UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE CIÊNCIAS EXATAS E DA TERRA
PROGRAMA DE PÓS-GRADUAÇÃO
EM MATEMÁTICA APLICADA E ESTATÍSTICA

JOÃO SATURNINO DA SILVA NETO

APLICAÇÃO DAS TÉCNICAS *PATH-RELINKING* E *VOCABULARY BUILDING* NA MELHORIA DE PERFORMANCE DO ALGORITMO MEMÉTICO PARA O PROBLEMA DO CAIXEIRO VIAJANTE ASSIMÉTRICO

NATAL
2009

JOÃO SATURNINO DA SILVA NETO

APLICAÇÃO DAS TÉCNICAS *PATH-RELINKING* E *VOCABULARY BUILDING* NA MELHORIA DE PERFORMANCE DO ALGORITMO MEMÉTICO PARA O PROBLEMA DO CAIXEIRO VIAJANTE ASSIMÉTRICO

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação de Matemática Aplicada e Estatística da UFRN, como requisito parcial para a obtenção do título de Mestre em Matemática Aplicada e Estatística.
Área de concentração: Otimização.

Orientador: Prof. Dr. Dario José Aloise.

NATAL
2009

Catalogação da Publicação na Fonte. UFRN / SISBI / Biblioteca Setorial
Centro de Ciências Exatas e da Terra CCET.

Silva Neto, João Saturnino da.
Aplicação das técnicas path-relinking e vocabulary building na melhoria de performance do algoritmo memético para o problema do caixeiro viajante assimétrico / João Saturnino da Silva Neto. -- Natal, 2009.
80f. il.:

Orientador: Prof. Dr. Dario José Aloise.

Dissertação (Mestrado) - Universidade Federal do Rio Grande do Norte. Centro de Ciências Exatas e da Terra. Programa de Pós-Graduação em Matemática Aplicada e Estatística.

1. Algoritmo memético - Dissertação. 2. Problema do caixeiro viajante assimétrico (PCV) - Dissertação. 3. Otimização combinatória. I. Aloise, Dario José. II. Título.

RN/UF/BSE-CCET CDU - 519.688

JOÃO SATURNINO DA SILVA NETO

APLICAÇÃO DAS TÉCNICAS *PATH-RELINKING* E *VOCABULARY BUILDING* NA MELHORIA DE PERFORMANCE DO ALGORITMO MEMÉTICO PARA O PROBLEMA DO CAIXEIRO VIAJANTE ASSIMÉTRICO

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação de Matemática Aplicada e Estatística da UFRN, como requisito parcial para a obtenção do título de Mestre em Matemática Aplicada e Estatística.
Área de concentração: Otimização.

Aprovada em: 10/07/2009.

BANCA EXAMINADORA

Prof. Dr. DÁRIO JOSÉ ALOISE
Orientador (Departamento de informática e Matemática Aplicada-UFRN)

Prof. Dr. FRANCISCO CHAGAS DE LIMA JÚNIOR
Examinador Externo (Departamento de Informática-UERN)

Prof. Dr. MARCELO GOMES PEREIRA
Examinador Interno (Departamento de Matemática-UFRN)

"A Maior Calamidade da
Humanidade é a injustiça, pois
possui armas nas mãos."
Aristóteles

Aos Meus pais, Luiza e Severino.
A Vanessa, com muito amor.

Agradecimentos

A Deus por ter guiado os meus passos, iluminado os meus caminhos, me confortado nos momentos de angústia e me dado fé e saúde necessárias para a conclusão de mais uma de muitas jornadas.

À minha família, em especial aos meus pais: Luiza Eulália e Severino Ramos, por toda a dedicação no processo criação, pelo exemplo de vida, carinho e Amor.

À minha noiva, Vanessa Danielle, pelo apoio, amizade, carinho, paciência e compreensão de sempre, fundamentais para o desenvolvimento deste trabalho e outros objetivos.

Ao Professor Dario José Aloise meus agradecimentos, minha admiração e meu respeito. Deus sabe o privilégio que tive ao ser aceito sob sua orientação.

A professora Dione, pelo apoio e encorajamento nos momentos de dificuldade, tenho uma eterna gratidão.

A todos os meus professores que contribuíram e contribuem com a minha formação não apenas de aluno, mas de cidadão.

Ao professor Paulo Sergio, meu co-orientador de coração, por ter contribuído na minha formação de mestre sempre apresentando com entusiasmo e experiência assuntos coerentes com o meu trabalho.

Aos meus amigos de mestrado pelo apoio e puxões de orelha, alunos da graduação pelos belos momentos de reflexão e conhecimentos partilhados .

Aos amigos do Programa de Ensino Tutorial e do Karatê, em especial ao Sensei Henrique por me ajudar na formação do caráter.

Aos meus Amigos do laboratório PROMETH em especial, Alisson Guedes, cuja ajuda foi de fundamental importância no desenvolvimento deste trabalho.

À Universidade Pública e Gratuita, que permitiu que eu chegasse até aqui.

A todas as pessoas que diretamente ou indiretamente fizeram parte da minha vida seja acadêmica ou não, contribuindo nos momentos de autoconhecimento e fé na humanidade.

Resumo

O presente trabalho propõe estratégias de melhoria em uma bem sucedida metaheurística evolucionária para a resolução do Problema do Caixeiro Viajante Assimétrico. Tal procedimento consiste em um algoritmo memético projetado especificamente para esse problema. Essas melhorias têm por base a aplicação de técnicas de otimização conhecidas como *Path-Relinking* e *Vocabulary Building*, sendo essa última técnica utilizada de dois modos distintos, com o intuito de avaliar os efeitos de melhoria sobre a metaheurística evolucionária empregada. Os métodos propostos foram implementados na linguagem de programação C++ e os experimentos computacionais foram realizados sobre instâncias disponibilizadas na biblioteca TSPLIB, tornando possível observar que os procedimentos propostos alcançaram êxito nos testes realizados.

PALAVRAS-CHAVE: Algoritmo Memético. *Path-Relinking*. *Vocabulary Building*. Problema do Caixeiro Viajante Assimétrico.

Abstract

The present essay shows strategies of improvement in a well succeded evolutionary metaheuristic to solve the Asymmetric Traveling Salesman Problem. Such steps consist in a Memetic Algorithm projected mainly to this problem. Basically this improvement applied optimizing techniques known as Path-Relinking and Vocabulary Building. Furthermore, this last one has being used in two different ways, in order to evaluate the effects of the improvement on the evolutionary metaheuristic. These methods were implemented in C++ code and the experiments were done under instances at TSPLIB library, being possible to observe that the procedures purposed reached success on the tests done.

KEYWORDS: Memetic Algoritm. Path-Relinking. Vocabulary Building. Asymmetric Travelling Salesman Problem.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

Sumário

Capítulo 1

Introdução

A área de pesquisa Otimização Combinatória tem como uma das metas, capacitar profissionais com métodos ou técnicas científicas que são acoplados em sistemas computacionais de apoio a decisão, os quais são utilizados com o objetivo de maximizar/minimizar o lucro/custo de empresas. Na maioria dos problemas reais, entretanto, a solução exata de problemas de otimização combinatória, exige um tempo computacional considerado inviável, necessitando assim de apenas um bom resultado aproximado em vez de resultados ótimos, favorecendo o uso cada vez mais freqüentes de métodos heurísticos e metaheuristicas na resolução de problemas dessa natureza. Esses problemas são classificados como NP-difíceis. O Problema do Caixeiro Viajante (PCV) tratado neste trabalho é um exemplo dessa categoria.

A escolha por esse problema se deu pelo fato de ser um dos problemas de Otimização Combinatória mais conhecidos e discutidos até hoje, pela sua importância quanto a complexidade computacional e por servir de modelo para aplicações práticas em muitos problemas reais. De fato, o PCV há décadas vem servindo como referência na área, inclusive como teste inicial para novas idéias de resolução de problemas combinatórios na forma exata ou aproximativa.

Especificamente, neste trabalho, foram aplicadas as técnicas ou estratégias *path-relinking* e *vocabulary building* na melhoria da performance de uma bem sucedida metaheurística evolucionária (algoritmo memético) e implementadas para o Problema do Caixeiro Viajante Assimétrico (PCVA). Este algoritmo evolucionário foi proposto no artigo "A New Memetic Algorithm for the Asymmetric Traveling Salesman Problem" (Buriol, França

e Moscato (2004)) e é tido, até o momento de finalização desta pesquisa, como sendo a melhor metaheurística existente para o PCVA, apresentando resultados muito bons em diversas instâncias do problema em questão - incluindo todas as instâncias assimétricas da famosa biblioteca TSPLIB[1].

O principal objetivo desta dissertação, portanto, consiste em melhorar a performance do algoritmo memético citado, que considerou-se neste trabalho como Estado-da-Arte. Para isso foram utilizados procedimentos de Reconexão por Caminhos (*Path-Relinking* - PR) e Construção de Vocábulários (*Vocabulary Building* - VB), este último implementado de duas formas diferentes.

O restante desse trabalho é organizado da seguinte forma: O Capítulo 2 é dedicado ao problema do caixeiro viajante (*Traveling Salesman Problem* - TSP) e tem o objetivo de conceituar e caracterizar o problema, visto que durante toda a dissertação esse problema é tomado como exemplo. Neste capítulo, o problema é conceituado, a formulação matemática e a complexidade do mesmo são apresentadas, algumas aplicações práticas são relatadas e são apresentadas, para esse trabalho. Além disso, são apresentados os métodos de resolução do problema através das metaheuristicas algoritmos genéticos, meméticos e das técnicas de *Path Relinking* e do *Vocabulary Building*. Uma revisão bibliográfica do mesmo é desenvolvida com o objetivo de situar adequadamente as contribuições propostas nesta dissertação. O Capítulo 3 descreve o algoritmo memético Estado da Arte. O Capítulo 4 descreve as implementações desse Algoritmo aplicado ao PCVA. No Capítulo 5 são apresentados os experimentos computacionais bem como seus resultados. Por fim, no Capitulo 6, são apresentas as conclusões da dissertação e algumas sugestões para trabalhos futuros.

[1]http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95

Capítulo 2

Problema do Caixeiro Viajante

O PCV (do inglês *Traveling Salesman Problem* (TSP)), é um exemplo de problemas de otimização combinatória, cuja finalidade é a determinação da rota que representa o menor custo para percorrer as cidades envolvidas no problema, uma única vez, regressando à cidade de origem.

A origem do nome *travelling salesman problem* é desconhecida, pelo que parece não existir qualquer documento que prove o autor do nome do problema. No entanto, Merril Flood, um dos investigadores mais influentes nas primeiras aplicações do problema, da Universidade de Princeton, proferiu o seguinte comentário: *I don't know who coined the peppier name "Traveling Salesman Problem" for Whitney's problem,...* (Applegate et al., cop. 2006, p.2).

Excluindo pequenas variações ortográficas, como *traveling* vs *travelling* ou *salesman* vs *salesman*'s, o nome do problema ficou globalmente conhecido por volta do ano 1950 (Applegate et al., cop. 2006, p.2).

No âmbito da logística, o PCV é largamente utilizado para a movimentação de equipamentos, pessoas e veículos.

Este capítulo tem como meta explicar mais detalhadamente o PCV, bem como as técnicas utilizadas neste trabalho para resolver o PCVA, além de mostrar as algumas aplicações do PCV e finaliza com uma revisão bibliográfica do problema analisando os re-

sultados obtidos por algoritmos exatos e heurísticos aplicados ao problema em questão, bem como aspectos relevantes sobre implementações e estudos teóricos do problema.

2.1 Definição

Dado um número de cidades, com os respectivos custos de viagem entre as mesmas, o PCV é utilizado com o objetivo de se obter o ciclo hamiltoniano mais econômico, que envolva a passagem por todas as cidades, uma única vez, e o regresso ao ponto de partida.

Este problema, apesar de parecer modesto, é na realidade um dos mais investigados na área de Otimização Combinatória, quer por cientistas, matemáticos ou pesquisadores de diversas áreas, tais como: logística, genética, manufatura, entre outros.

Embora a definição do problema seja normalmente referenciada para um problema de minimização, o problema também é encontrado na sua forma de maximização.

2.2 Formulação matemática

Dado um conjunto de n cidades c_i designadas como $C = \{c_1, ..., c_n\}$ e uma matriz de distâncias (ρ_{ij}), onde $\rho_{ij} = \rho(c_i, c_j)$ $(i, j \in \{1, ..., n\},\ \rho_{ij} = \rho_{ji},\ \rho_{ii} = 0)$, de forma a desenvolver permutações na seguinte forma:

$$\pi \in S_n = \{s : \{1, ..., n\} \to \{1, ..., n\}\}.$$

Dentre todas as permutações encontradas, a solução para o problema consiste naquela que minimiza a seguinte função objetivo (distância do circuito):

$$f : S_n \to \mathbb{R}, \qquad f(\pi) = \sum_{i=1}^{n-1} \rho\left(\pi(i), \pi(i+1)\right) + \rho\left(\pi(n), \pi(1)\right).$$

Salienta-se ainda a importância do tamanho do espaço de busca, nas permutações, que aumenta exponencialmente com o aumento de um número n de cidades:

$$\frac{(n-1)!}{2} \approx \frac{1}{2}\sqrt{2\pi(n-1)}\left(\frac{n-1}{e}\right)^{n-1}$$

rotas possíveis.

Apesar de vários pesquisadores proporem bons algoritmos aproximativos, para a resolução do problema do caixeiro viajante, o problema continua a oferecer grande atração para a aplicação de novos algoritmos heurísticos. Isto se deve essencialmente ao fato do problema ser:

- Facilmente perceptível, uma vez que se assemelha a problemas reais;
- Complexo, por se tratar de um problema representativo de uma larga escala de problemas da classe NP (*non-deterministic polynomial time*), conhecidos por não terem soluções através de algoritmos determinísticos[1].

2.3 Formulação por grafos

Na teoria matemática dos grafos cada cidade é identificada com um nó (ou vértice) e as linhas que ligam cada par de nós são identificadas como arcos (arestas ou setas). A cada uma destas linhas estarão associadas às distâncias (ou custos) correspondentes. Desde que seja possível ir diretamente de uma cidade para qualquer outra, o grafo diz-se completo.

Dependendo da importância que poderá ter o sentido das setas, o PCV pode-se distinguir em simétrico ou assimétrico. A figura 2.1 a seguir mostra graficamente a representação de um grafo para um PVC e suas rotas possíveis.

[1]Consideremos como sendo algoritmos que quando estão em um certo estado, processando um certo dado, podem se movimentar de uma única maneira rumo à próxima configuração.

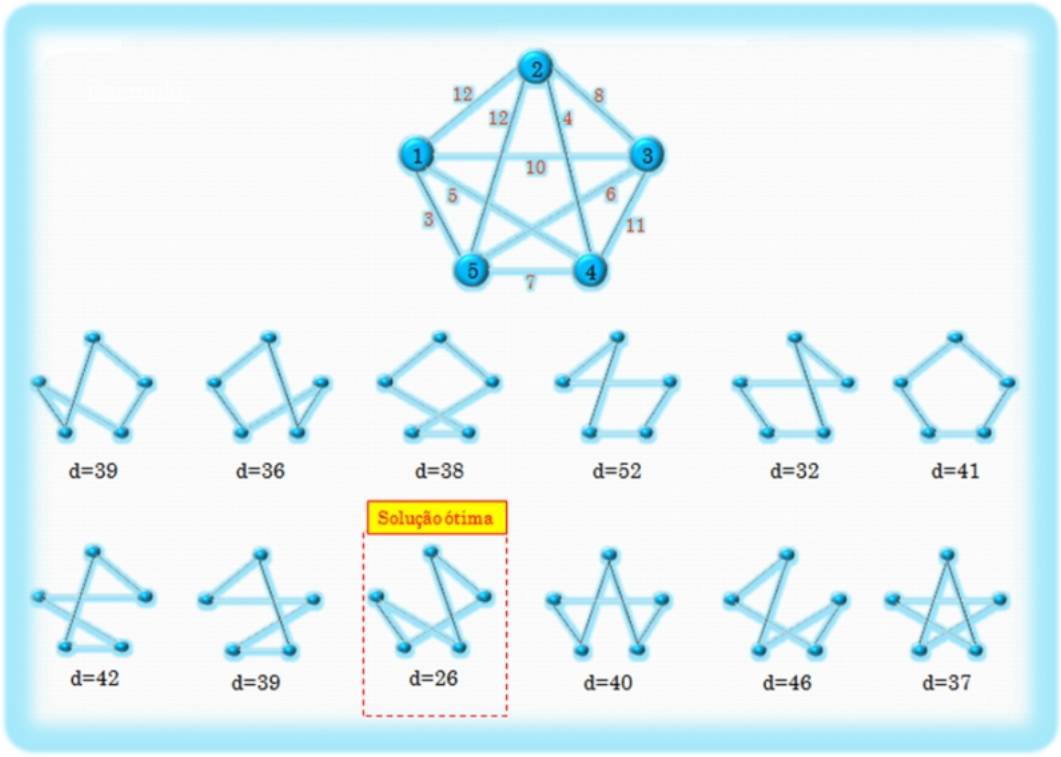

Figura 2.1: Grafo do PCV para cinco cidades.

2.3.1 PCV assimétrico

Para a formulação do PCV Assimétrico, com n cidades, tem-se:

$$x_{ij} = \begin{cases} 1, & \text{se } i \to j \text{ existe;} \\ 0, & \text{se } i \to j \text{ não existe.} \end{cases}$$

Assim, e dado o fato de que a cada nó do grafo apenas pode corresponder e sair uma seta, obtém-se uma atribuição clássica do problema. Contudo, estas restrições não são suficientes, uma vez que esta formulação permite a ocorrência de circuitos paralelos, ou seja, mesmo respeitando as condições impostas, considera-se a formação de aglomerados de cidades sem ligação. Por esta razão, a formulação do problema tem que possuir condições necessárias para remoção de circuitos paralelos. Assim, o problema é reformulado para a seguinte maneira:

Função objetivo: $\min \sum_{j=1}^{m} \sum_{i=1}^{m} c_{ij} x_{ij}$

Sujeito a:

$$s.t. \sum_{j=1}^{m} c_{ij} = 1 \quad \text{para } i = 1, ..., m$$

$$\sum_{i=1}^{m} x_{ij} = 1 \quad \text{para } i = 1, ..., m$$

$$\sum_{i \in K} \sum_{j \in K} x_{ij} \leq |K| - 1, \ \forall K \subset \{1, ..., m\}$$

$$x_{ij} = 0 \text{ ou } 1, \ i, j = 1, ..., n.$$

Para tal:

- K é um conjunto de cidades $1, ..., m$, não vazio;
- O custo c_{ij} pode ser diferente do custo c_{ji};
- Existem $m(m-1)$ variáveis.

2.3.2 PCV simétrico

Na formulação do PCV simétrico é atentado o fato de que as direções impostas, entre a origem e o destino, deixam de ter importância. Por este motivo, os custos c_{ij} e c_{ji} são iguais e os arcos entre cada dois nós passam a não ter direção, ao contrário da PCV assimétrico. Desta forma, $x_j \in \{0, 1\}$ é a variável de decisão, onde j percorre todas as arestas não direcionadas E, e c_j é o custo de percorrer cada aresta. Para encontrar uma rota neste gráfico, é preciso escolher um subconjunto de arestas tal que cada nó ligue exatamente duas das arestas selecionadas. Agora, a formulação pode ser dada como um problema 2-*matching* no grafo G^V contendo $m(m-1)/2$ variáveis, isto é, metade do número de variáveis do PCV assimétrico. Como este, O PCV Simétrico tem que possuir, na sua formulação, restrições para a eliminação de sub-circuitos. Então, o problema é reformulado da seguinte forma:

Função objetivo: $\min \frac{1}{2} \sum_{j=1}^{m} \sum_{k \in J_{(j)}} c_k x_k$

Sujeito a:

$$s.t. \sum_{k \in J_{(j)}} x_k = 2, \ \forall j = 1, ..., m$$

$$\sum_{j \in E(K)} x_j \leq |K| - 1, \ \forall K \subset \{1, ..., m\}$$

$$x_j = 0 \text{ ou } 1, \ \forall j \in E.$$

Neste caso, $J_{(j)}$ é o conjunto dos subconjuntos ligados ao nó j, e $E(K)$ é o subconjunto de todos as arestas que ligam as cidades pertencentes a um subconjunto, não vazio, K (Hoffman (2000)).

O problema simétrico é um caso especial do assimétrico, mas a experiência demonstra que os algoritmos para o assimétrico, desenvolvem-se não tão bem na abordagem simétrica. Como tal, é necessário desenvolver algoritmos dedicados para este último.

2.4 Complexidade computacional

A Complexidade Computacional é um ramo da Matemática Computacional que estuda a eficiência dos algoritmos.

Para medir a eficiência de um algoritmo, frequentemente usamos o tempo teórico que o programa leva para encontrar uma resposta em função dos dados de entrada. O cálculo é feito associando-se uma unidade de tempo para cada instrução que o algoritmo executa. Caso a dependência do tempo em relação aos dados de entrada for polinomial, o algoritmo é dito polinomial e se considera como bom. Caso contrário, é dito exponencial.

Nestas duas classes, dentre outras existentes, em que os algoritmos estão classificados, se pode fazer um comparativo da seguinte forma:

Dadas duas funções

$$\begin{aligned} f: \ \mathbb{R} &\longrightarrow \mathbb{R} \\ x &\longmapsto p(x) \end{aligned}$$

onde $p(x)$ é um polinômio, e

$$\begin{aligned} f: \ \mathbb{R} &\longrightarrow \mathbb{R} \\ x &\longmapsto a^x \end{aligned}$$

onde $a > 1$, obtemos a seguinte fórmula:

$$\lim_{x \to \infty} \frac{a^x}{|p(x)|} = \infty.$$

Uma vez que o crescimento exponencial é muito maior que o crescimento polinomial.

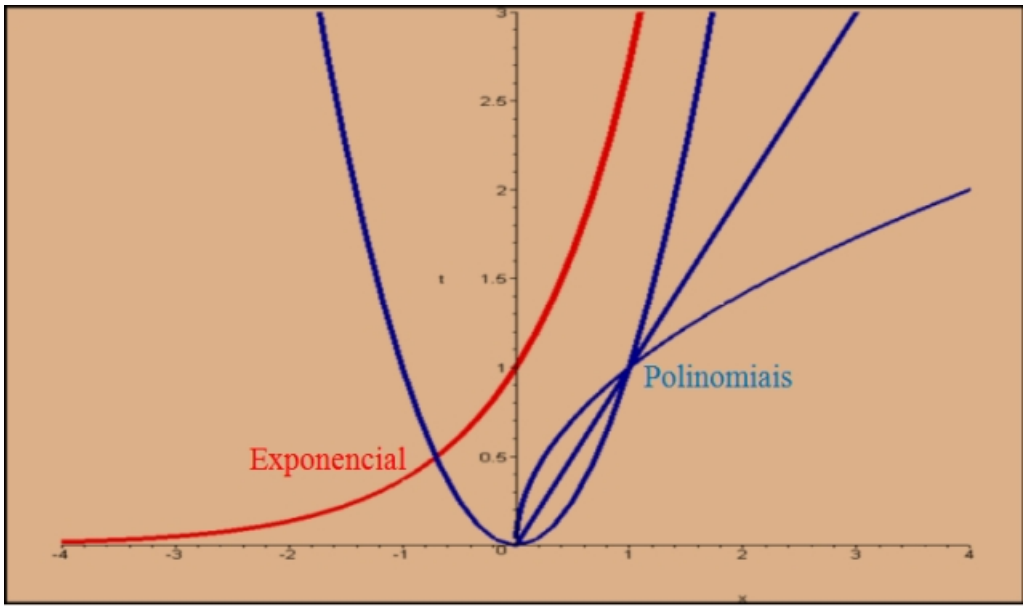

Figura 2.2: Gráfico dependência do tempo em relação ao tamanho das instâncias x.

Segundo Garey e Johnson (1979) na Teoria da **NP - Completude**, os problemas de otimização combinatória podem ser classificados, segundo sua complexidade, em P, NP, NP-difícil e NP-completo.

Definição 2.1 *A classe de algoritmos **P** (polinomial time) é formada pelos procedimentos para os quais existe um polinômio $p(x)$ que limita o número de passos do processamento se este for iniciado com uma entrada de tamanho x. Problemas pertencentes a essa classe são denominados viáveis ou tratáveis em tempo computacional, podendo ser resolvidos por algoritmos determinísticos.*

Definição 2.2 *A classe **NP** (Non-deterministic polinomial time) é constituída de problemas que podem ser resolvidos por algoritmos não-determinísticos*[2] *e cuja solução pode ser verificada, se correta, em tempo polinomial. Os problemas que fazem parte dessa classe são considerados inviáveis ou intratáveis computacionalmente, pois a dependência do tempo em relação ao tamanho da instância (dados de entrada) cresce exponencialmente.*

[2]São algoritmos que fazem uma escolha de forma aleatória, rumo a próxima configuração, dentre um número fixo de possibilidades.

Conclui-se que P $\subseteq$ NP, pois algoritmos determinísticos são um caso especial dos não determinísticos. Uma questão em aberto até hoje é saber se P=NP.

Definição 2.3 *Um problema Q diz-se pertencer a classe NP-difícil, se todo R $\in$ NP é redutível em tempo polinomial para Q.*

Definição 2.4 *Um problema B diz-se pertencer a classe* ***NP-completo*** *se ele pertencer a interseção das classes NP e NP-difícil.*

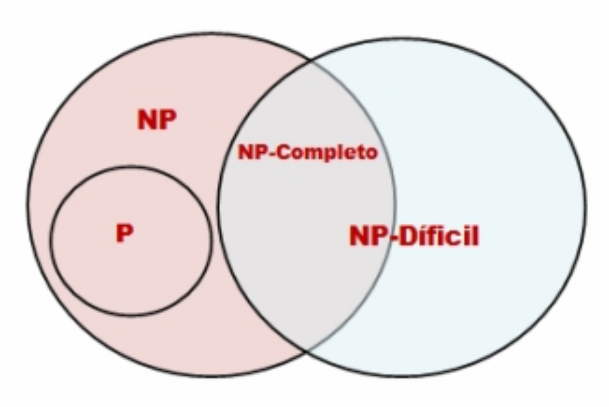

Figura 2.3: Uma configuração das classes.

O PCV pertence à classe dos problemas NP-difíceis (Garey e Johnson (1979)), ou seja, não pode ser resolvido por um algoritmo polinomial, a menos que a classe P = NP, tornando-o intratável para obtenção da solução por métodos exatos para problemas de grande porte. Com isso, abordagens exatas e aproximativas (heurísticas/metaheurísticas) têm sido buscadas para a resolução de instâncias de grande porte desse problema. Como referências para avanços na abordagem das metaheurísticas, destacamos as referências: Reeves (1993); Johnson e McGeoch (1997); Glover (1999); Hansen e Ribeiro (2001); Glover e Kochenberger (2002) e Ibaraki et al (2005).

2.5 Aplicações práticas

Nesta seção serão vistas algumas aplicações práticas para o PCV, com o intuito de justificar sua importância.

2.5.1 Fabricação de placas de circuitos eletrônicos

Apesar da fabricação de circuitos eletrônicos impressos envolver várias etapas tais como: perfuração da placa, inserção de componentes, soldagem de *chips*, teste do circuito, entre outras. Iremos falar a seguir apenas sobre as duas primeiras, como tarefas aplicadas ao PCV.

O processo de perfuração de placas: nessa etapa da fabricação de placas de circuito impresso, são realizadas centenas de furos para a inserção e soldagem dos componentes eletrônicos. A máquina de perfuração realiza todos os furos de mesmo diâmetro, retornando logo em seguida para o ponto de partida. Como não existe uma ordem de precedência na realização desses furos, o seria encontrar uma sequência que minimizasse o tempo de deslocamento da cabeça da máquina. Deste modo, este problema pode ser formulado como um PCVS. Onde a posição inicial e o conjunto de todos os furos que podem ser feitos seriam encarados como “cidades”. E o tempo gasto para mover a cabeça da máquina de uma ”cidade” para a outra, como sendo a “distância”. Cabe salientar que em uma placa são realizadas também, furos de diâmetros diferentes. Com isso, após a realização de uma seqüência de furos de mesmo diâmetro, a cabeça do equipamento retorna ao ponto de partida para logo em seguida retomar ao mesmo procedimento agora como um diâmetro diferente do anterior. Dessa forma, tal problema pode ser encarado como uma seqüência de PCVs simétricos, um para cada diâmetro.

O processo de inserção de componentes eletrônicos em placas de circuitos impressos: vimos que para fazer furos em uma placa, era utilizada uma máquina de especifica para tal atividade. Já para inserir os componentes eletrônicos não é diferente, ou seja, também se faz necessária o uso de um equipamento específico para tal tarefa, máquina de inserção. Cada máquina deve efetuar n tarefas, sendo necessários ajustes entre o fim de uma operação e o inicio de outra. Como tais tarefas consomem tempo e podem ser feitas sem ordem de precedência, então o problema se constituirá em encontrar uma seqüência apropriada de inserção dos componentes, de forma a minimizar o tempo de processamento. Logo tal problema pode ser modelado com um PCV.

2.5.2 Sequenciamento de tarefas

Suponha que n tarefas devam ser processadas seqüencialmente em uma determinada máquina. O tempo de preparo da máquina para processar a tarefa i e imediatamente a tarefa j é definido por c_{ij}. O problema de encontrar uma seqüência de execução para as tarefas, de modo que o tempo total de processamento seja minimizado, pode ser modelado como um TSP. (Laporte (1992)).

2.5.3 Roteamento de veículos, resolvido como um problema de *múltiplos*-caixeiros

Hoje devido a grande concorrência uma das grandes preocupações das empresas e indústrias é oferecer, a um custo mínimo, os melhores serviços. É sabido que no Brasil, o sistema rodoviário é usado como principal via para a coleta e/ou distribuição de mercadorias e serviços, estima-se que cerca de 10 a 15% do valor final destes produtos ou serviços está diretamente relacionado com suas despesas de transporte, dentre elas podemos citar fatores como a alta dos combustíveis, a cobrança de pedágios, a manutenção dos veículos, a conservação das vias entre outros.

A necessidade da diminuição destes gastos tem elevado o interesse no estudo do Problema de Roteamento de Veículos-PRVs e seus derivados, isso porque seu principal foco de estudo é a diminuição dos gastos com todo o processo de distribuição de mercadorias, visando a diminuição deste percentual a níveis mais satisfatórios e mais atrativos no que se trata em termos de mercado.

Para elucidar o PRVs, imagine a situação em que um fornecedor tenha que atender todas as encomendas de um determinado produto a que foi solicitado por diversos clientes, e que ele disponha de uma frota, digamos de m caminhões. Neste caso nos deparamos com o problema adicional de designar clientes a caminhões e encontrar um escalonamento de entrega para cada caminhão de modo que a sua capacidade de transporte não seja superado, bem como minimizar o custo total da viagem.

Não havendo restrição de tempo ou se o número de caminhões é um número fixo

m, como é do nosso exemplo, tal problema pode ser resolvido como um PCV. Neste caso, ao invés de apenas um caixeiro, são considerados m caixeiros, os quais se situam em alguma cidade $n+1$ e têm que visitar as cidades $1, 2, ..., n$ restantes. A tarefa consiste em selecionar estes caixeiros e lhes designar rotas tais que, no conjunto destas rotas, cada cidade seja visitada uma única vez. A seleção do caixeiro j inclui um custo fixo S_j. O custo da rota do caixeiro j é determinado pela soma das distâncias entre as cidades, iniciando e retornando para a cidade $n+1$. O problema dos *múltiplos*-caixeiros agora consiste em selecionar um subconjunto de caixeiros e designar uma rota para cada um deles de forma que cada cidade seja visitada por exatamente um caixeiro e que o custo total para visitar todas as cidades seja o menor possível. Este problema pode ser transformado em um TSP assimétrico envolvendo apenas um caixeiro. (Buriol et al., (2000)).

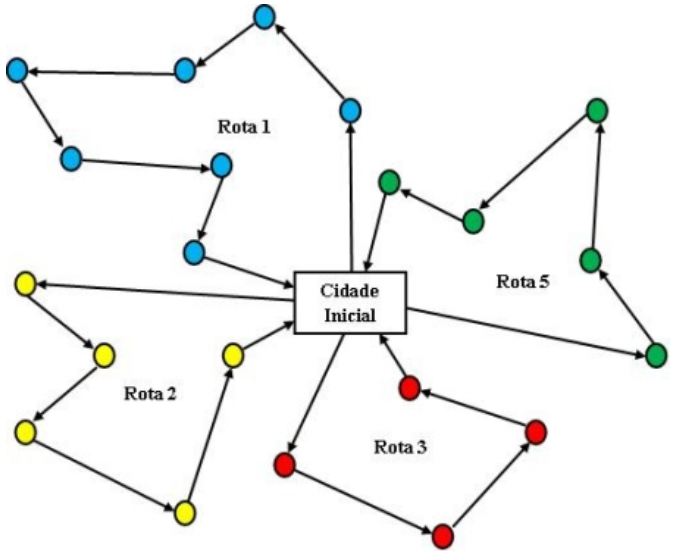

Figura 2.4: Múltiplos Caixeiros Viajantes.

2.5.4 Mapeamento físico de DNA

O fato do DNA ser composto por uma quantidade extremamente grande das bases (Adenina-A, Citosina-C, Guanina-G e Timina-T) implica que mesmo os computadores mais potentes existentes não conseguem sequenciar uma molécula inteira. E aliado ao fato de que tais computadores conseguirem sequenciar no máximo mil bases, que são procuradas estratégias para soluciona tal limitação, sem mencionar a de tempo viável para executar tal procedimento.

Figura 2.5: Representação das bases do DNA.

Para tentar solucionar o problema de mapear o DNA, respeitando estas limitações, foi adotada a seguinte estratégia:

- O DNA deve ser quebrado em pequenos pedaços denominados clones, e estes por sua vez, em pedaço de mil bases;
- Processados os resultados dos clones, a tarefa consiste em reconectá-los na ordem de inicio para que, juntos dêem consenso da molécula de DNA. Porém o fato de não se saber de que região da molécula cada clone procede, que torna a tarefa de mapeamento de DNA difícil.

E é com essa dificuldade de mapeamento de cada clone em sua respectiva região de origem no DNA (conhecido como o problema de mapeamento físico de DNA), que tal problema pode ser formulado como um PCV. (Cerqueira e Stelzer (2005)).

Figura 2.6: Representação do problema do mapeamento físico de DNA como um PCV.

2.5.5 Controle de robôs

Na indústria para ser fabricada uma simples peça, um robô deve ser capaz de realizar uma seqüência se de operações, tais como cortes, furos, soldagem entre outras. Só que para determinar tais operações em tempo reduzido não é uma tarefa tão simples, uma vez que há restrições de precedência. Para superar tal dificuldade, encaramos o problema como o de encontrar o menor caminho hamiltoniano que satisfaz relações precedência de operações. Basta que, para isso, se tome os tempos necessários para as trocas de ferramentas e das mudanças de posicionamento, como sendo as *distâncias*. Montamos assim, uma matriz de distancias entre cada cidade (tarefa) i, j, cuja meta é a de encontrar um caminho que, partindo da cidade inicial, passe exatamente uma vez por todas as outras, sem retornar ao ponto de partida. (Buriol et al., 2000).

2.6 Métodos de resolução

Uma vez classificado o problema devemos escolher, dentre os métodos existentes, o que melhor se adequa para resolver o mesmo. Antes de escolhermos tais métodos iremos fazer uma pequena revisão sobre os mais conhecidos e os que utilizaremos para este trabalho.

Começaremos falando sobre os **métodos exatos**, que são aqueles que garantem

encontrar a solução ótima para o problema.

A classificação mais geral para métodos de resolução de problemas de otimização diferencia métodos exatos de heurísticos. As técnicas mais utilizadas para resolver o PCV são *branch-and-bound* e *branch-and-cut*. Apesar de esses métodos nos retornarem a solução exata, muitas vezes tal processo pode demorar milhões ou até bilhões de anos, dependendo do tamanho da instância. Mesmo com uma máquina capaz de fazer um bilhão de adições por segundo.

Como foi dito no início do Capitulo, o PCV pertence a classe de problemas NP-difícil, portanto a obtenção da solução ótima por métodos exatos para instâncias muito grandes torna-o inviável em tempo computacional. Sendo assim, uma outra estratégia seria a escolha dos **métodos aproximativos**. Apesar desses métodos não garantirem a solução ótima para o problema, nos retornam uma solução "boa" com uma margem de erro aceitável e em tempo viável.

Seguindo o raciocínio de que a solução ótima nem sempre é a mais viável, introduziremos o conceito de **métodos heurísticos** ou simplesmente de **heurística**. Esses são procedimentos capazes de produzir uma solução aceitável para um problema em muitos casos práticos, mas para as quais não há nenhuma prova formal de estar correta. Tal solução pode até ser a ótima, mas não pode ser comprovada. As heurísticas são normalmente utilizadas quando não há método conhecido para encontrar uma solução ótima, seja por restrições (de espaço de tempo etc.).

As heurísticas podem ser enquadradas em algumas dos seguintes grupos:

- heurísticas construtivas - São aquelas onde uma ou mais soluções são construídas elemento a elemento (variáveis, arcos, vértices, etc.), seguindo algum critério heurístico de otimização, até que se obtenha uma solução viável; um exemplo de heurística construtiva é a do método guloso;
- heurísticas de busca em vizinhança, como a busca local - São aquelas que partem necessariamente de uma solução inicial viável (em alguns casos podendo ser somente

uma solução possível qualquer), com o intuito de melhorar esta solução através de operações de adição, troca ou remoção, até que não seja mais possível sua melhoria (atingindo o ótimo local) ou que algum outro critério de parada seja satisfeito;

- heurísticas híbridas - São aquelas resultantes da combinação de duas ou mais heurísticas com estratégias diferentes.

Existe ainda um tipo especial de heurísticas, muito popularizado e utilizada atualmente por usar estratégias que permitem sair de ótimos locais. São denominadas de Metaheurísticas.

Metaheurísticas são procedimentos heurísticos que guiam um operador de busca local na pesquisa de soluções viáveis, com o intuito de evitar a parada prematura em ótimos locais. O processo de encontrar uma boa solução consiste em aplicar a cada passo uma heurística subordinada que tenha sido projetada para cada problema particular. Como uma tentativa para escapar de ótimos locais, algumas delas permitem controlar a deterioração das soluções para diversificar a procura (Noronha, Aloise e Silva (2001)).

Antes de caracterizar as duas classes em que as metaheurísticas estão divididas, é necessário saber o que é vizinhança. Dados S o espaço de busca (conjunto de soluções viáveis) do problema e s uma solução do problema, tem-se:

Uma estrutura de vizinhança é uma função $N : S \rightarrow 2^s$ que determina, para todo $s \in S$, um conjunto de vizinhos $N(s) \subseteq S$. O conjunto $N(s)$ é chamado de vizinhança de s. Em outras palavras, uma solução s' diz-se fazer parte da vizinhança da solução s se, e somente se, s' resultar de uma modificação em s, de tal maneira que continue a fazer parte do conjunto de soluções possíveis. A vizinhança, então, é dependente do modo adotado para gerar soluções viáveis. Para tal, não existem regras definidas e rígidas que sirvam para todos os problemas (Coelho et al., (2006)).

A primeira das classes das metaheurística é constituída pelos métodos que exploram um único elemento da uma vizinhança a cada iteração. Esta busca do espaço de soluções gera um caminho ou trajetória de soluções, obtido pela transição de uma solução para outra

de acordo com os movimentos permitidos pela metaheurística. Dentre as técnicas deste tipo de classe mais conhecidas, podemos citar, *simulated annealing*, GRASP (*Greedy Randomized Adaptive Procedure*) e a busca Tabu.

Já a outra classe é constituída pelos métodos que fazem busca em uma população de indivíduos a cada iteração, explorando varias regiões do espaço de solução a cada vez. Com isso, ao longo das iterações, são construídas mais de um caminho ou trajetória de busca, pois novas soluções são obtidas através da combinação de soluções anteriores. Tais métodos são conhecidos como, algoritmos populacionais. Como pertencentes a esta classe podemos citar, algoritmos genéticos e algoritmos meméticos.

2.6.1 Algoritmos populacionais genéticos e meméticos

Os Algoritmos Genéticos - AGs são métodos evolutivos, ou seja, algoritmos populacionais que se baseiam na evolução natural das espécies. Tais algoritmos foram inspirados na Teoria Neo-Darviniana, a qual afirma que a vida é propagada devido à atuação de certos processos nas populações e espécies, que são: competição, seleção, reprodução(recombinação) e mutação.

Os Algoritmos Genéticos - AGs foram inicialmente estudados na década de 60, sendo formalizados por Holland na década de 70 (Holland, 1975). O objetivo inicial não foi desenvolver algoritmos para solução de problemas específicos, mas sim estudar formalmente os fenômenos de adaptação, naturais ou artificiais, com o propósito de importar estes mecanismos de adaptação para ambientes computacionais.

No contexto de AGs, uma *população de cromossomos* (população de soluções = conjunto de indivíduos) evolui através de varias *gerações* depois de ter sido submetida a um tipo de *seleção* (dirigida as melhores regiões do espaço de busca), a qual utiliza-se de operadores baseados na genética, *crossover*[3] (recombinação) e mutação, com o intuito de evoluir. A

[3]Apesar de a palavra crossover denotar, na biologia, o procedimento de união de exatamente dois pais para gerar um ou mais filhos, neste trabalho utilizá-la-emos com o sentido da palavra recombinação, que representa um caso mais geral, incluindo a possibilidade de vários pais fazerem parte desse processo. Além disso, a recombinação pode fazer uso das

seguir tem-se uma pequena ilustração da estrutura de um algoritmo genético tradicional.

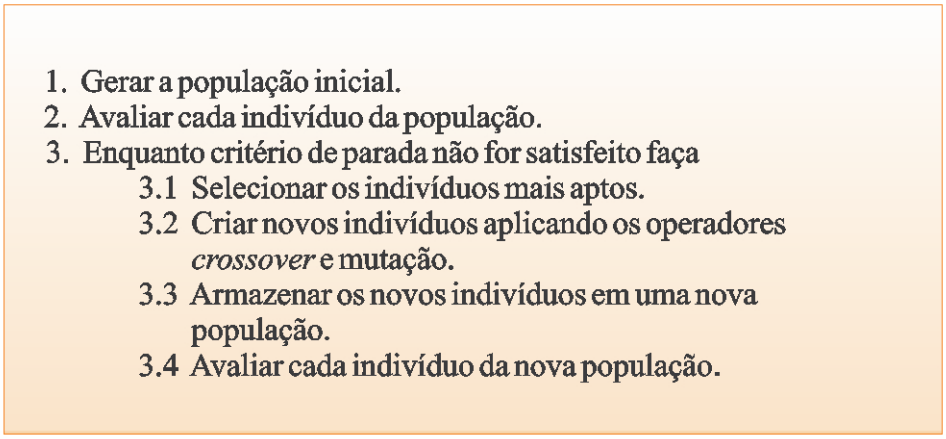

Figura 2.7: Algoritmo Genético Convencional.

Na fase inicial do AG, a população inicial é gerada ou por uma heurística construtiva, feita exclusivamente para o problema, ou de forma aleatória ou por uma combinação das duas.

O critério de parada pode ser definido, por um numero n de gerações (iterações), pela perda de diversidade, quando é encontrada a solução (quando essa é sabida) ou por convergência.

A fase avaliação funciona como uma função, função de Aptidão (*fitness*), dando uma nota para cada individuo (solução do problema), com o intuito identificar as soluções de melhor qualidade. A *fitness* pode ser igual à função objetivo ou baseada no *ranking* do indivíduo de melhor aptidão da população.

A fase de seleção, que imita a seleção natural, funciona da seguinte maneira: os melhores indivíduos, de maior *fitness*, são selecionados para gerar *filhos* através de *crossover* e mutação. Os tipos mais comuns de seleção são:

- Seleção proporcional a aptidão ou método da roleta, na qual os indivíduos mais aptos

informações da instância, como por exemplo para o TSP, priorizar a inserção dos arcos mais promissores. Na biologia o crossover não exerce este papel. (Buriol et. al. 2000)

têm maior probabilidade de serem escolhidos;

- Seleção por torneio - são escolhidos na população n indivíduos, comumente dois, de forma aleatória, onde o que possuir a maior *fitness* será o selecionado.

A procedimento de *crossover* controla, em alguns casos dependendo do modo de implementação, os *pais* que devem ser selecionados para formar um ou mais *filhos*. Afim de não adicionar indivíduos já existentes na população e assim explorar novas regiões do espaço de busca. Já o procedimento de mutação tem o papel exclusivo de trazer diversidade para a população.

Cabe aqui enfatizar a importância do *crossover* e da mutação, garantindo a diversidade a fim de permitir a exploração de regiões desconhecidas no espaço de busca, sendo assim as principais ferramentas de busca dos AGs.

Um Algoritmo Memético - AM, além de se basear no conceito evolução genética, utiliza também o conceito de evolução cultural, onde a informação cultural não é transmitida através de um processo de recombinação, mas pela comunicação entre agentes[4] da população. Isso quer dizer que a evolução dos indivíduos não se dá exclusivamente pela reprodução dos mais aptos, mas também pela troca de experiências pessoais e de informações adquiridas ao longo de sua existência, repassando-as para as próximas gerações.

O termo *meme* (unidade replicante de informação) foi idealizado por Richard Dawkins na década de 70. E por não estar vinculado ao mecanismo hereditário, pode acarretar na transmissão de informação de um agente para vários outros ou toda a população sem que seja transcorrida uma geração. A principal diferença entre genes e memes está no processo de transmissão aos seus descendentes. Os genes, no processo de evolução, são transmitidos de forma que os descendentes gerados herdem características e habilidades presentes em seus progenitores. Já quando o meme é transmitido, ele será adaptado pela entidade que o recebe, com base no seu conhecimento e para melhor atender às suas necessidades. Com isso, a informação memética é transmitida de modo mais rápido e flexível que a genética.

[4]No contexto de AM os agentes - são indivíduos que realizam explorações independentes no espaço de busca.

O mecanismo que permite a transmissão da informação memética é a introdução de um (ou mais) operador (es) de busca local. A idéia principal de um AM é explorar a vizinhança das soluções obtidas por um AG e caminhar em busca do ótimo local (para cada solução) antes de retornar para o AG e continuar o processo.

A primeira vez que o termo Algoritmo Mémetico foi usado na literatura, consta do ano de 1989, no trabalho intitulado, "*On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms*". Tal trabalho descreve o AG feito por Moscato, o qual utilizava a técnica *Simulated Annealing* como procedimento de busca local (como torneio competitivo e cooperativo e entres agentes), integrado com o operador de *crossover* (*Caltech Concurrent Computation Program*, *Report*. 790, 1989). Onde o PCV foi utilizado como método de teste de resultados.

Na Figura 2.8 é apresentada uma ilustração da estrutura geral de um AM. Cabe salientar que tal estrutura é genérica para os AMs baseadas em busca local, caracterizando muitas implementações diferentes que podem ser encontradas na literatura.

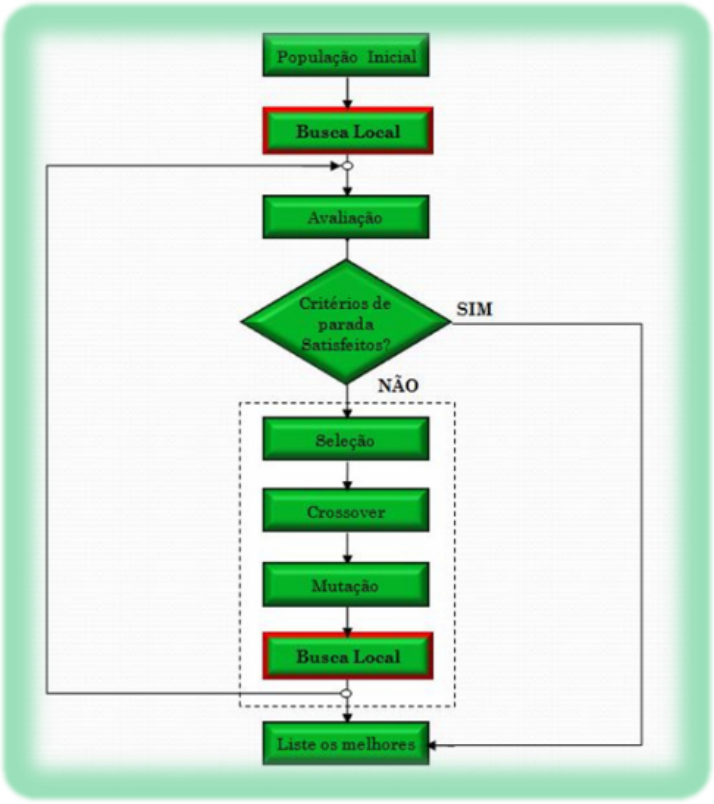

Figura 2.8: Estrutura básica de um algoritmo memético.

2.6.2 Reconexão por Caminhos (*Path-Relinking*)

Path-Relinking-PR é uma técnica de otimização que visa intensificar a busca sobre regiões próximas às soluções de alta qualidade já obtidas (Glover (1998), Glover (1999), Glover, Laguna e Marti (2000)). Um método que se enquadre no *framework*[5] de *Path-Relinking* realiza movimentos - dentro do espaço de soluções viáveis - partindo de uma solução-origem até atingir uma solução-destino. Tal método adicionaria - a cada passo - elementos da solução-destino à solução-origem e guardaria cada solução intermediária encontrada. A idéia é verificar se essas soluções intermediárias são melhores do que as já obtidas. A seguir, tem-se uma esquematização de *Path-Relinking*.

Na figura 2.9, é dado o pseudocódigo do que seria a idéia essencial de *Path-Relinking*.

[5]Um framework é um conjunto de classes colaborativas (abstratas e concretas) que permite reutilização de projeto e provê uma infra-estrutura genérica de soluções para um conjunto de problemas específicos; é uma abstração de um conjunto de classes inter-relacionadas (Buriol et (2000)).

```
procedimento Path_Relinking
entrada: solução-origem, solução-destino;
   início
         solução-atual = solução-origem;
         enquanto (solução-atual := solução-origem) faça
         solução-atual := solução-atual + mov(solução-destino);
         fim-enquanto

         retorna melhor solução-atual encontrada;
fim
```

Figura 2.9: *Framework* de *Path-Relinking*.

Porém, vários tipos de implementações podem se enquadrar na definição desse *framework*. Vejamos agora outra maneira de visualizar a técnica do PR seguindo a seguinte seqüência. Considere (S_1, S_2), um par de soluções, sendo uma delas guia e a outra de partida, chamada de solução corrente. O procedimento termina quando se chega à solução guia, isto é, quando a solução corrente tem todos os atributos da solução guia. Em cada iteração do procedimento é colocado um atributo da solução guia. A solução corrente sofre, então, uma busca local onde se fixam os atributos da solução guia que já foram incorporados à solução corrente. O procedimento termina quando se chega à solução guia, isto é, quando a solução corrente tem todos os atributos da solução guia. Observe a figura 2.10 na sequência.

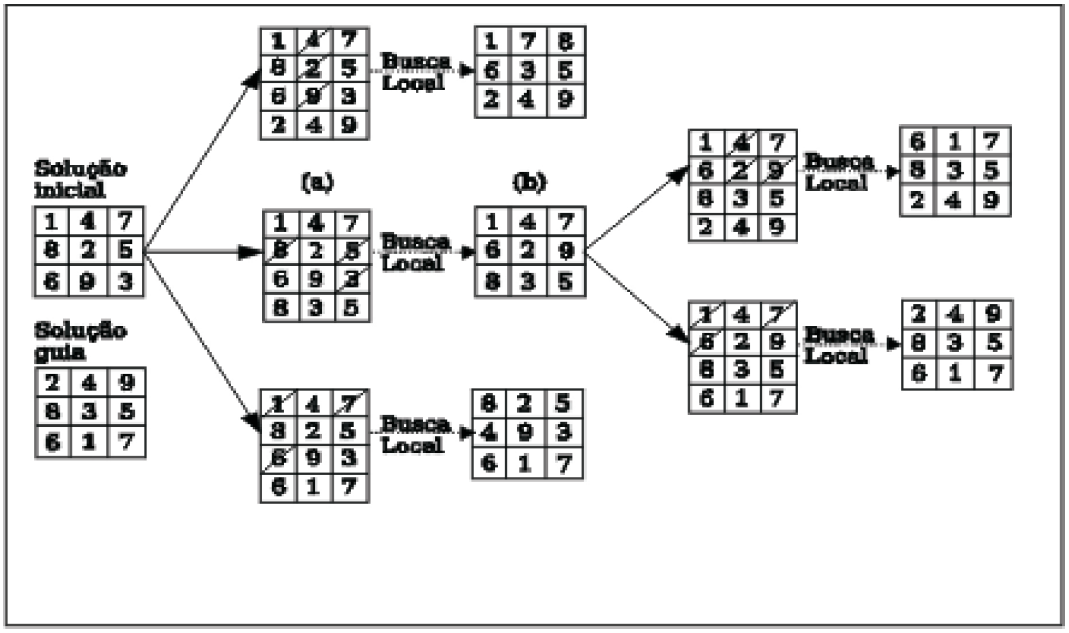

Figura 2.10: Seqüência de Intreações do *Path-Relinking*.

Originalmente Glover, explorava trajetórias que conectavam soluções elite (soluções de boa qualidade) obtidas durante a busca realizada pelo método Busca Tabu. Duas estratégias na técnica PR podem ser aplicadas: pós-otimização entre todos os pares de soluções elite e a intensificação a cada ótimo local obtido após a fase de busca local (**considerada mais eficiente**). Na estratégia de intensificação durante a busca local, aplica-se a reconexão aos pares (s_1, s_2), onde s_1 é um ótimo local e s_2 uma solução elite. Esta é escolhida aleatoriamente no conjunto elite. Para terminar podemos citar a PR *forward* que tem como ponto de partida um ótimo local S_1 e de chegada uma solução elite S_2 e a PR *backward* (mais eficiente) a qual caminha de uma solução elite S_2 a um ótimo local S_1.

2.6.3 Vocabulary Building

A técnica de otimização pouco conhecida denominada Construção de Vocábulário (*Vocabulary Building*) foi idealizada por Fred Glover, dentro do contexto de *Path-Relinking* (Glover (1992)). Em Glover e Laguna (1993), essa técnica foi proposta como uma estratégia a ser implementada dentro da Busca Tabu. Em 1995, surgiram dois trabalhos cujos conteúdos possuem ligação com as idéias relativas ao *Vocabulary Building* (Rochat e Taillard (1995); Kelly e Xu (1995)) aplicadas a problemas de roteamento de veículos. Em 1997, a técnica é mais uma vez mencionada em um trabalho que trata de *Scatter Search* e *Path Relinking*

(Glover (1997)). A partir de 1997, surgiram outras publicações que abrangem o *Vocabulary Building*, sendo quase todas elas do próprio Fred Glover, como: Glover e Laguna (1997); Scholl, Klein e Domschke (1998); Glover (1999); Glover, Laguna e Marti (2000); Glover (2003). Em 2006, a técnica foi aplicada em um Algoritmo Memético para o PVCA em (Guedes; Aloise (2006)) e, recentemente, em um algoritmo Busca Tabu para o Problema de Atribuição de Localidades em Anéis DH/SONET em (Soares (2008)) e em um Algoritmo Genético e Memético para o mesmo problema (Silva (2008)).

Dentre os trabalhos citados acima, onde são poucos os que tratam de uma aplicação prática real com implementação computacional da técnica *Vocabulary Building*, podemos destacar os de Scholl, Klein e Domschke (1998), o qual aborda o Problema de Sequenciamento de Linhas de Montagem Multi-Modelo e o de Guedes e Aloise (2006) que é feito especificamente para o PCVA. O *Vocabulary Building* é, portanto, uma técnica de otimização ainda pouco explorada. Isso leva a crer que qualquer utilização dessa técnica produzirá resultados originais, o que motivou a pesquisa realizada nesse trabalho.

A idéia central da Construção de Vocábulos consiste, primeiramente, em identificar boas soluções parciais (trechos de soluções completas) prosseguindo, posteriormente, com a busca pelo ótimo global. Uma boa solução parcial pode ser, por exemplo, uma configuração que esteja presente em várias soluções-elite. O objetivo é tomar vantagem destes contextos em que certas configuração parciais de soluções ocorrem com freqüência como componentes de boas soluções completas. Esta estratégia de busca por "configurações parciais" pode ajudar a evitar a explosão combinatória resultante da manipulação de apenas elementos primitivos.

Denominamos por "Vocábulo" um elemento identificado como sendo importante para a obtenção de soluções de boa qualidade (*fitness*) - por exemplo, uma determinada aresta da rota do Caixeiro Viajante. A partir desses vocábulos, é possível chegar a combinações mais complexas e úteis. O método também requer uma estrutura para armazenar os vocábulos identificados como úteis. Esse conjunto de vocábulos recebe o nome de "*Pool* ou Coleção de vocábulos" que funciona com uma espécie de memória adaptativa, ou seja, sofre modificações no decorrer da busca (Guedes e Aloise (2006)). Pois a cada momento,

soluções parciais distintas podem ser de diferente atratividade. Por esta razão, é interessante que o *Pool* mantenha somente as que forem mais promissoras dentro do processo de busca.

A idéia de *Vocabulary Building* é bem geral e pode ser utilizada para diversos problemas e de diversos modos. Neste trabalho, idealizou-se duas formas de implementações das idéias do *Vocabulary Building* para o PCVA. Na primeira delas, é utilizado um *pool* de vocábulos para manter trechos de rotas que são avaliados de acordo com a sua boa ou má influência durante a busca na qual denominaremos de método 1. Na segunda forma, verificam-se as arestas presentes em todas as rotas de um determinado conjunto de soluções e faz-se a condensação de trechos contíguos. A essa implementação das técnicas de *Vocabulary Building* que se baseia na contração de vértices denominaremos de método 2.

No capitulo 4, serão vistos os detalhes de implementação dos procedimentos dessa técnica utilizados nesse trabalho.

2.7 Revisão de literatura

Desde que foi provado que o PCV pertence à classe NP-difícil (Garey e Johnson, 1979), pesquisadores de todo mundo vem propondo algoritmos heurísticos e metaheurísticos, dentre outros tipos de algoritmos já conhecidos, como estratégias de resolução para as grandes instâncias do problema. Uma visão mais ampla de métodos de resolução para o PCV, pode ser encontrada em Balas e Toth (1985), Laporte (1992), Jünger et al.(1995).

Focando diretamente nos métodos exatos para a instância assimétrica do PCV, temos:

O algoritmo *branch-and-bound* proposto por Miller e Pekny (1991) o qual utiliza uma relaxação do PCVA, resultante no problema de designação (*Assignment Problem* - AP). São relatadas soluções ótimas encontradas para instâncias de até 500 mil cidades em tempos computacionais razoáveis, embora estas instâncias tenham sido aleatoriamente geradas com distribuição uniforme num dado intervalo. Instâncias geradas desta forma parecem ser fáceis para o algoritmo de Fischetti e Toth (1997), também baseado na relaxação-AP.

Similarmente, Carpaneto et al. (1995) resolvem problemas com até 2 mil cidades em menos de 1 minuto. Entretanto, há instâncias no qual a relaxação-AP pode encontrar dificuldades, tais como as com distâncias quase simétricas (isto é, $d_{ij} \approx d_{ji}$ para todo i, j).

Dentro dos métodos heurísticos, uma quantidade muito grande de técnicas meta-heurísticas tem sido proposta. Os melhores resultados têm sido obtidos pela combinação de algoritmos evolutivos com procedimentos de busca local, originando um algoritmo memético (*Memetic Algorithm* - MA) (Moscato, 1989, 1999; Moscato e Norman, 1992).

Diversos MAs têm sido propostos para resolver o PCVA. Baseado na análise de resultados computacionais que utilizam esta técnica, pode-se dizer que um bom MA deve conter as seguintes características:

i) Operadores de recombinação e mutação apropriados e robustos, inerentes a qualquer AG;

ii) Um eficiente e rápido operador de busca local, de fundamental importância para os MAs, pois 85% - 95% do tempo de CPU gasto, geralmente, é atribuído a tal procedimento;

iii) Uma população estruturada hierarquicamente. Muitos experimentos na literatura têm comprovado que a adoção de estruturas nas quais os agentes se relacionam conforme uma estrutura hierárquica, provaram ser mais eficazes quando comparados às implementações não estruturadas (Moscato, 1993; Gorges-Schleuter, 1997; França et al., 1999; Berretta e Moscato, 1999).;

iv) Estruturas de dados e mecanismos de codificação apropriados.

No AM proposto por Freisleben e Merz (1996), denomidado GLS (*Genetic Local Search*) foi ulitizado um novo operador de recombinação, chamado de DPX (*Distance Preserving Crossover*). Além dos operadores de busca local, *fast-3-Opt* testando instâncias do caso assimétrico e uma variação da heurística de *Lin-Kernighan* testando instâncias do caso simétrico. No ano seguinte estes resultados foram melhorados, adotando uma série de mecanismos de implementação mais sofisticados que melhoraram a desempenho do método (Merz e Freisleben, 1997). No caso do ATSP, foi adicionada uma variante do procedimento *4-Opt* para a busca local. Indo por outro caminho, Gorges-Schleuter (1997) tem optado por investir em populações estruturadas espacialmente, usando uma busca local simples, no lugar de

procedimentos de busca local complexos. No método, denominado *Asparagos96*, a população é organizada em *demes* (populações locais) espacialmente dispostos em um anel, recebendo o nome de *ladder-population* (Gorges-Schleuter, 1989), onde os operadores de busca local são o 2-Opt para o PCVS e o *3-Opt*, para o PCVA. Já o operador de recombinação utilizado é o MPX2, o qual difere em alguns aspectos do seu MPX (*Maximal Preservative Crossover*) (Gorges-Schleuter, 1989). Testes computacionais reduzidos são apresentados, limitando-se a poucas instâncias da TSPLIB. Por meio da análise destes resultados, conclui-se que para instâncias grandes do caso simétrico, *Asparagos96* é superior enquanto o GLS obtém melhores resultados para instâncias até 783 cidades. Para o caso assimétrico, *Asparagos96* obteve melhores resultados que o GLS nas 5 instâncias relatadas.

O operador de recombinação que pode ser usado em ambos os casos, PCVS ou PCVA, foi proposto por Nagata e Kobayashi (1997). O EAX (*Edge Assembly Crossover*) pode gerar vários filhos a partir de dois pais. Informação heurística é adicionada durante o processo de recombinação de forma a não incluir uma fase de busca local. Entretanto, pode-se considerar este algoritmo um AM, dado que é agregada informação adicional durante o processo de evolução. Já Walters (1998) utiliza o conceito de *soft brood selection* no seu AM. Em *brood selection* os dois pais podem gerar muitos filhos, no entanto apenas o melhor dentre eles, em relação à sua avaliação de função objetivo, é selecionado para permanecer na população. Dependendo da implementação, mais de um filho pode ser selecionado e pode-se evitar aqueles que já tenham um indivíduo idêntico na população. O operador de recombinação, chamado DER (*Directed Edge Repair*), é combinado com o operador de busca local *3-Opt* modificado, e são apresentados resultados computacionais para um conjunto reduzido de instâncias da TSPLIB.

E mais recentemente Buriol (2004), propõe um algoritmo memético, cuja principal contribuição é um novo operador de busca local, chamado de Inserção Recursiva de Arcos (do inglês RAI - *Recursive Arc Insertion*), desenvolvido especialmente para o Problema do Caixeiro Viajante Assimétrico. Um novo operador de cruzamento (*crossover*) também é proposto - *Strategic Arc Crossover* (SAX) - sendo esse uma adaptação de um operador semelhante, chamado *Strategic Edge Crossover* (SEX) (Moscato e Norman (1992)). Tal operador de cruzamento mostrou-se capaz de lidar de forma satisfatória com instâncias

assimétricas do PCV. Além disso, as estruturas de dados utilizadas pelo algoritmo proposto permitem que soluções de alta qualidade sejam encontradas rapidamente. Até onde podemos pesquisar, este Algoritmo Memético é considerado o Estado-da-Arte para o PCVA.

Capítulo 3

Algoritmo Memético Aplicado ao PCVA (estado-da-arte)

Neste capítulo, serão apresentados os detalhes de implementação do Algoritmo Memético proposto em Buriol et al. (2004) para a resolução do Problema do Caixeiro Viajante Assimétrico. Trata-se de uma bem-sucedida metaheurística evolutiva e acredita-se, com já citado, que seja o melhor procedimento Memético conhecido para o problema em apreço. Esse algoritmo foi implementado e utilizado como base para a aplicação dos procedimentos de *Path-Relinking* e *Vocabulary Building* aqui propostos.

O Algoritmo Memético em questão trata-se de um procedimento evolutivo especializado para as instâncias assimétricas do PCV. O método propõe uma nova estratégia de busca local e incorpora sofisticadas características que contribuem para a eficácia do algoritmo. Essas características são:

- Estruturação da população como uma árvore ternária completa de 13 (treze) nós;
- Organização hierárquica da população em grupos sobrepostos, permitindo um esquema especial de seleção;
- Manutenção da diversidade da população; e
- Estruturas de dados eficientes.

Em Buriol, França e Moscato (2004), um novo algoritmo memético (Moscato (1989 1999); Moscato e Norman (1992)) foi proposto para a resolução da variante assimétrica do

PCV. Como já foi mencionado, esse algoritmo incorpora sofisticados elementos que visam à melhoria do processo de busca. A seguir, tem-se a estrutura do algoritmo proposto.

```
procedimento AM_Buriol
início
        inicializa_pop();
        repita
                estrutura_pop();
                recombina_pop();
                muta_pop();
                otimiza_pop();
        até critério_de_parada = satisfeito;
fim
```

Figura 3.1: Algoritmo Memético proposto em Buriol et al (2004).

Em cada geração do procedimento proposto, a população é reestruturada pelos procedimentos *atualiza_pocket()* e *propaga_pocket()*, representados na figura 9 por *estrutura_pop()*. A população é organizada hierarquicamente como uma árvore ternária completa de 13 agentes agrupados em 4 sub-populações - onde cada sub-população é formada por 4 indivíduos. Treze novos indivíduos são gerados a cada iteração pelo procedimento *recombina_pop()*, onde cada um deles é modificado pelo procedimento de mutação com uma probabilidade de 5%. No final de cada geração, aplica-se *otimiza_pop()* - busca local - em cada um desses novos indivíduos. De acordo com o artigo em que é proposto, a principal contribuição do algoritmo memético descrito é um novo operador de busca local, chamado de Inserção Recursiva de Arcos (do inglês RAI - *Recursive Arc Insertion*), desenvolvido especialmente para o Problema do Caixeiro Viajante Assimétrico. Um novo operador de cruzamento (*crossover*) também é proposto - *Strategic Arc Crossover* (SAX) - sendo esse uma adaptação de um operador semelhante, chamado *Strategic Edge Crossover* (SEX) (Moscato e Norman (1992)). Tal operador de cruzamento mostrou-se capaz de lidar de forma satisfatória com instâncias assimétricas do PCV. Além disso, as estruturas de dados utilizadas pelo algoritmo proposto permitem que soluções de alta qualidade sejam encontradas rapidamente.

A seguir, são mostrados detalhes de implementação do algoritmo proposto por Buriol et al (2004).

3.1 Estruturas de dados

Uma característica importante na implementação de bons algoritmo meméticos é o uso de estruturas de dados adequadas, que permitam uma execução eficaz do algoritmo. A seguir, tem-se a descrição das estruturas de dados utilizadas pelo algoritmo memético em apreço.

3.1.1 Representação de soluções

A estrutura de dados utilizada para representar uma solução viável para o PCVA consiste de um vetor bidimensional com 2 linhas e n colunas. Cada índice i do vetor é referente à cidade i, e o vetor mantém os índices associados com as cidades predecessoras (prev) e sucessoras (prox) de cada cidade i. De acordo com os criadores do algoritmo, houve duas razões para a escolha dessa representação: primeiro, devido à sua simplicidade; segundo, devido ao fato de que cada um dos numerosos movimentos *3-opt* executados - chamados recursivamente pela busca local - podem ser feitos em tempo constante.

3.1.2 O conceito de agentes

A população do algoritmo memético de Buriol et al (2004) é composta por um conjunto de elementos chamados "agentes". Cada agente engloba duas soluções viáveis, as quais são chamadas de "*pocket*" e "*current*". Em cada geração do algoritmo, o procedimento de recombinação utiliza soluções *pockets* e *currents* como genitores de 13 novos indivíduos. Esses novos indivíduos tomam o lugar das soluções *current* da população.

3.1.3 Estrutura da população

A população do método aqui descrito é composta de 13 agentes organizados em uma árvore ternária completa de três níveis. Os agentes do nível intermediário fazem parte, concomitantemente, de duas sub-populações distintas. A forma estrutural da população é como indicado na figura 3.2.

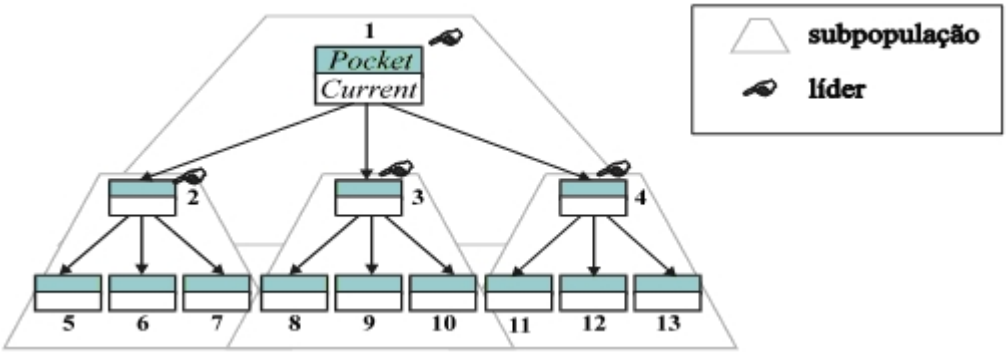

Figura 3.2: Estrutura da população.

Cada sub-população é composta de um único "líder" e três agentes "apoiadores", sendo os últimos localizados um nível abaixo do primeiro. O agente 1, localizado na raiz da árvore, é o líder da primeira sub-população e é apoiado pelos agentes 2, 3 e 4. O agente 2 é apoiado pelos agentes 5, 6 e 7 e assim por diante.

Antes de cada chamada ao procedimento de recombinação, a árvore é reestruturada utilizando-se dois operadores: *atualiza_pocket()* e *propaga_pocket()*. Quando a solução *current* é melhor do que a solução *pocket* de mesmo agente e não existe outro *pocket* com um valor de *fitness* repetido, as soluções *pocket* e *current* têm o seu papel invertido por *atualiza_pocket()*. Depois disso, o procedimento *propaga_pocket()* troca o *pocket* dos pais com o melhor *pocket* da sua subpopulação, criando um fluxo que garante que os melhores *pockets* se deslocam em direção à raiz da população.

3.2 Operadores do algoritmo memético

A seguir, os operadores do algoritmo são descritos.

3.2.1 População inicial

A população inicial foi gerada visando-se a obtenção de um conjunto de soluções diversificado. Para isso, utilizou-se o método construtivo Pach de Karp e Steele (1985) e método do vizinho mais próximo. O primeiro método foi utilizado para gerar a rota inicial da solução *pocket* do agente contido na raiz da população. As demais soluções foram geradas pelo outro método, que tende a utilizar arcos muito "caros" para as últimas cidades inseridas.

Sendo assim, utilizaram-se as três últimas cidades inseridas em um líder para inicializar os procedimentos de construção dos três agentes apoiadores. As soluções *pocket* fornecem as cidades para os *pockets* dos seus apoiadores e as soluções *currents* fazem-no para as suas respectivas soluções apoiadoras.

3.2.2 Operador de recombinação

O artigo de Buriol et al (2004) testou a utilização de quatro operadores diferentes de *crossover: Distance Preserving Crossover* (DPX) (Freisleben e Merz (1996)), *Multiple Fragment-Nearest Neighbor Repair Edge Recombination* (MFNN) (Holstein e Moscato (1999)), *Uniform Nearest Neighbor Crossover* (UNN) (adaptação de *Uniform Crossover Operator* - Goldberg (1989)) e *Strategic Arc Crossover* (SAX) (proposto no artigo de Buriol et al(2004)). Dentre eles, o operador SAX mostrou uma maior sinergia com a busca local RAI - que é um fator fundamental para a criação de procedimentos evolucionários.

O operador utilizado - SAX - é descrito a seguir: dadas duas rotas A e B, um grafo G é construído da união dos arcos contidos nos dois *tours*. A partir desse grafo, são geradas *strings* pelo procedimento *cria_strings()*, conforme mostrado na figura 3. Em seguida, as *strings* geradas são emendadas pelo procedimento *patch_strings()*, que faz a concatenação com base no procedimento construtivo do vizinho mais próximo, ou seja: dada uma string inicial, o procedimento busca emendar a ultima cadeia considerada aquela que incorrer no menor aumento de custo.

Passo 1: Seleciona-se aleatoriamente um vértice de G ainda não utilizado para ser o vértice inicial de uma nova string. Marca-se esse vértice como usado e o declara como a cauda t e a cabeça h da string atual.
Passo 2: Enquanto a cabeça da cadeia atual possui ao menos um arco de saída para um vértice não utilizado, faz-se o seguinte: escolhe-se aleatoriamente um dos vértices não utilizados para o qual h tem um arco de saída, adiciona-o ao final da lista, marca-o como usado e o declara como a nova cabeça da string atual.
Passo 3: Declara h como sendo o final da string atual.
Passo 4: Enquanto a cauda da cadeia atual possui ao menos um arco de entrada vindo um vértice não utilizado, faz-se o seguinte: escolhe-se aleatoriamente um dos vértices não utilizados do qual t recebe um arco de entrada, adiciona-o ao início da lista, marca-o como usado e o declara como a nova cauda da string atual.
Passo 5: Declara t como sendo o início da string atual.

Figura 3.3: Procedimento de criação de *strings* do *crossover* SAX.

3.2.3 Operador de seleção

O operador de seleção determina que soluções serão recombinadas - aos pares - para gerarem as novas soluções, sempre visando à manutenção da diversidade na população. O operador de seleção escolhe os pais de uma nova solução da seguinte forma, para cada sub-população:

- *current* do líder: recombina (*pocket*(líder), *current*(filho1));
- *current* do filho3: recombina (*pocket*(filho3), *current*(líder));
- *current* do filho1: recombina (*pocket*(filho1), *current*(filho2));
- *current* do filho2: recombina (*pocket*(filho2), *current*(filho3));

A designação de qual agente apoiador vai ser chamado de filho1, filho2 e filho3 é feita de forma aleatória a cada operação.

3.2.4 Operador de mutação

Com o intuito de se manter um bom nível de diversidade na população, o algoritmo faz com que cada solução *current* da população tenha uma probabilidade de 5% de ser

modificada pelo operador de mutação. Caso a alteração seja efetuada, ela se dá através da realocação de um vértice x qualquer para outra posição aleatória dentro do *tour*.

3.2.5 Operador de busca local

O operador de busca local proposto para algoritmo memético descrito até então consiste de melhorar rotas começando de cidades marcadas com *don't look bits* (Bentley (1990)) durante as fases de recombinação e mutação. Na fase de recombinação, as cidades das extremidades das *strings* têm seus *don't look bits* definidos em "falso". Na fase de mutação, as cidades marcadas são aquelas ligadas aos arcos envolvidos na realocação. Os vértices que possuem seus *don't look bits* definidos como "falso" são considerados críticos e são utilizados como pontos de partida para a busca local. Antes da recombinação e depois da busca local, todas as cidades possuem seus *don't look bits* definidos como "verdadeiro". Em uma dada rota, uma única passagem é feita sobre cada ponto de partida (cidade "crítica"), ou seja, para cada cidade crítica i, faz-se uma execução do algoritmo seguinte - figura 3.4.

Passo 1: Escolhe-se j diferente de prox(i), com j pertencente a s-out-neighrbor(i). Faz-se cont = 0 e vai para o passo 2;
Passo 2: Considera-se a adição do arco (i,j) e a remoção dos arcos (i,a) e (b,j), onde a = prox(i) e b = prev(j). Calcula-se D1 = c[i][a] + c[b][j] c[i][j], onde c[i][j], onde c[i][j] é o custo da aresta (i,j). Faz-se cont = cont+1 e vai para o passo 3;
Passo 3: Começando-se pelo arco (j, prox(j)), seleciona-se o primeiro arco (m, n) diferente de (i,j) no subtour {j, ..., i, j}tal que D2 < D1, D2 = c[m][a] + c[b][n] c[m][n]. Se tal arco é encontrado, vai para o passo 4. Caso contrário, vai para o passo 5.
Passo 4: Efetua-se o movimento sob consideração, ou seja, inserem-se os arcos (i, j), (m, a) e (b, n) e retiram-se os arcos (i, a), (b, j), (m, n). O procedimento é chamado recursivamente, usando como cidades críticas iniciais (cidade i) as cidades a, b, n, m, j e i, nessa ordem.
Passo 5: Se cont = 2, para. Caso contrário, seleciona-se j diferente de prev[i], com j pertencente a p-in-neighbor. Defini-se cont = 2 e invertem-se os índices i e j. Vai para o passo 2.

Figura 3.4: Procedimento de busca local RAI.

No algoritmo descrito - figura 3.4 - utilizam-se duas matrizes chamadas *s-out-neighbor* e *p-in-neighbor*. Cada linha dessas estruturas é referente a um vértice. A linha i da primeira matriz indica os cinco nós que possuem os arcos de menor custo, partindo do vértice i. Analogamente, a linha i de *p-in-neighbor* indica os cinco vértices que possuem os arcos de menor custo que chegam em i. A seguir, tem-se uma ilustração de um movimento da busca local RAI.

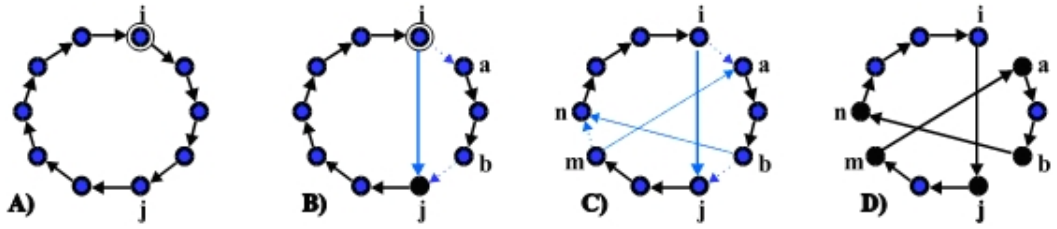

Figura 3.5: Passos da busca local RAI: A) Passo 1; B) Passo 2; C) Passo 3; D) Passo 4.

Capítulo 4

Algoritmos Propostos

Neste capítulo, serão apresentados os detalhes de implementação dos procedimentos propostos. Inicialmente, será vista a técnica de *Path-Relinking* utilizada, feita especificamente para este trabalho. Em seguida, serão apresentados os dois procedimentos de *Vocabulary Building* empregados, propostos originalmente em (Guedes e Aloise (2006)). Ao final do capítulo, será mostrada a forma de integração dos métodos de PR e VB implementados com o Algoritmo Memético utilizado como base.

4.1 Implementação do procedimento *Path-Relinking* aplicado ao problema do caixeiro viajante assimétrico

A seguir, tem-se os detalhes específicos da implementação da técnica de PR feita para esse trabalho.

```
procedimento Implementação de Path_Relinking
entrada: solução-origem, solução-destino;
início
   solução-atual = solução-origem;
   para cada par ordenado (x,y) de soluções pocket faça
     para cada vértice i não fixado  em ordem aleatória faça
        j := solucao-destino.prox(i);
        a := solucao-atual.prox(i);
        b := solucao-atual.prev(j);
        acrescenta em solução-atual a aresta (i, j);
         reloca o trecho {a, ..., b} na melhor posição possível, em solução-atual.
         fixa o vértice i;
      fim-para
    fim-para
    retorna melhor solução-atual encontrada;
fim
```

Figura 4.1: Implementação de *Path-Relinking*.

Na figura 4.1, observa-se que o processo de *Path-Relinking* é executado para cada par de soluções *pocket* contidas na população. Como a população contém 13 agentes, o loop interno do algoritmo dado é executado $13 \times 13 - 13 = 156$ vezes. A cada passagem desse *loop*, acrescenta-se à solução atual uma aresta de solução-destino que ainda não esteja presente - a aresta (i,j). Com o acréscimo dessa aresta, o trecho {a, ..., b} é desconectado da solução e precisa ser realocado. Feito isso, passa-se para a próxima iteração do *loop*.

É interessante notar que esse procedimento é semelhante à inserção de arcos proposta na busca local RAI, sendo fácil perceber que o processo de realocação feito é semelhante ao que é feito na fase de busca local. A diferença está na existência de uma determinação, a priori, dos arcos que devem ser inseridos a cada passo.

4.2 Implementação do procedimento construção de vocabulário (método-1) aplicado ao problema do caixeiro viajante assimétrico

Nesse primeiro método do VB, utilizaremos um *pool* de vocábulos, que serão usados e atualizados durante o processo de busca, além de um conjunto de soluções completas. Na figura 17, tem-se uma ilustração de uma possível solução completa e de um possível vocábulo aplicado ao PCV.

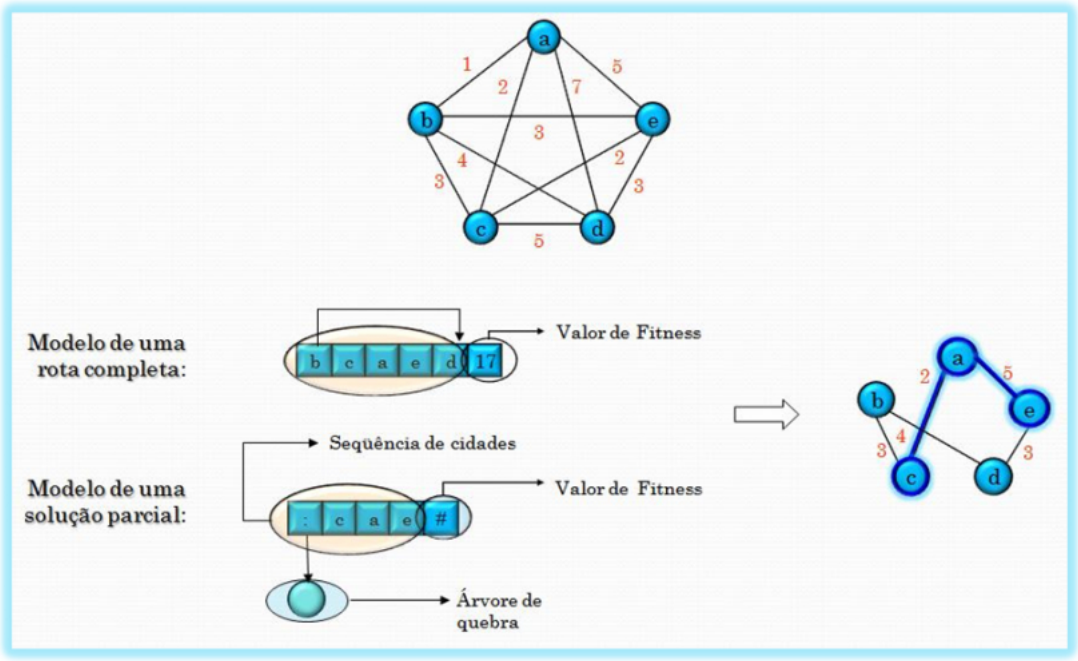

Figura 4.2: Representação de uma solução completa e de um vocábulo para o PCV.

O trecho {c, a, e} indica uma sequência de cidades que será utilizada com o intuito de melhorar alguma solução completa. O campo representado por ":" indica o histórico de construção do vocábulo; denominamos de "árvore de quebra" o círculo apontado por esse campo. O campo "#" funciona como uma *fitness* do vocábulo, indicando como vai a busca, o que é decisivo para a tomada de decisões sobre o destino do vocábulo.

O processo é iniciado com um pool de vocábulos, gerados de forma aleatória e que podem ser combinados a fim de gerar trechos mais úteis. Então, aplica-se o *pool* nas soluções-

elite, verificando quais trechos produzem melhoras e quais não produzem, a fim de manter os vocábulos de alta qualidade e alterar/descartar os de baixa qualidade.

4.2.1 Aplicação do *Pool* de Vocábulos

A aplicação do *Pool* de Vocábulos é realizada sobre uma solução completa (rota completa), com o intuito de melhorar a solução, colocando trechos disponíveis no pool de Vocábulos. A seguir, temos a ilustração do algoritmo desse procedimento.

```
procedimento aplicar_pool_em_solucao(P, S);
        Entrada: Pool de vocábulos P, Solução completa S;
        Saída: Melhor solução encontrada, das inserções realizadas;
1.      melhor.fitness  8;
2.      para cada vocábulo V em P faça
3.              auxiliar  S;
4.              atual  inserir_vocabulo_em_solucao(V, auxiliar, P);
5.              se atual.fitness < melhor.fitness então
6.                      melhor  atual;
7.              fim-se;
8.      fim-para;
9.      retorna melhor;
fim-procedimento;
```

Figura 4.3: Procedimento de aplicação de um *pool* de vocábulos em uma solução completa.

O algoritmo acima faz a inserção de cada vocábulo do *Pool*, separadamente, na solução dada como entrada (geralmente solução elite). Retorna-se a melhor solução encontrada como resultado de uma inserção. Caso que isso não ocorra, ou seja, que esta não tenha uma melhora em sua *fitness*, a inserção realizada será mantida na solução. Abaixo, tem-se o algoritmo do procedimento de inserção de um vocábulo em uma solução.

```
procedimento inserir_vocabulo_em_solucao(V, S, P);
        Entrada: Vocábulo V, Solução completa S, Pool de vocábulos P;
        Saída: Solução completa após V ser inserido em S;
1.      res  transcrever(V, S);
2.      se res.fitness < S.fitness então
3.              V.contador  V.contador + 1;
4.      senão
5.              V.contador  V.contador  1;
6.      fim-se;
7.      se V.contador = 0 então
8.              se V.arvore_quebra = NULO então // Vocábulo elementar
9.                      se rand() < PROB então // rand(), PROB  [0..1]
10.                             alterar_vocabulo(V, S);
11.                             V.contador  CONT_INI;
12.                     senão
13.                             V  novo_vocabulo_aleatorio();
14.                             V.contador  0;
15.                     fim-se;
16.             senão
17.                     desnaturar_vocabulo(V, X, Y);
18.                     X.contador  CONT_MAX;
19.                     Y.contador  CONT_MAX;
20.                     P  P  {V};
21.                     P  P  {X} {Y};
22.             fim-se;
23.     fim-se;
24.     se V.contador = CONT_MAX então
25.             combinar_vocabulo(V, P);
26.             V.contador  CONT_INI;
27.     fim-se;
28.     retorna res;
fim-procedimento;
```

Figura 4.4: Procedimento de inserção de um vocábulo em uma solução completa.

No procedimento acima, obtemos a rota S', retirando de S as cidades que estão contidas em V. Então, verifica-se qual a melhor posição para a inserção do trecho em S'. Depois de introduzido, retorna-se a nova rota completa obtida, mantendo-se a ordem das cidades, com exceção das que estão presentes no vocábulo. Esse procedimento é ilustrado na figura abaixo.

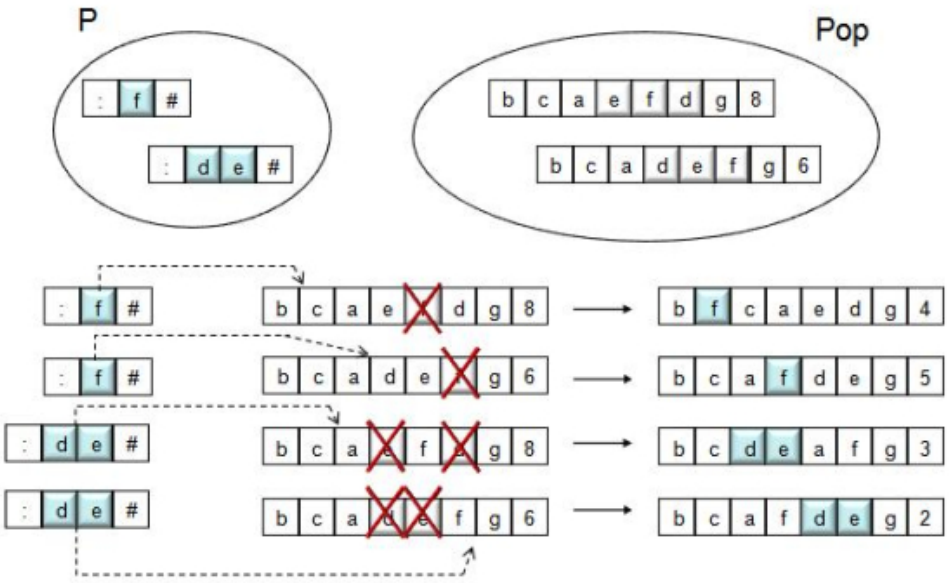

Figura 4.5: Inserção de vocábulo em solução completa .

Na figura 4.5 temos 2 rotas completas e 2 vocábulos a serem inseridos, um vetor de 2 posições e outro de uma. Repare que para ambos os casos o processo é o mesmo. Retira-se das rotas completas as cidades em comum com as do vocábulo. Verifica-se a estrutura obtida após a retirada das cidades contidas no vocábulo, para em seguida introduzir o vocábulo na rota incompleta obtida. Após verificar todas as posições possíveis, o procedimento retorna a solução que possui melhor *fitness*.

Feita a transcrição, o procedimento ***inserir_vocabulo_em_solucao()*** verifica se a operação causou uma melhora na solução dada. Em caso positivo, incrementa-se o contador do vocábulo; caso contrário, seu contador é decrementado. Depois se verifica: se o contador do vocábulo chegou a um valor muito alto, realiza-se a combinação desse vocábulo com algum outro do Pool. Caso contrário, é necessário saber se o vocábulo é composto (originado por dois outros vocábulos) ou elementar (representando apenas uma aresta). Se for um vocábulo

composto, ele é retirado do pool e quebrado em dois. As partes resultantes da quebra são devolvidas ao pool, com seus contadores reinicializados no valor máximo (é necessário que haja um valor máximo, pois um vocábulo pode se mostrar interessante em um momento da busca e, logo depois, se tornar incapaz de realizar melhorias). Para o caso de o vocábulo ser elementar (árvore de quebra não guarda nenhuma informação), gera-se um número aleatório e, dependendo de uma probabilidade PROB, pode-se realizar a alteração ou o descarte do vocábulo.

Partindo do princípio que o vocábulo já efetuou o processo de inserção e o trecho contido no vocábulo também está presente na solução completa, observemos a figura abaixo.

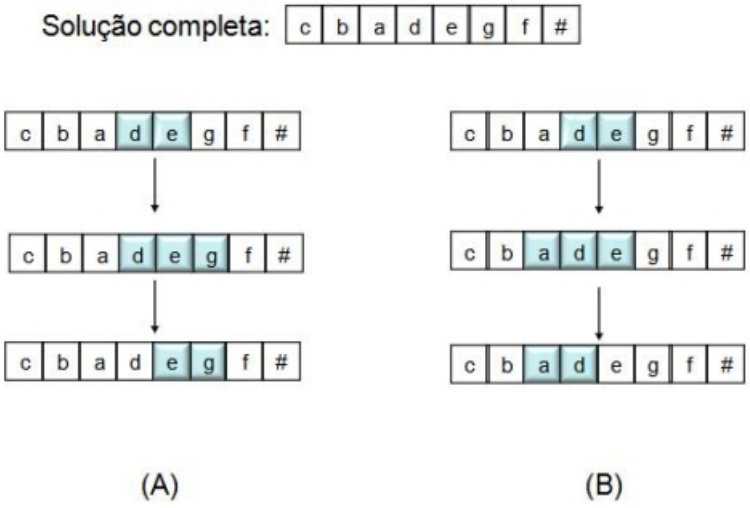

Figura 4.6: Alteração do vocábulo {d,e} utilizando-se como referência a solução completa {c,b,a,d,e,g,f,}.

A alteração do vocábulo é feita de acordo com o vetor-solução em que ele foi inserido.

Na figura acima, temos dois exemplos de alteração: em **4.6.A**, o trecho {d,e} passa a ser {e,g}; em **4.6.B**, o mesmo trecho se altera para {a,d}. Esse processo funciona como uma permuta de uma das partes do vocábulo com uma parte da solução. Onde é sempre escolhida a alteração que resultar na menor relação *peso da aresta inserida - peso da aresta retirada.*

O processo de combinação ocorre quando um vocábulo X atinge um alto valor em seu contador, indicando que ele tem realizado melhora no conjunto de soluções vigente. Com isso, procura-se um vocábulo Y que seja compatível e que tenha o maior contador possível. Dois vocábulos são considerados compatíveis quando não possuem nenhuma cidade em comum ou quando o início de um coincide com o final do outro; dessa forma, não existe a possibilidade de haver vocábulos com cidades duplicadas. Abaixo, segue uma ilustração do processo de combinação, que mostra a construção da árvore de quebra de um vocábulo.

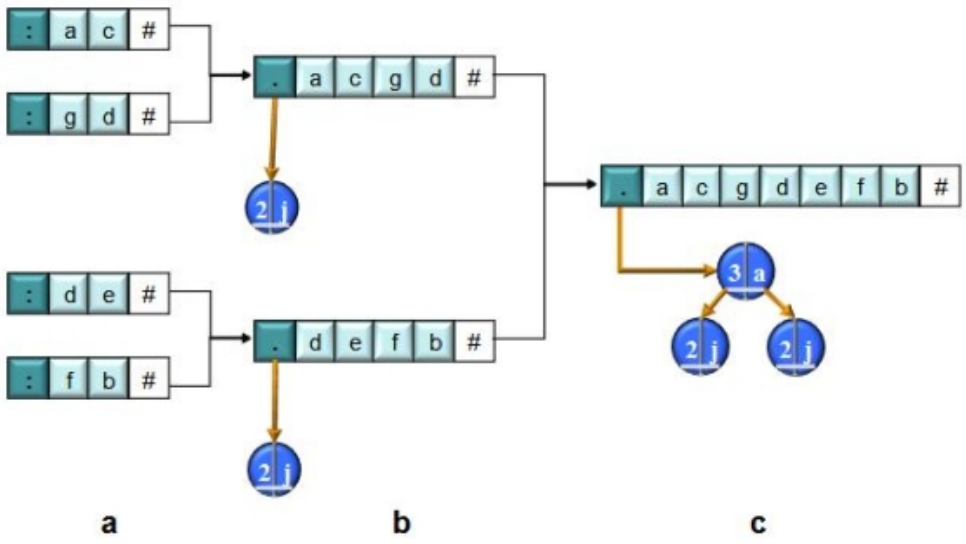

Figura 4.7: Combinação de vocábulos.

Em **4.7.a**, tem-se quatro vocábulos distintos. De **4.7.a** para **4.7.b**, ocorre a combinação dos dois vocábulos de cima e dos dois vocábulos de baixo. A árvore de quebra retém a informação necessária para realizar a separação desses vocábulos: o número 2 indica que o segundo vocábulo se inicia a partir da posição número 2 (terceira posição) do vetor-solução; a letra **j** (de Justaposição) indica que os vocábulos originais não possuíam nenhuma cidade em comum. De **4.7.b** para **4.7.c**, ocorre a combinação dos dois vocábulos recém-gerados. Como eles possuem uma cidade da extremidade em comum, esse fato está indicado pela letra **a** (de Aglutinação) da árvore de quebra do vocábulo resultante. O numeral do nó-raiz, mais uma vez, indica a posição a partir da qual o vetor armazena o segundo vocábulo da combinação. A grande vantagem conseguida com a árvore de quebra é possibilidade de realizar o processo inverso chamado de desnaturação, dentro do procedimento ***inserir_vocabulo_em_solucao()***.

4.3 Implementação do procedimento construção de vocábulos (método-2) aplicado ao problema do caixeiro viajante assimétrico

A segunda implementação do *Vocabulary Building* apresentada nesse trabalho utiliza a noção de contração de vértices e faz uso da idéia proposta por Glover. Apresenta-se um mecanismo de combinação que se utiliza de um conjunto de soluções para construir vocábulos e, posteriormente, realizar um processo de montagem para se obter soluções completas (nesse caso, ciclos Hamiltonianos). Como regras básicas, utiliza-se heurísticas construtivas.

O mecanismo proposto funciona da seguinte forma:

1. Cada heurística construtiva produz um determinado número de soluções;
2. Considerando-se todas as soluções geradas, identificam-se as arestas que estão presentes em todas elas;
3. Com esse conjunto de arestas em mãos, identificam-se e concatenam-se trechos que são formados por arestas adjacentes (processo que chamamos de *normalização*);
4. De posse desses trechos, cria-se um grafo auxiliar no qual cada trecho é representado por um único vértice (contração - ver figura 2.8); A matriz de custos $[a_{ij}]$ desse grafo auxiliar é calculada de acordo com YEO (1997) e GLOVER, GUTIN, YEO e ZVEROVICH (2001), sendo $[g_{ij}]$ a matriz de custos do grafo original:
   a. $a_{ij} = g_{ij}$; se i e j não representam trechos (i e j são vértices comuns, que não fazem parte de nenhuma aresta identificada pelo mecanismo descrito acima);
   b. $a_{ij} = g_{\text{fim}(i),j}$; se i representa um trecho e j representa um vértice comum. $\text{fim}(i)$ equivale ao último vértice do trecho representado por i;
   c. $a_{ij} = g_{i,\text{ini}(j)}$; se i representa um vértice comum. $\text{ini}(j)$ equivale ao primeiro vértice do trecho j; e
   d. $a_{ij} = g_{\text{fim}(i),\text{ini}(j)}$; se ambos i e j representam trechos
5. Com a matriz de custos definida, cria-se um conjunto de soluções para esse grafo auxiliar (neste trabalho, isso é feito através de métodos construtivos seguidos de busca local);

6. Faz-se a "tradução" das soluções produzidas com o grafo auxiliar para soluções viáveis com o grafo original. Isso é feito através da expansão dos nós que representam os trechos identificados anteriormente. A esse conjunto de soluções viáveis damos o nome de *População Auxiliar*;

7. Executa-se busca local nas soluções da *População Auxiliar*;

8. Insere-se a melhor solução do conjunto atual na *População Auxiliar*;

9. Exclui-se a pior solução da *População Auxiliar* e substitui-se o conjunto de soluções atual com a primeira.

A seguir, se tem uma ilustração da identificação e contração de trechos presentes em várias soluções.

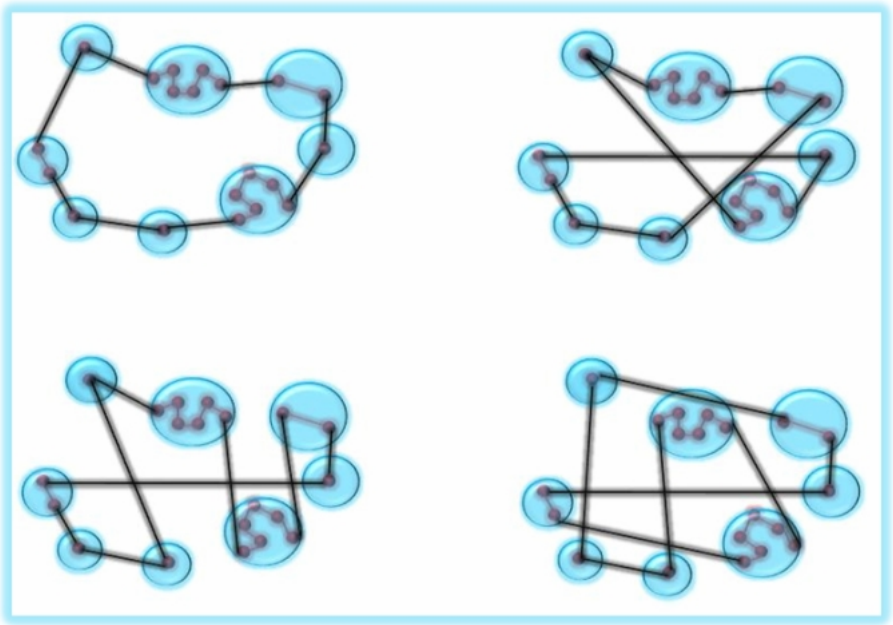

Figura 4.8: Identificação e contração de trechos comuns a soluções-elite.

Esses trechos são, após identificados, condensados. A partir desses novos nós, cria-se um grafo auxiliar e tenta-se resolver o problema sobre essa nova estrutura.

A principal idéia subjacente a esse procedimento é a suposição de que uma variável que está com um determinado valor em várias soluções de boa qualidade tende a possuir esse mesmo valor em uma solução ótima. Sendo assim, procura-se por arestas que estejam presentes em diversas soluções de alta qualidade e faz-se com que elas estejam automaticamente presentes nas próximas soluções. Além disso, cada heurística atua de uma maneira diferente, considerando-se a importância de uma determinada aresta para a construção de uma solução ótima. É interessante notar que essa suposição está relacionada com aquela feita por Glover (citada anteriormente).

4.4 Algoritmo memético estado-da-arte com construção de vocabuláio e *path-relinking* aplicado ao problema do caixeiro viajante assimétrico

Os procedimentos aqui implementados foram acrescidos ao procedimento do algoritmo memético orginal (estado-da-arte). A seguir, temos a ilustração do algoritmo memético estado-da-arte com PR + VB.

```
procedimento AM_Buriol
início
        inicializa_pop();
        VB_M1_inicializa_pool_de_vocábulos();
        repita
                estrutura_pop();
                VB_M1_aplica_pool_de_vocábulos();
                estrutura_pop();
                recombina_pop();
                muta_pop();
                otimiza_pop();
                estrutura_pop();
                VB_M2_contracao_de_vértices();
                estrutura_pop();
                Path_Relinking();
        até critério_de_parada = satisfeito;
fim
```

Figura 4.9: Algoritmo Memético estado da arte acrescido dos procedimentos de *Vocabulary Building* e *Path-Relinking* implementados.

Inicialmente, a população inicial é gerada, de acordo com o mecanismo utilizado no algoritmo memético original, vistos na subseção 3.2.1. Inicializa-se o pool de vocábulos, utilizado pelo primeiro método 1 de *vocabulary building* implementado. Cada vocábulo é criado de forma aleatória.

Em seguida, executa-se o laço principal do programa, até que o critério de parada seja satisfeito (20 execuções e $13 \times \log_2(13) \times \log_2(n^2)$ iterações para cada uma delas, ou 100 iterações sem melhorias).

Já dentro do laço principal, os seguinte passos são executados:

- A população é estruturada, de acordo com o mecanismo do algoritmo memético original;
- O pool de vocábulos é aplicado sobre todos as soluções *current* da população, de acordo com os procedimentos apresentado na seção 4.2.1;
- A população é novamente estruturada e o corpo principal do algoritmo memético

estado-da-arte é executado, sendo chamados os procedimentos de recombinação, mutação e otimização (busca local) sobre a população atual;

- Em seguida, chama-se o procedimento de contração de vértices, o segundo método de *vocabulary building* implementado;
- E, por fim, é estruturada mais uma vez a população e chama-se o algoritmo de *Path Relinking*.

Capítulo 5

Resultados Computacionais

A TSPLIB é uma biblioteca de instâncias para o PCV e problemas correlatos. Ela tem sido bastante utilizada como parâmetro na comparação entre diversos algoritmos para o PCV (simétrico e assimétrico) e pode ser considerada como uma base de testes padrão para esse tipo de problema. Os dados disponíveis são oriundos de várias fontes e, para o caso específico do PCVA, a biblioteca possui 27 instâncias. Cada uma dessas instâncias representa um problema real cuja solução ótima é conhecida. Na Tabela 5.1, tem-se a relação das instâncias assimétricas disponíveis, com o nome, o número de vértices e o valor ótimo para cada uma delas.

Instância	No. de cidades	Valor ótimo	Instância	No. de cidades	Valor ótimo
br17	17	39	**ftv100**	101	1788
ftv33	34	1286	**kro124p**	100	36230
ftv35	36	1473	**ftv110**	111	1958
ftv38	39	1530	**ftv120**	121	2166
p43	43	5620	**ftv130**	131	2307
ftv44	45	1613	**ftv140**	141	2420
ftv47	48	1776	**ftv150**	151	2611
ry48p	48	14422	**ftv160**	161	2683
ft53	53	6905	**ftv170**	171	2755
ftv55	56	1608	**rbg323**	323	1326
ftv64	65	1839	**rbg358**	358	1163
ft70	70	38673	**rbg403**	403	2465
ftv70	71	1950	**rbg443**	443	2720
ftv90	91	1579			

Tabela 5.1: Instâncias assimétricas da TSPLIB.

No presente trabalho, foram utilizadas todas as 27 instâncias assimétricas da TSPLIB, realizando-se 20 execuções e $13 \times \log_2(13) \times \log_2(n^2)$ iterações para cada uma delas, ou 100 iterações sem melhorias (conforme feito no artigo onde o Algoritmo Memético utilizado foi originalmente proposto).

Os testes foram realizados em uma máquina com processador AMD Athlon X2 2.3 Ghz, com 1 GB de memória RAM, sobre o sistema operacional Linux Ubuntu 8.04. Todos os algoritmos foram implementados utilizando-se a linguagem C++ e o compilador GCC (GNU C/C++ Compiler) versão 4.2.4.

5.1 Algoritmo memético estado-da-arte puro

Nesta seção, serão vistos os resultados obtidos pelo algoritmo estado-da-arte para a resolução do PCVA, conforme publicado em Buriol et al. (2004). Também serão vistos os resultados obtidos com implementação feita para este trabalho.

Na Tabela 5.2, pode-se observar os resultados apresentados em Buriol et al. (2004). Os dados exibidos são:

- **Gap Inicial (%)**: erro relativo entre o resultado obtido e o valor da solução ótima, em porcentagem; equivalente a: 100 × (resultado - ótimo)/ótimo;
- **Ótimo (Nº de vezes)**: Número de vezes, nas 20 execuções, em que foi alcançada a solução ótima;
- **Gap (%) na média**: erro relativo, calculado conforme a fórmula mostrada acima, do valor médio dos resultados obtidos nas 20 execuções;
- **Tempo Médio de CPU (seg)**: tempo médio de processamento, nas 20 execuções; e
- **Tempo Médio para o Ótimo (seg)**: tempo médio para se alcançar a solução ótima.

É possível observar que o algoritmo foi capaz de encontrar as soluções ótimas para todas as 27 instâncias. Algumas delas - ftv38, ftv44, ftt70, ftv120 - parecem ser mais difíceis de resolver do que as demais, tendo em vista que o algoritmo encontrou o valor ótimo um número de vezes menor do que para as outras instâncias, nas 20 execuções realizadas.

Instância	Gap Inicial	Ótimo/Tentativas	Gap(%)	Gerações	Tempo	Tempo OPT
br17	0,000	20	0,000	94,000	0,140	0,050
ftv33	12,830	20	0,000	107,800	0,290	0,050
ftv35	0,140	20	0,000	113,300	0,330	0,080
ftv38	0,130	5	0,100	105,700	0,310	0,260
p43	0,050	11	0,010	119,650	0,480	0,350
ftv44	7,010	7	0,440	116,750	0,410	0,360
ftv47	2,700	20	0,000	110,000	0,470	0,130
ry48p	5,420	17	0,030	126,350	0,560	0,320
ft53	18,200	20	0,000	120,750	0,560	0,200
ftv55	3,610	20	0,000	117,050	0,500	0,160
ftv64	3,810	20	0,000	126,900	0,660	0,240
ft70	1,880	8	0,030	141,000	0,960	0,860
ftv70	3,330	19	0,010	137,200	0,830	0,380
ftv90	3,670	20	0,000	124,500	0,950	0,280
ftv100	3,240	20	0,000	130,650	1,200	0,400
kro124p	6,460	18	0,010	142,150	1,370	0,740
ftv110	4,700	18	0,020	148,550	1,710	0,980
ftv120	8,310	7	0,140	156,950	2,190	2,000
ftv130	3,120	18	0,010	156,450	2,180	1,300
ftv140	2,230	14	0,080	164,950	2,660	2,110
ftv150	2,300	18	0,010	161,250	2,690	1,540
ftv160	1,710	16	0,020	161,150	3,280	2,040
ftv170	1,380	15	0,050	164,750	3,920	2,780
rbg323	0,000	20	0,000	0,000	0,070	0,070
rbg358	0,000	20	0,000	0,000	0,080	0,080
rbg403	0,000	20	0,000	0,000	0,080	0,080
rbg443	0,000	20	0,000	0,000	0,090	0,090
Média	3,564	16,704	0,036	112,881	1,073	0,664

Tabela 5.2: AM puro - resultados apresentados em Buriol et al. (2004).

A seguir, na tabela 5.3, são mostrados os valores alcançados com a implementação do algoritmo estado-da-arte feita para esta dissertação. É possível verificar que os valores encontrados são bem semelhantes àqueles publicados em Buriol et al. (2004), o que valida a implementação feita para o presente trabalho. Deste ponto em diante, toda referência ao algoritmo proposto em Buriol et al. (2004) será relativa a essa implementação.

Instância	Gap Inicial	Ótimo/Tentativas	Gap(%)	Gerações	Tempo	Tempo OPT
br17	0,000	20	0,000	100,000	0,050	0,000
ftv33	13,033	20	0,000	103,800	0,050	0,000
ftv35	1,249	19	0,007	119,200	0,050	0,000
ftv38	1,163	12	0,052	116,450	0,000	0,000
p43	0,349	19	0,001	130,400	0,100	0,050
ftv44	7,347	5	0,465	124,250	0,100	0,100
ftv47	0,732	20	0,000	106,600	0,050	0,000
ry48p	7,232	20	0,000	146,450	0,100	0,050
ft53	12,991	20	0,000	125,350	0,050	0,000
ftv55	4,478	20	0,000	113,750	0,050	0,000
ftv64	6,699	20	0,000	124,050	0,100	0,000
ft70	1,882	11	0,034	164,950	0,150	0,150
ftv70	3,385	20	0,000	129,800	0,100	0,050
ftv90	2,255	20	0,000	125,900	0,150	0,000
ftv100	1,823	20	0,000	131,650	0,150	0,100
kro124p	16,398	19	0,002	140,250	0,150	0,100
ftv110	2,574	19	0,005	165,250	0,250	0,150
ftv120	5,328	14	0,060	178,650	0,300	0,250
ftv130	3,351	18	0,026	163,250	0,350	0,200
ftv140	2,632	11	0,105	179,800	0,350	0,250
ftv150	2,579	19	0,050	179,500	0,450	0,100
ftv160	0,596	19	0,007	179,900	0,450	0,300
ftv170	2,178	15	0,116	200,600	0,700	0,400
rbg323	0,000	20	0,000	100,000	2,350	0,050
rbg358	0,000	20	0,000	0,000	0,100	0,100
rbg403	0,000	20	0,000	0,000	0,250	0,250
rbg443	0,000	20	0,000	0,000	0,350	0,350
Média	3,713	17,778	0,034	124,067	0,270	0,111

Tabela 5.3: AM puro - implementação do presente trabalho.

A seguir, na tabela 5.4, são mostrados os valores obtidos com o algoritmo memético estado-da-arte acrescido das técnicas de *path-relinking* (PR) e *vocabulary building* (VB) propostos. É possível observar que houve uma pequena melhoria no número de soluções ótimas obtidas e no gap médio das soluções encontradas. O número de gerações utilizadas também sofreu uma pequena redução. A maior diferença observada foi em relação ao tempo de computacional. O procedimento com PR e VB gastou, em média, mais tempo para finalizar a sua execução e para atingir uma solução ótima.

Instância	Gap Inicial	Ótimo/Tentativas	Gap(%)	Gerações	Tempo	Tempo OPT
br17	0,000	20	0,000	100,000	0,100	0,000
ftv33	13,033	20	0,000	103,200	0,250	0,000
ftv35	1,249	20	0,000	110,700	0,250	0,050
ftv38	1,163	10	0,065	113,200	0,300	0,100
p43	0,349	20	0,000	135,450	0,300	0,150
ftv44	7,347	6	0,434	109,050	0,300	0,250
ftv47	0,732	20	0,000	103,300	0,350	0,000
ry48p	7,232	20	0,000	144,300	0,600	0,300
ft53	12,991	20	0,000	114,700	0,400	0,100
ftv55	4,478	20	0,000	112,200	0,400	0,000
ftv64	6,699	20	0,000	119,300	0,500	0,050
ft70	1,882	18	0,009	178,500	0,850	0,450
ftv70	3,385	20	0,000	129,050	0,550	0,100
ftv90	2,255	20	0,000	121,100	0,750	0,150
ftv100	1,823	20	0,000	125,400	0,800	0,150
kro124p	16,398	20	0,000	130,350	0,900	0,350
ftv110	2,574	19	0,005	168,900	1,450	0,550
ftv120	5,328	11	0,078	181,000	1,900	1,400
ftv130	3,351	17	0,030	162,000	1,800	0,900
ftv140	2,632	12	0,093	196,050	2,500	1,850
ftv150	2,579	20	0,000	170,350	2,200	0,750
ftv160	0,596	17	0,020	169,050	2,650	1,350
ftv170	2,178	13	0,093	187,050	3,450	2,200
rbg323	0,000	20	0,000	0,000	0,100	0,100
rbg358	0,000	20	0,000	0,000	0,050	0,050
rbg403	0,000	20	0,000	0,000	0,100	0,100
rbg443	0,000	20	0,000	0,000	0,500	0,500
Média	3,713	17,889	0,031	117,933	0,900	0,443

Tabela 5.4: AM+PR+VB.

Com a obtenção desses resultados, inferiores ao esperado, decidiu-se modificar a estratégia de inicialização usada pelo algoritmo memético empregado. Em vez de utilizar métodos construtivos (gulosos) para gerar a população inicial, o procedimento foi modificado para que suas soluções inicias fossem geradas aleatoriamente. O algoritmo memético com inicialização aleatória foi então testado com e sem os procedimentos de PR e VB.

A seguir, na figura 5.1, são mostrados os gráficos comparativos entre o número de ótimos encontrados nas Tabelas 5.3 e 5.4 para as 27 instâncias.

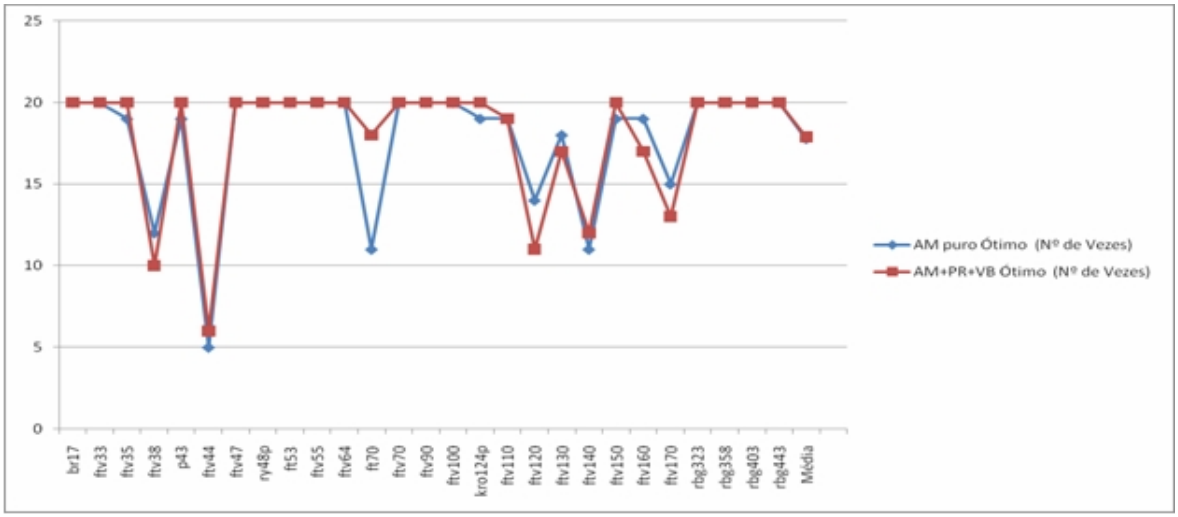

Figura 5.1: Gráfico comparativo do nº de ótimos encontrados pelo AM puro e pelo AM+PR+VB.

O AM Puro foi superior apenas em 5 instâncias contra 7 instâncias do AM+PR+VB.

A seguir, na Tabela 5.5, são apresentados os resultados obtidos com o algoritmo memético puro com inicialização aleatória. Verifica-se que essa variante obteve resultados levemente inferiores aos apresentados na Tabela 5.3. Mais uma vez, o tempo computacional foi o fator mais destoante, sendo necessário mais tempo para alcançar uma solução ótima e para finalizar a execução do algoritmo.

Instância	Gap Inicial	Ótimo/Tentativas	Gap(%)	Gerações	Tempo	Tempo OPT
br17	224,744	20	0,000	100,000	0,050	0,000
ftv33	196,979	20	0,000	102,650	0,050	0,000
ftv35	189,627	17	0,020	112,950	0,000	0,000
ftv38	200,239	12	0,052	116,100	0,000	0,000
p43	217,852	17	0,003	122,500	0,100	0,000
ftv44	237,582	6	0,468	130,700	0,100	0,100
ftv47	246,591	20	0,000	105,400	0,000	0,000
ry48p	235,330	20	0,000	127,900	0,100	0,000
ft53	242,134	20	0,000	125,600	0,100	0,050
ftv55	314,695	20	0,000	108,200	0,050	0,000
ftv64	335,544	20	0,000	126,700	0,100	0,000
ft70	77,790	15	0,018	184,050	0,150	0,150
ftv70	361,577	20	0,000	132,650	0,100	0,000
ftv90	547,549	19	0,019	118,450	0,150	0,050
ftv100	597,662	20	0,000	124,050	0,150	0,000
kro124p	385,981	20	0,000	128,900	0,150	0,050
ftv110	638,399	19	0,005	143,450	0,250	0,050
ftv120	641,540	13	0,069	199,150	0,400	0,300
ftv130	695,368	18	0,020	161,750	0,350	0,150
ftv140	714,736	7	0,163	185,200	0,450	0,300
ftv150	728,254	16	0,107	200,950	0,500	0,450
ftv160	766,198	12	0,093	179,050	0,550	0,500
ftv170	796,330	16	0,033	213,450	0,800	0,550
rbg323	348,024	19	0,004	115,900	3,350	0,900
rbg358	472,911	19	0,004	122,350	3,500	1,350
rbg403	201,229	20	0,000	100,200	3,200	0,550
rbg443	193,801	20	0,000	100,150	4,150	0,750
Média	400,321	17,222	0,040	136,607	0,698	0,231

Tabela 5.5: AM puro com inicialização aleatória.

Na tabela 5.6, são apresentados os resultados referentes ao algoritmo memético com inicialização aleatória e acrescido dos procedimentos de PR e VB. Mais uma vez, observa-se que os procedimentos de PR e VB foram capazes de alcançar melhorias nos resultados obtidos. Da mesma forma, também foi necessário um maior tempo computacional para o término da execução do algoritmo e para a obtenção de uma solução ótima.

Instância	Gap Inicial	Ótimo/Tentativas	Gap(%)	Gerações	Tempo	Tempo OPT
br17	224,744	20	0,000	100,000	0,200	0,000
ftv33	196,979	20	0,000	101,900	0,250	0,000
ftv35	189,627	20	0,000	108,900	0,350	0,050
ftv38	200,239	14	0,039	108,550	0,400	0,100
p43	217,852	20	0,000	129,250	0,450	0,050
ftv44	237,582	6	0,434	105,200	0,450	0,300
ftv47	246,591	20	0,000	102,650	0,450	0,050
ry48p	235,330	20	0,000	110,950	0,550	0,000
ft53	242,134	20	0,000	110,100	0,500	0,050
ftv55	314,695	20	0,000	103,450	0,500	0,000
ftv64	335,544	20	0,000	110,750	0,650	0,000
ft70	77,790	16	0,018	130,500	0,900	0,350
ftv70	361,577	20	0,000	116,450	0,800	0,100
ftv90	547,549	20	0,000	118,000	1,000	0,000
ftv100	597,662	19	0,017	120,600	1,200	0,250
kro124p	385,981	20	0,000	120,050	1,050	0,150
ftv110	638,399	18	0,031	132,250	1,700	0,500
ftv120	641,540	13	0,074	160,700	2,400	1,500
ftv130	695,368	18	0,026	141,400	2,200	0,750
ftv140	714,736	7	0,171	155,400	3,000	2,300
ftv150	728,254	18	0,015	155,050	3,200	1,650
ftv160	766,198	16	0,030	163,950	3,750	2,150
ftv170	796,330	16	0,065	162,900	4,350	2,650
rbg323	348,024	20	0,000	107,250	43,250	2,500
rbg358	472,911	20	0,000	114,400	57,150	7,200
rbg403	201,229	20	0,000	100,400	74,300	0,600
rbg443	193,801	20	0,000	100,050	97,500	0,600
Média	400,321	17,815	0,034	121,891	11,204	0,883

Tabela 5.6: AM+PR+VB com inicialização aleatória.

Visando melhorar ainda mais os resultados obtidos, foi empregada uma nova forma de inicialização da população inicial. Nessa nova proposta, as soluções também são geradas aleatoriamente. Contudo, uma nova solução só é incorporada ao conjunto inicial se menos de 15% das suas arestas já estiverem presentes nessa população. Caso contrário, a solução é descartada e uma outra é gerada.

A seguir, na figura 5.2, são mostrados os gráficos comparativos entre o número de ótimos encontrados nas Tabelas 5.5 e 5.6 para as 27 instâncias.

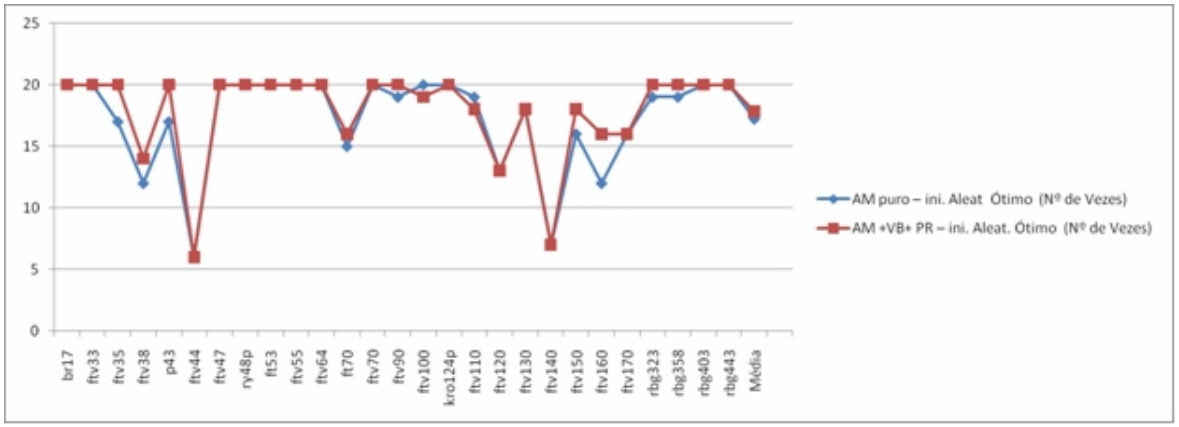

Figura 5.2: Gráfico comparativo do nº de ótimos encontrados pelo AM puro com inicialização aleatória e pelo AM+PR+VB com inicialização aleatória.

O AM Puro com inicialização aleatória foi superior apenas em 2 instâncias contra 9 instâncias do AM+PR+VB com inicialização aleatória.

Na Tabela 5.7, são apresentados os resultados para o algoritmo memético com inicialização aleatória diversificada. Com exceção do tempo necessário para obter-se uma solução ótima, os resultados obtidos são todos inferiores aos do algoritmo relativo à Tabela 5.5.

Instância	Gap Inicial	Ótimo/Tentativas	Gap(%)	Gerações	Tempo	Tempo OPT
br17	229,487	20	0,000	100,000	0,050	0,000
ftv33	196,979	20	0,000	102,250	0,000	0,000
ftv35	189,627	18	0,014	114,350	0,050	0,000
ftv38	200,239	10	0,065	117,700	0,050	0,050
p43	217,852	16	0,004	126,000	0,100	0,000
ftv44	237,582	5	0,465	111,700	0,050	0,000
ftv47	246,591	20	0,000	104,350	0,050	0,000
ry48p	235,330	20	0,000	120,050	0,050	0,000
ft53	242,134	20	0,000	124,100	0,050	0,000
ftv55	314,695	20	0,000	108,700	0,050	0,000
ftv64	335,544	20	0,000	120,650	0,100	0,050
ft70	77,790	13	0,026	183,600	0,150	0,150
ftv70	361,577	20	0,000	141,000	0,100	0,100
ftv90	547,549	20	0,000	122,600	0,150	0,050
ftv100	597,662	19	0,017	124,750	0,150	0,000
kro124p	385,981	20	0,000	128,350	0,150	0,000
ftv110	638,399	18	0,023	146,600	0,250	0,000
ftv120	641,540	14	0,046	186,900	0,350	0,250
ftv130	695,368	17	0,030	158,500	0,350	0,250
ftv140	714,736	8	0,155	185,650	0,450	0,350
ftv150	728,254	19	0,046	210,650	0,550	0,350
ftv160	766,198	12	0,160	190,150	0,500	0,400
ftv170	796,330	14	0,071	189,450	0,700	0,400
rbg323	348,024	18	0,008	116,350	3,400	1,350
rbg358	472,911	20	0,000	137,750	3,800	1,450
rbg403	201,229	20	0,000	100,150	3,350	0,300
rbg443	193,801	20	0,000	100,200	4,100	0,400
Média	400,497	17,074	0,042	136,019	0,707	0,219

Tabela 5.7: AM puro com inicialização aleatória diversificada.

Na tabela 5.8, são apresentados os resultados para o algoritmo memético com PR e VB e inicialização aleatória diversificada. Como todas as variantes que incorporaram as técnicas de PR e VB aqui apresentadas, essa versão necessitou de um maior tempo computacional para concluir a sua execução. Contudo, pode-se verificar que o número médio de soluções ótimas foi superior ao do algoritmo memético puro, assim como o gap médio do valor das soluções encontradas. Essa variante foi capaz de obter as melhores soluções dentre todos os algoritmos testados, conseguindo concluir a sua execução em tempos computacionais viáveis - todos eles menores do que 12 segundos.

Instância	Gap Inicial	Ótimo/Tentativas	Gap(%)	Gerações	Tempo	Tempo OPT
br17	229,487	20	0,000	100,000	0,150	0,000
ftv33	196,979	20	0,000	101,750	0,350	0,000
ftv35	189,627	20	0,000	108,300	0,350	0,000
ftv38	200,239	14	0,039	111,250	0,350	0,150
p43	217,852	20	0,000	118,000	0,450	0,050
ftv44	237,582	5	0,465	107,650	0,500	0,350
ftv47	246,591	20	0,000	102,450	0,450	0,000
ry48p	235,330	20	0,000	118,300	0,550	0,100
ft53	242,134	20	0,000	110,900	0,550	0,000
ftv55	314,695	20	0,000	104,100	0,500	0,050
ftv64	335,544	20	0,000	110,850	0,650	0,100
ft70	77,790	19	0,004	130,450	0,800	0,250
ftv70	361,577	20	0,000	118,100	0,750	0,150
ftv90	547,549	20	0,000	124,400	1,100	0,150
ftv100	597,662	20	0,000	121,950	1,250	0,200
kro124p	385,981	20	0,000	125,650	1,200	0,400
ftv110	638,399	19	0,008	133,850	1,650	0,500
ftv120	641,540	14	0,069	162,600	2,400	1,500
ftv130	695,368	18	0,026	137,550	2,250	0,950
ftv140	714,736	12	0,128	169,200	3,250	2,150
ftv150	728,254	20	0,000	148,100	3,050	1,200
ftv160	766,198	17	0,022	163,000	3,600	1,950
ftv170	796,330	17	0,038	154,850	4,300	2,100
rbg323	348,024	20	0,000	105,900	43,450	2,250
rbg358	472,911	20	0,000	111,550	55,650	5,800
rbg403	201,229	20	0,000	100,250	74,350	0,350
rbg443	193,801	20	0,000	100,200	97,700	0,800
Média	400,497	18,333	0,030	122,265	11,170	0,796

Tabela 5.8: AM+VB+PR com inicialização aleatória diversificada.

A seguir, na figura 5.3, são mostrados os gráficos comparativos entre o número de ótimos encontrados nas Tabelas 5.7 e 5.8 para as 27 instâncias.

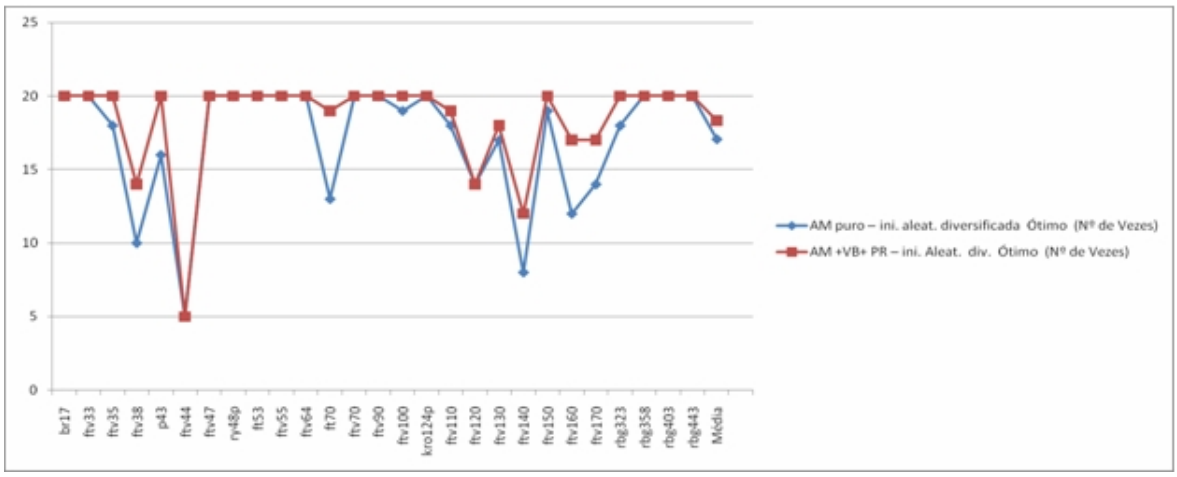

Figura 5.3: Gráfico comparativo do nº de ótimos encontrados pelo AM puro com inicialização aleatória diversificada e pelo AM+PR+VB com inicialização aleatória diversificada.

O AM Puro com inicialização aleatória diversificada foi não superior em sequer uma instância, já o AM+PR+VB com inicialização aleatória diversificada foi superior em 12 instâncias.

A seguir, na Tabela 5.9, tem-se uma quadro comparativo de todas as variantes testadas no presente trabalho. Nesta tabela, fica mais evidente a superioridade dos resultados alcançados com a última variante apresentada. Mais ainda, é possível observar que os procedimentos de PR e VB foram capazes de melhorar os resultados obtidos com o algoritmo memético estado-da-arte, mesmo que a custo de um tempo computacional maior (mas ainda viável e bastante rápido).

Ainda na Tabela 5.9, é possível verificar que o uso das técnicas de PR e VB, aliadas a uma nova forma de inicialização, foram capazes de melhorar em 3,12% o número médio de soluções ótimas encontradas, assim como houve uma melhora de 13,33% no gap final médio. Embora o tempo computacional tenha sido bastante ampliado, pode-se perceber que o algoritmo ainda é bastante eficiente, não precisando mais do que 12 segundos para concluir a sua execução. Um tempo menor ainda foi necessário para a obtenção de soluções ótimas, o que leva a crer que o algoritmo poderia ser melhorado com a utilização de uma nova forma de critério de parada.

Lista de Algoritmos	Gap Inicial	Ótimo/ Tentativas	Gap(%)	Gerações	Tempo	Tempo OPT
AM puro	3,713	17,778	0,034	124,067	0,270	0,111
AM puro – ini. aleatória	400,321	17,222	0,040	136,607	0,698	0,231
AM puro – ini. aleat. diversificada	400,497	17,074	0,042	136,019	0,707	0,219
AM+PR+VB	3,713	17,889	0,031	117,933	0,900	0,443
AM+PR+VB – ini. aleat.	400,321	17,815	0,034	121,891	11,204	0,883
AM+PR+VB – ini. aleat. divers.	400,497	18,333	0,030	122,265	11,170	0,796

Tabela 5.9: Quadro comparativo dos métodos testados.

Capítulo 6

Conclusões e Sugestões para Trabalhos Futuros

O presente trabalho abordou o Problema do Caixeiro Viajante Assimétrico. Essa variante desse famoso problema tem sido menos estudada na literatura, em relação a sua versão simétrica. Contudo, o caso assimétrico é mais geral do que o simétrico, ou seja, um algoritmo capaz de resolver bem o primeiro caso, também resolve o segundo com a mesma eficácia.

A presente dissertação teve como meta realizar melhorias no algoritmo memético considerado estado-da-arte para a resolução do PCVA. Para isso, foram utilizadas as técnicas de otimização conhecidas como *Path-Relinking* e *Vocabulary Building*, sendo a segunda técnica implementada de duas formas distintas.

Inicialmente, implementou-se o algoritmo memético estado-da-arte e verificou-se que os resultados obtidos foram similares aos publicados, validando o uso dessa implementação como base para as tentativas de melhorias. Em seguida, os módulos de *Path-Relinking* e *Vocabulary Building* desenvolvidos foram incorporados ao algoritmo. Testes computacionais foram realizados sobre todas as instâncias assimétricas da biblioteca TSPLIB e verificou-se que o procedimento resultante obteve soluções de qualidade superior ao algoritmo memético original. Em contrapartida, foi observado que houve um aumento no tempo computacional necessário para o término do algoritmo. Contudo, o tempo de processamento permaneceu bastante pequeno.

Também foi testada a influência das técnicas de PR e VB sobre variações do algoritmo memético estado-da-arte. Duas novas formas de inicialização foram verificadas e foi possível observar que ambas causaram uma degradação da performance do algoritmo memético puro. Por outro lado, pôde-se perceber que os novos tipos de inicialização ocasionaram uma melhora ainda mais acentuada nos resultados do algoritmo memético com PR e VB. Essa melhora foi obtida às custas de um aumento ainda maior no tempo computacional, mas o algoritmo ainda continuou bastante eficiente.

O presente trabalho deixa espaço para várias modificações futuras. A primeira delas seria a realização de um estudo sobre a convergência do algoritmo na direção das soluções ótimas. Poderia ser feito uma coleta de dados sobre os valores obtidos pelo algoritmo em cada iteração e seria verificado se os procedimentos de PR e VB conseguem fazer com que o algoritmo memético venha a convergir mais rapidamente para uma solução ótima. O mesmo pode ser feito com relação aos diferentes processos de inicialização testados.

Outra forma de melhoria seria a criação de novos critérios de parada, permitindo que o algoritmo terminasse a sua execução com um menor número de iterações. Cada um desses novos critérios poderia ser testado em conjunto com cada uma das variantes propostas nessa dissertação e seria observado qual a melhor combinação.

Outra forma de melhoria que pode ser explorada seria a utilização de métodos exatos para auxiliar na resolução do problema. Por exemplo, um algoritmo exato poderia ser utilizado para verificar a melhor forma de inserção de um ou mais vocábulos em uma solução, fazendo com que o algoritmo podesse obter melhores soluções com um menor número de iterações.

Por fim, uma outra modificação interessante seria a hibridização do algoritmo memético utilizado com outras formas de implementação de *Path-Relinking* e de *Vocabulary Building*, buscando alcançar uma maior sinergia entre esses processos e os operadores evolutivos do AM.

Referências Bibliográficas

[1] Aloise, D. (2005). *Heurísticas para o projeto de redes com funções de custo discretas.* Dissertação de Mestrado - Departamento de Informática, PUC-Rio, Rio de Janeiro.

[2] Applegate, D. L. (2006) - *The travelling salesman problem: a computational Study.* Princeton: Princeton University Press,. ISBN 978-0-691-12993-8.

[3] Balas, E. e P. Toth. (1985). "Branch-and-Bound Methods." In E. L. Lawler, J.K. Lentra, A. H. G. Rinnoy Kan e D. B. Shmoys (eds.), *The Traveling Salesman Problem*, New York: John Wiley e Sons.

[4] Berretta, R. e P. Moscato. (1999). "The Number Partitioning Problem: An Open Challenge for Evolutionary Computation." In D. Corne, M. Dorigo e F. Glover (eds.), *New Ideas in Optimization*, (chapter 17). Washington: McGraw-Hill.

[5] Buriol, L. (2000). Algoritmo Memético para o Problema do Caixeiro Viajante Assimétrico como Parte de um Framework para Algoritmos Evolutivos. Dissertação de Mestrado - Departamento de Engenharia de Sistemas da Faculdade de Engenharia Elétrica e de Computação, Unicamp.

[6] Buriol, L.; França, P. M.; Moscato, P (2004). A New Memetic Algorithm for the Asymmetric Traveling Salesman Problem. *Journal of Heuristics*, 10, 483-506.

[7] Campello, R. E.; Maculan Filho, N. (1994). Algortimos e Heurísticas: Desenvolvimento e Avaliação de Performance. EDUFF: Editora Universitária da Universidade Federal Fluminense, Niterói.

[8] Carpaneto, G., M. Dell´Amico e P. Toth. (1995). "Exact Solution of Large-Scale, Asymmetric Traveling Salesman Problems." *ACM Transactions on Mathematical Software* **21** (4), 394-409.

[9] Cerqueira, F. R e Stelzer, R. B. P (abr/set 2005). *Algoritmos genéticos aplicados a Mapeamento físico de DNA*. Revista Educação e Tecnologia, Ano 1, n,1.

[10] Cirasella, J.; Johnson, D.S.; Mcgeoch, L.A.; Zhang, W (2001). The Asymmetric Traveling Salesman Problem: Algorithms, Instance Generators and Tests. In: WORKSHOP ON ALGORITHM ENGINEERING AND EXPERIMENTS (ALENEX), 3, Washington. **Proceedings**... Washington: Springer, p.32-59.

[11] Coelho A. M. (2006). *Uma abordagem via algoritmos meméticos para a solução do problema de horário escolar*. Dissertação de Mestrado-CEFET-MG, Belo Horizonte.

[12] França, P. M., A. S. Mendes e P. Moscato. (1999). "Memetic Algorithms to Minimize Tardiness on a Single Machine with Sequence-Dependent Setup Times." In *Proceedings of the 5th International Conference of the Decision Sciences Institute*, Athens, Greece, 1708-1710.

[13] Fischetti, M. e P. Toth. (1997). "A Polyhedral Approach to the Asymmetric Traveling Salesman Problem." *Management Science* **43** (11), 1520-1536.

[14] Freisleben, B.; Merz, P. (1996). A Genetic Local Search Algorithm for Solving Symmetric and Asymmetric Traveling Salesman Problems. In Proceedings of the IEEE International Conference on Evolutionary Computation.

[15] Glover, F; Laguna, M. & Marti, R. (2000). Fundamentals of Scatter Search and Path Relinking. *Control and Cybernetics*, **29**(3), 653-684.

[16] Glover, F. (1992). New Ejection Chain and Alternating Path Methods for Traveling Salesman Problems. **In**: *Computer Science and Operations Research: New Developments in Their Interfaces* [editado por R. Sharda, O. Balci e S. Zenios], Cambridge: Elsevier, 449-509.

[17] Glover, F. (1998). A Template for Scatter Search and Path Relinking. **In**: *Lecture Notes in Computer Science* [editado por J.K. Hao, E. Lutton, E. Ronald, M. Shoenauer e D. Snyers], **1363**, 13-54.

[18] Glover, F. (1999). Scatter Search and Path Relinking. **In**: *New Ideas in Optimization* [editado por D. Corne, M. Dorigo e F. Glover], London: McGraw Hill, 297-319.

[19] Glover, F. (2003). Tutorial on Surrogate Constraint Approaches for Optimization in Graphs. *Journal of Heuristics*, **9**(3), 175-228.

[20] Glover, F.; Gutin, G.; Yeo, A.; Zverovich, A. (2001). Construction Heuristics for the Asymmetric TSP. **European Journal of Operational Research**, vol. 129, n. 3, p. 555-568.

[21] Glover, F. & Laguna, M. (1993). Tabu Search. **In**: *Modern Heuristic Techniques for Combinatorial Problems* [editado por C. Reeves], Oxford: Blackwell Scientific Publishing, 71-140.

[22] Glover, F. & Laguna, M. (1998). *Tabu search*. Massachusetts: Kluwer Academic Publishers.

[23] Goldberg, D.E. (1989). Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley.

[24] Gorges-Schleuter, M. (1989). "ASPARAGOS: An Asynchronous Parallel Genetic Optimization Strategy." In J. D. Schaffer (ed.), *Proceedings of the Third International Conference on Genetic Algorithms*, 422-427.

[25] Gorges-Schleuter, M. (1997). "Asparagos96 and the Traveling Salesman Problem." In T. Baeck, Z. Michalewicz e X. Yao (eds.), *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation*, Indianapolis, USA, 171-174.

[26] Guedes, A. B. da C.; Aloise. D. J. (2006). Um algoritmo memético para o problema do caixeiro viajante assimétrico: Uma abordagem baseada em vocabulary building. SBPO.

[27] Gutin, G.; Punnen, A. P. (2002). *The Traveling Salesman Problem and Its Variations*. Boston: Springer.

[28] Hoffman, Karla; Padberg, Manfred - *Traveling salesman problem* [Em Linha]. Fairfax [Virgínia]: Department of Electrical and Computer Engineering - George Mason University, 2000. [Consult. 8 Mar. 2009]. Disponível em WWW: ¡URL: http://iris.gmu.edu/∼khoffman/papers/trav_salesman.html¿

[29] Holstein, D.; Moscato, P. (1999). Memetic Algorithms Using Guided Local Search: A Case Study. In D. Corne, M. Dorigo, and F. Glover (eds.), New Ideas in Optimization, McGraw-Hill.

[30] Jünger, M., G. Reinelt e G. Rinaldi. (1995). "The Traveling Salesman Problem." In M. Ball, T. Magnanti, C.L. Monma e G. L. Nemhauser (eds.), *Handbooks in Operations Research and Management Sciences: Networks*. North-Holland.

[31] Karp, R. M.; Steele, J. M. (1985). Probabilistic analysis of heuristics. In: E.L. Lawler et al. (Eds.), *The Traveling Salesman Problem*, Wiley, New York.

[32] Kelly, J.P. & Xu, J. (1995). Tabu Search and Vocabulary Building for Routing Problems. Technical Report, Graduate School of Business Administration, University of Colorado at Boulder.

[33] Laporte, G. (1992). "The Traveling Salesman Problem: An Overview of Exact and Approximate Algorithms." *European Journal of Operational Research* **59** (2), 231-247.

[34] Leite, J. N. de F (2006). *Algoritmo Memético e Vocabulary Building: Uma Aplicação ao Problema do Caixeiro Viajante Assimétrico*. Dissertação de Mestrado - Departamento de Informática e Matemática Aplicada, UFRN, Natal.

[35] Merz, P. e B. Freisleben. (1997). "Genetic Local Search for the TSP: New Results." In *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation*, Indianapolis, EUA, 159-164.

[36] Miller, D. L. e J. F. Pekny. (1991). "Exact Solution of Large Asymmetric Traveling Salesman Problems." *Science* **251**, 754-761.

[37] Moscato, P. (1989). On Evolution, Search, Optimization, Genetic Algorithms, and Martial Arts: Towards Memetic Algorithms. Technical Report, Caltech Concurrent Computation Program, C3P Report 826.

[38] Moscato, P. e M. G. Norman. (1992). "A Memetic Approach for the Traveling Salesman Problem. Implementation of a Computational Ecology for Combinatorial Optimization on Message-Passing Systems." In M. Valero, E. Onate, M. Jane, J. L. Larriba e B. Suarez (eds.), Parallel Computing and Transputer Applications 187-194. Amsterdam: IOS Press.

[39] Moscato, P. e F. Tinetti. (1992). "Blending Heuristics with a Population-Based Approach: A Memetic Algorithm for the Traveling Salesman Problem."CeTAD, *Report 92-12*. Universidad Nacional de La Plata, Argentina.

[40] Moscato, P. (1993). "An Introduction to Population Approaches for Optimization and Hierarchical Objective Functions: A Discussion on the Role of Tabu Search."*Annals of Operations Research* **41**, 85-121.

[41] Moscato, P. (1999). Memetic Algorithms: A Short Introduction. **In** D. Corne, M. Dorigo, and F. Glover (eds.), *New Ideas in Optimization*. McGraw-Hill.

[42] Nagata, Y. e S. Kobayashi. (1997). "Edge Assembly Crossover: A High-power Genetic Algorithm for the Traveling Salesman Problem."In *Proceedings of the Seventh International Conference on Genetic Algorithms*, East Lansing, EUA, 450-457.

[43] Noronha, T. F; Aloise, D. J e Silva, M. M. (2001). *Uma Abordagem sobre Estratégias Metaheurísticas*. http://www.sbc.org.br/reic/edicoes/2001e1.

[44] Rochat, Y. & Taillard, E. (1995). Probabilistic Diversification and Intensification in Local Search for Vehicle Routing. *Journal of Heuristics*, **1**(1), 147-167.

[45] Scholl, A.; Klein, R. & Domschke, W. (1998). Pattern Based Vocabulary Building for Effectively Sequencing Mixed-Model Assembly Lines. *Journal of Heuristics*, **4**(4), 359-381.

[46] Soares, W. K. da S. (2008). Heurísticas Usando Construção de Vocabulário Aplicadas ao Problema da Atribuição de Localidades a Anéis em Redes SONET/SDH. [S.l.]: Dissertação de Mestrado - Programa de Pós-Graduação em Engenharia de Produção, UFRN.

[47] Silva, A. C. G. (2008). Busca Heurística Através de Algoritmo Genético e Memético com Construção de Vocábulos para o Problema de Atribuição de Localidades a Anéis SONET. Dissertação de Mestrado - Programa de Pós-Graduação em engenharia de Produção, UFRN.

[48] Walters, T. (1998). "Repair and Brood Selection in the Traveling Salesman Problem." In A. E. Eiben, T. Bäch, M. Schoenauer, H. P. Schwefel (eds.), *Proceedings of the Fifth International Conference on Parallel Problem Solving from Nature*, Amsterdam, The Netherlands, 813-822.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo