UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE CIÊNCIAS EXATAS E DA TERRA - CCET
DEPARTAMENTO DE INFORMÁTICA E MATEMÁTICA APLICADA
PROGRAMA DE PÓS-GRADUAÇÃO EM SISTEMAS E COMPUTAÇÃO

Uma Análise Experimental de Abordagens Heurísticas Aplicadas ao Problema do Caixeiro Viajante

Álvaro Nunes Prestes

Dissertação de mestrado submetida ao Programa de Pós-Graduação em Sistemas e Computação da Universidade Federal do Rio Grande do Norte como parte dos requisitos para obtenção do grau de Mestre em Sistemas e Computação.

Natal/RN – Brasil

Julho de 2006

Álvaro Nunes Prestes

Uma Análise Experimental de Abordagens Heurísticas Aplicadas ao Problema do Caixeiro Viajante

Orientadora: Prof.ª Dr.ª Elizabeth Ferreira Gouvêa Goldbarg

Co-orientadora: Prof.ª Dr.ª Iloneide Carlos de Oliveira Ramos

Dissertação de mestrado submetida ao Programa de Pós-Graduação em Sistemas e Computação da Universidade Federal do Rio Grande do Norte como parte dos requisitos para obtenção do grau de Mestre em Sistemas e Computação.

Natal/RN - Brasil

Julho de 2006

Divisão de Serviços Técnicos

Catalogação da Publicação na Fonte. UFRN / Biblioteca Central Zila Mamede

Prestes, Álvaro Nunes Prestes.
Uma análise experimental de abordagens heurísticas aplicadas ao problema do caixeiro viajante / Álvaro Nunes Prestes. – Natal, RN, 2006.
84 f.

Orientadora : Elizabeth Ferreira Gouvêa Goldbarg.
Co-Orientadora : Iloneide Carlos de Oliveira Ramos.

Dissertação (Mestrado) – Universidade Federal do Rio Grande do Norte. Centro de Ciências Exatas e da Terra. Programa de Pós-Graduação em Sistemas de Computação.

1. Computação – Dissertação. 2. Otimização combinatória – Dissertação. 3. Caixeiro viajante - Dissertação. I. Goldbarg, Elizabeth Ferreira Gouvêa. II. Ramos, Iloneide Carlos de Oliveira. III. Título.

RN/UF/BCZM CDU 681.3(043.3)

Uma Análise Experimental de Abordagens Heurísticas Aplicadas ao Problema do Caixeiro Viajante

Álvaro Nunes Prestes

Dissertação de mestrado submetida ao Programa de Pós-Graduação em Sistemas e Computação da Universidade Federal do Rio Grande do Norte como parte dos requisitos para obtenção do grau de Mestre em Sistemas e Computação.

Composição da Banca Examinadora:

_____

Prof.ª Dr.ª Elizabeth Ferreira Golvêa Goldbarg

_____

Prof.ª Dr.ª Iloneide Carlos de Oliveira Ramos

_____

Prof. Dr. Marco César Goldbarg

_____

Prof.ª Dr.ª Carla Silva Oliveira

Natal/RN – Brasil

Julho de 2006

AGRADECIMENTOS

Agradeço àquele que agora não conhecemos totalmente e nem mesmo podemos vê-lo, mas que, apesar de sua grandeza, se manifesta a nós de maneira que podemos amá-lo como a um pai e ter a certeza de que existe e também nos ama. A Deus, senhor e criador de todas as coisas, pai amoroso, eu agradeço!

Agradeço o apoio fantástico de um pai que terminou o 2º grau (na época) por correspondência e de uma mãe que não pôde ao menos terminar a 4ª série, mas que me deram a melhor educação que um filho pode receber (e não falo somente de estudo) e fizeram de tudo para que eu tivesse as melhores possibilidades. Agradeço profundamente ao meu pai Argeu e minha mãe Jandira.

Aos meus irmãos e irmãs pela grande colaboração ao meu ingresso no mestrado e pela paciência que tiveram e que ainda precisarão ter. Muito obrigado a vocês!

Agradeço muito à minha noiva, Helena Carolina, por estar comigo quando muito precisei e por aguardar com tanta paciência o meu retorno. Pelo colo e pelos tantos carinhos, te agradeço. Obrigado amor!

Agradeço à família que tão amavelmente me recebeu e cuidou de mim durante vários meses! Obrigado Pr. Toinho e irmã Maria José. Ao Márcio também.

De uma maneira bem especial quero agradecer aos professores e/ou doutores Wei Pang, Thiago Machado, Heitor Lopes, Hung Nguyen, David Johnson, Iloneide Ramos e Givanaldo Souza que, atendendo gentilmente ao meu pedido, cederam o código fonte e/ou executável de seus algoritmos, de maneira a tornar possível a realização deste estudo. Agradeço ainda a Keld Helsgaun e à Concorde que disponibilizam esse mesmo material em seus respectivos *web sites*.

Meus sinceros agradecimentos à minha orientadora, Prof.ª Drª Elizabeth, pelo apoio compreensão. À Iloneide Ramos, pela grande ajuda.

Aos grandes amigos do LOCO/LAE que, de uma forma ou outra, colaboraram com o desenvolvimento desse estudo. Em especial ao Givanaldo pelo abrigo quando precisei.

A todos, muito obrigado!

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

Uma Análise Experimental de Abordagens Heurísticas Aplicadas ao Problema do Caixeiro Viajante

Autor: Álvaro Nunes Prestes

Orientadora: Prof.ª Dr.ª Elizabeth Ferreira Gouvêa Goldbarg

Co-orientadora: Prof.ª Dr.ª Iloneide Carlos de Oliveira Ramos

RESUMO

Devido à grande dificuldade de solução exata dos Problemas de Otimização Combinatória, vários métodos heurísticos têm sido desenvolvidos e durante muitos anos, a análise de desempenho dessas abordagens não foi realizada de maneira sistemática. A proposta deste trabalho é fazer uma análise estatística de abordagens heurísticas aplicadas ao Problema do Caixeiro Viajante. O foco da análise é avaliar o desempenho de cada abordagem em relação ao tempo computacional necessário até a obtenção da solução ótima para uma determinada instância do PCV. Para essa análise, foi utilizada uma metodologia estatística chamada Análise de Sobrevivência, auxiliada por métodos para o teste da hipótese de igualdade entre funções. Para uma melhor compreensão, as abordagens avaliadas foram divididas em três classes: Algoritmos Lin-Kernighan, Algoritmos Evolucionários e Algoritmos de Otimização por Nuvem de Partículas. Além das abordagens já existentes, foi incluído na análise, um algoritmo memético (para instâncias simétricas e assimétricas do PCV) que utiliza o algoritmo de Lin e Kernighan como procedimento de busca local.

Palavras-Chaves: Otimização, Problemas do Caixeiro Viajante, Análise Experimental, Heurísticas.

Uma Análise Experimental de Abordagens Heurísticas Aplicadas ao Problema do Caixeiro Viajante

Autor: Álvaro Nunes Prestes

Orientadora: Prof.ª Dr.ª Elizabeth Ferreira Gouvêa Goldbarg

Co-orientadora: Prof.ª Dr.ª Iloneide Carlos de Oliveira Ramos

ABSTRACT

Due to great difficulty of accurate solution of Combinatorial Optimization Problems, some heuristic methods have been developed and during many years, the analysis of performance of these approaches was not carried through in a systematic way. The proposal of this work is to make a statistical analysis of heuristic approaches to the Traveling Salesman Problem (TSP). The focus of the analysis is to evaluate the performance of each approach in relation to the necessary computational time until the attainment of the optimal solution for one determined instance of the TSP. Survival Analysis, assisted by methods for the hypothesis test of the equality between survival functions was used. The evaluated approaches were divided in three classes: Lin-Kernighan Algorithms, Evolutionary Algorithms and Particle Swarm Optimization. Beyond those approaches, it was enclosed in the analysis, a memetic algorithm (for symmetric and asymmetric TSP instances) that utilizes the Lin-Kernighan heuristics as its local search procedure.

Key words: Optimization, Traveling Salesman Problem, Experimental Analysis, Heuristics.

Capítulo 1

1. Introdução

Os Problemas de Otimização Combinatória (POC), devido à grande dificuldade de solução e sua presença em várias situações do cotidiano, têm atraído cada vez mais a atenção de pesquisadores de diversas áreas que têm envidado esforços para desenvolver algoritmos cada vez mais eficientes para serem aplicados a tais problemas. Os problemas de alocação, de roteamento e programação de horários são exemplos de situações onde POCs estão presentes.

Nestes problemas, o objetivo é atribuir valores a um conjunto de variáveis de decisão, de tal modo que uma determinada função objetivo seja otimizada (minimizada ou maximizada, dependendo do objetivo) atendendo um determinado conjunto de restrições. O Problema do Caixeiro Viajante, o Problema da Mochila, da Cobertura Mínima por Conjuntos, da Árvore Geradora Mínima, da Árvore de Steiner e do roteamento de veículos são exemplos de Problemas de Otimização Combinatória.

Uma forma de resolver tais problemas seria simplesmente enumerar todas as soluções possíveis e guardar aquela de menor custo. Entretanto, essa é uma idéia ingênua, pois para a maioria dos problemas, esse método torna-se impraticável, já que existe um elevado número de soluções possíveis. Mesmo que se utilize um supercomputador para resolver o problema, o tempo de processamento pode ser de várias horas, vários dias ou até anos. Portanto, técnicas computacionais mais apuradas são necessárias para resolver esses problemas.

Os Problemas de Otimização Combinatória têm sido utilizados como modelos para diversas situações reais, tanto em suas versões com um único objetivo a ser otimizado como em suas versões com múltiplos critérios. Grande parte desses problemas pertence à classe dos problemas NP-árduos, o que significa que dificilmente, algum dia, serão apresentados algoritmos polinomiais que os solucionem.

Ao longo dos anos, diversas técnicas foram desenvolvidas para a construção de algoritmos exatos que resolvessem tais problemas como: Programação Dinâmica (Saxe, 1980; Chiang, 1992), "Branch-and-Bound" (Balas e Toth, 1985; Diderich e Gengler, 1996; Tschöke *et al.,* 1995) , "Branch-and-cut" (Brunetta et al., 1997; Padberg e Rinaldi, 1991; Padberg e Rinaldi, 1987; Crowder e Padberg, 1980), etc. Entretanto, como os tempos para solucionar instâncias de grande porte por algoritmos exatos são inviáveis, a opção, nesses casos, seria o uso de algoritmos heurísticos. As heurísticas são algoritmos que não têm garantia de encontrar a solução ótima, ou seja, a melhor solução existente para o problema. No entanto, são capazes de retornar uma solução em um tempo adequado para as necessidades da aplicação.

Existem diversas abordagens para a construção de algoritmos heurísticos. Inicialmente, os pesquisadores utilizavam técnicas ditas gulosas ou míopes no desenvolvimento de tais algoritmos. Com o passar dos anos, técnicas mais gerais, conhecidas como metaheurísticas, foram apresentadas. Estas técnicas, por meio de adaptações, podem ser usadas para vários problemas (Michalewicz e Fogel, 2000).

As Metaheurísticas são paradigmas de desenvolvimento de algoritmos heurísticos. Uma metaheurística é definida por uma estrutura genérica que deve ser seguida, adaptando suas partes de acordo com o problema que se pretende resolver. Diversas propostas de metaheurísticas surgiram nos últimos anos.

Nas últimas décadas, diversos *softwares* vêm sendo desenvolvidos para um grande número de aplicações onde são utilizados algoritmos de otimização. Em muitos casos, existe um tempo limitado para que o algoritmo embutido no sistema ou software encontre uma solução de "boa" qualidade. Desse modo, é importante que se possa avaliar a probabilidade de geração de uma solução de qualidade para problemas de otimização.

Os métodos de análise mais comuns são a análise do pior caso e a análise do comportamento médio dos algoritmos. Tais técnicas, desenvolvidas pelos teóricos da Ciência da Computação, são amplamente utilizadas. Entretanto, não são suficientes para uma análise real do desempenho dos algoritmos baseados em metaheurísticas. Alguns trabalhos recentes que abordam o tema da análise experimental para algoritmos heurísticos foram apresentados por Rardin e Uzsoy (2001), Johnson e McGeoch (2002) e Johnson (2002).

Durante muitos anos, a análise de desempenho de algoritmos foi realizada em relação à qualidade média das soluções e muitas vezes, sem o auxílio de algum método estatístico que pudesse validar os experimentos. A proposta deste trabalho é fazer uma análise experimental de abordagens heurísticas utilizando a técnica estatística conhecida como Análise de Sobrevivência. Esta técnica é uma ferramenta muito utilizada na Medicina e na Engenharia, mas também encontra aplicações em outras áreas tais como as Ciências Sociais e Econômicas. A Análise de Sobrevivência estuda o tempo decorrido até a ocorrência de um determinado evento. Neste trabalho, é investigada a utilização desta técnica na análise do tempo de processamento de algoritmos, onde, especificamente, o evento considerado, é a obtenção de uma solução com uma "certa" qualidade.

O estudo é feito com base no Problema do Caixeiro Viajante (PCV), um problema que tem servido de plataforma de teste para a investigação de diversas idéias algorítmicas, porque, além de ser uma problema de larga aplicabilidade no mundo real, é de fácil compreensão e descrição, mas de difícil solução, pois pertence à classe de problemas NP-Árduos. Por esses motivos, vários métodos têm sido desenvolvidos com o propósito de se resolver instâncias cada vez maiores desse problema e em menos tempo. Neste trabalho, são comparadas várias abordagens heurísticas aplicadas ao PCV com o objetivo de se avaliar o desempenho de cada uma, em relação ao tempo computacional necessário até a obtenção da solução ótima para uma determinada instância.

Além das abordagens já existentes, foi desenvolvido e incluído na análise realizada, um algoritmo memético (pertencente à classe de algoritmos evolucionários) que utiliza o algoritmo de Lin e Kernighan para o refinamento da busca por melhores soluções. O algoritmo desenvolvido se aplica tanto ao PCV simétrico quanto ao assimétrico, sendo que, para este último, foi utilizado um método para a transformação de instâncias assimétricas em instâncias simétricas. Essa transformação foi necessária porque o algoritmo de refinamento utilizado não suporta instâncias assimétricas.

Para tanto, o trabalho fica organizado da forma que segue. No Capítulo 2 é feita uma introdução ao Problema do Caixeiro Viajante (PCV) apresentando suas principais características, bem como um histórico de instâncias conhecidas que foram resolvidas ao longo dos anos e aquelas que, ainda hoje, são um grande desafio. São mostradas, também,

algumas das várias aplicações do PCV no mundo real que proporcionam grandes benefícios à sociedade como um todo.

O Capítulo 3 apresenta alguns métodos aplicados, tanto exatos como heurísticos, aplicados na solução do PCV e que têm recebido muita atenção nos últimos anos.

Uma introdução à Análise de Sobrevivência é apresentada no Capítulo 4. No Capítulo 5, o algoritmo memético desenvolvido é apresentado, bem como os experimentos computacionais para a definição das melhores configurações e versões.

A análise experimental utilizando a Análise de Sobrevivência é descrita no Capítulo 6. Para uma melhor organização e apresentação do estudo realizado, as heurísticas utilizadas foram classificadas em três diferentes classes, de acordo com a característica de cada uma. A primeira classe é composta por variações do algoritmo de Lin e Kernighan (1973). A segunda classe é composta por Algoritmos Evolucionários e a terceira, por algoritmos de Otimização por Nuvem de Partículas. Finalmente, no Capítulo 7, são apresentadas as considerações finais a cerca do estudo realizado.

Capítulo 2

2. O Problema do Caixeiro Viajante

O Problema do Caixeiro Viajante (PCV) tem sido muito utilizado no experimento de diversos métodos de otimização por ser, principalmente, um problema de fácil descrição e compreensão, grande dificuldade de solução, uma vez que é NP-Árduo (Karp, 1975), e larga aplicabilidade.

Sendo um grafo G = (N,E) onde N = {1,...,*n*} é o conjunto de nós e E = {1,...,*m*} é o conjunto de arestas de G, e custos, c_{ij}, associados com cada aresta ligando os vértices *i* e *j*, o problema consiste em localizar o menor ciclo Hamiltoniano do grafo G. O tamanho do ciclo é calculado pelo somatório dos custos das arestas que formam o ciclo. Os nós do grafo são, freqüentemente, referenciados como "cidades" e, em outras palavras, o objetivo é visitar todas as cidades passando apenas uma vez por cada cidade, retornando ao ponto de origem. Este percurso deve ser feito de forma a minimizar a distância total percorrida.

O PCV pode ainda ser classificado como simétrico ou assimétrico. É simétrico se, para todos os pares de nós {*i*,*j*}, os custos c_{ij} e c_{ji} forem iguais. Caso contrário, o mesmo é dito assimétrico. Uma revisão histórica e mais completa sobre o PCV pode ser encontrada em Hoffman e Wolfe (1985) e ainda em Gutin e Punnen (2002).

Em 1990, Reinelt (1990) criou uma uma biblioteca contendo várias instâncias do PCV que vinham sendo (e são até hoje) testadas e discutidas na literatura. Essa biblioteca é conhecida como TSPLIB e contém mais de 100 exemplos com tamanhos que variam de 14 até 85.900 cidades.

Desde abril de 2004, restam apenas três instâncias na biblioteca TSPLIB que ainda não foram resolvidas. Essas instâncias são d18512, pla33810 e pla85900 que têm 18.512, 33.810 e 85.900 cidades, respectivamente. A Tabela 1 mostra os melhores resultados já obtidos para estas instâncias destacando o custo do melhor percurso já encontrado, o melhor limite inferior e a taxa de afastamento entre o custo e o limite (Concorde, 2006).

Tabela 1: Melhores resultados para as instâncias ainda não resolvidas

Instância	Tamanho do melhor Percurso	Melhor limite	Taxa de afastamento entre melhor percurso e o melhor limite
d18512	645238	645230	0.0013%
pla33810	66050499	66041848	0.013%
pla85900	142382641	142348737	0.024%

Os limites inferiores apresentados foram obtidos utilizando uma fusão do algoritmo LK Concorde (Concorde, 2006) com um algoritmo *branch-and-cut* com busca restrita implementado por Dantzig, Fulkerson e Johnson (Dantzig *et al.*, 1954; 2006). Para a instância d18512 o melhor percurso foi obtido por Tamaki e Tokuyama (2003) usando uma abordagem chamada *tour-merging* combinada ao algoritmo de Lin-Kernighan (Lin e Kernighan, 1973)

implementado por Helsgaun (2000). Os percursos para as instâncias pla33810 e pla85900 foram obtidos por Helsgaun usando uma versão melhorada de seu algoritmo.

Tabela 2: Histórico da solução de instâncias do PCV

Ano	Instância	Tamanho	Pesquisadores
1954	dantzig42	49	Dantzig, Fulkerson e Johnson (1954)
1962	aleatória	64	Held e Karp (1962)
1974	aleatória	67	Camerini, Fratta e Maffioli (1974)
1980	gr120	120	Grötschel (1980)
1980	lin318	318	Crowder e Padberg (1980)
1987	att532	532	Padberg e Rinaldi (1987)
1991	gr666	666	Grötschel e Holland (1991)
1991	pr2392	2392	Padberg e Rinaldi (1991)
1995	pla7397	7.397	Applegate, Bixby, Chvàtal e Cook (1995)
1998	usa13509	13.509	Applegate, Bixby, Chvàtal e Cook (1998)
2001	d15112	15.112	Applegate, Bixby, Chvàtal e Cook (2001)
2004	sw24798	24.978	Applegate, Bixby, Chvàtal, Cook e Helsgaun (2004)

Apesar das instâncias ainda não solucionadas, é notável o avanço que se tem alcançado na solução do PCV nos últimos anos. A Tabela 2 mostra a evolução na solução de instâncias do PCV a partir de 1954 quando Dantzig, Fulkerson e Johnson resolveram uma instância com 42 cidades até 2004 quando Applegate, Bixby, Chvátal, Cook e Helsgaun (2004) solucionaram uma instância com 24.978 e provaram que não existe um percurso menor. Essa instância é, atualmente, a maior instância do Problema do Caixeiro Viajante (PCV) já resolvida. A Figura 1 mostra a instância sw24978 sobre o mapa da Suécia e o percurso ótimo encontrado.

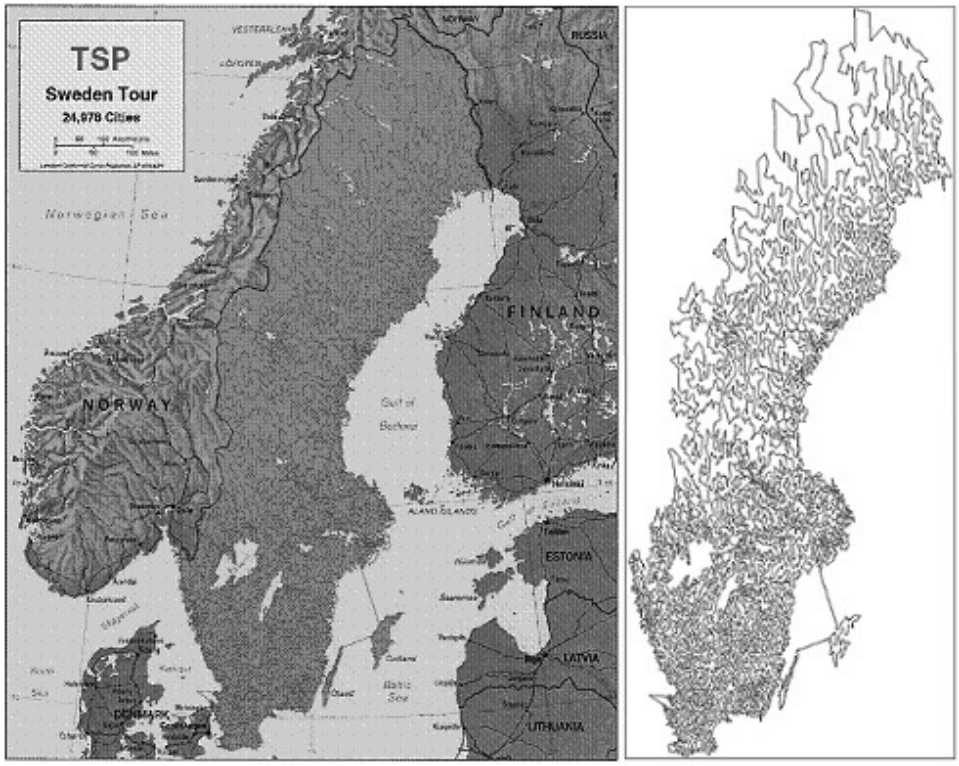

Figura 1: Instância sw24798 - mapa da Suécia à esquerda e o melhor percurso à direita

Uma das aplicações clássicas do PCV é na confecção de Placas de Circuito Impresso (PCB), na tarefa de perfuração. Uma placa de circuito impresso pode conter centenas ou milhares de furos, que são feitos para a soldagem dos componentes eletrônicos. Além disso, os furos podem ser de tamanhos diferentes. Para otimizar o processo, deve-se perfurar todos os furos de mesmo diâmetro por vez, já que a troca da ferramenta é um processo relativamente lento. Assim, a perfuração pode ser vista como uma seqüência de problemas do Caixeiro Viajante. Percorrendo-se o melhor caminho possível, economiza-se tempo, aumentando a produtividade do processo (Reinelt, 1994).

Além da aplicação na perfuração de placas de circuitos, pode-se ainda citar a aplicação na análise da estrutura de cristais (Bland e Shallcross, 1987), rede de gás, manipulação de materiais em um armazém (Ratliff e Rosenthal, 1981), seqüenciamento de tarefas, na definição de planos de rotas (Hoffman e Padberg, 2005), fabricação de *chips* (Korte, 1989), mapeamento de genoma (Guyon *et al*., 2003), seqüenciamento de DNA (Gonnet *et al*., 2000), dentre outras.

Capítulo 3

3. Métodos para a solução do PCV

Dada a importância do Problema do Caixeiro Viajante frente à sua larga aplicabilidade em situações reais, várias abordagens têm sido desenvolvidas com o propósito de se resolver um conjunto cada vez maior de instâncias desse problema. As abordagens aplicadas ao PCV podem ser divididas em, pelo menos, duas grandes categorias: Métodos Exatos e Métodos Heurísticos. Nesta Seção, são apresentados alguns dos principais métodos, tanto exatos como heurísticos, aplicados ao PCV.

3.1. Métodos Exatos

Uma forma natural e intuitiva para obter soluções ótimas para instâncias do PCV seria testar todas as possibilidades, mas se sabe que, para problemas de otimização combinatória como é o PCV, um problema NP-Árduo, esta abordagem é impraticável em função do crescimento exponencial das possibilidades em relação ao tamanho do problema (número de cidades). No entanto, existem outras estratégias baseadas na Programação Inteira que podem garantir a obtenção de uma solução ótima. Tais algoritmos, ditos exatos, garantem a obtenção da solução ótima e provam a otimalidade da solução obtida num tempo de execução finito ou provam que uma solução viável não existe (Dumitrescu e Stützle , 2003). Como exemplos de métodos exatos baseados em Programação Inteira, podem-se citar *Branch & Bound*, *Branch & Cut*, *Branch & Price*, Relaxação Lagrangeana e Programação Dinâmica.

Dumitrescu e Stützle (2003) destacam algumas vantagens e desvantagens relacionadas à aplicação de algoritmos exatos na solução de problemas de otimização combinatória. Uma vantagem importante dos algoritmos exatos que usam Programação Inteira está no fato de que se pode provar que soluções ótimas podem ser obtidas se o algoritmo tiver sucesso na execução. Além disso, informações valiosas sobre os limites inferior e superior em relação a solução ótima são obtidos, mesmo se o algoritmo for terminado antes de completar sua execução. Nesse ponto, métodos de Programação Inteira podem ser utilizados como abordagens aproximativas caso seja definido um critério de parada que anteceda a conclusão do algoritmo. Outra vantagem dos métodos de Programação Inteira é que estes permitem que sejam partes do espaço de busca em que a solução ótima não pode ser encontrada sejam desconsideradas.

Por outro lado, uma das maiores desvantagens dos métodos exatos está no fato de que, para muitos problemas, o tamanho das instâncias que podem ser solucionadas é limitado pelo custo computacional muito elevado onde o tempo de processamento, em função do número de possibilidades, cresce grandemente com o tamanho da instância. Outra dificuldade é em relação ao consumo de memória que pode ser muito alto levando ao término prematuro do programa (Dumitrescu e Stützle, 2003).

De uma forma geral, os algoritmos exatos compreendem os algoritmos que utilizam a abordagem de *branch-and-cut* que por sua vez é um algoritmo *branch-and-bound* no qual planos de cortes são gerados ao longo da árvore de busca (Merz, 2000). A mudança

provocada por essa filosofia está no fato de que a busca por soluções rápidas em cada nó é substituída pela procura por limites mais apertados.

Nos últimos anos um grande esforço tem sido devotado a métodos do tipo *Branch-and-Cut*, *Branch-and-Price* e *Relax-and-Cut* na resolução de problemas de otimização combinatória. Hoffman e Padberg (1985), Nemhauser e Wolsey (1988) e Wolsey (1998) apresentam um visão geral destas abordagens. Outras abordagens exatas aplicadas ao PCV podem ser encontradas em Laporte (1992).

3.1.1. "Branch & Bound" e "Branch & Cut"

O *Branch & Bound* é um algoritmo exato que pode ser aplicado ao PCV e utiliza métodos para a definição de limites superiores e inferiores e um esquema de enumeração. Considerando um problema de minimização, os limites superiores são, geralmente, obtidos por heurísticas eficientes que produzem soluções de boa qualidade num tempo relativamente curto. Os limites inferiores são obtidos pela relaxação do problema em questão, removendo uma ou mais restrições. O esquema de enumeração atua dividindo, em cada etapa, o problema maior em vários sub-problemas (problemas menores mais fáceis de serem resolvidos) de tal forma que a junção das soluções viáveis dos sub-problemas resultem numa solução viável para o problema original. Os sub-problemas são, por sua vez, divididos em novos sub-problemas até que sejam resolvidos, ou seja, até que o limite inferior seja igual ao superior ou o limite inferior seja maior que a melhor solução encontrada até o momento. Dessa forma, essa abordagem produz uma ramificação (*branching*) na qual cada nó corresponde a um problema e os nós descendentes, os sub-problemas (Merz, 2000).

O grande número de restrições muitas vezes impede que um problema seja tratado convenientemente utilizando-se apenas ferramentas de programação linear. Dessa forma, se uma solução ótima associada à relaxação linear é inviável, um novo problema de separação deve ser "resolvido" buscando a identificação de uma ou mais restrições violadas pela relaxação corrente (*cutting procedure*). O novo problema obtido (com restrições adicionais) é novamente resolvido via programação linear e o processo é repetido até que novas classes de desigualdades violadas não sejam mais encontradas. Em outras palavras, a idéia básica do *Branch & Cut* é encontrar uma relaxação na forma de um programa linear com a mesma solução ótima do problema original (Merz, 2000).

O *Branch & Cut* tem sido aplicado com sucesso em vários problemas de otimização combinatória tais como o PCV (Fischetti *et al.*, 1997), Steiner (Lucena e Beasley, 1998) e particionamentos de grafos (Brunetta *et al.,* 1997). Especialmente para o PCV, muitos progressos foram feitos nos últimos 30 anos. Padberg e Rinaldi (1991) resolveram o problema pr2392 da TSPLIB (Reinelt, 1990) em 4,3 horas (Rodrigues, 2000) usando um procedimento *branch-and-bound* auxiliado por um bom equipamento de hardware e um algoritmo bastante complexo mas eficiente. Tempos depois, Applegate *et al.* (1995) determinaram o valor ótimo para diversos problemas da TSPLIB com tamanhos variando entre 225 a 7397 cidades. Para isso utilizaram uma rede de estações de trabalho UNIX e um procedimento que combina diversos algoritmos, inclusive o de Padberg e Rinaldi (1991).

A Tabela 3 apresenta um histórico de instâncias que foram resolvidas de maneira exata por algoritmos *Branch & Cut* (ou que utilizam o *Branch & Cut*) destacando o ano, o tamanho da instância (dimensão) e os pesquisadores.

Tabela 3: Histórico das instâncias resolvidas com métodos baseados em *Branch & Cut*

Ano	Dimensão	Pesquisadores
1954	49	Dantzig, Fulkerson e Johnson (1954)
1977	120	Grötschel (1980)
1980	318	Crowder e Padberg (1980)
1987	532	Padberg e Rinaldi (1987)
1991	666	Grötschel e Holland (1991)
1991	2392	Padberg e Rinaldi (1991)
1992	3038	Applegate, Bixby, Chvàtal e Cook (1995)
1993	4461	Applegate, Bixby, Chvàtal e Cook (1995)
1994	7397	Applegate, Bixby, Chvàtal e Cook (1995)
1998	13509	Applegate, Bixby, Chvàtal e Cook (1998)
2001	15112	Applegate, Bixby, Chvàtal e Cook (2001)
2004	24.978	Applegate, Bixby, Chvàtal, Cook e Helsgaun (2004)

Não obstante os avanços que se tem conseguido no desenvolvimento de métodos exatos para problemas de otimização combinatória, existem problemas práticos que apresentam grandeza na ordem de dezenas de milhares de vértices, ou seja, muito acima dos limites que se tem atualmente.

Exemplos de algoritmos *Branch & Bound* e *Branch & Cut* aplicados ao Problema do Caixeiro Viajante podem ainda ser encontrados em Crowder e Padberg (1980), Balas e Toth (1985), Jfinger, Reinelt e Thienel (1994), Tschöke, Lübling e Monien (1995), Diderich e Gengler (1996), Thienel (1996) e Applegate *et al.* (1998).

3.2. Métodos Heurísticos

Tendo em vista a grande dificuldade na solução exata do Problema do Caixeiro Viajante, várias abordagens heurísticas têm sido aplicadas proporcionando soluções aproximadas de boa qualidade.

Um exemplo de abordagem heurística aplicada ao PCV é a Busca Tabu que foi introduzida por Hansen (1986) e Glover (1990). A idéia geral é ter um mecanismo para auxiliar o processo de busca, evitando que o mesmo espaço de soluções seja percorrido repetidas vezes, ou seja, evitando a realização de movimentos feitos recentemente. Essa característica permite uma melhor exploração do espaço de soluções além de auxiliar na fuga de ótimos locais. Exemplos de algoritmos Busca Tabu aplicados ao Problema do Caixeiro Viajante podem ser encontrados em Moscato (1993), Battiti e Tecchiolli (1994) e Zachariasen e Dam (1995).

O *Simulated Annealing* (SA) é outro método heurístico que também tem sido aplicado ao PCV. O SA foi introduzido por Kirkpatrick *et al.* (1983), com base no trabalho de Metropolis *et al.* (1953) que apresentou um algoritmo para simular o resfriamento de um material em um banho térmico (um processo conhecido como *annealing* ou têmpera). Um tipo de tratamento térmico consiste em submeter certos materiais inicialmente a altas temperaturas e reduzir gradualmente até atingir o equilíbrio térmico com aumentos e reduções do estado de energia, tornando-os consistentes e rígidos. No modelo computacional, o processo de diversificação acontece quando a temperatura está alta, fazendo com que o

algoritmo explore um maior espaço de soluções. À medida que a temperatura baixa, a aceitação de soluções piores vai reduzindo, intensificando a busca, convergindo para soluções que melhorem a função de custo. Um exemplo de algoritmo SA aplicado ao PCV pode ser encontrado em Souza (2005).

Outro exemplo de algoritmo heurístico é *Greedy Randomized Adaptive Search Procedures* (GRASP) que foi introduzido por Feo e Resende (1995) e é uma abordagem *multistart* que combina uma fase de construção com uma fase de busca local. Na primeira fase a solução é construída passo a passo e na fase de melhoramento da solução, é utilizado um procedimento de busca local. O GRASP utiliza ainda uma lista de candidatos (*RCL - restricted candidate list*) formada pelos melhores elementos que poderão compor a solução. As fases de construção e melhoramento são repetidas até que o critério de parada seja atendido. Um estudo mais amplo sobre esta abordagem pode ser encontrado no trabalho de Pitsoulis e Resende (2002). Além desse, exemplos de algoritmos GRASP aplicados ao Problema do Caixeiro Viajante podem ser encontrado em Silva *et al.* (2000) e Farias (2004).

Neste trabalho, algumas abordagens foram selecionadas para a realização de uma análise experimental. Tais abordagens foram escolhidas porque representam, atualmente, as heurísticas aplicadas ao PCV que mais têm recebido atenção, em função da qualidade das soluções que têm proporcionado. Essas heurísticas são apresentadas em mais detalhes nos tópicos seguintes.

3.2.1. O Algoritmo de Lin e Kernighan

O algoritmo proposto por Lin e Kernighan (1973) é uma das mais eficientes abordagens para o PCV simétrico e tem sido objeto de estudo para vários pesquisadores ao longo dos anos, podendo ser considerado um caso especial de Busca Local.

Algoritmos de busca local se baseiam em modificações simples no percurso, realizando trocas entre as cidades deste percurso, com o objetivo de reduzir o comprimento do mesmo (minimizar a distância do percurso pelas *n* cidades). Quando essa redução não é mais possível, tem-se um circuito localmente ótimo. Os algoritmos de Busca Local trabalham com a idéia de *vizinhança*: sendo *S* o espaço de busca e *f* a função-objetivo a ser minimizada, a vizinhança de s_0 (solução inicial) corresponde ao conjunto $N(s_0) \subseteq S$ que reúne um número determinado de soluções *s*. Cada solução $s \in N(s_0)$ é chamada de *vizinho* de s_0 e é obtido a partir de uma operação chamada de *movimento*.

De uma forma geral, uma busca local parte de uma solução inicial s_0 e segue navegando pelo espaço de busca através de movimentos. Os movimentos permitem *pular* de uma solução para outra que seja sua vizinha. Em geral, os movimentos são realizados quando a solução vizinha melhora a função objetivo. Os algoritmos de busca local terminam em uma solução que corresponde a um ótimo local, mas não há garantia de que este seja o ótimo global. A Figura 2 mostra a estrutura básica de um algoritmo de Busca Local.

```
Selecione uma solução inicial s0 ∈ S
Repita
        Selecione s ∈ N(s0) tal que f(s) < f(s0)
        s0 ← s
Até que f(s) > f(s0) para todo s ∈ N(s0)
```

Figura 2: Esquema geral do algoritmo de busca local

Para ilustrar o funcionamento desses algoritmos, considere a estrutura de vizinhança 2-opt que é uma das estruturas mais conhecidas na literatura proposta por Croes (1958). Esse movimento apaga duas arestas, quebrando o percurso em dois caminhos, e então reconecta esses caminhos de uma outra forma, alterando o percurso. Este algoritmo é aplicado, para uma dada solução corrente, realizando todas as 2-trocas possíveis. Desta forma, cada movimento corresponde a uma solução vizinha e a aquele que resultar no menor percurso, em cada iteração, é tida como a nova solução corrente. A Figura 3 mostra soluções vizinhas geradas a partir de 2-trocas sobre uma solução inicial. Ao término da execução, tem-se uma solução localmente ótima, ou seja, o percurso não pode mais ser melhorado a partir da troca de apenas duas arestas.

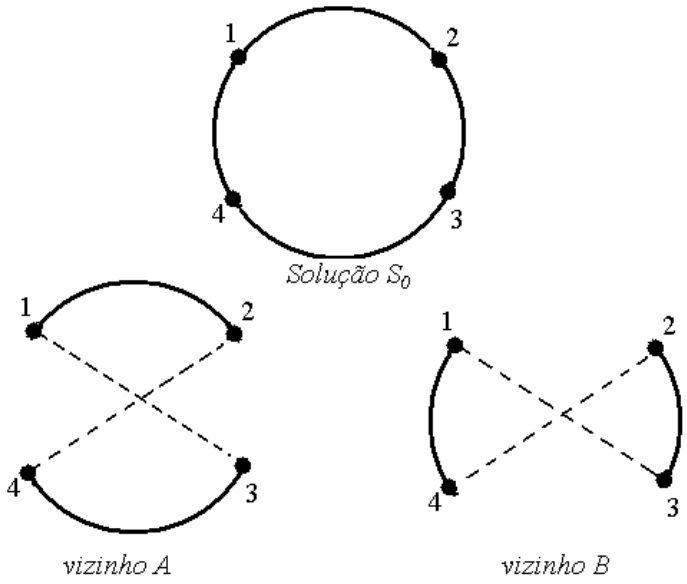

Figura 3: Estrutura de Vizinhança 2-opt

Como pode ser observado na Figura 3, uma solução vizinha da solução corrente é gerada a partir da remoção das arestas (1,4) e (2,3) e introdução das arestas (1,3) e (2,4), *vizinho A*; e outra solução vizinha é gerada a partir da remoção das arestas (1,2) e (3,4) e introdução das arestas (1,3) e (2,4).

Existem várias estruturas de vizinhança e escolher a adequada é fundamental sendo, ao mesmo tempo, uma das maiores dificuldades nos algoritmo de busca local. Essa dificuldade se deve ao fato de que problemas diferentes possuem características diferentes e, desta forma, o uso de uma determinada estrutura de vizinhança pode resultar em bons resultados para um problema P, mas pode não apresentar a mesma eficiência para um problema P'.

Originalmente, o algoritmo Lin-Kernighan (LK) se baseia na troca de arestas que tem como objetivo transformar um determinado percurso num percurso de menor custo, assim como no 2-opt. No entanto, o LK utiliza uma estrutura de vizinhança *k*-opt que é uma generalização do algoritmo 2-opt, ou seja, o número de arcos trocados em cada passo é variável para diferentes valores de *k*.

Uma das dificuldades da abordagem *k*-opt apontadas por Helsgaun (2000) é que o número de operações para testar todas as *k* trocas aumenta rapidamente com o aumento do

número de cidades. Além disso, é difícil saber que valor de k usar para se ter um melhor custo benefício entre o tempo de processamento e a qualidade da solução. Para reduzir essa dificuldade, Lin e Kernighan introduziram um mecanismo que, em cada iteração, busca o melhor valor para k verificando as trocas que podem resultar num percurso de menor custo. O algoritmo LK pode ser descrito como segue.

Passo 1	Gere um circuito T inicial
Passo 2	Faça $G^*= 0$ onde G^* é o melhor ganho alcançado até o momento. Escolha qualquer nó t_1 e seja x_1 um dos arcos de T adjacentes a t_1. Faça $i=1$.
Passo 3	Do outro ponto final t_2 de x_1, escolha y_1 para t_3 com $g_1 = \|x_1\|-\|y_1\| > 0$. Se não existe y_1, vá para o passo 6(d).
Passo 4	Faça $i= i+1$. Escolha x_i (que no momento liga t_{2i-1} *a* t_{2i} e y_i) como segue: a) x_i é escolhido de modo que, se t_{2i} é ligado com t_1, a configuração resultante é um circuito. b) y_i é algum arco disponível no ponto final t_{2i} compartilhado com x_i, sujeito a (c), (d) e (e). Se não existir y_i, vá para o passo 5. c) Para garantir que os x's e y's são disjuntos, x_i não pode ser um arco previamente introduzido (isto é, um y_j, $j < i$), e similarmente y_i não pode ser um arco previamente removido. d) $G_i=\sum_{j=1}^{i} g_j=\sum_{j=i}^{i}(\|x_j\|-\|y_j\|)>0$ e) Em seqüência para garantir que o critério de viabilidade de (a) possa ser satisfeito para $i+1$, o y_i escolhido deve permitir a remoção de x_{i+1}. f) Antes de y_i ser construído, deve-se verificar se fechando a ligação de t_{2i} para t_1 será observado ganho maior que o melhor alcançado anteriormente. Faça y_i^* ser um arco conectando t_{2i} a t_1 e faça $g_i^* = \|y_i^*\| - \|x_i\|$. Se $G_{i-1} + g_i > G^*$ faça $G^* = G_{i-1} + g_i$ e $k = i$.
Passo 5	Terminada a construção de x_i e y_i dentro dos passos de 2 até o 4 quando não houver mais arcos satisfazendo a os critérios de 4(c) a 4(e), ou quando $G_i \leq G^*$. Se $G^* > 0$ então considere o novo circuito T' faça $f(T') = f(T) - G^*$ e $T \leftarrow T'$ e vá para o passo 2.
Passo 6	Se $G^*=0$, um recuo limitado é realizado como segue: a) Repita os passos 4 e 5 escolhendo y_2's na ordem crescente do comprimento. Contanto que o critério de ganho $g_1 + g_2 > 0$ seja satisfeito. b) Se todas as escolhas de y_2 no passo 4(b) são exauridas sem ganho, retorne ao passo e tente escolher outro x_2. c) Se isto também falha, um passo atrás no passo 3 é promovido, aonde os y_i's são examinados na ordem do aumento do comprimento. d) Se os y_i's são também exauridos sem ganho, tenta-se alternar x_1 no passo 2. e) Se isto também falha, um novo t_1 é selecionado, e repete-se o passo 2.
Passo 7	O procedimento termina quando todos os n valores de t_1 tiverem sido examinados sem ganho.

Figura 4: Esquema geral do algoritmo LK (Lin e Kernighan, 1973).

A Figura 4 apresenta o funcionamento do algoritmo LK que considera, em cada passo, um conjunto crescente de possíveis trocas (iniciando com k=2). Opcionalmente, este processo pode ser repetido várias vezes tendo como percurso inicial T um percurso gerado

aleatoriamente ou mesmo por algum outro método. A Figura 5 mostra, ainda, um exemplo das trocas de arestas do algoritmo LK onde o percurso é representado como um círculo.

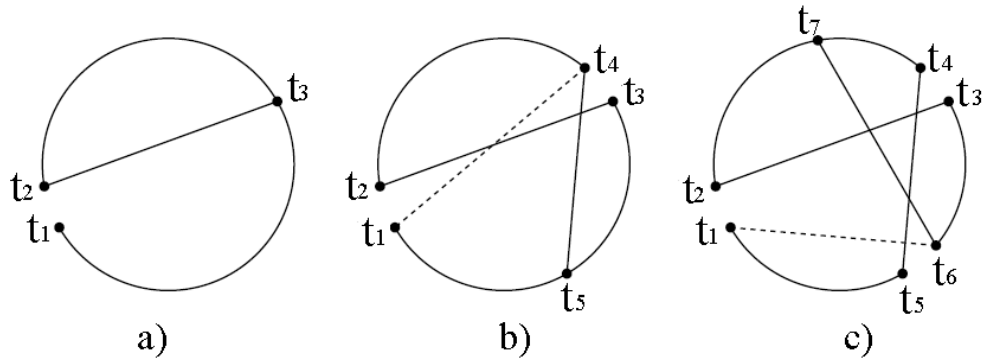

Figura 5: Movimento do LK em três níveis

Em cada passo, tem-se uma situação em que o percurso é quebrado formando um caminho equivalente a uma árvore geradora com 1 nó extra (1-tree) como mostra a Figura 5a. Esta árvore pode ser, facilmente, transformada numa solução viável do PCV pela quebra no nó de grau 3 (t_3) e da conexão de dois nós de grau 1 (Merz, 2000). Buscando melhorar o percurso a partir da cidade t_1, o algoritmo LK, primeiramente substitui a aresta (t_1,t_2) por uma aresta mais curta (t_2,t_3). Neste momento, o algoritmo considera o percurso completo pela conexão do predecessor de t_3, chamado t_4, com t_1 e então substitui a aresta (t_3,t_4) por (t_4,t_1). Neste caso, as arestas (t_1,t_2) e (t_4,t_3) são substituídas por (t_2,t_3) e (t_4,t_1). Uma outra opção seria substituir (t_3,t_4) por (t_4,t_5) que resultaria numa nova árvore 1-tree. Novamente, o percurso pode ser fechado pela conexão das cidades t_6 e t_1 ou continuar a busca conectando t_6 com t_7 como mostra a Figura 5c.

Como pode ser observado, o algoritmo LK realiza trocas seqüenciais de arestas até não encontrar mais trocas que possibilitem ou favoreçam a obtenção de *k*-trocas na iteração. Se a busca por melhores *k*-trocas falhar, vários níveis de *backtracking* são considerados como, por exemplo, alternativas para (t_2,t_3), no primeiro nível, e alternativas para (t_4,t_5), no segundo nível.

Lin e Kernighan, objetivando a melhoria da performance de seu algoritmo, definiram algumas regras que o algoritmo deve seguir. As regras de 1 a 6 foram introduzidas para limitar a busca do algoritmo, restringindo a escolha das arestas e as regras 7 e 8 tem como objetivo guiar o processo de busca:

1. Somente trocas seqüenciais são permitidas
2. O ganho parcial G_i deve ser positivo
3. O percurso deve resultar num ciclo
4. Uma aresta previamente excluída não pode ser adicionada, e uma aresta previamente adicionada não pode ser excluída
5. A busca é encerrada se o percurso encontrado é igual ao encontrado anteriormente
6. A busca por uma aresta a ser adicionada, $y_i = (t_{2i}, t_{2i+1})$, é restrita aos cinco vizinhos mais próximos do ponto atual (x_i)

7. Na escolha da aresta y_i (com i ≥ 2) a cada aresta é dada a prioridade $c(x_{i+1}) - c(y_i)$.
8. Se existem duas alternativas para x_i, é escolhida a que tiver o maior c(x_i).

A partir da idéia original de Lin e Kernighan, várias outras implementações e/ou melhorias para este algoritmo foram desenvolvidas. Nesse aspecto, Fredman *et al.* (1995) apresentam e comparam diversas estruturas para a representação de rotas, argumentando que um tempo considerável é desperdiçado no processo de inversões de segmentos presentes em algoritmos de troca de arcos, quando utilizada a representação de rotas através de vetores. As comparações realizadas por Fredman *et al.* mostram que o algoritmo LK utilizando estruturas diferentes da representação por vetores, podem ter um desempenho superior. Esta vantagem se torna ainda mais significativa à medida que o número de cidades aumenta. Ainda em relação à estrutura de dados, Helsgaun (2000) apresenta algumas discussões sobre diferentes estruturas utilizadas para a melhoria do desempenho do LK.

Applegate *et al.* (2000) apresentam um estudo comparativo em relação à utilização de métodos mais elaborados para a geração de uma solução inicial para o LK e Gamboa *et al.* (2005) apresentam uma breve revisão sobre as características de várias implementações do LK, tanto para o PCV simétrico quanto assimétrico. Várias abordagens que modificam e/ou utilizam o LK podem ser encontradas em DIMACS (2006).

3.2.1.1. Algoritmo Iterated Lin-Kernighan

Dentre as diversas variações do algoritmo LK, sem dúvida, a versão *chained* ou *Iterated* (iterativa), introduzida originalmente por Martin *et al.* (1991) e posteriormente modificada por outros autores, é a mais significativa.

A idéia geral do *Iterated* LK é realizar uma reinicialização rápida do algoritmo, sem a necessidade da repetição do processo de leitura do problema. O procedimento de busca é reaplicado em uma solução gerada através de uma perturbação da melhor solução obtida na iteração anterior. A principal característica da versão *Iterated* é, então, a utilização de um mecanismo de perturbação para alterar um percurso já obtido com o objetivo de explorar outro espaço de soluções. Alguns exemplos de *Iterated* LK para o problema simétrico do caixeiro viajante são as implementações desenvolvidas por Neto (1999), Applegate *et al*. (2003) e Helsgaun (2000). Para o PCV Assimétrico, pode-se citar o *Kanellakis-Papadimitriou Local Search* (Kanellakis e Papadimitriou, 1980).

Para o mecanismo de perturbação, Martin *et al.* (1991) propõe uma seqüência de trocas de quatro arestas no percurso original. Este procedimento é conhecido como *kick* ou *double-bridge*. Applegate *et al*. (2000) discutem e apresentam vários experimentos realizados com a finalidade de se definir a melhor estratégia de *kick* e de geração de soluções iniciais para sua versão do algoritmo *Iterated* LK.

O eficiente algoritmo LK desenvolvido por Helsgaun introduz uma seqüência de movimentos que inicia em 2-opt e se estende até movimentos 5-opt. Essa implementação apresenta os melhores resultados em relação à qualidade de solução. Os movimentos 5-opt são restritos pelo conjunto de candidatos e um algoritmo de árvore geradora mínima é usado para definir o conjunto de candidatos.

Uma outra implementação eficiente do LK é apresentada por Nguyen *et al*. (2006), LK-NYYY. Os autores do LK-NYYY modificaram a idéia de Helsgaun em relação à vizinhança. Eles utilizaram uma seqüência de movimentos que consiste de um movimento 5-

opt válido seguido por algum movimento 3-opt válido, como um movimento básico. Esta estratégia beneficia o custo-benefício entre a qualidade do percurso e tempo de execução. Essa implementação usa a técnica "*don't look bits*" e, como estrutura de dados, segmentos de árvores.

A literatura contém ainda relatórios de várias implementações do algoritmo LK. Johnson e McGeoh (2002) apresentam uma breve descrição de algumas dessas implementações destacando as principais diferenças entre cada elas. Eles apresentam uma comparação dessas técnicas onde a qualidade da solução e o tempo de execução são considerados. Neste trabalho, serão investigadas as versões do LK propostas por Applegate *et al*. (1999b), Helsgaun (2000) e Nguyen *et al*. (2006).

3.2.1.2. Lin-Kernighan de Applegate, Bixby, Chvatal e Cook

Esta versão do algoritmo LK foi apresentada por Applegate, Bixby, Chvàtal e Cook (LK-ABCC), desenvolvida a partir do relatório de Applegate *et al.* (1999b). Esta utiliza o método *Q-Boruvka* para geração de soluções iniciais, lista de soluções candidatas 12 *quadrant*, a estratégia *don't look bits* e uma árvore em 2 níveis como estrutura de dados. Além disso, o LK-ABCC limita a busca do LK a 50 movimentos. O LK-ABCC é parte do projeto Concorde (Concorde, 2006; Applegate *et al.,* 1999a) e é utilizado no *Concorde solver*, um método exato para solução do Problema do Caixeiro Viajante.

3.2.1.3. Lin-Kernighan de Helsgaun

Helsgaun (2000) faz uma revisão do algoritmo Lin-Kernighan original, propondo e desenvolvendo modificações que melhoram seu desempenho. O aumento na eficiência é conseguido, primeiramente, por uma revisão de regras da heurística de Lin e Kernighan para restringir e dirigir a busca. Mesmo que suas regras pareçam naturais, uma análise crítica mostra que estas apresentam deficiências consideráveis.

Analisando a Regra 5 do algoritmo LK que é utilizada para o refinamento da busca limitando-a aos cinco vizinhos mais próximos, Helsgaun chama a atenção para a existência de um determinado risco, mostrando que esta pode impedir que a solução ótima seja encontrada. Considerando isso, Helsgaun apresenta uma medida de proximidade que melhor reflete as possibilidades da obtenção de uma aresta que faça parte do percurso ótimo. Esta medida, chamada *α-nearness*, é baseada na análise de sensibilidade usando uma árvore geradora mínima 1-*trees*.

Nos movimentos básicos, pelo menos duas pequenas modificações no esquema geral são realizadas. Primeiro, em um caso especial, o primeiro movimento de uma seqüência deve ser um movimento 4-opt seqüencial; os movimentos seguintes devem ainda ser os movimentos 2-opt. Em segundo, os movimentos 4-opt não-seqüenciais são tentados quando o percurso pode não ser melhorado por movimentos seqüenciais.

Além disso, o algoritmo modificado (LK-H) revisa a estrutura básica da busca em outros pontos. O primeiro e principal ponto, é a mudança do movimento básico de 4-opt para uma movimento 5-opt seqüencial. Além deste, experiências computacionais mostraram que o *backtracking* não é muito necessário neste algoritmo e sua remoção reduz o tempo de execução, não compromete o desempenho final do algoritmo e simplifica extremamente a execução do mesmo.

O novo algoritmo melhora o desempenho em comparação com o algoritmo original de acordo com as observações feitas por Christofides e Eilon (1972). Estes observaram que 5-opt pode produzir uma melhoria relativamente superior ao movimento 4-opt em relação à melhoria do 4-opt sobre 3-opt.

Outra diferença do algoritmo original é encontrada na análise de trocas não-seqüenciais. Com o objetivo de fornecer uma melhor adaptação frente às possíveis melhorias que consistem em trocas não-seqüenciais, o simples movimento 4-opt não-seqüencial foi substituído por um conjunto mais poderoso de movimentos não-seqüenciais.

Em relação a percursos iniciais, o algoritmo Lin-Kernighan aplica trocas de arestas diversas vezes no mesmo problema usando percursos iniciais diferentes. As experiências com várias execuções do algoritmo LK-H mostraram que a qualidade das soluções finais não depende fortemente das soluções iniciais. Entretanto, a redução significativa no tempo pode ser conseguida escolhendo soluções iniciais mais próximas do ótimo obtidas a partir de heurísticas construtivas, por exemplo. Desta forma, Helsgaun, em seu trabalho, apresenta também uma heurística simples que utilizou na construção de soluções inicias em sua versão do Lin-Kernighan.

Por outro lado, Nguyen *et al.* (2006) afirma que a utilização do movimento 5-opt como movimento básico eleva grandemente o tempo computacional necessário, embora as soluções encontradas sejam de muito boa qualidade. Além disso, o consumo de memória e tempo necessário ao pré-processamento do algoritmo LK-H são muito elevados quando comparados a outras implementações do LK. Em seu trabalho, Nguyen *et al.* apresentam uma nova versão para o LK a partir de modificações feitas no algoritmo LK-H.

3.2.1.4. Lin-Kernighan de Nguyen, Yoshihara, Yamamori e Yasunaga

O algoritmo LK-NYYY se baseia no algoritmo LK proposto por Helsgaun (2000) com duas modificações principais ambas voltadas à melhoria do desempenho em relação ao tempo computacional. A primeira modificação diz respeito à estrutura de dados que o LK-NYYY utiliza uma estrutura de vetores (extra) junto com uma estrutura de árvore em dois níveis. A utilização dessa estrutura "extra" permitiu a realização das trocas temporárias em tempo constante.

A segunda modificação feita por Nguyen *et al.* foi a utilização do movimento 5-opt apenas no início do processo, utilizando o movimento 3-opt como movimento básico no processo de busca do LK. Na idéia original de Helsgaun, é utilizado, como movimento básico, o 5-opt. Embora os resultados sejam de boa qualidade, a utilização do 5-opt como movimento básico eleva grandemente o tempo computacional necessário.

Ao escolher o movimento 3-opt como movimento básico, Nguyen *et al.* tem a intenção de fazer o balanceamento entre a qualidade das soluções obtidas e o tempo computacional. Com essas modificações, o LK-NYYY chegou a ser aplicado em instâncias com 1 milhão de cidades com um tempo computacional razoável (Gamboa *et al.*, 2005).

No LK-NYYY o procedimento de busca é limitado ao centésimo vizinho e a estratégia *don't look bit*, que melhora o desempenho do algoritmo, é utilizada, além de um *cache* extra que armazena os tamanhos de todas as arestas do percurso corrente. O LK-NYYY usa, também, um algoritmo guloso para geração das soluções iniciais e uma lista de soluções vizinhas 12-quadrant. Alguns resultados obtidos por esse algoritmo podem ser encontrados em DIMACS (2006). Uma diferença bastante significativa apontada por Gamboa *et al.* (2005)

quando comparados o LK-H e o LK-NYYY é que este último pode ser aplicado na solução de instâncias com mais de 1.000.000 de cidades enquanto que o LK-H suporta instâncias de até 18.900 cidades além de apresentar um consumo de memória bastante elevado.

3.2.2. Algoritmos Meméticos

A combinação de algoritmos evolucionários, dos quais os Algoritmos Genéticos têm recebido destaque nos último anos, com busca local foi inicialmente chamado de *Algoritmo Memético* (AM) por Moscato (1989).

Os Algoritmos Genéticos (AGs), propostos por Holland (1975), procuram mostrar como o processo evolucionário (segundo a teoria da Seleção Natural de Darwin onde os indivíduos mais aptos sobrevivem) pode ser aplicado na solução (ou obtenção de soluções aproximadas) de vários problemas de otimização. Em outras palavras, os AGs tentam simular o processo evolutivo dos indivíduos. Estes transformam uma população, com um custo para cada indivíduo, criando uma nova geração de indivíduos usando operadores de reprodução, cruzamento e mutação.

Nos AGs, o espaço de busca do problema é representado por uma coleção de indivíduos sendo que estes indivíduos podem ser representados por vetores. A proposta dos AGs é encontrar um indivíduo no espaço de busca (*população)* com o melhor material genético (indivíduos que representem boas soluções). A qualidade do indivíduo é medida pela função de avaliação.

A melhoria (evolução) das soluções é obtida pela realização de cruzamento entre as melhores soluções correntes e pela mutação das mesmas. Para a realização destas operações, são necessários métodos próprios: os operadores genéticos. Tratando-se do PCV, as informações genéticas passadas dos pais para os filhos são traduzidas como *sub-tours* (sub-percursos) de uma solução, propagados para novas soluções.

As operações de mutação levam o algoritmo a explorar pontos diferentes no espaço de soluções, de forma que este não se prenda a um ótimo local. As operações de cruzamento melhoram gradativamente a qualidade da população (Larranaga *et al*., 1999), uma vez que a troca de informações entre boas soluções pode levar a soluções ainda melhores. Alguns exemplos de operadores genéticos de cruzamento e mutação são apresentados na próxima sub-seção.

Com o tempo, os experimentos foram mostrando que os AGs puro não eram muito apropriados para busca refinada em espaços de busca complexo e que a hibridização deste com outras técnicas proporcionaria uma melhor eficiência (Krasnogor e Smith, 2000).

A partir da identificação das dificuldades dos AGs, muitos pesquisadores passaram a introduzir, nos AGs, elementos de outras heurísticas (Freisleben e Merz, 1996). Esta hibridização inclui a introdução de heurísticas construtivas para geração da população inicial, heurísticas para melhoramento do percurso pela produção de um ótimo local, operadores mais elaborados para o PCV, etc. Embora os Algoritmos Meméticos se refiram mais especificamente à utilização de busca local nos AGs para o refinamento das soluções, estes são, também, retratados na literatura como Algoritmo Genético Híbridos, *Genetic Local Searchers*, Algoritmo Genético Lamarckiano, *Baldwinian Genetic Algorithms* etc.

O termo "Algoritmo Memético" se deve ao uso do conceito de *memes* como uma unidade de informação que se propaga como a troca de idéias entre pessoas. Dawkins (1976)

define um *meme* como sendo uma informação que é transmitida como uma característica cultural. Desta forma, *memes* podem ser entendidos como idéias ou experiências propagadas como uma cultura.

Um *meme*, como uma unidade de transmissão de cultura, é replicado por imitação (Moscato e Norman, 1992), mas antes de ser transmitido, esse é adaptado à pessoa que o transmite (a maneira como essa pessoa pensa, entende e processa o *meme*), enquanto que os genes (no caso dos AGs) são transmitidos na íntegra. Moscato e Norman (1992) comparam este pensamento com um refinamento local. Desta forma, o uso na busca local permite que indivíduos evoluam de forma autônoma para ótimos locais sem sofrer mutações e/ou cruzamentos. A evolução destes indivíduos se dá pelo acréscimo de *memes*.

Fazendo uma analogia à transmissão genética, a transmissão cultural corresponde ao fluxo de informação num processo evolucionário e os genes correspondem a partes de um cromossomo que servem como uma unidade viável de seleção natural. No caso dos AMs, os *memes* referem-se a estratégias (refinamento local, perturbação ou métodos construtivos, por exemplo) para melhoria dos indivíduos (Krasnogor e Smith, 2000).

Como a busca local é usada depois da aplicação dos operadores genéticos, isso implica dizer que os AMs são extensões dos AGs que aplicam a busca local separadamente, para refinar a busca nos espaço de soluções. Moscato (1989) utiliza os conceitos de cooperação e competição onde o primeiro denota os processos de troca de informações entre os indivíduos e o segundo denota os processos de seleção em que indivíduos competem entre si.

O sucesso dos AMs se deve à junção das habilidades dos AGs com as habilidades da busca local, combinando a adaptação evolucionária da população com os conhecimentos adquiridos pelo indivíduo durante sua existência (Krasnogor e Smith, 2000).

Pode-se diferenciar os AGs dos AMs dizendo que o primeiro tenta imitar o processo da evolução biológica e o segundo tenta imitar a evolução cultural. Enquanto que nos AMs ocorre uma evolução cultural, onde a informação é transmitida pela *comunicação* entre os indivíduos, nos AGs a evolução se dá por mecanismos de *recombinação* (cruzamento e mutação) (Garcia *et al.,* 2001).

Como exemplo de AM para o PCV tem-se o algoritmo desenvolvido por Freisleben e Merz (1996) (o termo adotado é *Genetic Local Serch*) que utiliza a abordagem construtiva do Vizinho Mais Próximo para a geração da população inicial, como mecanismo de busca local utiliza o Lin-Kernighan e, no processo de mutação, utiliza um operador que realiza 4 modificações não seqüenciais somado à execução do Lin-Kernigham. Além disso, Freisleben e Merz (1996) propõem e aplicam o operador de cruzamento *Distance Preserving Crossover* (DPX), desenvolvido com base nos estudos feitos por Boese (1995) que, analisando o PCV, notou que, definindo a distância entre dois percurso*s* como sendo o número de arestas que estão contidas no primeiro, mas não estão no segundo, a distância média entre o percurso localmente ótimo é similar à distancia média entre ótimo local e o global. Conseqüentemente, para permitir que o salto no espaço de busca, no qual o maior será, provavelmente o ótimo global, o objetivo do DPX é produzir um filho que tenha a mesma distância para os pais que de um pai para outro. A descrição completa do funcionamento do DPX pode ser encontrada no trabalho de Freisleben e Merz (1996). A Figura 6 mostra, como exemplo, o pseudocódigo deste algoritmo memético.

```
Procedimento AM
        Inicializa a população P com heurística do Vizinho Mais Próximo
        Para cada individuo i ∈ P faça Lin-Kernighan-Opt (i)
        Repete
                Para i=0 até #cruzamentos faça
                Seleciona aleatoriamente os pais i_a, i_b ∈ P
                i_c ← DPX-STSP (i_a, i_b)
                Lin-Kernighan-Opt (i_c)
                Com certa probabilidade faça Mutação (i_c)
                Substitui um indivíduo de P por i_c
        Fim
        Até convergir
Fim
```

Figura 6: Algoritmo Memético de Freisleben e Merz (1996)

Em Ozcan e Erenturk (2004) é apresentada uma breve revisão sobre aplicações dos AMs para PCV bem como comparações entre um conjunto de operadores. Sobre operadores, Aarts e Verhoeven (1997) apresentam resultados de testes de AMs para o PCV, usando 2-opt e Vizinhança Variável como busca local, o algoritmo Lin-Kernighan (Lin e Kernighan, 1973).

Buriol *et al. (*2003) apresentam um novo algoritmo memético para o PCV Assimétrico, com novos operadores e uma nova busca local chamada RAI – *Recursive Arc Insertion*, além de utilizar a população estruturada na forma de uma árvore ternária com 3 níveis.

Além das mencionadas, outras abordagens podem ser aplicadas aos AM como, por exemplo, Busca Tabu, *Simulated Annealing*, Busca Local Guiada e GRASP.

3.2.2.1. Operadores Genéticos

Para ilustrar o funcionamento dos operadores genéticos e sua importância no processo de "evolução" dos Algoritmos Genéticos e, conseqüentemente, dos Algoritmos Meméticos, são apresentados, nessa seção, os operadores de cruzamento **PMX** e **OX1**, além dos operadores de mutação **ISM** e **SIM**.

Sugerido por Goldberg e Lingle (1985), o **PMX** (*Operador Partially-Mapped*) é um operador de cruzamento que age trocando *blocos* de genes entre os pais, gerando novos filhos da seguinte forma:

1. Seleciona aleatoriamente dois pontos de corte. Supondo que as cidades pertencentes ao intervalo entre o ponto de corte *A* e o ponto *B* sejam (4 5 6) do primeiro pai, P_1,e (1 6 8) do segundo pai, P_2. Relacionando estes intervalos, tem-se um conjunto $S = \{(4,1),(5,6),(6,8)\}$.
2. Copia o intervalo de P_1 para F_2 e o intervalo de P_2 para F_1 onde F_1 e F_2 são os filhos resultantes do cruzamento.

3. Tenta mapear o elemento *i* de P_k (sendo $k = \{1,2\}$) para F_k. Se este elemento já existe em F_k, tenta-se mapear o elemento *i'* que esteja relacionando a *i* no conjunto *S*. Se o elemento *i'* já existir em F_k então, tenta-se mapear o elemento *i"* que esteja relacionando ao elemento *i'* no conjunto *S* e assim sucessivamente, até que o mapeamento seja possível.
4. Repete o Passo 3 até que F_k esteja completo.

A Figura 7 mostra um exemplo onde os pontos de corte sorteados foram 4 e 6. Os intervalos são copiados diretamente para os filhos e as posições restantes são preenchidas no passo seguinte.

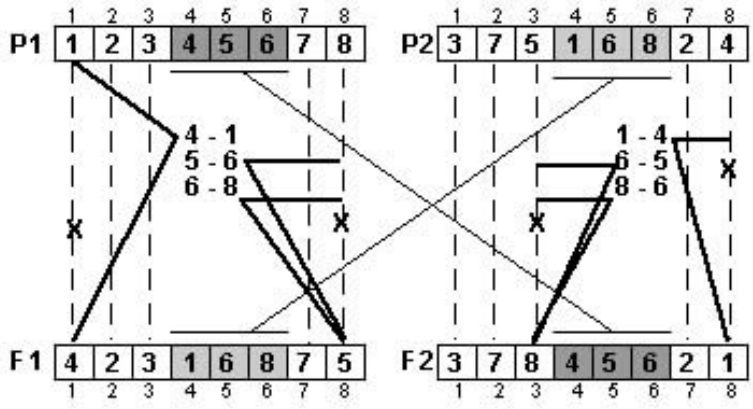

Figura 7: Cruzamento PMX (Larranaga *et al*., 1999)

Observa-se que é realizada uma tentativa de mapear a cidade 1 de P_1 para F_1, mas não é possível porque está já existe na posição 4. Pela relação, a próxima a se tentar mapear é a cidade 4.

Entre P_2 e F_2, não é possível mapear a cidade 5 de P_2 para a posição 3 de F_2 porque esta já existe. Pela relação, tenta-se mapear a cidade 6, mas este já existe também. A próxima tentativa, que é realizada com êxito, é mapear a cidade 8 para a posição 3.

O operador de cruzamento **OX1** foi proposto por Davis (1985) e funciona da seguinte forma:

1. Seleciona, aleatoriamente, dois pontos de corte *A* e *B*;
2. Copia o intervalo S_i (S_i corresponde aos elementos entre os pontos *A* e *B*) de P_i (*i-ésimo* pai) para F_i (*i-ésimo* filho);
3. Iniciando do ponto de corte *B*, segue preenchendo o restante do cromossomo, ignorando os elementos já presentes em F_i. Este preenchimento é feito de forma circular, ou seja, do último elemento volta-se para o primeiro até que todos os elementos sejam mapeados para F_i.

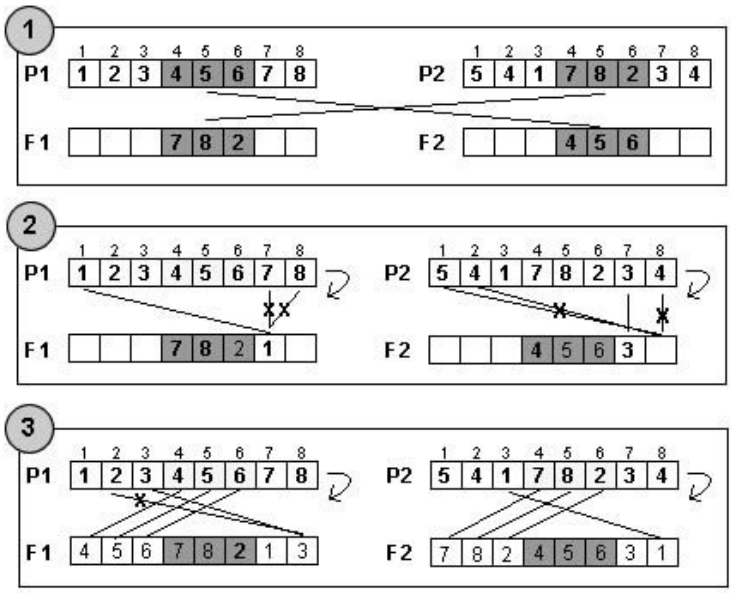

Figura 8: Operador de cruzamento OX1

Como pode ser visto na Figura 8, inicialmente, os elementos pertencentes ao intervalo de corte são copiados dos pais P_1 e P_2 para os filhos F_2 e F_1, respectivamente. Observando P_1 e F_1, percebe-se a tentativa de mapear o 7º e depois o 8º elemento de P_1 para F_1, o que não é possível pois F_1 já possui as cidades 7 e 8 em seu percurso. Chegando ao final, seguem-se as tentativas de mapeamento a partir do início da lista. Na terceira tentativa, a cidade 1 é mapeada para a posição 7 de F_1. E depois, tenta-se mapear o 2º elemento de P_1 para o 8º de F_1 o que também não é possível. O elemento 3 de P_1 é mapeado para a posição 8 de F_1 e, na seqüência, os elementos 4, 5 e 6 de P_1 são mapeados com sucesso para as posições 1, 2 e 3 de F_1, respectivamente. O mesmo comportamento pode ser observado entre P_2 e F_2.

O primeiro operador de mutação a ser apresentado é o ISM (*Insertion Mutation*) sugerido por Forgel (1988). Neste operador, uma cidade *k* é escolhida aleatoriamente e removida do percurso. Em seguida, a cidade *k* é inserida novamente no percurso, num outro local, selecionado aleatoriamente, como mostra a Figura 9.

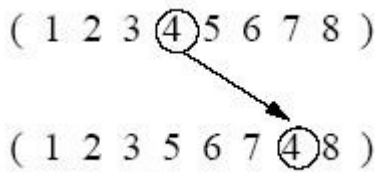

Figura 9: Operador de mutação ISM

Outro exemplo de operador de mutação é o SIM (*Simple Inversion Mutation*) proposto por Holland (1975) que tem seu funcionamento baseado na inversão de parte de um percurso. A Figura 10 ilustra seu funcionamento onde dois pontos de corte *A* (entre as cidades 3 e 4) e *B* (entre as cidades 6 e 7) são selecionados aleatoriamente. O passo seguinte é inverter *sub-percurso* de *A* para *B*. Na representação em grafos, é importante observar a direção dos arcos.

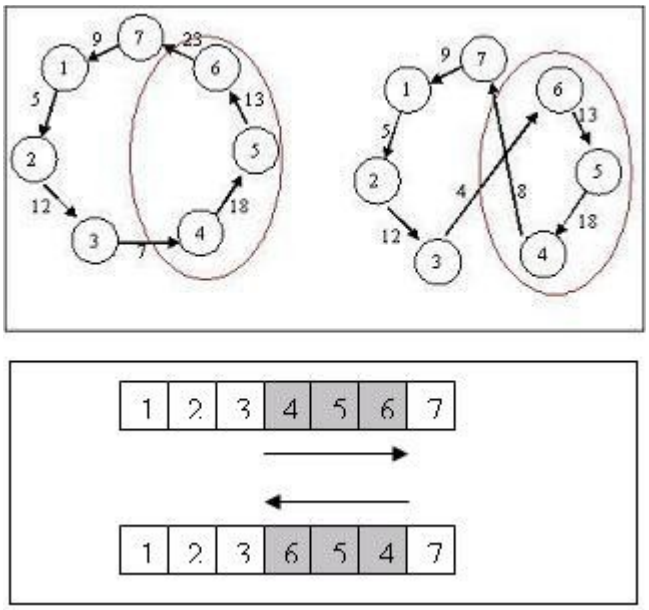

Figura 10: Operador de Mutação SIM

Outros operadores, tanto para mutação quanto para cruzamento, podem ser encontrados em Larranaga *et al*. (1999) onde vários operadores genéticos reportados na literatura são discutidos e comparados. No mesmo seguimento, Pohlheim (2001) apresenta um estudo sobre operadores de mutação e cruzamento além de métodos de seleção dos pais das próximas gerações e modelos de população. Ainda, Oliver *et al.* (1987), apresentam um estudo sobre o comportamento dos operadores de cruzamento para o PCV.

3.2.2.2. O Algoritmo Memético de Krasnogor e Smith

O Algoritmo Memético proposto por Krasnogor e Smith (2000) (AM_KRS) é composto de dois processos de otimização. O primeiro é um AG que controla o comportamento do esquema (os processos de otimização). O segundo é o método de Monte Carlo (MC) utilizado como busca local e processo de diversificação. No início, quando a população é diversa, o MC atua como um procedimento de busca local explorando a vizinhança. Nas outras etapas do processo de solução, quando a população está quase convergindo ou estagnada, o objetivo passa ser a diversificação da busca, permitindo que diferentes pontos de atração sejam explorados.

A auto-adaptação do comportamento da busca local para *intensificação* ou *diversificação* é controlada por um parâmetro de *temperatura*. Esta *temperatura* determina a probabilidade da realização dos *movimentos*. Quanto maior for o espalhamento da função de custo na população, maior será a probabilidade que um indivíduo tem de "mudar" para uma posição distante de sua posição original, explorando, assim, o espaço de busca.

No esquema geral, temperatura foi definida como:

$$\text{Temperatura} = 1 / (\text{custoMaximo} - \text{custoMínimo})$$

No entanto, para a aplicação no PCV, com o objetivo de produzir um deslizamento dinâmico, uma modificação foi feita nessa definição e a temperatura ficou definida como:

Temperatura= 1/(custoMáximo – ***custoMédio****)*

Para a exploração da vizinhança no processo de busca local, foi utilizado o *2-opt*. Além disso, um número aleatório entre 1 e 10% do número total de cidades, foi utilizado para determinar as várias aplicações deste movimento num determinado indivíduo. Depois da aplicação dos movimentos, o novo indivíduo é introduzido na população segundo uma distribuição de *Boltzmann* baseada na temperatura da população atual.

3.2.2.3. O Algoritmo Memético de Buriol

Em seu trabalho, Buriol *et al.* (2003) introduz um novo algoritmo memético que tem apresentado um bom desempenho em grandes instâncias assimétricas. Este algoritmo incorpora um novo método de busca local desenvolvido, especialmente, para instâncias assimétricas chamado *Recursive Arc Insertion* (RAI). Além do RAI, Buriol apresenta ainda um novo operador de recombinação chamado *Strategic Arc Crossover* (SAX), criado a partir do *Strategic Edge Crossover* (SEX) proposto por Moscato e Norman (1992), também adaptado para instâncias assimétricas.

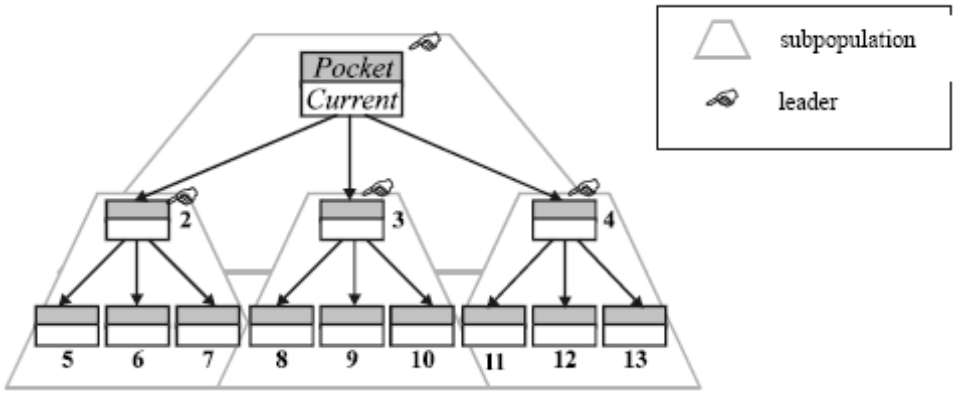

Figura 11: Estrutura da População (Buriol *et al.*, 2003)

Além do RAI e do SAX, o algoritmo apresentado por Buriol possui características que cooperam para sua eficiência como uma estrutura de dados eficiente, por exemplo. A população possui uma organização hierárquica na forma de uma árvore ternária tripla composta de 13 agentes distribuídos em 4 sub-populações, sendo cada agente composto por 4 indivíduos. Cada sub-população possui um líder e três outros indivíduos subordinados. A Figura 11 ilustra a forma de organização da população.

As comparações com outras abordagens, que Buriol apresenta em seu trabalho, mostram o bom desempenho de seu algoritmo.

3.2.3. Algoritmos Transgenéticos

A Transgenética Computacional é uma das abordagens mais recentes da computação evolucionária introduzida por Goldbarg e Gouvêa (2000) e sua metáfora está baseada em um processo de Endosimbiose / *Quorum Sensing*. Schmidt (2006) define um endossibionte como qualquer organismo que vive no interior do corpo ou das células de outro organismo,

realizando uma relação ecológica designada como endossimbiose. A habilidade das bactérias se comunicarem e coordenar o desempenho através de sinais moleculares recebeu o nome de *Quorum Sensing* (Margulis, 1991; Schmidt, 2006). Em outras palavras, a endosimbiose corresponde às regras no processo de comunicação realizado pelas bactérias (Schmidt, 2006). Segundo Goldbarg e Goldbarg (2002), os algoritmos transgenéticos trabalham basicamente com populações de indivíduos de naturezas distintas em um processo endosimbiótico de troca de informações genéticas e não genéticas.

Essa abordagem propõe, ainda, a utilização de um **Banco de Informações Genéticas (BIG)**, composto por informações externas obtidas *a priori* para guiar a ação dos vetores. Essas "informações externas" podem se referir a partes de uma Árvore Geradora Mínima que podem ser usadas como sub-percursos para aplicação na solução do PCV. Informações úteis que podem colaborar para a obtenção de uma boa solução podem ser encontradas em resultados teóricos do problema a ser solucionado ou mesmo através de procedimentos heurísticos. Além disso, informações obtidas ao longo do processo de evolução também são utilizadas (Schmidt, 2006).

Nos algoritmos transgenéticos, a "evolução" se dá em três níveis diferentes, nos quais as informações são armazenadas e gerenciadas. No primeiro nível, está a população de cromossomos que representa a memória corrente do processo de busca. No segundo nível estão os vetores transgenéticos que é uma população munida de "ferramentas" para a realização dos processos de diversificação e intensificação da busca no espaço de soluções. O terceiro nível é composto por regras que comandam a interação entre a população de cromossomos e a dos vetores transgenéticos (Ramos, 2005).

Nessa nova abordagem, a população de cromossomos tem estrutura similar à dos algoritmos genéticos, porém, a evolução dessa população está baseada na troca de informações entre esta e a população composta pelos vetores transgenéticos e não na troca de informações dentro da população de cromossomos, como é nos algoritmos genéticos.

3.2.3.1. Vetores Transgenéticos

Um vetor transgenético transporta uma ou mais cadeias de informação e dispõe de métodos (ou procedimentos) que definem sua atuação na população de cromossomos. O método inclui a programação necessária para que a cadeia seja capaz de atuar sobre um cromossomo, realizando a inserção da informação transportada. Um desses procedimentos corresponde à operação de reorganização do código manipulado que, na abordagem clássica, é denominado de "operador". O método deve conter, além do operador, as regras de avaliação do sucesso do ataque e as regras referentes à inviolabilidade da manipulação.

As cadeias de informação que serão transportadas pelos vetores transgenéticos podem ser obtidas através de células doadoras (cromossomos), recombinações endosimbióticas entre os vetores do processo, DNA obtido a *priori* ou procedimentos construtivos.

O processo de transportar e transcrever informações para a população de cromossomos foi formalizado por Gouvêa e Goldbarg (2003) e é descrito na seqüência. Seja

- P uma população de q indivíduos, $P = \{S_1, S_2, ..., S_q\}$, onde cada indivíduo S_i, $i = 1, ..., q$, é um conjunto de valores inteiros de comprimento n representando uma solução do problema;
- f a função adequação, $f: S_i \rightarrow \Re^*$, $i = 1, ..., q$.

Um vetor transgenético $\lambda \in C$, onde C é o conjunto de todos os vetores possíveis, é dado por $\lambda = (I, \Phi)$ em que:

- I representa a cadeia de informação transportada pelo vetor transgenético; e
- Φ representa o método de manipulação com $\Phi = (p_1, p_2 \ldots, p_s)$, onde p_j, $j = 1,\ldots, s$, são procedimentos que definem o processo de manipulação.

Os vetores transgenéticos são caracterizados pelo fato de o seu método de manipulação, dada a informação I, alterar a configuração do cromossomo através da manipulação $\Phi = S_i \xrightarrow{I} S'_i$. Os procedimentos descritos na Tabela 4 compõem o método de manipulação dos vetores da Transgenética Computacional.

Tabela 4: Métodos de Manipulação dos Agentes da Transgenética Computacional

Procedimento	Denominação	Descrição
Procedimento 1 (p_1)	Ataque (Λ)	Define o critério de avaliação que estabelece quando um cromossomo é suscetível à manipulação do vetor.
Procedimento 2 (p_2)	Operador de Transcrição (Γ)	Seja: $A: S_i \rightarrow$ falso ou verdadeiro, $i = 1,\ldots,q$. Se $A(S_i) =$ *"verdadeiro"*, esse procedimento define como a informação I, transportada pelo vetor, será transferida para o cromossomo.
Procedimento 3 (p_3)	Bloqueio da Informação Transcrita (ψ)	Torna o resultado da manipulação inviolável por um certo período de tempo – número de iteração, gerações de cromossomos, etc.
Procedimento 4 (p_4)	Desbloqueio da Informação Transcrita (ψ^1)	Torna o resultado da manipulação sem restrições.
Procedimento (p_5)	Operador de Identificação (Λ)	Identifica posições que serão utilizadas para limitar a operação do vetor.
Procedimento (p_6)	Operador de Recombinação (Π)	Identifica a origem e o comprimento de cada sub-cadeia de informação transportada pelo vetor de manipulação.

Alguns exemplos de vetores transgenéticos são os Vírus e as Partículas Genéticas Móveis (PGMs). O vetor transgenético $\lambda = (I, \Phi)$ é dito um Vírus quando incorpora os procedimentos de 1 a 4 da Tabela 4 e é chamado de PGM quando sua cadeia de informação (I) é traduzida no formato genético e seu método (Φ) utiliza somente os procedimentos 1 e 2.

3.2.3.2. Regras de Administração

As populações de cromossomos e de vetores transgenéticos evoluem trocando informações permanentemente. Essa troca de informações é gerenciada por regras de três tipos.

As *regras do tipo 1* são responsáveis pela construção de cadeias de informações que poderão ser incorporadas aos cromossomos, modificando sua estrutura e, provavelmente, sua função de adequação (função objetivo).

O modo como essas cadeias serão incorporadas ao cromossomo são definidas pelas *regras do tipo 2* que correspondem a orientações para constituição de operadores de manipulação genética (método de transcrição). Se um vetor não é bem-sucedido em sua tentativa de manipulação (ataque), diz-se que o cromossomo "resistiu" à manipulação. O critério de resistência à manipulação pode levar em conta a variação da função de adequação decorrente da manipulação.

As *regras do tipo 3* definem uma série de estratégias para a ação dos vetores transgenéticos. Essas regras podem se referir ao número de vetores que participarão de cada iteração no algoritmo, aos critérios que definirão quantos cromossomos serão atacados, à seqüência de ataque dos vetores, etc., constituindo, em seu todo, um esquema de manipulação.

Resumidamente, as *regras dos tipos 1 e 2* referem-se à construção de cadeias de informações e seu método de manipulação no cromossomo (cadeia + método) e as *regras do tipo 3* referem-se a orientações diversas que podem estar incorporadas ao algoritmo. O conjunto das regras de administração e os procedimentos inerentes aos vetores constituem um esquema de manipulação.

3.2.3.3. O Algoritmo Transgenético – ProtoG

A tese apresentada por Ramos (2005) apresenta, pelo menos, duas grandes contribuições. A primeira é a criação de uma heurística, denominada Operon, que utiliza as metodologias estatísticas Análise de Agrupamentos e Análise de Componentes Principais, e tem como finalidade construir, de forma dinâmica e de boa qualidade, cadeias de informações a fim de promover uma busca "inteligente" no espaço de soluções. A segunda grande contribuição é a utilização de análises estatísticas adequadas à avaliação da performance de algoritmos destinados à solução desses problemas. Além disso, Ramos apresenta um algoritmo transgenético ProtoG (Goldbarg *et al.*, 2001) aplicado ao Problema do Caixeiro Viajante.

O ProtoG baseia-se em duas fases: a fase de construção de uma base de dados e de definição de critérios para a construção dos vetores transgenéticos e a fase evolutiva. Os algoritmos ProtoG são baseados exclusivamente no paradigma intracelular para realizar a exploração do espaço de busca. Por este motivo, operações de cruzamento, mutação, por exemplo, não existem nessa abordagem. Uma descrição de um algoritmo ProtoG é apresentada na Figura 12.

Inicialmente, é gerada uma população de cromossomos (Passo 1) que, opcionalmente, pode ser submetida a um processo de refinamento através de alguma heurística, antes do início da primeira fase do ProtoG.

```
ProtoG
1.  Gerar e avaliar uma população inicial de cromossomos
2.  Construir uma base de dados para construção de vetores transgenéticos l  (1ª Fase)
3.  Repetir
4.      Gerar um subconjunto de vetores transgenéticos
5.      Para todo cromossomo da população, Faça
6.      Se o cromossomo é sensível à manipulação do vetor, então
7.      Início (2ª Fase)
8.          Realizar manipulação
9.          Reavaliar o cromossomo manipulado
10.         Se o cromossomo atende ao critério de seleção de realimentação, então incluir
            informações selecionadas do cromossomo na base de dados para construção de
            vetores transgenéticos l
11.         Fim
12.  Até que Critério_de_Parada seja atendido
Retornar
```

Figura 12: Algoritmo Transgenético Proto Gene – ProtoG (Gouvêa, 2001)

Na primeira fase, uma base de dados é construída a fim de possibilitar a geração de vetores transgenéticos (Passo 2). Nesse procedimento são definidos os elementos relativos às *regras dos tipos 1 e 2*. Conforme as regras definidas, um subconjunto de vetores transgenéticos é gerado no Passo 4. Nesse passo, um determinado número de vetores é gerado, cada vetor carregando uma cadeia de informação obtida de uma *regra do tipo 1* e possuindo um operador definido por uma *regra do tipo 2* (cadeia + método).

Na segunda fase, para cada cromossomo, faz-se o procedimento descrito a seguir (passos 8 a 10). Se o cromossomo é sensível à manipulação do vetor transgenético (ataque definido por p_1 da Tabela 4): a manipulação é realizada (transcrição definida por p_2, Tabela 4); o cromossomo é reavaliado; e, se ele é considerado como um bom exemplo de evolução então suas informações (selecionadas) são incluídas na base de dados para construção de vetores transgenéticos. Nessa versão do ProtoG, os vetores transgenéticos considerados são PGMs e, por isso, não são utilizados os demais procedimentos.

O algoritmo continua até que um critério de parada, previamente definido, seja atendido. Esse critério de parada pode ser estabelecido em função de um número de iterações sem melhoria, de um tempo de processamento máximo, de encontrar a solução ótima etc.

Ramos propõe também, uma estratégia de renovação de parte da população de cromossomos indicada pela adoção de um limite mínimo no coeficiente de variação da função de adequação dos indivíduos, calculado com base na população. Além disso, são propostas três análises estatísticas para avaliar a performance de algoritmos: Análise de Regressão Logística, Análise de Sobrevivência e Análise de Variância não paramétrica.

Foram realizados experimentos com sessenta e uma instâncias do PCV euclidiano, com tamanhos de até 1.655 cidades. Estes experimentos mostram que o ProtoG pode ser considerado um algoritmo competitivo para solucionar o PCV pois tem obtido resultados de boa qualidade com afastamento em relação à solução ótima inferior a 10%.

3.2.4. Otimização por Nuvem de Partículas

A Otimização por Nuvem de Partículas, do inglês *Particle Swarm Optimization* (PSO), é uma abordagem baseada em população proposta inicialmente por Kennedy e Eberhart (1995), com base nos estudos de Reynolds (1987) e Heppner e Grenander (1990) que apresentaram modelos de simulações de vôos de pássaros. O objetivo destes pesquisadores era possibilitar uma simulação gráfica do comportamento dos pássaros quando voando em bandos. Porém, descobriu-se que esse modelo poderia ser usado na otimização de problemas, onde a sincronia de comportamento do bando poderia ser trabalhada como uma função que mediria os esforços dos pássaros para manter uma distância ótima entre estes e seus "vizinhos".

Utilizando esse modelo natural na criação de um modelo computacional análogo (a PSO) onde problemas de otimização pudessem ser atacados, o bando foi mapeado como sendo uma nuvem ou população e cada pássaro uma solução em potencial, chamada **partícula,** no espaço de soluções. Todas as partículas possuem valores de custo que são avaliados pela função-objetivo e têm velocidades que direcionam o deslocamento das partículas no espaço de soluções. O deslocamento das partículas é orientado pelas partículas ótimas atuais. Além disso, cada partícula possui informações sobre sua posição, bem como a velocidade e o valor da função-objetivo para essa posição e a informação da melhor posição já atingida. Possui também informações sobre as partículas vizinhas tais como a melhor posição e o melhor valor da função-objetivo.

De uma forma geral, um algoritmo PSO é iniciado com um grupo de partículas aleatórias e procura por soluções ótimas atualizando as partículas. Nas iterações, cada partícula é atualizada seguindo os valores da melhor solução encontrada pela partícula ao longo da busca, geralmente chamada *pbest*; e a melhor solução encontrada por todas as partículas, chamado de *gbest* (*global best*). Durante o processo de busca, o comportamento de uma partícula depende de três possíveis escolhas:

- Seguir seu próprio caminho ignorando as partículas vizinhas;
- Seguir em direção à sua melhor posição já encontrada (*pbest*);
- Seguir em direção à melhor posição da melhor partícula vizinha (*gbest*);

Após encontrar os valores do *pbest* e do *gbest*, cada partícula atualiza sua velocidade e posição de acordo com as Equações (1) e (2) (Pang *et al.*, 2004)

$$V_i^{t+1} = \omega V_i^t + C_1 \,.\, Rand(\,) \,.\, (P_i^t - X_i^t) + C_2 \,.\, Rand(\,) \,.\, (P_g^t - X_i^t) \qquad (1)$$

$$X_i^{t+1} = X_i^t + V_i^{t+1} \qquad (2)$$

Nessas equações, *i* corresponde à *i-ésima* partícula do conjunto de partículas (nuvem) e *t* é o número da iteração; V_i é o vetor de velocidade da partícula *i* e é limitado por um valor V_{max} correspondente à velocidade máxima de cada partícula; X_i é o vetor de posição da partícula *i* num espaço de busca de dimensão *N*.

$V_i = \{V_{i1}, V_{i2}, \ldots, V_{iN}\}$

$X_i = \{X_{i1}, X_{i2}, \ldots, X_{iN}\}$

P_j é a melhor posição local já encontrada pela partícula *i* e P_g a melhor posição já encontrada por todas as partículas. *Rand( )* corresponde a um número aleatório entre (0,1), ω é

chamado de fator de inércia e C_1, C_2 são fatores de aprendizagem. Os três coeficientes ω, C_1 e C_2 significam respectivamente:

- O quanto a partícula confia em si mesma;
- O quanto ela confia na sua experiência;
- O quanto ela confia nos seus vizinhos.

Os parâmetros que precisam ser considerados quando da aplicação de um algoritmo PSO são o *Número de partículas*, que geralmente fica entre 20 e 40; a *Dimensão das partículas*, que é definida de acordo com a dimensão do problema a ser atacado; e o valor V_{max} que define velocidade máxima de mudança de uma partícula. Ainda, para os *Fatores de aprendizagem*, C_1 e C_2 geralmente se usa o valor 2, podendo variar de 0 a 4; e, como *Condição de parada*, comumente se utiliza um valor máximo de iterações e/ou uma taxa mínima de afastamento em relação à solução ótima (caso seja conhecida), mas outras estratégias podem estar sendo adotadas dependendo do objetivo do experimento e do problema abordado. A Figura 13 apresenta o esquema geral de um algoritmo PSO aplicado ao PCV.

```
Para cada partícula p ∈ P
  Inicie(p);
Fim_Para
Faça
  Para cada partícula p ∈ P
    Se Custo(p) < Custo(pbest) então
      pbest ← p
   Fim_para
  gbest ← . Mínimo(Custo(p)) para todo p ∈ P
  Para cada partícula p ∈ P
    Vp ← . Velocidade(p) //equação (1)
    p ← . AtualizePosição(p,Vp) //equação (2)
  Fim_Para
Enquanto (critério de parada) não é atendido
```

Figura 13: Esquema geral de um algoritmo PSO para o PCV

Souza (2006) faz uma comparação entre a abordagem PSO e os Algoritmos Genéticos dizendo que ambos atuam no melhoramento de um conjunto de soluções (a população) através da avaliação de um valor de custo associado a cada elemento desse conjunto e realizam modificações nessa população à procura de soluções ótimas. No entanto, diferentemente do AGs, o PSO não tem utiliza operadores genéticos pois as próprias partículas fazem sua atualização usando informações tais como sua velocidade. Além disso, as partículas possuem "memória", uma característica importante para essa abordagem, que lhes permite guardar a informação de sua melhor solução já encontrada. Ainda, no caso dos AGs, a disputa é pela sobrevivência através da adaptação enquanto que no PSO a estratégia utilizada é a da cooperação onde a população inteira se move como um grupo em direção a uma área ótima. Na PSO, somente o *gbest* fornece informação às outras partículas, a evolução somente "enxerga" a melhor solução e, na maioria dos casos, todas as partículas tendem a convergir rapidamente para a melhor solução.

Atualmente, a PSO tem sido utilizada em várias áreas tais como:

- Redes neurais artificiais para o treinamento da rede (Kennedy e Eberhart, 1995) levando algumas aplicações das áreas de diagnóstico médico e indústria a usarem uma rede neural guiada por um algoritmo PSO;
- Missões de inspeção/vigilância com base no problema do caixeiro viajante (Secrest, 2001);
- Otimização de funções de controle nebulosas (lógica *Fuzzy*) (Esmin *et al.*, 2002);
- Identificação de partículas por dispersão de luz (Parsopoulos e Vrahatis 2002);
- Programação de sistemas de manufatura (Jerald *et al.*,2004);
- Controle reativo de tensão elétrica e potência (Yoshida *et al.*, 2001).

Como exemplos de algoritmos de Otimização por Nuvem de Partículas aplicados ao PCV, pode-se citar o algoritmo de Wang *et al.* (2003), Clerc (2004), Pang *et al.* (2004), Machado e Lopes (2005) e Souza (2006). Estes dois últimos, representando as abordagens mais recentes, são descritos em mais detalhes nos tópicos seguintes.

3.2.4.1. PSO de Machado e Lopes

Machado e Lopes apresentam um algoritmo PSO (PSO-ML) hibridizado com busca local e técnicas dos Algoritmos Genéticos. O procedimento de busca utilizado é o *Fast Local Search* (FLS) que é uma busca local guiada baseada no 2-opt e é aplicada para o refinamento das soluções obtidas explorando a vizinhança de cada partícula. Dos Algoritmos Genéticos, é utilizado o conceito de operadores genéticos para o deslocamento das partículas no espaço de solução.

Nesse algoritmo, cada partícula representa uma possível solução, ou seja, um percurso completo. As partículas são compostas por três vetores principais que guardam informações sobre a posição atual da partícula, a melhor posição atingida e a melhor solução de todas.

Inicialmente, cada partícula da população (nuvem) é gerada aleatoriamente, ou seja, o percurso e a posições atual são inicialmente aleatórios. No entanto, para se garantir uma boa diversidade, não são admitidas, na população inicial, partículas que assumam a mesma posição no espaço de soluções.

Como no PSO-ML cada posição no espaço de busca representa uma solução, a distância entre duas partículas A e B é calculada comparando os vetores a partir de um ponto (cidade) comum. Inicialmente a distância é nula e, para cada posição do vetor, os valores correspondentes são comparados e sempre que forem diferentes, a distância é incrementada em 1.

O movimento das partículas no espaço de soluções é baseado nas equações de deslocamento de um algoritmo PSO comum e no operador de cruzamento OX adaptado. A aplicação do OX resulta em duas novas partículas que são avaliadas e aquela que representar uma melhor solução será considerada a nova posição da partícula corrente.

Outros exemplos de algoritmos PSO híbridos podem ser encontrados em Lvbjeg *et al.* (2001) e Naka *et al.* (2003).

3.2.4.2. PSO de Sousa

Souza (2006) também utiliza um PSO hibridizado (PSO-S) que utiliza o bem conhecido algoritmo LK como busca local e, como operador de velocidade, um algoritmo

path-relink que é uma técnica de intensificação proposta por Glover (1963) que consiste, basicamente, em gerar um caminho entre duas soluções, criando novas soluções.

Cada posição *p* corresponde a uma seqüência de permutações $x_1, x_2, ..., x_N$, onde *N* é o número de cidades. O primeiro passo do algoritmo é definir as probabilidades associadas a cada velocidade, onde p1, p2 e p3 correspondem, respectivamente, à probabilidade de que a partícula siga seu próprio caminho, siga em direção a sua melhor posição já atingida (*pbest*) e siga em direção a melhor posição encontrada até o momento (*gbest*). O algoritmo segue modificando a posição das partículas de acordo com o operador de velocidade aleatoriamente escolhido. Ao final, as probabilidades são atualizadas.

```
procedimento PSO_TSP{
  Defina probabilidades iniciais para os movimentos:
    p1 ← x          // seguir o próprio caminho
    p2 ← y          // seguir pbest
    p3 ← z          // seguir gbest
  Inicie a população de partículas
  Faça
    Para cada partícula p faça
      Avalie a partícula p
      Se a adequação de p é melhor que a adequação de pbest então
        pbest ← p
    Fim_Para
    Defina a partícula de melhor adequação de todas como gbest
    Para cada partícula p faça
      Escolha a velocidade de p
      Atualize a posição de p
    Fim_Para
    Atualize as probabilidades:
      p1 ← p1× 0.95
      p2 ← p2×1.01
      p3 ← 100% - (p1+p2)
  Enquanto (critério de parada) não é satisfeito
}
```

Figura 14: Esquema geral do algoritmo PSO-S (Souza, 2006)

As partículas são iniciadas com uma versão aleatória adaptada da heurística do vizinho mais próximo (Bellmore e Nemhauser, 1968). O procedimento é similar à fase construtiva de um algoritmo GRASP onde, a primeira cidade é escolhida aleatoriamente e as outras cidades são adicionadas à solução em cada passo. Além disso, uma lista de cidades candidatas é construída com 5% das cidades mais próximas à última inserida na solução. Uma cidade é escolhida aleatoriamente dessa lista e então adicionada à solução. Este passo se repete até que uma solução para o PCV esteja completa.

Capítulo 4

4. Análise de Sobrevivência

A análise experimental tem sido a principal ferramenta utilizada por pesquisadores na comparação de seus algoritmos heurísticos com outras abordagens. No entanto, a análise empírica tem sido aplicada às heurísticas por décadas e, somente recentemente, alguns trabalhos têm focado a definição de métodos e ferramentas para o projeto de experimentos na comparação de heurísticas (Rardin e Uzsoy, 2001; Johnson e McGeoch, 2002).

A Análise de Sobrevivência (AS), auxiliada pelo aprimoramento de técnicas estatísticas e pelo uso de computadores com alto poder de processamento, tem se destacado como uma das áreas da estatística que mais tem crescido.

A AS consiste de um conjunto de técnicas e modelos estatísticos apropriados ao estudo de variáveis observadas conforme a ocorrência de um determinado evento como, por exemplo, tempo de vida restante a um paciente, o efeito de doenças, o efeito de medicações ou mesmo o número de iterações em um algoritmo evolucionário até a ocorrência de sucesso. Quando se trata de pesquisa industrial o termo mais apropriado é Análise de Qualidade ou Análise de Confiabilidade e o estudo em questão pode ser, por exemplo, o tempo até a necessidade de um reparo, tempo observado até que ocorra uma falha num equipamento eletrônico ou o tempo de utilização de uma peça até que ocorra seu desgaste. Além disso, essa metodologia é aplicada, também, na economia, conhecida por Análise de Duração ou Análise de Transição, e em sociologia como Análise de Eventos Históricos. A literatura estatística, em geral, bem como os *softwares* estatísticos apropriados para essa análise utilizam termos associados à denominação Análise de Sobrevivência, como, por exemplo: função de sobrevivência, risco de morte e tábua de vida.

Obtendo um número r de execuções independentes de um algoritmo A, e um limite T para o tempo de processamento, a solução ótima será encontrada por A em q execuções, $q \leq r$, com tempo de processamento menor que T. Seja $s = r - q$, então em s execuções A não encontra o ótimo no tempo limite. As s execuções (ou observações) contêm somente uma informação parcial sobre a variável de interesse. Em todo o caso, esta informação parcial não é completamente descartada com o uso da Análise de Sobrevivência. Essas observações são chamadas "censuradas".

Allison (1995) destaca pelo menos duas características importantes inerentes à Análise de Sobrevivência:

i) a distribuição da variável resposta, que é, geralmente, assimétrica; e

ii) a presença do que se denomina *censura*, que acontece quando o registro da variável resposta não pôde ser realizado para certos indivíduos.

Tábua de vida, regressão exponencial, regressão logística (log-normal) e o estimador da Kaplan-Meier (KM) da função de sobrevivência são alguns dos métodos descritivos para estimação da distribuição do tempo de sobrevivência (Allison, 1995). Neste trabalho, foi utilizado o estimador Kaplan-Meier (1958) (KM).

Dessa forma, a função de sobrevivência pode ser definida como:

$$S(t) = P\ (\ T > t)$$

Essa função refere-se à probabilidade de que o tempo de máquina, necessário para a obtenção da solução desejada, do problema em questão, ultrapasse um determinado tempo *T*. No entanto, um melhor enfoque é dado a partir da função de distribuição acumulada, dada por:

$$F(t) = P(T \leq t) = 1 - S(t)$$

onde *F(t)* é a probabilidade de que o tempo de máquina seja inferior a um determinado tempo *T*. Ou seja, sob o ponto de vista de que, quanto menor for o tempo de máquina envolvido nessa solução, melhor é o desempenho do algoritmo, deseja-se que *F(t)* seja alta para um tempo *t* curto e baixa para um tempo *t* longo.

A análise desenvolvida trata da censura à direita que ocorre pela impossibilidade em se registrar o momento em que a solução desejada é obtida, em função de se ter um limite de tempo para a execução do algoritmo. Dessa forma, diz-se que o dado foi censurado no tempo limite estabelecido, através de um corte à direita na escala horizontal na qual o tempo está representado. Para indicar que uma observação está censurada, utiliza-se o símbolo "+" ao lado do valor observado (Ramos, 2005).

Para dados não censurados, o estimador KM da função de sobrevivência $\hat{S}(t)$ é dado, simplesmente, pela proporção amostral do número de observações com um tempo maior do que *t*. Se uma observação é censurada, o estimador KM não é definido. Obviamente, quanto menor é o tempo de execução de um algoritmo para atingir a solução ótima, melhores são os resultados desse algoritmo. Assim, quanto mais uma "curva" se localizar à direita do gráfico, piores são os resultados do algoritmo que gera essa "curva".

Como as funções de sobrevivência para os tempos de processamento dos algoritmos investigados não são conhecidas, um teste de hipótese é feito para verificar a igualdade entre essas funções. Neste trabalho, o teste de hipótese Log-Rank (Mantel e Haenszel, 1959) é aplicado. A estatística Log-Rank (*L)* tem uma distribuição assintótica de Qui-quadrado com um grau de liberalidade (Collett, 1991) e o *p-valor* associado possibilita o teste da hipótese de igualdade entre as funções de sobrevivência dos grupos examinados. Se $P(\chi_1^2 > L) < 0{,}05$, onde χ_1^2 é a distribuição Qui-quadrado, então a hipótese de igualdade é rejeitada ao nível de significância de 5%. Para o cálculo da estatística Log-Rank, utiliza-se a equação dada por:

$$L = \frac{\left[\sum_{j=1}^{k}(d_{1j} - \frac{n_{1j}d_j}{n_j})\right]^2}{\sum_{j=1}^{k}\frac{n_{1j}n_{2j}d_j(n_j - d_j)}{n_j^2(n_j - 1)}}$$

onde d_{ij} é número de "mortes" relativo ao algoritmo *i* no tempo t_j; n_{ij} é número de indivíduos em risco de "morte" relativo ao algoritmo *i* no tempo t_j; d_j é número total de "mortes" no tempo t_j e n_j é número total de indivíduos em risco de "morte" no tempo t_j. Exemplos da aplicação da Análise de Sobrevivência usando o teste de Log-Rank extraídos de Ramos (2005) são apresentados abaixo.

Sejam os tempos (em segundos) registrados em 30 execuções de um algoritmo A (ALG-A), para uma instância *I* do PCV, e suas respectivas freqüências dadas na Tabela 5. Nas duas últimas colunas, encontram-se as estimativas da função de sobrevivência pelo método de Kaplan-Meier e da função de distribuição acumulada, calculadas como explicado a seguir.

Tabela 5: Função de Sobrevivência estimada – ALG-A, instância *I*

t(s)	Freqüência	$\hat{S}(t)$	$1-\hat{S}(t)$
0	0	1	0
0,64	1	0,967	0,033
0,79	1	0,933	0,067
0,81	1	0,9	0,1
0,88	1	0,867	0,133
0,91	1	0,833	0,167
0,97	1	0,767	0,233
1	2	0,733	0,267
1,1	1	0,667	0,333
1,12	2	0,633	0,367
1,13	1	0,6	0,4
1,42	1	0,567	0,433
1,5	1	0,533	0,467
1,51	1	0,5	0,5
1,69	1	0,467	0,533
1,7	1	0,433	0,567
1,79	1	0,367	0,633
1,96	2	0,333	0,667
1,98	1	0,3	0,7
2,06	1	0,267	0,733
2,22	1	0,2	0,8
2,3	2	0,167	0,833
2,99	1	0,133	0,867
3,42	1	0,1	0,9
3,83	1	0,067	0,933
6,13	1	0,033	0,967
12,00+	1	-	-

Como foi mencionado anteriormente, para dados não censurados, o estimador de Kaplan-Meier da função de sobrevivência $\hat{S}(t)$ é dado pela proporção amostral do número de observações com um tempo maior do que *t*, e a estimativa de $F(t)$ ($\hat{F}(t)$) é dada por $1- \hat{S}(t)$. Dessa forma:

$$\hat{S}(0,64) = 29/30 = 0,967 \quad e \quad \hat{F}(0,64) = 1 - \hat{S}(0,64) = 0,033$$

$$\hat{S}(1,12) = 19/30 = 0,633 \quad e \quad \hat{F}(1,12) = 1 - \hat{S}(1,12) = 0,367$$

$$\hat{S}(6,13) = 1/30 = 0,033 \quad e \quad \hat{F}(6,13) = 1 - \hat{S}(6,13) = 0,033$$

Se uma observação é censurada, o estimador de Kaplan-Meier da Função de Sobrevivência não é definido. Assim, como o algoritmo não encontrou a solução ótima até 12 segundos, a observação foi censurada (simbolizada por 12,00+) e não há informação para $\hat{S}(t)$.

Para o cálculo da estatística Log-Rank (L), sejam os tempos obtidos em trinta execuções dos algoritmos A (ALG-A) e B (ALG-B) para a instância I, e os dados necessários ao cálculo da estatística Log-Rank como mostra a Tabela 6.

Alguns cálculos iniciais auxiliam no cálculo de L: para cada tempo t_k (dispostos em ordem crescente), calculam-se, os valores esperados e a variância de d_{1j} (resultados das duas últimas colunas da Tabela 6), dados, respectivamente, por:

$$\text{valor\_esperado}(d_{1j}) = \frac{n_{1j} d_j}{n_j} \quad e \quad \textit{Variância}(d_{1j}) = \frac{n_{1j} n_{2j} d_j (n_j - d_j)}{n_j^2 (n_j - 1)}$$

Tabela 6: Teste Log-Rank - Tempo de execução

Tempo	ALG-A		ALG-B		Total		Valor	Variância
$t_j(S)$	d_{1j}	n_j	d_{2j}	n_{2j}	d_j	n_j	Esperado	
1,98	1	30	0	30	1	60	0,5	0,25
2,86	0	29	1	30	1	59	0,492	0,25
3,26	1	29	0	29	1	58	0,5	0,25
3,76	1	28	0	29	1	57	0,491	0,25
4,56	1	27	0	29	1	56	0,482	0,25
4,58	1	26	0	29	1	55	0,473	0,249
4,8	1	25	0	29	1	54	0,463	0,249
5,3	1	24	0	29	1	53	0,453	0,248
6,32	1	23	0	29	1	52	0,442	0,247
6,67	1	22	0	29	1	51	0,431	0,245
6,68	1	21	0	29	1	50	0,42	0,244
7,96	1	20	0	29	1	49	0,408	0,242
13,4	1	19	0	29	1	48	0,396	0,239
15,44	1	18	0	29	1	47	0,383	0,236
16,39	1	17	0	29	1	46	0,37	0,233
16,6	0	16	1	29	1	45	0,356	0,229
16,7	0	16	1	28	1	44	0,364	0,231
18,05	1	16	0	27	1	43	0,372	0,234
18,32	1	15	0	27	1	42	0,357	0,23
30,4	0	14	1	27	1	41	0,341	0,225
30,5	0	14	1	26	1	40	0,35	0,228
30,7	0	14	1	25	1	39	0,359	0,23
32,47	1	14	0	24	1	38	0,368	0,233
37,3	1	13	0	24	1	37	0,351	0,228
43,06	1	12	0	24	1	36	0,333	0,222
44,27	1	11	0	24	1	35	0,314	0,216
50,12	1	10	0	24	1	34	0,294	0,208
56,23	1	9	0	24	1	33	0,273	0,198
58	0	8	1	24	1	32	0,25	0,188
58,1	0	8	1	23	1	31	0,258	0,191
76,37	1	8	0	22	1	30	0,267	0,196
110,49	1	7	0	22	1	29	0,241	0,183
128	0	6	1	22	1	28	0,214	0,168
Σ -	24	-	-	-	-	-	12,367	7,517

Assim, os resultados de L e do *p-valor* para o teste da hipótese de que as funções de sobrevivência, relativas aos algoritmos ALG-A e ALG-B são iguais, para a instância *I*, com base nos valores da Tabela 6, são dados, respectivamente, por:

$$L = \frac{(24 - 12{,}367)^2}{7{,}517} = 18{,}00 \quad e \quad P(\chi^2 > 18{,}00) = 0{,}00$$

Daí, conclui-se que a hipótese de igualdade entre as funções de sobrevivência é rejeitada ao nível de significância de 5%.

Finalmente, a Análise de Sobrevivência utilizando o estimador de Kaplan-Meier, juntamente com o teste de Log-Rank para o teste de hipótese de igualdades entre as funções de sobrevivência, foi utilizada neste trabalho para o estudo do tempo computacional de abordagens heurísticas aplicadas ao Problema do Caixeiro Viajante.

Capítulo 5

5. Um Algoritmo Memético para o PCV

O sucesso dos AMs se deve à junção das habilidades dos AGs com as habilidades da busca local, combinando a adaptação evolucionária da população com os conhecimentos adquiridos pelo indivíduo durante sua existência (Krasnogor e Smith, 2000). Com base nesse princípio, foi desenvolvido um algoritmo memético para o Problema do Caixeiro Viajante que utiliza o algoritmo LK de Applegate *et al.*(1999b) como procedimento de busca local. O algoritmo desenvolvido se aplica tanto ao PCV simétrico quanto ao assimétrico, sendo que, para o caso assimétrico foi utilizado um método para a transformação das instâncias em instâncias simétricas. Essa transformação se fez necessária porque o algoritmo de Lin-Kernighan, originalmente, não se aplica a instâncias assimétricas.

```
AlgoritmoMemeticoPCVS
  InicializaPopulacao(Pop);
  Para cada indivíduo pi da População{ BuscaLocalInversao(pi) };
  melhorIndividuo ← MelhorIndividuo(Pop);
  Enquanto criterioParada ← false{
    Seleciona dois indivíduos p1 e p2 dentre os 50% melhores;
    f1 e f2 ← Cruzamento(p1,p2);
    Com certa probabilidade faz-se a mutação em f1 e f2;
    BuscaLocalLinKernighan(f1);
    BuscaLocalLinKernighan(f2);
    Seleciona dois indivíduos p1 e p2 dentre os 50% piores;
    Se Custo(p1) > Custo(f1) então substitui p1 por f1
    Se Custo(p2) > Custo(f2) então substitui p2 por f2
    Se MelhorIndividuo(Pop) for melhor que o melhorIndividuo)
      melhorIndividuo ← MelhorIndividuo(Pop);
    Se melhorIndividuo = otimoGlobal
      criterioParada ← true;
```

Figura 15: Esquema do algoritmo memético desenvolvido

A população inicial é gerada utilizando o algoritmo do Vizinho Mais Próximo (VMP) da seguinte maneira. Enquanto a população não estiver completa (em relação ao tamanho determinado para a população), uma cidade *c* nunca selecionada anteriormente, é escolhida aleatoriamente para iniciar um percurso *p*. A partir de *c*, o percurso *p* é construído através do método VMP. Depois de completo, *p* é avaliado e faz-se uma tentativa de introduzir *p* na população *P* da seguinte forma: se existir um percurso *q* pertencente a *P* com custo igual ao custo de *p*, uma solução aleatória é atribuía a *p* e tenta-se, novamente, introduzir *p* na população; caso contrário, *p* é introduzido na população e o processo reinicia com uma nova cidade inicial *c* escolhida. As tentativas de introduzir *p* na população são repetidas até a obtenção do sucesso.

Depois de completa, a população é avaliada, aplicando-se, sobre cada indivíduo, a busca local por Inversão. Esta estrutura de vizinhança foi utilizada nessa fase do algoritmo porque que tem um baixo custo em termos de tempo computacional e proporciona soluções de boa qualidade.

A estrutura de vizinhança por *Inversão* é a baseada na idéia apresentada por Holland (1975) e atua invertendo um determinado *sub-percurso*, buscando soluções de melhor qualidade. A Figura 16 ilustra o funcionamento desta estrutura.

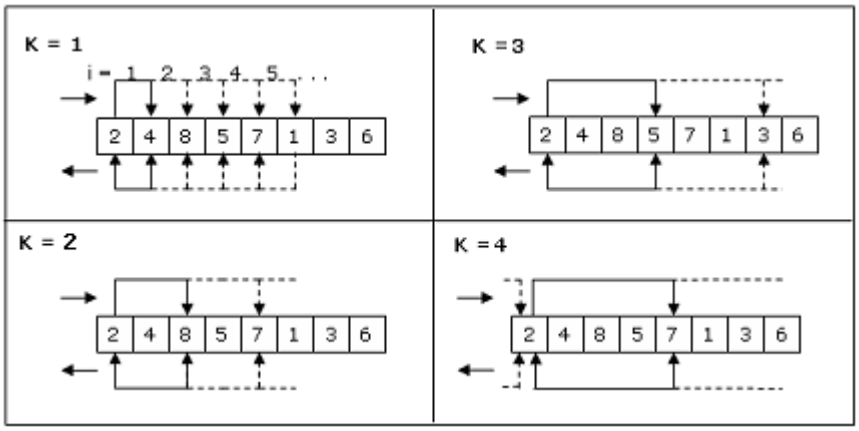

Figura 16: Vizinhança por Inversão – Representação por permutação

A Figura 16 ilustra os movimentos que seriam testados para diferentes valores de *k* onde este representa o tamanho do percurso a ser invertido. No primeiro quadrante (*k=1*), o intervalo a ser invertido é composto por apenas duas cidades, para cada valor de *i*. No quadrante em que *k = 2*, o intervalo a ser invertido compreende agora 3 cidades e no quadrante seguinte 4 cidades e assim por diante. Quando a inversão de determinado *sub-percurso* produzir uma solução vizinha de melhor qualidade, esta passa a ser a nova solução corrente. O processo é repetido até que não sejam mais encontrados sub-percursos capazes de melhorar a solução atual. Após a realização da busca por Inversão, a população é avaliada e ordenada utilizando o método *QuickSort* (população *ranqueada*).

No passo seguinte é feita a seleção dos pais para a próxima geração escolhendo-se, aleatoriamente, dois indivíduos distintos (avaliado pela função de custo) p_1 e p_2 dentre os 50% melhores dessa população, ou seja, para uma população ranqueada de tamanho 6, são sorteados dois pais dentre os indivíduos 1,2 e 3. Esse esquema de seleção exige que o tamanho da população seja maior ou igual a 4.

O cruzamento dos indivíduos selecionados, p_1 e p_2, (feito através da aplicação de algum operador de cruzamento) resulta na geração de dois novos indivíduos f_1 e f_2. Com uma probabilidade de 8%, é aplicado, sobre os indivíduos f_1 e f_2,um determinado operador de mutação. Depois do cruzamento e de uma possível mutação, é aplicado, para uma maior refinamento da solução, o algoritmo LK-ABCC nos indivíduos f_1 e f_2 resultantes.

O processo de introdução dos novos indivíduos na população é realizado através de torneio, da seguinte maneira: um indivíduo dentre os 50% piores da população é selecionado para competição e, se este tiver um custo maior do que o indivíduo que está se tentando introduzir, é substituído pelo novo indivíduo. O processo é feito para os dois indivíduos resultantes do cruzamento. Caso algum novo indivíduo seja introduzido na população, faz-se um *reordenamento* simples onde apenas o novo indivíduo é ajustado, uma vez que a população estava ordenada antes de sua inclusão na mesma. Com a introdução ou não, de novos indivíduos na população, o ciclo evolutivo é concluído. O processo evolutivo é repetido até que a solução ótima seja obtida ou o critério de parada seja atendido. Nesse algoritmo, o critério de parada utilizado foi o tempo máximo de execução.

Originalmente, o algoritmo LK foi desenvolvido apenas para instâncias simétricas e para que este pudesse ser utilizado como procedimento de busca local em instâncias assimétricas, foi necessária a utilização de um procedimento para a transformação de instâncias assimétricas. Para essa transformação foi utilizado o método de Jonker e Volgenant (1983) que transforma um problema assimétrico de tamanho n num problema simétrico de tamanho $2n$, como descrito abaixo.

Seja $C = (c_{ij})$ a matriz de custo $n \times n$ do problema assimétrico e $C'_{ij} = (c'_{ij})$ a matriz simétrica de dimensão $2n \times 2n$, referente ao mesmo problema, calculada da seguinte forma:

$$C'_{n+i,j} = C'_{j,n+i} = c_{ij} \qquad \text{para } i = 1,2,\ldots n, \quad j = 1,2,\ldots, n \quad e \quad i \neq j$$

$$C'_{n+i,i} = C'_{i,n+i} = -M \qquad \text{para } i = 1,2, \ldots,n$$

$$C'_{ij} = M \qquad \text{caso contrário,}$$

onde M é um valor suficientemente grande como, por exemplo, $M = Max\ (C_{ij})$.

Sendo assim, quando uma instância é assimétrica, o AM_PCVA cria uma instância simétrica correspondente e, a cada iteração, antes de realizar a busca local, transforma a solução assimétrica numa solução simétrica correspondente. Essa solução simétrica é, então, fornecida ao algoritmo LK utilizado para realização da busca local. De maneira semelhante, a solução simétrica retornada pela busca local, é transformada numa solução assimétrica correspondente e o algoritmo prossegue com esta nova solução localmente ótima.

Além do ajuste para a realização da busca local, os operadores genéticos utilizados no AM_PCVA também foram modificados. Essas modificações se referem, principalmente, ao cálculo da melhoria obtida a partir de determinado movimento, considerando que a distância de uma cidade B para A pode ser diferente da distância de A para B.

5.1. Experimentos Computacionais

Em função da diversidade de operadores genéticos, foram testadas várias versões do algoritmo memético utilizando diferentes combinações de operadores de cruzamento e mutação. O objetivo dos experimentos reportados nessa seção é determinar quais versões do AM_PCVS e AM_PCVA apresentam o melhor desempenho.

Os operadores genéticos que deram origem às versões apresentadas foram pré-selecionados a partir dos testes reportados em Larragna *et al*. (1999) e Ozcan e Erenturk (2004). Nesses experimentos, tanto os operadores de cruzamento PMX e OX1, quanto os operadores de mutação SIM e ISM, aparecem entre aqueles que proporcionam soluções de melhor qualidade. Dessa forma, foram comparadas as seguintes versões simétricas do algoritmo memético proposto:

- AM_PCVS-PS – cruzamento PMX e mutação SIM;
- AM_PCVS-PI – cruzamento PMX e mutação ISM;
- AM_PCVS-OS – cruzamento OX1 e mutação SIM;
- AM_PCVS-OI – cruzamento OX1 e mutação ISM;

Além das versões para o PCV simétrico, foram comparadas, também, as seguintes versões para o PCV assimétrico:

- AM_PCVA-PS – cruzamento PMX e mutação SIM;
- AM_PCVA-PI – cruzamento PMX e mutação ISM;
- AM_PCVA-OS – cruzamento OX1 e mutação SIM;
- AM_PCVA-OI – cruzamento OX1 e mutação ISM;

Com o objetivo de obter o melhor desempenho, cada versão, tanto simétrica como assimétrica, do algoritmo proposto foi testada com o tamanho da população variando entre 10, 20, 50 e 100 indivíduos.

Para as versões simétricas, foram utilizadas 20 instâncias da biblioteca TSPLIB (Reinel, 1990) com número de cidades entre 1000 e 2392: dsj1000, pr1002, u1060, vm1084, pcb1173, d1291, rl1304, rl1323, nrw1379, fl1400, u1432, fl1577, d1655, vm1748, u1817, rl1889, d2103, u2152, u2319 e pr2392. Essas instâncias foram utilizadas por serem não triviais permitindo uma boa exploração dos algoritmos.

Para as instâncias assimétricas, foram realizados testes com as 19 instâncias da biblioteca TSPLIB com número de cidades entre 17 e 443: br17, ftv33, ftv38, p43, ftv44, ftv47, ry48p, ft53, ftv55, ftv64, ft70, ftv70, kto124p, ftv170, rbg323, rbg358, rbg403 e rbg443.

Nesse experimento, foi utilizado um computador com processador Pentium IV 2.8 GHz, 512 MB de memória RAM e Sistema Operacional Windows XP Professional. Foram realizadas 30 execuções independentes para cada algoritmo, tendo como critério de parada a obtenção da solução ótima ou o tempo máximo de execução de 300 segundos.

O desempenho de cada versão foi avaliado, principalmente, em relação à média da taxa de afastamento em relação à solução ótima calculada pela expressão:

$$\textit{taxa de afastamento} = (\textit{valor} - \textit{otimo}) / \textit{otimo},$$

onde *valor* e *ótimo* correspondem a média da melhor solução obtida em N execuções e ao valor da solução ótima, respectivamente.

Da combinação das versões simétricas com os diferentes tamanhos de população, as versões que obtiveram os melhores resultados foram:

- AM_PCVS-OI com população de tamanho 10;
- AM_PCVS-PS com população de tamanho 10;
- AM_PCVS-PI com população de tamanho 20;
- AM_PCVS-OS com população de tamanho 20;

A Tabela 7 apresenta, para as quatro melhores versões, o tempo médio gasto pelo algoritmo até a obtenção de sua melhor solução (não necessariamente a solução ótima), T(s); e a média da taxa de afastamento em relação ao ótimo (Média) para 30 instâncias.

Tabela 7: Ajuste de Parâmetros AM_PCVS – Tempo médio e média de afastamento

INST	AM_PCVS-OI (População 10)		AM_PCVS-PS (População 10)		AM_PCVS-PI (População 20)		AM_PCVS-OS (População 20)	
	T(s)	Média	T(s)	Média	T(s)	Média	T(s)	Média
dsj1000	101,62	0,0005	202,76	0,000431	204,23	0,000017	157,16	0,000513
pr1002	36,35	0	38,28	0	48,18	0	44,04	0
u1060	123,05	0,000051	140	0,000017	128,94	0,000017	151,54	0
vm1084	59,38	0,000087	133,41	0,000043	150,08	0,000023	130,27	0,000022
pcb1173	101,53	0,000002	108,97	0	81,78	0	83,29	0
d1291	88,50	0,000171	108,47	0	184,48	0	143,27	0
rl1304	113,55	0,000171	150,24	0	141	0,000034	167,2	0,000138
rl1323	127,68	0,00004	151,05	0,000097	142,19	0,000217	156,61	0,000056
nrw1379	217,50	0,000122	236,43	0,000025	210,73	0,000095	224,16	0,000106
fl1400	69,94	0	91	0	97,97	0	112,99	0
u1432	120,52	0,000049	157,8	0	172,82	0,000048	136,34	0,000092
fl1577	217,29	0,004162	237,07	0,002342	206,48	0,004234	202,27	0,002342
d1655	146,18	0,00061	173,98	0	214,93	0,000003	218,6	0,000029
vm1748	168,49	0,000305	232,66	0,000137	278,21	0,00023	213,43	0,000175
u1817	187,26	0,000969	252,25	0,001262	237,82	0,001374	252,62	0,00107
rl1889	194,49	0,000655	257,78	0,000145	263,66	0,000353	270,38	0,000686
d2103	226,38	0,006772	261,33	0,007292	284,61	0,006839	277,66	0,006674
u2152	224,79	0,000868	266,7	0,000628	277	0,001434	281,36	0,001863
u2319	152,80	0,000948	241,63	0,000773	259,77	0,000871	268,37	0,000878
pr2392	284,15	0,00066	298,34	0	299,21	0,001182	297,52	0,001916
Média	148,07	0,000857	187,01	0,000660	194,20	0,000849	189,45	0,000828

A Tabela 7 mostra que, embora o melhor tempo médio tenha sido apresentado pelo AM_PCVS-OI, a versão que obteve as melhores soluções foi utilizando a configuração AM_PCVS-PS que combina o operador de cruzamento PMX com o operador SIM para mutação e população de tamanho 10.

Para as instâncias assimétricas, todas as versões testadas obtiveram a solução ótima em todas as execuções para as instâncias de até 170 cidades. Considerando as médias do tempo gasto até a obtenção da melhor solução e a média da taxa de afastamento em relação a solução ótima, as versões que apresentaram os melhores resultados foram:

- AM_PCVA-OS com população de tamanho 10;
- AM_PCVA-PI com população de tamanho 10;
- AM_PCVA-OI com população de tamanho 10;
- AM_PCVA-OI com população de tamanho 20;

Tabela 8: Ajuste de Parâmetros AM_PCVA – Tempo médio e média de afastamento

INST	AM_PCVA-OS (População 10)		AM_PCVA-PI (População 10)		AM_PCVA-OI (População 10)		AM_PCVA-OI (População 20)	
	T(s)	Média	T(s)	Média	T(s)	Média	T(s)	Média
br17	0,00	0	0,00	0	0,13	0	0,00	0
ftv33	0,42	0	0,30	0	0,31	0	0,62	0
ftv35	0,96	0	0,94	0	2,79	0	0,90	0
ftv38	6,69	0	6,70	0	6,39	0	5,47	0
p43	1,35	0	2,58	0	3,87	0	2,14	0
ftv44	1,69	0	1,81	0	2,02	0	1,18	0
ftv47	0,58	0	0,41	0	0,49	0	0,53	0
ry48p	7,30	0	3,68	0	6,46	0	4,22	0
ft53	7,89	0	6,20	0	4,27	0	7,48	0
ftv55	0,41	0	0,44	0	0,26	0	0,40	0
ftv64	0,74	0	1,29	0	0,74	0	0,87	0
ft70	15,28	0	6,22	0	5,98	0	10,38	0
ftv70	1,02	0	1,04	0	1,03	0	1,11	0
kro124p	5,22	0	4,01	0	4,93	0	8,42	0
ftv170	9,61	0	12,60	0	10,10	0	11,61	0
rbg323	298,35	0,013273	289,33	0,013198	298,58	0,010784	293,38	0,014480
rbg358	297,69	0,022098	251,27	0,020206	294,83	0,019003	271,92	0,023044
rbg403	211,50	0,001907	279,98	0,001947	222,99	0,001501	282,99	0,001866
rbg443	289,40	0,001544	298,91	0,001471	225,23	0,001177	259,21	0,001618
Médias	60,85	0,002043	61,46	0,001938	57,42	0,001709	61,20	0,002158

A Tabela 8 mostra que, embora a diferença não grande, o melhor tempo médio é apresentado pela versão AM_PCVA-OI que apresenta, também, a melhor média em relação à qualidade das soluções (taxa de afastamento). A versão AM_PCVA-OI combina o operador de cruzamento OX1 com o operador de mutação ISM e população de tamanho 10.

Para os demais experimentos reportados nesse trabalho, as versões utilizadas pelos algoritmos AM_PCVS e AM_PCVA foram AM_PCVS-PS e AM_PCVA-OI, respectivamente.

Capítulo 6

6. Análise Experimental

A proposta deste trabalho é fazer uma análise experimental de abordagens heurísticas, aplicadas ao Problema do Caixeiro Viajante, utilizando a metodologia estatística conhecida como Análise de Sobrevivência (AS), auxiliada pelo teste de Log-Rank para o teste da hipótese de igualdade entre as funções de sobrevivência. Essa metodologia se aplica ao estudo do tempo, observado até a ocorrência de um determinado evento. Para a aplicação da AS no estudo de abordagens heurísticas, o evento considerado pode ser a obtenção de uma solução com determinada qualidade (taxa de afastamento em relação ao ótimo), avaliando-se o desempenho dessas abordagens, em relação ao tempo computacional.

Neste trabalho, como para o conjunto de instâncias utilizado, a solução ótima é conhecida, o evento considerado é a obtenção dessa solução ótima e a avaliação feita se refere aos algoritmos que têm maior probabilidade de encontrar o ótimo, antes de um determinado tempo computacional. Além disso, também está sendo investigada a utilização de um algoritmo LK como parte de um algoritmo evolucionário, nesse caso, de um algoritmo memético. Nesse aspecto, pretende-se avaliar a eficiência dessa hibridização com relação a outras abordagens evolucionárias.

Para uma melhor direcionamento e uma melhor compreensão do estudo realizado, os algoritmos utilizados nos experimentos foram divididos em três classes, de acordo com a característica de cada abordagem. A Classe I é composta por versões iterativas do algoritmo de Lin e Kernighan (1973) implementadas por diferentes autores. A Classe II compreende os Algoritmos Evolucionários e, portanto, estão incluídos os algoritmos meméticos e algoritmos transgenéticos. A Classe III é composta por algoritmos de Otimização por Nuvem de Partículas.

Na escolha das abordagens, procurou-se utilizar as mais recentes, que melhores resultados têm produzido (com base em outros trabalhos da literatura) e com disponibilidade de código-fonte e/ou executável. Além disso, buscou-se utilizar abordagens baseadas em diferentes "modelos" tais como Busca Local, Algoritmos Evolucionários e Otimização por Nuvem de Partículas.

Das várias implementações para o algoritmo de Lin-Kernighan, aplicadas na solução do PCV, foram utilizadas as seguintes versões iterativas: LK de Aplegate *et al.* (1999) (LK-ABCC), o LK de Helsgaun (2000) (LK-H) e o LK de Nguyen *et al*. (2006) (LK-NYYY). Esses algoritmos foram selecionados por serem, reconhecidamente, algumas das mais eficientes implementações do algoritmo LK com resultados reportados em DIMACS (2006).

Dos Algoritmos Evolucionários, aplicados ao PCV simétrico, foram utilizados o algoritmo memético de Krasnogor e Smith (2000) (AM_KRAS), o algoritmo transgenético ProtoG de Ramos (2005) e o algoritmo memético proposto neste trabalho (AM_PCVS). Para as instâncias assimétricas, foram, ainda, utilizados o algoritmo memético de Buriol *et al.* (2003) e o algoritmo memético proposto neste trabalho, na sua versão assimétrica (AM PCVA).

Como abordagens de Otimização por Nuvem de Partícula, foram utilizados os algoritmos de Machado e Lopes (2005) (PSO-ML) e o de Souza (2006) (PSO-S).

Para a realização dos experimentos, foram utilizadas 50 instâncias simétricas da biblioteca TSPLIB (Reinelt, 1990) dividas em dois grupos:

- instâncias pequenas com número de cidades entre 100 e 783: kroA100, kroB100, kroC100, KroD100, kroE100, lin105, pr107, pr124, ch130, pr136, pr144, ch150, kroA150, KroB150, pr152, u159, rat195, kroA200, kroB200, lin318, rd400, fl417, pr439, d493, u574, rat575, p654, d657, u724 e rat783.
- Instâncias médias com número de cidades entre 1000 e 2392: dsj1000, pr1002, u1060, vm1084, pcb1173, d1291, rl1304, rl1323, nrw1379, fl1400, u1432, fl1577, d1655, vm1748, u1817, rl1889, d2103, u2152, u2319 e pr2392.

Além das 50 instâncias simétricas, foram feitos experimentos com 19 instâncias assimétricas da biblioteca TSPLIB: br17, ft53, ft70, ftv33, ftv35, ftv38, ftv44, ftv47, ftv55, ftv64, ftv70, ftv170, kro124p, p43, rbg323, rbg358, rbg403, rbg443 e ry48p.

Para as abordagens pertencentes à Classe I, foram utilizadas as 20 instâncias de tamanho médio por serem instâncias não triviais, permitindo uma melhor exploração das potencialidades de cada abordagem. Para as Classes II e III foram utilizadas as 50 instâncias.

Foram realizadas 30 execuções de cada algoritmo para cada uma das instâncias. Para as instâncias pequenas foi dado um tempo máximo de 300 segundos. Para as instâncias de tamanho médio e para as assimétricas, o tempo máximo foi de 600s. O computador utilizado foi um Pentium IV 2.8 GHz com 512 MB de memória RAM e Sistema Operacional Windows XP Professional.

6.1. Classe I – Algoritmos Lin-Kernighan

Primeiramente, a Tabela 9 apresenta os resultados de um experimento convencional onde três grupos de resultados, para cada algoritmo, são apresentados: o tempo médio de processamento, em segundos (T), até a obtenção da melhor solução; a menor taxa de afastamento em relação à solução ótima (MIN) e a média da taxa de afastamento em relação à solução ótima (Média). A coluna INST lista o nome das instâncias utilizadas nos experimentos. Dos dados apresentados estão destacados em **negrito** os valores MIN, quando iguais a zero, e <u>sublinhados</u>, os valores da menor média da taxa de afastamento em relação à solução ótima.

Tabela 9: Algoritmos *Iterated LK* - Tempo médio e média de afastamento

INST	LK-ABCC			LK-NYYY			LK-H		
	T(s)	MIN	Média	T(s)	MIN	Média	T(s)	MIN	Média
dsj1000	100,51	0,000349	0,002211	27,32	0,000027	0,000371	24,39	**0**	0
pr1002	14,22	**0**	0	5,3	**0**	0,000491	0,99	**0**	0
u1060	152,42	**0**	0,000102	8,29	**0**	0,00022	100,31	**0**	0
vm1084	74,65	**0**	0,000179	3,63	0,0001	0,000209	21,89	**0**	0,000087
pcb1173	62,06	**0**	0,000032	3,46	**0**	0,000051	8,16	**0**	0
d1291	59,72	**0**	0,000919	8,92	**0**	0,000974	55,1	**0**	0
rl1304	47,14	**0**	0,000327	6,28	**0**	0,000311	10,32	**0**	0
rl1323	109,57	**0**	0,000131	9,21	**0**	0,000131	96,06	**0**	0,00003
nrw1379	253,63	**0**	0,000254	5,41	0,000071	0,000381	24,82	**0**	0
fl1400	73,84	**0**	0	32,89	**0**	0	125,62	0,001838	0,001928
u1432	155,66	**0**	0,000243	8,41	**0**	0,000307	1,37	**0**	0
fl1577	160,01	0,008602	0,008656	110,27	0,002027	0,002234	342,58	0,002027	0,002324
d1655	151,27	**0**	0,001276	28,16	**0**	0,000338	121,55	**0**	0,00001
vm1748	214,75	**0**	0,000659	16,55	**0**	0,000477	45,56	**0**	0
u1817	197,6	0,000559	0,001962	13,88	0,00042	0,001832	97,83	**0**	0,000626
rl1889	114,49	0,000736	0,00243	22,61	0,000013	0,00116	57,07	**0**	0
d2103	54,96	0,007667	0,008174	34,64	0,006279	0,008509	296,28	0,006579	0,006758
u2152	154,49	0,001105	0,001544	17,31	0,000887	0,003166	194,15	**0**	0,000089
u2319	281,04	0,00035	0,000408	17,59	0,00035	0,000618	2,015	**0**	0
pr2392	307,38	**0**	0,001232	15,18	0,000315	0,001116	19,81	**0**	0
Médias	136,97	0,000968	0,001537	19,77	0,000524	0,001145	82,29	0,000522	0,000593

A Tabela 9 mostra que o LK-H encontra a solução ótima para 17 (85%) das 20 instâncias testadas, enquanto que o LK-ABCC e o LK-NYYY encontram a solução ótima para 13 (65%) e 10 (50%) instâncias, respectivamente. Além disso, apenas o LK-H chegou ao ótimo nas instâncias dsj1000, u1817, rl1889, u2152 e u2319.

A performance média em relação à solução mínima encontrada pelos algoritmos LK-ABCC e LK-NYYY para todas as instâncias não é muito diferente. Essa é pequena até mesmo quando comparados estes com o LK-H. Dessa forma, em relação somente ao tempo de processamento, o melhor desempenho foi exibido pelo LK-NYYY, seguido pelo LK-H e LK-ABCC.

A Tabela 10 apresenta os dados relativos ao mesmo experimento porém com os tempos ajustados para a aplicação da Análise de Sobrevivência, ou seja, se a solução ótima não é encontrada antes de 600 segundos (tempo máximo), o tempo não é registrado. O objetivo nesse experimento é analisar a função de sobrevivência dada pela probabilidade que cada um dos algoritmos tem em encontrar a solução ótima até um tempo *T*. Dessa forma, as colunas da Tabela 10 mostram, para cada algoritmo, a Média(*t*) e Mediana(*t*) que correspondem, respectivamente, à média e mediana do tempo de processamento gasto até a obtenção da solução ótima. Além destes, é mostrado, também, a percentagem de soluções ótimas encontradas nas 30 execuções independentes (%Otm).

Tabela 10: *Iterated* LK - performance em relação ao tempo e soluções ótimas

INST	LK-ABCC			LK-NYYY			LK-H		
	Média(*t*)	Mediana(*t*)	%Otm	Média(*t*)	Mediana(*t*)	%Otm	Média(*t*)	Mediana(*t*)	%Otm
dsj1000	-	-	0	-	-	0	24,4	16,6	**100**
pr1002	14,2	9,4	**100**	-	6,6	80	1	0,8	**100**
u1060	-	-	50	-	-	10	100,3	80,5	**100**
vm1084	-	-	40	-	-	0	-	87,1	60
pcb1173	-	-	40	-	-	30	8,2	4,2	**100**
d1291	-	-	40	-	-	10	55,1	35,5	**100**
rl1304	-	71,4	80	-	5,8	70	10,3	5,2	**100**
rl1323	-	442,5	60	-	-	10	-	168,6	70
nrw1379	-	-	10	-	-	0	24,8	9,6	**100**
fl1400	73,8	54,9	**100**	32,9	22,6	**100**	-	-	0
u1432	-	-	50	-	-	40	1,4	1	**100**
fl1577	-	-	0	-	-	0	-	-	0
d1655	-	-	40	-	-	30	-	-	40
vm1748	-	-	10	-	-	0	45,6	40,6	**100**
u1817	-	-	0	-	-	0	-	-	10
rl1889	-	-	0	-	-	0	57,1	36	**100**
d2103	-	-	0	-	-	0	-	-	0
u2152	-	-	0	-	-	0	-	233,6	90
u2319	-	-	0	-	-	0	2	1,9	**100**
pr2392	-	-	20	-	-	0	19,8	18,5	**100**
Média			32			19			73,5

Em relação à obtenção da solução ótima, a Tabela 10 mostra que o LK-H apresenta o melhor desempenho chegando ao ótimo em 73,5% das execuções, seguido pelo LK-ABCC e LK-NYYY com 32% e 19% respectivamente.

Em relação ao tempo computacional, a média e a mediana dos tempos de execução do LK-H para a obtenção das soluções ótimas são, em muitos casos, menores que os tempos de execução dos outros dois algoritmos. Dessa forma, essa tabela mostra que o LK-H exibe uma performance superior aos outros algoritmos testados, exceto para a instância fl1400, onde o LK-ABCC e o LK-NYYY chegam ao ótimo em todas as execuções; e para as instâncias fl1577 e d2103 onde nenhum algoritmo chegou à solução ótima.

Finalmente, a Tabela 11 apresenta as estatísticas Log-Rank (L) e o p-valor (p) para o teste da hipótese de igualdade entre as funções de sobrevivência dos algoritmos testados. Em alguns casos esses valores são omitidos porque os algoritmos não geraram um número suficiente de observações (soluções ótimas) para que o teste pudesse ser aplicado. Nessa tabela, estão destacados em **negrito** os valores do *p-valor (p)* quando inferior a 0,05 indicando a rejeição da hipótese de igualdade a um nível de significância de 5%.

Tabela 11: Algoritmos *Iterated* LK - Teste de hipótese para instância pequenas

INST	LK-NYYY × LK-ABCC		LK-ABCC × LK-H		LK-H × LK-NYYY	
	L	*p*	*L*	*p*	*L*	*p*
dsj1000	-	-	4,26776	**0,00002**	4,26776	**0,00002**
pr1002	0,07609	0,93935	-3,20255	**0,00136**	-3,14929	**0,00164**
u1060	1,77541	0,07583	-2,78002	**0,00544**	-3,63769	**0,00028**
vm1084	2,17498	**0,02963**	1,09739	0,27247	2,83805	**0,00454**
pcb1173	0,26474	0,79121	3,40668	**0,00066**	2,95306	**0,00315**
d1291	1,40468	0,16012	2,74936	**0,00597**	3,63769	**0,00028**
rl1304	0,74232	0,45790	2,95725	**0,00310**	1,38710	0,16541
rl1323	2,13387	**0,03285**	0,42650	0,66993	2,49095	**0,01274**
nrw1379	-	-	4,07573	**0,00005**	-	-
fl1400	-1,56400	0,11782	-4,26776	**0,00002**	-4,26776	**0,00002**
u1432	-0,00336	0,99732	3,52180	**0,00043**	3,47019	**0,00052**
d1655	0,17784	0,85885	-0,14859	0,88188	0,17784	0,85885
vm1748	-	-	4,07573	**0,00005**	4,26776	**0,00002**
rl1889	-	-	4,26776	**0,00002**	4,26776	**0,00002**
u2152	-	-	3,88155	**0,00010**	3,88155	**0,00010**
u2319	-	-	4,26776	**0,00002**	4,26776	**0,00002**
pr2392	1,45244	0,14638	3,90944	**0,00009**	4,26776	**0,00002**

A Tabela 11 mostra que a hipótese nula é rejeitada para 14 das 17 comparações feitas entre LK-H e LK-ABCC. Dos 14 casos, o LK-H apresenta os melhores tempos computacionais para 13. O LK-ABCC teve melhor desempenho que o LK-H apenas na instância fl1400.

Comparando LK-H e o LK-NYYY tem-se um resultado similar onde a hipótese nula é rejeitada para 14 das 16 instâncias cujo teste pôde ser realizado. Mais uma vez, o LK-H apresenta os melhores tempos, perdendo apenas na instância fl1400.

Na comparação da função de sobrevivência entre LK-ABCC e LK-NYYY, a hipótese nula é rejeitada apenas para as instâncias vm1084 e rl1323 nas quais o LK-ABCC mostrou o melhor desempenho em relação ao tempo computacional. Para as outras 9 instâncias, para as quais o teste pôde ser realizado, os algoritmos apresentaram desempenho equivalente considerando um nível de significância de 5%.

Os dados mostram que, na maioria dos casos, o LK-H exibe uma performance superior aos outros algoritmos. Comparando LK-ABCC e LK-NYYY, o primeiro encontra um maior número de vezes a solução ótima para 12 instâncias e ambos encontram o mesmo número para 8 instâncias. Em média, o número de vezes em que a solução ótima é encontrada pelo LK-ABCC e LK-NYYY é 32 e 19, respectivamente. Sendo assim, de forma descritiva, o algoritmo LK-ABCC apresenta uma performance superior quando comparado ao LK-NYYY, no entanto, uma análise inferencial (dados da Tabela 11) não comprova esse resultado, exceto para as instâncias vm1084 e rl1323.

6.1.1.1. Tempo de Pré-processamento

Nos experimentos apresentados, o LK-H, de uma forma geral, se mostrou mais eficiente que os demais algoritmos, chegando à solução ótima em quase todas as instâncias e num tempo computacional bastante baixo. No entanto, é importante destacar que nestes

experimentos o tempo computacional para o pré-processamento dos algoritmos utilizado na inicialização e na geração da solução inicial não foi computado. A Tabela 12 apresenta o tempo gasto por cada algoritmo desde a inicialização até a conclusão da primeira iteração.

Tabela 12: Tempo para inicialização dos algoritmos e construção da solução inicial

INST	TEMPO (s)		
	LK-H	LK-ABCC	LK-NYYY
dsj1000	4,687	0,219	0,344
pr1002	5,797	0,156	0,218
u1060	4,844	0,156	0,266
vm1084	5,062	0,156	0,172
pcb1173	6,688	0,157	0,172
d1291	6,969	0,156	0,312
rl1304	8,797	0,172	0,266
rl1323	8,812	0,140	0,219
nrw1379	9,078	0,172	0,187
fl1400	8,781	0,250	1,328
u1432	8,579	0,172	0,266
fl1577	10,812	0,188	0,687
d1655	12,641	0,187	0,422
vm1748	15,484	0,203	0,360
u1817	13,813	0,188	0,281
rl1889	16,500	0,187	0,406
d2103	24,250	0,188	0,594
u2152	23,312	0,187	0,265
u2319	27,844	0,313	0,250
pr2392	32,453	0,218	0,360
Total (s)	255,203	3,765	7,375

Como pode ser observado na Tabela 12, o tempo que o LK-H necessita para o pré-processamento é bem superior aos demais algoritmos. Somando o tempo necessário para cada uma das instâncias, o LK-H, somente nesta etapa, necessita de mais de 255 segundos, enquanto que o LK-ABCC, o mais econômico, necessita de 3,765 e o LKNYYY 7,375 segundos.

O tempo computacional necessário para inicializar o algoritmo LK-H, cresce proporcionalmente com o número de cidades de cada instância. No entanto, foi observado que o LK-H utiliza esse tempo apenas na primeira iteração e, portanto, quanto maior o número de iterações realizadas pelo algoritmo, menos significativo será esse custo inicial. Por esse motivo, o tempo do pré-processamento foi desconsiderado para os experimentos reportados nesse trabalho.

6.2. Classe II - Algoritmos Evolucionários

Nesta seção são apresentados os experimentos computacionais realizados com o objetivo de avaliar o desempenho dos seguintes algoritmos evolucionários: o AM_PCVS, apresentado nesse trabalho; o algoritmo memético de Krasnogor e Smith (2000); e o algoritmo transgenético ProtoG (Ramos, 2005).

Como o número de instâncias simétricas disponíveis é bem maior que o número de instâncias assimétricas, nos experimentos com as simétricas foram utilizados dois grupos de

instâncias para que se pudesse analisar melhor o tempo computacional (nesse ponto as instâncias menores tendem a possibilitar uma melhor análise) e também analisar as potencialidades de cada abordagem aplicando-as nas instâncias de médio porte.

6.2.1. Instâncias Simétricas

A Tabela 13 apresenta uma análise "convencional" onde são mostrados três grupos de valores: a média do tempo computacional gasto até a obtenção da melhor solução T(s), a menor taxa de afastamento em relação ao ótimo (MIN) e a média da taxa de afastamento em relação à solução ótima (Média). Destacados em **negrito** estão os valores MIN quando iguais a zero, ou seja, situações em que a solução ótima foi encontrada. Sublinhados estão os valores que correspondem a melhor Média da taxa de afastamento em relação a solução ótima. A Tabela 14 apresenta as mesmas informações porém para as instâncias com mais de 1000 cidades (instâncias de médio porte).

Tabela 13: Algoritmos Evolucionários - Tempo médio e média de afastamento (Instâncias Pequenas)

INST	AM_KRAS			ProtoG			AM_PCVS		
	T(s)	MIN	Média	T(s)	MIN	Média	T(s)	MIN	Média
kroA100	139,74	0,000470	0,021218	3,53	**0,0**	0,00	0,10	**0,00**	0,00
kroB100	163,47	0,003794	0,023414	27,22	**0,00**	0,00	0,19	**0,00**	0,00
kroC100	191,65	0,003374	0,014769	7,9	**0,00**	0,00	0,10	**0,00**	0,00
KroD100	158,04	0,005964	0,026684	6,46	**0,00**	0,00	0,11	**0,00**	0,00
kroE100	164,06	0,003127	0,013117	40,08	**0,00**	0,00	0,20	**0,00**	0,00
lin105	188,07	0,001530	0,017375	22,43	**0,00**	0,000077	0,11	**0,00**	0,00
pr107	196,01	0,004040	0,022129	17,08	**0,00**	0,00	0,19	**0,00**	0,00
pr124	145,81	0,000966	0,014436	4,7	**0,00**	0,00	0,40	**0,00**	0,00
ch130	208,67	0,010475	0,037780	178,16	**0,00**	0,001007	0,44	**0,00**	0,00
pr136	231,12	0,016472	0,048758	236,24	**0,00**	0,000736	0,24	**0,00**	0,00
pr144	150,06	0,005894	0,022794	6,2	**0,00**	0,00	0,35	**0,00**	0,00
ch150	196,09	0,010417	0,038286	183,41	**0,00**	0,000437	0,18	**0,00**	0,00
kroA150	227,53	0,015948	0,047360	200,39	**0,00**	0,000064	0,30	**0,00**	0,00
KroB150	233,88	0,013165	0,036905	261,2	**0,00**	0,000182	0,39	**0,00**	0,00
pr152	207,52	0,002687	0,018944	60,29	**0,00**	0,000092	1,08	**0,00**	0,00
u159	228,75	0,005798	0,035989	14,39	**0,00**	0,00	1,10	**0,00**	0,00
rat195	230,41	0,034008	0,052045	299,75	0,000430	0,006134	0,14	**0,00**	0,00
kroA200	271,69	0,029079	0,065538	227,93	**0,00**	0,000996	0,74	**0,00**	0,00
kroB200	248,54	0,020043	0,053233	290,18	**0,00**	0,000408	0,28	**0,00**	0,00
lin318	289,4	0,078874	0,124191	299,28	0,006472	0,016668	0,29	**0,00**	0,00
rd400	293,07	0,171978	0,224204	298,46	0,037628	0,046463	2,19	**0,00**	0,00
fl417	295,84	0,108591	0,201062	298,23	0,000759	0,001762	7,76	**0,00**	0,00
pr439	297,25	0,153996	0,265681	298,06	0,003861	0,018558	44,14	**0,00**	0,000010
d493	296,19	0,228930	0,270415	297,03	0,047140	0,051764	2,23	**0,00**	0,00
u574	204,00	0,315795	0,471734	296,51	0,055954	0,072817	3,24	**0,00**	0,00
rat575	247,05	0,358930	0,534734	296,83	0,073527	0,080592	70,21	**0,00**	0,00
p654	294,84	0,451578	0,604810	293,13	0,010074	0,013773	10,31	**0,00**	0,00
d657	292,62	0,537291	0,653566	294,98	0,055283	0,068846	72,60	0,000020	0,000020
u724	281,09	0,642390	0,713450	293,78	0,063445	0,080487	8,26	**0,00**	0,000010
rat783	239,10	0,825930	0,837516	293,05	0,076198	0,088326	30,91	**0,00**	0,00
Médias	227,05	0,135384	0,18374	178,23	0,014359	0,01834	8,63	0,00000068	0,000001

Pode-se observar, na Tabela 13, que o AM_KRAS não obtém a solução ótima para nem mesmo uma instância tendo, como melhor resultado, uma taxa de afastamento em relação ao ótimo de 0,00047 na instância kroA100. Já o algoritmo ProtoG apresenta um desempenho consideravelmente melhor encontrando a solução ótima pelo menos uma vez para 18 (60%) das 30 instâncias pequenas. No entanto, o melhor desempenho (considerando conjunto de informações da tabela acima) é apresentado pelo AM_PCVS que obteve a solução ótima para 29 (96,66%) instâncias. Nenhum dos algoritmos chegou à solução ótima da instância d657.

Em relação aos tempos médios, a Tabela 13 mostra que o AM_PCVS foi melhor em todas as instâncias chegando rapidamente ao ótimo em quase todas as instâncias. O algoritmo transgenético ProtoG teve o segundo melhor desempenho.

Tabela 14: Algoritmos Evolucionários - Tempo médio e média de afastamento (Instâncias Médias)

INST	AM_KRAS			ProtoG			AM_PCVS		
	T(s)	MIN	Média	T(s)	MIN	Média	T(s)	MIN	Média
dsj1000	579,80	0,881870	1,1809	576,35	0,0722	0,07981	412,08	0,000027	0,000271
pr1002	592,10	0,958501	1,16182	556,31	0,072	0,0823	44,84	**0**	0
u1060	586,50	0,991521	1,15335	525,11	0,0778	0,08262	143,81	**0**	0,000017
vm1084	568,30	1,254788	1,56821	519,19	0,0623	0,0756	100,75	**0**	0,000043
pcb1173	583,70	1,176053	1,53582	515,60	0,0957	0,09956	29,92	**0**	0
d1291	564,20	1,965375	2,5636	526,60	0,08	0,09023	57,67	**0**	0
rlt1304	572,60	2,283509	2,61416	521,90	0,075	0,08431	80,88	**0**	0
rl1323	555,20	2,333602	2,82941	527,63	0,0721	0,07841	201,95	**0**	0,000097
nrw1379	552,30	1,362266	1,74737	531,39	0,0869	0,0894	196,05	**0**	0,000021
fl1400	570,20	1,470562	3,22353	535,36	0,0288	0,03767	41,70	**0**	0
u1432	583,70	1,344649	1,5300	520,90	0,1	0,10342	345,94	**0**	0
fl1577	542,08	3,120068	4,35917	526,94	0,0549	0,06406	127,75	0,002	0,002279
d1665	557,55	2,214010	3,03289	534,70	0,0964	0,10249	51,12	**0**	0
vm1748	559,52	2,944259	3,37511	524,26	0,0762	0,08336	58,44	**0**	0,000162
u1817	535,01	2,654079	3,19692	525,81	0,1216	0,12555	49,66	**0**	0,001052
rl1889	568,13	3,617070	4,27496	532,48	0,0778	0,08868	373,22	**0**	0,000139
d2103	517,65	3,625838	3,91603	526,76	0,138	0,14698	393,48	0,0062	0,006750
u2152	502,66	3,127107	3,57396	54,14	0,1336	0,13812	156,34	**0**	0,000627
u2319	545,54	2,064767	2,37013	516,30	0,0659	0,07147	330,84	0,0008	0,000773
pr2392	544,86	3,089019	3,99564	518,71	0,102	0,11185	189,70	**0**	0,000107
Médias	559,08	2,123946	2,66015	505,82	0,08446	0,09179	169,31	0,00045	0,000617

Nos experimentos com instâncias de médio porte, o AM_PCVS, apresenta também o melhor desempenho como pode ser observado na Tabela 14. Este encontra a solução ótima para 16 (80%) das 20 instâncias médias e apresenta a melhor Média(t) em todas as instâncias. Os outros algoritmos não chegaram à solução ótima para nem mesmo uma instância. O melhor resultado do ProtoG foi uma solução com taxa de afastamento em relação ao ótimo de 0,0288 na instância fl400. Para as instâncias médias o melhor resultado do AM_KRAS foi uma taxa de afastamento em relação ao ótimo de 0,881870 na instância dsj1000.

A Tabela 15 apresenta os dados obtidos pelos experimentos, porém com os tempos registrados para a aplicação da Análise de Sobrevivência. Nessa tabela, são apresentadas a

percentagem de soluções ótimas (%Otm) obtidas para cada instância, bem como a média e a mediana do tempo computacional necessário até a obtenção da solução ótima – Média(*t*) e Mediana(*t*), respectivamente.

Tabela 15: ProtoG x AM_PCVS - performance em relação ao tempo e soluções ótimas

INST	ProtoG			AM_PCVS		
	Média(*t*)	Mediana(*t*)	%Otm	Média(*t*)	Mediana(*t*)	%Otm
kroA100	3,28	2,4	**100**	0,1	0,09	**100**
kroB100	28,06	16,34	**100**	0,19	0,17	**100**
kroC100	8,54	5,37	**100**	0,1	0,09	**100**
kroD100	6,65	4,91	**100**	0,11	0,11	**100**
kroE100	36,42	22,51	**100**	0,2	0,21	**100**
lin105	-	3,75	96,67	0,11	0,11	**100**
pr107	15,66	10,41	**100**	0,19	0,17	**100**
pr124	4,74	4,02	**100**	0,4	0,37	**100**
ch130	-	180,42	56,67	0,24	0,21	**100**
pr136	-	-	33,33	0,35	0,32	**100**
pr144	6,3	4,77	**100**	0,18	0,17	**100**
ch150	-	196,43	70	0,3	0,3	**100**
kroA150	-	237	63,33	0,39	0,38	**100**
kroB150	-	-	30	1,08	0,6	**100**
pr152	-	43,5	96,67	1,1	1	**100**
u159	14,27	12,85	**100**	0,14	0,14	**100**
rat195	-	-	0	0,74	0,44	**100**
kroA200	-	-	50	0,29	0,28	**100**
kroB200	-	-	10	0,29	0,28	**100**
lin318	-	-	0	2,19	1,64	**100**
rd400	-	-	0	7,76	3,16	**100**
fl417	-	-	0	6,34	5,42	**100**
pr439	-	-	0	2,23	2,03	**100**
d493	-	-	0	70,21	47,89	**100**
u574	-	-	0	10,31	5,09	**100**
rat575	-	-	0	72,6	62,27	**100**
p654	-	-	0	8,27	6,19	**100**
d657	-	-	0	-	-	0
u724	-	-	0	98,36	35,45	80
rat783	-	-	0	6,65	5,41	**100**
Médias			46,89			96

Os dados da Tabela 15 reforçam o desempenho superior do AM_PCVS aplicados às instâncias pequenas mostrando que este obtém a solução ótima em 96% das execuções realizadas contra 46,89% do ProtoG. Além disso, em relação ao tempo computacional, o AM_PCVS apresenta as melhores médias e medianas em todas as instâncias.

Para as instâncias de médio porte o AM_PCVS mantém-se melhor chegando à solução ótima em 61% dos casos. Para este conjunto de instâncias tanto o AM_KRAS quanto o ProtoG não chegam à solução ótima em nenhuma instância. Estas informações são mostradas na Tabela 16.

Tabela 16: Algoritmos Evolucionários - performance em relação ao tempo e soluções ótimas (Instâncias Médias)

INST	AM_KRAS			ProtoG			AM_PCVS		
	Média(t)	Mediana(t)	%Otm	Média(t)	Mediana(t)	%Otm	Média(t)	Mediana(t)	%Otm
dsj1000	-	-	0	-	-	0	80,31	57,47	**100**
pr1002	-	-	0	-	-	0	124,80	89,24	**100**
u1060	-	-	0	-	-	0	-	-	0
vm1084	-	-	0	-	-	0	-	-	0
pcb1173	-	-	0	-	-	0	33,69	25,30	**100**
d1291	-	-	0	-	-	0	-	-	0
rlt1304	-	-	0	-	-	0	-	483,50	60
rl1323	-	-	0	-	-	0	68,58	56,72	**100**
nrw1379	-	-	0	-	-	0	25,33	19,64	**100**
fl1400	-	-	0	-	-	0	-	231,87	90
u1432	-	-	0	-	-	0	110,73	99,94	**100**
fl1577	-	-	0	-	-	0	-	127,54	60
d1665	-	-	0	-	-	0	-	-	50
vm1748	-	-	0	-	-	0	-	161,99	80
u1817	-	-	0	-	-	0	156,85	128,19	**100**
rl1889	-	-	0	-	-	0	-	-	20
d2103	-	-	0	-	-	0	-	-	20
u2152	-	-	0	-	-	0	-	-	0
u2319	-	-	0	-	-	0	-	92,44	80
pr2392	-	-	0	-	-	0	-	367,00	60
Médias			0			0			61,00

A Tabela 17 apresenta as estatísticas Log-Rank (L) e o *p-valor* (p) para o teste da hipótese de igualdade entre as funções de sobrevivência dos algoritmos testados. Em alguns casos esses valores são omitidos porque os algoritmos não obtiveram um número suficiente de observações para que o teste pudesse ser aplicado. Na Tabela 17 estão destacados em **negrito** os valores de p quando inferior a 0,05 indicando a rejeição da hipótese de igualdade a um nível de significância de 5%.

Tabela 17: Algoritmos Evolucionários – Teste de hipótese para instâncias pequenas

INST	AM_PCVS x AM_KRAS		AM_PCVS x ProtoG		ProtoG x AM_KRAS	
	L	*p*	*L*	*p*	*L*	*p*
kroA100	7,62	**0**	5,208195	**0**	7,530877	**0**
kroB100	7,536057	**0**	5,518042	**0**	7,523658	**0**
kroC100	7,602837	**0**	5,184917	**0**	7,521247	**0**
kroD100	7,518668	**0**	5,165167	**0**	7,52383	**0**
kroE100	7,528777	**0**	5,507981	**0**	7,524563	**0**
lin105	7,565402	**0**	5,307924	**0**	7,293037	**0**
pr107	7,531127	**0**	5,49654	**0**	7,526403	**0**
pr124	7,528072	**0**	5,541639	**0**	7,52397	**0**
ch130	7,536445	**0**	6,047556	**0**	4,806913	**0**
pr136	7,534257	**0**	6,536994	**0**	3,42996	**0,0006**
pr144	7,531436	**0**	5,491376	**0**	7,524046	**0**
ch150	7,53	**0**	5,834418	**0**	5,591866	**0**
kroA150	7,53	**0**	5,925242	**0**	5,196618	**0**
kroB150	7,529189	**0**	6,639369	**0**	3,223321	**0,00127**
pr152	7,526148	**0**	5,568624	**0**	7,288447	**0**
u159	7,566818	**0**	5,380779	**0**	7,5247	**0**
rat195	7,531135	**0**	7,531135	**0**	-	-
kroA200	7,54	**0**	6,164502	**0**	4,419212	**0,00001**
kroB200	7,530128	**0**	7,16924	**0**	1,762001	0,07807
lin318	6,587008	**0**	6,587008	**0**	-	-
rd400	7,525224	**0**	7,525224	**0**	-	-
fl417	7,525874	**0**	7,525874	**0**	-	-
pr439	7,523145	**0**	7,523145	**0**	-	-
d493	6,59	**0**	6,587008	**0**	-	-
u574	-	-	7,523145	**0**	-	-
rat575	-	-	7,523145	**0**	-	-
p654	7,523145	**0**	7,523145	**0**	-	-
u724	-	-	6,201552	**0**	-	-
rat783	-	-	7,523145	**0**		-

Como pode ser observado na Tabela 17, a hipótese de igualdade é rejeitada para todas as instâncias de pequeno porte, quando comparados o AM_PCVS com o AM_KRAS e, também, o AM_PCVS com o ProtoG. Essas informações confirmam o melhor desempenho do AM_PCVS sobre as outras abordagens uma vez que este apresenta os melhores tempos em todas as instâncias.

Quando comparados ProtoG com o AM_KRAS, a hipótese de igualdade entre as funções de sobrevivência é rejeitada para 17 dos 18 casos em que o teste pôde ser realizado. Na instância kroB200 o desempenho destes algoritmos é considerado equivalente (a um nível de significância de 5%) tendo o ProtoG encontrado a solução ótima para esta instância em 10% dos casos e o AM_KRAS 0% conforme mostra a Tabela 15.

Tabela 18: Algoritmos Evolucionários - Teste de hipótese para instâncias médias

INST	AM_PCVS x AM_KRAS		AM_PCVS x ProtoG	
	L	*p*	*L*	*p*
pr1002	4,267761	**0,00002**	4,267761	**0,00002**
u1060	3,5188	**0,00043**	3,5188	**0,00043**
vm1084	3,5188	**0,00043**	3,5188	**0,00043**
pcb1173	4,267761	**0,00002**	4,267761	**0,00002**
d1291	4,267761	**0,00002**	4,267761	**0,00002**
rlt1304	4,267761	**0,00002**	4,267761	**0,00002**
rl1323	2,838045	**0,00454**	2,838045	**0,00454**
nrw1379	2,838045	**0,00454**	2,838045	**0,00454**
fl1400	4,267761	**0,00002**	4,267761	**0,00002**
d1665	4,267761	**0,00002**	4,267761	**0,00002**
vm1748	2,507822	**0,01215**	2,507822	**0,01215**
u1817	1,452436	0,14638	1,452436	0,14638
rl1889	2,507822	**0,01215**	2,507822	**0,01215**
u2152	1,452436	0,14638	1,452436	0,14638
pr2392	3,881548	**0,0001**	3,881548	**0,0001**

A Tabela 18 mostra que, para as instâncias de médio porte, a hipótese de igualdade é rejeitada para 13 de 15 instâncias para as quais o teste pôde ser realizado. Nessa tabela não aparecem os resultados para o AM_KRAS e ProtoG porque os mesmos não encontraram a solução ótima nenhuma vez. Nas instâncias u1817 e u2152, embora o AM_PCVS tenha chegado ao ótimo em 20% dos casos e os outros algoritmos nem mesmo uma vez, segundo o teste de hipótese realizado, não se pode dizer que o AM_PCVS é melhor nessas instâncias, contestando uma conclusão que seria obtida a partir de uma análise convencional (Tabela 14).

6.2.2. Instâncias Assimétricas

Nesta seção são apresentados os experimentos computacionais para as instâncias assimétricas da biblioteca TSPLIB. Os algoritmos envolvidos no experimento são o AM_PCVA (proposto neste trabalho) e o algoritmo memético proposto por Buriol *et al.* (2003).

A Tabela 19 apresenta uma análise convencional onde, para cada instância, são mostrados a média do tempo, em segundos, gasto até a obtenção da solução ótima (T), a menor taxa de afastamento em relação ao ótimo (MIN) e a média da taxa de afastamento (Média) obtidos por cada um dos algoritmos.

Observando a Tabela 19, pode-se notar o bom desempenho do AM_BURIOL que chegou ao ótimo em todas as instâncias obtendo uma taxa de afastamento média igual a zero em 12 (63,15%) instâncias. Com um desempenho um pouco melhor, o AM_PCV encontra a solução ótima em 15 (78,94%) instâncias e só não obtém uma taxa de afastamento média igual a zero nas instâncias rbg323, rbg358, rbg403 e rbg443. Dessa forma, o AM_BURIOL se mostra melhor nas 4 instâncias maiores e, por sua vez, o AM_PCVA tem melhor desempenho em 7 instâncias sendo registrados ainda, 8 empates.

Tabela 19: AM_BURIOL x AM_PCVA - Tempo médio e média de afastamento (Instâncias Assimétricas)

INST	AM_BURIOL			AM_PCVA		
	T(s)	MIN	Média	T(s)	MIN	Média
br17	0,064	**0**	0	0	**0**	0
ftv33	0,139	**0**	0	0,424	**0**	0
ftv35	0,158	**0**	0	0,824	**0**	0
ftv38	0,153	**0**	0,091503	2,632	**0**	0
p43	0,261	**0**	0,007117	2,243	**0**	0
ftv44	0,205	**0**	0,433974	2,063	**0**	0
ftv47	0,267	**0**	0	1,011	**0**	0
ry48p	0,300	**0**	0	9,043	**0**	0
ft53	0,260	**0**	0	5,2	**0**	0
ftv55	0,358	**0**	0	0,545	**0**	0
ftv64	0,548	**0**	0	1,365	**0**	0
ft70	0,466	**0**	0,037235	7,322	**0**	0
ftv70	0,294	**0**	0,025641	1,328	**0**	0
kro124p	0,786	**0**	0,003036	7,755	**0**	0
ftv170	2,347	**0**	0,054446	16,247	**0**	0
rbg323	0,022	**0**	0	412,44	0,01056	0,013273
rbg358	0,020	**0**	0	358,891	0,01806	0,0227
rbg403	0,031	**0**	0	245,76	0,00081	0,001542
rbg443	0,033	**0**	0	342,847	0,0011	0,001765
Médias	0,353	0,000	0,034366	74,628	0,002	0,002

Em relação ao tempo computacional, nota-se a rápida convergência do algoritmo memético de Buriol que apresenta seu melhor resultado antes de 0,5 segundos em 16 das 19 instâncias testadas e chegando a seu melhor resultado sempre antes dos 3 segundos até mesmo nas instâncias com mais de 300 cidades nas quais o AM_PCVA não chegou ao ótimo.

Na Tabela 20 são apresentados os resultados do experimento com os tempos ajustados para aplicação da Análise de Sobrevivência. São destacados nessa tabela, além da percentagem de soluções ótimas obtidas pelos algoritmos em cada instância, a Média(t) e Mediana(t) do tempo computacional necessário até a obtenção da solução ótima.

Tabela 20: Algoritmos Evolucionários – performance em relação ao tempo e soluções ótimas (Instâncias Assimétricas)

INST	AM_BURIOL			AM_PCVA		
	Média(*t*)	Mediana(*t*)	%Otm	Média(*t*)	Mediana(*t*)	%Otm
br17	0,064	0,06	**100**	0	0	**100**
ftv33	0,139	0,14	**100**	0,424	0,35	**100**
ftv35	0,158	0,16	**100**	0,824	0,73	**100**
ftv38	-	-	30	2,632	0,85	**100**
p43	-	0,27	60	2,243	1,78	**100**
ftv44	-	-	30	2,063	1,31	**100**
ftv47	0,267	0,25	**100**	1,011	0,92	**100**
ry48p	0,294	0,3	**100**	9,043	8,17	**100**
ft53	0,3	0,3	**100**	5,2	4,12	**100**
ftv55	0,26	0,27	**100**	0,545	0,57	**100**
ftv64	0,358	0,36	**100**	1,365	1,47	**100**
ft70	-	-	30	7,322	6,54	**100**
ftv70	0,466	0,48	**100**	1,328	1	**100**
kro124p	-	0,79	90	7,755	7,63	**100**
ftv170	-	2,33	70	16,247	15,63	**100**
rbg323	0,022	0,02	**100**	-	-	0
rbg358	0,02	0,02	**100**	-	-	0
rbg403	0,031	0,03	**100**	-	-	0
rbg443	0,033	0,03	**100**	-	-	0
Médias			84,73			78,94

A Tabela 20 mostra que o AM_BURIOL encontra a solução ótima em 84,73% das execuções contra 78,94% do AM_PCVA. Nas instâncias com menos de 170 cidades o AM_PCVA se mostra melhor, porém, para as instâncias maiores, o AM_BURIOL é bem superior.

Observando somente a Tabela 19, numa análise descritiva, a conclusão seria que o AM_PCVS é superior porque apresenta uma média de qualidade de soluções melhor que o AM_BURIOL. No entanto, aplicando o método Log-Rank para o teste da hipótese de igualdade entre as funções de sobrevivência, tem-se os dados da Tabela 21 onde a hipótese nula é rejeitada para 16 das 19 instâncias testadas. Com essas informações, pode-se afirmar que o AM_PCVA é superior apenas nas instâncias br17, ftv38, ftv44 e ft70 pois apresenta as melhores médias e medianas nesses casos. Da mesma forma, pode-se concluir que o AM_BURIOL é melhor nas instâncias ftv33, ftv35, ftv47, ry48p, ft53, ftv55, ftv64, ftv70, rbg323, rbg358, rbg403 e rbg443.

Tabela 21: Algoritmos Evolucionários - Teste de hipótese para instâncias assimétricas

INST	AM_BURIOL x AM_PCVA	
	L	*p*
br17	3,124908	**0,00178**
ftv33	-3,27565	**0,00105**
ftv35	-3,154	**0,00161**
ftv38	-2,52252	**0,01165**
p43	-0,736854	0,46121
ftv44	2,439914	**0,01469**
ftv47	-3,19861	**0,00138**
ry48p	3,166151	**0,00154**
ft53	3,174564	**0,0015**
ftv55	-3,24443	**0,00118**
ftv64	-3,23576	**0,00121**
ft70	2,445564	**0,01446**
ftv70	-3,21082	**0,00132**
kro124p	-1,80918	0,07042
ftv170	-0,049575	0,96046
rbg323	-4,27726	**0,00002**
rbg358	-4,28716	**0,00002**
rbg403	-4,34389	**0,00001**
rbg443	-4,31754	**0,00002**

Em relação às instâncias kro124p, ftv170 e p43, não se pode afirmar que o AM_PCVA é superior, porque, como mostra a Tabela 21, os algoritmos apresentam equivalência de desempenho a um nível de significância de 5%. Sendo assim, a Análise de Sobrevivência com o teste de Log-Rank mostram que o AM_BURIOL é, de fato, superior em 12 (63,15%) instâncias e o AM_PCVA em apenas 4 (21,05%).

6.3. Classe III – Nuvem de Partículas

Esta seção apresenta a análise estatística dos resultados obtidos pelos algoritmos de Otimização por Nuvem de Partículas PSO-ML e PSO-S. O primeiro foi desenvolvido por Machado e Lopes (2005) e o segundo por Souza (2006).

A Tabela 22 apresenta os resultados obtidos, destacando, para cada algoritmo, o tempo médio de processamento em segundos (T), a taxa de afastamento em relação à solução ótima (MIN) e a média da taxa de afastamento em relação à solução ótima (Média). A coluna INST lista o nome das instâncias utilizadas.

Tabela 22: Algoritmos PSO – Tempo médio e média de afastamento (Instâncias pequenas)

INST	PSO-S			PSO-ML		
	T(s)	MIN	Média	T(s)	MIN	Média
kroA100	0,18	**0**	0	190,633	**0**	0,002548
kroB100	0,25	**0**	0	193,6	**0**	0,001276
kroC100	0,17	**0**	0	99,5	**0**	0,000289
kroD100	0,23	**0**	0	77,033	**0**	0,00319
kroE100	0,24	**0**	0	34,467	0,000209	0,000209
lin105	0,19	**0**	0	217,167	0,019686	0,036514
pr107	0,29	**0**	0	48,533	**0**	0
pr124	0,4	**0**	0	217,7	0,000134	0,003481
bier127	0,49	**0**	0	199,167	0,000093	0,00257
ch130	0,36	**0**	0	185,533	0,000327	0,003424
pr136	0,44	**0**	0	85,133	**0**	0,000018
pr144	0,42	**0**	0	110,867	0,000014	0,000707
ch150	0,27	**0**	0	240,467	0,035066	0,051848
kroA150	0,39	**0**	0	245,333	0,000034	0,00498
kroB150	0,89	**0**	0	242	0,002136	0,701978
pr152	1,75	**0**	0	256,233	0,019156	0,035295
u159	0,25	**0**	0	200,733	0,056717	0,074787
rat195	0,61	**0**	0	179,967	0,072827	0,083126
kroA200	0,54	**0**	0	68,333	**0**	0,000396
kroB200	0,52	**0**	0	44,367	0,000048	0,000638
lin318	3,88	**0**	0	229,2	0,01462	0,027696
rd400	6,75	**0**	0	101,2	**0**	0,002932
fl417	8,61	**0**	0	35	0,000141	0,000141
pr439	2,49	**0**	0	247,333	0,010647	0,023617
pcb442	2,98	**0**	0	115,6	**0**	0,000507
d493	70	**0**	0,000023	193,7	0,058298	0,074785
u574	7,48	**0**	0	160,8	0,068667	0,084365
rat575	84,33	**0**	0,000163	189,467	0,082995	0,093632
p654	10,53	**0**	0	227,633	0,041382	0,055982
d657	162,12	0,00002	0,000091	251,233	0,012982	0,019768
u724	161,55	**0**	0,000256	186,5	0,082028	0,085985
rat783	14,32	**0**	0	175,2	0,031254	0,05279
Médias	16,99	0,0000006	0,000017	164,051	0,0190457	0,0477961

Como pode ser observado na Tabela 22, numa análise descritiva, o PSO-S mostra um desempenho superior chegando à solução ótima em 31 (96,87%) das 32 instâncias utilizadas e, ainda, uma média da taxa de afastamento em relação ao ótimo igual a zero em 18 (56,25%) instâncias. O algoritmo PSO-ML encontrou a solução ótima para 9 (28,12%) instâncias e uma Média igual a zero apenas na instância pr107. Em relação ao tempo computacional o PSO-S apresenta também as melhores médias e medianas.

Para as instâncias com mais de 1000 cidades, o PSO-S, em geral, apresenta, também, um desempenho superior quando comparado ao PSO-ML. A Tabela 23 apresenta essas informações destacando também o tempo médio T em segundos e a Média e Mediana da taxa de afastamento, porém para as instâncias de médio porte.

Tabela 23: Algoritmos PSO – Tempo médio e média de afastamento (Instâncias Médias)

INST	PSO-S			PSO-ML		
	T(s)	MIN	Média	T(s)	MIN	Média
dsj1000	476,28	**0**	0,000045	476,8	0,07624	0,08136
pr1002	56,234	**0**	0	308	0,07527	0,09349
u1060	452,881	**0**	0,0001	393,5	0,08639	0,09083
vm1084	304,066	**0**	0,000023	411,8	0,07207	0,08227
pcb1173	356,953	**0**	0,000021	368,8	0,10065	0,10777
d1291	322,68	**0**	0,000356	440,1	0,09089	0,10748
rl1304	437,969	**0**	0,000076	433,8	0,07154	0,09271
rl1323	402,719	**0**	0,000084	435,4	0,07798	0,09157
nrw1379	561,602	**0**	0,000208	422	0,08955	0,09707
fl1400	63,258	**0**	0	458	0,04239	0,06235
u1432	454,913	**0**	0,000248	440,7	0,10026	0,10708
fl1577	591,664	0,00234	0,006526	452,2	0,07429	0,18481
d1655	576,589	**0**	0,000359	498,1	0,10963	0,11954
vm1748	425,063	**0**	0,000086	391,2	0,09431	0,27928
u1817	601,978	0,00047	0,001725	368,2	0,12838	0,39013
rl1889	603,55	0,000082	0,000252	99,9	0,10496	0,80621
d2103	604,77	0,00627	0,006634	489,35	0,97421	0,97446
u2152	605,983	0,0003	0,001468	580,52	0,97302	0,97344
u2319	603,67	0,00042	0,000738	479,36	0,95881	0,95932
pr2392	607,82	0,000019	0,000342	523,12	0,97369	0,97422
Médias	455,532	0,000495	0,000965	423,543	0,26373	0,33377

Observando os dados da Tabela 23 não é difícil notar o bom desempenho do PSO-S que encontra a solução ótima para 13 (65%), enquanto o PSO-ML não encontra a solução ótima para nenhuma instância. Além disso, o PSO-S apresenta a melhor Média em todas as instâncias, não obstante a pequena diferença do tempo médio de processamento.

É importante destacar que os dados das Tabelas 22 e 23 correspondem a uma análise tradicional onde o tempo registrado corresponde ao instante em que o algoritmo encontrou sua melhor solução que não, necessariamente, corresponde à solução ótima. No entanto, para a Análise de Sobrevivência, o tempo observado corresponde ao instante em que a solução ótima foi encontrada e, se o algoritmo não encontra a solução ótima no tempo máximo determinado, o tempo não é registrado. Dessa forma, as Tabelas 24 e 25 apresentam, para cada algoritmo, a Média e a Mediana do tempo de processamento gasto até a obtenção da solução ótima, Média(t) e Mediana(t), respectivamente; e a percentagem de soluções ótimas encontradas (%Otm) em cada instância.

Tabela 24: Algoritmos PSO – performance em relação ao tempo e soluções ótimas (Instâncias de pequeno porte)

INST	PSO-G			PSO-ML		
	Média(*t*)	Mediana(*t*)	%Otm	Média(*t*)	Mediana(*t*)	%Otm
kroA100	0,181	0,1700	**100**	-	-	16,67
kroB100	0,252	0,2355	**100**	-	-	16,67
kroC100	0,170	0,1600	**100**	-	183,50	56,67
kroD100	0,235	0,2300	**100**	-	-	3,33
kroE100	0,242	0,2100	**100**	-	-	0
lin105	0,186	0,1800	**100**	-	-	0
pr107	0,290	0,2855	**100**	48,533	36,00	**100**
pr124	0,403	0,3500	**100**	-	-	0
bier127	0,502	0,4800	**100**	-	-	0
ch130	0,358	0,3000	**100**	-	-	0
pr136	0,437	0,4310	**100**	-	70,50	96,67
pr144	0,424	0,4200	**100**	-	-	0
ch150	0,286	0,2700	**100**	-	-	0
kroA150	0,392	0,3510	**100**	-	-	0
kroB150	0,872	0,4410	**100**	-	-	0
pr152	1,748	1,6420	**100**	-	-	0
u159	0,255	0,2555	**100**	-	-	0
rat195	0,606	0,4350	**100**	-	-	0
kroA200	0,539	0,5410	**100**	-	72,50	80
kroB200	0,522	0,5210	**100**	-	-	0
lin318	3,792	2,2335	**100**	-	-	0
rd400	8,694	5,7280	**100**	-	135,00	60
fl417	8,606	6,7750	**100**	-	-	0
pr439	2,465	1,9575	**100**	-	-	0
pcb442	3,079	2,5890	**100**	-	-	50
d493	-	79,6595	60	-	-	0
u574	7,732	6,414000	**100**	-	-	0
rat575	-	-	20	-	-	0
p654	10,401	7,170000	**100**	-	-	0
d657	-	-	0	-	-	0
u724	-	-	13,33	-	-	0
rat783	14,295	13,374000	**100**	-	-	0
Médias			90,42			15,0

Os dados da Tabela 24 reafirmam a superioridade do algoritmo PSO-S nas instâncias pequenas mostrando que este encontra a solução ótima em 90,42% das execuções contra 15% de soluções ótimas encontradas pelo PSO-ML. Além disso, a Média e a Mediana do tempo computacional gasto pelo PSO-S para a obtenção das soluções são consideravelmente melhores.

Para as instâncias de médio porte, como mostra a Tabela 25, o PSO-S mantém-se melhor que o PS-ML, porém com uma percentagem de soluções ótimas encontradas bem reduzida. Nesse conjunto de instâncias o PSO-ML não encontrou a solução ótima para nenhuma instância e o PSO-S não encontrou a solução ótima para as instâncias fl1577, u1817, rl1889, d2103, u2152, u2319 e pr2392.

Tabela 25: Algoritmos PSO – performance em relação ao tempo e soluções ótimas (Instâncias de médio porte)

INST	PSO-G			PSO-ML		
	Média(*t*)	Mediana(*t*)	%Otm	Média(*t*)	Mediana(*t*)	%Otm
dsj1000	-	-	30	-	-	0
pr1002	56,24	51,80	**100**	-	-	0
u1060	-	-	50	-	-	0
vm1084	-	236,12	70	-	-	0
pcb1173	-	-	50	-	-	0
d1291	-	284,12	60	-	-	0
rl1304	-	-	40	-	-	0
rl1323	-	-	40	-	-	0
nrw1379	-	-	20	-	-	0
fl1400	63,26	48,83	**100**	-	-	0
u1432	-	-	40	-	-	0
fl1577	-	-	0	-	-	0
d1655	-	-	10	-	-	0
vm1748	-	409,29	60	-	-	0
u1817	-	-	0	-	-	0
rl1889	-	-	0	-	-	0
d2103	-	-	0	-	-	0
u2152	-	-	0	-	-	0
u2319	-	-	0	-	-	0
pr2392	-	-	0	-	-	0
Médias			3,35			0

Aplicando o teste Log-Rank para teste da hipótese de igualdade entre as funções de sobrevivência para as instâncias de pequeno porte, têm-se os dados da Tabela 26. Como pode ser observado, a hipótese nula é rejeitada para todas as instâncias exceto para a d657 na qual o teste não pôde ser realizado. Esses dados comprovam estatisticamente que, para esse conjunto de instâncias, o PSO-S é superior ao PSO-ML em um nível de significância de 5% pois apresenta, também, as melhores Médias e Medianas (Tabela 24).

Tabela 26: Algoritmos PSO – Teste de hipótese para instâncias pequenas

INST	PSO-G x PSO-ML	
	L	*p*
kroA100	-6,88831	**0**
kroB100	-6,94041	**0**
kroC100	-5,93968	**0**
kroD100	-7,42142	**0**
kroE100	-7,54701	**0**
lin105	-7,55076	**0**
pr107	-5,38227	**0**
pr124	-7,54472	**0**
bier127	-7,53823	**0**
ch130	-7,53096	**0**
pr136	-5,35532	**0**
pr144	-7,54124	**0**
ch150	-7,53139	**0**
kroA150	-7,53167	**0**
kroB150	-7,54524	**0**
pr152	-7,52324	**0**
u159	-7,53024	**0**
rat195	-7,54811	**0**
kroA200	-5,56524	**0**
kroB200	-7,5147	**0**
lin318	-7,52788	**0**
rd400	-5,88946	**0**
fl417	-7,52707	**0**
pr439	-7,52944	**0**
pcb442	-6,16379	**0**
d493	-5,00149	**0**
u574	-7,52563	**0**
rat575	-2,55921	**0,01049**
p654	-7,52614	**0**
u724	-2,05248	**0,04012**
rat783	-7,52614	**0**

Tabela 27: Algoritmos PSO - Teste de hipótese para instâncias médias

INST	PSO-G x PSO-ML	
	L	*p*
dsj1000	1,82922	0,06737
pr1002	4,26776	**0,00002**
u1060	2,50782	**0,01215**
pcb1173	2,50782	**0,01215**
d1291	2,83805	**0,00454**
rl1304	2,17498	**0,02963**
rl1323	2,17498	**0,02963**
nrw1379	1,45244	0,14638
fl1400	4,26776	**0,00002**
u1432	2,17498	**0,02963**
fl1577	3,17297	**0,00151**
vm1748	2,83805	**0,00454**

A Tabela 27 apresenta as estatísticas Log-Rank (L) e o p-valor (p) para o teste da hipótese de igualdade das funções de sobrevivência para as instâncias de médio porte. Essa tabela mostra que a hipótese nula é rejeitada para 10 das 12 instâncias em que o teste pôde ser aplicado. Dessa forma, os dados da Tabela 27 vêm confirmar a superioridade do PSO-S quando comparado ao PSO-ML, uma vez que o PSO-S apresenta as melhores médias em todas as instâncias. Por outro lado, mesmo com as informações mostradas na Tabela 23, o resultado, em relação ao melhor desempenho, não é conclusivo para as instâncias dsj1000 e nrw1379 onde o PSO-S encontrou a solução ótima em apenas 30% e 20% dos casos, respectivamente. Nessas duas instâncias, a diferença entre as funções de sobrevivência não é significativa a um nível de 5% e as abordagens podem ser consideradas equivalentes.

Capítulo 7

7. Considerações Finais

A Análise de Sobrevivência auxiliada pelo teste de Log-Rank foi utilizada, para o estudo do tempo computacional necessário até que a solução ótima para determinada instância do PCV seja obtida. Para o estudo da Análise de Sobrevivência, nesse caso, um evento só é registrado quando a solução ótima é encontrada, ou seja, mesmo que determinada abordagem proporcione soluções bem próximas do ótimo, o instante em que esta solução foi obtida não é registrado. Desta forma, principalmente nas instâncias acima de 1500 cidades, muitas observações foram censuradas a ponto de não permitir a conclusão da análise para algumas instâncias. Eventualmente, pode-se considerar, para a Análise de Sobrevivência, ao invés da solução ótima, soluções que tenham uma determinada taxa de afastamento (maior que zero) em relação ao ótimo.

Para os algoritmos *Iterated* LK utilizados, a análise mostrou que o LK-H é melhor que o LK-ABCC e LK-NYYY na maioria das instâncias. Na comparação entre o LK-ABCC e LK-NYYY estes apresentaram, segundo a Análise de Sobrevivência, um desempenho equivalente, exceto nas instâncias vm1084 e rl1323 onde a hipótese nula é rejeitada e o LK-ABCC apresenta os melhores resultados em termos de tempo computacional.

Esse trabalho apresentou, também, um Algoritmo Memético para a solução do Problema do Caixeiro Viajante simétrico e assimétrico utilizando como procedimento de busca local uma implementação do algoritmo Lin-Kernighan desenvolvida por Applegate *et al.* (1999b). Para a versão assimétrica, foi utilizado o método de transformação de instâncias assimétricas em simétricas porque o LK, originalmente, não se aplica ao PCV assimétrico. Os experimentos mostraram que a utilização do LK (nesse caso o LK-ABCC) como busca local em algoritmos evolucionários é bastante eficiente proporcionando soluções de boa qualidade num tempo computacional relativamente baixo.

Os resultados computacionais mostraram que dentre os Algoritmos Evolucionários utilizados, a versão simétrica do algoritmo proposto (AM_PCVS) apresentou o melhor desempenho chegando à solução ótima em 96% dos testes feitos com as instâncias de pequeno porte. Para este conjunto de instâncias, o teste de hipótese de igualdade entre as funções de sobrevivência foi rejeitado em 100% e por apresentar as melhores médias e medianas, fica comprovada a superioridade deste algoritmo quando comparado às outras abordagens.

Para o conjunto de instâncias assimétricas, se fosse considerada somente a média da taxa de afastamento em relação à solução ótima (Tabela 19), chegaria-se à conclusão de que o AM_PCVA é melhor que o AM_BURIOL. No entanto através da análise realizada, mostrou-se que não é tão simples assim. Apesar dos bons resultados nas instâncias pequenas, o AM_PCVA apresentou dificuldades na solução de instâncias maiores, nas quais o AM_BURIOL chegou ao ótimo em todas as execuções. Com o auxílio do teste de hipótese de igualdade entre as funções de sobrevivência mostrou-se que o AM_BURIOL, de fato, é superior em 63,15% das instâncias e o AM_PCVA em apenas 21,05%.

Para os algoritmos PSO pertencentes à Classe III, a aplicação da Análise de Sobrevivência não revelou grandes surpresas. O algoritmo PSO-S foi superior na maioria das instâncias apresentando os melhores resultados. Para esses algoritmos, o teste não foi conclusivo apenas para as instâncias dsj1000 e nrw1379 onde, apesar de o PSO-S ter encontrado a solução ótima em 30% e 20% dos casos, respectivamente, e o PSO-ML não obtendo a solução ótima nenhuma vez, não se pode afirmar estatisticamente que o PSO-S seja melhor nessas instâncias (considerando um nível de significância de 5%).

A análise experimental apresentada nesse trabalho tem, pelo menos, dois grandes diferenciais em relação a outras análises já reportadas na literatura. O primeiro diferencial está no fato de se utilizar uma metodologia estatística para a realização da análise. Durante anos a análise de desempenho vem sendo baseada apenas na qualidade média das soluções e sem o suporte de um outro método estatístico. O segundo diferencial se refere à variável analisada que, nesse caso, foi o tempo computacional. Considerando que, para a maioria dos problemas do mundo real, os métodos exatos já desenvolvidos se tornam impraticáveis em função do longo tempo de processamento necessário (meses e até anos), uma análise que permite a avaliação das heurísticas que têm maior probabilidade de encontrar uma solução com determinada qualidade, antes de um certo tempo, ganha muita importância.

Futuramente, para melhorar o desempenho do AM_PCVA, operadores genéticos mais eficientes em instâncias assimétricas podem ser desenvolvidos e testados, proporcionando, possivelmente, melhores resultados. Pode-se testar, também, a utilização do LK-H como procedimento de busca local, embora, a expectativa dessa ação é que os tempos computacionais sejam muito elevados. Além disso, este estudo do tempo computacional pode ser grandemente ampliado com a introdução de heurísticas não abordadas aqui.

Referências Bibliográficas

AARTS, E. H. L.; VERHOEVEN, M. G. A. Genetic local search for the traveling salesman problem, Handbook of Evolutionary Computation, pp.G9.5:1-7, IOP publishing Ltd and Oxford University Press. 1997.

ALLISON, P. D. Survival Analysis using the SAS System: A Practical Guide, Cary, NC: SAS Institute Inc., 1995.

APPLEGATE, D.; BIXBY, R.; CHVÀTAL, V.; COOK, W. Finding Cuts in the TSP (A preliminary report), Technical Report 95-05, DIMACS, 1995.

APPLEGATE, D.; BIXBY, R.; CHVÀTAL, V.; COOK, W. On the Solution of Traveling Salesman Problems, Documenta Mathematica, vol. Extra Volume ICM III, pp. 645-656, 1998.

APPLEGATE, D.; BIXBY, R.; CHVÀTAL, V.; COOK, W. Concorde: A code for solving Traveling Salesman Problems, 1999a.

APPLEGATE, D.; BIXBY, R.; CHVÀTAL, V.; COOK, W. Finding Tours in the TSP. Tech. Rep. TR99-05, Dept. Comput. Appl. Math., Rice Univ., 1999b.

APPLEGATE, D.; COOK, W.; ROHE, A. Chained Lin-Kernighan for large traveling salesman problems. Tech. Rep., Dept. Comput. Appl. Math., Rice Univ., 2000.

APPLEGATE, D.; BIXBY, R.; CHVÀTAL, V.; COOK, W. 2002. To appear.

APPLEGATE, D.; COOK, W.; ROHE, A. Chained Lin-Kernighan for Large Traveling Salesman Problems, INFORMS Journal on Computing, 15, pp. 82-92, 2003.

APPLEGATE, D.; BIXBY, R.; CHVÀTAL, V.; COOK, W.; HELSGAUN, K. Optimal tour of Sweden. 2004. Disponível na Internet em http://www.tsp.gatech.edu/sweden/ com último acesso em junho de 2006.

BATTITI, R.; TECCHIOLLI, G. The Reactive Tabu Search, ORSA Journal of Computing, Vol. 6, N. 2, pp. 126-140. 1994.

BALAS, E.; TOTH, P. Branch and bound methods. In The Traveling Salesman Problem, Edited by Lawler, E. L.; Lenstra, J. K.; Rinnooy Kan, A. H. G. and Shmoys, D. B. John Wiley & Sons Ltd, 361-401, 1985.

BELLMORE, M.; NEMHAUSER, G. L. The traveling salesman problem: a survey. Operations Research 16, 538–558.

BLAND, R. E.; SHALLCROSS, D. F. Large Traveling Salesman Problem Arising from Experiments in X-ray Crystallography: a Preliminary Report on Computation, Relatório Técnico No. 730, School of OR/IE, Cornell University, Ithaca, New York, 1987.

BOESE, K. Cost versus Distance in the Traveling Salesman Problem. Relatório Técnico. UCLA CS Departament, 1995.

BURIOL, L.; FRANÇA, P. M.; MOSCATO, P. A New Memetic Algorithm for the Asymmetric Traveling Salesman Problem. 2003.

BRUNETTA, L.; CONFORTI, M.; RINALDI, G. A. Branch-and-Cut Algorithm for the Equicut Problem, Mathematical Programming, vol. 78, no. 2, pp. 243-263, 1997.

CHIANG, Y.; TAMASSIA, R. Dynamic algorithms in computational geometry. Proc. IEEE, 80:1412-1434, 1992.

CHRISTOFIDES, N.; EILON, S. Algorithms for Large-scale Travelling Salesman Problems, Oper. Res. Quart., 23, 511-518. 1972.

CLERC, M. Discrete Particle Swarm Optimization, New Optimization Techniques in Engineering Springer-Verlag. 2004.

COLLETT, D. Modelling survival data in medical research. London: Chapman & Hall. 1991.

CONCORDE. Concorde TSP Solver. Disponível na Internet em http://www.tsp.gatech.edu/concorde/index.html com últmo acesso em 12 de junho de 2006.

CROES, G. A. A method for solving traveling salesman problems. Operations Research, 1958.

CROWDER, H.; PADBERG, M. W. Solving Large-Scale Symmetric Traveling Salesman Problems to Optimality, Management Science, vol. 26, pp. 495-509, 1980.

DANTZIG, G. B.; FULKERSON, D. R.; JOHNSON, S. M. Solution of a Large-Scale Traveling Salesman Przoblem,Operations Research, vol. 2, pp. 393-410, 1954.

DANTZIG, G. B.; FULKERSON, D. R.; JOHNSON, S. M. Cutting-Plane Method. Disponível na Internet em http://www.tsp.gatech.edu/methods/dfj/index.html com último acesso em julho de 2006.

DAVIS, L. Applying Adaptive Algorithms to Espistatic Domains. Proceedings of the International Joint Conference on Artificial Intelligence, 1985.

DAWKINS, R. The selfish gene. Oxford University Press, Oxford, 1976.

DIDERICH, C. G.; GENGLER, M. Solving Traveling Salesman Problems Using a Parallel Synchronized Branch and Bound Algorithm. HPCN Europe 1996: 633-638

DIMACS. 8th DIMACS Implementation Challenge: The Traveling Salesman Problem. Disponível na Internet em http://www.research.att.com/~dsj/chtsp/results.html com último acesso em 7 de junho de 2006.

DUMITRESCU, I.; STÜTZLE, T. A survey of methods that combine local search and exact algorithms. Technical Report AIDA-03-07, FG Intellektik, FB Informatik, TU Darmstadt, Germany, 2003. Submitted to EJOR.

ESMIN, A. A. A.; AOKI, A. R.; LAMBERT-TORRES, G. Particle swarm optimization for fuzzy membership functions optimization. In: IEEE International Conference on Systems, Man and Cybernetics, Tunisia, 2002.

FARIAS, J. P. F. Algoritmo GRASP para o Caixeiro Viajante. Notas de aula, Metaheurísticas, Universidade Federal do Rio Grande do Norte. 2004.

FEO, T. A.; RESENDE, M. G. C. Greedy Randomized Adaptive Search Procedure. Journal of Global Optmization. Vol 6, pag 109-133, 1995.

FISCHETTI, M.; GONZÁLEZ, J. J. S.; TOTH, P. A Branch-and-Cut Algorithm for the Symmetric Generalized Traveling Salesman Problem, Operations Research, vol. 45,pp. 378-394, 1997.

FOGEL, D. B. An Evolutionary Approach to the Traveling Salesman Problem, Biological Cybernetics, 60, pp. 139-144. 1988.

FREDMAN, M. L.; JOHNSON, D.S.; MCGEOCH, L.A.; OSTHEIMER, G. Data Structures for Travelling Salesmen. J. Algorithms 18, 432-479, 1995.

FREISLEBEN, B.; MERZ, P. New Genetic Local Search Operators for the Traveling Salesman Problem. 4th Conference on Parallel Problem Solving from Nature Springer. 1996.

GAMBOA, D.; OSTERMAN, C.; REGO, C.; GLOVER, F. Ejection Chain Algorithms for the Traveling Salesman Problem ORP3 Meeting, Valencia. Setembro 6-10, 2005.

GARCIA, V. J.; MENDES, A. DE S.; FRANÇA, P. M.; MOSCATO, P. A. Algoritmo Memético paralelo aplicado a Problemas de Sequenciamento em Máquina Simples. Departamento de Engenharia Elétrica de Sistemas. Universidade Estadual de Campinas, Campinas – SP. 2001.

GLOVER, F. Parametric combinations of local job shop rules. Chapter IV, ONR Research Memorandum N. 117, GSIA, Carnegie Mellon University, Pittsburgh, PA. 1963.

GLOVER, F. Tabu Search: A Tutorial. Interfaces, July-August 1990, pp74-94. 1990.

GOLDBARG, M. C.; GOUVÊA, E. F. Computational transgenetic. In: Congresso Latinoibero-americano de Investigación Operaciones y Sistemas, 10, 2000.

GOLDBARG, M. C.; GOUVÊA, E. F.; SILVA, L. M. ProtoG: a Computational Transgenetic Algorithm. In 4Th Metaheutistics International Conference – MIC. 2001, 321-326.

GOLDBARG, M. C.; GOLDBARG, E. F. G. Transgenética Computacional: Uma aplicação ao Problema Quadrático de Alocação. Pesquisa Operacional 22(3), 359-386. 2002.

GOLDBERG, D. E.; LINGLE, JR. R. Alleles, Loci and the TSP, in Grefenstette, J.J. (Ed.) Proceedings os the Fisrt International Conference on Genetic Algorithms and Their Applications, Lawrence Erlbaum, Hillsdale, New Jersey, pp. 154-159. 1985.

GONNET, G.; KOROTENSKY, C.; BENNER, S. Evaluation measures of multiple sequence alignments, Journal of Computational Biology 7(1-2), 261-276. 2000.

GOUVÊA, E. F. Transgenética computacional: um estudo algorítmico. 2001. Tese de Doutorado – Universidade Federal do Rio d Janeiro, Rio de Janeiro, 2001.

GOUVÊA, E. F.; GOLDBARG, M. C. A transgenetic algorithm applied to the quadratic assignment problem. Journal of Heuristics . Submetido em 2003.

GRÖTSCHEL, M. On the Symmetric Traveling Salesman Problem: Solution of a 120-city problem, Mathematical Programming Studies, vol. 12, pp. 61-77, 1980.

GRÖTSCHEL, M.; HOLLAND, O. Solution of Large-scale Symmetric Travelling Salesman Problems, Mathematical Programming, vol. 51, pp. 141-202, 1991.

GUTIN, G.; PUNNEN, A. The Traveling Salesman Problem and Its Variations, Kluwer Academic Publishers, Boston, pp. 369-443, 2002.

GUYON, R.; LORENTZEN, T. D.; HITTE, C.; KIM, L.; CADIEU, E.; PARKER, H. G.; QUIGNON, P.; et al. A 1-Mb resolution radiation hybrid map of the canine genome, Proceedings of the National Academy of Sciences of the United States of America 100(9): 5296-5301. 2003.

HANSEN, P. The Steepest Ascent Mildest Descent Heuristic for Combinatorial Programming. Congress Numerical Methods in Combinatorial Optimization, Capri, Italia, 1986.

HELD, M.; KARP, R.M. A dynamic programming approach to sequencing problems. Journal of the Society of Industrial and Applied Mathematics 10, 196-210, 1962.

HELSGAUN, K. An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic, European Journal of Operational Research, 126, pp. 106-130, 2000.

HEPPNER, F.; GRENANDER, U. A Stochastic nonlinear model for coordinated bird flocks. In S. Krasner, Ed., The Ubiquity of Caos. AAAS Publications, Washington, DC. 1990.

HOFFMAN, A. J.; WOLFE, P. History in The Traveling Salesman Problem, Lawler, Lenstra, Rinooy Kan and Shmoys, eds., Wiley, 1-16, 1985.

HOFFMAN, K.; PADBERG, M. LP-based Combinatorial Problem Solving, Annals of Operations Research 4, 145-194, 1985.

HOFFMAN, K.; PADBERG, M. Traveling Salesman Problem. Karla Hoffman home page, Encyclopedia Articles. Disponível na Internet no site http://iris.gmu.edu/%7Ekhoffman/papers/trav_salesman.html, acessado em 8 de novembro de 2005.

HOLLAND, J. H. Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. Universidade de Michigan Press. 1975.

JERALD, J.; ASOKAN, P.; PRABAHARAN, G.; SARAVANAN, R. Scheduling optimization of flexible manufacturing systems using particle swarm optimization algorithm. Advanced Manufacturing Technology. Springer-Verlag London Limited. 2004.

JFINGER, M.; REINELT, G.; THIENEL, S. Provably good solutions for the traveling salesman problem. Zeitschrift ffir Operations Research 40, 183-217. 1994.

JONKER, R.; VOLGENANT, T. Transforming asymmetric into symmetric traveling salesman problems, Operations Research Letters 2, pp 161-163. 1983.

JOHNSON, D. S. A Theoretician's Guide to the Experimental Analysis of Algorithms, In: M. H. Goldwasser, D. S. Johnson, L. A. McGeoch, (eds.): Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges, American Mathematical Society, Providence, pages 215-250, 2002.

JOHNSON, D. S.; MCGEOCH, L. A. Experimental Analysis of Heuristics for the STSP, em G. Gutin e A. Punnen, eds., The Traveling Salesman Problem and Its Variations, Kluwer Academic Publishers, Boston, pp. 369-443, 2002.

KANELLAKIS, P. C.; PAPADIMITRIOU, C. H. Local Search for the Asymmetric Traveling Salesman Problem. Operations Research 28:1086-1099, 1980.

KAPLAN, E. L.; MEIER, P. Nonparametric estimation from incomplete observations. Journal of American Statistical Association, 53, p. 457-481, 1958.

KARP, R. M. On the Computational Complexity of Combinatorial Problems, Networks 5, 45-68, 1975.

KENNEDY, J.; EBERHART, R. Particle swarm optimization. Neural Networks. Proceedings, IEEE International Conference on. Perth, WA, Australia, vol. 4, 1942-1948. 1995.

KIRKPATRICK, S.; GELATT, C. D.; VECCHI, M. P. Optimization by Simulated Annealing. Science, vol. 220, pp.671-680. 1983.

KORTE, B. H. Applications of combinatorial cptimization, in: Mathematical Programming: Recent Developments and Applications, M. Iri and K. Tanabe, Eds., Kluwer, Dordrecht, pp. 1-55. 1989.

KOZA, J. R. Survey of Genetic Algorithms and Genetic Programming. Depto. de Ciência da Computação. Universidade de Stanford. Stanford, Califórnia, 1995.

KRASNOGOR, N.; SMITH, J. A. A Memetic Algorithm With Self-Adaptative Local Search: TSP as a case study. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO2000), 2000.

LARRANAGA, P.; KUIJPERS, C. M. H.; MURGA, R. H.; INZA, I.; DIZDAREVIC, S. Genetic Algorithms for the Traveling Salesman Problem: A Review of Representations and Operators. Ciências da Computação e Inteligência Artificial. Universidade de Basque, San Sebatián, Espanha, 1999.

LAPORTE, G. The Traveling Salesman Problem: An overview of exact and approximate algorithms. European Journal of Operational Research, 59(2):231--247, 1992.

LIN, S.; KERNIGHAN, B. An Effective Heuristic Algorithm for the Traveling Salesman Problem. Operational Research, 21, 1973, Pages 498-516.

LUCENA, A.; BEASLEY, J. E. A Branch and Cut Algorithm for the Steiner Problem in Graphs, Networks: An International Journal, vol. 31, pp. 39-59, 1998.

LVBJERG, M.; RASMUSSEM, T. K.; KRINK, T. Hybrid particle sarm optimizer with breeding and subpopulations. In Proceedings of Gnetic and Evolutionary Computation Conference, volume 1, pages 469-476, 2001.

MACHADO, T. R., LOPES, H. S. A hybrid particle swarm optimization model for the traveling salesman problem. In: Ribeiro, H., Albrecht, R. F., Dobnikar, A. (eds.): Natural Computing Algorithms, Wien: SpringerWienNewYork, 255-258. 2005.

MANTEL, N.; HAENSZEL, W. Statistical spects of the analysis of data from retrospective studies of disease. Journal of the National Cancer Institute, 22, p. 719-748. 1959.

MARGULIS, L. Symbiosis as a Source of Evolutionary Innovation: Speciation and Morphogenesis, The MIT Press. 1991.

MARTIN, O.; OTTO, S.W.; FELTEN, E.W. Large-Step Markov chains for traveling salesman problem. Complex Systems 5, 299-326, 1991.

MERZ, P. Memetic algorithms for combinatorial optimization problems: Fitness landscapes and effective search strategies, Ph.D. Thesis, Parallel Systems Research Group. Department of Electrical Engineering and Computer Science. University of Siegen, 2000.

METROPOLIS, N.; ROSENBLUTH, A. W.; ROSENBLUTH, M. N.; TELLER, A.H.; TELLER, E. Equation of State Calculation by Fast Computing Machines. Journal of Chemical and Physical, v. 21, 1087-1092. 1953.

MICHALEWICZ, Z; FOGEL, D. B. Evolutionary computation 2: Advance algorithms and operators. Bristol, UK: IOP, 2000.

MOSCATO, P. On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithm. Caltech Concurrent Computation Program. California Institute of Technology. USA, 1989.

MOSCATO, P. An Introduction to Population Approaches for Optimization and Hierarchical Objective Functions: A Discussion on the role of Tabu Search. Annals of Operations Research , Vol. 41, Number 1-4, pp. 85-121, 1993.

MOSCATO, P.; NORMAN, M.G. A Memetic Approach for the Traveling Salesman Problem - Implementation of a Computational Ecology for Combinatorial Optimization on Message-Passing Systems. M. Valero, E. Onate, M. Jane, J.L. Larriba, and B. Suarez (Eds.): Parallel Computing and Transputer Applications, IOS Press, Amsterdam.

NAKA, S.; GENJI, T.; YURA, T.; FUKUYAMA, Y. A hybrid particles swarm optimization for distributionstate estimation. IEEE Trasnsactions on Power Systems, 60-68, 2003.

NEMHAUSER, G.; WOLSEY, L. Integer and Combinatorial Optimization, John Wiley&Sons, 1988.

NETO, D. Efficient Cluster Compensation for Lin-Kernighan Heuristics. Departamento de Ciência da computação, Universidade de Toronto, 1999.

NGUYEN, H. D.; YOSHIHARA, I.; YAMAMORI, K.; YASUNAGA, M. NGUYEN-YOSHIHARA-YAMAMORI-YASUNAGA LIN-KERNIGHAN VARIANT. Disponível na Internet em http://public.research.att.com/~dsj/chtsp/results.html com acesso em 6 de abril de 2006.

OLIVER, M.; SMITH, D. J.; HOLLAND, J. R. C. A study of permutation crossover operators on the traveling salesman problem, Genetic algorithms and their applications. Second Int. Conf. On Genetic Algorithms, J. J. Grefenstette, ed., Hillsdale, NJ, Lawrence Erlbaum Assoc. 1987.

OZCAN, E.; ERENTURK, M. A Brief Review of Memetic Algorithms for Solving Euclidean 2D Traveling Salesrep Problem. Department of Computer Engineering, Universidade de Yeditepe , Istanbul, Turquia. 2004.

PADBERG, M. RINALDI, G. A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems," SIAM Review, vol. 33, pp. 60-100,March 1991.

PADBERG, M. RINALDI, G. Optimization of a 532-city Symmetric Traveling Salesman Problem by Branch and Cut, Operations Research Letters, vol. 6, pp. 1-7, 1987.

PANG, W.; WANG, K. P.; ZHOU, C. G.; DONG, L. J.; LIU, M.; ZHANG, H. Y.; WANG, J. Y. Modified particle swarm optimization based on space transformation for solving traveling salesman problem. Proceedings of the Third International Conference on Machine Learning and Cybernetics, 2342-2346, 2004.

PARSOPOULOS, K. E.; VRAHATIS, M. N. Recent approaches to global optimization problems through Particle Swarm Optimization. Natural Computing 1, 235-306. 2002.

POHLHEIM, H. Evolutionary Algorithms: Overview, Methods and Operators. Documentation for: Genetic and Evolutionary Algorithm Toolbox for use with Matlab, version 3.3, 2001.

PITSOULIS, L. S.; RESENDE, M. G. C. Greedy Randomized Adaptive Search Procedure. Handbook of Applied Optmization. Oxford University Press: 2002, pag 168-183.

RAMOS, I. C. de O. Metodologia estatística na solução do problema do caixeiro viajante e na avaliação de algoritmos: um estudo aplicado à transgenética computacional. Tese de Doutorado, Universidade Federal do Rio Grande do Norte, Centro de Tecnologia, Programa de Pós-graduação em Engenharia Elétrica. Natal – RN. 2005.

RARDIN, R. L.; UZSOY, R. Experimental Evaluation of Heuristic Optimization Algorithms: A Tutorial, Journal of Heuristics 7(3), 2001, pages 261-304.

RATLIFF, H. D.; ROSENTHAL, A.S. Order-Picking in a Rectangular Warehouse: A Solvable Case for the Traveling Salesman Problem, PDRC Report Series No. 81-10. Georgia Institute of Technology, Atlanta, Georgia, 1981.

REINELT, G. TSPLIB: biblioteca de instâncias para o PCV. http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/ último acesso em 06 de maio de 2006.

REYNOLDS, C. W. Flocks, herds and schools: a distributed behavioral model. Computer Graphics, 21(4), 24-34. 1987.

RODRIGUES, M. A. P. Problema do Caixeiro Viajante: um algoritmo para resolução de problemas de grande porte baseado em busca local dirigida. Dissertação de mestrado submetida à Universidade Federal de Santa Catarina, Programa de Pós-Graduação em Engenharia de Produção, Florianópolis, Santa Catarina, Brasil. 2000.

SAXE, J.B. Dynamic programming algorithms for recognizing small-bandwidth graphs in polynomial time. SIAM J. Algebraic and Discrete Methods, 1:363-369, 1980.

SCHMIDT, C. C. Uma Abordagem através de Algoritmos Transgenéticos para o Problema da Configuração de uma Rede de Distribuição de Gás Natural. Relatório Técnico, Universidade Federal do Rio Grande do Norte, Departamento de Informática e Matemática Aplicada. Natal, Brasil. 2006.

SECREST, B. R. Traveling salesman problem for surveillance mission using particle swarm optimization. AFIT/GCE/ENG/01M-03, Air Force Institute of Technology. 2001.

SILVA, M. B. da; DRUMMOND, L. M. A.; OCHI, L. S. Metaheurísticas GRASP + VNS para a solução de Problemas de Otimização Combinatória. Artigo a ser apresentado no XXXII Simpósio Brasileiro de Pesquisa Operacional – SBPO. 2000.

SOUZA, G. R. Simulated Annealing Aplicado ao Problema do Caixeiro Viajante. Notas de aula, Metaheurísticas, Universidade Federal do Rio Grande do Norte (UFRN). 2005.

SOUZA, G. R. Uma Abordagem por Nuvem de Partículas para Problemas de Otimização Combinatória. Dissertação de Mestrado, Universidade Federal do Rio Grande do Norte, Centro de Ciências Exatas e da Terra, Programa de Pós-Graduação em Sistemas e Computação, Fevereiro de 2006.

TAMAKI, H.; TOKUYAMA, T. A characterization of planar graphs by pseudo-line arrangements. Algorithmica 35(3):269–285. 2003.

THIENEL, S. A simple TSP-solver, Technical report, Institut fur Informatik, Universitat zu Koln, 1996.

TSCHÖKE, S.; LÜBLING, R.; MONIEN, B. Solving the traveling salesman problem with a distributed branch-and-bound algorithm on a 1024 processor network. 9th International Parallel Processing Symposium p. 182, 1995.

WANG, K. P.; HUANG, L.; ZHOU, C. G.; PANG, W. Particle swarm optimization for traveling salesman problem. Proceedings of the Second International Conference on Machine Learning and Cybernetics, Xi'an. 1583-1585. 2003.

WOLSEY, L. A. Integer Programming, Wiley-Interscience Series in Discrete Mathematics and Optimization, John Wiley&Sons, 1998.

YOSHIDA H.; KAWATA, K.; FUKUYAMA, Y. A particle swarm optimization for reactive power and voltage control considering voltage security assessment. IEEE Transactions on Power Systems 15, 1232-1239. 2001.

ZACHARIASEN, M.; DAM, M. Tabu Search on the Geometric Traveling Salesman Problem. Presented at Metaheuristics International Conference 95, Breckenridge, Colorado, USA. 1995.