de dados é estabelecido com um algoritmo de construção FI e com um algoritmo de consenso por interseção DEC. São introduzidos, nesta seqüência, o algoritmo de melhoria 2OPT (configuração 2), o algoritmo 3OPT (configuração 3) e o algoritmo LK (configuração 4). A Configuração 5 contém apenas os algoritmos de melhoria 2OPT e LK.

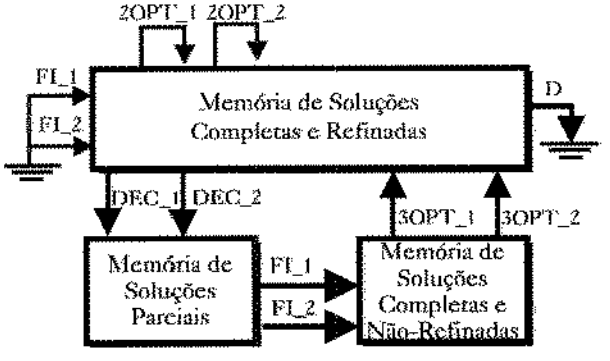
Configuração 1:



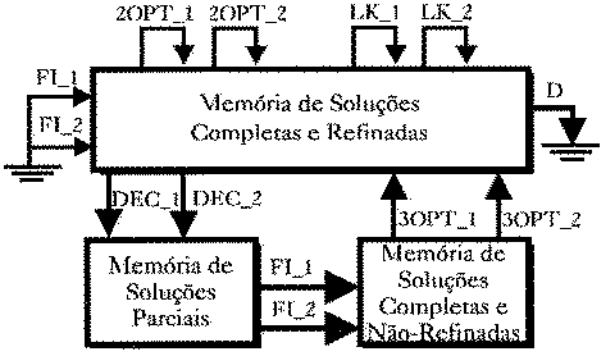
Configuração 2:



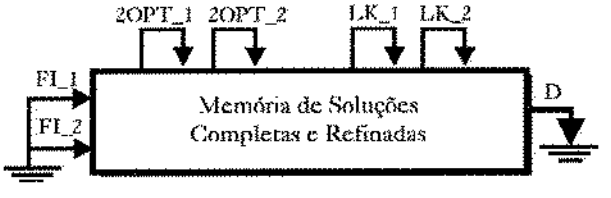
Configuração 3:



Configuração 4:



Configuração 5:



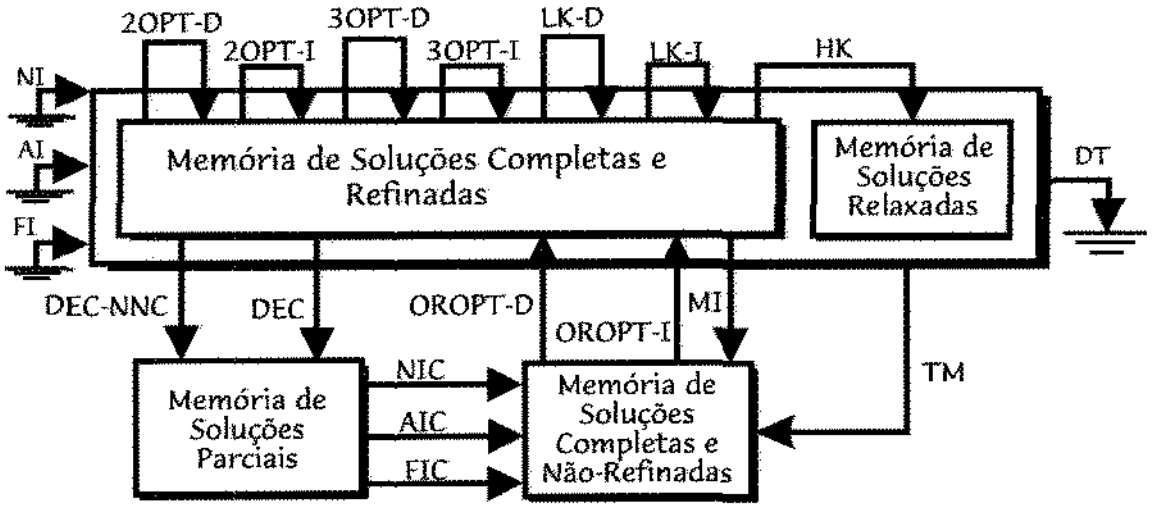
Na figura 7.16 é apresentada a configuração utilizada na maioria dos testes computacionais com os Times Assíncronos: **Configuração 6**. Nela foram agrupados tanto algoritmos aplicados à apenas uma matriz de distância quanto à múltiplas matrizes simultaneamente.

Ao todo são 17 agentes, sendo 8 algoritmos de melhoria, 3 algoritmos de construção, 2 algoritmos de consenso por interseção, 1 algoritmo de consenso por união, 1 algoritmo de relaxação, 1 algoritmo de factibilidade, 1 algoritmo de destruição.

É composta, ainda, por 4 tipos de memórias de soluções: Memória de Soluções Completas e Refinadas; Memória de Soluções Completas e Não-Refinadas; Memória de Soluções Parciais; e, Memória de Soluções Relaxadas.

Figura 7. 16 Configuração contendo algoritmos específicos para o MDTSP e também algoritmos já existentes para o TSP. São ao todo 17 agentes e 4 memórias de soluções. Sendo 8 algoritmos de melhoria, 3 de construção, 2 de consenso por interseção, 1 de consenso por união, 1 de relaxação, 1 e factibilidade e 1 de destruição.

Configuração 6:



As configurações de 1 a 4 foram as utilizadas para a verificação da eficiência em escala de A-Team para o MDTSP.

Os resultados obtidos com a aplicação da **Configuração 6**, tanto para instâncias do 2DTSP quanto para instâncias do 3DTSP, foram os melhores em todas as estratégias testadas.

7.4.2. Analisando o Tamanho das Memórias

Foram realizados testes envolvendo todas as instâncias, até ser determinado um número satisfatório de soluções na memória principal - a Memória de Soluções Completas e Refinadas - durante a execução de um A-Team para o MDTSP, dadas as restrições do ambiente computacional disponível.

Com base nos testes realizados por Souza [Sou93], cujos resultados foram constatados por vários outros [Pei95, Cav95, Lon95, Cam95], e que afirmou que um

bom tamanho de memória seria entre n e $2n$, onde n corresponde ao tamanho da instância, foi estipulado inicialmente um tamanho $m=2n$.

O tamanho exato do Pareto Ótimo para o MDTSP com até 10-cidades e 10-dimensões pôde ser estabelecido exatamente, cujos dados estão reportados na seção 7.2, o que já dá indícios de que 2n não é um tamanho suficiente. Na verdade, se verificou que o crescimento do número de soluções do Pareto Ótimo parece ser exponencial, tanto em função do número de cidades quanto em função do número de dimensões. Desta forma, por maior que seja o tamanho da memória, é impossível estipular um tamanho que comporte todo o Pareto Ótimo. E isto nem mesmo é o desejado. O ideal é um tamanho de memória que possibilite a obtenção de mostras de soluções varrendo todo o espectro esperado do Pareto Ótimo, com soluções com boas características segundo todos os objetivos envolvidos.

Para os testes envolvendo o 2DTSP, o tamanho da memória ficou estipulado em $3n$. Isto devido aos testes preliminares realizados envolvendo memórias com capacidades de $2n$, $3n$, $4n$ e $5n$. As figuras 7.17 e 7.18 apresentam os gráficos com os conjuntos de soluções não-dominadas obtidos para as instâncias de 10 e 100-cidades, respectivamente. Para 10-cidades (fig.7.17), os tamanhos $3n$, $4n$ e $5n$ propiciaram a geração do mesmo conjunto de soluções não-dominadas. Isto porque tal instância é muito pequena e a convergência do A-Team é muito rápida (em poucos segundos). Apesar disto, os tamanhos n e $2n$ já se mostram inadequados. Para 100-cidades (fig.7.18), o tamanho $3n$ propiciou a geração do melhor conjunto não-dominado. Na verdade, os resultados envolvendo os tamanhos $3n$ e $4n$ foram sempre muito similares e como ocorre uma degradação do desempenho do A-Team, no tempo de processamento, quando a memória aumenta, estipulou-se o tamanho da memória em $3n$.

Figura 7.17 : Análise do tamanho de memória para o 2DTSP com 10-cidades. Neste caso, como a instância é muito pequena, a memória com tamanhos $3n$, $4n$ e $5n$ propiciaram o mesmo conjunto de soluções não-dominadas. Mesmo neste caso, já pode ser constatado que os tamanho n e $2n$ são inadequados.

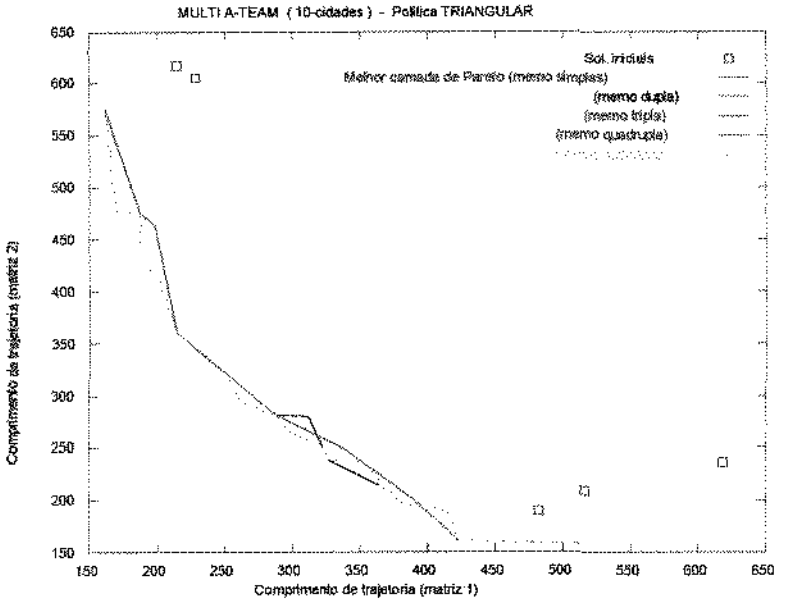
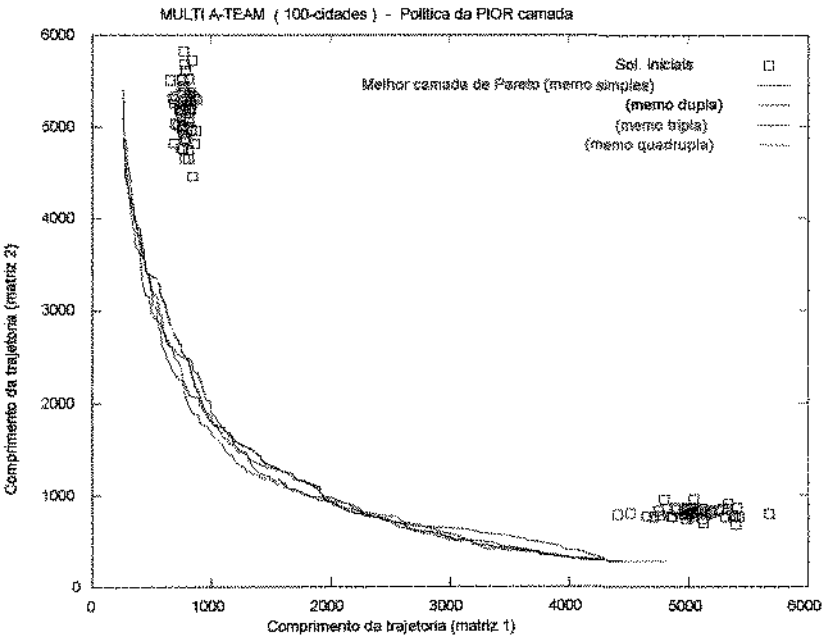


Figura 7.18: Análise do tamanho de memória para o 2DTSP com 100-cidades. Neste caso, a memória com tamanho $3n$ foi a que propiciou o melhor conjunto não-dominado.



Já para os testes envolvendo o 3DTSP, os melhores conjuntos não-dominados foram obtidos para um tamanho de memória $m=5n$, dentre o intervalo $n \leq m \leq 5n$. Talvez, uma memória com capacidade superior a $5n$ permitisse uma maior eficiência no desempenho do A-Team para grandes instâncias, entretanto, devido a limitação do ambiente computacional utilizado nos testes (alta demanda de usuários, principalmente), tais experimentos não puderam ser realizados.

7.4.3. Eficiência em Escala

O conceito de eficiência em escala foi introduzido por Talukdar e Souza [TS92], mostrando A-Teams como uma organização eficiente em escala (ver Cap.5). Resumidamente, a eficiência em escala é constatada se a inclusão de um agente no time proporciona uma melhora no desempenho do A-Team. Isto não implica que inexista uma ordem que não ocasione uma melhora monotônica no desempenho do A-Team, e sim, na existência de pelo menos uma ordem em que se consiga uma melhora monotonicamente crescente no seu desempenho.

Os testes para a verificação da eficiência em escala foram realizados sobre as quatro primeiras configurações apresentadas na fig. 7.15: **Configuração 1**; **Configuração 2**; **Configuração 3**; e, **Configuração 4**.

As instâncias utilizadas foram de 100-cidades e de 9-cidades, para o A-Team c/ métrica-L_2 (Distância Euclidiana), sem manter o melhor conjunto não-dominado na memória do A-Team em tempo de execução. Em todas as estruturas utilizadas nos testes desta seção o agente que inicia o preenchimento da memória com soluções válidas é o FI (Furthest Insertion). Na primeira configuração, apenas um ciclo é estabelecido. Os algoritmos utilizados foram somente dois: o algoritmo de construção FI (uma versão para

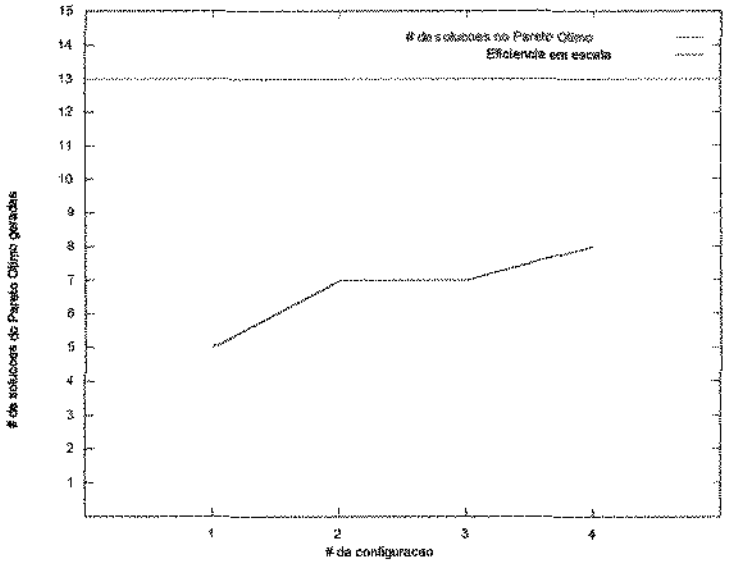
cada matriz de distância do problema) e o algoritmo de consenso por interseção DEC (Deconstructor). Na segunda configuração, foi inserido o algoritmo de melhoria 2OPT. Na terceira configuração, foi inserido o algoritmo de melhoria 3OPT e uma Memória de Soluções Completas e Não-Refinadas. Po fim, na quarta configuração, foi inserido o algoritmo de melhoria LK.

Os resultados obtidos, comprovando a eficiência em escala, foram melhores à medida que o número de algoritmos foi aumentando, conseguindo-se os melhores resultados com a utilização da configuração 4.

Uma configuração adicional foi testada, envolvendo apenas algoritmos de melhoria, já que individualmente o desempenho dos mesmos é muito superior ao desempenho das demais classes de algoritmos. Os resultados obtidos com o A-Team contendo apenas os algoritmos de melhoria (Configuração 5) foram inferiores aos obtidos pelas Configurações 3 e 4. Isto atesta que a sinergia de todos os algoritmos do time é maior do que a de um algoritmo ou subgrupo destes algoritmos quando executados isoladamente.

A figura 7.19 mostra o número de soluções que pertencem ao Pareto Ótimo (13 soluções ao todo) para a instância de 9-cidades e o número de soluções do Pareto Ótimo obtida em cada uma das configurações do A-Team. Quanto mais algoritmos vão sendo inseridos, maior é o número de soluções do Pareto Ótimo alcançadas: para a Configuração 1, obteve-se um subconjunto do Pareto ótimo com 5 soluções; para as Configurações 2 e 3, obteve-se um subconjunto de 7 soluções; e, para a Configuração 4, obteve-se um subconjunto do Pareto ótimo com 8 soluções, somente 5 a menos do que o total de soluções do Pareto Ótimo. O número total de soluções do Pareto Ótimo para a instância de 9-cidades foi obtido através da aplicação de um algoritmo enumerativo, como descrito na seção 7.1.

Figura 7. 19: Eficiência em escala do A-Team para o MDTSP. Comparação do número de soluções não-dominadas geradas pelo A-Team em cada uma das 4 configurações, para a instância de 9-cidades.



Os resultados obtidos para o A-Teams com 100-cidades utilizando a métrica-L_2 (sem manter o melhor conjunto não-dominado), e que comprovam a eficiência em escala são apresentados na tabela 7.3.

Vale ressaltar que os índices de pertinência apresentados são relativos ao conjunto $\hat{\Theta}$, que é resultante da união dos conjuntos não-dominados fornecidos em cada configuração (quatro conjuntos ao todo). Apesar dos índices de pertinência dos conjuntos não-dominados gerados através do A-Team com Distância Euclidiana serem baixos, ainda assim, o propósito é alcançado. Ocorre uma nítida melhora no índice de pertinência, a medida em que novos algoritmos são inseridos no time.

Tabela 7.3: Eficiência em escala utilizando distância Euclidiana.Tabela com os índices de pertinência obtidos para as instâncias de 9 e 100 cidades utilizando distância Euclidiana - sem manter o melhor conjunto não-dominado.

	Índice de Pertinência (%)	
# da Configuração	9-cidades	100-cidades
1	31,6	19,0
2	48,3	26,4
3	59,8	41,2
4	66,6	50,9

Tal experimento foi realizado, também, com a estratégia de Camadas de Soluções Não-Dominadas e, novamente, foi constatada a eficiência em escala de A-Teams para o MDTSP. Para as instâncias de 9 e 10 cidades, o A-Teams encontrou 100% das soluções pertencentes ao Pareto Ótimo. Na tabela 7.4, pode ser observado que para a Configurações 2, 3 e 4, o índice de pertinência para 9 e 10 cidades é de 100%, com o asterisco indicando que se trata do próprio Pareto Ótimo do problema. Entre parênteses está o número de soluções do Pareto Ótimo, que no caso de 9-cidades é de 13 soluções e de 10-cidades, 22 soluções.

Tabela 7.4: Eficiência em escala utilizando Camadas de Conjuntos Não-Dominados. Tabela com os índices de pertinência obtidos para as instâncias de 9, 10 e 100 cidades utilizando Camadas de Conjuntos Não-Dominados.

	Índice de Pertinência (%)		
# da Configuração	9-cidades	10-cidades	100-cidades
1	87	83	49
2	100* (13)	100* (22)	74
3	100* (13)	100* (22)	82
4	100* (13)	100* (22)	88

Os índices de pertinência dos conjuntos não-dominados gerados pelo A-Team através da estratégia de Camadas de Soluções Não-Dominadas, em todas as configurações, são bem superiores aos gerados pelo A-Team com Distância Euclidiana. Vale ressaltar, entretanto, que não se pode afirmar qual é a melhor estratégia com base em tais dados, já que o conjunto $\hat{\Theta}$ foi gerado com base nos conjuntos não-dominados de *cada* estratégia, isoladamente.

Uma comparação entre a estratégia de Camadas de Soluções Não-Dominadas e as demais estratégias é apresentada no final deste capítulo.

7.4.4. Analisando as Políticas de Destruição

Para o teste com as políticas de destruição, utilizou-se o A-Team implementado com a métrica-L_2, para a instância do 2DTSP (2 matrizes de distância) com 100-cidades. Esta estratégia foi a escolhida para esta análise porque pode-se perceber nitidamente, com o uso da métrica-L_2 - *sem* manter o melhor conjunto de soluções não-dominadas na memória durante a execução do A-Team -, o desempenho e a característica de cada uma das políticas de destruição adotadas.

A fig.7.20 apresenta o desempenho das 4 políticas de destruição: DP, DU, DL e DT. Observe que cada conjunto não-dominado final apresenta um formato diferente, caracterizando a respectiva política de destruição aplicada.

No caso da política da pior solução (DP), observa-se que as soluções finais ficaram aglomeradas no que seria a parte central de um bom conjunto não-dominado, ou seja, as soluções finais e conseqüentemente o melhor conjunto de soluções não-dominadas não incorporam as melhores soluções segundo cada matriz de distância, que também fazem parte do melhor conjunto não-dominado.

No caso da política uniforme (DU), percebe-se um comportamento oposto, onde o formato do melhor conjunto não-dominado encontrado, apesar de incorporar soluções dentro da faixa esperada de um bom conjunto não-dominado, agrupa a maioria das soluções nas extremidades do conjunto não-dominado e, na parte central do mesmo, poucas soluções foram obtidas.

O ideal seria uma política que reunisse as características das políticas DP e DU.

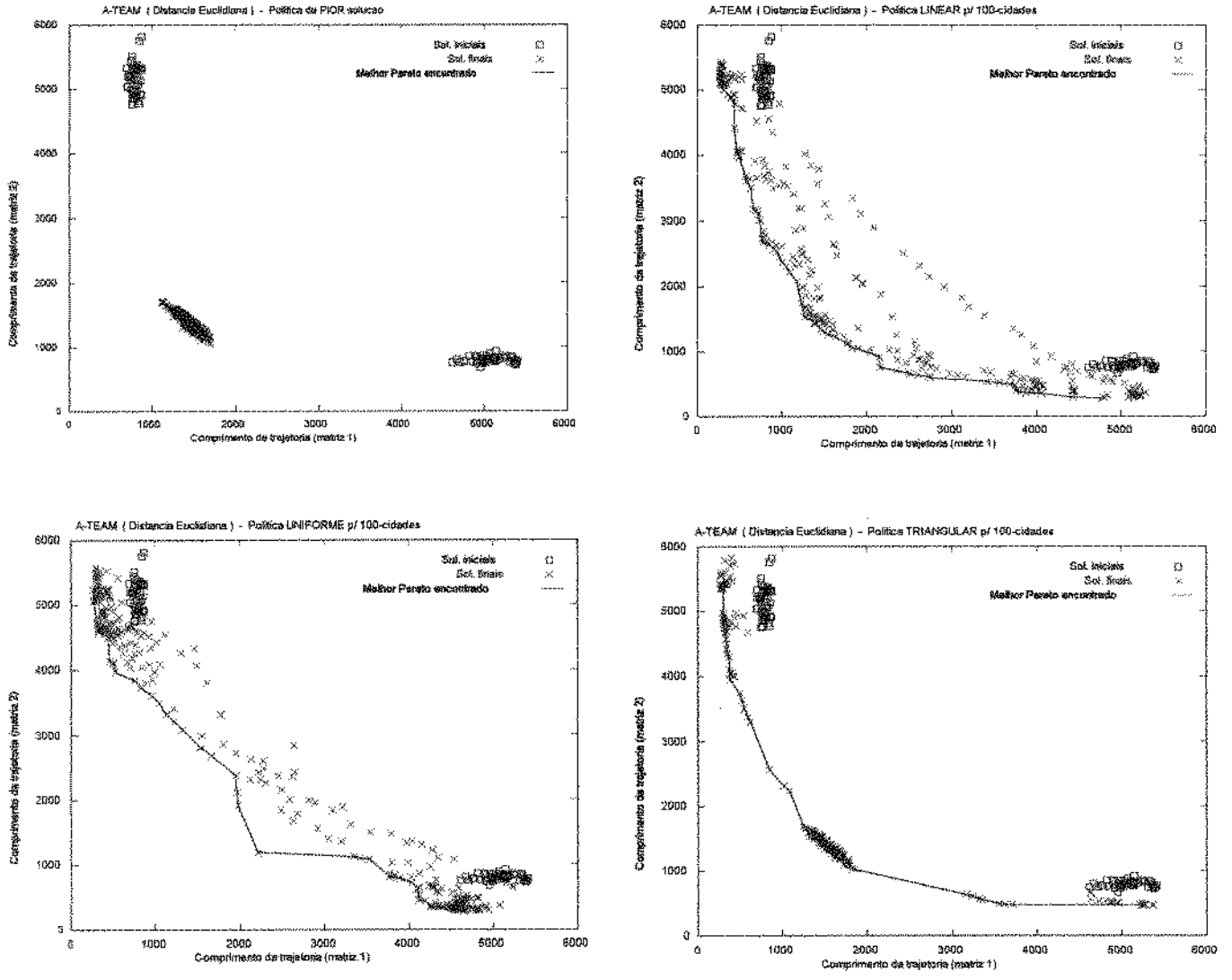
Com a aplicação da política linear, percebe-se que o melhor conjunto não-dominado obtido reúne as características desejadas, contendo soluções que abrangem todo o espectro desejado, ou melhor, as solução não se aglomeram numa única região do espaço factível de objetivos, e sim, se encontram espalhadas numa ampla faixa de valores envolvendo tanto soluções boas segundo cada uma das matrizes envolvidas quanto as que estabelecem os melhores compromisso para as duas.

Na aplicação da política triangular (DT), percebe-se que o formato do conjunto de soluções não-dominados, apesar de abranger todo o espectro desejado, concentra soluções na parte central do conjunto não-dominado e, também, nas extremidades dos mesmos, restando apenas algumas soluções nos intervalos entre a parte central e as duas extremidades.

Baseando-se na análise acima, uma possível conclusão seria a de que a melhor política seria a DL, acontece que reunindo-se os dois conjuntos não-dominados - os

obtidos por DL e DT - e gerando um novo a partir dos dois (o conjunto $\hat{\Theta}$), constata-se que o conjunto não-dominado gerado por DT tem um grau de pertinência de 62% enquanto que o gerado por DL tem apenas 44%. Isto significa dizer que apesar do formato do melhor conjunto não-dominado gerado por DL ser mais próximo do desejado, a aplicação da política triangular gerou as melhores soluções não-dominadas dentre as quatro políticas testadas, isto para o 2DTSP com 100-cidades utilizando a estratégia da métrica Euclidiana.

Figura 7. 20 Desempenho das políticas de destruição DP, DU, DL e DT para o 2DTSP de 100-cidades.



7.4.5. Times Assíncronos com Distância Euclidiana

Um dos objetivos deste trabalho era apresentar estruturas de manipulação de soluções multidimensionais em A-Teams mais adequadas a problemas multiobjetivos do que a utilizada por Murthy [Mur92].

Murthy aplicou A-Teams a um problema de design (ver Cap.5) - que são problemas inerentemente multiobjetivos. Entretanto, a abordagem utilizada por ele foi a monobjetiva, em que os agentes de destruição eliminavam soluções da memória com as maiores distâncias Euclidianas em relação ao melhor conjunto não-dominado contido na memória durante a execução do A-Team.

No primeiro A-Team para o MDTSP desenvolvido, aplicou-se a métrica-L_2 com o intuito de verificar o comportamento da mesma estratégia desenvolvida por Murthy, desta feita para o MDTSP, e fazer uso de tal estratégia como base de comparação com as outras estratégias que seriam posteriormente desenvolvidas.

A métrica-L_2 foi utilizada para organizar e eliminar soluções não promissoras da Memória de Soluções Completas e Refinadas, beneficiando todos os agentes envolvidos.

Para a instância de 9-cidades e de 100-cidades foram aplicadas as primeiras quatro configurações (fig.7.15). A Configuração 1 foi composta pelos algoritmos DEC (Deconstructor), FI (Furthest Insertion) e o Destroyer DP (destruição da pior solução contida na memória). Todas as três configurações posteriores são versões mais complexas das anteriores, ou seja, novos algoritmos vão sendo incorporados, cada vez mais, aos já existentes. Desta forma, para formar a configuração 2 foi adicionado o algoritmo 2OPT, para a configuração 3 foi adicionado o algoritmo 3OPT e para a configuração 4 foi adicionado o algoritmo Lin-Kernighan. Para todas as outras instâncias, foi testada somente a configuração 4 (fig.7.15).

Nota-se na fig.7.15, que a Configuração 4 do A-Team (a mais sofisticada destes testes) possui 3 memórias e 6 agentes (Furthest Insertion - FI, Deconstructor - DEC, 2OPT, 3OPT, Lin-Kernighan - LK e Destroyer DP). Cada um dos algoritmos sendo aplicados, aleatoriamente, para uma determinada matriz de distância. Por exemplo, 2OPT_1 corresponde ao algoritmo 2OPT segundo a matriz 1, 2OPT_2 segundo a matriz 2 e assim para todos os outros algoritmos (a exceção fica por conta do Destroyer D, que não se baseia diretamente nas matrizes de custo e, sim, nas soluções que vão sendo geradas).

Inicialmente, a Memória de Soluções Completas e Refinadas foi preenchida com soluções geradas pelo algoritmo FI e depois os algoritmos vão sendo executados, sempre aleatoriamente, até a condição de parada estipulada; foi estipulado um determinado tempo de execução que varia conforme o tamanho da instância. Por exemplo, para a instância de 9 cidades, obteve-se convergência em questão de segundos e, para a instância de 500 cidades, em duas dimensões, o tempo máximo de execução ficou em mais de dez horas.

As figuras 7.21, 7.22, 7.23, 7.24 e 7.25, apresentam o melhor conjunto não-dominado encontrado para as instâncias de 9, 50, 100, 200 e 500 cidades, respectivamente, usando a Configuração 4. Os dois conjuntos de pontos localizados próximos aos valores mais altos dos eixos representam as soluções iniciais geradas pelo

algoritmo Furthest Insertion (soluções iniciais). Cada eixo representa o comprimento da trajetória de uma solução com relação às duas matrizes de distância.

Figura 7. 21: O melhor conjunto de soluções não-dominadas gerado pelo A-Team c/ Distância Euclidiana, para a instância de 9-cidades.

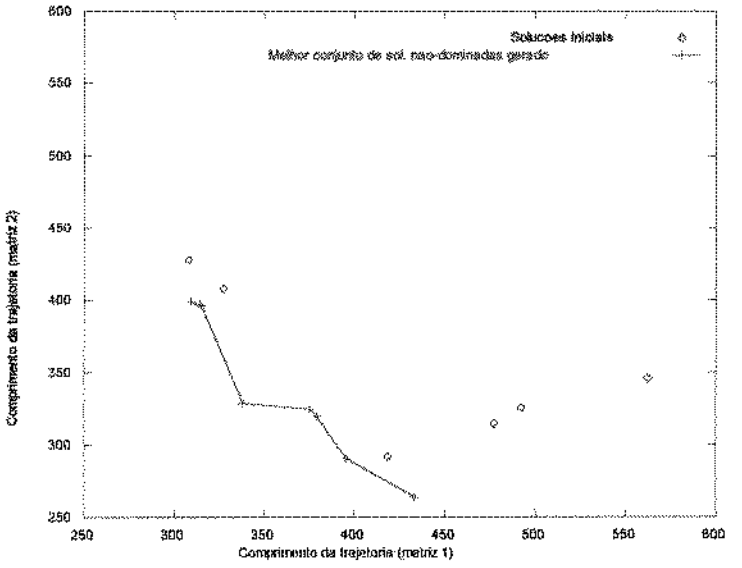


Figura 7. 22: O melhor conjunto de soluções não-dominadas gerado pelo A-Team c/ Distância Euclidiana, para a instância de 50-cidades.

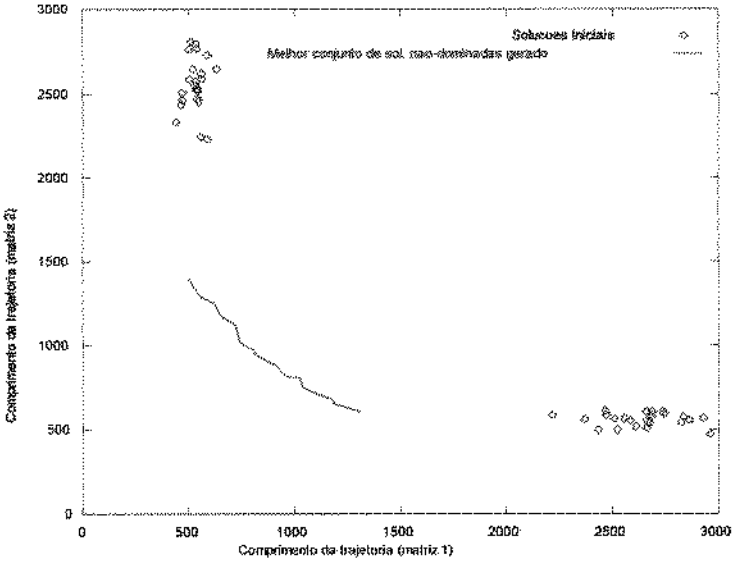


Figura 7. 23: O melhor conjunto de soluções não-dominadas gerado pelo A-Team c/ Distância Euclidiana, para a instância de 100-cidades.

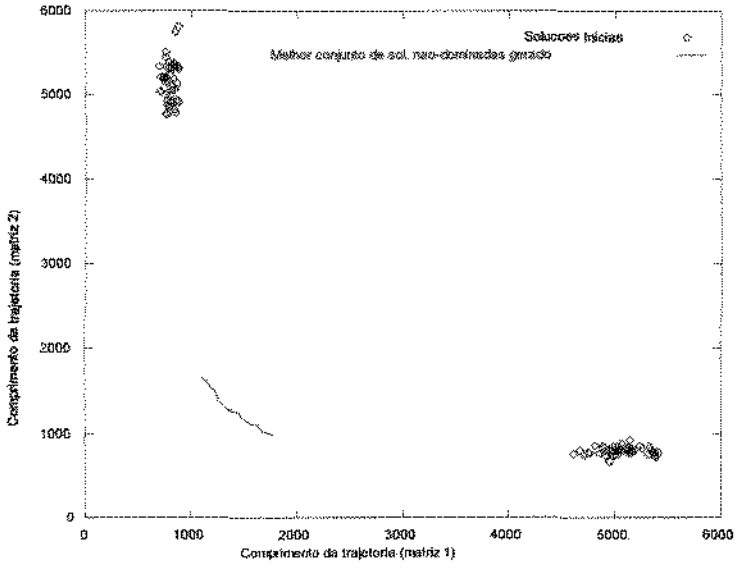


Figura 7. 24: O melhor conjunto de soluções não-dominadas gerado pelo A-Team c/ Distância Euclidiana, para a instância de 200-cidades.

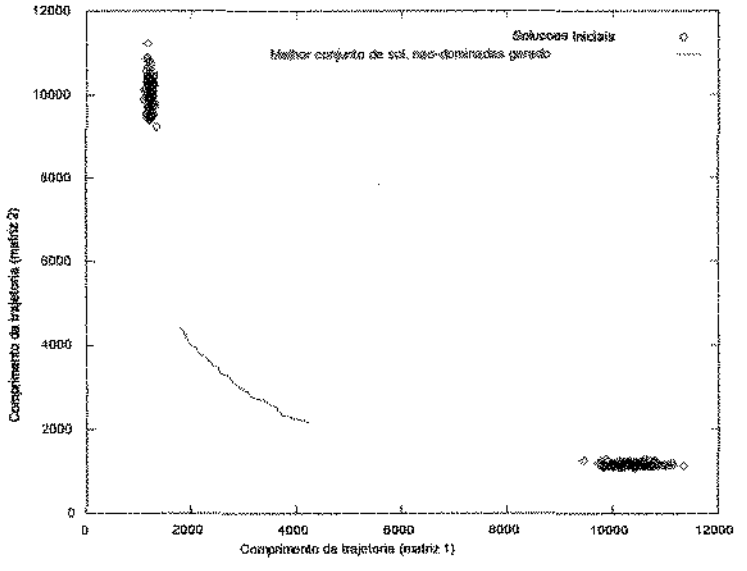
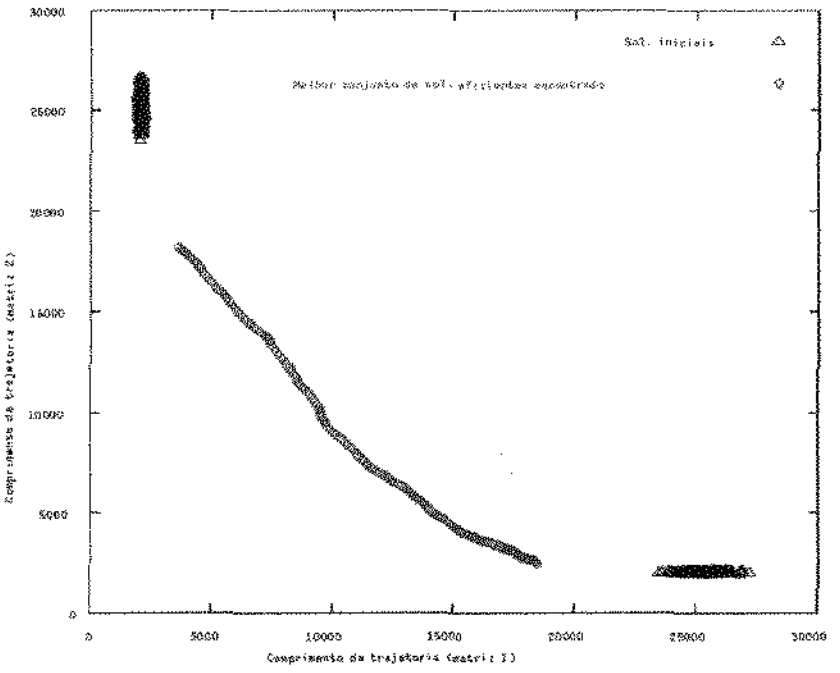


Figura 7. 25: O melhor conjunto de soluções não-dominadas gerado pelo A-Team c/ Distância Euclidiana, para a instância de 500-cidades.



Através dos resultados obtidos, pode ser comprovado que A-Teams forneceu como resultado conjuntos com um número considerável de soluções não-dominadas, para todas as instâncias do MDTSP testadas.

Entretanto, pode ser observado em todos os gráficos apresentados nesta subseção, resultantes da aplicação da métrica-L_2, que os pontos extremos (soluções) do gráfico não são mantidos no conjunto de soluções não-dominadas final, apesar de pertencerem ao melhor conjunto não-dominado. Isto é devido às características da métrica-L_2 na condução da evolução das soluções durante a execução do A-Team. Como seu contorno gráfico (ver Cap.6) favorece a região central do que se espera de um conjunto não-dominado, as soluções das extremidades são rapidamente eliminadas.

Este problema pode ser parcialmente solucionado com a adoção de outras políticas de destruição, como pôde ser observado no gráfico da figura 7.20. Também, com a manutenção do melhor conjunto não-dominado em tempo de execução (avaliação das soluções segundo o valor fornecido pela métrica, mas, sem eliminar nenhuma solução pertence ao melhor conjunto não-dominado, durante a execução - ver Cap.6), tal problema é significativamente reduzido.

Outra alternativa é a aplicação de estratégias mais eficazes, possibilitando um refinamento cada vez maior dos conjuntos não-dominados obtidos, como poderá ser observado no restante do capítulo.

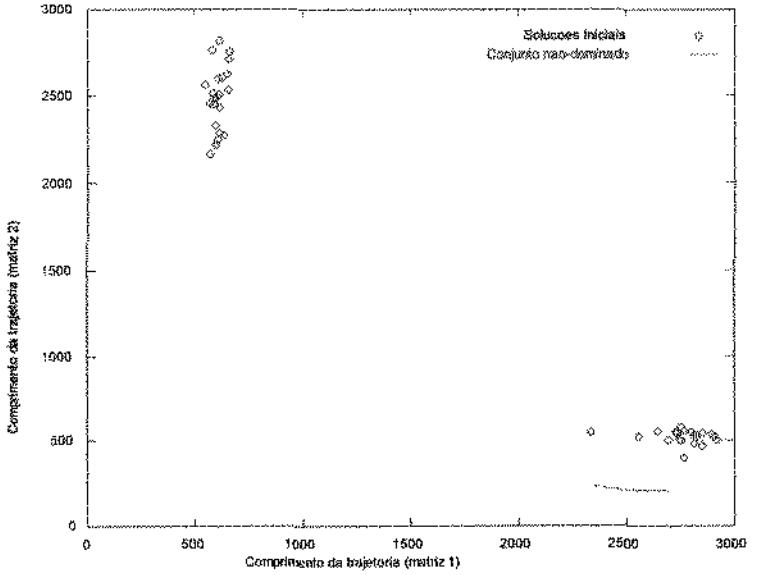
Estes primeiros resultados possibilitaram o desenvolvimento de um artigo apresentado no *5th Workshop of the DGOR-Working Group*, em Lambrecht, Alemanha, com publicação nos anais do evento [RS95a].

7.4.6. Times Assíncronos com Equação que Descreve uma Hipérbole

A estratégia envolvendo a equação que descreve uma hipérbole não apresentou bons resultados. A fig.7.26 apresenta o resultado da aplicação desta estratégia para o 2DTSP com 50-cidades.

Tal estratégia demonstrou uma sensibilidade muito grande às características das matrizes de distância envolvidas, ou seja, pequenas variações no intervalo numérico que contém os comprimentos das trajetórias geradas causam uma grande distorção no contorno gráfico dos conjuntos não-dominados gerados, como pode ser observado na figura 7.26.

Figura 7. 26: Resultado da aplicação da estratégia que utiliza a equação que descreve uma hipérbole para o 2DTSP de 50-cidades.



Mediante este fato, ocorrido em todas as execuções do A-Team quando usava estava estratégia, a mesma foi descartada. Desta forma, a estratégia que envolve a equação da hipérbole não será comparada com as demais, devido ao seu desempenho ter sido considerado muito ruim.

7.4.7. Times Assíncronos com Métricas-Lp em Relação à Origem dos Eixos

Todos os resultados apresentados nesta seção dizem respeito à aplicação de A-Team utilizando a Configuração 6 aplicada à instância do 2DTSP de 100-cidades.

Dentre as métricas L_p, a que possibilitou a geração, em média, do melhor conjunto não-dominado foi a métrica L_4.

A comparação é feita calculando-se o conjunto $\hat{\Theta}$ (conjunto não-dominado resultante da união de todos os conjuntos não-dominados gerados por cada métrica). A partir daí, calcula-se o índice de pertinência do conjunto não-dominado gerado por cada métrica em relação à $\hat{\Theta}$ (ver Cap.6).

A tabela 7.5 apresenta os resultados desta comparação:

Tabela 7. 5: Comparação entre as taxas de pertinência das métricas L_p, quando usadas em relação à origem dos eixos.

Métrica	Índice de Pertinência(%)
L_1	13
L_2	61
L_3	65
L_4	71
L_5	63
L_6	54
L_7	51
L_8	52
L_9	48
L_{10}	43
L_{∞}	11

Apesar da métrica L_4 gerar sempre um conjunto não-dominado com o maior índice de pertinência, se forem utilizadas como base de comparação as cinco faixas de valores apresentadas no Cap. 6, seção 6.7, as métricas L_2, L_3, L_4 e L_5 geram conjuntos não-dominados dentro da mesma faixa de qualidade. Isto porque todas as quatro garantiram um índice de pertinência, em média, entre 60 e 80% do conjunto $\hat{\Theta}$ (tab.7.5).

Portanto, será considerado que as quatro métricas - L_2, L_3, L_4 e L_5 -, em relação à origem dos eixos, tiveram o melhor desempenho, fornecendo conjuntos não-dominados de igual qualidade.

7.4.8. Times Assíncronos com Métricas-Lp em Relação a um Limite Inferior

Também neste caso, os resultados são referentes à aplicação do A-Team utilizando-se a Configuração 6 para o 2DTSP de 100-cidades.

Conforme estabelecido no Capítulo 6, a letra I anexada ao nome da métrica representa o uso da mesma para o cálculo da distância entre uma solução contida na memória do A-Team e uma solução "fabricada", servindo de Limite Superior para todos os cálculos de distância realizados.

Serão apresentados os resultados da aplicação das métricas L_p em relação a um Limite Superior. A comparação entre os conjuntos não-dominados gerados pelas diferentes métricas é feito da mesma maneira apresentada na subseção anterior.

Os índices de pertinência dos conjuntos gerados por cada métrica, em média, em relação à $\hat{\Theta}$, são apresentados na tabela 7.6.

Tabela 7.6 Comparação entre as taxas de pertinência das métricas L_p, quando usadas em relação a um Limite Superior.

Métrica	Índice de Pertinência(%)
L_1I	12
L_2I	66
L_3I	57
L_4I	75
L_5I	69
L_6I	59
L_7I	55
L_8I	52
L_9I	53
$L_{10}I$	42
$L_\infty I$	12

Neste caso, a métrica L_4I também possibilitou a geração, em média, do conjunto não-dominado de mais alto índice de pertinência. Já a métrica L_3I, por exemplo, teve sua média significativamente reduzida.

O resultado acima, com base nos cinco intervalos de qualidade apresentados no Cap.6, indica que as métricas L_2I, L_4I e L_5I possibilitaram a geração de conjuntos não-dominados de igual qualidade.

7.4.9. Origem dos Eixos *versus* Limite Superior

Em todos os dois casos, isoladamente, as métricas que possibilitaram a geração dos melhores conjuntos não-dominados foram: L_2, L_2I, L_3, L_4, L_4I, L_5, L_5I.

Esta subseção apresenta o resultado da comparação entre estas sete métricas. A tabela 7.7 apresenta o índice de pertinência dos conjuntos não-dominados, gerados por cada métrica, em relação à $\hat{\Theta}$ (que representa o melhor conjunto não-dominado resultante da união dos conjuntos não-dominados geradas por cada métrica individualmente).

Tabela 7.7 Comparação entre as melhores métricas em relação à origem dos eixos *versus* as melhores métricas em relação a um Limite Superior.

Métrica	Índice de Pertinência(%)
L_2	60
L_2I	62
L_3	52
L_4	65
L_4I	76
L_5	59
L_5I	64

A métrica L_4I possibilitou a geração do conjunto não-dominado com maior índice de pertinência dentre todas as outras métricas.

Mais uma vez, entretanto, se fará uso das cinco faixas de qualidade para concluir que:

- As métricas L_4I, L_4, L_5I, L_2I e L_2 possibilitaram a geração de conjuntos não-dominados de mesma qualidade;
- Em média, as métricas aplicadas a um Limite Superior tiveram um desempenho superior às métricas aplicadas à origem dos eixos;
- Os conjuntos não-dominados gerados abrangem toda a região esperada de um conjunto não-dominado.

Apesar dos resultados favoráveis, mesmo o melhor conjunto não-dominado - o fornecido pela métrica L4I -, possui muitas soluções não pertencentes ao melhor conjunto não-dominado gerado como resultado da comparação entre as métricas (o conjunto $\hat{\Theta}$). O ideal seria uma estratégia que conseguisse gerar um conjunto não-dominado o mais parecido possível com o conjunto $\hat{\Theta}$, quando comparada com qualquer outra estratégia.

A estratégia de Camadas de Soluções Não-Dominadas satisfaz esta necessidade, como poderá ser observado na subseção seguinte.

7.4.10. Times Assíncronos com Camadas de Soluções Não-Dominadas

A estratégia de resolução do MDTSP através de Times Assíncronos com Camadas de Soluções Não-Dominadas foi a que possibilitou a geração dos melhores conjuntos de soluções não-dominadas em todos os testes realizados.

A seguir, são apresentados e analisados os resultados obtidos para instâncias do 2DTSP e do 3DTSP, respectivamente.

A configuração que possibilitou a obtenção dos melhores conjuntos não-dominados foi a **Configuração 6**, para todas as instâncias e em todas as dimensões testadas.

7.4.10.1. 2 Dimensões

Os melhores resultados na aplicação da estratégia de Camadas de Soluções Não-Dominadas para instâncias do 2DTSP foram obtidos utilizando os seguintes parâmetros:

- Política de destruição de soluções com distribuição triangular de probabilidade;
- Tamanho de memória em $3n$ (n é o tamanho da instância);
- Algoritmos que lidam com uma matriz de distância de cada vez;
- Algoritmos que lidam com várias matrizes de distância simultaneamente para a geração de soluções.

Inicialmente, foram realizados inúmeros testes com todas as instâncias para a verificação do comportamento do A-Team quanto ao: número de camadas de soluções na memória, número de soluções em cada camada e freqüência com que as soluções

entravam ou saíam do melhor conjunto não-dominado. Isto em todas as fases do processamento. Se verificou que no início da execução muitas camadas iam se formando, com poucas soluções em cada uma delas. À medida que novas soluções iam sendo geradas, o número de camadas diminuía até chegar a um ponto em que restavam somente de uma a três camadas e, mesmo assim, a melhor camada com a maioria absoluta das soluções. Também, a freqüência com que novas soluções eram inseridas no melhor conjunto e de que antigas retiradas, diminuía ao longo do processo de execução, até um ponto em que uma alteração demorasse muito para acontecer.

Com base nesta análise empírica, dois pontos importantes foram considerados na definição do critério de parada do A-Team:

- Número de camadas na memória;
- Oscilação do melhor conjunto não-dominado na memória (incidência de alterações - soluções entrando ou saindo do melhor conjunto).

Desta forma, assumiu-se como convergência do A-Team o momento em que o número de camadas na memória fosse o menor possível e a oscilação do melhor conjunto não-dominado fosse praticamente inexistente.

Durante os testes, pôde-se perceber que, para cada tamanho de instância, existia um momento em que tais condições não mais se alteravam: nenhum agente conseguia gerar uma solução que pertencesse ao melhor conjunto não-dominado da memória. Então, foi prefixado um tempo de execução, para cada tamanho de instância, que possibilitasse ao A-Team chegar a este estado de "convergência".

Analisando os gráficos para as diferentes instâncias do 2DTSP, podemos perceber que todos apresentam o mesmo comportamento. As soluções iniciais foram geradas pelos algoritmos de construção NI, FI e AI, aplicados a cada matriz de uma vez. Em seguida, interagiram algoritmos aplicados a apenas uma matriz de cada vez e, também, os novos algoritmos que incorporam conceitos de dominância e decisão de compromisso. Esta combinação possibilitou a obtenção de melhores conjuntos não-dominados, mais rapidamente.

Para todas as instâncias, o A-Team forneceu como resultado grandes conjuntos de soluções não-dominadas, abrangendo todo o espectro esperado de um conjunto não-dominado. Apesar de boas soluções serem geradas na parte central do contorno geométrico desses conjuntos, isto não excluiu a obtenção de boas soluções nos extremos do gráfico, ou seja, as soluções muito boas num eixo, mesmo que ruins no outro, também foram geradas, já que fazem parte do melhor conjunto não-dominado.

Para as instâncias de 9 e 10-cidades, o A-Team com Camadas de Soluções Não-Dominadas gerou todo o conjunto de soluções eficientes - o Pareto Ótimo, como pode ser observado nas figuras 7.27 e 7.28.

Para as instâncias maiores - 50, 100, 200 e 500 -, o número de soluções não-dominadas fornecidas pelo A-Team como resultado foi restringido pelo tamanho da memória, ou seja, as soluções finais foram, em sua maioria absoluta, pertencentes ao melhor conjunto não-dominado.

As figuras 7.27, 7.28, 7.29, 7.30, 7.31 e 7.32 apresentam os gráficos com os melhores conjuntos não-dominados obtidos para o 2DTSP, com instâncias de 9, 10, 50, 100, 200 e 500-cidades, respectivamente.

Figura 7. 27: Resultado da aplicação da estratégia de Camadas de Soluções Não-Dominadas para o 2DTSP de 9-cidades. O A-Team gerou o próprio Pareto Ótimo do problema, com 13 soluções.

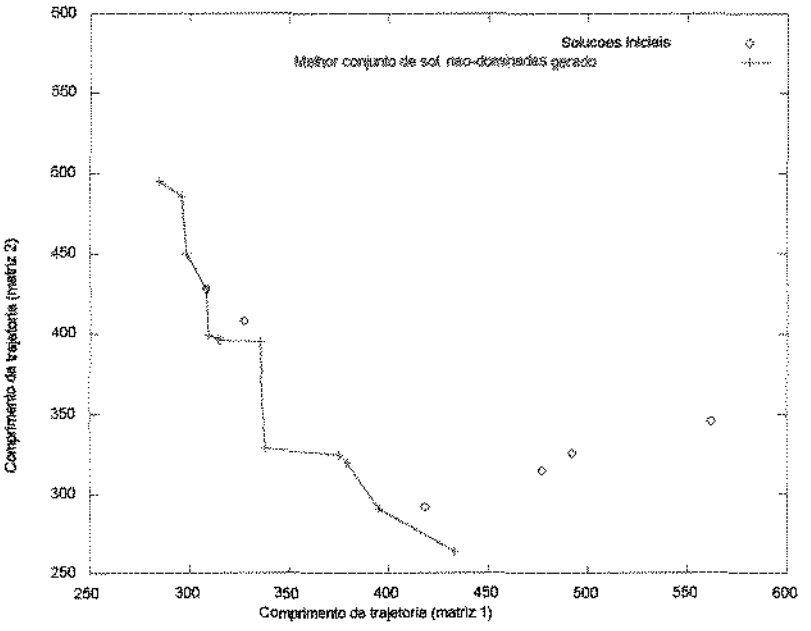


Figura 7. 28: Resultado da aplicação da estratégia de Camadas de Soluções Não-Dominadas para o 2DTSP de 10-cidades. Neste caso, o A-Team também gerou todas as soluções pertencentes ao Pareto Ótimo, num total de 22 soluções.

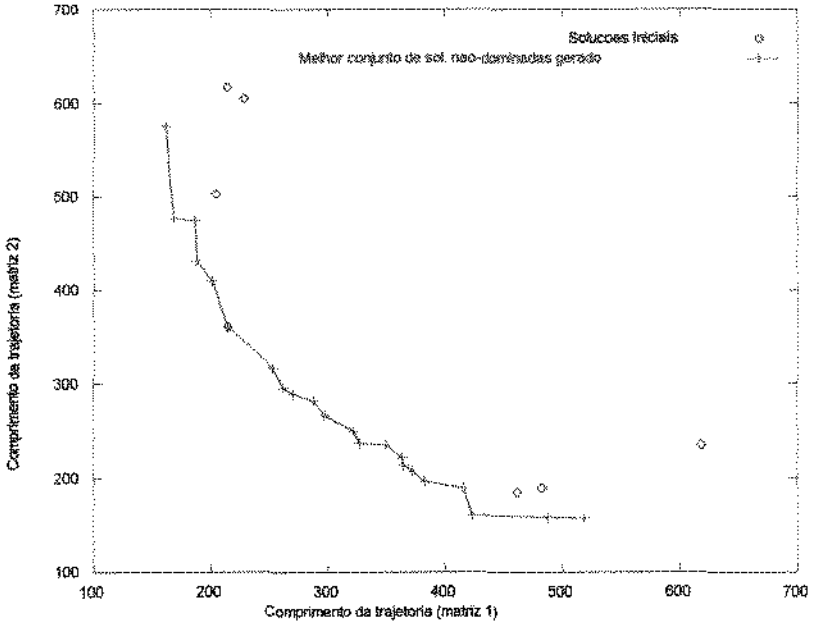


Figura 7. 29: Resultado da aplicação da estratégia de Camadas de Soluções Não-Dominadas para o 2DTSP de 50-cidades.

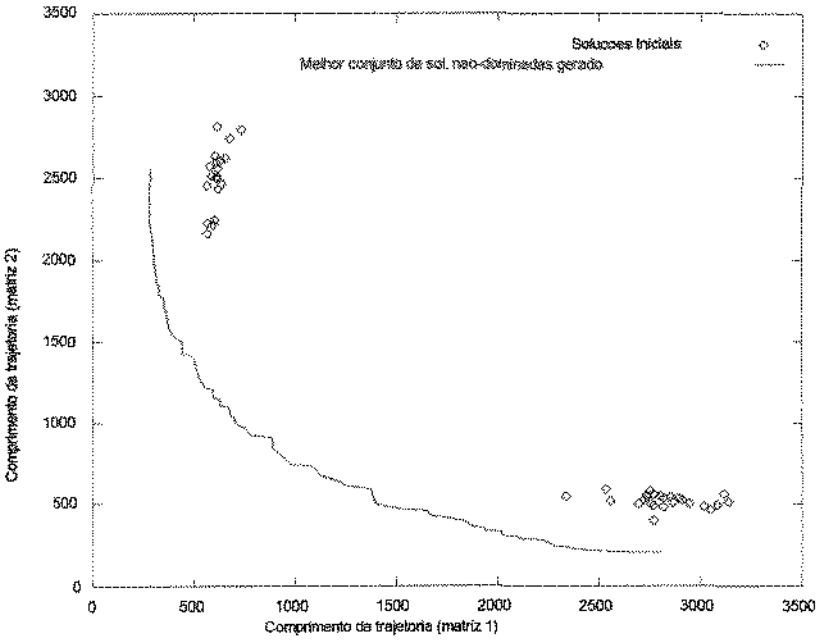


Figura 7. 30: Resultado da aplicação da estratégia de Camadas de Soluções Não-Dominadas para o 2DTSP de 100-cidades.

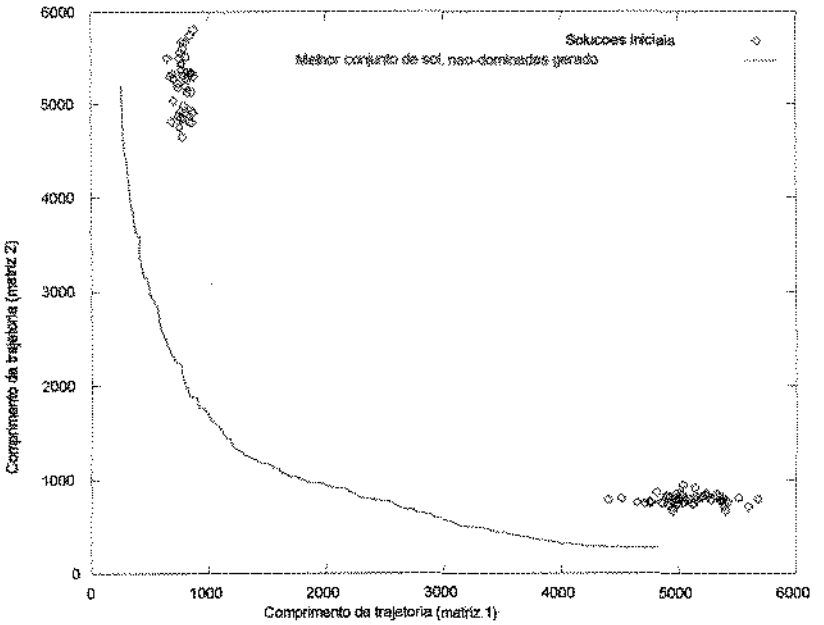
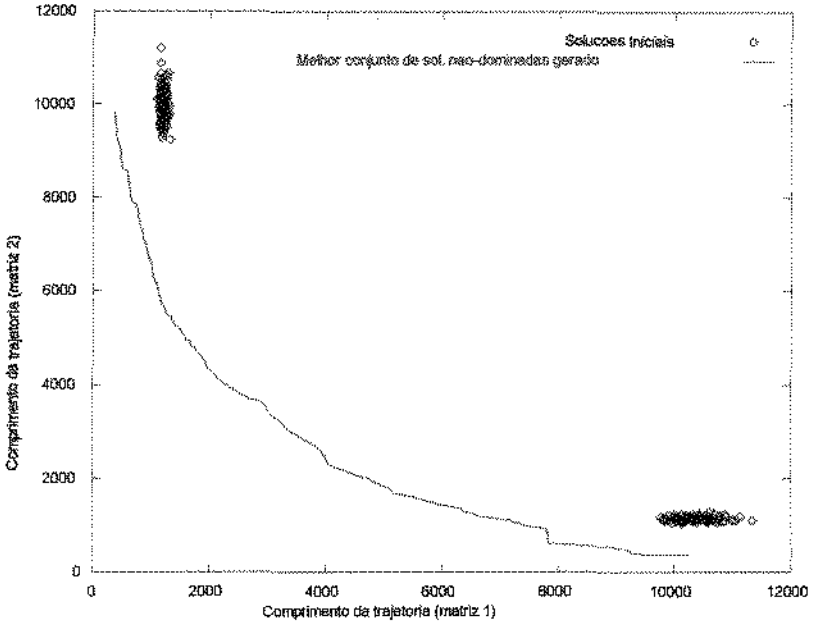


Figura 7. 31: Resultado da aplicação da estratégia de Camadas de Soluções Não-Dominadas para o 2DTSP de 200-cidades.



O Problemas das 24 Principais Cidades Brasileiras

O A-Team forneceu como resultado para a instância real de 24-cidades (Problema das Principais Cidades Brasileiras), um conjunto de 35 soluções não-dominadas, conforme pode ser observado no gráfico da figura 7.33.

A seguir, são apresentadas nove soluções para o problema. Todas iniciam com a primeira cidade das matrizes (AJU - Aracaju), conforme pode ser observado no Apêndice B. Na tabela 6.4 (Cap.6) encontram-se os códigos das 24-cidades.

A primeira solução é a que possui o menor valor segundo a primeira matriz (custo da viagem), cujos valores foram (885,1090):

AJU MCZ REC JPA NAT FOR SLZ MCP BEL MAO PVH CGB CGR SAO IGU CWB POA FLN RIO VIX BHZ GYN BSB SSA

A fig. 7.33 apresenta esta trajetória sobre o mapa do Brasil, onde pode ser claramente visualizado a ordem e a distância a ser percorrida. Vale ressaltar que não existe cruzamentos entre as arestas (caminhos entre as cidades), o que comprova a afirmação anterior (Cap.6) de que a matriz de custo é diretamente proporcional à matriz de distância.

Figura 7. 33: Melhor trajetória para o Problema das 24 Principais Cidades Brasileiras segundo a matriz de custo.

A segunda solução, possui o menor valor segundo a segunda matriz (qualidade do vôo), cujos valores foram (2916,815):

AJU MCZ REC JPA NAT FOR SLZ MCP BEL BSB MAO PVH CGB CGR SAO RIO FLN POA CWB IGU GYN BHZ VIX SSA

A fig. 7.34 apresenta esta trajetória sobre o mapa do Brasil, onde percebe-se a existência de grandes caminhos (arestas) e de cruzamentos entre os mesmos.

Figura 7. 34: Melhor trajetória para o Problema das 24 Principais Cidades Brasileiras segundo a matriz de qualidade.

As duas soluções abaixo possuem os mesmos valores segundo as duas matrizes de distância - (908,1055) - entretanto, a ordem em que as cidades devem ser visitadas é diferente:

AJU MCZ REC JPA NAT FOR SLZ MCP BEL MAO PVH CGB CGR SAO RIO CWB IGU FLN POA BSB GYN BHZ VIX SSA

AJU MAO GYN MCZ REC JPA NAT FOR BSB BEL SAO SLZ POA BHZ VIX CWB RIO FLN IGU PVH CGR CGB MCP SSA

Novamente, as duas soluções abaixo possuem os mesmos valores segundo as duas matrizes de distância - (953,1030) - mas, a ordem das cidades é diferente:

AJU MCZ REC SAO RIO CWB GYN MAO PVH CGB CGR IGU FLN VIX BHZ POA SLZ MCP BEL BSB FOR NAT JPA SSA

AJU MCZ REC SAO SLZ POA BHZ VIX FLN IGU CGR CGB PVH MAO GYN CWB RIO MCP BEL BSB FOR NAT JPA SSA

As três soluções abaixo são bons exemplos de soluções pertencentes a parte central do contorno gráfico do melhor conjunto não-dominado. A primeira, de comprimentos (902,1055), a segunda, (1009,1020), e a terceira, (1970,855):

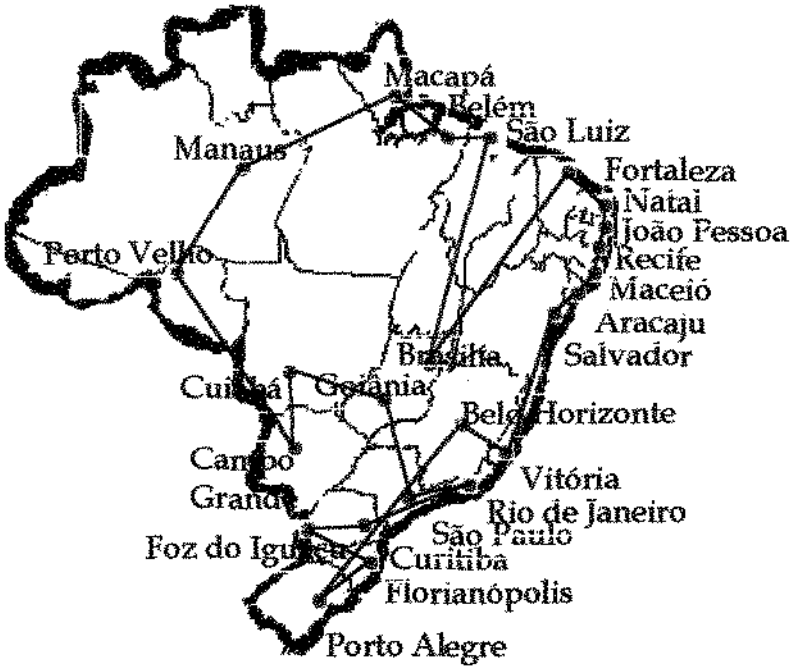
AJU MCZ REC JPA NAT FOR SLZ MCP BEL BSB MAO PVH CGB CGR IGU CWB POA FLN RIO SAO GYN BHZ VIX SSA

AJU MCZ REC JPA NAT FOR BSB SLZ BEL MCP MAO PVH CGR CGB GYN SAO RIO CWB IGU FLN POA BHZ VIX SSA

AJU MCZ REC JPA NAT FOR SLZ MCP BEL MAO PVH CGB CGR RIO FLN POA CWB IGU SAO GYN BSB BHZ VIX SSA

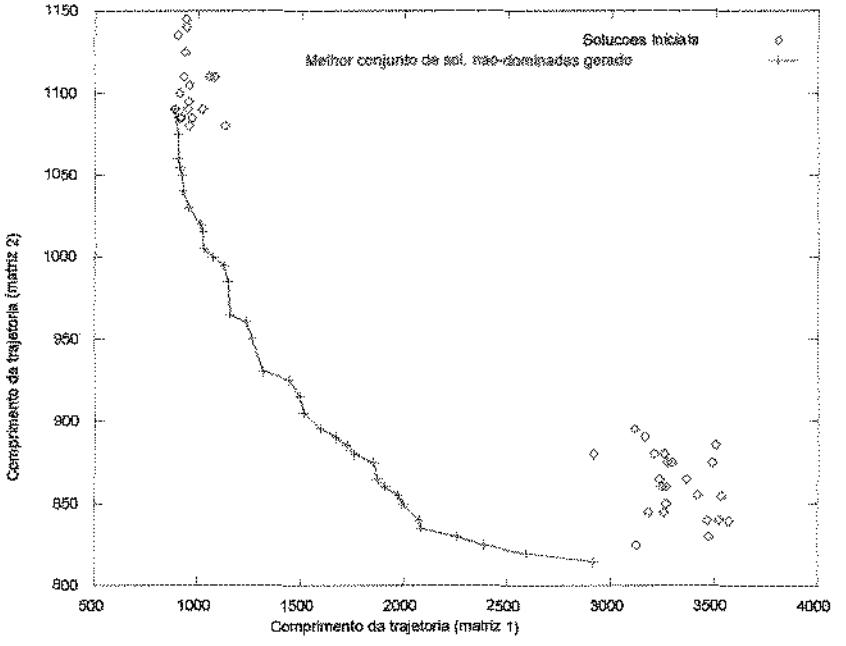
A fig. 7.35 apresenta a trajetória de comprimentos (1009,1020) sobre o mapa do Brasil. Esta trajetória é de compromisso entre a qualidade e o custo envolvidos no problema. Pode ser observado, também neste caso, a existência de grandes caminhos (arestas) e de cruzamentos entre eles.

Figura 7. 35: Trajetória de compromisso entre a matriz de custo e a matriz de qualidade, pertencente ao melhor conjunto não-dominado gerado pelo A-Team e de comprimentos (1009,1020).



Apesar de não se tratar de um problema complexo, as matrizes envolvidas possuem valores distribuídos em intervalos numéricos diferentes, o que é uma forte característica de problemas reais. Mesmo assim, o A-Team propiciou a geração de um conjunto não-dominado abrangendo todo o espectro esperado: de um extremo ao outro (melhores valores segundo cada matriz), fornecendo várias soluções intermediárias aos mesmos (fig.7.36).

Figura 7. 36: Resultado da aplicação da estratégia de Camadas de Soluções Não-Dominadas para a instância real do 2DTSP com 24-cidades.



7.4.10.2. 3 Dimensões

As figuras 7.37, 7.38, 7.39 e 7.40 apresentam os gráficos com os melhores conjuntos não-dominados obtidos para o 3DTSP, onde são reportados os resultados obtidos para as instâncias de 10 e 100-cidades.

Os melhores resultados na aplicação da estratégia de Camadas de Soluções Não-Dominadas para instâncias do 2DTSP foram obtidos utilizando os seguintes parâmetros:

- Política de destruição de soluções com distribuição triangular de probabilidade;
- Tamanho da memória em $5n$ (n é o tamanho da instância);
- Algoritmos que lidam com uma matriz de distância de cada vez;
- Algoritmos que lidam com várias matrizes de distância simultaneamente para a geração de soluções.

Também para o 3DTSP, o A-Team utilizando Camadas de Soluções Não-Dominadas foi capaz de gerar o Pareto Ótimo para as instâncias de 9 e 10-cidades.

As figuras 7.37 e 7.38 apresentam o Pareto Ótimo obtido para o caso do 3DTSP com 10-cidades. A figura 7.37 apresenta o resultado em 3-dimensões. A figura 7.38 apresenta dois gráficos contendo o mesmo conteúdo do gráfico anterior, mas agora em 2-dimensões, onde se fixou a matriz_1, no eixo x, como referência. No primeiro gráfico (fig.7.38), é apresentado o conjunto não-dominado segundo a matriz_1 (eixo x) e a matriz_2 (eixo y). No segundo gráfico, permanece a matriz_1 (eixo x) agora relacionada com a matriz_3 (eixo y).

Em seguida, é apresentado o melhor conjunto não-dominado obtido para o 3DTSP com 100-cidades. Os gráficos são apresentados da mesma maneira apresentada para o caso de 10-cidades. A figura 7.39 apresenta os dados em 3-dimensões. A figura 7.40 apresenta os dois gráficos em 2-dimensões, onde a matriz_1 é fixada no eixo x e, no eixo y, permuta-se a matriz_2 e a matriz_3, respectivamente, em cada gráfico.

Pode ser observado que o A-Team, mesmo na resolução do MDTSP com três matrizes de distância, possibilitou a geração de um conjunto não-dominado abrangendo toda a região esperada de um conjunto não-dominado, com soluções pertencentes aos extremos (favoráveis a determinada matriz, num determinado eixo) e várias soluções pertencentes ao intervalo entre esses extremos.

Figura 7. 37: Gráfico em 3 dimensões do resultado da aplicação da estratégia de Camadas de Soluções Não-Dominadas para o 3DTSP c/ 10-cidades.

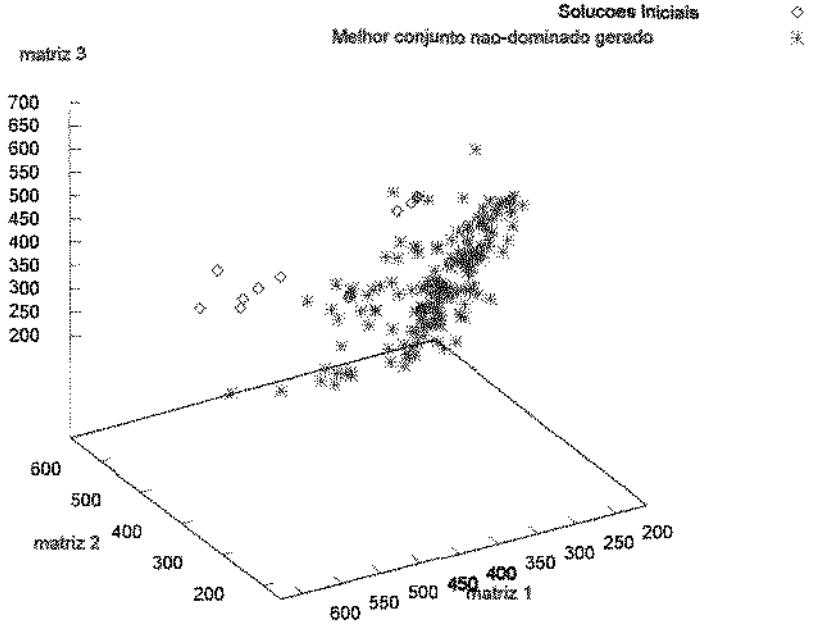


Figura 7. 38: Gráficos em 2 dimensões representando o resultado da estratégia de Camadas de Soluções Não-Dominadas, para o 3DTSP c/ 10-cidades. O eixo *x* está fixado com os valores da matriz de distância 1 e o eixo *y* varia entre a matriz de distância 2, no primeiro gráfico, e a matriz de distância 3, no segundo gráfico.

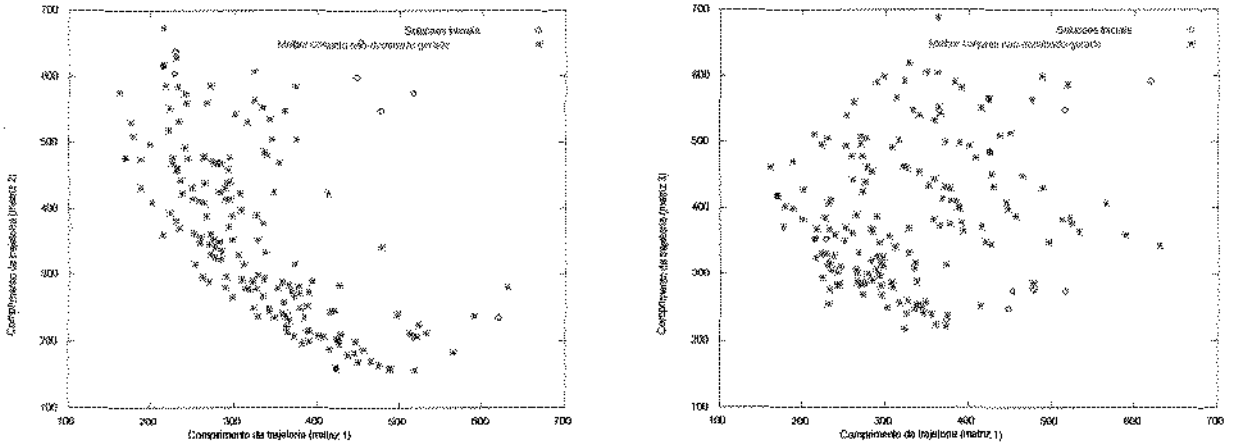


Figura 7. 39: Gráfico em 3 dimensões do resultado da aplicação da estratégia de Camadas de Soluções Não-Dominadas para o 3DTSP c/ 100-cidades.

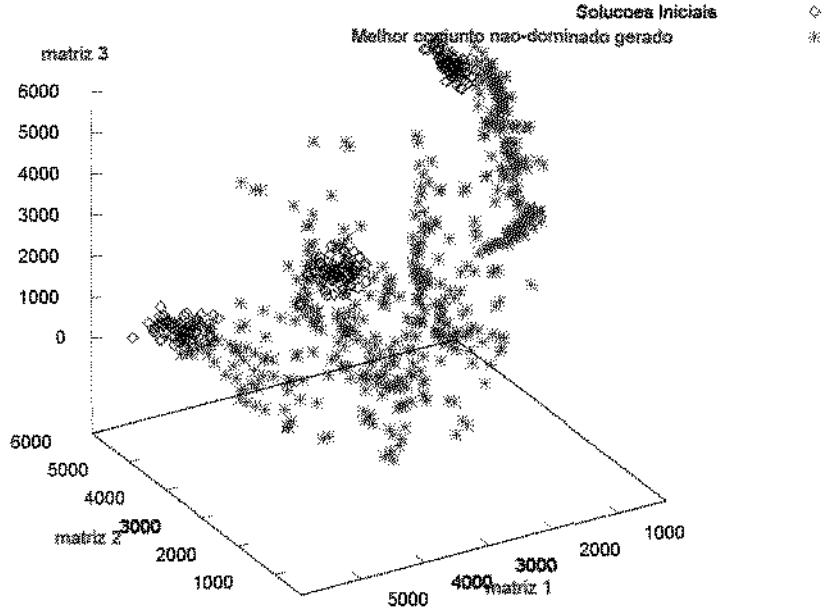
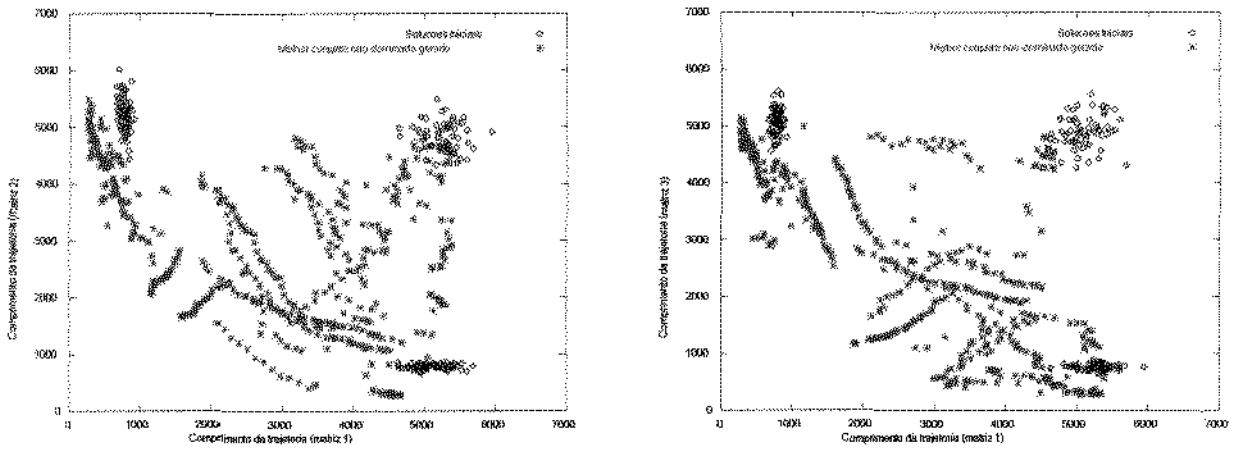


Figura 7. 40: Gráficos em 2 dimensões representando o resultado da estratégia de Camadas de Soluções Não-Dominadas, para o 3DTSP c/ 100-cidades. O eixo *x* está fixado com os valores da matriz de distância 1 e o eixo *y* varia entre a matriz de distância 2, no primeiro gráfico, e a matriz de distância 3, no segundo gráfico.



7.4.11. Comparação entre Todas as Estratégias Implementadas

Esta subseção apresenta uma análise comparativa, com base no método de índice de pertinência descrito no Capítulo 6, entre as melhores estratégias desenvolvidas para a resolução do Multi-Distance Traveling Salesman Problem - MDTSP através de Asynchronous Teams - A-Teams.

A tabela abaixo (tab.7.8) contém os resultados desta comparação. Na primeira coluna encontram-se as estratégias utilizadas e, nas demais, os índices de pertinência de cada conjunto não-dominado obtido por cada uma das estratégias em relação ao conjunto $\hat{\Theta}$, que agora representa o melhor conjunto possível resultante da união dos melhores conjuntos não-dominados gerados por cada uma das estratégias, durante todos os testes realizados.

As instâncias apresentadas são de 9, 10, 50, 100, 200 e 500 cidades. Para as instâncias de 9 e 10 cidades, o A-Team com Camadas de Soluções Não-Dominadas forneceu o próprio Pareto Ótimo do problema. O asterisco ao lado da porcentagem, indica que 100%, neste caso, é o próprio Pareto Ótimo. Entre parênteses está o número de soluções pertencentes ao Pareto Ótimo que cada estratégia foi capaz de gerar (isto só é possível para 9 e 10-cidades, casos em que é conhecido o Pareto Ótimo do problema).

Tabela 7. 8: Comparação entre os índices de pertinência das melhores estratégias desenvolvidas para a resolução do MDTSP através de A-Teams.

Índice de Pertinência (%)

Estratégia \ Instância	9	10	50	100	200	500
Camadas Não-Dominadas	100*(13)	100*(22)	100	91	77	64
Métrica-L_2	61.53 (8)	59.09 (13)	59	40	34	21
Métrica-L_2I	69.23 (9)	59.09 (13)	67	42	31	25
Métrica-L_4	69.23 (9)	54.54 (12)	69	43	33	21
Métrica-L_4I	69.23 (9)	63.63 (14)	71	56	37	28
Métrica-L_5	69.23 (9)	63.63 (14)	63	53	32	22

A Camada de Soluções Não-Dominadas possibilitou a geração dos melhores conjuntos não-dominados para todas as instâncias testadas e garantiu índices de pertinências, em relação ao melhor conjunto $\hat{\Theta}$ obtido, muito superiores à qualquer uma das outras estratégias utilizadas.

7.4.12. Análise do Tempo de Processamento do A-Team

Em relação ao tempo de processamento, todas as métricas-L_p apresentam o mesmo desempenho, entretanto, a estratégia de Camadas de Soluções Não-Dominadas necessita de um tempo de execução muito superior ao necessário para as demais, isto para a geração dos melhores conjuntos não-dominados.

A tabela 7.9 apresenta os tempos de execução das métricas-L_p *versus* Camada de Soluções Não-Dominadas. Esta comparação não foi feita equiparando-se a qualidade dos conjuntos não-dominados gerados, isto porque o conjunto não-dominado gerado através do uso de métricas L_p é sempre inferior ao obtido pela aplicação de Camadas de Soluções Não-Dominadas, mesmo se o tempo de execução for excessivamente longo. Mediante este fato, o tempo de execução de cada estratégia foi estipulado com base na "convergência" do A-Team, onde *convergência*, neste caso, indica que o melhor conjunto não-dominado contido na memória do A-Team passou uma quantidade predeterminada de tempo sem que nenhuma nova solução fosse inserida ou retirada do mesmo.

Tabela 7. 9: Tempo de execução do A-Team aplicado ao 2DTSP: estratégia de Camada de Soluções Não-Dominadas *versus* métricas L_p.

Tempo de Execução (s)

Estratégia \ Instância	9	10	50	100	200	500
A-Team c/ Camadas de Soluções Não - Dominadas	10	50	10800	39600	61200	100800
A-Team c/ Métricas-L_p	1	5	2900	15120	18000	25200

O tempo de processamento do A-Team aumenta significativamente à medida que a dimensão do problema aumenta. A tab.7.10 contém os tempos de execução do A-Team com Camadas de Soluções Não-Dominadas, para a instância de 100-cidades, quando se tem uma, duas e três matrizes de distância envolvidas.

Tabela 7. 10: Tempo de processamento do A-Team, variando o número de dimensões do problema.

Tempo de Execução (s)

100-cidades

Estratégia \ Instância	1D	2D	3D
A-Team c/ Camadas de Soluções Não - Dominadas	14400	39600	113600
A-Team c/ Métricas-L_p	7200	15120	35800

Apesar de muito superior, o tempo de execução do A-Team com Camadas de Soluções Não-Dominadas ainda está numa faixa tolerável. Entretanto, este tempo de execução é significativamente melhorado quando se utiliza processamento paralelo, conforme poderá ser observado na subseção seguinte.

7.4.13. Processamento Paralelo

Os resultados obtidos com a versão paralela do A-Team é resumido na tabela 7.10. Foi comparado o desempenho do A-Team com 1, 2, 4 e 8 processadores.

Os testes desta subseção forma realizados com a Configuração 6. A estratégia usada é a de Camadas de Soluções Não-dominadas.

Para efeito de comparação entre os conjuntos não-dominados gerados quando usados um, dois, quatro ou oito processadores, executou-se o A-Team concorrente para a instância de 100-cidades várias vezes (dez vezes) para, então, ser determinado o conjunto $\hat{\Theta}$ (melhor conjunto não-dominado da união de todos os conjuntos das várias execuções do A-Team). Desta forma, fixa a condição de comparação, que é a qualidade do conjunto não-dominado gerado e o que se deseja saber é em quanto tempo o A-Team consegue obter um conjunto não-dominado com um nível de qualidade igual ou superior a 80% de $\hat{\Theta}$ (índice de pertinência de cada conjunto em $\hat{\Theta}$), em suas versões concorrente e paralela com dois, três e quatro processadores.

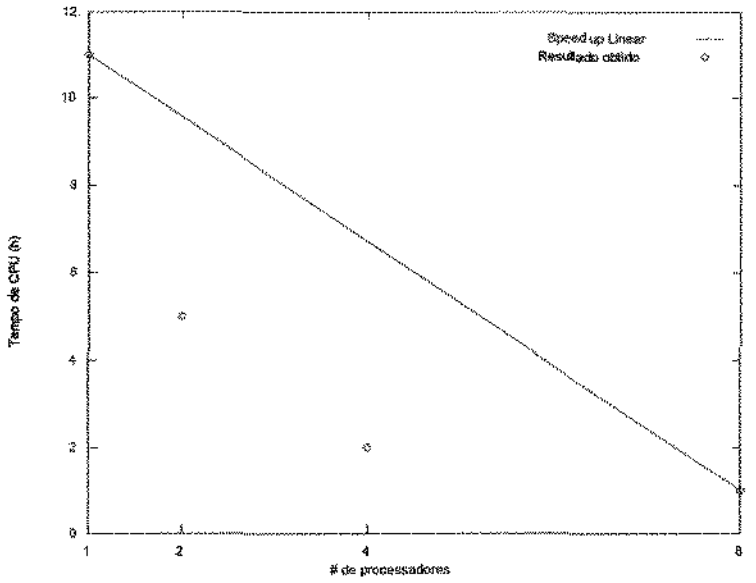
Resumindo, o A-Team foi executado para instância do 2DTSP com 100-cidades, em cada caso, até que atingisse um índice de pertinência igual ou maior a 80%. Os resultados são apresentados na tabela abaixo (tab. 7.11):

Tabela 7. 11: Comparação do desempenho do A-Team usando um, dois, quatro e oito processadores.

100-cidades	Nº de Processadores	1	2	4	8
	Tempo de CPU (h)	11	5,5	3,1	1,6

Tais resultados apresentam um *speed up* próximo do linear, conforme pode ser melhor observado no gráfico da fig.7.41.

Figura 7. 41: Resultado da execução paralela do A-Team com Camadas de Soluções Não-Dominadas, para a instância do 2DTSP com 100-cidades, utilizando um, dois, quatro e oito processadores. A curva representa a redução do tempo de execução em relação ao aumento do número de processadores utilizados.



O tempo de execução do A-Team diminui consideravelmente com o aumento do número de processadores, o que torna ainda muito mais indicado a adoção da estratégia de Camadas de Soluções Não-Dominadas, já que o fator tempo deixa de ser muito problemático quando se paraleliza o A-Team e, por outro lado, a qualidade dos conjuntos não-dominados gerados pelo A-Team através da estratégia de Camadas de Soluções Não-Dominadas é muito superior à qualidade dos conjuntos fornecidos pelas demais estratégias.

7.5. Conclusões

Dos testes realizados neste capítulo, concluir que:

- As adaptações realizadas na metodologia proposta por Peixoto [Pei95], de forma a englobar também problemas multiobjetivos, foi aplicada com sucesso ao MDTSP;
- Os A-Teams desenvolvidos apresentaram eficiência em escala, ou seja, à medida que novos agentes eram inseridos no time, o desempenho do A-Team melhorava monotonicamente, possibilitando a geração de conjuntos não-dominados de qualidade cada vez melhor;
- Os A-Teams desenvolvidos também apresentaram *speed up* muito próximo do linear, em relação ao número de processadores utilizados. Isto quer dizer que, quanto mais processadores são utilizados proporcionalmente menor é o tempo de processamento do A-Team;
- Dentre as funções objetivos escalar desenvolvidos, a que usa a métrica L_4 em relação a um Limite Superior (L_4I) apresentou os melhores resultados. As métricas L_2 e L_5,

também em relação a um Limite Superior (L_2I e L_5I), e L_2 e L_4, em relação à origem dos eixos, tiveram um desempenho muito próximo do obtido pela melhor métrica e os conjunto não-dominados gerados pelo A-Team utilizando tais métricas podem ser considerados de igual qualidade;

- A estratégia de organização das soluções na memória do A-Team de Camadas de Soluções Não-Dominadas foi a melhor dentre todas as desenvolvidas. Grandes conjuntos de soluções não-dominadas, abrangendo todo o espectro esperado foram obtidos para todas as instâncias e em todas as dimensões testadas. Para as instâncias do 2DTSP e do 3DTSP com até 10-cidades *todas* as soluções do Pareto Ótimo foram encontradas;
- Os conjuntos de soluções não-dominados fornecidos pelo A-Team, para instâncias acima de 10-cidades do 2DTSP e para todas do 3DTSP, são limitados pelo tamanho da memória, o seja, o A-Team possibilitou a geração de conjuntos não-dominados tão grandes quanto o tamanho da memória, em todos os casos;
- O desempenho dos A-Teams, quando neles embutidos os algoritmos que manipulam conceitos de dominância e decisão de compromisso, foram mais eficazes, em tempo de processamento e qualidade das soluções geradas.

7.6. Resumo

Neste capítulo foram apresentados os resultados computacionais, e suas respectivas análises, referentes à aplicação de A-Team para o MDTSP.

A implementação de todos os A-Teams desenvolvidos foi realizada através da **linguagem C**. Todos os resultados foram representados graficamente através do software **gnuplot**, tanto em ambiente UNIX quanto em DOS. As versões paralelas através do pacote *Parallel Virtual Machine* - **PVM**. Para o cálculo de limites inferiores foram utilizados o pacote **CPLEX**, através da resolução do assignment problem e o algoritmo de Held-Karp (ver Apêndices A e B).

A maioria dos algoritmos utilizados foram obtidos da literatura. Pequenas alterações em suas estruturas possibilitaram um melhor desempenho dos mesmos na geração de soluções de compromisso para os objetivos envolvidos.

A estratégia de Camadas de Soluções Não-Dominadas possibilitou a manipulação de várias funções objetivo, simultaneamente, para a geração de soluções. Os conjuntos não-dominados gerados através desta estratégia foram os melhores conjuntos obtidos, para qualquer instância e em todas as dimensões testadas.

7.7. Notas Bibliográficas

Alguns dos resultados apresentados neste capítulo foram previamente divulgados em [RS94, RS95a, RS95b, RS95c, RS96a, RS96b].

Capítulo
8

Conclusão

"Todas as coisas se movem para o seu fim."
Rabelais.

Todos os objetivos propostos inicialmente foram alcançados com êxito. Os resultados obtidos corroboraram com o principal objetivo da dissertação que foi mostrar a adequabilidade de Times Assíncronos para a resolução de Problemas Multiobjetivos.

Fez-se uso da metodologia proposta por Peixoto e Souza [PS95], sobre a qual foram realizadas as adaptações necessárias de forma a abranger também problemas com múltiplas funções objetivos. Esta é a primeira vez que se adapta tal método atendendo as características de problemas multiobjetivos. Diferentes estratégias para a manipulação de soluções pelos algoritmos de um A-Team foram desenvolvidas. Métricas L_p foram utilizadas como forma de eliminar soluções não promissoras da memória. Um novo tipo de organização de soluções na memória foi apresentado, onde as soluções são dispostas em camadas cada vez mais crescente de dominância: a Memória de Camadas de Soluções Não-Dominadas. Algoritmos de inserção e eliminação de soluções num espaço *k*-dimensional (onde *k* é o número de objetivos do problema) foram desenvolvidos, de maneira a suportar a estrutura de memória proposta. Novas políticas de destruição e seleção de soluções foram necessárias, também visando atender a nova estrutura de memória. Ainda, esta nova estrutura de memória resultou numa melhor estratégia para a eliminação de soluções e conseqüente evolução das soluções no A-Team, fornecendo soluções não-dominadas melhores do que as obtidas pela estratégia desenvolvida por Murthy [Mur92] e melhores do que todas as outras estratégias testadas. Sugestões para o aproveitamento de algoritmos existentes para o caso de problemas monobjetivos e para o desenvolvimento de novos algoritmos foram comentadas, bem como sugestões em todos os parâmetros de projeto, de tal forma que esta dissertação pode servir como base de

consulta para futuras aplicações de A-Teams para Problemas Combinatórios Multiobjetivos.

Foi apresentado um novo problema NP-difícil, o Problema do Caixeiro Viajante com Múltiplas Distâncias (*Multi-Distance Traveling Salesman Problem - MDTSP*), que é um problema de otimização combinatória multiobjetivo, proposto como uma generalização do clássico TSP, caracterizado por possuir k custos entre as cidades. Sua definição, complexidade, aplicações práticas e algoritmos foram apresentados. Os algoritmos para o novo problema proposto foram desenvolvidos com a capacidade de manipular várias matrizes de distância através da incorporação de conceitos de dominância e decisão de compromisso.

Foram desenvolvidas versões concorrentes e paralelas de A-Teams para o MDTSP. O processamento paralelo foi obtido através do pacote PVM (*Parallel Virtual Machine*). Instâncias geradas randomicamente foram utilizadas durante a realização da maioria dos testes computacionais, com até 500-cidades e 3 matrizes de distância. Uma instância real envolvendo as principais cidades brasileiras, com 24-cidades, em relação à demanda de vôos comerciais também foi utilizada.

Foram obtidos conjuntos de soluções não-dominadas bem distribuídas dentro de uma ampla faixa de valores fornecidos pelas funções objetivos, para todas as instâncias do MDTSP testadas. Isto significa que os melhores conjuntos de soluções não-dominadas obtidos são numerosos e contém soluções significativamente distintas entre si, o que era o objetivo deste trabalho. Para as menores instâncias pôde-se constatar que os melhores conjuntos de soluções não-dominadas gerados pelo A-Team fora o próprio Pareto Ótimo do problema.

8.1. Principais Contribuições

As principais contribuições desta dissertação podem ser resumidas como segue:

- Adaptação do método multi-algorítmico de Times Assíncronos para a resolução de Problemas de Otimização Combinatória Multiobjetivos;
- Proposição de um novo problema, o Multi-Distance Traveling Salesman Problem - MDTSP, como uma generalização do NP-difícil TSP através da existência de k custos entre cada par de cidades;
- Diferentes estratégias para a resolução de Problemas Multiobjetivos através de A-Teams, envolvendo a aplicação de métricas-Lp e, principalmente, de uma nova abordagem - manipulação de conjuntos não-dominados - que possibilita considerar todos os objetivos do problema simultaneamente para a geração de soluções, sem que nenhum seja priorizado ou preterido;
- Estrutura de manipulação de soluções k-dimensionais, bem como algoritmos de inserção e eliminação de soluções em tal estrutura;
- Desenvolvimento de algoritmos capazes de manipular várias matrizes de distância, através da incorporação de conceitos de dominância e decisão de compromisso;

- Método para a avaliação de qualidade entre conjuntos não-dominados (ou conjuntos de soluções multidimensionais);
- Levantamento bibliográfico sobre problemas combinatórios e multiobjetivos e seus métodos de resolução;
- Comprovação da adequabilidade de A-Teams para a resolução de Problemas Multiobjetivos;
- Comprovação da Eficiência em Escala de A-Team aplicado ao MDTSP.

8.2. Possíveis Extensões

Existe um campo de pesquisa muito amplo no que se refere ao método de Times Assíncronos e, especificamente, no que diz respeito à sua aplicação a Problemas Multiobjetivos. Algumas possíveis extensões deste trabalho são:

- Desenvolvimento de um A-Team que possibilite a intervenção em tempo de execução de um Agente de Decisão: um A-Team interativo;
- Aplicação de A-Teams para outros problemas multiobjetivos;
- Desenvolvimento de novos algoritmos para o MDTSP;
- Desenvolvimento de novos fluxos de dados para o MDTSP.
- Incorporação de inteligência mais refinada nos agentes do A-Team, de tal forma que cada um seja capaz de avaliar sua própria performance, aumentando ou diminuindo sua atuação nas memórias compartilhadas (selecionando ou eliminando soluções).

Apêndice A

Limites Inferiores e Matrizes de Distância: Instâncias Geradas Randomicamente

Este apêndice apresenta limites inferiores para as matrizes de distância utilizadas nos testes com os Times Assíncronos desenvolvidos, bem como o menor valor gerado pelo A-Team para cada uma delas.

São listadas, também, as matrizes de distância que compuseram as instâncias de 9, 10 e 50 cidades. Isto para que outros pesquisadores que trabalharem com tal problema, possam comparar seus resultados com os divulgados neste trabalho.

A.1. Limites Inferiores

As tabelas A.1 à A.6 contêm os limites inferiores para cada matrizes de distância utilizada nos testes apresentados no Capítulo 7, envolvendo 9, 10, 50, 100, 200 e 500 cidades. Os melhores limites inferiores, para todas as matrizes de distância, foram obtidos através do algoritmo de Held-Karp (HK). Utilizou-se também o *Assignment Problem* (AP), através do pacote CPLEX, mas os limites inferiores gerados pelo Algoritmo de Held-Karp foram melhores em todos os casos testados. O AP não propicia a geração de bons limites inferiores, a menos que o mesmo seja utilizado em conjunto com algum algoritmo de corte (por exemplo, início de um algoritmo que utilize desigualdades de eliminação de sub-trajetórias - ver Cap.4).

Não foram realizados procedimentos mais sofisticados para a obtenção de limites inferiores porque o objetivo era somente observar o comportamento do A-Team quando aplicado ao MDTSP (em relação à cada matriz de distância) e, principalmente, mostrar que é possível utilizar no MDTSP, os mesmos algoritmos e procedimentos existentes para o cálculo de limites inferiores do TSP.

Em todas as tabelas desta subseção, na primeira coluna é indicado a dimensão da matriz e as respectivas matrizes testadas (as linhas das tabelas indicam qual é a matriz usada). Na segunda coluna, encontram-se os melhores *lower bounds* (limites inferiores) obtidos para cada matriz de distância. Na terceira coluna, está o menor comprimento obtido pelo A-Team em cada matriz.

Tabela A.1: Limite Inferior e menor comprimento gerado pelo A-Team para as matrizes de distância de 9x9 (9-cidades).

9x9	Melhor Limite Inferior obtido	Menor comprimento gerado pelo A-Team
matriz 1	136	205*
matriz 2	184	215*
matriz 3	183	237*
matriz 4	92	108*

* Solução ótima

Tabela A.2: Limite Inferior e menor comprimento gerado pelo A-Team para as matrizes de distância de 10x10 (10-cidades).

10x10	Melhor Limite Inferior obtido	Menor comprimento gerado pelo A-Team
matriz 1	114	161*
matriz 2	116	158*
matriz 3	139	219*
matriz 4	130	189*

* Solução ótima

Tabela A.3: Limite Inferior e menor comprimento gerado pelo A-Team para as matrizes de distância de 50x50 (50-cidades).

50x50	Melhor Limite Inferior obtido	Menor comprimento gerado pelo A-Team
matriz 1	177	278
matriz 2	143	204
matriz 3	196	268

Tabela A.4: Limite Inferior e menor comprimento gerado pelo A-Team para as matrizes de distância de 100x100(100-cidades).

100x100	Melhor Limite Inferior obtido	Menor comprimento gerado pelo A-Team
matriz 1	200	258
matriz 2	204	280
matriz 3	208	293

Tabela A.5: Limite Inferior e menor comprimento gerado pelo A-Team para as matrizes de distância de 200x200 (200-cidades).

200x200	Melhor Limite Inferior obtido	Menor comprimento gerado pelo A-Team
matriz 1	313	383
matriz 2	322	392

Tabela A.6: Limite Inferior e menor comprimento gerado pelo A-Team para as matrizes de distância de 500x500 (500-cidades).

500x500	Melhor Limite Inferior obtido	Menor comprimento gerado pelo A-Team
matriz 1	518	746
matriz 2	529	941

A.2. Matrizes de Distância

Todas as matrizes utilizadas nos testes são completas e simétricas.

São listadas, a seguir, as quatro matrizes utilizadas nos testes computacionais com dimensões 9x9, 10x10 e 50x50.

Tabela A.7: Lista das quatro matrizes de distância relativas a 9-cidades.

Matriz 1

0	30	52	63	70	79	92	21	82
30	0	46	38	2	57	80	20	88
52	46	0	66	79	55	75	55	18
63	38	66	0	13	94	32	6	86
70	2	79	13	0	72	83	97	78
79	57	55	94	72	0	86	3	11
92	80	75	32	83	86	0	88	59
21	20	55	6	97	3	88	0	12
82	88	18	86	78	11	59	12	0

Matriz 2

0	78	34	59	56	74	79	19	31
78	0	96	86	63	45	40	42	12
34	96	0	17	75	12	12	78	83
59	86	17	0	53	78	5	51	39
56	63	75	53	0	81	56	45	34
74	45	12	78	81	0	94	20	80
79	40	12	5	56	94	0	42	70
19	42	78	51	45	20	42	0	88
31	12	83	39	34	80	70	88	0

Matriz 3

0	40	46	80	96	71	44	67	18
40	0	59	72	35	11	60	82	78
46	59	0	30	96	5	39	95	34
80	72	30	0	83	17	59	60	53
96	35	96	83	0	88	85	79	26
71	11	5	17	88	0	26	58	32
44	60	39	59	85	26	0	8	79
67	82	95	60	79	58	8	0	71
18	78	34	53	26	32	79	71	0

Matriz 4

0	97	43	36	42	1	80	37	24
97	0	48	85	10	34	22	10	4
43	48	0	18	81	23	59	96	71
36	85	18	0	10	72	94	29	87
42	10	81	10	0	8	11	55	93
1	34	23	72	8	0	51	66	25
80	22	59	94	11	51	0	7	59
37	10	96	29	55	66	7	0	92
24	4	71	87	93	25	59	92	0

Tabela A.8: Lista das quatro matrizes de distância relativas a 10-cidades.

Matriz 1

0	97	21	92	9	58	81	13	2	24
97	0	56	31	52	100	67	90	45	11
21	56	0	9	39	58	17	94	25	60
92	31	9	0	60	72	15	59	9	29
9	52	39	60	0	73	30	22	80	11
58	100	58	72	73	0	10	90	90	16
81	67	17	15	30	10	0	54	66	73
13	90	94	59	22	90	54	0	64	34
2	45	25	9	80	90	66	64	0	18
24	11	60	29	11	16	73	34	18	0

Matriz 2

0	20	43	36	87	51	11	10	74	49
20	0	29	73	84	40	49	25	70	54
43	29	0	93	93	93	17	55	2	69
36	73	93	0	79	2	29	17	79	12
87	84	93	79	0	36	6	26	11	28
51	40	93	2	36	0	75	45	48	64
11	49	17	29	6	75	0	49	5	47
10	25	55	17	26	45	49	0	94	38
74	70	2	79	11	48	5	94	0	26
49	54	69	12	28	64	47	38	26	0

Matriz 3

0	53	81	52	46	79	93	1	32	53
53	0	37	27	48	42	91	74	12	97
81	37	0	21	48	63	73	15	50	66
52	27	21	0	73	89	75	7	24	75
46	48	48	73	0	90	24	93	51	39
79	42	63	89	90	0	11	17	53	63
93	91	73	75	24	11	0	2	46	11
1	74	15	7	93	17	2	0	68	55
32	12	50	24	51	53	46	68	0	67
53	97	66	75	39	63	11	55	67	0

Matriz 4

0	7	21	42	12	56	2	19	86	79
7	0	46	49	68	42	65	19	49	91
21	46	0	57	5	15	4	92	26	88
42	49	57	0	12	36	96	47	2	22
12	68	5	12	0	80	78	82	67	68
56	42	15	36	80	0	78	43	45	50
2	65	4	96	78	78	0	66	50	69
19	19	92	47	82	43	66	0	95	84
86	49	26	2	67	45	50	95	0	45
79	91	88	22	68	50	69	84	45	0

Tabela A.9: Lista das quatro matrizes de distância relativas à 50-cidades.

Matriz 1

0 26 100 44 85 33 35 53 8 64 60 31 14 41 55 59 36 63 55 30 79 65 7 93 44 25 73 63 63 55 70 92 94 85 38 71 90 81 43 37 87 51 83 21 96 45 40 44 17 41
26 0 53 91 9 97 74 82 8 55 91 11 35 100 98 32 27 63 27 71 44 57 83 39 59 78 95 32 89 53 61 98 37 60 19 39 90 24 2 98 81 82 89 30 29 61 68 30 21 15
100 53 0 19 60 68 37 31 69 54 39 91 22 99 82 39 58 24 77 49 55 35 27 40 19 56 87 53 3 85 18 100 4 47 34 18 25 82 22 57 46 85 64 96 66 37 85 15 12 95
44 91 19 0 85 81 78 33 83 66 11 35 34 49 91 9 67 34 72 60 46 4 80 3 52 58 51 26 81 90 90 68 28 41 46 9 73 49 69 17 2 11 94 50 48 39 10 86 8 63
85 9 60 85 0 29 77 76 24 9 70 23 87 44 70 2 22 39 10 37 73 18 97 67 47 81 78 87 94 74 82 85 21 60 73 14 2 73 95 47 42 26 64 65 19 89 31 100 84 44
33 97 68 81 29 0 16 66 28 93 63 44 58 88 77 93 43 20 46 89 43 87 91 58 35 30 67 53 66 30 34 27 18 50 23 39 81 29 5 40 36 21 59 33 96 16 22 5 19 27
35 74 37 78 77 16 0 82 29 40 1 97 34 2 25 5 62 10 79 21 22 46 75 75 72 52 7 77 11 62 68 26 73 10 29 30 1 17 16 12 47 52 16 62 12 52 55 55 93 51
53 82 31 33 76 66 82 0 52 94 77 69 73 38 98 91 29 76 89 24 89 18 78 97 23 69 62 86 19 95 71 64 58 25 6 46 13 1 20 24 31 83 46 98 77 16 95 21 13 36
8 8 69 83 24 28 29 52 0 96 70 64 83 44 88 71 77 47 20 98 58 40 21 92 49 77 27 3 26 53 96 76 23 25 38 78 85 2 62 26 73 84 71 60 41 64 38 78 54 13
64 55 54 66 9 93 40 94 96 0 58 60 21 95 70 13 62 35 27 11 10 51 89 23 82 33 66 74 57 78 60 50 71 81 54 70 98 96 80 84 44 10 67 89 92 89 78 52 96 81
60 91 39 11 70 63 1 77 70 58 0 62 72 93 3 84 54 29 90 16 72 14 22 5 96 2 4 55 72 72 30 95 81 80 21 36 90 69 48 69 60 2 56 48 87 62 43 65 18 59
31 11 91 35 23 44 97 69 64 60 62 0 69 2 8 81 54 5 13 38 89 24 64 57 6 45 77 27 72 36 46 98 65 48 100 76 26 1 36 9 40 48 11 68 41 32 21 94 64 63
14 35 22 34 87 58 34 73 83 21 72 69 0 27 13 6 35 20 88 38 14 70 81 81 12 74 50 42 40 11 60 64 77 43 12 12 56 96 92 83 77 53 88 25 99 90 62 69 59 25
41 100 99 49 44 88 2 38 44 95 93 2 27 0 53 90 19 73 73 96 28 19 44 59 9 85 45 84 83 38 91 24 72 9 79 4 24 89 79 66 76 60 15 48 33 30 95 47 26 53
55 98 82 91 70 77 25 98 88 70 3 8 13 53 0 20 52 46 24 93 96 2 23 52 66 28 71 85 3 36 73 42 8 72 29 76 23 73 75 68 75 97 80 79 43 36 12 86 28 61
59 32 39 9 2 93 5 91 71 13 84 81 6 90 20 0 42 6 80 66 89 23 20 46 20 92 41 26 25 38 91 33 4 85 29 61 72 11 38 28 71 35 97 80 80 96 22 67 41 88
36 27 58 67 22 43 62 29 77 62 54 54 35 19 52 42 0 90 97 54 79 63 61 48 49 17 82 40 25 77 69 19 36 3 31 98 29 85 39 74 64 86 79 71 51 21 81 32 26 77
63 63 24 34 39 20 10 76 47 35 29 5 20 73 46 6 90 0 56 39 99 24 31 67 69 93 17 61 48 68 24 44 7 65 18 66 61 7 78 85 32 37 13 26 94 96 97 93 52 52
55 27 77 72 10 46 79 89 20 27 90 13 88 73 24 80 97 56 0 14 39 54 55 15 39 8 54 55 40 7 52 97 26 75 56 77 92 39 32 87 56 18 53 54 25 89 87 100 28 94
30 71 49 60 37 89 21 24 98 11 16 38 38 96 93 66 54 39 14 0 84 74 54 67 85 35 83 99 5 75 73 45 25 26 56 53 80 66 95 72 4 69 38 52 61 23 2 43 16 5
79 44 55 46 73 43 22 89 58 10 72 89 14 28 96 89 79 99 39 84 0 6 23 66 20 95 39 69 59 36 21 99 78 57 28 58 84 51 67 53 64 70 19 64 5 4 10 92 88 11
63 57 35 4 18 87 46 18 40 51 14 24 70 19 2 23 63 24 54 74 6 0 36 78 59 46 53 92 29 7 79 81 70 94 14 62 53 29 35 89 8 93 8 75 7 13 7 23 87 98
7 83 27 80 97 91 75 78 21 89 22 64 81 44 23 20 61 31 55 54 23 36 0 94 80 95 97 9 88 75 83 75 5 29 50 19 11 71 25 51 11 68 36 49 59 37 88 27 6 59
93 39 40 3 67 58 75 97 92 23 5 57 81 59 52 46 48 67 15 67 66 78 94 0 38 35 33 3 59 71 1 53 34 44 80 18 66 22 47 14 17 25 94 60 62 51 79 63 88 53
44 59 19 52 47 35 72 23 49 82 96 6 12 9 66 20 49 69 39 85 20 59 80 38 0 61 69 77 14 71 97 37 25 50 20 52 4 17 31 42 12 61 20 55 64 90 93 95 92 50
25 78 56 58 81 30 52 69 77 33 2 45 74 85 28 92 17 93 8 35 95 46 95 35 61 0 46 62 34 51 69 31 24 52 51 62 6 62 41 31 54 3 45 2 23 78 20 92 11 42
73 95 87 51 78 67 7 62 27 66 4 77 50 45 71 41 82 17 54 83 39 53 97 33 69 46 0 67 37 56 68 16 68 43 54 49 45 41 80 26 13 44 65 61 94 86 69 72 59 42
63 32 53 26 87 53 77 86 3 74 55 27 42 84 85 26 40 61 55 99 69 92 9 3 77 62 67 0 37 70 23 12 20 10 91 67 100 95 47 92 79 16 66 93 20 21 96 74 34 26
63 89 3 81 94 66 11 19 26 57 72 72 40 83 3 25 25 48 40 5 59 29 88 59 14 34 37 37 0 68 94 85 44 72 17 16 26 73 16 85 90 41 22 11 26 30 35 28 68 11
55 53 85 90 74 30 62 95 53 78 72 36 11 38 36 38 77 68 7 75 36 7 75 71 71 51 56 70 68 0 27 100 39 47 47 7 54 63 16 14 56 93 77 35 30 2 49 19 64 58
70 61 18 90 82 34 68 71 96 60 30 46 60 91 73 91 69 24 52 73 21 79 83 1 97 69 68 23 94 27 0 23 97 16 31 98 11 78 42 50 12 58 78 1 69 90 41 34 48 28
92 98 100 68 85 27 26 64 76 50 95 98 64 24 42 33 19 44 97 45 99 81 75 53 37 31 16 12 85 100 23 0 96 63 3 73 44 16 8 10 70 82 51 9 60 33 2 50 30 44
94 37 4 28 21 18 73 58 23 71 81 65 77 72 8 4 36 7 26 25 78 70 5 34 25 24 68 20 44 39 97 96 0 72 78 7 90 95 15 100 27 79 59 9 86 3 23 99 40 90
85 60 47 41 60 50 10 25 25 81 80 48 43 9 72 83 3 65 75 26 57 94 29 44 50 52 43 10 72 47 16 63 72 0 67 14 80 24 88 74 94 30 71 49 85 30 71 74 85 51
38 19 34 46 73 23 29 6 38 54 21 100 12 79 29 29 31 18 56 56 28 14 50 80 20 51 54 91 17 47 31 3 78 67 0 62 68 84 77 37 79 87 36 67 75 69 14 78 18 28
71 39 18 9 14 39 30 46 78 70 36 76 12 4 76 61 98 66 77 53 58 62 19 18 52 62 49 67 16 7 98 73 7 14 62 0 26 59 46 99 33 2 67 12 7 35 72 19 68 70
90 90 25 73 2 81 1 13 85 98 90 26 56 24 23 72 29 61 92 80 84 53 11 66 4 6 45 100 26 54 11 44 90 80 68 26 0 11 83 60 80 3 31 91 23 66 94 32 98 72
81 24 82 49 73 29 17 1 2 96 69 1 96 89 73 11 85 7 39 66 51 29 71 22 17 62 41 95 73 63 78 16 95 24 84 59 11 0 26 15 27 28 69 45 98 24 94 32 8 63
43 2 22 69 95 5 16 20 62 80 48 36 92 79 75 38 39 78 32 95 67 35 25 47 31 41 80 47 16 16 42 8 15 88 77 46 83 26 0 68 91 38 58 48 71 93 32 45 36 1
37 98 57 17 47 40 12 24 26 84 69 9 83 66 68 28 74 85 87 72 53 89 51 14 42 31 26 92 85 14 50 10 100 74 37 99 60 15 68 0 88 62 19 36 40 50 87 44 54 97
87 81 46 2 42 36 47 31 73 44 60 40 77 76 75 71 64 32 56 4 64 8 11 17 12 54 13 79 90 56 12 70 27 94 79 33 80 27 91 88 0 22 61 85 98 25 81 90 89 39
51 82 85 11 26 21 52 83 84 10 2 48 53 60 97 35 86 37 18 69 70 93 68 25 61 3 44 16 41 93 58 82 79 30 87 2 3 28 38 62 22 0 40 55 91 65 1 5 33 6
83 89 64 94 64 59 16 46 71 67 56 11 88 15 80 97 79 13 53 38 19 8 36 94 20 45 65 66 22 77 78 51 59 71 36 67 31 69 58 19 61 40 0 46 83 77 48 74 43 20
21 30 96 50 65 33 62 98 60 89 48 68 25 48 79 80 71 26 54 52 64 75 49 60 55 2 61 93 11 35 1 9 9 49 67 12 91 45 48 36 85 55 46 0 82 43 21 76 97 33
96 29 66 48 19 96 12 77 41 92 87 41 99 33 43 80 51 94 25 61 5 7 59 62 64 23 94 20 26 30 69 60 86 85 75 7 23 98 71 40 98 91 83 82 0 31 50 82 81 57
45 61 37 39 89 16 52 16 64 89 62 32 90 30 36 96 21 96 89 23 4 13 37 51 90 78 86 21 30 2 90 33 3 30 69 35 66 24 93 50 25 65 77 43 31 0 34 20 74 52
40 68 85 10 31 22 55 95 38 78 43 21 62 95 12 22 81 97 87 2 10 7 88 79 93 20 69 96 35 49 41 2 23 71 14 72 94 94 32 87 81 1 48 21 50 34 0 61 70 41
44 30 15 86 100 5 55 21 78 52 65 94 69 47 86 67 32 93 100 43 92 23 27 63 95 92 72 74 28 19 34 50 99 74 78 19 32 32 45 44 90 5 74 76 82 20 61 0 90 82
17 21 12 8 84 19 93 13 54 96 18 64 59 26 28 41 26 52 28 16 88 87 6 88 92 11 59 34 68 64 48 30 40 85 18 68 98 8 36 54 89 33 43 97 81 74 70 90 0 34
41 15 95 63 44 27 51 36 13 81 59 63 25 53 61 88 77 52 94 5 11 98 59 53 50 42 42 26 11 58 28 44 90 51 28 70 72 63 1 97 39 6 20 33 57 52 41 82 34 0

Matriz 2

0 61 52 13 19 28 43 1 18 3 71 83 24 41 57 38 24 33 75 75 8 49 57 24 35 8 64 75 78 14 89 78 90 72 79 85 62 27 42 93 33 99 95 62 83 19 6 78 38 48
61 0 35 51 77 12 58 20 47 75 23 91 48 39 7 47 1 92 48 49 20 58 20 91 69 42 83 28 83 96 32 8 45 11 3 51 6 52 13 82 85 46 12 36 89 57 37 30 56 8
52 35 0 80 27 75 83 42 98 42 78 98 32 2 31 65 71 88 9 89 94 28 91 54 58 42 62 25 14 77 38 5 31 60 49 39 67 41 96 53 82 79 3 18 4 48 17 40 22 35
13 51 80 0 76 29 10 93 54 22 47 51 50 19 65 82 79 31 10 30 80 36 6 55 36 100 33 3 11 50 73 21 71 79 68 23 44 31 5 26 100 87 53 7 10 29 42 89 80 71
19 77 27 76 0 61 78 76 40 98 18 5 70 72 82 86 59 53 99 78 94 43 94 91 48 36 36 92 98 23 84 72 6 37 83 26 85 40 30 15 12 91 90 41 53 87 42 88 31 92
28 12 75 29 61 0 34 55 38 48 94 52 4 36 38 49 85 22 22 65 32 45 45 6 86 80 92 50 38 72 44 14 11 13 35 64 44 91 66 94 56 20 50 20 31 47 20 25 96 2
43 58 83 10 78 34 0 18 50 59 78 22 33 18 78 70 17 95 42 66 59 81 33 64 63 94 98 56 24 12 73 5 3 80 5 19 77 96 92 79 65 23 15 9 9 20 56 28 8 4
1 20 42 93 76 55 18 0 78 9 34 90 43 82 80 16 52 33 37 2 64 49 91 33 34 45 69 83 91 86 40 88 98 97 82 22 84 2 74 20 23 11 30 89 22 26 3 47 17 51
18 47 98 54 40 38 50 78 0 41 36 32 99 63 85 92 16 49 80 86 14 24 60 65 37 56 17 6 56 15 18 46 50 83 3 60 40 78 12 59 77 83 25 59 72 2 35 25 38 69
3 75 42 22 98 48 59 9 41 0 41 59 62 72 25 39 71 83 71 98 85 77 5 51 72 64 11 14 32 88 35 21 78 92 20 9 5 58 43 68 95 83 24 33 55 58 70 40 79 23
71 23 78 47 18 94 78 34 36 41 0 15 48 75 51 29 48 47 47 33 2 9 54 69 26 37 91 59 85 73 24 99 67 92 82 90 1 31 39 28 59 91 74 55 59 88 74 22 66 10
83 91 98 51 5 52 22 90 32 59 15 0 66 46 30 60 28 69 93 72 32 81 71 67 47 54 41 96 5 72 94 13 39 83 59 14 68 88 73 25 25 33 38 22 59 8 66 9 35 61
24 48 32 50 70 4 33 43 99 62 48 66 0 76 91 97 26 81 42 57 63 50 37 91 47 37 75 2 30 33 29 51 25 35 2 40 95 23 19 79 14 31 12 93 73 12 39 34 10 96
41 39 2 19 72 36 18 82 63 72 75 46 76 0 39 36 93 34 10 16 99 32 76 52 77 84 100 18 12 47 10 5 79 3 79 12 59 49 38 81 53 93 80 39 2 20 92 93 46 34

57 7 31 65 82 38 78 80 85 25 51 30 91 39 0 26 48 90 37 98 25 2 80 53 28 75 89 87 79 5 12 78 52 36 15 81 89 50 56 83 84 50 7 79 99 36 88 79 4 95
38 47 65 82 86 49 70 16 92 39 29 60 97 36 26 0 81 47 5 60 10 64 87 53 28 67 88 3 45 64 85 26 10 63 15 86 89 36 2 25 14 80 9 88 18 53 29 69 59 64
24 1 71 79 59 85 17 52 16 71 48 28 26 93 48 81 0 37 39 23 22 8 52 68 49 52 50 100 21 36 44 47 61 44 83 48 57 88 11 36 75 45 3 82 35 63 74 68 92 44
33 92 88 31 53 22 95 33 49 83 47 69 81 34 90 47 37 0 71 76 1 23 90 21 41 84 75 92 48 9 66 59 30 72 89 6 49 92 11 48 41 85 14 27 92 35 28 28 63 14
75 48 9 10 99 22 42 37 80 71 47 93 42 10 37 5 39 71 0 81 57 2 94 74 4 26 22 3 6 31 6 11 13 55 20 55 4 37 8 4 32 98 91 38 90 40 48 21 39 26
75 49 89 30 78 65 66 2 86 98 33 72 57 16 98 60 23 76 81 0 98 8 15 8 67 52 49 95 29 89 25 33 39 7 75 93 80 11 34 56 43 6 31 52 53 13 22 67 93 53
8 20 94 80 94 32 59 64 14 85 2 32 63 99 25 10 22 1 57 98 0 25 7 84 47 64 11 18 97 60 15 30 1 88 67 5 10 93 39 60 92 26 23 18 99 63 80 71 9 49
49 58 28 36 43 45 81 49 24 77 9 81 50 32 2 64 8 23 2 8 25 0 71 48 45 16 12 60 77 11 43 50 83 57 48 18 65 36 94 70 12 41 86 61 75 63 51 23 54 78
57 20 91 6 94 45 33 91 60 5 54 71 37 76 80 87 52 90 94 15 7 71 0 23 57 57 45 37 7 54 11 45 70 4 35 88 87 81 89 16 93 75 69 7 13 94 67 23 50 41
24 91 54 55 91 6 64 33 65 51 69 67 91 52 53 53 68 21 74 8 84 48 23 0 9 60 100 16 68 13 67 29 64 25 92 77 84 92 68 79 98 100 8 41 28 22 75 76 11 32
35 69 58 36 48 86 63 34 37 72 26 47 47 77 28 28 49 41 4 67 47 45 57 9 0 67 76 13 3 62 70 53 22 98 98 19 11 55 39 73 14 11 61 72 6 41 54 22 89 76
8 42 42 100 36 80 94 45 56 64 37 54 37 84 75 67 52 84 26 52 64 16 57 60 67 0 5 20 99 38 85 76 21 41 6 38 59 7 58 75 96 47 28 21 77 64 6 37 42 71
64 83 62 33 36 92 98 69 17 11 91 41 75 100 89 88 50 75 22 49 11 12 45 100 76 5 0 81 100 51 44 99 73 25 24 4 36 35 34 10 37 61 22 46 6 5 2 53 30 16
75 28 25 3 92 50 56 83 6 14 59 96 2 18 87 3 100 92 3 95 18 60 37 16 13 20 81 0 99 3 20 57 31 28 15 7 63 51 92 66 1 53 25 3 22 85 67 8 36 98
78 83 14 11 98 38 24 91 56 32 85 5 30 12 79 45 21 48 6 29 97 77 7 68 3 99 100 99 0 12 70 57 7 86 58 99 34 38 52 99 2 23 93 32 18 9 62 90 87 23
14 96 77 50 23 72 12 86 15 88 73 72 33 47 5 64 36 9 31 89 60 11 54 13 62 38 51 3 12 0 80 25 46 34 34 97 21 23 6 54 44 32 18 34 65 11 65 55 87 12
89 32 38 73 84 44 73 40 18 35 24 94 29 10 12 85 44 66 6 25 15 43 11 67 70 85 44 20 70 80 0 27 17 2 47 55 40 58 16 80 61 2 47 40 94 47 84 93 55 42
78 8 5 21 72 14 5 88 46 21 99 13 51 5 78 26 47 59 11 33 30 50 45 29 53 76 99 57 57 25 27 0 71 67 89 40 48 26 22 38 48 43 83 42 79 19 6 55 23 98
90 45 31 71 6 11 3 98 50 78 67 39 25 79 52 10 61 30 13 39 1 83 70 64 22 21 73 31 7 46 17 71 0 29 12 40 2 46 60 73 95 47 94 50 83 33 71 89 85 54
72 11 60 79 37 13 80 97 83 92 92 83 35 3 36 63 44 72 55 7 88 57 4 25 98 41 25 28 86 34 2 67 29 0 57 3 95 22 40 33 25 53 97 50 4 30 69 41 37 79
79 3 49 68 83 35 5 82 3 20 82 59 2 79 15 15 83 89 20 75 67 48 35 92 98 6 24 15 58 34 47 89 12 57 0 47 3 53 78 62 22 99 43 39 21 73 58 67 54 42
85 51 39 23 26 64 19 22 60 9 90 14 40 12 81 86 48 6 55 93 5 18 88 77 19 38 4 7 99 97 55 40 40 3 47 0 27 94 46 18 68 96 14 20 70 6 67 89 12 44
62 6 67 44 85 44 77 84 40 5 1 68 93 59 89 89 57 49 4 80 10 65 87 84 11 59 36 63 34 21 40 48 2 95 3 27 0 57 47 38 29 13 59 29 86 16 61 34 72 52
27 52 41 31 40 91 96 2 78 58 31 88 23 49 50 36 88 92 37 11 93 36 81 92 55 7 35 51 38 23 58 26 46 22 53 94 57 0 42 56 63 30 15 78 53 25 5 52 39 56
42 13 96 5 30 66 92 74 12 43 39 73 19 38 56 2 11 11 8 34 39 94 89 68 39 58 34 92 52 6 16 22 60 40 78 46 47 42 0 78 8 53 11 93 51 19 27 85 79 23
93 82 53 26 15 94 79 20 59 68 28 25 79 81 83 25 36 48 4 56 60 70 16 79 73 75 10 66 99 54 80 38 73 33 62 18 38 56 78 0 41 10 25 59 63 16 4 77 79 96
33 85 82 100 12 56 65 23 77 95 59 25 14 53 84 14 75 41 32 43 92 12 93 98 14 96 37 1 2 44 61 48 95 25 22 68 29 63 8 41 0 40 34 13 98 30 68 100 8 88
99 46 79 87 91 20 23 11 83 83 91 33 31 93 50 80 45 85 98 6 36 41 75 100 11 47 61 53 23 32 2 43 47 53 99 96 13 30 53 10 40 0 51 59 76 14 95 86 7 66
95 12 3 53 90 50 15 30 25 24 74 38 12 80 7 9 3 14 91 31 23 86 69 8 61 28 22 25 93 18 47 83 94 97 43 14 59 15 11 25 34 51 0 71 19 38 18 53 51 38
62 36 18 7 41 20 9 89 59 33 55 22 93 39 79 88 82 27 38 52 18 61 7 41 72 21 46 3 32 34 40 42 50 50 39 20 29 78 93 59 15 59 71 0 44 87 70 47 21 54
83 89 4 10 53 31 9 22 72 55 59 59 73 2 99 18 35 92 90 53 99 75 13 28 6 77 6 22 18 65 94 79 83 4 21 70 86 53 51 63 98 76 19 44 0 47 100 61 43 70
19 57 48 29 87 47 20 26 2 58 88 8 12 20 36 53 63 35 40 13 63 63 94 22 41 64 5 85 9 11 47 19 33 30 73 6 16 25 19 16 30 14 38 87 47 0 10 81 21 52
6 37 17 42 42 20 56 3 35 70 74 66 39 92 88 29 74 28 48 22 80 51 67 75 54 6 2 67 62 65 84 6 71 69 58 67 61 5 27 4 68 95 18 70 100 10 0 55 2 10
78 30 40 89 88 25 28 47 25 40 22 9 34 93 79 69 68 28 21 67 71 23 23 76 22 37 53 8 90 55 93 55 89 41 67 89 34 52 85 77 100 86 55 47 61 81 55 0 34
88
38 56 22 80 31 96 8 17 38 79 66 35 10 46 4 59 92 63 39 93 9 54 50 11 89 42 30 36 87 87 55 23 85 37 54 12 72 39 79 79 8 7 51 21 43 21 2 34 0 70
48 8 35 71 92 2 4 51 69 23 10 61 96 34 95 64 44 14 26 53 49 78 41 32 76 71 16 90 23 12 42 98 54 79 42 44 52 56 23 96 88 66 38 54 70 52 10 88 70 0

Matriz 3

0 36 93 60 12 25 86 99 42 78 95 65 67 21 18 49 99 81 4 93 20 82 77 17 71 80 40 80 64 38 18 32 49 7 56 91 51 45 61 77 32 78 21 39 18 87 92 12 46 71
36 0 87 34 26 57 12 95 22 82 56 64 73 54 10 34 11 4 76 41 70 39 74 93 53 48 79 86 41 54 21 11 67 71 36 36 52 63 38 55 27 52 2 79 34 94 85 11 90 45
93 87 0 85 14 98 81 27 30 57 55 22 57 62 30 96 36 93 81 26 30 85 18 80 54 16 32 14 58 73 67 86 42 85 54 67 5 4 46 11 96 18 78 27 38 52 29 30 46 91
60 34 85 0 13 67 23 38 4 83 62 77 16 7 35 31 64 69 25 17 54 69 56 96 10 37 85 13 62 15 87 81 49 35 17 8 32 63 59 9 99 39 17 69 34 84 15 71 49 34
12 26 14 13 0 54 19 37 6 93 56 76 63 84 87 1 74 98 16 15 24 14 14 21 68 97 98 55 20 62 45 7 80 8 67 31 99 46 43 62 99 97 41 51 67 7 46 24 29 71
25 57 98 67 54 0 1 50 62 30 87 75 22 35 22 32 2 23 33 17 68 87 7 84 47 61 43 29 7 49 28 29 8 78 40 15 9 17 91 16 24 59 7 14 84 39 79 92 87 71
86 12 81 23 19 1 0 33 39 7 10 52 72 5 80 76 79 89 7 24 34 36 36 80 19 72 55 68 69 12 35 76 35 48 35 35 67 8 23 66 92 52 55 28 8 7 76 77 73 64
99 95 27 38 37 50 33 0 69 14 77 98 91 40 93 26 61 95 95 72 94 62 76 48 98 36 51 42 21 43 48 98 34 66 14 91 33 61 96 98 59 81 63 85 99 90 45 97 97
76
42 22 30 4 6 62 39 69 0 58 82 20 66 31 44 1 52 89 5 62 75 57 95 14 72 27 72 67 87 39 86 93 61 8 68 92 22 9 12 92 39 23 27 98 6 56 13 63 51 91
78 82 57 83 93 30 7 14 58 0 54 99 20 43 86 10 55 3 50 93 55 9 70 82 67 17 89 29 2 13 5 9 21 96 45 63 27 43 8 61 95 33 65 90 60 7 28 94 73 60
95 56 55 62 56 87 10 77 82 54 0 95 97 68 11 46 6 54 70 40 93 8 87 56 58 90 47 81 51 34 81 21 42 77 86 33 25 16 95 31 78 47 21 78 47 19 66 5 26 49
65 64 22 77 76 75 52 98 20 99 95 0 65 84 19 12 57 55 6 65 29 84 35 50 84 18 85 64 38 26 94 98 87 77 3 88 5 63 9 51 98 66 69 23 66 77 64 34 83 81
67 73 57 16 63 22 72 91 66 20 97 65 0 17 63 94 22 86 82 5 82 82 99 36 81 43 85 45 66 2 96 7 44 12 38 50 48 54 29 18 28 2 40 80 22 100 49 20 50 6
21 54 62 7 84 35 5 40 31 43 68 84 17 0 34 14 24 58 4 24 48 56 50 69 25 84 62 31 62 11 37 77 23 80 79 55 54 73 21 87 4 87 26 95 88 76 11 72 94 7
18 10 30 35 87 22 80 93 44 86 11 19 63 34 0 29 6 68 42 100 36 62 83 53 13 54 97 68 69 13 67 92 89 81 49 4 16 60 10 9 48 51 96 79 61 75 78 17 94 9
49 34 96 31 1 32 76 26 1 10 46 12 94 14 29 0 37 84 35 78 19 100 94 76 11 57 47 73 54 75 83 43 53 55 49 16 17 46 27 83 31 19 13 72 73 15 92 31 7 94
99 11 36 64 74 2 79 61 52 55 6 57 22 24 6 37 0 75 17 30 58 86 67 95 9 86 18 44 79 98 14 98 11 20 5 87 29 10 59 41 20 88 71 87 88 60 12 83 40 10
81 4 93 69 98 23 89 95 89 3 54 55 86 58 68 84 75 0 77 31 81 82 94 62 49 21 19 25 88 45 43 85 98 35 98 82 44 87 84 33 85 64 14 66 91 10 100 82 27
18
4 76 81 25 16 33 7 95 5 50 70 6 82 4 42 35 17 77 0 68 24 60 70 19 89 92 69 61 13 81 87 36 8 87 85 86 87 17 57 50 82 71 66 31 16 40 91 29 63 98
93 41 26 17 15 17 24 72 62 93 40 65 5 24 100 78 30 31 68 0 5 21 39 84 79 59 36 74 39 75 85 29 64 20 23 29 99 48 10 89 98 1 87 32 10 88 11 37 5 53
20 70 30 54 24 68 34 94 75 55 93 29 82 48 36 19 58 81 24 5 0 72 80 71 38 32 59 37 72 89 13 61 10 41 62 63 60 31 27 83 84 48 4 59 57 87 9 83 97 64
82 39 85 69 14 87 36 62 57 9 8 84 82 56 62 100 86 82 60 21 72 0 6 96 59 23 94 86 17 13 48 57 29 23 41 100 89 38 77 3 73 21 98 96 42 24 9 24 40 11
77 74 18 56 14 7 36 76 95 70 87 35 99 50 83 94 67 94 70 39 80 6 0 14 48 42 4 68 93 85 62 73 36 93 28 76 38 6 95 61 87 19 63 10 73 77 19 61 27 94
17 93 80 96 21 84 80 48 14 82 56 50 36 69 53 76 95 62 19 84 71 96 14 0 96 70 87 61 92 69 74 59 97 38 18 22 71 41 77 65 71 51 92 73 54 70 34 61 80
43
71 53 54 10 68 47 19 98 72 67 58 84 81 25 13 11 9 49 89 79 38 59 48 96 0 90 39 7 18 98 19 20 21 19 10 25 33 90 42 86 55 30 18 59 3 62 38 53 48 86
80 48 16 37 97 61 72 36 27 17 90 18 43 84 54 57 86 21 92 59 32 23 42 70 90 0 8 83 6 72 72 15 99 96 67 48 60 65 25 93 93 25 2 69 79 79 100 75 35 63
40 79 32 85 98 43 55 51 72 89 47 85 85 62 97 47 18 19 69 36 59 94 4 87 39 8 0 67 85 88 73 53 94 17 12 85 53 73 75 4 8 47 84 1 70 71 55 86 58 6
80 86 14 13 55 29 68 42 67 29 81 64 45 31 68 73 44 25 61 74 37 86 68 61 7 83 67 0 5 36 98 75 75 35 25 61 88 14 71 16 30 28 88 46 62 12 16 80 96 56
64 41 58 62 20 7 69 21 87 2 51 38 66 62 69 54 79 88 13 39 72 17 93 92 18 6 85 5 0 9 99 65 32 72 55 93 18 83 26 5 6 17 3 20 74 99 92 55 16 48
38 54 73 15 62 49 12 43 39 13 34 26 2 11 13 75 98 45 81 75 89 13 85 69 98 72 88 36 9 0 2 94 88 68 49 54 27 7 62 89 20 47 34 53 39 74 51 70 57 92
18 21 67 87 45 28 35 48 86 5 81 94 96 37 67 83 14 43 87 85 13 48 62 74 19 72 73 98 99 2 0 88 64 16 92 78 12 31 26 12 81 39 93 95 64 11 62 17 38 48
32 11 86 81 7 29 76 98 93 9 21 98 7 77 92 43 98 85 36 29 61 57 73 59 20 15 53 75 65 94 88 0 80 9 63 46 29 89 8 49 98 87 96 95 87 92 67 57 80 21
49 67 42 49 80 8 35 34 61 21 42 87 44 23 89 53 11 98 8 64 10 29 36 97 21 99 94 75 32 88 64 80 0 12 6 93 86 7 19 2 63 54 49 46 82 64 90 66 83 99
7 71 85 35 8 78 48 66 8 96 77 77 12 80 81 55 20 35 87 20 41 23 93 38 19 96 17 35 72 68 16 9 12 0 14 18 2 80 1 96 17 2 8 11 51 69 88 48 17 51
56 36 54 17 67 40 35 14 68 45 86 3 38 79 49 49 5 98 85 23 62 41 28 18 10 67 12 25 55 49 92 63 6 14 0 21 76 81 98 33 38 84 65 87 78 55 36 23 49 48
91 36 67 8 31 15 35 91 92 63 33 88 50 55 4 16 87 82 86 29 63 100 76 22 25 48 85 61 93 54 78 46 93 18 21 0 8 28 46 6 23 55 92 83 16 54 6 19 49 73
51 52 5 32 99 9 67 33 22 27 25 5 48 54 16 17 29 44 87 99 60 89 38 71 33 60 53 88 18 27 12 29 86 2 76 8 0 95 33 89 9 96 70 23 52 35 9 54 62 44
45 63 4 63 46 17 8 61 9 43 16 63 54 73 60 46 10 87 17 48 31 38 6 41 90 65 73 14 83 7 31 89 7 80 81 28 95 0 68 13 97 70 72 52 15 5 69 100 91 93
61 38 46 59 43 91 23 96 12 8 95 9 29 21 10 27 59 84 57 10 27 77 95 77 42 25 75 71 26 62 26 8 19 1 98 46 33 68 0 83 93 60 13 72 23 29 85 22 77 11
77 55 11 9 62 16 66 98 92 61 31 51 18 87 9 83 41 33 50 89 83 3 61 65 86 93 4 16 5 89 12 49 2 96 33 6 89 13 83 0 3 9 67 8 72 22 70 11 69 95
32 27 96 99 99 24 92 59 39 95 78 98 28 4 48 51 20 85 82 98 84 73 87 71 55 93 8 30 6 20 81 98 63 17 38 23 9 97 93 3 0 8 23 63 58 12 42 64 15 69
78 52 18 39 97 59 52 81 23 33 47 66 2 87 51 19 88 64 71 1 48 21 19 51 30 25 47 28 17 47 39 87 54 2 84 55 96 70 60 9 8 0 56 1 50 33 53 60 75 90
21 2 78 17 41 7 55 63 27 65 21 69 40 26 96 13 71 14 66 87 4 98 63 92 18 2 84 88 3 34 93 96 49 8 65 92 70 72 13 67 23 56 0 21 52 82 65 45 94 6
39 79 27 69 51 14 28 85 98 90 78 23 80 95 79 72 87 66 31 32 59 96 10 73 39 69 1 46 20 53 95 95 46 11 87 83 23 52 72 8 63 1 21 0 29 26 89 75 10 3

18 34 38 34 67 84 8 99 6 60 47 66 22 88 61 73 88 91 16 10 57 42 73 54 3 79 70 62 74 39 64 87 82 51 78 16 52 15 23 72 58 50 52 29 0 54 80 32 8 95
87 94 52 84 7 39 7 90 56 7 19 77 100 76 75 15 60 10 40 88 87 24 77 70 62 79 71 12 99 74 11 92 64 69 55 34 35 5 29 22 12 33 82 26 54 0 40 7 40 72
92 85 29 15 46 79 76 45 13 28 66 64 49 11 78 92 12 100 91 11 9 9 19 34 38 100 55 16 92 51 62 67 90 88 36 6 9 69 85 70 42 53 65 89 80 40 0 53 54 97
12 11 30 71 24 92 77 97 63 94 5 34 20 72 17 31 83 82 29 37 83 24 61 61 53 75 86 80 55 70 17 57 66 48 23 19 54 100 22 11 64 60 45 75 32 7 53 0 91
32
46 90 46 49 29 87 73 97 51 73 26 83 50 94 94 7 40 27 63 5 97 40 27 80 48 35 58 96 16 57 38 80 83 17 49 49 62 91 77 69 15 75 94 10 8 40 54 91 0 25
71 45 91 34 71 71 64 76 91 60 49 81 6 7 9 94 10 18 98 53 64 11 94 43 86 63 6 56 48 92 48 21 99 51 48 73 44 93 11 95 69 90 6 3 95 72 97 32 25 0

Matriz 4

0 23 43 34 28 86 48 62 35 34 57 71 21 40 45 40 67 36 5 21 94 30 51 83 23 87 36 98 46 66 39 99 39 5 64 4 67 95 21 21 96 62 90 98 38 43 6 44 16 33
23 0 59 96 42 80 52 91 45 63 40 15 39 68 7 69 58 10 24 45 24 90 18 25 36 81 85 22 84 23 21 67 27 25 22 46 15 56 95 71 44 16 31 6 89 81 19 69 68 84
43 59 0 68 58 76 15 9 52 23 90 43 15 100 12 42 73 66 46 61 51 78 50 54 34 100 30 80 87 73 40 57 12 34 94 54 47 7 22 16 22 13 16 56 2 35 57 4 10 98
34 96 68 0 55 44 82 91 30 87 15 16 48 97 84 37 78 79 36 6 16 64 73 97 81 46 48 31 56 23 86 99 2 3 52 33 82 90 62 58 75 59 30 26 62 45 7 16 12 55
28 42 58 55 0 94 8 9 14 95 66 4 13 53 28 76 71 26 61 30 100 77 60 78 35 38 46 47 84 14 21 72 74 82 20 13 29 71 48 10 84 40 71 12 14 97 79 88 28 92
86 80 76 44 94 0 32 37 70 6 48 26 60 9 2 31 37 95 33 46 46 14 30 36 75 9 96 12 58 45 51 30 88 67 69 70 85 83 94 67 62 67 20 41 31 54 20 27 83 62
48 52 15 82 8 32 0 23 62 64 99 70 32 75 82 89 58 82 40 39 27 34 70 2 73 31 18 19 73 9 45 91 33 8 29 19 15 79 56 41 52 91 75 85 18 79 69 5 28 25
62 91 9 91 9 37 23 0 7 6 69 4 3 9 76 59 80 24 50 58 40 4 62 56 46 67 51 92 64 83 11 94 19 92 29 4 74 59 34 61 62 84 29 54 16 82 17 53 59 27
35 45 52 30 14 70 62 7 0 81 44 9 92 56 27 29 12 54 87 34 17 30 55 76 43 33 4 44 82 10 61 33 22 2 90 53 95 75 47 16 33 67 75 27 72 20 55 62 67 22
34 63 23 87 95 6 64 6 81 0 35 86 8 83 13 56 55 24 82 39 1 15 5 48 9 87 50 93 57 36 58 64 22 62 77 27 52 6 9 100 100 28 88 55 20 35 82 36 58 70
57 40 90 15 66 48 99 69 44 35 0 79 98 11 23 84 22 34 72 79 57 57 97 4 75 63 88 39 57 58 75 61 77 71 58 77 21 9 8 96 44 8 64 90 60 58 32 53 59 27
71 15 43 16 4 26 70 4 9 86 79 0 71 63 23 34 71 75 89 41 76 21 19 99 28 21 76 79 22 73 79 88 70 20 48 12 97 71 72 6 59 56 49 19 78 38 10 43 9 17
21 39 15 48 13 60 32 3 92 8 98 71 0 57 1 61 61 19 32 14 76 2 61 5 74 90 99 82 95 28 28 59 71 58 4 49 95 75 88 80 56 62 44 33 44 84 97 13 54 7
40 68 100 97 53 9 75 9 56 83 11 63 57 0 67 14 5 5 93 98 56 67 42 61 62 47 70 53 48 78 54 75 94 25 35 70 41 7 85 3 71 16 87 31 14 31 86 12 89 19
45 7 12 84 28 2 82 76 27 13 23 23 1 67 0 46 71 48 59 13 8 96 5 89 33 15 44 78 56 85 9 28 26 90 9 11 63 86 6 99 65 15 62 91 25 3 47 43 8 68
40 69 42 37 76 31 89 59 29 56 84 34 61 14 46 0 54 11 50 35 99 62 64 69 25 70 54 99 76 84 91 68 4 74 39 27 39 61 97 25 56 59 32 6 52 73 98 90 86 43
67 58 73 78 71 37 58 80 12 55 22 71 61 5 71 54 0 48 14 65 54 99 55 10 65 99 83 51 32 21 42 27 55 23 3 44 4 52 4 71 41 55 4 8 11 4 68 64 93 18
36 10 66 79 26 95 82 24 54 24 34 75 19 5 48 11 48 0 93 20 88 85 4 64 80 91 11 53 90 30 21 12 88 88 27 41 18 77 3 27 62 43 17 49 74 47 87 93 9 14
5 24 46 36 61 33 40 50 87 82 72 89 32 93 59 50 14 93 0 3 83 93 73 8 38 84 27 29 44 39 2 26 99 13 12 90 91 91 44 25 25 66 7 19 82 14 2 80 39 32
21 45 61 6 30 46 39 58 34 39 79 41 14 98 13 35 65 20 3 0 32 54 14 52 70 74 70 29 7 89 60 12 6 67 6 13 16 38 85 97 13 85 1 55 59 81 96 26 5 67
94 24 51 16 100 46 27 40 17 1 57 76 76 56 8 99 54 88 83 32 0 61 53 71 99 12 30 94 16 42 12 66 30 69 63 71 57 6 87 11 96 24 66 30 41 43 79 2 83 33
30 90 78 64 77 14 34 4 30 15 57 21 2 67 96 62 99 85 93 54 61 0 87 53 59 3 20 17 42 40 13 62 17 33 9 85 22 13 76 72 15 69 10 77 2 38 99 86 10 43
51 18 50 73 60 30 70 62 55 5 97 19 61 42 5 64 55 4 73 14 53 87 0 26 15 38 57 40 36 83 49 54 78 89 14 10 85 35 21 26 23 81 72 37 69 30 22 21 38 80
83 25 54 97 78 36 2 56 76 48 4 99 5 61 89 69 10 64 8 52 71 53 26 0 9 80 1 62 13 51 5 76 88 79 47 77 37 32 52 71 6 73 7 80 84 27 34 49 22 41
23 36 34 81 35 75 73 46 43 9 75 28 74 62 33 25 65 80 38 70 99 59 15 9 0 56 100 55 2 57 24 28 18 73 11 29 11 49 45 34 87 36 6 41 70 40 91 18 89 29
87 81 100 46 38 9 31 67 33 87 63 21 90 47 15 70 99 91 84 74 12 3 38 80 56 0 33 52 79 5 13 5 84 71 30 52 92 49 19 44 85 59 56 54 42 89 73 72 85 35
36 85 30 48 46 96 18 51 4 50 88 76 99 70 44 54 83 11 27 70 30 20 57 1 100 33 0 38 98 77 90 47 78 79 98 36 49 79 73 25 96 74 10 12 78 82 10 78 58 3
98 22 80 31 47 12 19 92 44 93 39 79 82 53 78 99 51 53 29 29 94 17 40 62 55 52 38 0 14 45 75 34 35 40 56 28 25 55 20 8 95 78 94 64 74 23 98 91 72
48
46 84 87 56 84 58 73 64 82 57 57 22 95 48 56 76 32 90 44 7 16 42 36 13 2 79 98 14 0 92 23 2 61 75 45 47 20 13 5 57 79 20 53 11 20 50 2 83 16 42
66 23 73 23 14 45 9 83 10 36 58 73 28 78 85 84 21 30 39 89 42 40 83 51 57 5 77 45 92 0 80 9 93 33 56 48 25 15 65 48 95 78 68 93 35 27 53 46 45 18
39 21 40 86 21 51 45 11 61 58 75 79 28 54 9 91 42 21 2 60 12 13 49 5 24 13 90 75 23 80 0 52 12 38 95 65 13 26 94 83 61 50 58 11 26 14 61 4 62 14
99 67 57 99 72 30 91 94 33 64 61 88 59 75 28 68 27 12 26 12 66 62 54 76 28 5 47 34 2 9 52 0 49 71 13 6 31 77 86 6 25 30 26 72 74 36 56 44 5 99
39 27 12 2 74 88 33 19 22 22 77 70 71 94 26 4 55 88 99 6 30 17 78 88 18 84 78 35 61 93 12 49 0 13 30 7 97 50 18 89 87 77 5 11 62 69 20 34 13 64
5 25 34 3 82 67 8 92 2 62 71 20 58 25 90 74 23 88 13 67 69 33 89 79 73 71 79 40 75 33 38 71 13 0 5 96 70 87 19 5 84 32 66 35 21 56 56 79 17 65
64 22 94 52 20 69 29 29 90 77 58 48 4 35 9 39 3 27 12 6 63 9 14 47 11 30 98 56 45 56 95 13 30 5 0 83 98 27 76 84 42 80 88 17 8 25 86 90 76 59
4 46 54 33 13 70 19 4 53 27 77 12 49 70 11 27 44 41 90 13 71 85 10 77 29 52 36 28 47 48 65 6 7 96 83 0 42 13 5 33 16 5 89 82 29 18 67 78 11 97
87 15 47 82 29 85 15 74 95 52 21 97 95 41 63 39 4 18 91 16 57 22 85 37 11 92 49 25 20 25 13 31 97 70 98 42 0 94 17 65 12 45 26 84 71 3 90 55 23 16
95 56 7 90 71 83 79 59 75 6 9 71 75 7 86 61 52 77 91 38 6 13 35 32 49 49 79 55 13 15 26 77 50 87 27 13 94 0 71 1 20 46 46 52 26 52 46 19 74 94
21 95 22 62 48 94 56 34 47 9 8 72 88 83 6 97 4 3 44 85 87 76 21 52 45 19 73 20 5 65 94 86 18 19 76 5 17 71 0 2 18 29 46 61 71 7 100 52 10 11
21 71 16 58 10 67 41 61 16 100 96 6 80 3 99 25 71 27 25 97 11 72 26 71 34 44 25 8 57 48 83 6 89 5 84 33 65 1 2 0 58 95 45 24 15 19 77 54 55 84
96 44 22 75 84 62 52 62 33 100 44 59 56 71 65 56 41 62 25 13 96 15 23 6 87 85 96 95 79 95 61 25 87 84 42 16 12 20 18 58 0 66 19 23 4 26 50 31 4 45
62 16 13 59 40 67 91 84 67 28 8 56 62 16 15 59 55 43 66 85 24 69 81 73 36 59 74 78 20 78 50 30 77 32 80 5 45 46 29 95 66 0 67 44 21 50 44 50 75 81
90 31 16 30 71 20 75 29 75 88 64 49 44 87 62 32 4 17 7 1 66 10 72 7 6 56 10 94 53 68 58 26 5 66 88 89 26 46 46 45 19 67 0 35 32 82 12 16 15 48
98 6 56 26 12 41 85 54 27 55 90 19 33 31 91 6 8 49 19 55 30 77 37 80 41 54 12 64 11 93 11 72 11 35 17 82 84 52 61 24 23 44 35 0 56 42 99 89 15 41
38 89 2 62 14 31 18 16 72 20 60 78 44 14 25 52 11 74 82 59 41 2 69 84 70 42 78 74 20 35 26 74 62 21 8 29 71 26 71 15 4 21 32 56 0 30 79 46 63 28
43 81 35 45 97 54 79 82 20 35 58 38 84 31 3 73 4 47 14 81 43 38 30 27 40 89 82 23 50 27 14 36 69 56 25 18 3 52 7 19 26 50 82 42 30 0 59 50 14 22
6 19 57 7 79 20 69 17 55 82 32 10 97 86 47 98 68 87 2 96 79 99 22 34 91 73 10 98 2 53 61 56 20 56 86 67 90 46 100 77 50 44 12 99 79 59 0 58 25 90
44 69 4 16 88 27 5 53 62 36 53 43 13 12 43 90 64 93 80 26 2 86 21 49 18 72 78 91 83 46 4 44 34 79 90 78 55 19 52 54 31 50 16 89 46 50 58 0 48 95
16 68 10 12 28 83 28 59 67 58 59 9 54 89 8 86 93 9 39 5 83 10 38 22 89 85 58 72 16 45 62 5 13 17 76 11 23 74 10 55 4 75 15 15 63 14 25 48 0 88
33 84 98 55 92 62 25 27 22 70 27 17 7 19 68 43 18 14 32 67 33 43 80 41 29 35 3 48 42 18 14 99 64 65 59 97 16 94 11 84 45 81 48 41 28 22 90 95 88 0

Apêndice B

Limites Inferiores e Matrizes de Distância: Instância Real de 24-cidades

Este apêndice contém as informações adicionais referentes à instância real de 24-cidades, intitulada de "Problema das Principais Cidades Brasileiras".

São apresentados, também, limites inferiores obtidos para cada uma delas, conforme descrito nos capítulos 6 e 7, e os menores comprimentos gerados pelo A-Team.

B.2. Limites Inferiores

Esta seção apresenta os melhores limites inferiores obtidos para as três matrizes da instância real de 24-cidades. Os melhores limites inferiores, para todas as matrizes de distância, foram obtidos através do algoritmo de Held-Karp.

A primeira coluna indica qual a matriz de distância abordada (as linhas indicam o tamanho da matriz ou, o número de cidades do problema). A segunda coluna contém o limite inferior para cada matriz e, a terceira, o menor comprimento gerado pelo A-Team.

Tabela B.1: Limites Inferiores para as matrizes relacionadas com o Problema das Principais Cidades Brasileiras (24-cidades).

24-cidades	Melhor Limite Inferior obtido	Menor comprimento gerado pelo A-Team
matriz 1	699	865
matriz 2	495	815
matriz 3	3420	

B.1. Matrizes de Distância

As matrizes para a instância real de 24-cidades também são completas e simétricas.

São apresentadas as três matrizes de distância para a instância real de 24-cidades. A primeira matriz é relativa ao custo da viagem (custo do caminho entre duas cidades). A segunda matriz é relativa à qualidade do vôo. A terceira matriz contém as distâncias (em Km) entre cada duas cidades.

Tabela B.2: Matriz de Distância 1: Tarifas (em Reais R$) para o Problema das Principais Cidades Brasileiros.

	AJU	BEL	BHZ	BSB	CGR	CGB	CWB	FOR	FLN	GYN	IGU	JPA	MAO	MCP	MCZ	NAT	POA	PVH	REC	RIO	SAO	SCZ	SSA	VIX
AJU	0	384	295	282	387	441	386	244	409	305	437	156	494	504	89	186	450	500	136	308	348	329	108	254
BEL	384	0	386	320	442	538	475	259	485	367	518	331	284	120	362	318	520	377	341	426	426	159	408	430
BHZ	295	386	0	176	323	298	214	396	255	196	283	399	423	422	320	366	287	463	341	128	161	353	236	133
BSB	282	320	176	0	225	308	272	336	299	88	319	342	382	360	307	369	329	369	332	225	218	324	249	232
CGB	387	442	323	225	0	169	345	437	360	200	398	438	351	800	408	454	394	260	426	343	305	410	361	365
CGR	441	538	298	308	169	0	270	494	306	276	331	487	416	706	460	505	334	328	477	277	225	510	417	330
CW	386	475	214	272	345	270	0	460	105	263	168	437	498	504	407	447	178	458	425	194	121	455	352	258
FOR	244	259	396	336	437	494	460	0	486	361	505	175	407	503	214	150	508	486	184	416	426	184	285	380
FLN	409	485	255	299	360	305	105	486	0	281	209	457	523	523	429	473	151	477	446	232	173	472	383	290
GYN	305	367	196	88	200	276	263	361	281	0	304	368	382	397	329	381	328	389	357	251	212	346	274	252
IGU	437	518	283	319	398	331	167	505	209	304	0	484	541	548	456	495	264	504	473	272	213	512	406	317
JPA	156	331	399	342	438	487	457	175	457	368	483	0	464	570	114	73	486	557	65	365	401	270	202	317
MA	494	284	423	382	351	416	498	407	523	382	541	464	0	319	489	454	544	203	472	468	468	543	466	484
MCP	504	120	422	360	800	706	504	503	523	397	548	570	319	0	399	351	557	408	372	454	454	212	443	458
MCZ	89	362	320	307	408	460	407	214	429	329	456	114	489	399	0	149	465	518	89	333	371	305	151	281
NAT	186	318	366	369	454	505	447	150	473	381	495	73	454	351	149	0	495	522	102	377	419	256	223	338
POA	450	520	287	329	394	334	178	508	151	328	264	486	544	557	465	495	0	506	475	266	218	513	408	388
PVH	500	377	463	369	260	328	458	486	477	389	504	557	203	408	518	522	506	0	539	468	424	422	475	503
REC	136	341	341	332	426	477	425	184	446	357	473	65	472	372	89	102	475	539	0	352	389	282	183	304
RIO	308	426	128	225	343	277	194	416	232	251	272	365	468	454	333	377	266	468	352	0	133	413	269	141
SAO	348	426	161	218	305	225	121	426	173	212	213	401	468	454	371	419	218	424	389	133	0	426	320	206
SCZ	329	159	353	324	410	510	455	184	472	346	512	270	543	212	305	256	513	422	282	413	426	0	356	427
SSA	108	408	236	249	361	417	352	285	383	274	406	202	466	443	151	223	408	475	183	269	320	356	0	220
VIX	254	430	133	232	365	330	258	380	290	252	317	317	494	458	281	338	388	503	304	141	206	427	219	0

Tabela B.3: Matriz de Distância 2: Qualidade de Vôos (níveis de 10 a 100) para o Problema das Principais Cidades Brasileiros.

	AJU	BEL	BHZ	BSB	CGB	CGR	CWB	FOR	FLN	GYN	IGU	JPA	MA	MCP	MCZ	NAT	POA	PVH	REC	RIO	SAO	SLZ	SSA	VIX
AJU	0	50	50	35	50	50	50	50	50	50	50	50	30	50	50	50	45	50	50	50	35	50	50	50
BEL	50	0	45	15	45	45	45	45	45	45	45	50	30	50	45	45	45	45	45	15	15	45	45	50
BHZ	50	45	0	50	50	50	50	50	50	50	50	45	45	50	45	45	30	50	50	45	45	45	50	45
BSB	35	15	50	0	45	45	30	15	30	50	45	50	15	45	35	50	50	45	35	15	45	30	45	45
CGB	50	45	50	45	0	50	50	50	50	50	50	50	45	45	50	50	45	50	50	15	50	50	50	50
CGR	50	45	50	45	50	0	50	50	50	50	50	50	45	45	50	50	45	50	50	15	50	50	50	50
CWB	50	45	50	30	50	50	0	50	50	45	50	50	30	50	45	45	45	50	50	10	50	50	50	45
FOR	50	45	50	15	50	50	50	0	45	45	45	45	45	45	45	45	45	45	45	30	30	50	45	45
FLN	50	45	50	30	50	50	50	45	0	45	45	50	30	45	45	45	45	50	50	10	45	50	50	45
GYN	50	45	50	50	50	50	45	45	45	0	50	50	30	50	45	45	45	50	45	30	45	50	50	50
IGU	50	45	50	45	50	50	50	45	45	50	0	45	45	45	45	45	45	45	45	30	45	45	45	60
JPA	50	50	45	50	50	50	50	45	50	50	45	0	45	45	45	45	45	45	45	45	45	45	45	50
MA	30	30	45	15	45	45	30	45	30	30	45	45	0	45	30	30	15	50	45	10	15	45	45	30
MCP	50	50	50	45	45	45	50	45	45	50	45	45	45	0	50	45	45	50	45	15	45	45	45	50
MCZ	50	45	45	35	50	50	45	45	45	45	45	45	30	50	0	50	45	50	45	45	35	50	50	50
NAT	50	45	45	45	50	50	45	45	45	45	45	45	30	45	50	0	45	45	50	35	45	50	50	50
POA	45	45	30	30	45	45	45	45	45	45	45	45	15	45	45	45	0	45	35	10	45	15	45	45
PVH	50	45	50	45	50	50	50	45	50	50	45	45	50	50	50	45	45	0	50	10	45	50	50	50
REC	50	45	50	35	50	50	50	45	50	45	45	45	45	45	45	50	35	50	0	15	15	50	45	45
RIO	50	15	45	15	15	15	10	30	10	30	30	45	10	15	45	35	10	10	15	0	10	15	45	50
SAO	35	15	45	45	50	50	50	30	45	45	45	45	15	45	35	45	45	45	15	10	0	10	30	50
SLZ	50	45	45	30	50	50	50	50	50	50	45	45	45	45	50	50	15	50	50	15	10	0	45	50
SSA	50	45	50	45	50	50	50	45	50	50	45	45	45	45	50	50	45	50	45	45	30	45	0	50
VIX	50	50	45	45	50	50	45	45	45	50	60	50	30	50	50	50	45	50	45	50	50	50	50	0

Tabela B.4: Matriz de Distância 3:Duração de Vôos (em min) para o Problema das Principais Cidades Brasileiros.

	AJU	BEL	BHZ	BSB	CGB	CGR	CWB	FOR	FLN	GYN	IGU	JPA	MAO	MCP	MCZ	NAT	POA	PVH	REC	RIO	SAO	SLZ	SSA	VIX
AJU	0	144	82	89	148	173	137	68	149	100	170	34	230	166	14	45	172	281	26	98	114	111	17	73
BEL	144	0	148	107	166	204	179	76	196	118	216	112	86	22	130	105	215	137	118	164	165	33	144	174
BHZ	82	148	0	40	93	91	56	132	67	51	88	115	169	169	140	167	90	169	108	24	33	142	65	26
BSB	89	107	40	0	58	97	72	113	89	11	108	118	129	129	100	127	108	133	110	62	56	102	72	66
CGB	148	166	93	58	0	39	111	171	122	69	145	176	126	188	158	185	128	75	169	107	88	160	131	125
CGR	173	204	91	97	39	0	81	210	184	106	114	215	164	226	197	213	116	114	199	83	58	199	156	109
CWB	137	179	56	72	111	81	0	188	16	76	15	171	200	208	152	177	36	186	169	45	23	174	120	73
FOR	68	76	132	113	171	210	188	0	196	124	220	36	162	98	68	29	220	213	42	145	165	43	68	124
FLN	149	196	67	89	122	184	16	196	0	87	53	182	212	218	168	188	24	197	175	50	34	191	132	76
GYN	100	118	51	11	69	106	76	124	87	0	108	129	140	140	110	138	110	144	121	72	53	113	88	77
IGU	170	216	88	108	143	114	15	220	53	108	0	205	238	238	184	209	114	218	196	79	55	210	163	105
JPA	34	112	115	118	176	215	171	36	182	129	205	0	198	134	20	9	206	251	7	132	148	79	51	107
MA	230	86	169	129	126	164	200	162	212	140	238	198	0	108	216	191	235	51	204	189	177	119	212	195
MCP	166	22	169	129	188	226	208	98	218	140	238	134	108	0	152	127	243	159	140	185	185	65	166	195
MCZ	14	230	140	100	158	197	152	68	168	110	184	20	216	152	0	29	186	233	12	112	129	97	31	112
NAT	45	105	167	127	185	213	177	29	188	138	209	9	191	127	29	0	213	242	17	138	154	72	57	113
POA	172	215	90	108	128	116	36	220	24	110	114	206	235	243	186	213	0	220	198	75	57	210	155	107
PVH	281	137	169	133	75	114	186	215	197	144	218	251	51	159	233	242	220	0	244	187	163	170	206	200
REC	26	118	108	110	169	199	169	42	175	121	196	7	204	140	12	17	198	244	0	124	141	85	45	99
RIO	98	164	24	62	107	83	45	145	50	72	79	132	189	185	112	138	75	187	124	0	24	150	81	28
SAO	114	165	33	56	88	58	23	165	34	53	55	148	177	185	129	154	57	163	141	24	0	175	97	50
SLZ	111	33	142	102	160	199	174	43	191	113	210	79	119	55	97	72	210	170	85	150	175	0	111	167
SSA	17	144	65	72	131	156	120	66	132	82	155	51	212	166	31	57	155	206	43	81	97	111	0	56
VIX	73	174	26	66	125	109	73	124	76	77	105	107	195	195	112	113	107	200	99	28	50	167	56	0

Referências Bibliográficas

[ABNT89] ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS - ABNT. Normalização da Documentação no Brasil; (PNB 66) - Referências Bibliográficas, Rio de Janeiro: IBBD, 1989.

[AHU83] AHO, A. V., HOPCROFT, John E., ULLMAN, Jeffrey. Data Structures and Algorithms. MA: Addison-Wesley, 1983.

[AHU74] _____. *The Design and Analysis of Computer Algorithms*. MA: Addison-Wesley, 1974.

[Aks90] AKSOY, Y. Interactive multiple objective decision making: A bibliography (1975-1988). *Management Research News*, v. 13, 1990. p. 1-8.

[AL85] AARTS, E. H. L., LAARHOVEN, P. J. M. van. Statistical Cooling: A General Approach to Combinatorial Optimization Problems. *Philips Journal of Research*, 1985.

[AK89] AARTS, E. H. L., KORST, J. *Simulated Annealing and Boltzmann Machines*. Chichester: John Wiley & Sons, 1989.

[AM91] ARMAND, P., MALIVERT, C. Determination of the Efficient Set in Multiobjective Linear Programming. *Journal of Optimization Theory and Applications*, [ ]: Plenum, v. 10, n. 3, Sept. 1991. p. 467-489.

[ATG92] AMARAL, J. N., TUMER, K., GHOSH, J. Applying genetic algorithms to the state assignment problem: A case study. *Proceedings of the SPIE - the International Society for Optical Engineering*. Orlando, Apr. 1992.

[Aur86] AURÉLIO Buarque de Holanda. *Novo Dicionário da Língua Portuguesa*. Rio de Janeiro: Nova Fronteira, 1986.

[Aus90] AUSTIN, S. An Introduction to Genetic Algorithms. *AI Expert*, Mar. 1990. p. 49-53.

[AzM91] AZEVEDO, J. A., MARTINS, E. Q. V. An Algorithm for the Multiobjective Shortest Path on Acyclic Networks. *Investigação Operacional*, v. 11, n. 1, 1991. p. 52-69.

[Bal95] BALAS, E., CERIA, S., CORNUÉJOLS, G., NATRAJ, N. Gomory cuts revisited. *Operations Research Letters*, 1995.

[Bal65] BALAS, E., An additive algorithm for solving linear programs with zero-one variables. *Operations Research*, n. 13, 1965. p. 517-546.

[Bar94] BARNHART, C. JOHSON, E., NEMHAUSER, G., SAVELSBERGH, M., VANCE, P. Branch-and-Price: Column generation for solving huge integer programs. In: *Mathematical Programming*; State of the Art, 1994. J. Birge, K. Murty (eds.), University of Michigan, 1994.

[Bar92] BARRY, R. G. *Atmosphere, weather and climate*. London: Routledge, 1992.

[BC81] BALAS, E., CHRISTOFIDES, N. A restricted Lagrangean approach to the traveling salesman problem. *Mathematical Programming*, n. 21, 1981. p. 19-36.

[BCC95] BALAS, E., CERIA, S., CORNUÉJOLS, G. Mixed 0-1 Programming by lift-and-project method. *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithm*, 1993. p. 232-242.

[BCC93] BALAS, E., CERIA, S., CORNUÉJOLS, G. A lift-and-project cutting plane algorithm for mixed 0-1 programs. *Mathematical Programming*, v. 58, 1993. p. 295-324.

[BD62] BELLMAN, R., DREYFUS, S. *Applied Dynamic Programming*, Princeton, NJ: Princeton University Press, 1962.

[Ben86] BENSON, Harold P. An algorithm for optimizing over the weakly-efficient set. *European Journal of Operational Research*, North-Holland: Elsevier Science, v. 25, n. 2, May 1986. p. 192-199.

[Ben92] BENTLEY, Jon-Louis. Fast Algorithms for Geometric Traveling Salesman Problems. *ORSA Journal on Computing*. n. 4, 1992. p. 387-441.

[Ben90] _____. Experiments on Geometric Traveling Salesman Heuristics. *Computing Science Technical Report*. New Jersey: AT&T Bell Laboratories, n. 151, Aug. 1990. 92 p.

[Bit79] BITRAN, G. Theory and algorithms for linear multiple objective programs with zero-one variables. *Mathematical Programming*, v. 17, 1979. p. 362-390.

[Bit77] _____. Linear multiple objective programs with zero-one variables. *Mathematical Programming*, v. 13, 1977. p. 121-139.

[BJS90] BAZARAA, Mokhtar S., JARVIS, John J. SHERALI, Hanif D. *Linear Programming and Network Flows*. 2. ed. [USA]: John Wiley & Sons, 1990. 684 p.

[BKP82] BURKARD, R.E., KRARUP, J., PRUZAN, P.M. Efficiency and Optimality in Minisum, Minimax 0-1 Programming Problems. *Journal of the Operational Research Society*, v. 33, n. 2, 1982. p. 137-151.

[BL80] BITRAN, G.R., LAWRENCE, K.D. Locating Service Offices; A Multicriteria Approach. *Omega*, v. 8, n. 2, 1980. p. 201-206.

[Bla74] BLAU, P. M. *On the Nature of Organizations*. New York: John Wiley, 1974.

[BM81] BALL, M., MAGAZINE, M. The Design and Analysis of Heuristics, 11(2), 1981. p. 215-219.

[BM76] BONDY, J. A., MURTY, U. S. R. *Graph Theory with Applications*. New York: American Elsevier Publishing, 1976.

[BMT71] BENAYOUN, R., MONTGOLFIER, J., TERGNY, J. Linear Programming With Multiple Objective Functions: STEP Method (STEM). *Mathematical Programming*, v. 1, 1971. p. 366-375.

[BMV84] BRANS, J.-P., MARESCHAL, B., VINCKE, P. PROMETHEE: A new family of outranking methods in multicriteria analysis. *Operational Research '84*, IFORS, Amsterdan: Elsevier, 1984. p. 477-490.

[BP82] BARTHOLDI, J. J., PLATZMAN, L. K. An O(n log n) Planar Traveling Salesman Heuristic Based on Spacefilling Curves. *Operations Research Letters*, n. 4, 1982. p. 121-125.

[BR84] BURKARD, R. E., RENDL, F. A termodynamically motivated simulation procedure for combinatorial optimization problems. *European Journal of Operational Research*. 17(2) New York: American Elsevier Publishing, 1984. 169-174.

[BR82] BITRAN, G.R., RIVERA, J.M. A Combined Approach to Solve Binary Multicriteria Problems. *Naval Research Logistics Quarterly*, v. 29, n. 2, 1982. p. 181-201.

[BT90] BERTSEKAS, D. P., TSITSIKLIS, J. N. *A survey of some aspects of parallel and distributed iterative algorithms*. Internal Report CICS-P-189. Cambridge, MA: MIT, 1990.

[BT96] BAERENTZEN, L., TALUKDAR, S. N. Improving Cooperation Among Autonomous Agents in Asynchronous Teams. *Technical Report EDRC*. Pittsburgh, PA: Carnegie Mellon University, July 1996.

[BT89b] _____. *Convergence rate and termination of asynchronous algorithms*. Internal Report CICS-P-147. Cambridge, MA: MIT, 1989.

[BT85] BALAS, E, TOTH, P. Branch and bound methods. In: The Traveling Salesman Problem. E. L. Lawler, J.K. Lenstra, A. H. G. Rinnooy Kan, D. B. Shimoys (ed.). Chichester: John Wiley & Sons, 1985.

[Bur81] BURKARD, R.E. et al. Relationship Between Optimality end Efficiency in Multicriteria 0-1 Programming Problems. *Computers and Operations Research*, v. 8, n. 4, 1981. p. 241-248.

[Cal91] CALLOWAY, D. L. Using a Genetic Algorithm to Design Binary Phase-Only Filters for Pattern Recognition. *Proceedings of the Fourth International Conference on Genetic Algorithms*. Morgan Kaufmann. Los Altos, 1991.

[Cam95] CAMPONOGARA, Eduardo. *Times Assíncronos para um Problema de Derivados de Petróleo*. Campinas: Instituto de Computação, Universidade Estadual de Campinas, 1995. (Dissertação, Mestrado em Ciência da Computação).

[CaS95c] CAMPONOGARA, E., SOUZA, P. S. de. A Extensão de um Modelo de Escalonamento Aplicada a um Problema de Suprimento de Demandas. *Anais do XXVII Simpósio Brasileiro de Pesquisa Operacional*, Vitória, ES, nov. 1995.

[CaS95b] _____. Modelo Matemático de um Problema de Suprimento de Demandas. *Anais do II Simpósio Brasileiro de Automação Inteligente*. CEFET-PR, set. 1995.

[CaS95a] _____. A Solução de um Problema de Transporte Através de sua Decomposição. *Anais do XXII Seminário Integrado de Software e Hardware*, v. II, Canela, RS, jul. 1995. p. 1396-1400.

[Cav95] CAVALCANTE, Victor F. *Times Assíncronos na Solução do Job Shop Scheduling Problem;* Heurísticas de Construção. Campinas: Instituto de Computação, Universidade Estadual de Campinas, 1995. (Dissertação, Mestrado em Ciência da Computação).

[CC61] CHARNES, A., COOPER, W. W. *Management Models and Industrial Applications of Linear Programming*, New York: Wiley, 1961.

[CDM91] COLORNI, A., DORIGO, M., MANIEZZO, V. Distributed Optimization by Ant Colonies. *Proceedings of the First European Conference on Artificial Life*. Paris, Dec. 1991.

[CEG88] COLLINS, N. E., EGLESE, R. W., GOLDEN, B. L. Simulated Annealing: An Annotated Bibliography. *Amer. J. Math. and Mgmt. Science*, n. 8, 1988. p. 205-307.

[Ceg94] CEGALLA, Domingos Paschoal. *Novíssima Gramática da Língua Portuguesa*. São Paulo: Editora Nacional, 37 ed., 1994.

[Cer96] CERIA, Sebastián. Algoritmos para programación entera. *3ra Escuela latinoamericana de verano de investigación operativa*. Bariloche, Argentina: ALIO, jan. 1996. 72 p.

[Cer95] CERIA, S. et al. *Cutting Planes for Integer Programs with General Integer Variables*. Technical Report, Dec. 1995. 18p.

[Cer85] CERNY, V. Thermodinamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm. *Journal of Opt. Theory Applied*, n. 45, 1985.

[CFN85] CORNUÉJOLS, G., FONLUPT, J., NADDEF, D. The traveling salesman problem on a graph and some related polyhedra. *Mathematical Programming*, n. 33, 1985. p. 1-27.

[CH83] CHANKONG, Vira, HAIMES, Yacov Y. *Multiobjective Decision Making*: Theory and Methodology. North Holland Series in System Science and Engineering, 8, North Holand: Elsevier Science, 1983. 406 p.

[Che92] CHEN, C. L. *Bayesian Nets and A-Teams for Power System Fault Diagnosis*. Pittsburgh: Electrical and Computer Engineering Department, Carnegie Mellon University, 1992. (Thesis, Ph. D. in Electrical and Computer Engineering).

[Che72] CHERNOFF, H. The use of faces to represent points in k-dimensional space graphically. *Journal of the American Statistical Association*, v. 68, 1972. p. 361-367.

[Chr76] CHRISTOFIDES, N. Worst-Case Analysis of a New Heuristic for the Traveling Salesman Problem. Report 388, Graduate School of Industrial Administration. Pittsburgh, PA: Carnegie Mellon University, 1976.

[Chr70] _____. The shortest hamiltonial chain of a graph. *SIAM J. Appl. Math.*, v. 19, 1970. p. 689-696.

[Chv73] CHVÁTAL, Václav. Edmonds Polytopes and Weakly Hamiltonian Graphs. *Mathematical Programming*, Amsterdan: North-Holland Publishing Company, v. 5, n. 1, Aug. 1973. p. 29-40.

[Chv83] CHVÁTAL, Vasek. *Linear programming*. A Series of Books in the Mathematical Sciences, New York: W. H. Freeman, 1983. 478 p.

[CJP83] CROWDER, H., JOHNSON, E., PADBERG, M. Solving large-scale zero-one linear programming problems. *Operations Research*, n. 31, 1983. p. 803-834.

[CK91] CORNER, J. L., KIRKWOOD, C. W. Decision analysis applications in the Operations Research literature, 1970-1989. *Operations Research*, v. 39, 1991. p. 206-219.

[CLE86] CHALMET, L. G., LEMONIDIS, L., ELZINGA, D. J. An algorithm for the bi-criterion integer programming problem. *European Journal of Operational Research*, North-Holland: Elsevier Science, v. 25, n. 2, May 1986. p. 292-300.

[Cli95] CLÍMACO, J. A Programação Matemática com Objectivos Múltiplos no Apoio à Decisão - Uma Perspectiva e Principais Vias de Desenvolvimento. In: XXVII Simpósio Brasileiro de Pesquisa Operacional, nov. 1995.

[CM80] CLÍMACO, J., MARTINS, E. Q. V. On the Determination of the Nondominated paths in a Multiobjective Network Problem. In: *Symposium über Operations Research*, Köln, Aug. 1980.

[CNS95] CERIA, S., NOBILI, P., SASSANO, A. *A Lagrangian-based Heuristic for Large-scale Set Covering Problems*. Technical Report, Apr. 1995. 15 p.

[Coo71] COOK, S. A. On the complexity of theorem-proving procedures. *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, 1971. p. 151-158.

[Cor91] CORMEN, T. *Introduction to Algorithms*. New York: McGraw Hill, 1991. 1027 p.

[Cos88] COSTA, Carlos Bana e. Introdução Geral às Abordagens Multicritério de Apoio à Tomada de Decisão. *Investigação Operacional*, Lisboa: CESUR, v. 8, n. 1, jun. 1988. p. 117-139.

[CP80] CROWDER, Harlan, PADBERG, Manfred W. Solving large-scale symmetric traveling salesman problems to optimality. *Management Science*, USA: [ ], v. 26, n. 5, May 1980. p. 495-509.

[CPLE94] CPLEX. Using the CPLEX Callable Library - Using the CPLEX Linear Optimizer with CPLEX Barrier and Mixed Integer Solvers, CPLEX Optimization, Inc. Version 3.0, 1994.

[CS95] CAVALCANTE, Victor Fernandes, SOUZA, Pedro Sérgio de. Solving the Job Shop Scheduling Problem by Asynchronous Teams. *Proceedings of International Symposium of Operational Research - ISORA*. Beijing, China, 1995.

[CS94] _____. Times Assíncronos Aplicado ao Job Shop Problem. *Anais do XXVI Simpósio Brasileiro de Pesquisa Operacional*, Florianópolis, SC, nov. 1994. p. 791-796.

[CS89] CLEVELAND, G. A., SMITH, S. F. Using Genetic Algorithms to Schedule Flow Shop Releases. *Proceedings of the Third International Conference on Genetic Algorithms*. Los Altos, CA: Morgan Kauffmann, 1989.

[Dav91] DAVID, Lawrence. *Handbook of Genetic ALgorithms*. New York: Van Nostrand Reinhold, 1991.

[Dav85] _____. Job Shop Scheduling with Genetic Algorithms. *Proceedings of the First International Conference on Genetic Algorithms*, Pittsburgh, PA, 1985.

[DFJ59] DANTZIG, G. B., FULKERSON, D. R., JOHNSON, S. M. On a linear-programming, combinatorial approach to the traveling salesman problem. *Operations Research*, n. 7, 1959. p. 58-66.

[DFJ54] _____. Solution of a large-scale traveling salesman problem. *Operations Research*, n. 2, 1954. p. 393-410.

[Dun78] DUNCAN, W. J. *Organizational Behavior*. Boston, MA: Houghton Mifflin, 1978.

[DW87] DURBIN, R., WILLSHAW, G. An analogue approach to the traveling salesman problem using an elastic net method. *Nature*, n. 326, 1987.

[DW83] DECKO, R. F., WINKOFSKY, E. P. Solving zero-one multiple objective programs through implicit enumeration. *European Journal of Operational Research*, v. 12, 1983. p. 362-374.

[Dye92] DYER, J. S. et al. Multiple criteria decision making, multiattribute utility theory: The next ten years. *Management Science*, 1992.

[EG93] ELMER, B., GILLET, B. E. *The Design, Analysis and Implementation of Parallel Simulated Annealing and Parallel Genetic Algorithms for Composite Graph Coloring Problem.* Ph. D. Thesis (Master of Computer Science), Rolla (MO): University of Missouri, 1993. 210 p.

[Fis80] FISHER, Marshall L. Worst-Case Analysis of Heuristic Algorithms. *Management Science*, USA: The Institute of Management Science, v. 26, n. 4, Jan. 1980. p. 1-16.

[FKO85] FELTEN, E., KARLIN, S., OTTO, S. W. The traveling salesman problem on hypercubic, mimd computer. *Proceedings of the 1985 International Conference on Parallel Processing*, St. Charles, PA, 1985.

[Fle87] FLETCHER, R. *Practical Methods of Optimization*, Chirchester, UK: John Wiley & Sons, 1987.

[Fox79] FOX, M. S. Organizational Structuring: Designing Large, Complex Software. Technical Report CMU-CS-79-155, Carnegie Mellon University, 1979.

[FR95b] FEO, T., RESENDE, M. A greedy randomized adaptive search procedure for maximum independent set. *Operations Research*, v. 42, 1994. p. 860-879.

[FR95a] _____. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, v. 6, 1995. p. 109-133.

[Fre95] FREDMAN, M. L. et al. Data Structures for Traveling Salesmen. *Journal of Algorithms*, v. 18, 1995. p. 432-479.

[FM92] FRIEDMAN, Lea, MEHREZ, Abraham. Towards a theory of determining size in multi-objective service systems. *European Journal of Operational Research: Multicriteria Decision Support (EJORDT)*, North-Holland: Elsevier Science, v. 3, n. 63, Dec. 1992. p. 398-408.

[FR95] FEO, Thomas A., RESENDE, Maurício G. C. Greedy Radomized Adaptive Search Procedures. *Journal of Global Optimization*, Boston: Kluwer Academic Publishers, 1995. p. 1-27.

[FS88] FAIGLE, U., SCHRADER. On the convergence of stationary distributions in simulated annealing algorithms. *Inf. Process Letters*, n. 27, 1988. p. 189-194.

[FW91] FRITZKE, B., WILKE, P. *FLEXMAP - A Neural Network for the Traveling Salesman Problem with Linear Time and Space Complexity.* Research Report, Universität Erlangen-Nürnberg, 1991.

[GA94] GLIENKE, M. V., ARAGÃO, M. V. Poggi de. Métodos Multi-Algorítmicos Aplicados a Problemas de Roteamento de Veículos. Departamento de Ciência da Computação. Campinas: Universidade Estadual de Campinas, 1994. (Proposta de Dissertação, Ciência da Computação).

[GA90] GUIMARÃES, E. M., ABREU, N. M. M. Avaliação e Análise de Soluções do Problema do Caixeiro Viajante. *Pesquisa Operacional*, Rio de Janeiro: Sociedade Brasileira de Pesquisa Operacional, v. 10, n. 1, June 1990. p. 19-35.

[Gal92] GALPERIN, E. A. Nonscalarized Multiobjective Global Optimization. *Journal of Optimization Theory and Applications*, [ ]: Plenum, v. 75, n. 1, Oct. 1992. p. 69-85.

[Gas89] GASSER, L. Distributed Artificial Intelligence. *AI Expert*, July 1989.

[GD90] GOMES, L. F. A. Monteiro, DUARTE Jr, Antonio Marcos. Uma Comparação entre Quatro Métodos Multicriteriais de Auxílio à Decisão. Rio de Janeiro: Asociación Latino-Ibero-Americana de Investigación Operativa, v. 2, n. 1, dic. 1990. p. 23-31.

[Gei94b] GEIST, Al et al. PVM: Parallel Virtual Machine - A Users' Guide and Tutorial for Networked Parallel Computing, Cambridge: Massachusetts Institute of Technology, 1994.

[Gei94a] _____. *PVM 3 Users' Guide and Reference Manual*, Tenessee: Oak Ridge National Laboratory, 1994.

[GGRG85] GREFENSTETTE, J., GOPAL, R., ROSMAITA, B., GUCHT, D. V. Genetic Algorithms for the Traveling Salesman Problem. *Proceedings of an International Conference on Genetic Algorithms*. Pittsburgh, PA, July 1985.

[GHD82] GOICOECHEA, Ambrose, HANSEN, Don R., DUCKSTEIN, Lucien. *Multiobjective Decision Analysis with Engineering and Business Aplications*. New York: John Wiley & Sons, 1982. 519 p.

[GHL92] GENDREAU, M., HERTZ, A., LAPORT, G. New insertion and post-optimization procedures for the traveling salesman problems. *Operations Research*, 1992.

[GJ79] GAREY, Michael R., JOHNSON, David. *Computers and Intractabilidade*; A Guide to NP-Completeness. San Francisco, CA: W. H. Freeman, 1979.

[GK92] GRAMA, A., KUMAR, V. Parallel Processing of Discrete Optimization Problems: A survey. Department of Computer Science, University of Minneapolis, MN, 1992.

[GL93] GRÖTSCHEL, M., LOVÁSZ, L. *Combinatorial Optimization*: A Survey. DIMACS Technical Report 93-29, May 1993.

[GL85] GOLDBEG, D., LINGLE JR., R. Alleles, Loci and the Traveling Salesman Problem. *Proceedings of an International Conference on Genetic Algorithms and their Applications*, Alabama: L. Erbaum, 1985. p. 154-159.

[Glo92] GLOVER, Fred. New Ejection Chain and Alternating Path Methods for The Traveling Salesman Problem. *Proceedings of the ORSA Meetings on Computer Science in Operation Research*. Williamsburg, VA, 1992.

[Glo90b] _____. Tabu Search: A tutorial. *Interfaces*, [ ]: Institute of Management Science, v. 75, n. 4, June-Aug. 1990. p. 74-94.

[Glo90a] _____. Tabu Search: Part II. *ORSA Journal of Computing*, v.1, n. 2, 1990. p. 4-32.

[Glo89] _____. Tabu Search: Part I. *ORSA Journal of Computing*, 1989. p. 190-206.

[Glo86] _____. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, v. 13, n. 5, 1986. p. 533-549.

[GM86] GABBANI, D., MAGAZINE, M. J. An interactive heuristic approach for multiobjective integer programming problems. *Journal of the Operational Research Society*, v. 37, 1986. p. 285-291.

[GM74] GILLET, B. E., MILLER, L. R. A heuristic algorithm for the Vehicle-Dispatch Problem. *Operational Research*, v. 22, n. 3, 1974. p. 340-349.

[Gol89] GOLDBERG, David E. Genetic Algorithms in Search, Optimization and Machine Learning. MA: *Addison-Wesley*, 1989.

[Gol85] _____. Alleles, Loci, and the Traveling Salesman Problem. *Proceedings of an International Conference on Genetic Algorithms*. Pittsburgh, PA, July 1985.

[Gom60] GOMORY, R. E. Solving linear programming problems in integers. *Proc. Sympos. Appl. Math.*, v. 10, 1960. p. 211-215.

[GP85] GRÖTSCHEL, Martin, PADBERG, Manfred W. Polyhedral theory. In: *The Traveling Salesman Problem*. E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, D. B. Shmoys (eds.), Chichester: John Wiley & Sons, 1985. p. 251-305.

[GP79b] _____. On the Symmetric Traveling Salesman Problem II: Lifting Theorems and Facets. *Mathematical Programming*, Amsterdan: North-Holland Publishing, v. 16, n. 3, May 1979. p. 281-302.

[GP79a] _____. On the Symmetric Traveling Salesman Problem I: Inequalities. *Mathematical Programming*, Amsterdan: North-Holland Publishing, v. 16, n. 3, May 1979. p. 265-280.

[GR88] GIBBONS, A., RYTTER, W. Efficient Parallel Algorithms. Great Britain, Cambridge: University Press, 1988.

[Grö80] GRÖTSCHEL, Martin. On the Symmetric Traveling Salesman Problem: Solution of a 120-city Problem. *Mathematical Programming Study 12 - Combinatorial Optimization*, Amsterdan: North-Holland Publishing, v. 16, n. 3, May 1979. p. 265-280.

[GS92] GARDINER, L., STEUER, R. E. Unified interactive multiobjective objective programming. *European Journal of Operational Research*, 1992.

[GS85] GOLDEN, B. L., STEWART, W. R. Empirical analysis of heuristics.

[GS82] GRIMMETT, G. R., STIRZAKER, D. R. *Probability and Random Process*. Oxford, UK: Clarendon Press, 1982.

[GW86] GAL, Tomas, WOLF, Karin. Stability in vector maximization - A survey. *European Journal of Operational Research*, North-Holland: Elsevier Science, v. 25, n. 2, May 1986. p. 169-182.

[Haw80] HWANG, C. L., PAIDY, S. R., YOON, K., MASUD, A.S.M. Mathematical Programming with Multiple Objectives: A Tutorial. *Computer & Operacional Research*, Great Britain: Pergamon Press, v. 7, 1980. p. 5-31.

[HC88] HAUSER, J. R., CLAUSING, D. The house of quality. *Harvard Business Review*, v. 66, 988. p. 63-73.

[Hen85] HENIG, Mordechai J. The shortest path problem with two objectives functions. *European Journal of Operational Research*, North-Holland: Elsevier Science, v. 25, n. 2, May 1985. p. 281-291.

[HK71] HELD, Michael, KARP, Richard M. The Traveling-Salesman Problem and Minimum Spanning Trees: Part II. *Mathematical Programming*, [ ]: North-Holland, 1, 1971. p. 6-25.

[HK70] _____. The Traveling-Salesman Problem and Minimum Spanning Trees. *Operations Research*, v. 18, 1970. p. 1138-1162.

[HLL91] HSIA, W. S., LEE, T. Y., LEE, J. Y. Lagrange Multiplier Theorem of Multiobjective Programming Problems with Set Functions. *Journal of Optimization Theory and Applications*, [ ]: Plenum, v. 70, n. 1, July 1991. p. 137-155.

[HM79] HWANG, C. L., MASUD, A. *Multiple Objective Decision Making - Methods and Applications: A State-of-the-Art Survey*, Berlim: Springer Verlag, 1979.

[HN93] HORN, Jeffrey, NAFPLIOTIS, Nicholas. *Multiobjective Optimization Using the Niched Pareto Genetic Algorithm*. Technical Report, Illinois Genetic Algorithms Laboratory, Department of General Engineering, Illinois: University of Illinois at Urbana-Champaign, n. 93005, July 1993. 32 p.

[HP93] HOFFMAN, K. L., PADBERG, M. Solving airline crew scheduling problems by branch-and-cut. *Management Science*, v. 39, 1993. p. 657-682.

[HP91] _____. Techniques for improving the LP-representation of sero-one liner programming problems. *ORSA Journal on Computing*, v. 3, 1991. p. 121-134.

[HS94] HADDAD, Elaine G., SOUZA, Pedro Sérgio de. *Times Assíncronos na solução do Job Shop Scheduling Problem: heurísticas de melhoria*. Campinas: Universidade Estadual de Campinas, 1994. (Proposta de Dissertação, Mestrado em Ciência da Computação).

[Hu70] HU, T. C. *Integer Programming and Network Flows*. 2. ed. Massachusetts: Addison Wesley, 1970. 452 p.

[Hur91] HURKENS, C. A. J. Nasty TSP instances for classical insertion heuristics. Eindhoven, University of Technology, 1991.

[HT85] HOPFIELD, J. J., TANK, D. W. Neural computation of decisions in optimization problems. *Biol. Cybernetics*, n. 52, 1985. p. 141-152.

[HW87] HERTZ, A., WERRA, Lausanne D. de. Using Tabu Search Techniques for Graph Coloring. *Computing*, [ ]: Springer-Verlag, 39, May 1987. p. 345-351.

[Ign76] IGNIZIO, J. P. *Goal Programming and Extensions*, Lexington, MA, Heath, 1976.

[IM92] INOUCHI, H., McLOUGHLIN, N. Parallel techniques for image processing and artificial neural network simulation. London: Springer-Verlag, 1992.

[Ish92] ISHIZUKA, Y. Optimality Conditions for Directionally Differentiable Multi-Objective Programming Problems. *Journal of Optimization Theory and Applications*, [ ]: Plenum, v. 72, n. 1, Jan. 1992. p. 91-111.

[Joh91] JOHNSON, D. S. et al. Optimization by simulated anneling: An experimental evaluation. *Operations Research*, v. 37, 1991. p. 865-892 (Part I), p. 39, 378-406 (Part II).

[Joh90] _____. Local Optimization and the Traveling Salesman Problem. *Proc. 17th. Colloquium on Automata, Languages and Programming*. Springer-Verlag, 1990. p. 446-461.

[JPY88] JOHNSON, D. S., PAPADIMITRIOU, C. H., YANNAKAKIS, M. How easy is local search?. *J. Comp. Syst. Sciences*, v. 37, 1988. p. 79-100.

[JRR94] JÜNGER, M., REINELT, G., RINALDI, G. *The Traveling Salesman Problem*. Preprint 94-12, Heidelberg: IWR, Mar. 1994. 112 p.

[Kar95] KARAIVANOVA, Jasmina et al. A reference direction approach to multiple objective linear programming. *European Journal of Operational Research - The Analysis of Survey Data*, [ ]: Elsevier Science, 81, 1995. p. 176-187.

[Kar91] KARAINOVA, J. Algorithms for solving multicriteria integer programming problems, Sofia, Bulgaria, 1991. (Thesis, Institute of Informatics).

[Kar76] KARP, Richard M. The Probabilistic Analysis of some Combinatorial Search Algorithms. Separata de TRAUB, J. F. *Algorithms and Complexity New Directions and Recent Results*. Nova York: Academic Press, 1976. 19 p.

[Kar72] _____. Reductibility among combinatorial problems. In: *Complexity of Computer Computations*. R. E. Miller, J. W. Thatcher. New York: Plenum Press, 1972. p. 85-103.

[Kat89] KATOH, Naoki. An Efficient Algorithm for Bicriteria Minimum-Cost Circulation Problem. *Journal of the Operations Research Society of Japan, ...*, v. 32, n. 4, Dec. 1989. p. 420-440

[Kel89] KELLER, C. Peter. Algorithms to Solve the Orienteering Problem; A Comparision. *European Journal of Operacional Research*, v. 41, 1989.

[KF71] KROLAK, P., FELTS, W. A Man-Machine Approach Toward Solving the Traveling Salesman Problem. *Communications of the ACM*. v. 14, n. 5, May 1971.

[KG89] KNOX, J., GLOVER, F. Comparative Testing of Traveling Salesman Heuristics Derived from Tabu Search, Genetic Algorithms and Simulated Annealing. *Center for Applied Artificial Intelligence*, University of Colorado, 1989.

[KGV83] KIRKPATRICK, S., GELATT JR., C. D., VECCHI, M. P. Optimization by Simulated Annealing. *Science*, v. 220, n. 4598, May 1983. p. 671-679.

[KH82] KLEIN, D., HANNAN, E.L. An Algorithm for the Multiple Objective Integer Programming Problem, *European Journal of Operational Research*, v. 9, n. 4, 1982. p. 378-385.

[KH67] KARP, R. M., HELD, M. H. Finite state processes and dynamic programming. *SIAM J. Appl. Math.*, v. 15, 1967. p. 693-718.

[KHP95] KAO, J-H., HEMMERLE, J.S., PRINZ, F.B. Asynchronous-Teams Based Collision Avoidance in PAWS. Technical Report EDRC 05-95-95, Engineering Design Research Center, Carnegie Mellon University, 1995.

[Kim91] KIM, K. et al. Fuzzy multicriteria methodologies for quality function deployment. *CMME Working Paper Series N. 91-10-1*, West Lafayette: Purdue University, 1991.

[KK88] KUMAR, V., KANAL, L. The cdp: A unifying formulation for heuristic search, dynamic programming, and branch-and-bound. *Search in Artificial Intelligence*. New York: Springer Verlag, 1988.

[KL86] KORHONEN, P., LAAKSO, J. A visual interactive method for solving the multiple criteria problem. *European Journal of Operational Research*, v. 24, 1986, p. 277-287.

[KMW92] KORHONEN, Pekka, MOSKOWITZ, Herbert, WALLENIUS, Jyrki. Multiple criteria decision support - A review. *European Journal of Operational Research*, North-Holland: Elsevier Science, v. 63, n. 3, Dec. 1992. p. 361-375.

[KNW89] KORHONEN, P., NARULA, S., WALLENIUS, J. An evolutionary approach to decision-making, with an application to media selection. *Mathematical and Computer Modelling*, v. 12, 1989. p. 1239-1244.

[Kor91] KORHONEN, P. Using harmonious houses for visual pairwise comparision of multiple criteria alternatives. *Decision Support Systems*, v. 7, 1991. p. 47-54.

[Kor88] _____. A visual reference direction approach to solving discrete multiple criteria problems. *European Journal of Operational Research*, v. 34, 1988.

[Kor86] _____. A hierarchical interactive method for ranking alternatives with multiple qualitative criteria. *European Journal of Operational Research*, v. 24, 1986.

[KR76] KEENEY, R. L., RAIFFA, H. *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*. New York: Wiley, 1976.

[Kre89] KREGLEWSKI, T. J. et al. IAC-DIDAS-N - A Dynamic Interactive Decision Analysis and Support for multicriteria analysis of nonlinear models. In: *Aspiration Based Decision Support Systems*, Berlim:Springer-Verlag, 1989. p. 378-381.

[Kru84] KRUSE, Robert L. *Data Structures & Program Design*. Englewood Cliffs, New Jersey: Prentice-Hall, 1984.

[KS88] KORHONEN, P., SOISMAA, M. A multiple criteria model for pricing alcoholic beverages. *European Journal of Operations Research*, v. 37, 1988. p. 165-175.

[KSS94] KAUL, R. N., SUNEJA, S. K., SRIVASTAVA, M. K. Optimality Criteria and Duality in Multiple-Objective Optimization Involving Generalized Invexity. *Journal of Optimization Theory and Applications*, [ ]: Plenum, v. 80, n. 3, mar. 1994. p. 465-482.

[KSW90] KORHONEN, P., SILJAMÄKI, A., WALLENIUS, J. A multiple criteria Decision Support System VIG in corporate planning. Numplan, 1990.

[KV86] KIRKWOOD, C., VAN der FELTSZ, L. Personal computer programs for decision and Information Systems. *College of Business*, Arizona State University, 1986.

[KW88] KORHONEN, P., WALLENIUS, J. A multiple objective linear programming decision support system. *Decision Support Systems*, v. 6, 1990. p. 243-251.

[KW88] _____. A Pareto Race. *Naval Research Logistics*, v. 35, 1988. p. 615-623.

[KY83] KIZILTAN, G., YUCAOGLU, E. An algorithm for multiobjective zero-one linear programming. *Management Science*, v. 29, 1983. p. 1444-1453.

[KZV81] KARWAN, M.H., ZIONTS, S., VILLARREAL, B. An Improved Multicriteria Integer Programming Algorithm. *Working Paper n. 530*, School of Management, State University of New York at Buffalo, Buffalo, 1981.

[LA87] LAARHOVEN, P. J. M. van, AARTS, E. H. L. Simulated Annealing: Theory and Applications. *Reidel*, 1987.

[Lag95] LAGUNA, M. Tabu Search Tutorial. In: II Escuela de Verano Latino-Americana de Investigación Operativa, Rio de Janeiro, jan. 1995.

[Lag94] _____. A Guide to Implementing Tabu Search. *Investigación Operativa*, Rio de Janeiro: Asociación Latino-Ibero-Americana de Investigación Operativa, v. 4, n. 1, avr. 1994. p. 5-25.

[Lap92] LAPORTE, Gilbert. The Traveling Salesman Problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, North-Holland: Elsevier Science, v. 3, n. 59, 1992. p. 345-358.

[Lau92] LAU, C. (ed.). *Neural Networks*. New York: IEEE Press, 1992.

[LD60] LAND, A. H., DOIG, A. G. An automatic method for solving discrete programming problems. *Econometrica*, v. 28, 1960. p. 497-520.

[Lew89] LEWANDOWSKI, A., et al. Decision Support Systems of DIDAS family. In: *Aspiration Based Decision Support Systems*. A. Lewandowski, A. Wierzbicki (eds.), Berlin: Springer-Verlag, 1989. p. 21-47.

[Lip87] LIPPMAN, R. P. Introduction to compution with neural nets. *IEEE Acoustics, Speech, and Signal Processing Magazine*, Apr. 1987.

[Liu68] LIU, C. L. *Introduction to Combinatorial Mathematics*. New York: MacGraw Hill, 1968.

[LK75] LENSTRA, J. K., KAN, A. H. G. Rinnoy. Some simple applications of the traveling salesman problem. *Oper. Res. Quart.*, n. 26, 1975. p. 717-733.

[LK73] LIN, S., KERNIGHAN, B. W. An Effective Heuristic Algorithm for the Traveling Salesman Problem. *Operations Research*, v. 21, n. 2, Mar.-Apr. 1973. p. 498-516.

[LLKS85] LAWLER, E. L., LENSTRA, J. K., KAN, A. H. G. Rinnooy, SHMOYS, D. B. (ed.) *The Traveling Salesman Problem*. Chichester: John Wiley & Sons, 1985. 465 p.

[LMM90] LARICHEV, O., MECHITOV, A., MOSHKOVITCH, H. Sequential choice procedure. *Software System Developed at VNIISI*, Moscou, 1990.

[Lon95] LONGO, Humberto José. *Aplicação de A-Teams ao Problema de Recobrimento*. Instituto de Matemática, Estatística e Ciência da Computação. Campinas: Universidade Estadual de Campinas, 1995. 68 p. (Dissertação, Mestrado em Ciência da Computação).

[Lor?] LORASCHI, A. et al. Distributed Genetic Algorithms with an Application to Portfolio Selection. *Notes*.

[LP86] LEVINE, P., POMEROL, J.-Ch. PRIAM, an interactive program for chosing among multiple attribute alternatives. *European Journal of Operational Research*, North-Holland: Elsevier Science, v. 25, n. 2, May 1986. p. 272-280.

[LSZ92] LOFTI, V., STEWART, T., ZIONTS, S. An aspiration-level interactive method for multiple criteria decision making. *Computers and Operations Research*, 1992.

[LT91] LOFTI, V., TEICH, J. Multicriteria decision making using persnal computers. In: *Multiple Criteria Decision Support*, Lecture Notes in Economics, v. 356, Berlin: Springer-Verlag, 1991.

[Mae94] MAEDA, T. Constraint Qualifications in Multiobjective Optimization Problems; Differentiable Case. *Journal of Optimization Theory and Applications*, [ ]: Plenum, v. 80, n. 3, Mar. 1994. p. 483-500.

[Mal89b] MALEK, M. et al. Serial and Parallel Search Techniques for the Traveling Salesman Problem. *Annals of OR: Linkages with Artificial Intelligence*, 1989.

[Mal89a] _____. A Fault Tolerant Implementation of the Traveling Salesman Problem. *Research Report*, Dept. of Electrical and Computer Engineering, University of Texas at Austin, 1989.

[Man89] MANBER, Udi. *Introduction to algorithms;* A Creative Approach. Massachusetts: Addison Wesley, 1989. 478 p.

[Mar92] MARGOT, F. Quick Updates for p-OPT TSP heuristics. *Operations Reseach Letters*, v. 11, 1992.

[MB87] MOSKOWITZ, H., BUNN, D. Decision and risk analysis. *European Journal of Operational Research*, v. 28, 1987. p. 247-260.

[Met53] METROPOLIS, N., et al. Equation of State Calculations by Fast Computing Machines. *Journal of Chem. Physics*, n. 21, 1953.

[MGK88] MÜHLENBEIN, H., GORGES-SCHLEUTER, M., KRÄMER, O. Evolution algorithms in combinatorial optimization. *Parallel Computing*, v. 7, 1988. p. 65-85.

[MM87] MOCKUS, J. B., MOCKUS, L. B. Some algorithms of global and multiobjective optimization and their computer implementation. *Optimal Decision Theory*, Vilnius: Institute of Math. and Cybernetics, v. 2, 1987.

[MM95] MAK, King-Tim, MORTON, Andrew J. Distances between traveling salesman tours. *Discrete Applied Mathematics - Combinatorial Algorithms Optimization and Computer Science*, North-Holland: Elsevier Science, v. 58, n. 3, Apr. 1995. p. 281-291.

[MM93] _____. A Modified Lin-Kernighan Traveling Salesman Heuristic. *Operations Research Letters*, v. 13, 1993. p. 127-132.

[MNT75] MAGAZINE, M. J., NEWHAUSER, G. L., TROTTER Jr, L. E. When the Greedy Solution Solves a Class of Knapsack Problems. *Operations Research*, v. 23, n. 2, Mar.-Apr. 1975.

[MOF92] MARTIN, O., OTTO, S. W., FELTEN, E. W. Large-step Markov Chains for the TSP incorporating local search heuristics. *Operations Research Letters*, v. 11, 1992. p. 219-224.

[Moc89] MOCKUS, Jonas. Bayesian approach to global optimization; Theory and Application. *Mathematical and its applications: Soviet series*. D. Reidel, 1989.

[MP91] MILLER, D. L., PECKNY, J. F. Exact solution of large asymmetric traveling salesman problem. *Science*, n. 251, 1991. p. 754-761.

[MP89] _____. Results from a parallel branch and bound algorithm for solving large asymmetric traveling salesman problem. *Operations Research Letters*, n. 8, 1989. p. 129-135.

[MR86] McCLELLAND, J. L., RUMELHART, D. E. (ed.) *Parallel Distributed Processing*. Cambridge, MA: MIT Press, v. 2, 1986.

[MSV91] MÜHLENBEIN, H., SCHOMISCH, M., BORN, J. The Parallel Genetic Algorithms as Function Optimizer. *Proceedings of the Fourth International Conference on Genetic Algorithms*, San Mateo, CA: Morgan Kaufmann, 1991.

[Müh90] MÜHLENBEIN, H. Parallel genetic algorithms; Populations Genetics and Combinatorial Optimization. *Proceedings of the Third International Conference on Genetic Algorithms*, San Mateo, CA: Morgan Kaufmann, 1989.

[Mül81] MÜLLER-MERBACH, H. Heuristics and their design: a survey. *European Journal of Operational Research*, v.1, n.8, 1981, p. 1-13.

[Mur92] MURTHY, Seshashayee S. *Synergy in Cooperating Agents;* Designing Manipulators from Task Specifications. Department of Electrical and Computer Engineering, Pittsburgh: Carnegie Mellon University, Sept. 1992. 306 p. (Thesis, Ph. D. in Electrical and Computer Engineering).

[New69] NEWELL, A. Heuristic programming: ill-structured problems. *Progress in Operations Research*, v. III, New York: John Wiley, 1969.

[NHH95] NAGAR, Amit, HADDOCK, Jorge, HERAGU, Sunderesh. Multiple and bicriteria scheduling: A literature survey. *European Journal of Operational Research*, North-Holland: Elsevier Science, 81, Feb. 1995. p. 88-104.

[Nic69] NICHOLSON, T. Optimization in Industry. *Optimization Techniques v. I*. London: Longman Press, 1971.

[NMS96] NASCIMENTO, H.A.D., MENDONÇA, C. F. X, SOUZA, P. S. de. Sinergia em Desenho de Grafos usando Springs e Pequenas Heurísticas. A ser publicado nos *Anais do XXIII Seminário Integrado de Software e Hardware*, Recife, PE, ago. 1996

[NR93] NADDEF, Denis, RINALDI, Giovanni. The graphical relaxation: A new framework for the Symmetric Traveling Salesman Polytope. *Mathematical Programming*, North-Holland: [ ], 58, 1993. p. 53-88.

[NW88] NEWHAUSER, George L., WOLSEY, Laurence A. *Integer and Combinatorial Optimization*. Wiley-Interscience series in discrete mathematics and optimization, Chichester, UK: John Wiley & Sons, 1988.

[OC91] OLSON, D., COURTNEY, J. F. *Decision Support Models and Expert Systems*, v. 5, 1991.

[OIM84] OHYA, T., IRI, M., MUROTA, K. Improvements of the Incremental Method for the Voronoi Diagram with Computational Comparasion of Various Algorithms. *Journal of the Operations Research Society of Japan*. v. 27, 1984. p. 306-337.

[Or76] OR, I. Traveling salesman-type combinatorial problems and their relations to the logistics of regional blood banking. Evanston, IL: Northwestern University, 1976. (Ph. D. Thesis).

[OS90] OGBU, F. A., SMITH, D. K. The Application of the Simulated Annealing Algorithm to the Solution of the $n/m/C_{max}$ Flow Shop Problem. *Computers Operation Research*, Great Britain: Pergamon Press, v. 17, n. 3, 1990. p. 243-253.

[OW78] OSTER, G. F., WILSON, E. O. *Caste and Ecology in the Social Insects*. Princeton, NJ: Princeton University Press, 1978.

[Pap90] PAPADIMITRIOU, C. H. The Complexity of the Lin-Kernighan Heuristic for the Traveling Salesman Problem. San Diego: University of California, 1990.

[Par96] PARETO, V. *Cours d'Economie Politique*. Rogue, Lousanne, 1896.

[Pei95] PEIXOTO, Hélvio Pereira. *Metodologia de Especificação de Times Assíncronos para Problemas de Otimização Combinatória*. Instituto de Matemática, Estatística e Ciência da Computação. Campinas: Universidade Estadual de Campinas, 1995. 127 p. (Dissertação, Mestrado em Ciência da Computação).

[Pet90] PETERSON, C. Parallel Distributed Approaches to Combinatorial Optimization: Benchmark Studies on Traveling Salesman Problem. *Neural Computation*, n. 2, 1990. p. 261-269.

[PFA95] PUREZA, V., FRANÇA, P. M., ARAGÃO, M. V. P. de. Modelo para Análise de Desempenho de Metaheurísticas Baseadas em Busca Tabu. *XXVII Simpósio Brasileiro de Pesquisa Operacional*, nov. 1995.

[PH80] PADBERG, Manfred W., HONG, Saman. On the Symmetric Traveling Salesman Problem: A Computacional Study. *Mathematical Programming Study 12 - Combinatorial Optimization*, Amsterdan: North-Holland Publishing Company, Apr. 1980. p. 78-107.

[PLC87] PLANTE, R. D., LOWE, T. J., CHANDRASEKARAN, R. The Product Matrix Traveling Salesman Problem: An Application and Solution Heuristics. *Operations Research*, v. 35, 1987. p. 772-783.

[Pol73] PÓLYA, George. *How to Solve it: A New Aspect of Mathematical Method*. Princeton, New Jersey: Princeton University Press, 2 ed., 1973.

[PPR95] PARDALOS, P., PITSOULIS, L., RESENDE, M. A parallel GRASP implementation for the quadratic assignment problem. In: *Parallel Algorithms for Irregularly Structured Problems - Irregular'94*. A. Ferreira, J. Rolim (eds.), Kluwer Academic, 1995.

[PR91] PADBERG, M., RINALDI, G. A Branch-and-Cut algorithm for the resolution of the large-scale symmetric traveling salesman problems. *SIAM Review*, v. 33, n. 1, 1991. p. 60-100.

[PR88] PARKER, R. Gary, RARDIN, Ronald L. *Discrete Optimization*. Computer Science and Scientific Computing. San Diego, MA: Academic Press, 1988. 472 p.

[PR87] PADBERG, M., RINALDI, G. Optimization of a 532-city symmetric Traveling Salesman Problem by Branch and Cut. *Operations Research Letters*, v. 6, n. 1, Mar. 1987.

[PS94b] PEIXOTO, Hélvio Pereira, SOUZA, Pedro Sérgio de. Uma Metodologia de Especificação de Times Assíncronos. *Anais do XXVI Simpósio Brasileiro de Pesquisa Operacional - SBPO*, Florianópolis, SC, nov. 1994. p. 284-289.

[PS94a] _____. Times Assíncronos: uma nova técnica para o Flow Shop Problem. *Anais do XXI Seminário Integrado de Software e Hardware - SEMISH*. Caxambu, MG, jul. 1994.

[PS85] PREPARATA, Franco P., SHAMOS, Michael Ian. *Computacional Geometry*; An Introduction. 2. ed. New York: Springer-Verlag New York, 1985. 398 p.

[PS82] PAPADIMITRIOU, Christos H., STEIGLITZ, Kenneth. *Combinatorial Optimization*: Algorithms and Complexity. Prentice-Hall, 1982.

[PS75] _____. *On the Complexity of Local Search for the Traveling Salesman Problem*. New Jersey: Princeton, 1975.

[PTW83] PÓLYA, G. TARJAN, R. E., WOODS, Donald R. *Notes on Introductory Combinatorics*. Progress in Compute Science, Birkhäuser Boston, n. 4, 1983.

[PY89] PANG, Jong-Shi, YU, Chang-Sung. A min-max resource allocation problem with substitutions. *European Journal of Operacional Research*, North-Holland: Elsevier Science, 41, 1989. p. 218-223.

[Pyo85] PYO, S. S. Asynchronous *Algorithms for Distributed Processing*. Department of Electrical and Computer Engineering. Pittsburg, PA: Carnegie Mellon University, 1991. (Thesis, Ph. D. in Electrical and Computer Engineering).

[Qua91] QUADREL, R. W. Asynchronous *Design Environments: Architecture and Behavior*. Department of Architecture. Pittsburg, PA: Carnegie Mellon University, 1991. (Thesis, Ph. D. in Architecture).

[QW93] QUEYRANNE, Maurice, WANG, Yaoguang. Hamiltonian path and symmetric traveling salesman polytopes. *Mathematical Programming*, North-Holland: [ ], 58, p. 89-110, 1993.

[Ram94] RAMESH, V.C. Inertial Search and Asynchronous Decompositions. Technical Report EDRC 02-32-94. Carnegie Mellon University, 1995.

[Ras86] RASMUSSEN, M. L. Zero-one programming with multiple criteria. *European Journal of Operational Research*, v. 26, 1986. p. 83-95.

[Rei94] REINELT, G. Contributions to Practical Traveling Salesman Problem Solving. *Lecture Notes in Scientific Computing*, Springer, 1994.

[Rei92] _____. Fast Heuristics for Large Geometric Traveling Salesman Problems. *ORSA Journal on Computing*, v. 2, 1992. p. 206-217.

[RFS95] RESENDE, M., FEO, T., SMITH, S. FORTRAN subroutines for approximate solution of maximum independent set problems using GRASP. *Technical Report*, AT&T Bell Laboratories, Murray Hill, NJ, 1995.

[Rib96] RIBEIRO, C. C. Heurísticas para Problemas Combinatórios. *3ra. Escuela Latinoamericana de Investigación Operativa*. Bariloche, Argentina, jan. 1996.

[RiS95] RIBEIRO, W. E., SOUZA, P. S. de. Heurísticas de Construção para o Prize Collecting TSP. *XXVII Simpósio Brasileiro de Pesquisa Operacional*, nov. 1995.

[RKZ90] RAMESH, R., KARWAN, Mark H., ZIONTS, Stanley. An Interative Method for Bicriteria Integer Programming. *IEEE Transation on SMC*, v. 20, n. 2, Mar.-Apr. 1990.

[Rom95] ROMMELFANGER, H. Fuzzy Linear Programming and Applications. *European Journal of Oper. Research*, 1995..

[Rom91] _____. FULP-A PC-Supported Procedure for Solving Multicriteria Linear Programming Problems With Fuzzy Data. *Lecture Notes in Economics and Mathematical Systems*, n. 368, Berlin: Springer-Verlag, 1991.

[Rom90] _____. FULPAL - An Interactive Method for Solving (Multiobjective) Fuzzy Linear Programming Problems. *Stochastic versus Fuzzy Approaches to Multiobjective Programming under Uncertainty*, Netherlands: Kluwer Academic Publishers, 1990. p. 279-299.

[Rom86] ROMERO, Carlos. A survey of generalized goal programming (1970-1982). *European Journal of Operational Research*, North-Holland: Elsevier Science, v. 25, n. 2, May 1986. p. 183-191.

[Rou82] ROUBENS, M. Preference relations on actions and criteria in multicriteria decision making. *European Journal of Operational Research*, v. 10, 1982. p. 51-55.

[Roy85] ROY, B. *Méthodologie Multicritère d'Aide à la Décision*. Ecomica, Paris, 1985.

[RR95] RESENDE, M. G. C., RIBEIRO, C. C. A GRASP for Graph Planarization. *Technical Report*, Oct. 1995.

[RS96b] RODRIGUES, Rosiane de Freitas, SOUZA, Pedro Sérgio de. Manipulando Conjuntos de Soluções Não-Dominadas através de Times Assíncronos: uma aplicação ao Problema do Caixeiro Viajante Multiobjetivo. A ser publicado nos *Anais do VIII Latin-Iberian-American Congress on Operations Research and System Engineering - CLAIO / XXVIII Simpósio Brasileiro de Pesquisa Operacional - SBPO*, Rio de Janeiro: SOBRAPO, ago. 1996.

[RS96a] _____. Algumas Estratégias para a Resolução de Problemas Multiobjetivos através de Times Assíncronos. *3ra. Escuela Latinoamericana de Verano de Investigacion Operativa*, Bariloche, Argentina, jan. 1996.

[RS95c] _____. Times Assíncronos: uma Abordagem Multi-Algorítmica para Problemas de Otimização Combinatória com Múltiplos Objetivos. *Anais do XXVII Simpósio Brasileiro de Pesquisa Operacional*, Vitória, ES, nov. 1995.

[RS95b] _____. *Asynchronous Teams for Solving Combinatorial Optimization Problems*. Relatório Técnico do Departamento de Ciência da Computação DCC-18-95, Campinas, SP: Universidade Estadual de Campinas, nov. 1995. 11 p.

[RS95a] _____. Asynchronous Teams: a Multi-Algorithm Approach for Solving Combinatorial Multi-Objective Optimization Problems. *Proceedings of the 5th Workshop of the DGOR-Working Group: Multicriteria Optimization and Decision Theory*. Pfalzakademie, Dietmar Schweigert (ed.). Lambrecht, Germany: Universität, Mar. 1995. p. 9-20.

[RS94] _____. *Times Assíncronos para Problemas de Otimização Combinatória com Multiplas Funções Objetivo*. Instituto de Matemática, Estatística e Ciência da Computação. Campinas: Universidade Estadual de Campinas, set. 1994. (Proposta de Dissertação, Ciência da Computação). 32 p.

[RSL77] ROSENKRANTZ, D. J., STEARNS, R. E., LEWIS, P. M. An analysis of several heuristics for the traveling salesman problem. *SIAM J. Computing*, v. 6, 1977. p. 563-581.

[Rut89] RUTENBAR, R. A. Simulated annealing algorithms: an overview. *IEEE Circuits and Devices Magazine*, v. 5, n. 1, Jan. 1989.

[RW91] ROY, A., WALLENIUS, J. Nonlinear and unconstrained multiple-objective optimization: Algorithm, computation and application. *Naval Research Logistics*, v. 38, 1991. p. 623-635.

[RW87] ROY, T.J. Van, WOLSEY, L.A. Solving Mixed integer programming problems using automatic reformulation. *Operations Research*, v. 35, 1987. p. 45-57.

[Saa80] SAATY, T. L. *The Analytic Hierarchy Process*. New York: MgGraw Hill, 1980.

[Sav90] SAVELSBERGH, M. W. P. An efficient implementation of local search algorithms for constrained routing problems. *European Journal of Operational Research*, North-Holland: Elsevier Science, 47, 1990. p. 75-85.

[SC83] STEUER, R. E., CHOO, E. U. An interactive weighted Tchebycheff procedure for multiple objective programming. *Mathematical Programming*, v. 26, 1983. p. 326-344.

[SDK83] SYSLO, M. M., DEO, N., KOWALIK, J. S. *Discrete Optimization Algorithms with Pascal Programs*. Englewood Cliffs, NJ: Prentice Hall, 1983.

[Sev] SEVERINO, A. J. *Metodologia do Trabalho Científico*. São Paulo: Cortez, 18 ed., 1992.

[Sed90] SEDGEWICK, Robert. *Algorithms in C*. Massachusetts: Addison Wesley, 1990. 657 p.

[Sen85] SENGUPTA, J. K. *Optimal Decisions Under Uncertainty*: Methods, Models and Management. Heidelberg, Germany: Springer Verlag, 1985. 286 p.

[SF96] SOUZA, P. S. de, FAVILLA, J. R. Asynchronous Teams for Steel Industry: Mini Mills.To appear in: *International Conference on Information, Systems, Analysis and Synthesis*, Jul. 1996.

[SGD86] SZIDAROVSZKY, Ferenc, GERSHON, Mark E., DUCKSTEIN, Lucien. *Techniques for Multiobjective Decision Making in Systems Management*. Advances in Industrial Engineering, 2, Netherlands: Elsevier Science, 1986. 506 p.

[Sha76] SHAPIRO, J.F. Multiple Criteria Public Investment Decision Making by Mixed Integer Programming. *Lecture Notes in Economics and Mathematical Systems*, n. 130, Berlin> Springer-Verlag, 1976. p. 170-182.

[Shi91] SHI, D. S. Contigent Derivative of the Perturbation Map in Multiobjective Optimization. *Journal of Optimization Theory and Applications*, [ ]: Plenum, v. 70, n. 2, Aug. 1991. p. 385-396.

[SNT85] SAWARAGI, Yoshikazu, NAKAYAMA, Hirotaka, TANINO, Tetsuzo. *Theory of Multiobjective Optimization.* Mathematics in Science and Engineering, 176. Orlando, Florida: Academic Press, 1985. 296 p.

[SO87] SHAFRITZ, J. M., OTT, J. S. (ed.). *Classics of Organization Theory.* Chicago, IL: Dorsey Press, 1987.

[Soi88] SOISMAA, M. Game theoretic analysis of the Finnish timber market. *Helsinki School of Economics,* Series A:62, Helsinki, 1988.

[Sou94] SOUZA, Pedro Sérgio de. Notas de Aula. In: *Tópicos em Teoria da Computação.* Campinas: Universidade Estadual de Campinas, 1994.

[Sou93] _____. *Asynchronous Organization for Multi-Algorithm Problems.* Department of Electrical and Computer Engineering, Pittsburgh: Carnegie Mellon University, Apr. 1993. 139 p. (Thesis, Ph. D. in Electrical and Computer Engineering).

[SR86] SADAGOPAN, S., RAVINDRAN, A. Interactive algorithms for multiple criteria nonlinear programming problems. *European Journal of Operational Research,* North-Holland: Elsevier Science, v. 25, n. 2, May 1986. p. 247-257.

[SR91] SHIN, W. S., RAVINDRAN, A. An Interative Method for Multiple-Objective Mathematical Programming Problems. *Journal of Optimization Theory and Applications,* [ ]: Plenum, v. 68, n. 3, Mar. 1991. p. 539-561.

[SRB92] SHIN, W. S., RAMACHANDRAN, A., BULLINGTON, S. F. Interactive Bicriterion Solution Method and Its Application to Critical Path Method Problems. *Journal of Optimization Theory and Applications,* [ ]: Plenum, v. 72, n. 3, Mar. 1992. p. 511-527.

[ST93b] SOUZA, Pedro Sérgio de, TALUKDAR, Sarosh N. *Comunicação pessoal,* 1993.

[ST93a] _____. Asynchronous Organization for Multi-Algorithm Problems. In: *Proceedings of the 1993 ACM Symposium on Applied Computing,* Indianapolis, IN, Feb. 1993.

[ST91] _____. Genetic Algorithms in Asynchronous Teams. *Proceedings of the Fourth International Conference on Genetic Algorithms,* Los Altos: Morgan Kaufmann, 1991.

[STo93] SALUKVADZE, M. E., TOPCHISHVH. Weakly-Efficient Solutions of Limiting Multicriteria Optimization Problems. *Journal of Optimization Theory and Applications,* [ ]: Plenum, v. 77, n. 2, May 1993. p. 373-386.

[Ste86] STEUER, R.E. *Multiple Criteria Optimization: Theory, Computation and Application.* New York: Wiley, 1986. 546 p.

[Ste85] STEELE, J.M. Probabilistic algorithm for the directed traveling salesman problem. *Math. Oper. Research,* 1985.

[SVW80] SILVER, Edward A., VIDAL, R. Victor V., WERRA, Dominique de. A tutorial on heuristic methods. *European Journal of Operational Research,* [ ]: North-Holland Publishing Company, 5, 1980. p. 153-162.

[Tai93] TAILLARD, E. Benchmarks for basic scheduling problems. *European Journal of Operational Research.* v. 64, n.2, 1993. p. 278-285.

[Tai90] _____. Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research* , North-Holland: Elsevier Science, v. 47, n. 1, July 1990.

[Tak92] TAKEFUJI, Y. *Neural Network parallel computing.* Boston: Kluwer Academic, 1992.

[Tal93] TALUKDAR, Sarosh. Asynchronous Teams. *Fourth Symposium on Expert Systems Application to Power Systems.* Melbourn, Austrália, Jan. 1993.

[Tal95] TALUKDAR, Sarosh, SOUZA, P. S. de, QUADREL, R., RAMESH, V.C. A-Teams: Multi-Agent Organizations for Distributed Iteration. Technical Report EDRC 18-30-95. Carnegie Mellon University, 1995.

[Tal92] TALUKDAR, Sarosh. Asynchronous Teams. *Fourth Symposium on Expert Systems Application to Power Systems*. Melbourn, Austrália, Jan. 1993.

[Tan92] TANENBAUM, A. *Modern Operating Systems*. New Jersey: Prentice-Hall, 1992, 728 p.

[TBGS96] TALUKDAR, S. N., Baerentzen, L., GOVE, A., Souza, P. S. de. Cooperation Schemes for Autonomous Agents. A ser publicado. 1996.

[TCH91] TROUTT, M., CLINTON, R. J., HEMMING, T. Method of Forces for Direction Finding in Interactive Multicriterion Optimization Applications. *Journal of Optimization Theory and Applications*, Plenum, v. 68, n. 3, Mar. 1991. p. 583-601.

[TGS95] TALUKDAR, Sarosh, GOVE, A., SOUZA, P. S. de. Asynchronous Teams: Near-Scale-Effective Organizations for Distributed, Computer-Based Agents. Technical Report EDRC 05-94-95. Carnegie Mellon University, 1995.

[TK86] TEGHEM, J., KUNSCH, P. L. A survey of techniqes for finding efficient solutions to multiobjective integer linear programming. *Asia-Pacific Journal of Operational Research*, v. 3, 1986. p. 95-108.

[TK85] _____. Interactives methods for multiobjective integer linear programming. In: *Large Scale Modelling and Interactive Decision Analysis*. G. Fandel, M. Grauer, A. Kurzhanski, A. Wierzbicki (eds.). Berlin: Springer-Verlag, 1985. p. 75-87.

[TK84] TAYLOR, H. M., KARLIN, S. *An Introduction to Stochastic Modeling*. Boston, MA: Academic Press, 1984.

[TR93] TALUKDAR, S. N., RAMESH, V. C. A parallel global optimization algorithm and its application to the CCOPF problem. *Proceedings of the Power Industry Computer Applications*. Phoenix, May 1993.

[TR92c] _____. Cooperative Methods and Security Planning. Report EDRC 18-38-92. Carnegie Mellon University, 1992.

[TR92b] _____. Contingency Constrained Operations. Report EDRC 18-36-92. Carnegie Mellon University, 1992.

[TR92a] _____. A-Teams for Real-Time Operations. Technical Report EDRC 18-34-92. Carnegie Mellon University, 1992.

[TRQC92] TALUKDAR, S. N., RAMESH, V. C. QUADREL, R., CHRISTIE, R. Multiagent Organizations for Real-Time Operations. *Proceedings of the IEEE*, v. 80, n. 5, May 1992.

[Tra76] TRAUBER, J. F. *Algorithms and Complexity*; New Directions and Recent Results. New York: Academic Press, 1976.

[TS93] TALUKDAR, Sarosh N., SOUZA, Pedro S. Objects Organizations and Super-Objects. In: *Engineering Design: the creation of products and process*. Pittsburg, PA: McGraw Hill, 1993.

[TS92] _____. Scale Efficienty Organizations. In: *Proceedings of the 1992 IEEE International Conference on Systems, Man and Cybernetics*. Chicago, IL , Oct. 1992.

[TS90] _____. Asynchronous Teams. In: *Second SIAM Conf. on Linear Algebra: Signals, Systems, and Control*, San Francisco, CA, Nov. 1990.

[TSM93] TALUKDAR, Sarosh N., SOUZA, Pedro S., MURTHY, S. Designing Organizations for Computer-Based Agents. *International Journal of Engineering Intelligent Systems for Electrical Engineering and Communications*, 1993.

[Uld90] ULDER, N. L. J. et al. Improving TSP Exchange Heuristics by Population Genetics. Preprint, Erasmus Universiteit Rotterdam, 1990.

[Vin86] VINCKE, Philippe. Analysis of multicriteria decision aid in Europe. *European Journal of Operational Research*, North-Holland: Elsevier Science, v. 25, n. 2, May 1986. p. 160-168.

[VK81b] VILLARREAL, B., KARWAN, M.H. Multicriteria Integer Programming: A (Hybrid) Dynamic Programming Recursive Approach. *Mathematical Programming*, v. 21, n. 2, 1981. p. 204-223.

[VK81a] _____. Parametric Multicriteria Integer Programming. In: *Studies on Graphs and Discrete Programming*, Amsterdan: North-Holland, 1981. p. 371-379.

[VV89] VANDERPOOTEN, D., VINCKE, P. Description and analysis of some representative interactive multicriteria procedures. *Mathematical and Computer Modelling*, v. 12, 1989. p. 1221-1238.

[Wal91] Wallenius, H. Implementating interactive multiple criteria decision methods in public policy. *Studies in Computer Science, Economics and Statistics*, v. 18, Finland: University of Jyväskylä, 1991.

[Wan93] WANG, S. Y. Existence of a Pareto Equilibrium. *Journal of Optimization Theory and Applications*, [ ]: Plenum, v. 79, n. 2, Nov. 1993. p. 373-384

[Wei75] WEINER, P. Heuristics. *Networks*, v. 5, n. 1, 1975. p. 101-103.

[WH89b] WIDMER, Marino, HERTZ, Alain. A new heuristic method for the flow shop sequencing problem. *European Journal of Operacional Research*, North-Holland: Elsevier Science, v. 41, n. 2, July 1989. p. 186-193.

[WH89a] WERRA, D. de, HERTZ, A. Tabu Search Techniques: A tutorial and an Application to Neural Networks. *OR Spektrum*, [ ]: Springer-Verlag, 11, 1989. p. 131-141.

[Whi90] WHITE, D. J. A Bibliography on the applications of mathematical programming multiple-objective methods. *Journal of the Operations Research Society*, v. 41, 1990. p. 669-691.

[Wie80] WIERZBICKI, A. The use of reference objectives in multiobjective optimization. In: *Multiple Criteria Decision Making, Theory and Application*. G. Fandel, T. Gal (eds.), Berlin: Springer-Verlag, 1980. p. 468-486.

[Wil90] WILLIANSON, D. P. Analysis of the Held-Karp Heuristic for the Traveling Salesman Problem. Massachusetts Institute of Technology, MA, 1990. (Dissertation, Master in Electrical Engineering and Computer Science).

[WP88] WILSON, G. V., PAWLEY, G. S. On the stability of the traveling salesman problem algorithm of Hopfield and Tank. *Biol. Cybernet*, n. 58, 1988.

[WY91] WANG, S. Y., YANG, F. M. Techical Note: A Gap between Multiobjective Optimization and Scalar Optimization. *Journal of Optimization Theory and Applications*, [ ]: Plenum, v. 68, n. 2, Feb. 1991. p. 389-391.

[WZ76] WALLENIUS, J., ZIONTS, S. Some tests of an interactive programming method for multicriteria optimization and an attempt at implementation. In: *Multiple Criteria Decision Making*. Berlin: Springer-Verlag, 1976. p. 319-331.

[Yu85] YU, P. L. *Multiple Criteria Decision Making: Concepts, Techniques, and Extensions*. New York: Plenum, 1985.

[Zel76] ZELENY, M (ed.). *Multiple Criteria Decision Making*. (Lectures notes in economics and mathematical systems, 123). Heidelberg: Springer-Verlag Berlin, 1976. 345 p.

[Zel82] _____. *Multiple Criteria Decision Making*. New York: McGraw-Hill, 1982.

[ZE81] ZANAKIS, Stelios H., EVANS, James R. Heuristic "Optimization": Why, When, and How to Use It. *Interfaces*, v. 11, n.5, Oct. 1981.

[ZEV89] ZANAKIS, Stelios H., EVANS, James R., VAZACOPOULOS, Alkis A. Heuristic methods and applications: A categorized survey. *European Journal of Operational Research*, North-Holland: Elsevier Science, 43,1989. p. 88-110.

[Zio79] ZIONTS, S. A survey of multiple criteria integer programming methods. *Annals of Discrete Mathematics*, n. 5, 1979. p. 389-398.

[ZYW92] ZHANG, D., YU, P. L., WANG, P. Z. State-Dependent Weights in Multicriteria Value Functions. *Journal of Optimization Theory and Applications*, [ ]: Plenum, v. 74, n. 1, July 1992. p. 1-21.