JOÃO GUILHERME SAUER

ABORDAGEM DE EVOLUÇÃO DIFERENCIAL HÍBRIDA COM BUSCA LOCAL APLICADA AO PROBLEMA DO CAIXEIRO VIAJANTE

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia de Produção e Sistemas da Pontifícia Universidade Católica do Paraná como requisito parcial para obtenção do título de Mestre em Engenharia de Produção e Sistemas.

CURITIBA

2007

JOÃO GUILHERME SAUER

ABORDAGEM DE EVOLUÇÃO DIFERENCIAL HÍBRIDA COM BUSCA LOCAL APLICADA AO PROBLEMA DO CAIXEIRO VIAJANTE

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia de Produção e Sistemas da Pontifícia Universidade Católica do Paraná como requisito parcial para obtenção do título de Mestre em Engenharia de Produção e Sistemas.

Área de Concentração: Automação e Controle de Processos.

Orientador: Prof. Dr. Leandro dos Santos Coelho

CURITIBA

2007

SAUER, João Guilherme.

ABORDAGEM DE EVOLUÇÃO DIFERENCIAL HÍBRIDA COM BUSCA LOCAL APLICADA AO PROBLEMA DO CAIXEIRO VIAJANTE, 2007. 100P.

Dissertação – Pontifícia Universidade Católica do Paraná. Programa de Pós-Graduação em Engenharia de Produção e Sistemas.

1. Problema do Caixeiro Viajante 2. Otimização Combinatória I. Pontifícia Universidade Católica do Paraná. Centro de Ciências Exatas e de Tecnologia. Programa de Pós-Graduação em Engenharia de Produção e Sistemas.

“Só sei que nada sei”

Sócrates (470 a.C. - 399 a.C.)

Agradecimentos

Gostaria de agradecer a todas as pessoas por terem, de alguma forma, contribuído para a conclusão desta dissertação de mestrado. De modo especial, gostaria de agradecer:

Ao meu orientador, Leandro dos Santos Coelho, que me ajudou em todas as etapas deste trabalho, sempre me incentivando e me ajudando em tudo que fosse necessário.

Aos meus familiares, que sempre me incentivaram, mesmo nos momentos mais difíceis. Em especial aos meus pais, Nestor João Sauer e Ana Maria Sauer que foram para mim uma fonte de estímulo e perseverança e a minha esposa Elaina Lourenço Sauer que nos momentos difíceis estava ao meu lado me incentivando e me ajudando no que fosse necessário.

Gostaria ainda de fazer uma menção honrosa aos meus avós, que partiram no meio desta minha jornada. A eles, agradeço ao apoio que sempre me deram e o orgulho que tinham de saber que um neto fazia um mestrado. Sei que levarei comigo todos os ensinamentos que recebi.

À Pontifícia Universidade Católica do Paraná, por ter dado todo apoio material e financeiro de que precisei.

Por fim, gostaria de agradecer a todos os meus colegas do trabalho e universidade, que sempre estiveram dispostos a me ajudar no que fosse possível.

A todos, o meu muito obrigado!

Resumo

O Problema do Caixeiro do Viajante (PCV) é um dos mais clássicos problemas da área de Otimização Combinatória. O objetivo do PCV é determinar um caminho pelo qual passe por todos os nós (cidades) de um grafo apenas uma vez (caminho hamiltoniano) com um menor *tour*, isto é, menor valor de função objetivo (distância euclidiana). Um dos aspectos envolvidos neste problema, e certamente uma das questões das áreas de Otimização e Computação, é verificar se existe algum algoritmo que consiga resolver o PCV em um determinado tempo polinomial. Até prova em contrário, o PCV é classificado como um problema de otimização NP (Não Polinomial) difícil. Neste contexto, diversas técnicas de otimização denominadas metaheurísticas têm sido propostas na literatura, entre as quais: algoritmos genéticos, *simulated annealing*, colônia de formigas e busca tabu. Esta dissertação propõe um estudo comparativo de alguns algoritmos de otimização. Os algoritmos de otimização abordados são os seguintes: (i) uma nova proposta de evolução diferencial com representação discreta para o PCV, (ii) evolução diferencial com método de *Lin-Kernighan* Helsgaun (LKH) e (iii) evolução diferencial com Busca na vizinhança e com método *Lin-Kernighan Helsgaun*. O método de evolução diferencial é uma metaheurística que divide a população em diferentes vetores que podem convergir. Já o LKH se baseia na troca seqüencial de vários nós através de buscas locais. Por fim, o VNS, sigla em inglês para Busca em Vizinhança Variável (VNS) é uma metaheurística de busca local, que divide a população em partes menores e faz uma busca local separadamente para cada parte, até que se consiga abranger toda a população. Os resultados obtidos em problemas que podem ser encontrados na biblioteca TSPLIB, são comparados com resultados apresentados na literatura do PCV. O resultado obtido comprova que o método LKH é o melhor para se conseguir um resultado ótimo, porém, comprovou-se que para grandes instâncias, o ED+VNS melhorou o tempo de busca no LKH. Já para instâncias pequenas, o resultado obtido ficou igual, com ou sem a entrada de um *tour* inicial fornecido através dos resultados do ED+VNS.

Palavras-Chave: Problema do Caixeiro Viajante, Otimização Combinatória, Evolução Diferencial, Busca Local, Busca em Vizinhança Variável.

Abstract

The Traveling Salesman Problem (TSP) is one of the most difficult problems in the Combinatorial Optimization area. The goal of TSP is to find one path that can travel between all the nodes (instances) of the graph just once (Hamiltonian tour) in the smallest tour, that is, smallest Euclidian distance. One of the main questions in this problem, and certain is one of the most important in Optimization and Computation fields, is verify that there are any kind of algorithm that can solve the TSP in polynomial time. The TSP is classified like a NP (Not Polynomial)-Hard. In this context, many other meta-heuristics techniques have been propose in literature, such as simulated annealing, genetic algorithm, ant colony, and tabu search. The evaluate optimization approach for the TSP are: (i) discrete new differential evolution approach to TSP; (ii) differential evolution with local search using *Lin-Kernighan Heulsgaun* (LKH) method and (iii) differential evolution with local search based on Variable Neighbor Search (VNS) and together with *Lin-Kernighan Heulsgaun* method. The differential evolution is an metaheuristcs that split the population in different vetor that can converge between them. The LKH method is based in the sequencial change of the instances using local search procedure. Finally, Variable Neighbor Search using Local Search, but first divide the population in small parts and make a search first in only one population and after find a good solution for that population, join the neighbor to start a new local search. In the end, all the population is used in the local search procedure. The results evaluated in the TSP that can be find in the TSPLIB library. Those results are compared with the other results in the recent TSP literature. The obtained results show that LKH method is the best method to reach an optimal result, but for largest problems, the ED+VNS improve the simulation results. Using smallest problems, the results was the same either using the ED+VNS or just LKH using default input, it generates randomly.

Key-Words: Traveling Salesman Problem, Combinatorial Optimization, Differential Evolution, Local Search, Variable Neighbor Search.

Sumário

Lista de Figuras

Lista de Quadros

Lista de Tabelas

Lista de Abreviaturas

AA	Analizador de Agrupamento
AG	Algoritmos Genéticos
AI	Agrupador Iterativo
AO	Algoritmo de Otimização local
ED	Evolução Diferencial
GRASP	*Greedy Randomized Adaptive Search Procedure*
LK	*Lin-Kernighan*
LKH	*Lin-Kernighan Helsgaun*
LUT	*Look-Up Table*
MH	MetaHeurística
MIPS	Milhões de Instruções por Segundo
MT	Máquina de Turing
NP	*Non-Polynomial*
PC	*Personal Computer*
PCV	Problema do Caixeiro Viajante
RAM	*Random Access Memory*
RW	*Read/Write*
SA	*Simulated Annealing*
SS	*Scatter Search*
TSP	*Traveling Salesman Problem*
VNS	*Variable Neighbourhood Search*

Capítulo 1

Introdução

A Otimização Combinatória é uma área da Computação que estuda problemas de otimização em um domínio finito de soluções.

Em um problema de otimização combinatória tem-se uma (ou mais) função(ões) objetivo e um conjunto de restrições (opcional), ambos relacionados às variáveis de decisão pertencentes ao domínio dos números inteiros. O problema pode ser de minimização ou de maximização de uma função objetivo. A resposta para o problema à solução é de minimização (ou maximização) do valor de uma função objetivo para o qual o valor atribuído às variáveis não viole nenhuma restrição do problema (caso estas existam).

O Problema do Caixeiro Viajante (PCV) é um dos mais tradicionais problemas de otimização combinatória. Existem casos nos quais se pode aplicar o PCV, entre os quais pode-se citar a otimização de movimento de robôs em uma fábrica [KIT99], roteamento de veículos [ALV04] e roteamento de linhas telefônicas [HO03]. Outras áreas de aplicação incluem a Estatística (análise de dados), a Economia (matrizes de entrada/saída), a Física (estados de energia mínima), a Biologia molecular e proteínas, entre outras [HOL75], [MOR97].

1.1 Motivação e Justificativa

O objetivo do PCV é encontrar um caminho no qual percorra apenas uma vez cada um dos nós do grafo (instâncias) com menor custo, isto é, a menor soma da distância Euclidiana entre os nós de um grafo.

A solução do PCV é muito discutida no meio acadêmico, pois o tempo computacional para testar todas as soluções (busca exaustiva) é de crescimento fatorial. Por exemplo, para calcular o melhor trajeto de 10 cidades (n=10) em um computador de tecnologia atual, que suporta fazer até 2000 MIPS (Milhões de instruções por segundo), seria de apenas alguns nanosegundos. Para se calcular a quantidade de caminhos possíveis utiliza-se a seguinte equação: $\frac{(n-1)!}{2}$. Logo, para 10 cidades seriam apenas 181.440 caminhos possíveis. Para se calcular a quantidade de resultados por segundo que o computador pode verificar, utiliza-se a seguinte equação:

$$\left(\frac{MIPS}{n-1}\right) \tag{1.1}$$

onde n é a quantidade de rotas. Porém, se for aumentado para 20 cidades (n=20), o tempo de processamento passa a ser aproximadamente 36 anos, pois serão calculados aproximadamente 105 milhões de rotas por segundo e aproximadamente $0{,}6\text{x}10^{17}$ rotas possíveis. O tempo necessário seria então $\left(\frac{0{,}6\text{x}10^{17}}{1{,}05\text{x}10^{8}} = 5{,}79\text{x}10^{8}\right)$ segundos ou seja, aproximadamente 36 anos para a resolução do problema.

Devido a este motivo é necessária à elaboração de técnicas de otimização visando à obtenção de um custo computacional aceitável, uma boa solução para o PCV.

Além disso, deve ser mencionado que a importância do PCV é devida:

- Sua ampla aplicação prática, uma vez que pode ser aplicado em diversos tipos de projetos nas mais diversas áreas do conhecimento;
- Uma forte relação com outros modelos de otimização combinatória já existentes;
- Dificuldade da determinação de uma solução ótima para instâncias de maior dimensão.

Esta dissertação realiza um estudo comparativo de alguns algoritmos de otimização. Os algoritmos de otimização abordados são os seguintes: (i) uma nova proposta de evolução diferencial com representação discreta para o PCV, (ii) evolução diferencial com método de *Lin-Kernighan Helsgaun (LKH)*, e (iii) evolução diferencial com Busca na vizinhança e com método *Lin-Kernighan Helsgaun*. O método de evolução diferencial é uma metaheurística que emprega a população de vetores solução, que apresenta potencialidade de fácil implementação e robustez para problemas de otimização global. Já o LKH se baseia na troca seqüencial de várias instâncias através de buscas locais. Por fim, a Busca em Vizinhança Variável (VNS,

sigla em inglês para *Variable Neighbor Search*) é uma metaheurística de busca local, que divide a população em partes menores e faz uma busca local separadamente para cada parte, até que consiga abranger toda a população. Os resultados obtidos em problemas que podem ser encontrados na biblioteca TSPLIB [REI91], são comparados com resultados apresentados na literatura do PCV.

1.2 Métodos Abordados

O tipo de metodologia usada nesta dissertação está vinculada a proposta e validação de uma nova abordagem para o PCV, conhecida como Evolução Diferencial (ED) e também a configuração de sistemas híbridos combinando ED, LKH e VNS.

Nesta proposta é apresentado o algoritmo evolutivo denominado Evolução Diferencial que foi primeiramente proposto por Storn e Price, em 1995 [STO95]. Este método está sendo difundido em várias áreas de pesquisa, por sua facilidade de implantação, no qual, muitas vezes, não passa de 20 linhas de código computacional. A idéia geral do método é gerar novos indivíduos (soluções potenciais), denominados doadores ou vetores modificados, pela adição da diferença vetorial ponderada entre os parâmetros de indivíduos da população escolhidos aleatoriamente.

Originalmente, o ED foi proposto para solucionar problemas de otimização contínua, porém, modificou-se o código original para que fossem solucionados problemas de otimização discreta, que é aonde está localizado o PCV. Para tal modificação, ordenou-se os valores gerados pelo ED, fazendo que fosse gerada uma lista de indivíduos que juntos criam um ciclo Euclediano.

Em contra-partida, o método de Lin-Kernighan [LIN73] é um método consagrado na resolução do PCV, uma vez que é um dos métodos mais poderosos e robustos existentes para tal aplicação. A idéia geral de LK, por sua vez, é concretizar substituições contínuas entre as rotas do PCV para que se possa melhorar os roteiros já existentes. Este tipo de metodologia é uma das mais rápidas e se consegue chegar a resultados promissores com sua aplicação.

Já a metaheurística VNS é um método que utiliza também internamente uma busca local. Sua idéia é fazer com que se faça buscas locais em pequenas instâncias e conforme se melhore os resultados, aumente cada vez mais a quantidade de vizinhos até que acabe fazendo uma busca local em todas as possíveis instâncias.

Porém, os métodos avaliados nesta dissertação começam com um *tour* aleatório e a partir dele tenta-se chegar a uma rota com uma menor função custo, ou seja, menor soma da distância Euclidiana de um *tour* completo. Por isso, a proposição de sistemas híbridos, aonde se substitui a aleatoriedade inicial pela resposta conseguida em outro sistema, é também uma das abordagens de otimização tratadas por este trabalho. O ED, por ser um método de otimização geralmente usado para busca global, não é um método eficiente para se garantir um resultado-ótimo. Entretanto, a ED pode gerar um *tour* inicial que auxilie o LKH a obter tal resultado. Porém, para se melhorar ainda mais o *tour*, refina-se o resultado obtido através do ED passando primeiramente para o VNS, que por sua vez irá fazer uma busca local para poder obter um melhoramento do *tour*, inicialmente gerado pelo ED. Desta forma, um resultado melhorado é migrado para o LKH para que o mesmo possa gerar resultados próximos aos valores ótimos para o PCV para instâncias da TSPLIB.

1.3 Organização da Dissertação

O restante da dissertação está organizada em mais 4 capítulos. O Capítulo 2 discute uma breve revisão da literatura sobre Otimização Combinatória (PCV e Teoria de Grafos) e Teoria da Complexidade. Em seguida, o Capítulo 3 apresenta um apanhado dos principais algoritmos utilizados para resolver, utilizando metaheurísticas, o PCV. O Capítulo 4 mostra os resultados obtidos após a execução dos métodos citados anteriormente. Por fim, o Capítulo 5 apresenta-se as conclusões e propostas de projetos futuros.

Capítulo 2

Revisão da Literatura

Neste capítulo são apresentados os conceitos gerais sobre Teoria da Complexidade, Otimização Combinatória e os detalhes sobre o PCV e a Teoria de Grafos.

2.1 Otimização Combinatória

O objetivo dos problemas de otimização combinatória é atribuir valores a um conjunto de variáveis de decisão, de tal modo que uma função com base nestas variáveis (função objetivo) seja minimizada ou maximizada na presença de um conjunto de restrições [LAW85], [LIN65].

Matematicamente, pode-se definir um problema de otimização combinatória através de um conjunto finito representado por $N=(1,...,n)$, com ponderação (peso) c_j associadas a cada $j \in N$, e um conjunto F formado por subconjuntos viáveis de N.

Desta forma, deseja-se determinar os elementos de F, tais que o somatório das ponderações associadas S seja mínimo, isto é, determinar:

$$\min_{S \subseteq N} \left\{ \sum_{j \in S} c_j : S \in F \right\} \qquad \textbf{(2.1)}$$

Em problemas deste tipo, uma solução trivial para se resolver este tipo de problema seria verificar todas as possíveis soluções e a partir dos resultados verificar qual seria a melhor solução. Porém, em problemas de maior porte, ocorre uma explosão combinatória, pois o tempo computacional para solucioná-las, cresce de maneira fatorial, o que impossibilita testar todas as possíveis soluções até se encontrar a de menor custo. Por exemplo, para a

resolução do PCV assimétrico, as possíveis soluções são da ordem de $(n-1)!$, onde *n* é a quantidade de instâncias.

Note pelo apresentado na Tabela 1, que usando como exemplo um modelo computacional que seja possível resolver 10^{20} soluções por segundo em determinado computador se forem testadas todas os possíveis *tours* para um problema do PCV, seria praticamente impossível de se conseguir um resultado ótimo em um período de tempo aceitável. O que se tem proposto, recentemente, para se resolver sistemas deste tipo, são sistemas que primeiro tentam alcançar um resultado promissor em poucas avaliações da função objetivo. Tais sistemas são explicados mais adiante nesta proposta no capítulo 4, que explica sobre metaheurísticas.

n	$(n-1)!$	**Em dias**	**Em anos**	**Em bilhões de anos**
10	$1{,}81.10^{5}$	$2{,}10.10^{-20}$	$5{,}75.10^{-23}$	$5{,}75.10^{-31}$
100	$4{,}67.10^{155}$	$5{,}40.10^{130}$	$1{,}48.10^{128}$	$1{,}48.10^{119}$
1000	$2{,}01.10^{2564}$	$2{,}33.10^{2539}$	$6{,}38.10^{2536}$	$6{,}38.10^{2527}$

Tabela 1: Explosão combinatória e tempo estimado

2.2 Máquina de Turing

A Máquina de Turing (MT) é um modelo computacional, proposto inicialmente por Alan Turing [TUR36], para descrever o mínimo indispensável para se obter um método efetivo de computação.

Existem vários tipos de máquinas criadas por Turing. Porém, todos partem de um modelo básico, este explicado na próxima subseção.

2.2.1 Máquina de Turing básica

Conforme [MOR01], uma MT é composta por 2 componentes principais:

- Uma fita dividida em células, que tem um comprimento infinito;
- Uma cabeça de leitura/escrita (RW) que atua sobre uma célula da fita por vez.

Supondo-se que a fita é infinita à esquerda e à direita. No início, supõe-se que os dados (seqüência finita de símbolos de um alfabeto) se encontram escritos na fita, um símbolo em cada célula, e as restantes células da fita contêm um caractere especial da fita, designado por caractere branco. No exemplo da Figura 1 tem-se uma fita dividida em células e um leitor que lê apenas uma célula por vez.

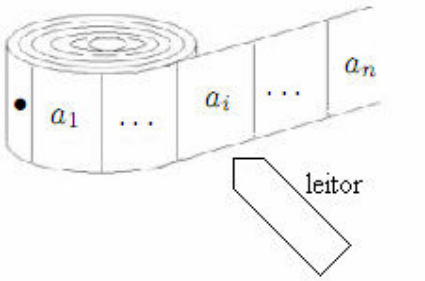

Figura 1: Representação gráfica de uma Máquina de Turing.

Num movimento, dependendo do símbolo lido na fita pela cabeça e do estado do controlo finito, a máquina tem as seguintes possibilidades:

1. muda de estado;
2. escreve um símbolo na célula que está debaixo da cabeça;
3. move a cabeça à esquerda ou à direita.

Formalmente, uma MT é um *n-upla*, tal que:

$$M = (S, \Sigma, \Gamma, \delta, s_0, b, F) \tag{2.2}$$

na qual

S é um conjunto finito de estados;

Γ é o conjunto finito de símbolos da fita;

Σ é um subconjunto de S x Γ, que não inclui •. É o conjunto dos símbolos de entrada;

δ é a função de transição, função parcial de S x Γ em S x Γ x {→, ←};

s_0 é o estado inicial;

b é um símbolo de Γ, designado por branco;

$F \subseteq S$ é o conjunto de estados finais.

Existem máquinas de Turing determinísticas e não-determinísticas. As determinísticas são aquelas que quando estão em um certo estado, lendo um certo dado, podem se movimentar de um único modo rumo à próxima configuração. Entretanto, as não-determinísticas podem se mover para diversas configurações, a partir do dado lido e da configuração interna atual. Evidentemente as máquinas determinísticas formam uma subclasse das não-determinísticas.

2.3 Teoria da Complexidade

Estimar a complexidade de um algoritmo é uma tarefa crucial. O caminho de medir empiricamente o desempenho de um algoritmo esbarra em dificuldades que vão desde o equipamento e compilador empregado até as habilidade do programador.

A Complexidade Computacional é um ramo da Matemática Computacional que estuda a eficiência dos algoritmos. Do ponto de vista prático, de nada nos adianta um algoritmo "perfeito" se sua implementação computacional demora uma centena de anos para ser processada [TUR36].

Mesmo tarefas relativamente simples, como o produto de dois números com muitos dígitos, podem demorar alguns minutos para ser concluído nos atuais computadores. Se for considerado que alguns algoritmos necessitam multiplicar números grandes milhares de vezes, esses alguns minutos podem se transformar em um tempo excessivamente longo.

Para a medição da eficiência de um algoritmo, gera-se para cada operação básica do programa a ser testado um valor fixo na unidade de tempo. Assim, pode-se comparar vários programas sem estar levando em conta o ambiente computacional que o mesmo foi testado. Se a dependência do tempo com relação aos dados de entrada for polinomial, o programa é considerado rápido. Se, entretanto, a dependência do tempo for exponencial o programa computacional é considerado lento.

2.3.1 Definição

A teoria da complexidade pode ser definida através de um *esquema de representação,* formado por palavras de 1's e 0's que representam as instâncias e as soluções do problema. Com efeito, denomina-se problema a um subconjunto Π de $\{0,1\}^* \times \{0,1\}^*$, onde $\{0,1\}^*$ denota o conjunto de todas as palavras finitas de 0's e 1's. Cada palavra $\sigma \in \{0,1\}^*$ é denominada instância ou entrada de Π e cada $\tau \in \{0,1\}^*$ tal que $(\sigma, \tau) \in \Pi$ é denominada solução ou saída de Π. Assume-se que para cada entrada em Π existe pelo menos uma solução. Para um esquema de representação específico, define-se tamanho de uma entrada ($\boldsymbol{L}$) para uma instância de um problema, como o número de elementos que compõem a palavra que representa tal entrada. Um algoritmo é uma seqüência de passos que devem ser executados para a obtenção de uma solução para um problema. Neste contexto, algoritmos diferentes podem ser propostos para resolver um mesmo problema [MAR00].

Para um esquema de representação específico, define-se *tamanho de uma entrada (L)* para uma *instância* de um problema, como a quantidade de elementos que compõem a palavra que representa a entrada desse esquema específico.

Um algoritmo é um seqüenciamento lógico de passos que devem ser executados até se conseguir solucionar um determinado problema. Como existem várias maneiras de se resolver um mesmo problema, pode-se, às vezes, unir dois ou mais algoritmos para se alcançar um algoritmo mais rápido ou mais eficiente do que os mesmos separados.

Ainda conforme [MAR00], no caso de ser um esquema de representação apenas para aquele problema em específico, a *função de complexidade de tempo* $f : N \to N$ de um algoritmo, expressa o tempo máximo, apenas utilizando operações elementares, que é necessário para resolver qualquer instância de tamanho $n \in N$.

Algoritmo de tempo polinomial, é o algoritmo no qual a sua função de complexidade de tempo f é tal que $f(n) \leq p(n)$ para todo $n \in N$, para algum polinômio p .

Existe uma classe de problemas denominados *problemas de decisão*. Neste tipo de problemas existem apenas duas soluções possíveis (ou saídas), quais sejam "sim" ou "não", no qual se pode determinar que é uma decisão binária. A classe formada por todos os problemas de decisão que possuem um *algoritmo de tempo polinomial* é denominada *classe* P .

Uma outra classe de problemas de decisão é a denominada *classe* NP . Caso a seguinte propriedade: "Se a resposta para uma instância de *P* é "sim", então este fato pode ser provado em tempo polinomial". Seja aplicável ao problema, então pode-se dizer que a mesma é uma classe NP .

Vê-se que $P \subseteq NP$. Acredita-se que $P \neq NP$, embora não haja prova desse fato.

Uma transformação polinomial é um algoritmo que dada uma instância codificada de um problema de decisão Π , é capaz de transformá-la em tempo polinomial numa instância codificada Π' tal que: para toda instância $\sigma \in \Pi$ a resposta para σ é "sim", se e somente a resposta para a transformação de σ em uma instância $\sigma' \in \Pi'$ é "sim".

Um problema de otimização combinatória não implica que ele também seja um problema de decisão. Porém, pode-se transformar qualquer problema de otimização combinatória em um sistema de decisão. Seja o problema de otimização:

$$\min\{cx : x \in F\} \qquad \textbf{(2.2)}$$

onde x é uma representação de F e cx é a custo de x.

Pode-se transformar este problema de otimização no seguinte problema de decisão: "há um $x \in F$ tal que $cx \leq k$". Caso exista um sistema que consiga resolver o problema de minimização em tempo polinomial, então pode-se executar o seguinte: resolver o problema de minimização, em seguida o de decisão, comparando a solução gerada pelo primeiro com o valor k.

Por fim, [MAR00] conclui que caso haja um algoritmo capaz de resolver o problema de decisão em tempo polinomial, pode-se então resolver o problema de minimização através de sucessivos testes realizados para diferentes valores de k.

Sejam os problemas Π e Π'. Informalmente, uma redução de Turing de tempo polinomial de Π em Π' é um algoritmo A que resolve Π pelo uso de uma sub-rotina hipotética A' para resolver e Π' de tal modo que se A' fosse um algoritmo de tempo polinomial para Π', então A seria um algoritmo de tempo polinomial para Π.

Um problema de decisão Π é dito NP-completo, se P pertence a NP e todo problema em NP pode ser transformado em tempo polinomial para P. Como conseqüência dessa definição, tem-se que, se um problema NP-completo puder ser resolvido em tempo polinomial então todos os problemas NP também podem sê-lo. Neste sentido, os problemas NP-completos, são os problemas mais difíceis da classe NP.

Um problema Π é denominado NP-fácil se existe um problema $\Pi' \in NP$ tal que Π pode ser Turing reduzido a Π'. Um problema Π é chamado NP-difícil se existe um problema de decisão Π' NP-completo tal que Π' pode ser Turing reduzido a Π.

Uma visão aprofundada sobre complexidade pode ser vista em Garey e Johnson [GAR79] ou em Grötschel *et al*. [GRO88].

2.3.2 Máquina de Turing e Teoria da Complexidade

Primeiramente, precisa-se entender sobre a ligação que existe entre uma máquina de Turing e uma linguagem de programação. Conforme [MOR01], pode-se levar em consideração que se **L** é uma linguagem sobre um alfabeto **S**, isto é, se **L** é um subconjunto finito de seqüências de letras de **S**, diz-se que uma máquina de Turing **M** *aceita* a linguagem **L** se para toda palavra construída com as letras de **S** colocada como entrada (*input*), após o processamento **M** entra em um estado de aceitação (respondendo de algum modo "sim") se a palavra pertencer à linguagem. A palavra é *recusada* por **M** se, após o processamento **M** entra num estado de rejeição (respondendo "não" de algum modo) ou se ela falhar em completar sua execução computacional.

Neste contexto, o problema **P** *versus* **NP** pode ser representado através de um novo problema, tendo como base a MT:

"É verdade que toda linguagem aceita em tempo polinomial por uma máquina de Turing não-determinística é também aceita, em tempo polinomial, por uma máquina determinística?".

Caso essa pergunta tenha uma resposta afirmativa, tem-se uma resposta em tempo polinomial para problemas do tipo NP-difícil, no qual se encaixa o PCV. Com isso, quase todos os sistemas de segurança, que usam criptografia como sua base de segurança, poderiam também ser descriptografados em tempos polinomiais.

2.4 Problema do Caixeiro Viajante

O PCV é um problema do tipo NP-difícil. Porém, [REI91] e [REI94] vão mais além, ao afirmarem que o PCV é um dos mais proeminentes dentre um amplo conjunto de problemas de otimização combinatória. Segundo [REI94], o estudo do PCV tem atraído pesquisadores de diferentes áreas de conhecimento, entre as quais Pesquisa Operacional, Matemática, Física, Biologia, Inteligência Artificial, Computação e Engenharia. Isso se deve ao fato de que, apesar da simplicidade da sua formulação, no PCV é possível encontrar a maioria das questões que envolvem otimização combinatória. Conseqüentemente, o mesmo tem sido proposto como *benchmark* para avaliação de novos algoritmos e estratégias de soluções para problemas combinatórios, tais como busca tabu, algoritmos genéticos, *simulated annealing* e redes neurais.

O problema do PCV, segundo [CRO58], é que admitindo-se que um caixeiro viajante visite uma única vez cada uma das cidades (conhecido como caminho Hamiltoniano) que figuram numa lista e regresse à cidade donde partiu e ainda admitindo-se que ele conhece o custo da viagem entre qualquer par de cidades (podendo o custo não ser igual em ambos os sentidos, no qual se diz que o mesmo é um PCV assimétrico), ele obtenha uma seqüência de cidades que constitui o percurso associado a um custo total que seja o menor (ou maior) possível (denominado de percurso ótimo).

Na Figura 2 são apresentados todos os possíveis caminhos entre as capitais das regiões Sul e Sudeste. Desta forma, pode-se notar que existem diferentes *tours* para se fazer todo o trajeto, porém, alguns são maiores do que outros.

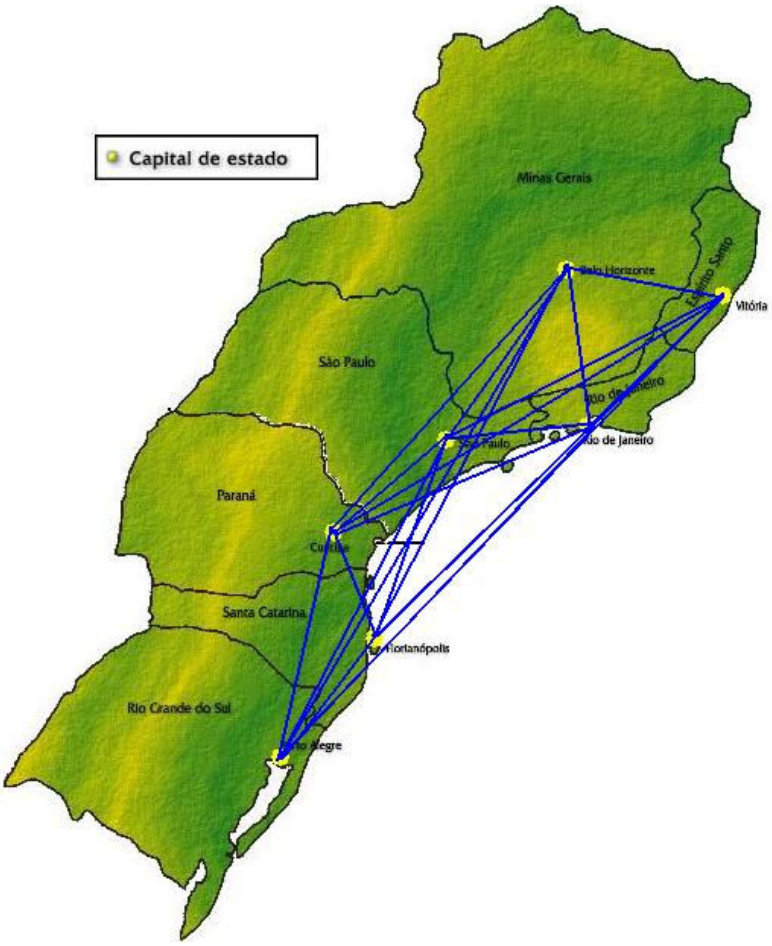

Figura 2: Todos os possíveis caminhos entre as capitais da Região Sul e Sudeste do Brasil. (Fonte: www.redebrasileira.com)

2.4.1 Histórico

Um breve histórico sobre o problema matemático pode ser encontrado no website desenvolvido por [APP95], que foi um dos desenvolvedores do método Concorde, que é um dos melhores programas computacionais já criados.

O problema matemático do caixeiro viajante nasceu originalmente através de um jogo que foi proposto pelo matemático Irlandês Sir Willian Rowan Hamilton e matemático Britânico Thomas Penyngton Kirkman no século XIX, no qual tinha-se que conseguir fazer uma "viagem" por 20 cidades, podendo usar apenas conexões específicas.

Já em 1930, Karl Menger em Viena e Harvard fizeram os seus primeiros estudos sobre o PCV e, mais tarde, por Hassler Whitney e Merril Flood de Princenton.

Desde então já apareceram vários estudos e técnicas para achar o melhor caminho, algumas delas serão detalhadas neste documento. Em 1962, por exemplo, houve um concurso

para quem conseguisse determinar o melhor caminho para os patrulheiros passarem por 33 cidades. Nesse caso, teve-se como ganhador o professor Gerald Thompson da Universidade de Carnegie.

Em 1977, Groetschel encontrou o melhor caminho para as maiores 120 cidades da Alemanha Ocidental.

Já Padberg e Rinaldi, em 1987, acharam a solução para 532 cidades dos Estados Unidos, em um trabalho feito na AT&T que na época era a maior empresa de telefonia do mundo.

No mesmo ano, teve-se o melhor caminho para se viajar ao redor do mundo conhecendo 666 lugares escolhidos como mais interessantes do mundo. O Brasil teve 13 lugares escolhidos. Este feito foi obtido pelos pesquisadores Groetschel e Holland [GRO87].

Novamente em 1987, foi realizada a otimização de 2392 pontos de um *layout*, mostrando que a aplicação do problema também pode ser usado em diversas áreas e não apenas para cidades, conforme proposto desde a criação do problema. Este feito foi obtido na Tektronics Incorporated pelos funcionários Padberg e Rinaldi.

Obteve-se em 1994 um novo recorde para 7397 cidades. Esse feito foi obtido nos laboratórios da AT&T Bell pelos pesquisadores Applegate, Bixty, Chvátal e Cook [APP95].

Finalmente em 1998, graças ao avanço tecnológico, As mesmas pessoas da AT&T Bell, quebraram a barreira da casa dos 10000 e chega-se então a 13509 cidades, que eram na época todas as cidades que continham mais de 500 habitantes no Estados Unidos da América.

Ainda com os mesmos pesquisadores, obteve-se um novo recorde em 2001, aonde se obteve a melhor rota para se visitar todas as 15112 cidades da Alemanha, que na época tornou-se um país reunificado.

Em 2004, [HEL04] conseguiu obter novamente o recorde de cidades. O mesmo grupo de pesquisadores, juntamente com Helsgaun, conseguiram mapear a melhor otimização para se visitar todas as 24978 cidades da Suécia. Este feito foi obtido em 14 meses, a contar da data do primeiro teste, em março de 2003, até a checagem do mesmo (maio de 2004). O equipamento computacional utilizado foi um super servidor em *cluster*, contendo 96 nós de servidores, sendo que cada um era um Intel Xeon de 2.8 GHz, com 2 Gbytes de memória. Todos eram interligados a uma rede de 100 Mbits.

Atualmente, foi desenvolvido o maior problema para o TSP. Adota-se, neste caso, um milhão de cidades ao redor do mundo, criou-se o maior problema já existente neste tipo de otimização. Atualmente, o *software* criado por Helsgaun é o que obteve o melhor resultado.

Infelizmente, por serem tantas cidades, ainda não é possível de se confirmar se o *tour* proposto pelo sistema é o melhor caminho ou não.

2.4.2 Problemas Correlatos

Existem variantes para o PCV que normalmente foram criadas para poder resolver um determinado problema no qual apenas o PCV clássico não poderia representar de maneira satisfatória. A seguir algumas variantes são apresentadas e brevemente apresentadas:

- PCV simétrico

O PCV é simétrico quando a função objetivo *C* associada ao seu arco, é $C_{ij} = C_{ji}$, para todo $i,j \in V$. Sendo assim, o PCV simétrico pode ser definido como um grafo não direcionado completo $G= (V,E)$ com n vértices $V=\{1,2,...,n\}$, onde E é o conjunto de todos os arcos conhecidos de i até j ou j até i, e $C_{ij} = C_{ji}$ [CRO80].

Pode-se ainda ter um PCV em que é necessário passar por determinadas seqüências de vértices, denominado de *cluster* ou agrupamento. Um exemplo é a rota que um determinado ônibus escolar deve percorrer em determinado bairro.

- PCV generalizado

Neste tipo de PCV, geram-se vários agrupamentos de nós, com isso é necessário percorrer um determinado número de nós para que se percorra o ciclo do caixeiro viajante [ONG82].

Existe outro tipo denominado de *equality*, no qual se exige que se passe por apenas um vértice de cada agrupamento para se completar o ciclo. Como se pode observar em configurações de circuitos integrados.

- PCV com Backhauls

Este tipo é em que se particiona dois grupos, nos quais o algoritmo é obrigado a passar em todos do primeiro grupo para só então ir para o outro grupo. Um exemplo é a entrega e recebimento de peças que um determinado caminhão é obrigado a fazer. O mesmo precisa primeiro descarregar todas as peças para só depois recolher [GOL00].

- PCV com Janelas de Tempo

Nesse caso pode-se fazer com que a resposta fique em uma determinada janela de tempo, aonde a resposta ótima seria nesses valores passados. Cada aresta dos vértices teria um determinado tempo adicionado a ela [SIM98].

- PCV múltiplo

Nesse problema é preciso obter vários caminhos, aonde se passa o início e o fim dos vértices. Usado quando tem-se mais de um caixeiro para se fazer à entrega[GOL00].

- PCV com gargalo ou MinMax

Pode-se obrigar que o ciclo do caixeiro tenha uma aresta com um tamanho máximo, ou seja, não pode-se ter longas distâncias para se alcançar o menor custo [BUR91].

- PCV com bônus

Podem-se colocar valores para cada aresta, aonde quando o caixeiro passar por ela, ele ganharia mais pontos, no final, pode-se exigir um bônus mínimo, para que se alcance o melhor ciclo [FIS88].

- PCV seletivo

Pode-se dizer que é um complemento do PCV com bônus, aonde pode-se agregar um bônus maior para um determinado subconjunto desde que o comprimento do ciclo não ultrapasse um valor pré-definido [LAP87].

- PCV Estocástico

Nesta situação tem-se o oposto ao PCV clássico, aonde tem-se todos os valores já definidos e sem alteração até o final do ciclo. Neste caso, pode-se ter os custos, os vértices e os tempos mudando constantemente, conforme o resultados obtidos durante a execução para a solucionar o PCV [KAO78].

2.4.3 Modelagem Matemática

Matematicamente, o PCV se encontra definido dentro da Teoria de Grafos. Um grafo é um conjunto de pontos, chamados vértices (ou nós), conectados por linhas, chamadas de arestas (ou arcos).

Dependendo da aplicação, arestas podem ou não ter direção, pode ser permitido ou não arestas ligarem um vértice a ele próprio e vértices e/ou arestas podem ter um peso (numérico) associado. Se as arestas têm uma direção associada (indicada por uma seta na representação gráfica) temos um grafo direcionado, ou dígrafo [COR01]. Um exemplo de um grafo com direção definida é apresentado na Figura 3.

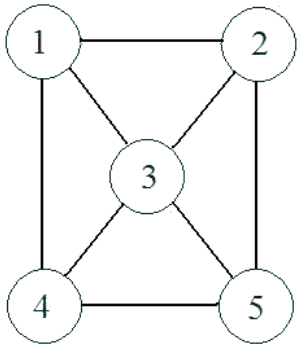

Figura 3: Grafo de cinco vértices e oito arestas

O PCV pode ser tratado como um problema no qual se deseja obter um ciclo Hamiltoniano máximo. Pode-se então definir o problema como o de um caixeiro viajante que deseja visitar N cidades (vértices) de certa localização e que, entre alguns pares de cidades existem rotas (arcos ou arestas), através das quais ele pode viajar a partir de uma cidade para outra. Cada rota contém um número associado que pode representar a distância ou o custo necessário para percorrê-la. Assim, o caixeiro viajante deseja encontrar um caminho que passe por cada uma das N cidades apenas uma vez. Além disso, que tenha um custo menor que certo valor onde o custo do caminho é a soma dos custos das rotas percorridas. Note que a existência de tal caminho nem sempre é possível [MIL91].

Utilizando a notação de grafos, então podem-se definir o problema como sendo $G = (V, E)$ um grafo (direto ou indireto), e conjunto F uma família de todos dos ciclos Hamiltonianos (*tours*) de G. Para cada aresta $a \in A$ existe um custo c_a associado. O problema do caixeiro viajante é encontrar um *tour* (ciclo Hamiltoniano) em G, onde a soma dos custos das arestas (também chamado de instâncias) seja o menor possível.

2.4.4 Métodos de Resolução

Existem dois métodos distintos de se resolver um sistema como o PCV:

- Métodos exatos: São os métodos que tentam alcançar sempre o resultado ótimo, por isso normalmente precisam de um maior esforço computacional para que se tenha o resultado em um tempo satisfatório;
- Métodos aproximados: Também conhecidos como métodos heurísticos, são métodos que garantem uma solução aproximada demandando um tempo curto. Pode ocorrer que se encontre a solução ótima nestes métodos, mas não se pode garantir que isto sempre ocorra.

2.4.4.1 Métodos Exatos

Os métodos exatos compreendem alguns algoritmos como: *branch-and-bound*, planos de corte ou combinações dos dois, tais como *branch-and-cut* e programação dinâmica. Os mais conhecidos e utilizados algoritmos são do tipo *branch-and-bound* e são definidos a partir de uma regra de alocação e de uma regra de corte que, em geral, é ditada por um limite inferior do problema.

Como o avanço tecnológico permite uma resolução mais rápida do problema, estão sendo utilizados procedimentos nos quais se combinam algumas técnicas para se alcançar o objetivo final de maneira rápida.

A programação dinâmica é uma abordagem para a construção de algoritmos para a resolução de problemas computacionais, em especial os de otimização combinatória. Ela é aplicável a problemas nos quais a solução ótima pode ser calculada a partir da solução ótima previamente calculada e memorizada - de forma a evitar recálculo - de outros subproblemas que, sobrepostos, compõem o problema original.

Os métodos de plano de corte foram desenvolvidos para resolver problemas de otimização convexa. Esses métodos geram uma seqüência de problemas de programação linear que refinam as aproximações poliedrais do problema original. As desigualdades lineares que definem as aproximações são geradas por um oráculo como hiperplanos separando um ponto teste do conjunto solução. A questão crucial no desenvolvimento de algoritmos de plano de corte é a escolha do ponto teste na aproximação poliedral.

A técnica *branch-and-cut*, cuja denominação foi proposta originalmente por Padberg e Rinaldi [PAD91], surge como uma estratégia alternativa às anteriores e explora o politopo definido pelas soluções viáveis do problema considerado. A vantagem dos algoritmos *branch-*

and-cut sobre os de planos de corte é o uso de cortes que são válidos para o politopo formado por todas as soluções viáveis, definindo facetas. Tais cortes associados às facetas são mais significativos que os produzidos pelo método de planos de corte, pois a solução encontrada converge para uma solução ótima de maneira mais satisfatória.

Porém, mesmo com o atual avanço tecnológico, o esforço computacional necessário para se resolver problemas com dezenas de milhares de cidades torna impraticável apenas o uso de métodos exatos, para que se obtenha um resultado de maneira satisfatória em relação ao tempo necessário.

2.4.4.2 Métodos Heurísticos

Quando não se necessita achar uma solução ótima para o sistema, muitas vezes, por causa do tempo necessário para a mesma, utilizam-se então métodos heurísticos, que conseguem dar uma resposta próxima ao resultado ótimo. Os métodos aproximativos podem se enquadrar nesta categoria, acrescentando-se que, para estes casos, são conhecidas propriedades com garantia do pior caso. Além disso, conforme Osman e Laporte [OSM96], é comum, na literatura de Otimização Combinatória, os algoritmos aproximativos serem tratados como algoritmos heurísticos. As heurísticas para o PCV abrangem os seguintes tipos: métodos construtivos, métodos de enumeração limitada e métodos de melhoria. Ao final desta seção, destacam-se os algoritmos aproximativos com propriedades matemáticas para garantir o pior caso, mas que dado à dificuldade do PCV, são aplicados apenas a instâncias particulares.

Os métodos construtivos baseiam-se na construção de uma solução viável para o problema partindo de uma solução trivial, e sobre a qual são aplicadas diferentes técnicas de modo a melhorar a qualidade da solução a ser obtida [ROM04]. Considere os conjuntos A e L, o primeiro, de objetos alocados e o segundo, de localizações ocupadas, ambos inicialmente vazios. Em tais métodos a construção de uma permutação π é feita de forma heurística, escolhendo-se a cada passo, a próxima alocação (i, j), tal que $i \notin A$, $j \notin L$, e fazendo-se $\pi(i) = j$. Para um problema de ordem *n,* o processo é repetido até completar uma permutação na ordem do problema [ROS77].

Estes métodos permitem que ao final do processo enumerativo, se chegue a uma solução ótima, fazendo com que se obtenha os mesmos resultados de uma solução ótima. A diferença é que, muitas vezes, logo nas primeiras iterações do programa, já se obtem um

resultado aceitável, o qual apenas será mais refinado a ponto de se obter o resultado ótimo. Observa-se que, quanto melhores são as informações utilizadas para guiar a enumeração, maiores serão as chances de encontrar prematuramente soluções com boa qualidade. Porém, como explicado anteriormente nos métodos exatos, caso se queira fazer o ciclo completo do algoritmo a ponto de se obter o resultado ótimo, custará um tempo operacional grande. Por isso, são impostas condições de parada para limitar esta enumeração, tais como: um número máximo de iterações realizadas pelo algoritmo, finalizar o algoritmo se após um número pré-determinado de passos não ocorrer nenhuma melhoria da função custo; tempo máximo de execução, entre outras. Porém, caso o programa pare por causa de qualquer um destes critérios de parada, ocasionará a eliminação da solução ótima [RIB96], [REE96].

Os métodos de melhoria compreendem os algoritmos de busca local, que será estudado mais adiante. A maioria das heurísticas aplicada ao PCV faz parte desta categoria. Um método de melhoria inicia-se com uma solução viável e tenta melhorá-la, procurando outras soluções em sua vizinhança. O processo é repetido até que nenhuma melhoria possa ser encontrada. Os métodos nesta categoria são comumente utilizados pelas metaheurísticas.

Capítulo 3

Abordagens para resolução do PCV

Neste capítulo são apresentados os fundamentos das metaheurísticas. Além disso, as técnicas de Evolução Diferencial (ED) e sistemas híbridos combinados entre a ED com LK e também a ED com VNS e LKH propostas, neste trabalho, para resolução do Problema do Caixeiro Viajante são também apresentadas.

3.1 Metaheurísticas

Até o final da década de 80, os métodos heurísticos propostos para resolver problemas de otimização combinatória eram, em sua maioria, específicos e dedicados a um dado problema. A partir daí, esse paradigma evolui e surge um crescente interesse em técnicas que sejam mais gerais e flexíveis e, por isso, aplicáveis a diversos problemas. Estas técnicas são conhecidas por metaheurísticas. Dentre as metaheurísticas destacam-se: *simulated annealing*, busca tabu, algoritmos genéticos, *scatter search*, GRASP (*Greedy Randomized Adaptive Search Procedure*), colônia de formigas, busca em vizinhança variável, entre outras.

3.1.1 PCV, Heurísticas e Metaheurísticas

As heurísticas são procedimentos de solução que, muitas vezes, se apóiam em uma abordagem intuitiva, na qual a estrutura particular do problema possa ser considerada e explorada de forma inteligente, para a obtenção de uma solução adequada [CUN97].

Assim, na maioria dos casos, as heurísticas propostas tendem a ser específicas e particulares para um determinado problema, carecendo de robustez, isto é, não conseguem

produzir boas soluções para problemas com características, condicionantes ou restrições pouco diferentes daquelas para as quais foram desenvolvidas.

Os procedimentos heurísticos para o PCV podem ser divididos em dois grupos: métodos de construção de roteiros e métodos de melhorias de roteiros.

3.1.1.1 Heurísticas de Construção de Roteiros

Nas heurísticas de construção de roteiros, nós (cidades) são incluídos no roteiro gradualmente, de modo seqüencial, seguindo alguma regra de construção, sem que a solução parcial obtida seja melhorada. Em outras palavras, um roteiro é construído iterativamente, sem modificação posterior das seqüências parciais de cidades, definidas ao longo do processamento do algoritmo. Segundo [LAP92], a construção do roteiro pode se dar através:

- Do método do vizinho mais próximo, no qual o caixeiro inicia em uma cidade qualquer e então segue para a cidade mais próxima; a partir desta, busca-se a cidade mais próxima ainda não visitada, e assim sucessivamente, até que todas as cidades sejam visitadas, retornando então à cidade de origem. Este método também conhecido como método GREEDY ou como método guloso;
- De métodos de inserção, nos quais, partindo-se de um roteiro inicial com apenas duas cidades, e considerando as demais cidades ainda não incluídas no roteiro, seleciona-se para inserção, no roteiro parcial sendo construído, aquela que atenda um determinado critério. Por exemplo, a cidade que proporciona o menor acréscimo de distância total percorrida, ou a cidade mais próxima do roteiro, ou a mais distante, ou ainda aquela que forma o maior ângulo com duas cidades já inseridas no roteiro. Esse procedimento é repetido sucessivamente, com a análise da inserção entre cada par de cidades do roteiro parcial , até que todas as cidades sejam inseridas no roteiro.
- A heurística das economias de Clarke e Wright (CW), bastante conhecida e ainda muito utilizada como parte de outros procedimentos, foi originalmente desenvolvida para resolver o problema clássico de roteamento de veículos. Baseia-se na noção de economias, que pode ser definido como o custo da combinação, ou união, de duas subrotas existentes. Trata-se de uma heurística iterativa de construção baseada numa função gulosa de inserção

Esses procedimentos são apresentados em detalhes por [NOV01] e também ilustrados através de exemplos práticos.

3.1.1.2 Heurísticas de Melhoria de Roteiros

Genericamente, essas heurísticas procuram melhorar o roteiro do caixeiro viajante obtido através de algum outro modo. Os métodos de melhorias mais utilizados são do tipo *k-optima* ou *k-opt*, conforme proposto por [LIN73], nos quais *k* arcos são removidos de um roteiro e substituídos por outros *k* arcos, com a finalidade de diminuir a distância total percorrida. Conforme [LAP92], quanto maior o valor de *k*, melhor a precisão do método, porém maior também é o esforço computacional. Na prática, são considerados os métodos 2-*opt* e 3-*opt*, isto é, *k* pode assumir valores 2 ou 3, conforme ilustrado nas figuras 4 e 5, respectivamente. De forma resumida, o método 2-*opt* testa trocas possíveis entre pares de arcos, refazendo as conexões quando houver uma melhoria no roteiro. O processo termina quando não for possível mais realizar nenhuma troca que resulte em melhoria. Este procedimento possui ordem de complexidade $O(n^2)$. Como pode ocorrer inversão de sentido em parte do roteiro, conforme mostrado na Figura 4, pressupõe-se a simetria de distâncias.

No caso do 3-*opt* são considerados três arcos ao invés de dois, a fim de se avaliarem as alterações nas conexões entre os nós, o que resulta em sete possíveis combinações, conforme pode ser observado na Figura 5. Destas, apenas quatro combinações (4, 5, 6 e 7) representam trocas entre três arcos; as combinações 1, 2 e 3 correspondem a trocas do tipo 2-*opt*.

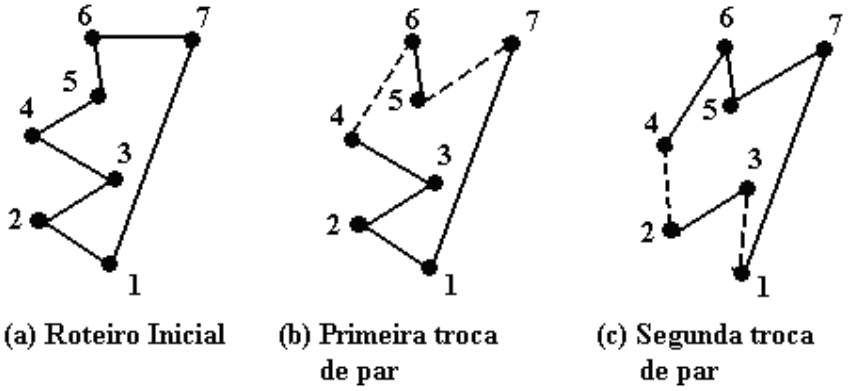

Figura 4: Representação de troca de pares de arcos (2-opt).

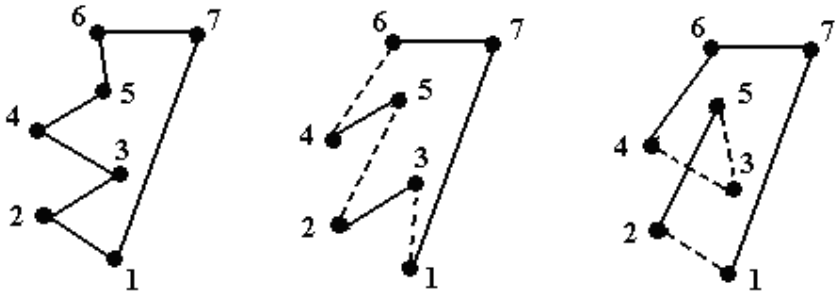

Figura 5: Representação de troca de trios de arcos (3-*opt*).

3.1.2 Busca Local

A busca local, também referida na literatura como busca na vizinhança, são procedimentos utilizados para melhorar uma solução viável. Obtida, por exemplo, através de uma heurística construtiva. Estes métodos são mais genéricos e podem auxiliar uma ampla gama de problemas sem a necessidade de uma compreensão mais profunda do mesmo [JOH90], [JOH97].

Considere o par (S, g), onde S é um conjunto de soluções viáveis para um problema de otimização e g é a função objetivo que associa cada elemento $s \in S$ a um número real. Em um problema de minimização de uma função objetivo é determinar um elemento $s^* \in S$, tal que $g(s^*) \leq g(s) \forall s \in S$.

Conforme [MAR00], no caso do PCV, por exemplo, para uma solução corrente viável (uma rota válida) a vizinhança pode ser definida como: "todas as rotas que podem ser geradas pela substituição de dois arcos presentes na rota corrente por outros dois arcos que não estão nesta rota e que gerem também uma rota viável".

3.1.3 *Simulated Annealing*

É um algoritmo de busca local que explora a analogia entre os problemas de Otimização Combinatória e os da Mecânica Estatística [KIR83]. Tal analogia é realizada associando-se as soluções viáveis dos problemas de Otimização Combinatória a estados dos sistemas físicos tal que seus custos são associados à energia desses estados. Considerando-se dois estados sucessivos de energia E_i e E_{i+1}, correspondendo a duas soluções vizinhas e tomando-se $\Delta E = E_{i+1} - E_i$, as seguintes situações podem ocorrer:

- se $\Delta E < 0$, há redução de energia e o processo continua, ou seja, há redução no valor da função objetivo do problema e a nova alocação deve ser aceita;
- se $\Delta E = 0$, há situação de estabilidade e portanto, não há alteração de energia, isto é, a função objetivo do problema permanece inalterada;
- se $\Delta E > 0$, fica caracterizado um aumento de energia, útil no processo físico para permitir uma futura acomodação das partículas, ou seja, a função custo do problema sofre aumento. Ao invés desta alocação ser eliminada, esta poderá eventualmente ser aproveitada. Para isso, uma função de probabilidade deverá ser acionada para evitar a convergência para soluções que caracterizam-se em mínimos locais indesejáveis.

Um pseudocódigo é mostrado na Quadro 1.

Quadro 1: Pseudocódigo do SA

Início

$s \leftarrow s_0$; {Solução corrente}
$s' \leftarrow s$; {Melhor solução obtida até então}
$T \leftarrow T_0$;{Temperatura corrente}
$IterT \leftarrow 0$; {Número de iterações na temperatura T}
$Samax \leftarrow 0$; {Número máximo de iterações de T }
$s^* \leftarrow s'$; {Melhor solução obtida em todo o processo}
enquanto ($T > 0$) faça
    enquanto ($IterT < SAmax$) faça
        $IterT \leftarrow IterT + 1$;
        Gere um vizinho qualquer $s' \in N(s)$;
        $\Delta = f(s') - f(s)$;
        se ($\Delta > 0$) então
            $s \leftarrow s'$;
        fim-se
        se $f(s') > f(s^*)$ então
            $s^* \leftarrow s'$;
        senão
            Gere um valor aleatório $x \in [0,1]$;
            se $x < e^{\Delta/T}$ então

$s \leftarrow s'$;

fim-se

fim-se;

fim-enquanto;

$T \leftarrow \alpha$ x T;

$IterT \leftarrow 0$;

fim-enquanto;

Retorne s*;

Fim

3.1.4 Busca Tabu

É um algoritmo de busca local introduzido por [GLO89-a] e [GLO89-b] para problemas de Programação Inteira, com o objetivo de determinar soluções de boa qualidade. Caracteriza-se pela existência de uma lista atualizada das melhores soluções encontradas no decorrer da execução do algoritmo, onde cada solução possui um valor de prioridade ou critério de aspiração. Suas características básicas são: (i) Uma lista tabu, para armazenar um histórico da evolução do processo de busca; (ii) Um mecanismo que permite aceitar ou rejeitar uma nova alocação na vizinhança, baseado nas informações armazenadas na lista tabu e suas respectivas prioridades; e (iii) Um mecanismo que permite alternar entre estratégias de diversificação e intensificação na vizinhança.

Um pseudocódigo é mostrado no Quadro 2.

Quadro 2: Pseudocódigo da Busca Tabu

Início

$Iter \leftarrow 0$;{Número de iterações}

$MelhorIter \leftarrow 0$;{Salva melhor iteração}

$s \leftarrow s_0$; {Solução corrente}

$s' \leftarrow s$; {Melhor solução obtida até então}

$s^* \leftarrow s'$; {Melhor solução obtida em todo o processo}

enquanto (Critério de parada não satisfeito) faça

$Iter \leftarrow Iter + 1$;

Atualize a lista tabu;

```
            s ← s';
            se (f(s) < f(s*)) então
                  s* ← s;
                  MelhorIter ← Iter;
            fim-se;
      fim-enquanto;
      s ← s*;
      retorne s;
Fim
```

3.1.5 Algoritmos Genéticos

São técnicas que se apóiam nos mecanismos de seleção e adaptação natural das espécies. Através de operadores genéticos e de critérios de seleção, a cada iteração substitui-se uma população de indivíduos por outra, com valores de aptidão, em média, melhores. A idéia básica consiste em acreditar que os melhores indivíduos sobrevivem e geram descendentes com suas características hereditárias, de maneira semelhante à teoria biológica das espécies [DAV87].

Conforme explicado em [RAM05], os algoritmos genéticos definem um processo evolucionário desdobrado em três níveis, nos quais as informações são armazenadas e gerenciadas. A população de cromossomos está no primeiro nível e representa a memória corrente do processo de busca. O segundo nível é composto por uma população denominada de vetores genéticos, cuja natureza difere daquela dos cromossomos. Essa população é provida de ferramentas capazes de promover a diversificação e intensificação da busca no espaço de soluções. O terceiro nível é composto por regras que comandam a interação entre essas duas populações.

No primeiro nível, a população de cromossomos tem estrutura similar à dos algoritmos genéticos. No entanto, sua forma de evolução é diferenciada baseando-se na troca de informações entre as populações citadas e não na troca de informações dentro da população de cromossomos, peculiar dos algoritmos genéticos.

A estrutura dos vetores genéticos (do segundo nível) e sua atuação na população de cromossomos, assim como as regras (do terceiro nível) que comandam a interação entre as referidas populações estão detalhadas a seguir.

Um pseudocódigo é mostrado no Quadro 3.

Quadro 3: Pseudocódigo do AG

Início

inicialize P_a ; { população inicial }

avalie P_a ; { adaptação proporcional }

para todo $s_k \in P_a$

calcule $\delta(s_k)$ { cálculo do rank }

fim_para

enquanto (Critério de parada não satisfeito) faça

para todo $s_k \in P_a$ satisfazendo $a < \delta(s_k)$ faça { teste de evolução }

selecione P_a de $P_{a\text{-}\varepsilon}$; { operador de reprodução }

recombine P_a ; { operadores de recombinação }

avalie P_a ; { adaptação proporcional }

fim-para

para todo novo $s_k \in P_a$

calcule $\delta(s_k)$ { cálculo do rank }

fim-para

fim-enquanto

Fim

3.1.6 *Scatter Search*

Foi introduzida por [GLO77] em um estudo heurístico de problemas de Programação Linear Inteira. É um método evolutivo que considera combinações lineares de vetores-solução para produzir novos vetores-solução através de gerações sucessivas. Esta metaheurística é composta por uma fase inicial e outra evolutiva. A primeira se constitui em um conjunto de soluções, das quais as melhores são escolhidas para fazerem parte de um conjunto referência. Na fase evolutiva, novas soluções são geradas utilizando combinações de subconjuntos referência que são selecionados estrategicamente. A partir daí, um conjunto das melhores soluções geradas é incluído no conjunto referência.

Um pseudocódigo é mostrado no Quadro 4.

Quadro 4: Pseudocódigo do SS

```
Início
    TamanhoP ← Tamanho da População P;
    CriePopulação(P, TamanhoP);
    i ← 0;
    enquanto i < MaxIterações faça
        i ← i + 1
        x ← CriaNovaGeração();
        y ← CombinResultado(x, P);
        P0 ← IncrementaSolução(y);
        Se P0 < P faça
            P ← P0
        Fim-se
    fim-enquanto
Fim
```

3.1.7 GRASP (*Greedy Randomized Adaptive Search Procedures*)

É uma técnica iterativa baseada em transições aleatórias onde, a cada iteração, uma solução aproximada para o problema é obtida. A melhor solução resultante das iterações é a solução final. Em cada iteração, a primeira solução é construída através de uma função gulosa aleatória e as soluções posteriores são obtidas aplicando-se, sobre a solução anterior, um algoritmo de busca local que forneça uma nova solução melhor que a anterior. Ou seja, cada iteração é composta por duas fases, uma de construção e outra de busca local. Ao final de todas as iterações, a solução resultante é a melhor solução gerada. Nada garante que soluções geradas por uma construção GRASP não recaiam em ótimos locais. Assim, é importante aplicar a fase de busca local na tentativa de melhorar tais soluções. Com o uso de estruturas de dados personalizadas e uma implementação cuidadosa, uma fase construtiva eficiente pode gerar boas soluções iniciais, possibilitando uma busca local eficiente [CHR72], [CHR75].

Um pseudocódigo é mostrado no Quadro 5.

Quadro 5: Pseudocódigo de GRASP

```
Início
    F(s) ← ∞;
    para k=1 até iterações faça
        x ← GulosoAleatorio(F(s));
        x ← BuscaLocal(x);
        Se f(s') < f(s) então
            s ← s';
        fim-se
    fim-para
    return s;
Fim
```

3.1.8 Colônia de Formigas

Conforme [WAG03], o algoritmo de colônia de formigas consiste na simulação do comportamento de um conjunto de agentes que cooperam entre si para resolver um problema de otimização por meio de procedimentos de comunicação simples.

As formigas são capazes de encontrar seu caminho (do ninho para uma fonte de comida e o caminho de volta ou vencendo um obstáculo) com relativa facilidade. Estudos na área de entomologia apresentam resultados que levam a crer que, em muitos casos, essa capacidade é o resultado da interação via comunicação química entre as formigas (mediante uma substância chamada feromônio) e um fenômeno causado pela presença de um grupo de formigas [RIB96], [DOR96] e [DOR97].

A principal característica do método é que a interação desses agentes gera um efeito sinergístico, pois a qualidade da solução obtida aumenta quando tais agentes trabalham juntos, interagindo entre si, para a resolução de um mesmo problema.

Um pseudocódigo é mostrado no Quadro 6.

Quadro 6: Pseudocódigo da colônia de formigas

```
Início
    i ← 0;
    P ← 0; {Solução corrente}
```

```
        P0 ← 0; {Melhor solução obtida até então}
        P* ← P; {Melhor solução obtida em todo o processo}
        enquanto i < iterações faça
                i ← i +1
                P ← CriarFormigas();
                enquanto j < tamanhoP faça
                        j ← j +1
                        para P(j) faça
                                P0 ← ConstruirSolução(P(j));
                                P(j) ← AtualizarFeromônio(P0);
                        fim-para;
                fim-enquanto
                P(j) ← EscalonarFormigas(P(j));
                P* ← AtualizaFeromônio(P(j)),
                se P* < P faça
                        P ← P*
                Fim-se
        Fim-enquanto
Fim
```

3.1.9 Busca em Vizinhança Variável

A busca em vizinhança variável, (VNS, *Variable Neighbourhood Search*), foi introduzida por [MLA95] e [MLA97]. É baseada na mudança sistemática das vizinhanças utilizadas na busca e vem sendo aplicada na resolução de problemas combinatórios de maior porte. Contrariamente a outras metaheurísticas baseadas em métodos de busca local, o método VNS não segue uma trajetória, mas sim explora vizinhanças gradativamente mais "distantes" da solução corrente e focaliza a busca em torno de uma nova solução se, e somente se, um movimento de melhora é realizado. O método inclui, também, um procedimento de busca local a ser aplicado sobre a solução corrente. Esta rotina de busca local também pode usar diferentes estruturas de vizinhança.

Em [HO03], pode-se verificar um problema de redes de alta velocidade, conhecidas como 4G, que podem ser otimizadas através do método de busca na vizinhança.

Um pseudocódigo é mostrado no Quadro 7.

Quadro 7: Pseudocódigo da VNS

```
Início
    kmax = número de vizinhanças;
    solução inicial s;
    enquanto (critério de parada não satisfeito) faça
        k ← 1;
        enquanto (k ≤ kmax) faça
            Gere um vizinho s'qualquer na vizinhança Nk(s);
            Aplique um método de busca local em s' obtendo um ótimo local (s'');
            se (s'' for melhor que s) faça
                s ← s'';
                k ← 1;
            senão
                k ← k + 1;
            fim-se
        fim-enquanto
    fim-enquanto
    retornar s;
Fim
```

3.1.10 Busca por Agrupamento ou *Clustering Search* (CS)

Conhecida por CS foi introduzida inicialmente por [OLI04] e aprimorada por [LOR07]. Essa metaheurística consiste inicialmente num processo de detecção para se tentar localizar regiões supostamente promissoras em um espaço de busca. Este método tem potencial para determinar um resultado de boa qualidade, uma vez que conforme as regiões sejam descobertas, podem-se mudar as estratégias de busca sobre elas. As regiões encontradas podem então ser consideradas sub-espaços de buscas definidos uma relação de vizinhança.

Conforme [CHA05], pode-se dizer que no CS um processo de agrupamento iterativo é executado simultaneamente com uma metaheurística, identificando grupos de soluções que merecem especial interesse. As regiões destes grupos de soluções devem ser exploradas tão logo sejam detectadas, através de heurísticas de busca local específicas. Espera-se uma

melhoria no processo de convergência associado a uma diminuição no esforço computacional em virtude do emprego adequado dos métodos de busca local.

O ECS procura localizar regiões promissoras através do enquadramento destas por *clusters*. Um *cluster* é definido pela tripla $G = \{c, r, \beta\}$, onde c e r são, respectivamente, o centro e o raio de uma área de busca promissora. Existe também uma estratégia de busca β associada ao *cluster*.

O centro é um indivíduo (ou solução) representante do *cluster*, que identifica a sua localização dentro do espaço de busca. O raio estabelece a distância máxima, a partir do centro, até a qual um indivíduo pode ser associado ao *cluster*.

Em um problema de otimização combinatória, o raio pode ser definido em termos de número de movimentos necessários para transformar uma solução candidata em outra dentro de uma vizinhança definida por uma heurística qualquer.

O CS consiste em quatro componentes conceitualmente independentes com diferentes atribuições:

- Uma metaheurística (MH);
- Um agrupador iterativo (AI);
- Um analisador de agrupamentos (AA);
- Um algoritmo de otimização local (AO).

Um pseudocódigo é mostrado no Quadro 8.

Quadro 8: Pseudocódigo de CS

Início

para (número de iterações não satisfeito) faça

s = Ø

enquanto (solução não construída) faça

produzir Lista de Candidatos (C_e)

LCR = C_e * α

e = selecionar elemento aleatoriamente (LCR)

$s = s \cup \{e\}$

atualizar lista de Candidatos (C)

fim enquanto

k_{max} = número de vizinhanças

```
        enquanto (critério de parada não satisfeito ) faça
            k ← 1
            enquanto ( k ≤ k_max )
                gere um vizinho s′ qualquer na vizinhança N^k (s)
                s″ = Aplicar VND em s′
                aplique o componente AI (s″)
                aplique o componente AA (cluster ativo)
                se (cluster ativo for promissor) faça
                    aplicar o componente AO
                se (s″ for melhor que s) faça
                    s ← s″
                    k ← 1
                senão
                    k ← k + 1
            fim enquanto
        fim enquanto
    fim para
Fim
```

3.1.11 Métodos Híbridos

Os sistemas híbridos podem ocorrer da união entre sistemas existentes para a configuração de um sistema que necessite de um esforço computacional menor ou que se consiga obter uma resposta em um menor espaço de tempo.

Muitos autores já fizeram pesquisas recentes nesta área, a citar por exemplo:

- Colônia de formigas com busca em vizinhança variável [TSA04];
- 2-opt com algoritmos genéticos [TAN00];
- Algoritmos genéticos com técnicas de *branch-and-bound* [COT95];
- Busca Local com algoritmos genéticos proposto por [NGU07];
- 2-*opt* e 3-*opt* com algoritmos genéticos abordado em [BAR01].

3.1.12 Algoritmo Lin-Kernighan

O algoritmo LK (*Lin-Kernighan*) utiliza como base o algoritmo *r-opt move* que consiste em trocar *r* ligações por outras *r* ligações de forma que o novo trajeto seja mais curto, e continua fazendo trocas até que não seja possível melhorar o trajeto.

Quando não é possível reduzir o custo do trajeto fazendo a mudança de *r* ligações, diz-se que o trajeto é *r-ótimo*. Desta definição, se um trajeto é *r-ótimo*, ele é também *r'-ótimo* desde que $1 \leq r' \leq r$. Tem-se ainda que um trajeto é ótimo se e somente se ele é *n-ótimo* (*n* é o número de cidades).

Quanto maior o valor de *r*, mais curto será o trajeto, porém o tempo de execução cresce muito rapidamente com o valor de *r* (da ordem de n^r). Para obter a melhor relação entre o tempo de execução e a qualidade do resultado, Lin e Kernighan desenvolveram um algoritmo que realiza testes para decidir qual o valor de *r* a ser utilizado a cada etapa.

Conforme [LIN73], o modelo de Lin-Kernighan pode ser apresentado da seguinte forma: Dado um trajeto inicial *T*, a cada iteração o algoritmo encontra dois conjuntos de ligações:

$X=\{x_1, x_2, \ldots, x_r\}$ e $Y=\{y_1, y_2, \ldots, y_r\}$ onde:

- *X* deve pertencer a *T*;
- *X* e *Y* são disjuntos. Esta regra reduz o tempo de execução do algoritmo.
- x_i e y_i devem compartilhar um ponto. Seja $x_i=(t_{2i-1}, t_{2i})$, então $y_i=(t_{2i}, t_{2i+1})$, onde t_i é um ponto qualquer.

Uma condição necessária, mas não suficiente é que as ligações $\{x_1, y_1, x_2, y_2, \ldots, x_r, y_r\}$ devem formar uma cadeia fechada de ligações, ou seja, $y_r=(t_{2r}, t_1)$.

- A escolha de $x_i=(t_{2i-1}, t_{2i})$ deve ser feita de modo que se t_{2i} está ligado a t_1, o resultado seja um trajeto. Esta regra é usada para $i \leq 3$ e garante que é possível o trajeto.

i) Seja $g_i = c(x_i) - c(y_i)$, onde $c(x_i)$ é o custo da ligação x_i; y_i deve ser escolhido de forma que $G_i = g_1+g_2+\ldots+g_i$ seja positivo. Esta regra aumenta a eficiência do algoritmo.

O algoritmo original é mostrado na Quadro 9:

Quadro 9: Pseudocódigo de LK

1. Gerar um trajeto aleatório T.
2. Seja $i = 1$, escolher t_1.
3. Escolher $x = (t_1 , t_2) \in T$.
4. Escolher $y_1 = (t_2 , t_3) \notin T$ tal que $G_1 > 0$. Se não for possível, ir para o passo 12.
5. $i = i + 1$.
6. Escolher $x_i = (t_{2i-1}, t_{2i}) \in T$ tal que:
   a) $x_i \neq y_s$ para todo s < i.
   b) Se t_{2i} é ligado a t_1, a configuração resultante é um trajeto T', e se este é um trajeto melhor que T ir para o passo 2.
7. Escolher $y_i = (t_{2i} , t_{2i+1}) \in T$ tal que:
   a) $G_i > 0$.
   b) $y_i \neq x_s$ para todo $s \leq i$.
   c) x_{i+1} deve existir.
8. Se houver uma alternativa (ligação) não executada para y_2: $i = 2$ e ir para o passo 7.
9. Se houver uma alternativa (ligação) não executada para x_2: $i = 2$ e ir para o passo 6.
10. Se houver uma alternativa (ligação) não executada para y_1: $i = 1$ e ir para o passo 4.
11. Se houver uma alternativa (ligação) não executada para x_1: $i = 1$ e ir para o passo 3.
12. Se houver uma alternativa (ponto) não executada para t_1: ir para o passo 2.
13. Parar (ou ir para o passo 1).

Para aumentar a eficiência do algoritmo, algumas regras foram impostas para limitar a escolha das ligações:

1) Somente trocas seqüenciais são permitidas.
2) O ganho parcial G_i deve ser positivo.
3) O trajeto deve ser fechado (com exceção de $i = 2$).
4) Uma ligação previamente excluída não pode ser adicionada, e uma ligação previamente adicionada não pode ser excluída.
5) A procura pela solução pára se o trajeto encontrado é igual ao encontrado anteriormente.
6) A procura por uma ligação a ser adicionada, $y_i = (t_{2i}, t_{2i+1})$, é limitada aos cinco vizinhos mais próximos do ponto atual (x_i).
7) E ainda algumas regras para direcionar a procura:

a) Na escolha da ligação y_i (com $i \geq 2$) a cada ligação é dada a prioridade $c(x_{i+1}) - c(y_i)$

b) Se há duas alternativas para x_4, aquela onde $c(x_4)$ é maior é escolhida.

A regra 6 direciona o algoritmo para trajetos menores e reduz significativamente o tempo de execução, porém se a solução ótima contém uma ligação onde seus pontos não estão entre os cinco vizinhos mais próximos, o algoritmo terá dificuldades de encontrar a solução ótima. Por exemplo, para um problema com 532 nós há uma ligação onde o próximo ponto é o 22º vizinho mais próximo. No entanto, aumentar o número de candidatos a fazer a ligação para 22 irá aumentar muito o tempo de execução. Para resolver este problema, Lin e Kernighan [LIN77] criaram um critério de proximidade, denominado α-*nearness*, a partir do qual são selecionados os candidatos a fazer parte da próxima ligação.

3.1.13 Lin-Kernighan Helsgaun

Conforme [HEL98], Keld Helsgaun fez várias mudanças significativas em relação ao LK original, principalmente na parte referente à busca de um novo caminho. Enquanto que no LK original, usa-se pequenas buscas locais, o LKH, assim chamado o sistema de Keld Helsgaun, usa buscas de longas distâncias. Outro importante ponto a ser notado é que LKH usa um método mais complexo em seus passos para se chegar no resultado ótimo. Ainda conforme [HEL98], o LKH tem um tempo de execução que seria $n^{2,2}$ vezes mais rápido que LK, onde n é a quantidade de cidades. E ainda LKH consegue obter um melhor resultado neste mesmo período.

Helsgaun, ao verificar o método proposto por Lin e Kernighan, notou que existiam várias restrições que criavam barreiras para se conseguir obter o caso ótimo. Um exemplo seria o caso deles apenas verificarem os vizinhos de até 5 níveis para fazer uma troca. Com isso, caso o vizinho correto para se fazer uma troca fosse o 22º nível, ter-se-ia que trocar sucessivamente 22 vezes até se conseguir tal resultado. Porém, neste mesmo exemplo, podería-se ter uma troca em que não se conseguiria chegar ao melhor resultado. No LKH, pode-se ter até 20 níveis de troca, o que permite um intervalo de trocas maior, porém continua-se a ter o problema de obter um caminho no qual não se permite mais chegar a solução ótima.

3.1.14 Evolução Diferencial

A Evolução Diferencial (ED) foi criada após Ken Price tentar usá-la para resolver o Problema do Polinômio de Chebychev no qual foi apresentado a ele por Rainer Storn. A descoberta aconteceu quando Ken teve a idéia de utilizar diferentes vetores para recriar o vetor população. Desde então foram feitos vários testes e aperfeiçoamentos substancialmente importantes no qual tornaram o algoritmo ED versátil e robusto [STO95].

Conforme [ARA06], o conceito do ED é simples. O algoritmo é iniciado criando uma população inicial escolhida aleatoriamente com distribuição uniforme (soluções potenciais) atendendo ao limite do espaço de busca.

A idéia da evolução diferencial é gerar novos indivíduos, denotados vetores modificados ou doadores, pela adição da diferença ponderada entre dois indivíduos aleatórios da população a um terceiro indivíduo. Esta operação é denominada mutação.

As componentes do indivíduo doador são misturadas com as componentes de um indivíduo escolhido aleatoriamente (denotado vetor alvo), para resultar o chamado vetor tentativa, ou vetor experimental. A operação de combinar os parâmetros é referida freqüentemente como "cruzamento" na comunidade dos algoritmos evolutivos.

Se o vetor experimental resultar um valor da função objetivo menor que o vetor alvo, então o vetor experimental substitui o vetor alvo na geração seguinte. Esta última operação é chamada seleção. O procedimento é finalizado através de algum critério de parada.

Uma possível razão para que o ED trabalhe bem é que a mutação é dirigida pela diferença entre os valores dos parâmetros dos membros atuais da população. Isto permite que cada parâmetro faça uma troca automática e de uma apropriada redução no processo de otimização, ajudando-a a convergir para a solução apropriada. Um pseudocódigo do algoritmo do ED é apresentado no Quadro10.

Quadro 10: Pseudocódigo da ED

```
Início
    Inicie a população P
    para k=1 até iterações faça
        equanto i < ps faça
            i ← i+1
            Aleatoriamente selecione os pais.
```

Crie candidatos iniciais.
Crie candidato final *C* cruzando os genes dos candidatos Pais e Iniciais.
Resolva *C*[*i*]
se (*C*[*i*] for melhor que *P*[*i*]) faça
P0[*i*] ← *C*[*i*]
senão
P0[*i*] ← *P*[*i*]
fim-se
P ← *P0*
fim-enquanto
Fim

3.1.14.1 Operadores da Evolução Diferencial

Seguindo a idéia da evolução diferencial, o algoritmo tem operadores que se baseiam nesta idéia para que se consiga manter a diversidade e as convergências prematuras, no qual acabe ocorrendo um retrocesso na busca da solução ótima. Tais fatores são as seguintes:

Operador de Mutação: Sejam os vetores X_α, X_β e X_γ escolhidos aleatoriamente e distintos entre si. Na geração q um par de vetores (X_β, X_γ) define uma diferença $X_\beta - X_\gamma$. Esta diferença é multiplicada por $F > 0$, sendo denotada por diferença ponderada, e é usada para perturbar o terceiro vetor X_α ou o melhor vetor, denominado de X_{best} da população. Este processo resulta o vetor doador $V^{(q+1)}$ que pode ser escrito matematicamente como

$$V^{(q+1)} = X_\alpha^{(q)} + F\left(X_\beta^{(q)} - X_\gamma^{(q)}\right) \text{ ou } V^{(q+1)} = X_{best}^{(q)} + F\left(X_\beta^{(q)} - X_\gamma^{(q)}\right) \quad \textbf{(3.1)}$$

onde os índices escolhidos aleatoriamente $\alpha, \beta, \gamma \in \{1,..., Np\}$ são inteiros distintos entre si. O número de indivíduos da população, Np, deve ser maior ou igual a 4. F é um número $\in \Re$ e constante pertencente ao intervalo [0,2] que controla a amplitude da diferença ponderada. A Figura 15 mostra um exemplo bidimensional que ilustra os diferentes vetores que participam da geração do vetor doador $V^{(q+1)}$.

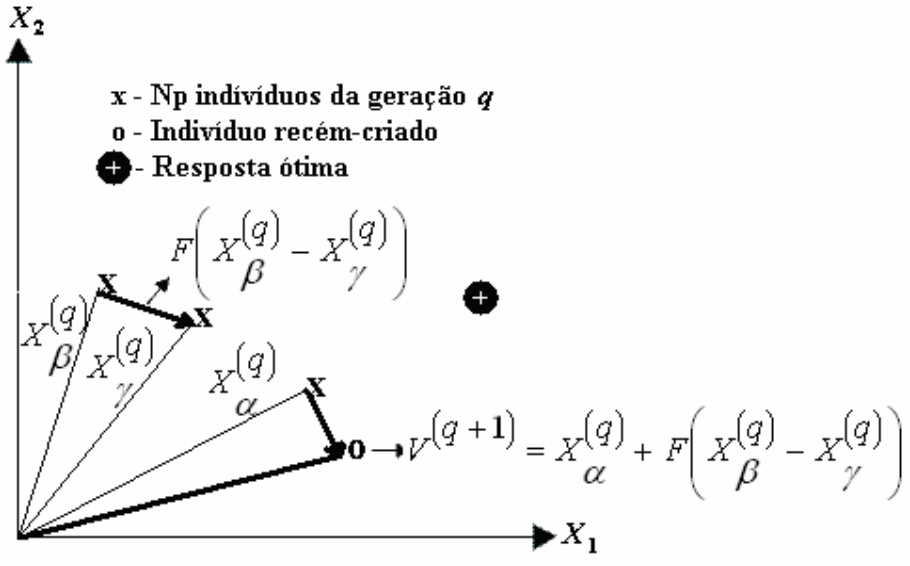

Figura 6: Processo de gerar o vetor doador $V^{(q+1)}$ para uma função objetivo para um problema bidimensional.

Se o número de indivíduos da população é grande o suficiente, a diversidade da população pode ser melhorada usando duas diferenças ponderadas para perturbar um vetor existente, ou seja, cinco vetores distintos são escolhidos aleatoriamente na população atual. O vetor diferença ponderada usa dois pares de diferenças ponderadas e é usado para perturbar o quinto vetor ou o melhor vetor da população atual. Este processo pode dado por:

$$V^{(q+1)} = X_{\alpha}^{(q)} + F\left(X_{\lambda}^{(q)} - X_{\beta}^{(q)} + X_{\gamma}^{(q)} - X_{\delta}^{(q)}\right) \qquad \textbf{(3.2)}$$

Ou

$$V^{(q+1)} = X_{best}^{(q)} + F\left(X_{\lambda}^{(q)} - X_{\beta}^{(q)} + X_{\gamma}^{(q)} - X_{\delta}^{(q)}\right) \qquad \textbf{(3.3)}$$

Os índices escolhidos aleatoriamente $\alpha, \beta, \gamma, \lambda, \delta \in \{1,..., Np\}$, são inteiros mutuamente distintos e diferentes do índice d, tal que $Np \geq 6$.

Operador de Cruzamento: O cruzamento é introduzido para aumentar a diversidade dos indivíduos que sofreram a mutação. Assim, as componentes do vetor experimental $U^{(q+1)}$ são formadas conforme a expressão:

$$U_i^{(q+1)} = \begin{cases} V_i^{(q+1)}, & se \quad r_i \le CR \\ x_{d,i}^{(q)}, & se \quad r_i > CR, i = 1,...,n \end{cases} \tag{3.4}$$

onde r_i é um número gerado aleatoriamente com distribuição uniforme no intervalo [0,1]. Os elementos $x_{d,i}$ constituem o vetor alvo $X_d^{(q)}$, CR é a probabilidade do cruzamento ocorrer, representa a probabilidade do vetor experimental herdar os valores das variáveis do vetor doador, e está compreendida entre 0 e 1, sendo fornecida pelo usuário ou projetista. Quando CR = 1, por exemplo, todas as componentes do vetor experimental virão do vetor doador $V^{(q+1)}$. Por outro lado, se CR = 0, todas as componentes do vetor experimental virão do vetor alvo $X_d^{(q)}$.

Se após o cruzamento uma ou mais componentes do vetor experimental estiver fora da região de busca, fazem-se as correções:

$$\begin{aligned} &\text{Se } u_i < x_i^L \text{ , então } u_i = x_i^L; \\ &\text{Se } u_i < x_i^U \text{ , então } u_i = x_i^U. \end{aligned} \tag{3.5}$$

Seleção: A seleção é o processo de produzir melhores filhos. Diferentemente de outros algoritmos evolutivos, a evolução diferencial não usa hierarquia (elitismo) nem seleção proporcional. Em vez disso, o custo do vetor experimental $U^{(q+1)}$ é calculado e comparado com o custo do vetor alvo $X_d^{(q)}$. Se o custo do vetor alvo for menor que o custo do vetor experimental, o vetor alvo é permitido avançar para a próxima geração. Caso contrário, o vetor experimental substitui o vetor alvo na geração seguinte. Em outras palavras, este processo pode ser escrito tal que:

$$\begin{aligned} &\text{Se } f\left(U^{(q+1)}\right) \le f\left(X_d^{(q)}\right), \text{ então } X_d^{(q+1)} = U^{(q+1)}; \\ &\text{Se } f\left(U^{(q+1)}\right) > f\left(X_d^{(q)}\right), \text{ então } X_d^{(q+1)} = X_d^{(q)}. \end{aligned} \tag{3.6}$$

O procedimento mencionado é finalizado através de algum critério de parada, sendo que um número máximo de gerações deve ser estabelecido.

3.1.14.2 Estratégias da ED

As estratégias da evolução diferencial podem variar de acordo com o tipo de indivíduo a ser modificado na formação do vetor doador, o número de indivíduos considerados para a perturbação e o tipo de cruzamento a ser utilizado, podendo ser escritas como: ED/*a/b/c*.

a. Especifica o vetor a ser perturbado, podendo ser "*rand*" (um vetor da população escolhido aleatoriamente) ou "*best*" (o vetor de menor custo da população);

b. Determina o número de diferenças ponderadas usadas para a perturbação de *a*;

c. Denota o tipo de cruzamento (*exp*: exponencial; *bin*: binomial).

Em [STO95], Storn e Price deram o princípio de trabalho da estratégia básica usando apenas o operador cruzamento binomial (devido aos experimentos binomiais independentes), onde o cruzamento é executado em cada variável sempre que um número $r \in [0,1]$ aleatório for menor que a probabilidade de cruzamento *CR*.

Alguns anos mais tarde, Storn e Price [STO97] desenvolveram mais estratégias usando o operador cruzamento exponencial, em que o cruzamento é executado nas variáveis em um laço até que esteja dentro do limite de *CR*. A primeira vez que um número $r \in [0, 1]$ aleatório ultrapassa o valor de *CR*, nenhum cruzamento é executado e as variáveis restantes não são modificadas.

A tabela 2 mostra a formulação usada para cada uma das 10 estratégias:

Tabela 2: Estratégias usadas na ED com suas formulações

Número	Mutação	Notação
1	$V^{(q+1)} = X_{\alpha}^{(q)} + F\left(X_{\beta}^{(q)} - X_{\gamma}^{(q)}\right)$	ED/rand/1/bin
2	$V^{(q+1)} = X_{best}^{(q)} + F\left(X_{\beta}^{(q)} - X_{\gamma}^{(q)}\right)$	ED/best/1/bin
3	$V^{(q+1)} = X_{\alpha}^{(q)} + F\left(X_{\lambda}^{(q)} - X_{\beta}^{(q)} + X_{\gamma}^{(q)} - X_{\delta}^{(q)}\right)$	ED/rand/2/bin
4	$V^{(q+1)} = X_{best}^{(q)} + F\left(X_{\alpha}^{(q)} - X_{\beta}^{(q)} + X_{\gamma}^{(q)} - X_{\delta}^{(q)}\right)$	ED/best/2/bin
5	$V^{(q+1)} = X_{old}^{(q)} + F\left(X_{best}^{(q)} - X_{old}^{(q)} + X_{\gamma}^{(q)} - X_{\delta}^{(q)}\right)$	ED/rand-to-best/2/bin
6	$V^{(q+1)} = X_{\alpha}^{(q)} + F\left(X_{\beta}^{(q)} X_{\gamma}^{(q)}\right)$	ED/rand/1/exp
7	$V^{(q+1)} = X_{best}^{(q)} + F\left(X_{\beta}^{(q)} - X_{\gamma}^{(q)}\right)$	ED/best/1/exp
8	$V^{(q+1)} = X_{\alpha}^{(q)} + F\left(X_{\lambda}^{(q)} - X_{\beta}^{(q)} + X_{\gamma}^{(q)} - X_{\delta}^{(q)}\right)$	ED/rand/2/exp

9	$V^{(q+1)} = X_{best}^{(q)} + F\left(X_{\alpha}^{(q)} - X_{\beta}^{(q)} + X_{\gamma}^{(q)} - X_{\delta}^{(q)}\right)$	ED/best/2/exp
10	$V^{(q+1)} = X_{old}^{(q)} + F\left(X_{best}^{(q)} - X_{old}^{(q)} + X_{\gamma}^{(q)} - X_{\delta}^{(q)}\right)$	ED/rand-to-best/2/exp

A idéia principal atrás do ED é o esquema de geração de vetores de parametrização. Mutações e cruzamentos são usados para gerar novos vetores e a seleção é a que irá determinar qual desses vetores irá sobreviver para a próxima geração. Por ser algo totalmente customizável, acaba-se perdendo tempo na correta configuração do sistema até se conseguir obter o melhor resultado. O problema é que o sistema de tentativa e erro, usado para se tentar achar a melhor parametrização só pode ser testado rodando-se novamente o sistema e verificando-se o seu resultado com o que já tinha se obtido. Para resolver tal problema, [BRE06] propôs um sistema auto-adaptativo para o controle dos parâmetros. Para isso, estendeu-se cada indivíduo da população para ter os seus valores customizados. Conforme ocorre a sobrevivência dos elementos, acaba-se se gerando também os melhores valores para customização.

Para análise foram usadas as 6 primeiras estratégias, sendo as 5 primeiras de modo binário e a última em modo exponencial. Desta forma pode-se fazer uma análise completa entre todos os modos binários e ainda compara-los com um modo exponêncial.

Capítulo 4

Implementação Computacional e Análise de Resultados

A heurística Lin-Kernighan [LIN73] continua sendo um dos métodos mais eficazes para se conseguir chegar ao um resultado promissor ao problema do PCV. Porém, tal método ao ser iniciado faz uma geração aleatória de uma possível solução do problema e usa-a como base para sua troca inicial. Porém, dependendo do resultado aleatório o resultado final pode variar ou então demorar mais tempo para se alcançá-lo. Para que se minimize a função objetivo de tal problema, propõe-se a implementação de um sistema híbrido, onde um (ou mais) métodos diferentes ajudam o LKH a escolher um resultado próximo ao resultado ótimo quando conhecido na literatura, facilitando a quantidade de iterações para se obter tal resultado. A implementação computacional adotada foi a de validar cinco abordagens de métodos de otimização combinatória baseados em:

i) Lin-Kernighan Helsgaun sem nenhuma modificação e recebendo como *tour* inicial uma possível solução gerada aleatoriamente pelo próprio programa;
ii) Evolução Diferencial aplicada ao problema do PCV.
iii) Híbrido Evolução Diferencial e Lin-Kerninghan Helsgaun;
iv) Híbrido Evolução Diferencial e VNS;
v) Híbrido Evolução Diferencial, VNS e Lin Kernighan Helsgaun.

A Figura 17 mostra os testes realizados.

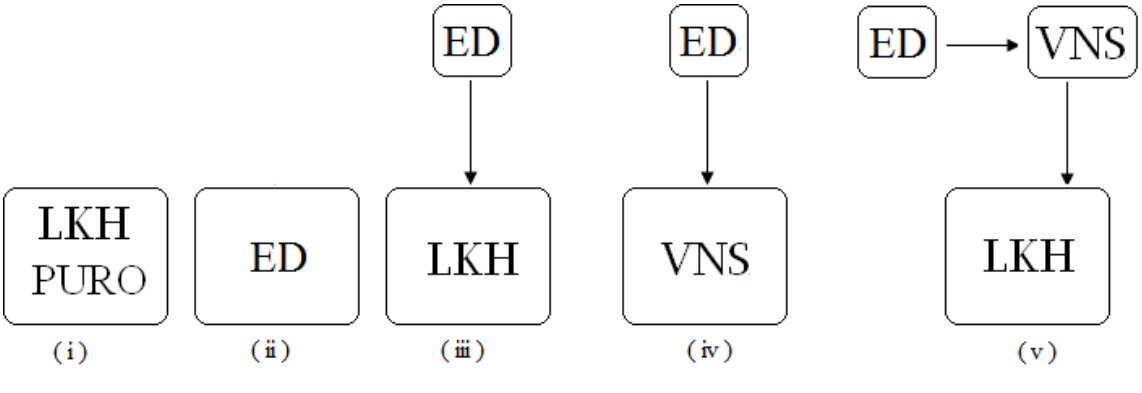

Figura 7: Representação dos modelos propostos.

Como pode ser observado na Figura 17, tem-se primeiramente uma análise dos *tours* para os métodos LKH e ED rodados separadamente, logo após, passa-se como *tour* inicial para o LKH, o resultado obtido no ED e por fim, usa-se o resultado da ED como *tour* inicial do VNS e logo após passa-se o resultado obtido para o LKH. Para os modelos híbridos, pode-se verificar nos Quadros 11 e 12, as mudanças ocorridas nos pseudocódigos mostrados anteriormente.

Quadro 11: Pseudocódigo para o sistema híbrido ED + LKH

1. Receber melhor resultado de ED;

2. Seja $i = 1$, escolher t_1.

3. Escolher $x = (t_1 , t_2) \in T$.

4. Escolher $y_1 = (t_2 , t_3) \notin T$ tal que $G_1 > 0$. Se não for possível, ir para o passo 12.

5. $i = i + 1$.

6. Escolher $x_i = (t_{2i-1}, t_{2i}) \in T$ tal que:

   a) $x_i \neq y_s$ para todo s < i.

   b) Se t_{2i} é ligado a t_1, a configuração resultante é um trajeto T', e se este é um trajeto melhor que T ir para o passo 2.

7. Escolher $y_i = (t_{2i} , t_{2i+1}) \in T$ tal que:

   c) $G_i > 0$.

   d) $y_i \neq x_s$ para todo $s \leq i$.

   e) x_{i+1} deve existir.

8. Se houver uma alternativa (ligação) não executada para y_2: $i = 2$ e ir para o passo 7.

9. Se houver uma alternativa (ligação) não executada para x_2: $i = 2$ e ir para o passo 6.

10. Se houver uma alternativa (ligação) não executada para y_1: $i = 1$ e ir para o passo 4.

11. Se houver uma alternativa (ligação) não executada para x_1: $i = 1$ e ir para o passo 3.
12. Se houver uma alternativa (ponto) não executada para t_1: ir para o passo 2.
13. Parar se critério de parada alcançado (ou ir para o passo 1).

Quadro 12: Pseudocódigo para o sistema híbrido ED + VNS + LKH

1. **ED passa para o VNS seu melhor resultado;**
2. **VNS gera *tour* inicial;**
3. **Receber melhor resultado de VNS;**
4. Seja $i = 1$, escolher t_1.
5. Escolher $x = (t_1 , t_2) \in T$.
6. Escolher $y_1 = (t_2 , t_3) \notin T$ tal que $G_1 > 0$. Se não for possível, ir para o passo 12.
7. $i = i + 1$.
8. Escolher $x_i = (t_{2i-1}, t_{2i}) \in T$ tal que:

a) $x_i \neq y_s$ para todo s < i.

b) Se t_{2i} é ligado a t_1, a configuração resultante é um trajeto T', e se este é um trajeto melhor que T ir para o passo 2.

9. Escolher $y_i = (t_{2i} , t_{2i+1}) \in T$ tal que:

c) $G_i > 0$.

d) $y_i \neq x_s$ para todo $s \leq i$.

e) x_{ii+1} deve existir.

10. Se houver uma alternativa (ligação) não executada para y_2: $i = 2$ e ir para o passo 7.
11. Se houver uma alternativa (ligação) não executada para x_2: $i = 2$ e ir para o passo 6.
12. Se houver uma alternativa (ligação) não executada para y_1: $i = 1$ e ir para o passo 4.
13. Se houver uma alternativa (ligação) não executada para x_1: $i = 1$ e ir para o passo 3.
14. Se houver uma alternativa (ponto) não executada para t_1: ir para o passo 2.
15. Parar se critério de parada alcançado (ou ir para o passo 1).

Os resultados foram obtidos através de 50 execuções para cada um dos métodos em um PC (processador de Intel Core2 Dual 1.6GHz com 1024MB de RAM (*Random Access Memory*).

4.1 Forma de Execução

Todos os dados referentes aos *tours*, como posição e quantidade de cidades foram obtidas da TSPLIB [APP95]. Esta biblioteca é um conjunto de instâncias as quais já foram usadas em outras análises e que também já tem o seu resultado ótimo, determinado a *priori* por outros autores, o que pode ser usado como referência para determinar a porcentagem de acerto do método testado. As instâncias utilizadas foram selecionadas aleatoriamente, seguindo apenas como base a seguinte regra nos casos de problema de PCV simétricos:

- 2 instâncias menores que 200;
- 2 instâncias entre 200 e 2000;
- 3 instâncias maiores que 2000.

Cada uma das instâncias foi executada 50 vezes, para obtenção de uma melhor análise dos resultados. Todas elas seguiram as seguintes etapas:

1. Primeiramente roda-se a ED usando sempre uma nova semente, que faz com que se gere aleatoriamente sempre um novo *tour* toda vez que é executado. Deste resultado, gera-se uma sequência de cidades que são repassadas para o VNS como sendo o *tour* inicial.
2. Após terminar o ED, executa-se o VNS, que executa juntamente no mesmo código e usa as mesmas variáveis utilizadas na finalização do ED. O VNS está configurado para executar 100 vezes; para cada instância executada, a população total é dividida em 6 menores grupos, na qual cada uma delas é uma vizinhança. O programa começa então aexecutar para cada vizinhança, 100 vezes uma busca local do tipo 2-*opt*. Após rodar a busca local, verifica-se o resultado e, caso seja melhor do que o anterior, aumenta-se o tamanho da vizinhança, até no final acabar verificando todos os possíveis caminhos. No final, gera-se um arquivo com terminação *.int.
3. Após terminar o VNS, executa-se o LKH, passando para o mesmo arquivo com as cidades já determinadas pelo VNS. Desta forma, toda vez que se executa o LKH, tem-se um novo *tour* inicial gerado pelo VNS, anteriormente, e também a escolha dos caminhos trocados também é aleatória, uma vez que também tem-se um novo gerador de números aleatórios a cada rodada. Deste resultado gera-se um arquivo chamado *.out.
4. Tem-se ainda mais uma comparação que é a execução do LKH diretamente com o resultado do ED, sem passar pelo VNS, para se analisar o resultado

obtido com ou sem o VNS. Desta forma pode-se verificar o desempenho do sistema apenas com o ED.

5. Por fim, executa-se novamente o LKH, porém, sem passar o *tour* inicial gerado pelo VNS. Desta forma, o programa gera aleatoriamente um novo *tour* inicial a cada rodada, pois a semente usada é diferente. Assim, pode-se utilizar este resultado como parâmetro de comparação entre as execuções com ou sem o melhoramento do *tour* inicial para o LKH.

Todos os procedimentos foram executados da mesma maneira em todas as instâncias e também no mesmo computador. O *software* utilizado foi executado diretamente no Sistema Operacional Windows XP. A plataforma de desenvolvimento foi o DEV-C++ que é um ambiente de desenvolvimento completo, no qual usa-se como compilador *default* o Mingw, que é um compilador ANSI C/C++ gratuito, que utiliza a licença de uso chamada GPL (*General Public License*), que, permite que seja usada gratuitamente desde que seja disponibilizado pelo usuário o nome do autor do programa e o seu código fonte quando for redistribuir o programa.

As Tabelas 3 a 9 foram divididas por problemas selecionados na biblioteca TSPLIB. Cada uma delas apresenta os resultados obtidos para cada tipo de método que existe no ED. Como são 6 ao total, tem-se então 6 rodadas do ED. As outras linhas são referentes aos resultados obtidos através do VNS e também do ED+VNS+LKH. Todos os resultados mostram também a porcentagem que o resultado está próximo do resultado ótimo.

Os tempos usados nas tabelas foram calculados na média gerada após 50 execuções. Os tempos estão todos em segundos e seu valor corresponde ao custo isolado de cada método.

O critério de parada usado no ED e no VNS foram dois parâmetros: a obtenção de uma solução ótima ou o número de iterações(100). Após o critério de parada ser alcançado, usa-se o resultado obtido como sendo o *tour* inicial no LKH.

De modo geral, tem-se um resultado não satisfatório apenas usando a heurística ED, porém, a mesma não tem como finalidade achar uma solução ótima para o problema, e sim, determinar um *tour* considerado bom que ajude o LKH a determinar um melhor resultado. Neste quesito, pode-se notar que o ED ajudou para obtenção de resultados ótimos em vários casos, entre eles o ATT532 e o PLA7397. O resultado médio e a média de tempo foi sempre melhor quando se passou o resultado obtido primeiramente no ED. Mesmo nos

outros casos que são considerados de tamanho grande, como o U2152 e o PCB3038, consegue-se ao menos, sempre um método do ED ou um método ED+LKH com resultado melhor do que o obtido apenas com o LKH puro.

Já com a implantação do VNS, têm-se melhores resultados que os obtidos ao ED, isso se deve ao fato de se estar sempre passando por um resultado já obtido, anteriormente, pela ED e que o VNS apenas melhora fazendo sucessivas buscas locais. O tempo de execução do VNS é realmente rápido se comparado com o ED e com isso melhorou o desempenho do LKH, mostrando que a combinação ED+LKH consegue obter um melhor caminho.

No caso do problema EIL101, todos os algoritmos que passaram um *tour* inicial ao LKH conseguiram obter um resultado ótimo, conforme visto na tabela 4. Porém, vale salientar que o os algoritmos que usaram o ED tiveram um acréscimo de tempo por causa do tempo de execução do mesmo, mas o resultado passado ao LKH auxiliou em uma obtenção mais rápida do resultado ótimo. Comparando-se o resultado obtido com [DUA06], nota-se que houve um melhor desempenho usando-se o ED juntamente com VNS, porém com um custo de tempo maior que o encontrado na solução com algoritmos de busca exata. Já no caso com a resposta sendo usada como *tour* inicial para o LKH, obteve-se um desempenho superior ao dele, uma vez que o tempo de processamento chega a ser menor do que 1 segundo.

Tabela 3: Resultados obtidos para o problema EIL101 – Resultado ótimo: 629

Método	Função Objetivo						Desvio Padrão	Tempo(s)
	Mínimo (%)		Média (%)		Máximo (%)			
ED – Método 1	2189	28,73	2276	27,63	2343	26,84	36,11	1,95
ED – Método 2	2179	28,86	2285	27,52	2368	26,56	42,13	2,00
ED – Método 3	2167	29,02	2274	27,66	2364	26,60	47,66	1,98
ED – Método 4	2088	30,12	2278	27,61	2348	26,78	49,73	1,99
ED – Método 5	2172	28,95	2285	27,52	2367	26,57	44,61	2,03
ED – Método 6	1788	35,17	2011	31,27	2223	28,29	112,39	1,99
ED – Método 1 + VNS	1460	43,08	1612	39,01	1765	35,63	68,35	0,41
ED – Método 2 + VNS	1489	42,24	1611	39,04	1779	35,35	69,80	0,08
ED – Método 3 + VNS	1470	42,78	1613	38,99	1794	35,06	75,36	0,08
ED – Método 4 + VNS	1485	42,35	1604	39,21	1667	37,73	61,24	0,14
ED – Método 5 + VNS	1431	43,95	1616	38,92	1803	34,88	75,25	0,13
ED – Método 6 + VNS	1455	43,23	1576	29,91	1822	34,52	76,31	0,29
ED – Método 1 + VNS + LKH	629	100	629	100	629	100	0,00	0,02
ED – Método 2 + VNS + LKH	629	100	629	100	629	100	0,00	< 0,01
ED – Método 3 + VNS + LKH	629	100	629	100	629	100	0,00	0,02
ED – Método 4 + VNS + LKH	629	100	629	100	629	100	0,00	0,00
ED – Método 5 + VNS + LKH	629	100	629	100	629	100	0,00	0,03
ED – Método 6 + VNS + LKH	629	100	629	100	629	100	0,00	< 0,01
ED – Método 1 + LKH	629	100	629	100	629	100	0,00	< 0,01
ED – Método 2 + LKH	629	100	629	100	629	100	0,00	< 0,01
ED – Método 3 + LKH	629	100	629	100	629	100	0,00	0,02
ED – Método 4 + LKH	629	100	629	100	629	100	0.00	< 0,01
ED – Método 5 + LKH	629	100	629	100	629	100	0.00	< 0,01
ED – Método 6 + LKH	629	100	629	100	629	100	0.00	< 0,01
LKH	629	100	629	100	629	100	0.00	0.02

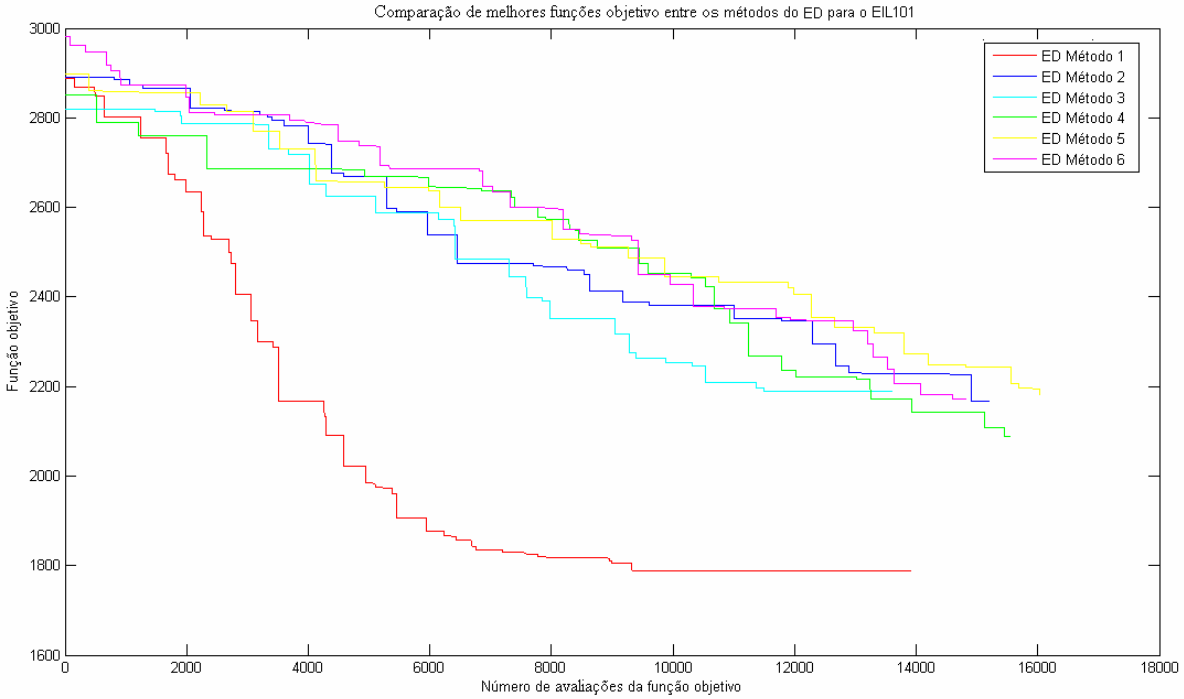

Figura 8: Comparação de melhores funções objetivo entre os métodos do ED para o EIL101

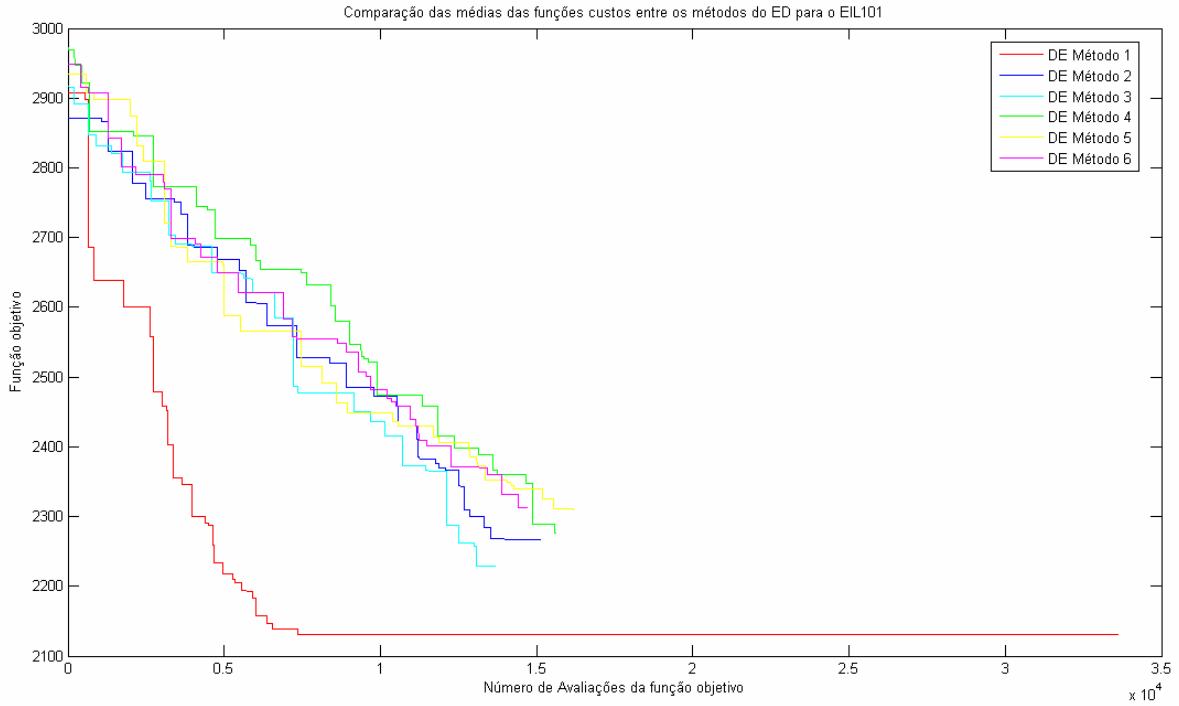

Figura 9: Comparação das médias das funções objetivo entre os métodos do ED para o EIL101

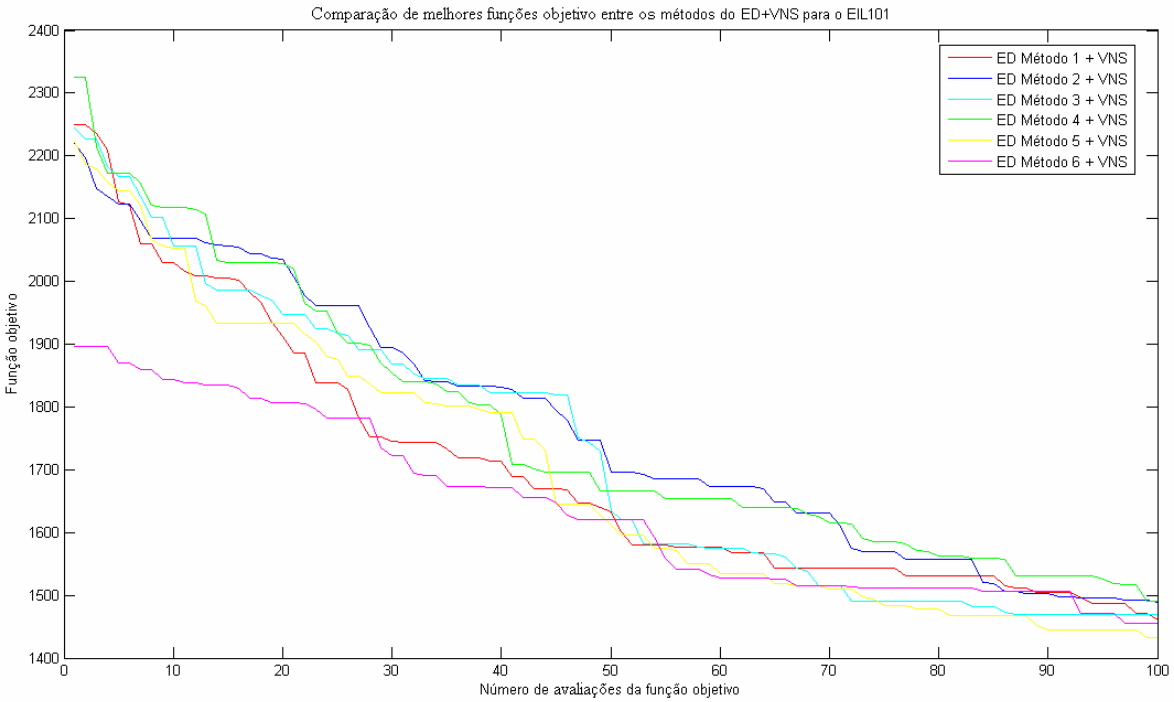

Figura 10: Comparação das melhores funções objetivo entre os métodos do ED+VNS para o EIL101

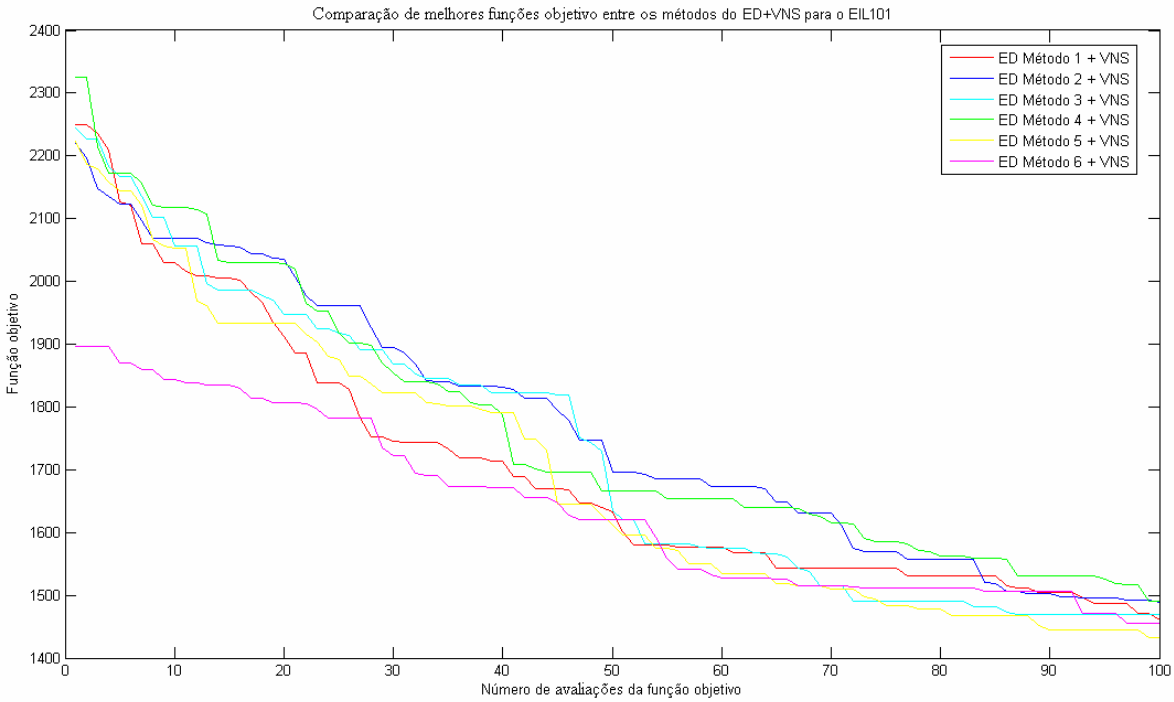

Figura 11: Comparação das médas das funções objetivo entre os métodos do ED+VNS para o EIL101

Em termos do problema KROA150, também obteve-se resultados ótimos em todos os métodos analisados, mas novamente, o tempo de execução do LKH melhorou em alguns métodos. Mas pode-se também notar que os resultados obtidos pelo método 5 do ED fizeram com que o LKH piora-se no tempo de execução. Isso se deve ao motivo de que o método 5 fez uma busca que piora-se a melhor execução em relação aos outros métodos. Se compararmos o resultado obtido com [DUA06], pode-se notar que o ED isoladamente conseguiu obter um resultado aproximadamente 15% melhor ao busca tabu e 5% melhor que os algoritmos genéticos. Já com a combinação entre ED+VNS, obtivemos um resultado aproximadamente 25% melhor que a busca tabu e 10% melhor no caso do algoritmo genético.

Tabela 4: Resultados para o problema KROA150 – Resultado ótimo: 26524

Método	Função Objetivo						Desvio Padrão	Tempo(s)
	Mínimo (%)		Média (%)		Máximo (%)			
ED – Método 1	168126	15,77	177449	14,94	182049	14,56	3008,65	2,58
ED – Método 2	172109	15,41	180050	14,73	185077	14,33	2238,78	2,63
ED – Método 3	170487	15,55	176889	14,99	181643	14,60	2742,13	2,65
ED – Método 4	170376	15,56	178963	14,82	184784	14,35	2842,02	2,56
ED – Método 5	168804	15,71	179898	14,74	183759	14,40	3020,63	2,64
ED – Método 6	143048	18,54	157270	16,86	207023	12,81	2989,42	2,58
ED – Método 1 + VNS	107224	24,73	119526	22,19	127406	20,81	3742,95	0,67
ED – Método 2 + VNS	108353	24,47	120019	22,09	131124	20,22	4433,32	0,13
ED – Método 3 + VNS	104327	25,42	119652	22,16	126884	20,90	4209,34	0,14
ED – Método 4 + VNS	110696	23,96	119480	22,19	123236	21,52	4566,16	0,23
ED – Método 5 + VNS	106460	24,51	118810	22,32	126106	21,03	4933,83	0,22
ED – Método 6 + VNS	109025	24,32	118190	22,44	109025	24,32	4295,77	0,47
ED – Método 1 + VNS + LKH	26524	100	26524	100	26524	100	0,00	0,06
ED – Método 2 + VNS + LKH	26524	100	26524	100	26524	100	0,00	0,02
ED – Método 3 + VNS + LKH	26524	100	26524	100	26524	100	0,00	0,01
ED – Método 4 + VNS + LKH	26524	100	26524	100	26524	100	0,00	0,02
ED – Método 5 + VNS + LKH	26524	100	26524	100	26524	100	0,00	0,04
ED – Método 6 + VNS + LKH	26524	100	26524	100	26524	100	0,00	0,02
ED – Método 1 + LKH	26524	100	26524	100	26524	100	0,00	0,04
ED – Método 2 + LKH	26524	100	26524	100	26524	100	0,00	< 0,01
ED – Método 3 + LKH	26524	100	26524	100	26524	100	0,00	< 0,01
ED – Método 4 + LKH	26524	100	26524	100	26524	100	0,00	0,02
ED – Método 5 + LKH	26524	100	26524	100	26524	100	0,00	0,06
ED – Método 6 + LKH	26524	100	26524	100	26524	100	0,00	0,00
LKH	26524	100	26524	100	26524	100	0,00	0,04

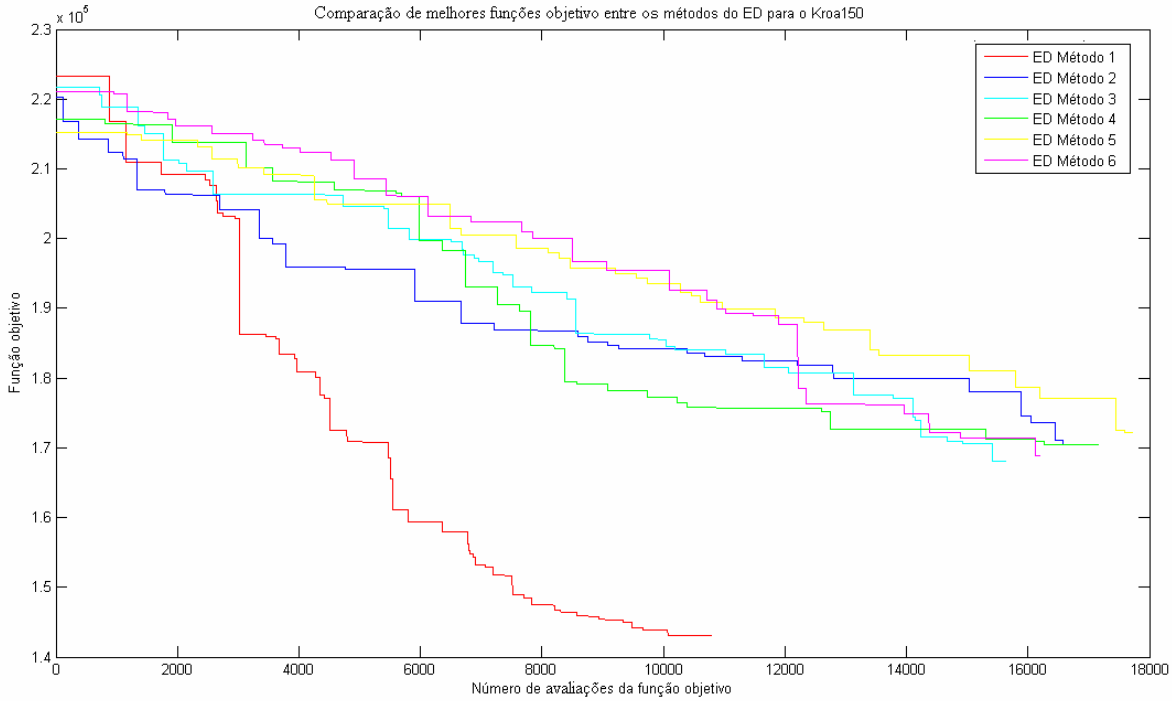

Figura 12: Comparação de melhores funções objetivo entre os métodos do ED para o KROA150

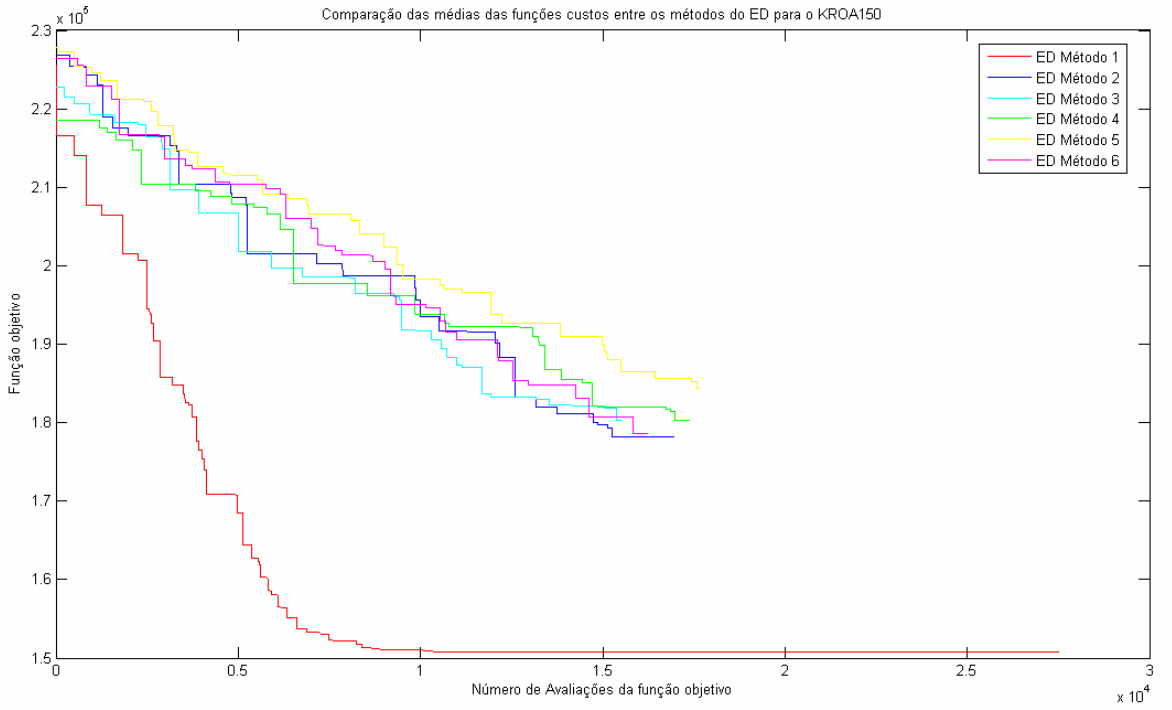

Figura 13: Comparação das médias das funções objetivos entre os métodos do ED para o KROA150

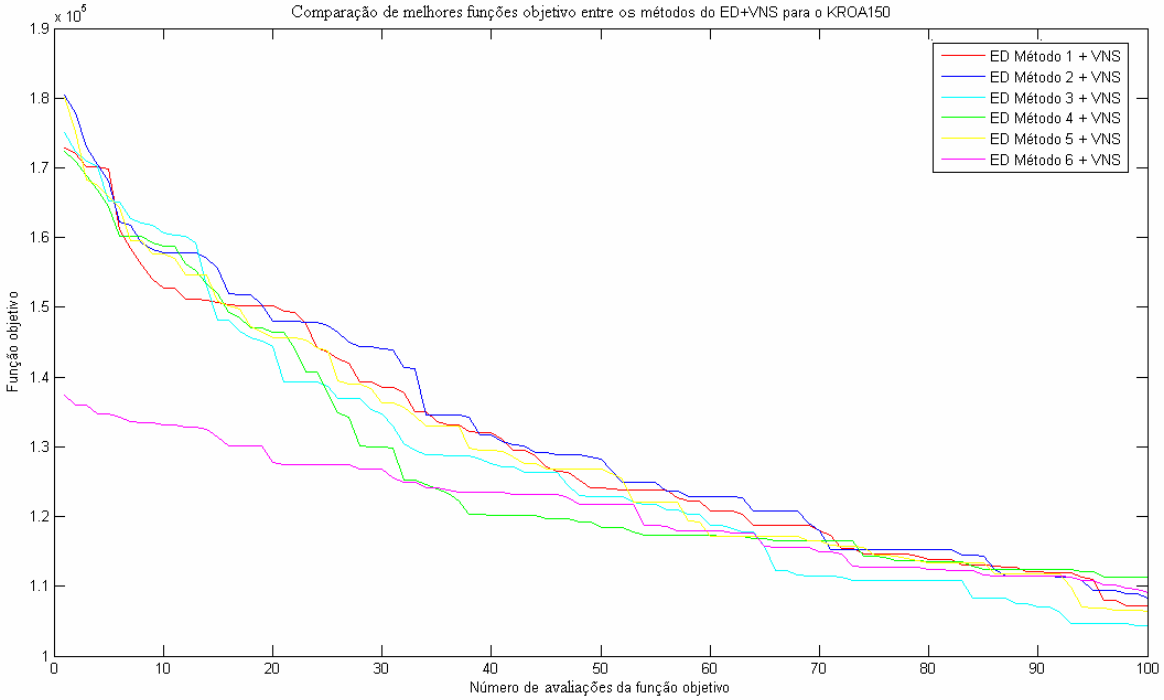

Figura 14: Comparação das melhores funções objetivo entre os métodos do ED+VNS para o KROA150

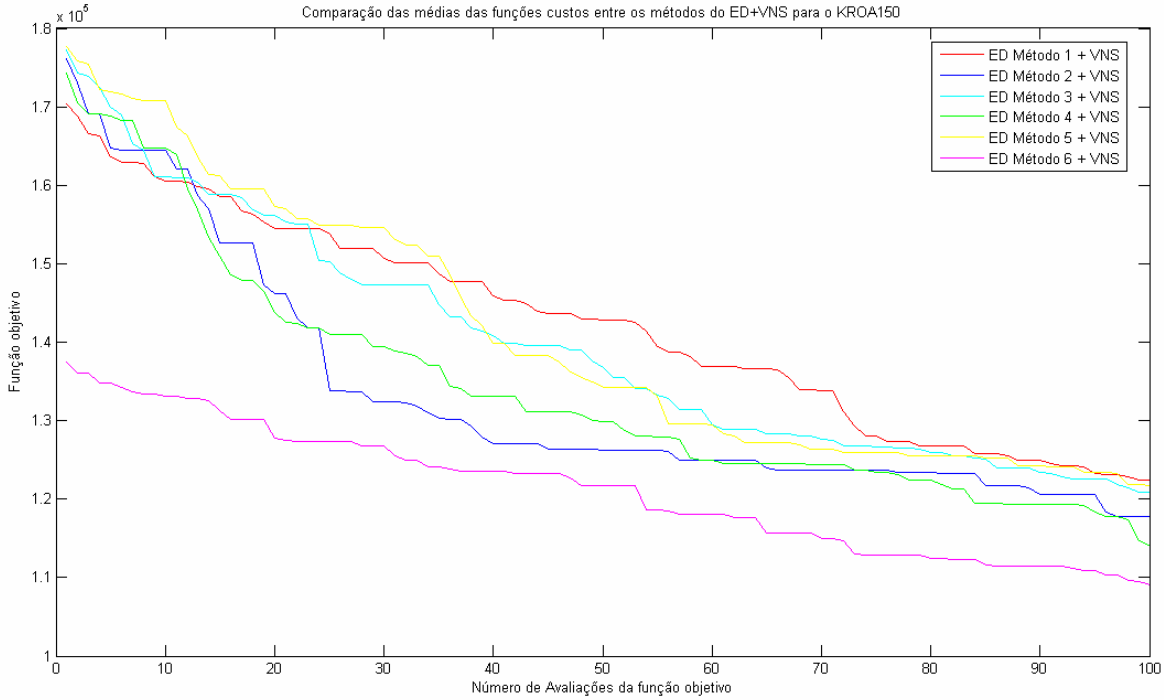

Figura 15: Comparação das méidas das funções objetivo entre os métodos do ED+VNS para o KROA150

Baseado nos resultados de execução do problema LIN318, nota-se que na média, nenhum método conseguiu atingir 100% de precisão, mas que os todos métodos tiveram um desempenho igual, ou melhor, em relação a achar o melhor *tour*. Porém, nota-se que a pior média, é a que conseguiu executar de maneira mais rápida, mas a melhor média na distância ainda conseguiu rodar de maneira mais rápida do que o LKH com uma solução gerada aleatoriamente pelo próprio programa. Comparando-se esse resultado com [DUA06], nota-se resultado parecido com o caso do kroa150, porém, a média encontrada com ED ou com ED+VNS é inferior a média encontrada quando se utiliza o AG. Ao se comparar os resultados encontrados em [DOR96], consegue-se ter a mesma porcentagem de ótimos encontrados, mas o tempo de execução chegou a passar de 500 segundos, o que demonstra um acréscimo de aproximadamente 1000% em relação ao método ED+VNS+LKH.

Tabela 5: Resultados para o problema LIN318 – Resultado ótimo: 42029

Método	Função Objetivo						Desvio Padrão	Tempo(s)
	Mínimo (%)		Média (%)		Máximo (%)			
ED – Método 1	467121	08,99	476169	08,82	484940	08,66	4353,08	4,97
ED – Método 2	466907	09,00	482983	08,70	490516	08,56	5025,56	5,20
ED – Método 3	459468	09,14	475580	08,83	483849	08,68	5179,2	5,12
ED – Método 4	472169	08,91	482372	08,71	490431	08,56	4297,80	4,96
ED – Método 5	463845	09,06	479667	08,76	487100	08,62	4398,14	5,32
ED – Método 6	405114	10,37	501050	08,38	524048	08,02	28131	5,09
ED – Método 1 + VNS	317948	13,21	337404	12,45	359152	11,70	8242,69	1,56
ED – Método 2 + VNS	318254	13,20	337721	12,44	352126	11,93	7095,97	0,33
ED – Método 3 + VNS	314695	13,35	337712	12,44	349336	12,03	7557,84	0,34
ED – Método 4 + VNS	320522	13,11	338679	12,40	355119	11,83	7298,55	0,54
ED – Método 5 + VNS	320986	13,09	338447	12,41	356861	11,77	8851,89	0,54
ED – Método 6 + VNS	320522	13,11	337043	12,46	355119	11,83	8741,02	1,12
ED – Método 1 + VNS + LKH	42029	100	42066	0,99	42143	0,99	53,51	1,48
ED – Método 2 + VNS + LKH	42029	100	42060	0,99	42143	0,99	51,70	0,30
ED – Método 3 + VNS + LKH	42029	100	42065	0,99	42143	0,99	53,71	0,38
ED – Método 4 + VNS + LKH	42029	100	42080	0,99	42143	0,99	56,72	0,76
ED – Método 5 + VNS + LKH	42029	100	42058	0,99	42143	0,99	49,11	0,48
ED – Método 6 + VNS + LKH	42029	100	42061	0,99	42143	0,99	51,04	1,00
ED – Método 1 + LKH	42029	100	42066	0,99	42143	0,99	52,21	0,50
ED – Método 2 + LKH	42029	100	42067	0,99	42143	0,99	53,28	0,64
ED – Método 3 + LKH	42029	100	42066	0,99	42143	0,99	53,51	0,48
ED – Método 4 + LKH	42029	100	42060	0,99	42143	0,99	50,34	0,48
ED – Método 5 + LKH	42029	100	42064	0,99	42143	0,99	51,28	0,50
ED – Método 6 + LKH	42029	100	42070	0,99	42143	0,99	55,27	0,42
LKH	42029	100	42067	0,99	421439	0,99	52,91	0,50

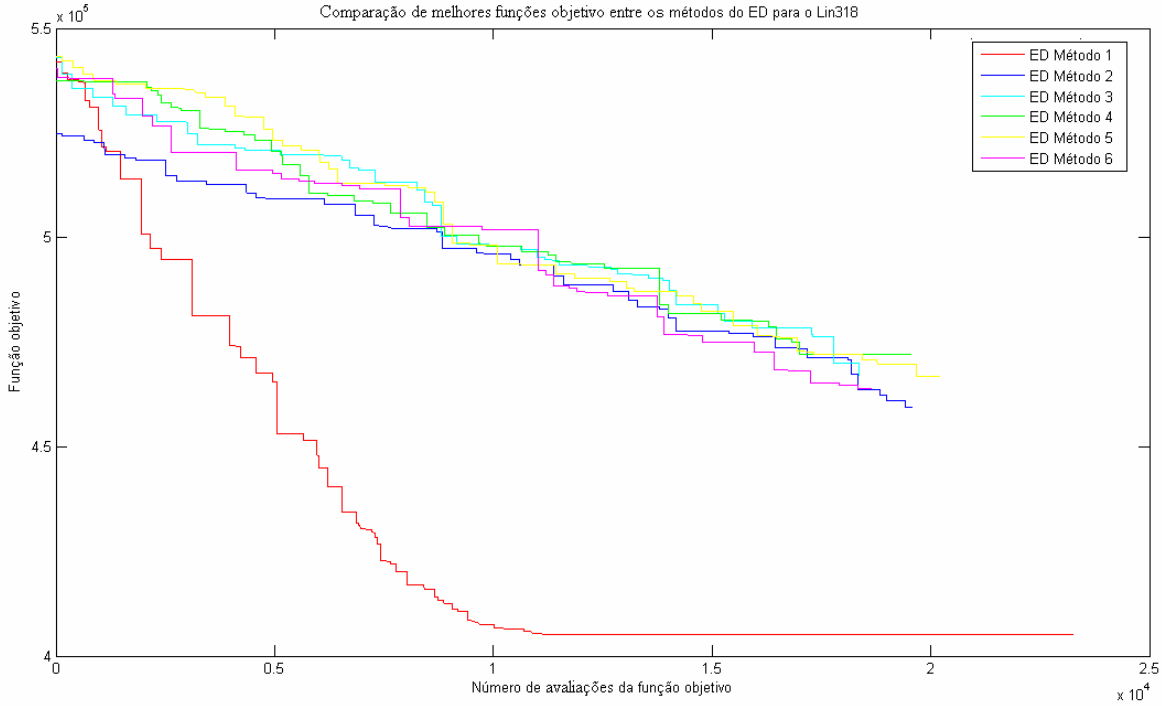

Figura 16: Comparação de melhores funções objetivo entre os métodos do ED para o LIN318

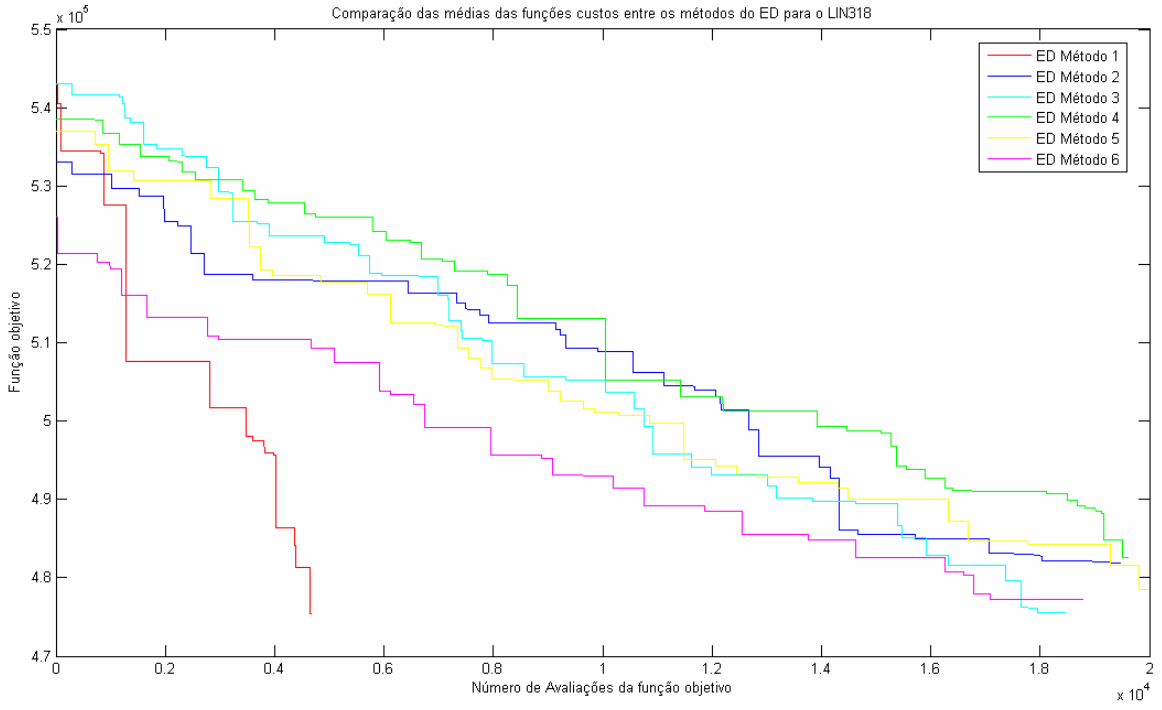

Figura 17: Comparação das médias das funções objetivo entre os métodos do ED para o LIN318

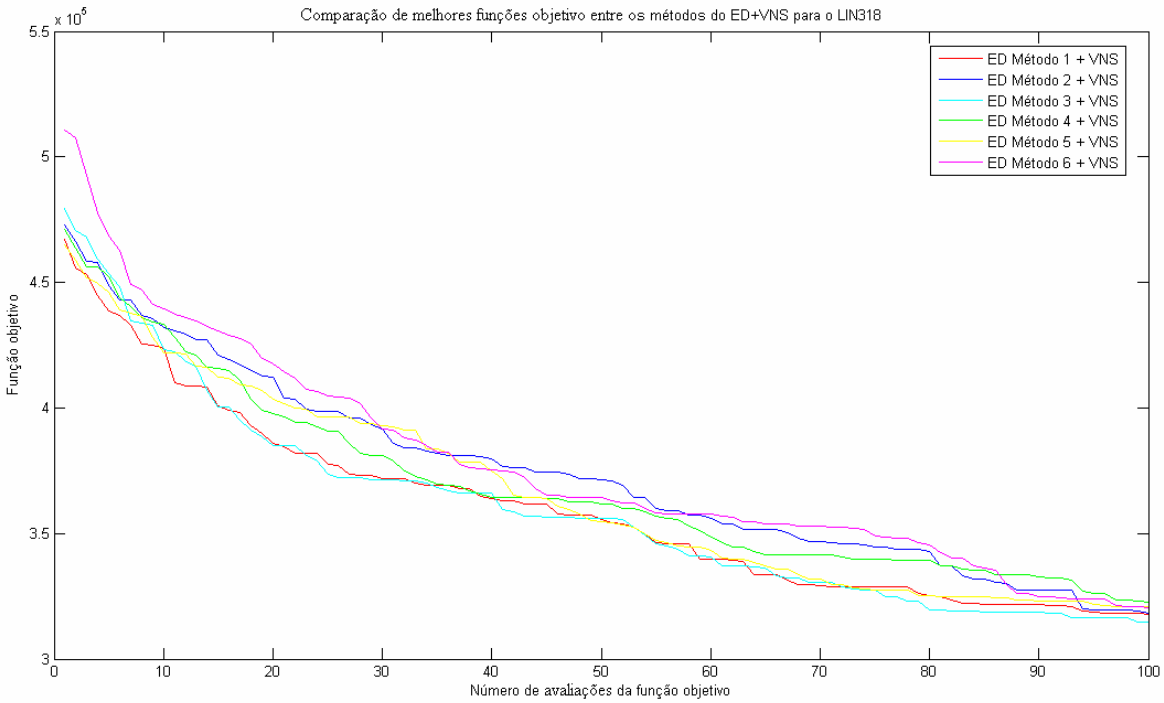

Figura 18: Comparação das melhores funções objetivo entre os métodos do ED+VNS para o LIN318

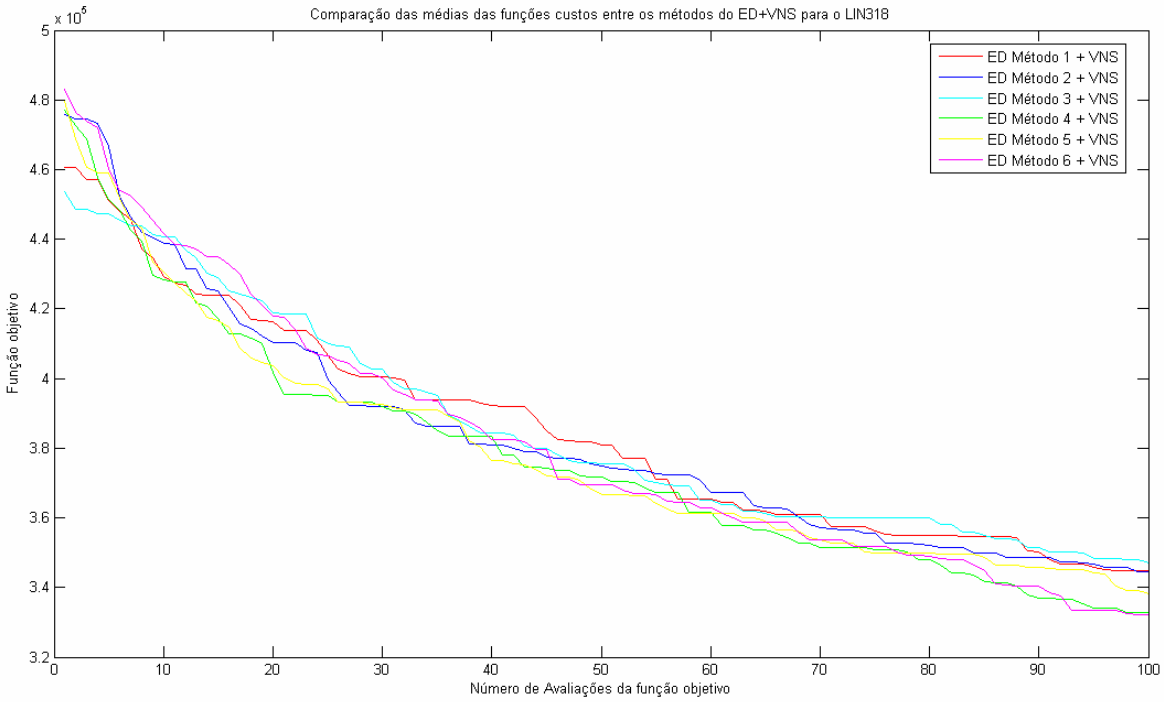

Figura 19: Comparação das médias das funções objetivo entre os métodos do ED+VNS para o LIN318

Em relação ao problema ATT532, pode-se dizer que foi o problema que teve o maior melhoramento após o uso do ED e do VNS. Todos os métodos usados para este problema foram melhores do que o LKH com entrada aleatória. Na média geral todos obtiveram 100% de resultado ótimo e o tempo de execução dos métodos analisados também melhoram. O melhor neste caso foi o método 3 do ED que conseguiu 100% em todas as vezes em que foi executado. Comparando-se com [JUN02] nota-se que o mesmo consegue ter um desempenho igual ao sistema em seu melhor caso, mas na média, houve casos em que não conseguiu achar o melhor caminho. E o seu tempo computacional foi muito superior ao encontrado no método ED+VNS+LKH, chegando a ser aproximadamente 50% mais lento.

Tabela 6: Resultados para o problema ATT532 – Resultado ótimo: 27686

Método	Função Objetivo						Desvio Padrão	Tempo(s)
	Mínimo (%)		Média (%)		Máximo (%)			
ED – Método 1	1343428	02,06	1372841	02,01	1395667	01,98	10863,05	8,44
ED – Método 2	1357546	02,03	1396580	01,98	1419442	01,95	12634,69	8,83
ED – Método 3	1349596	02,05	1378059	02,00	1396414	01,98	11475,83	8,99
ED – Método 4	1370342	02,02	1391801	01,98	1418671	01,95	11109,84	8,53
ED – Método 5	1359629	02,03	1385712	01,99	1407197	01,96	11921,81	9,07
ED – Método 6	1355481	02,04	1442200	01,91	1466610	01,88	23759,00	8,84
ED – Método 1 + VNS	901324	03,07	965280	02,86	1013743	02,73	24323,70	2,66
ED – Método 2 + VNS	911767	03,03	971366	02,85	1023286	02,70	22434,08	0,62
ED – Método 3 + VNS	934762	02,96	972350	02,84	1024879	02,70	20479,16	0,86
ED – Método 4 + VNS	942942	02,93	975590	02,83	1002295	02,76	17529,33	0,93
ED – Método 5 + VNS	929788	02,97	971550	02,84	1030289	02,68	21179,95	0,99
ED – Método 6 + VNS	932823	02,96	970941	02,85	1031975	02,68	20888,72	1,93
ED – Método 1 + VNS + LKH	27686	100	27686	100	27706	0,99	3,95	3,34
ED – Método 2 + VNS + LKH	27686	100	27686	100	27686	100	0,00	0,50
ED – Método 3 + VNS + LKH	27686	100	27686	100	27686	100	0,00	1,03
ED – Método 4 + VNS + LKH	27686	100	27686	100	27686	100	0,00	1,36
ED – Método 5 + VNS + LKH	27686	100	27686	100	27686	100	0,00	1,04
ED – Método 6 + VNS + LKH	27686	100	27686	100	27686	100	0,00	1,85
ED – Método 1 + LKH	27686	100	27686	100	27703	0,99	2,40	1,18
ED – Método 2 + LKH	27686	100	27686	100	27706	0,99	2,82	1,26
ED – Método 3 + LKH	27686	100	27686	100	27686	100	0,00	1,04
ED – Método 4 + LKH	27686	100	27686	100	27706	0,99	2,82	0,90
ED – Método 5 + LKH	27686	100	27686	100	27703	0,99	2,40	1,02
ED – Método 6 + LKH	27686	100	27686	100	27686	100	0,00	1,14
LKH	27686	100	27688	0,99	27706	0,99	5,71	1,38

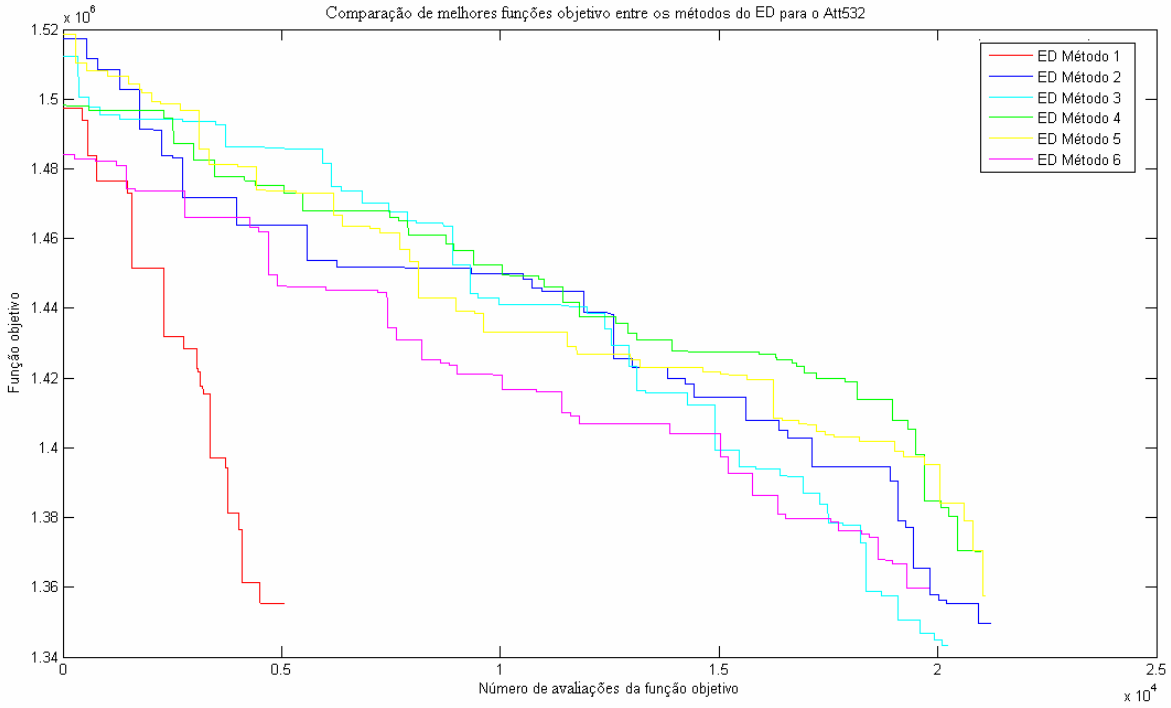

Figura 20: Comparação de melhores funções objetivo entre os métodos do ED para o ATT532

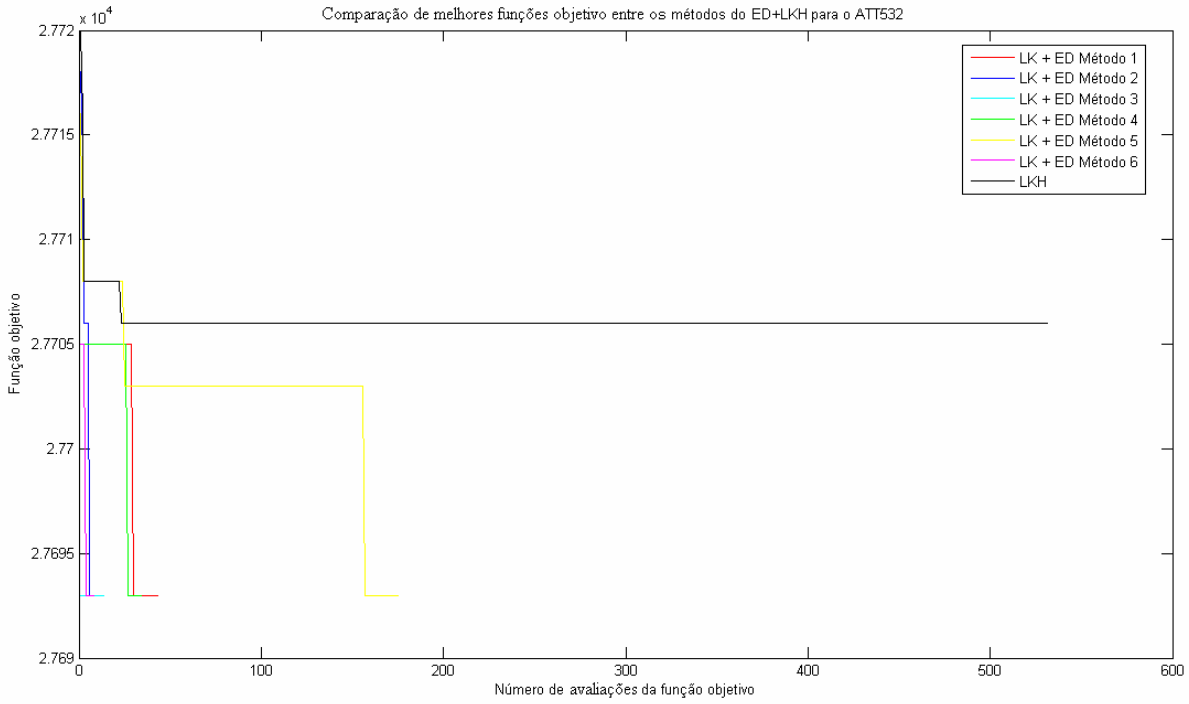

Figura 21: Comparação de melhores funções objetivo entre os métodos do LKH+ED para o ATT532

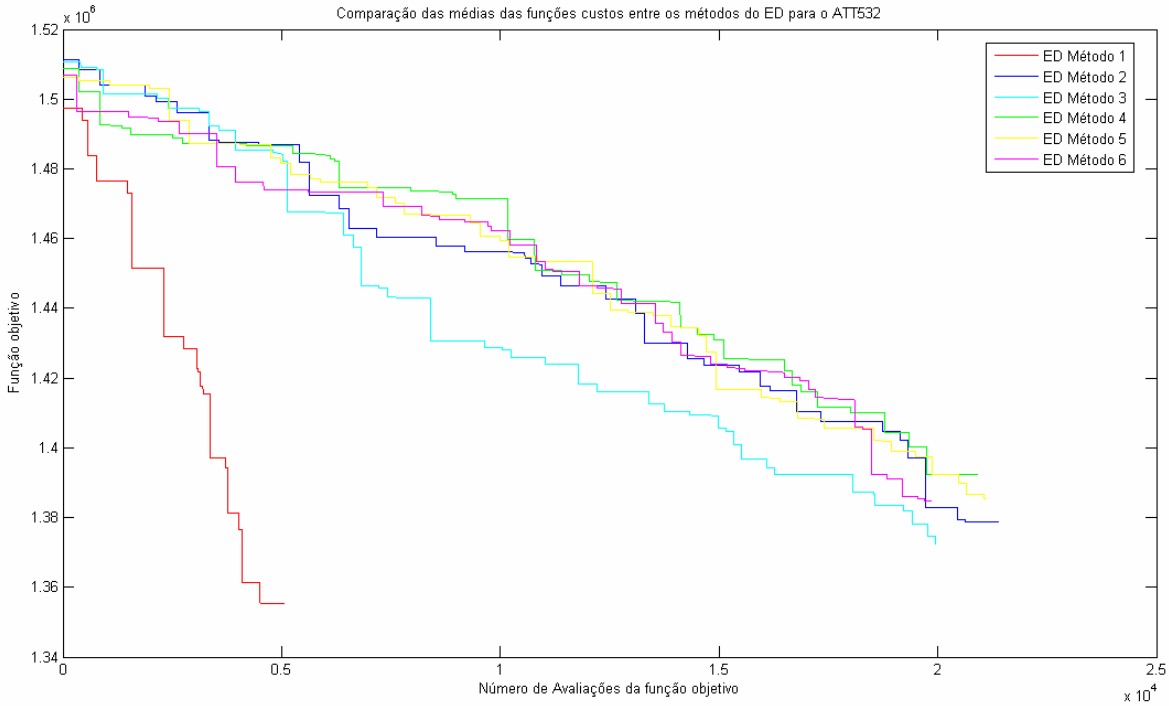

Figura 22: Comparação das médias das funções objetivo entre os métodos do ED para o ATT532

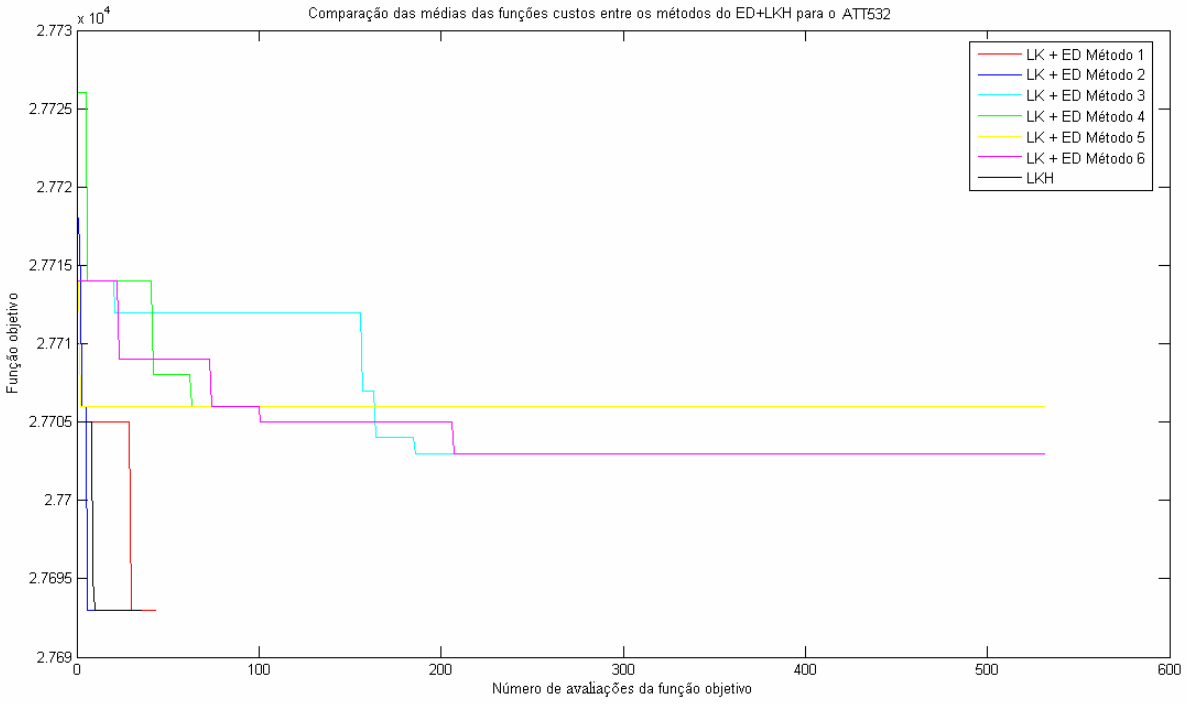

Figura 23: Comparação dos piores funções objetivo entre os métodos do LKH+ED para o ATT532

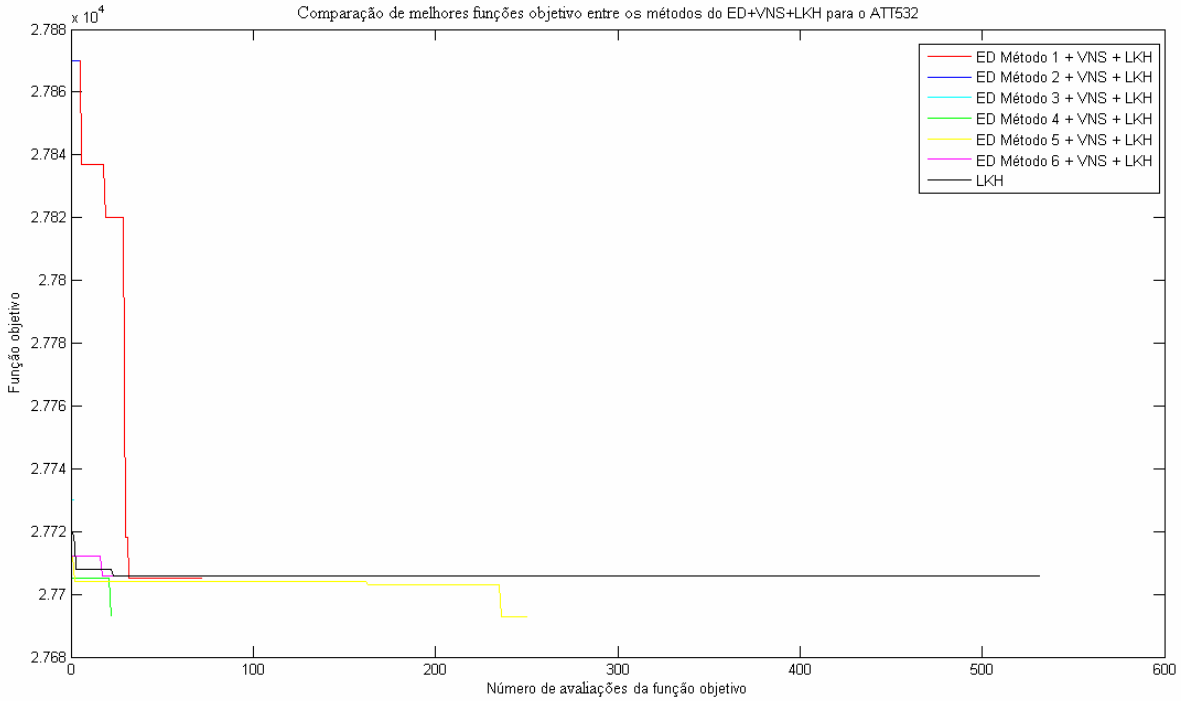

Figura 24: Comparação das melhores funções objetivo entre os métodos do ED+VNS+LKH para o ATT532

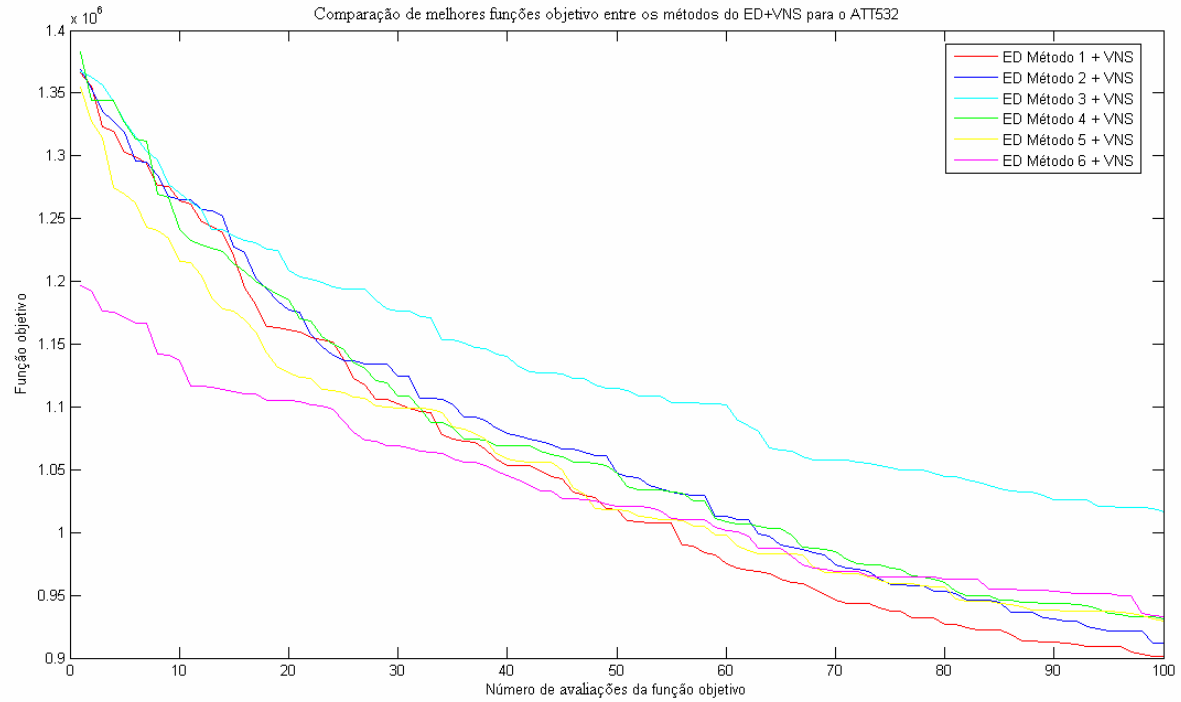

Figura 25: Comparação das melhores funções objetivo entre os métodos do ED+VNS para o ATT532

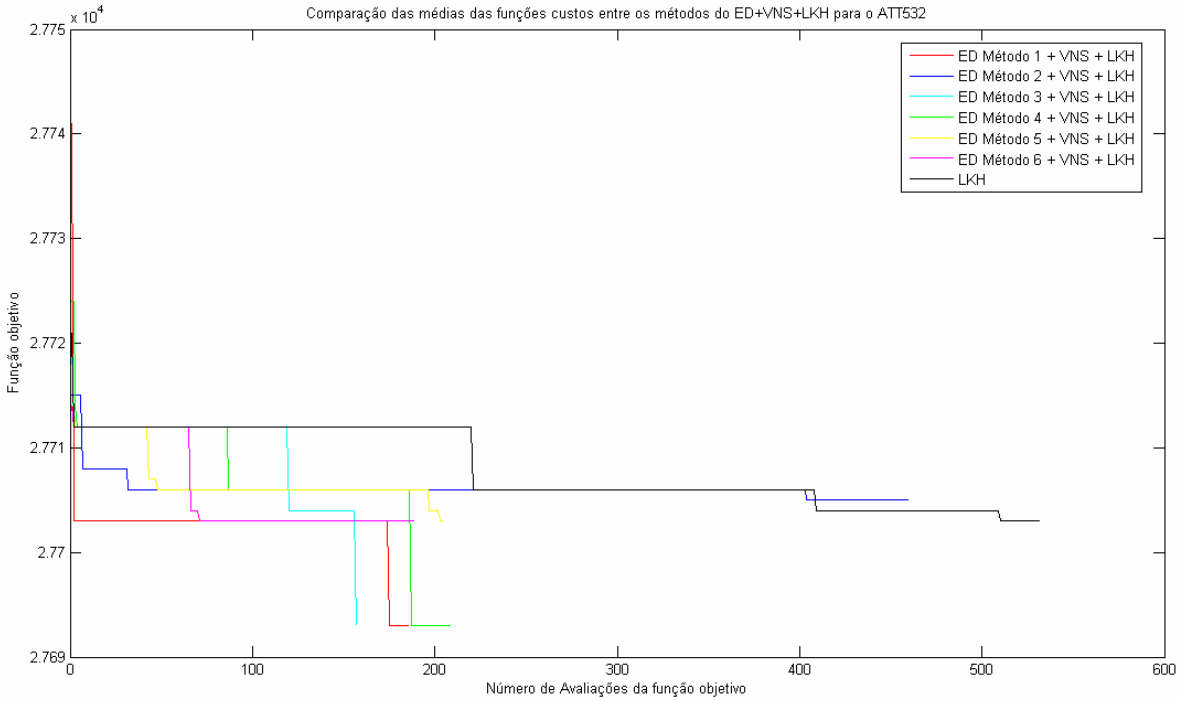

Figura 26: Comparação das médias das funções objetivo entre os métodos do ED+VNS+LKH para o ATT532

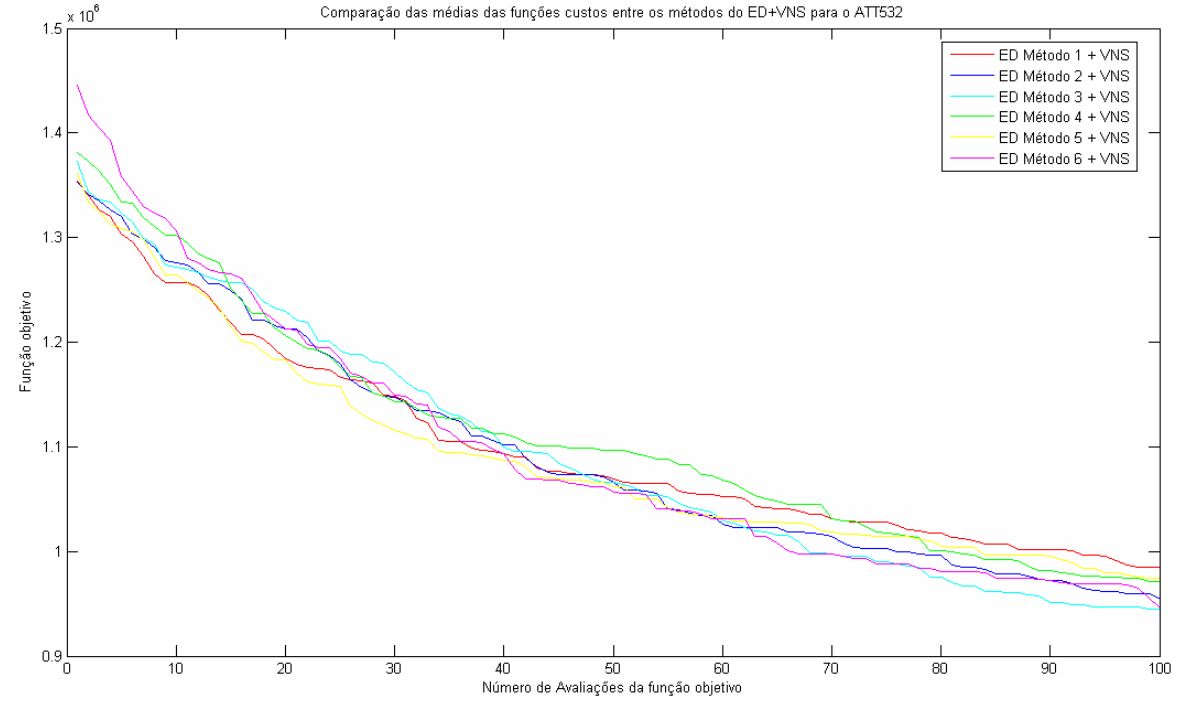

Figura 27: Comparação das piores funções objetivo entre os métodos do ED+VNS para o ATT532

Para o problema U2152, apenas um método conseguiu ser melhor do que o método do LKH com entrada aleatória. O ED-método 1 conseguiu obter na média de *tours* um resultado melhor e também na média de tempo de execução o tempo médio foi melhor. Vale salientar que neste problema, muitos métodos acabaram mostrando-se piores do que o próprio LKH, isto significa que estes métodos não passaram um *tour* inicial promissor. Comparando-se novamente os resultados obtidos com [DUA06], novamente o método híbrido ED+VNS+LKH conseguiu obter um resultado melhor do que o encontrado pelo outro método. Com um diferencial de se obter o resultado esperado em um tempo médio 65% mais rápido. Porém, em [TSA04] obteve-se o mesmo resultado, mas com uma taxa de convergência de apenas 40% maior que a encontrada em [DUA06].

Tabela 7: Resultados para o problema U2152 – Resultado ótimo: 64253

Método	Função Objetivo						Desvio Padrão	Tempo(s)
	Mínimo (%)		Média (%)		Máximo (%)			
ED – Método 1	2358326	02,72	2387348	02,69	2400930	02,67	9857,27	39,15
ED – Método 2	2374736	02,70	2400422	02,67	2418667	02,65	11064,33	40,95
ED – Método 3	2369315	02,71	2392153	02,68	2407415	02,66	8699,28	40,21
ED – Método 4	2363261	02,71	2399454	02,67	2417874	02,65	10395,11	39,42
ED – Método 5	2379610	02,70	2398336	02,67	2414187	02,66	7978,91	41,56
ED – Método 6	2383756	02,69	2415900	02,65	2431496	02,66	9573,80	40,65
ED – Método 1 + VNS	1865125	03,44	1909209	03,33	1954569	03,28	17958,15	11,33
ED – Método 2 + VNS	1869203	03,43	1911566	03,36	1952110	03,29	18376,85	2,83
ED – Método 3 + VNS	1879682	03,41	1914603	03,35	1962538	03,27	18933,86	2,92
ED – Método 4 + VNS	1890494	03,39	1909548	03,35	1930968	03,32	17824,58	4,04
ED – Método 5 + VNS	1871610	03,43	1910308	03,36	1940000	03,31	14767,75	4,47
ED – Método 6 + VNS	1886443	03,40	1912352	03,35	1939220	03,31	14440,37	8,40
ED – Método 1 + VNS + LKH	64253	100	64281	0,99	64337	0,99	23,15	237,38
ED – Método 2 + VNS + LKH	64253	100	64281	0,99	64337	0,99	24,55	59,62
ED – Método 3 + VNS + LKH	64253	100	64278	0,99	64337	0,99	25,43	61,30
ED – Método 4 + VNS + LKH	64253	100	64279	0,99	64297	0,99	25,32	91,28
ED – Método 5 + VNS + LKH	64253	100	64282	0,99	64344	0,99	27,47	89,72
ED – Método 6 + VNS + LKH	64253	100	64271	0,99	64310	0,99	20,66	164,64
ED – Método 1 + LKH	64253	100	64272	0,99	64324	0,99	22,60	83,08
ED – Método 2 + LKH	64253	100	64287	0,99	64320	0,99	20,79	94,90
ED – Método 3 + LKH	64253	100	64272	0,99	64324	0,99	22,85	85,81
ED – Método 4 + LKH	64253	100	64278	0,99	64320	0,99	23,50	86,06
ED – Método 5 + LKH	64253	100	64282	0,99	64337	0,99	23,20	93,44
ED – Método 6 + LKH	64253	100	64285	0,99	64337	0,99	24,05	89,06
LKH	64253	100	64278	0,99	64324	0,99	23,57	86,58

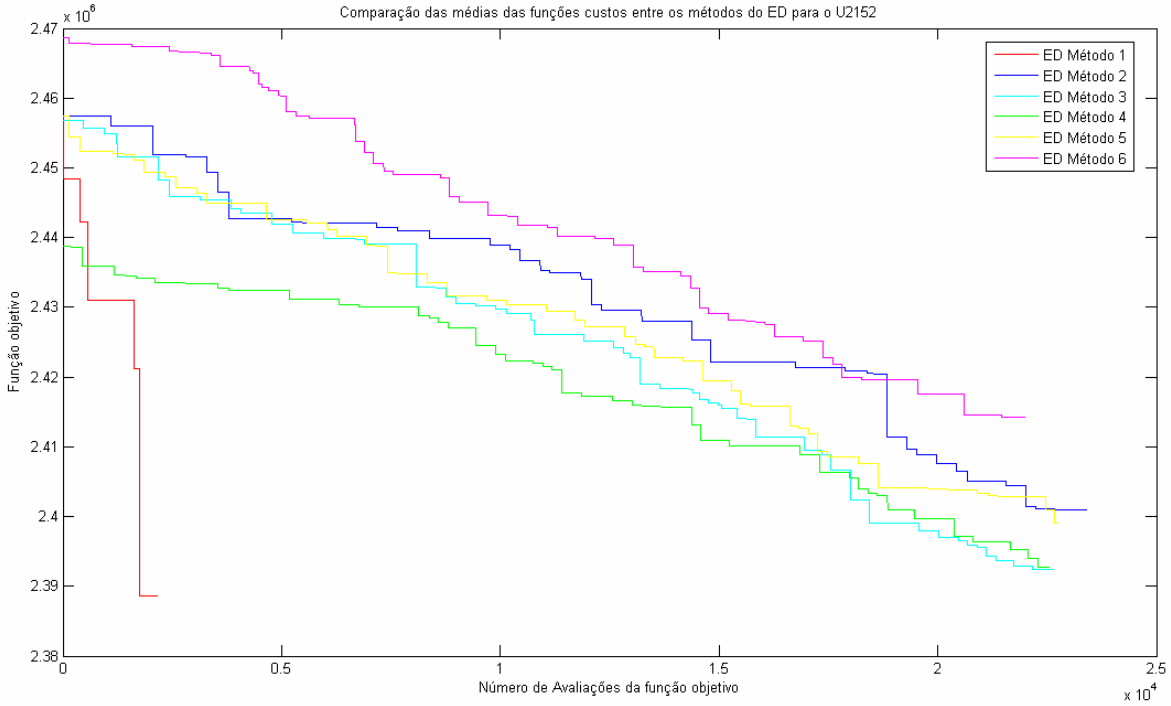

Figura 28: Comparação de melhores funções objetivo entre os métodos do ED para o U2152

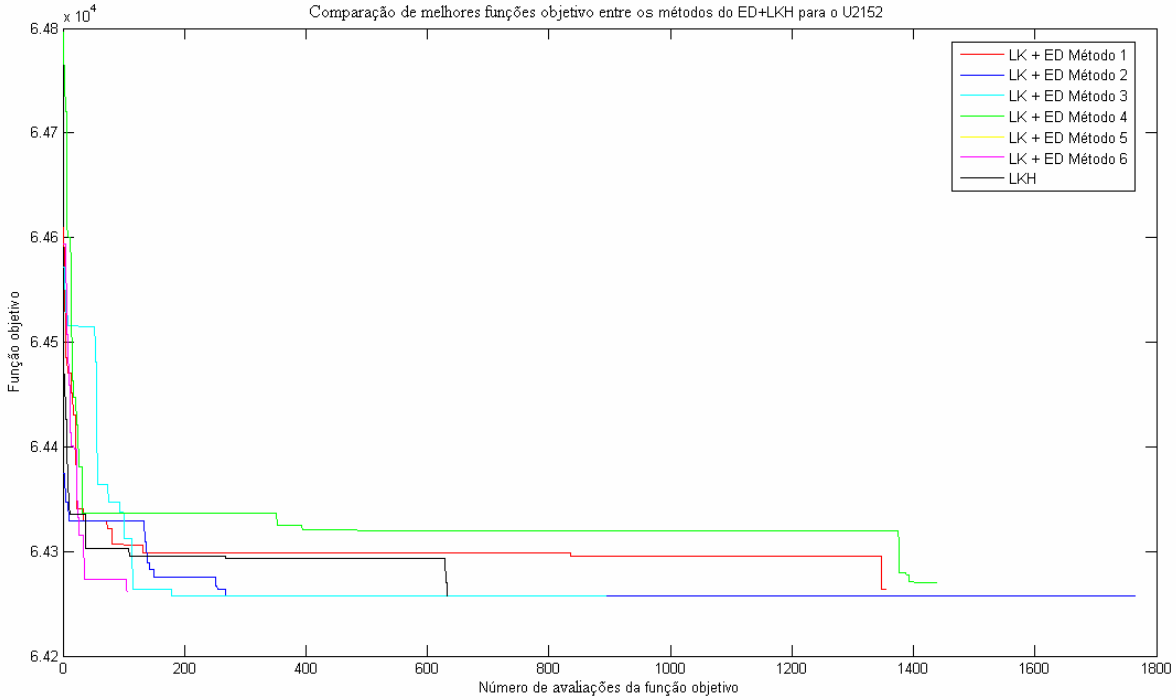

Figura 29: Comparação de melhores funções objetivo entre os métodos do LKH+ED para o U2152

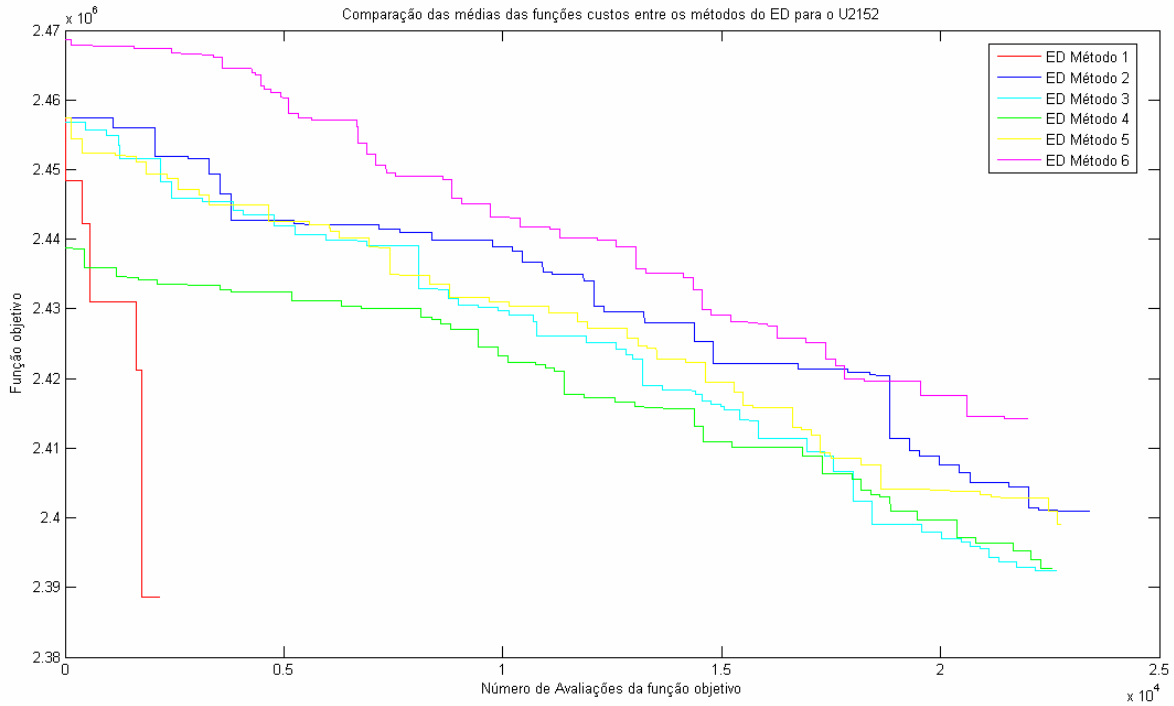

Figura 30: Comparação das médias das funções objetivo entre os métodos do ED para o U2152

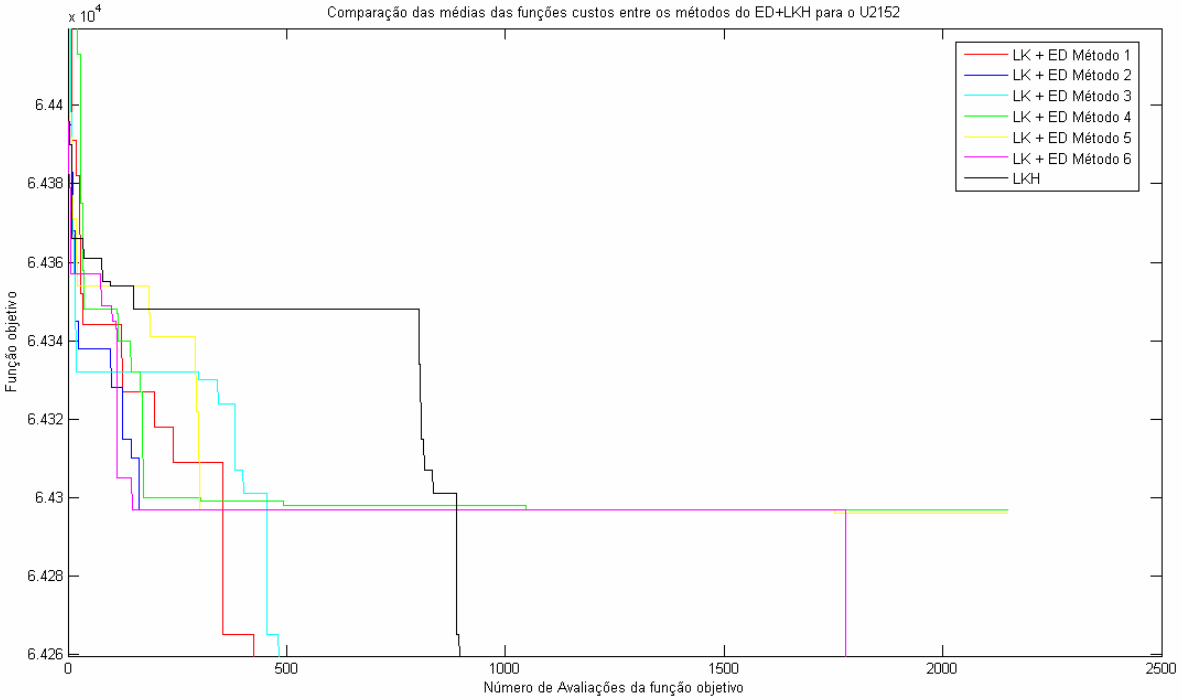

Figura 31: Comparação dos piores funções objetivo entre os métodos do LKH+ED para o U2152

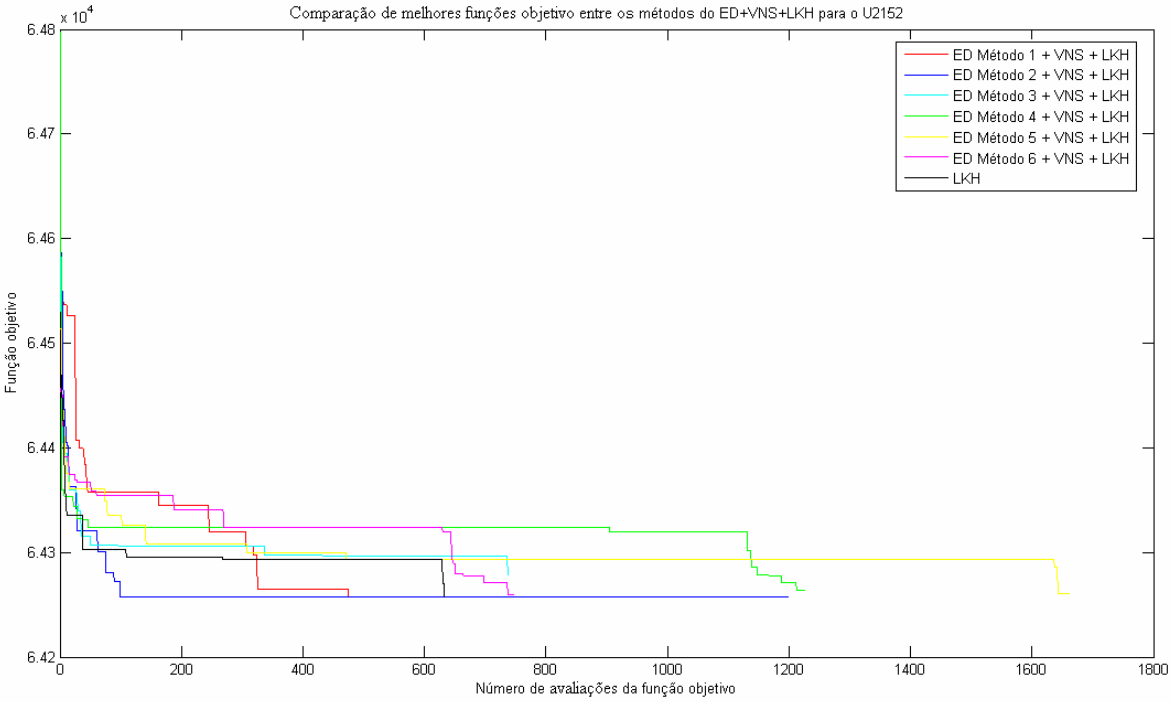

Figura 32: Comparação das melhores funções objetivo entre os métodos do ED+VNS+LKH para o U2152

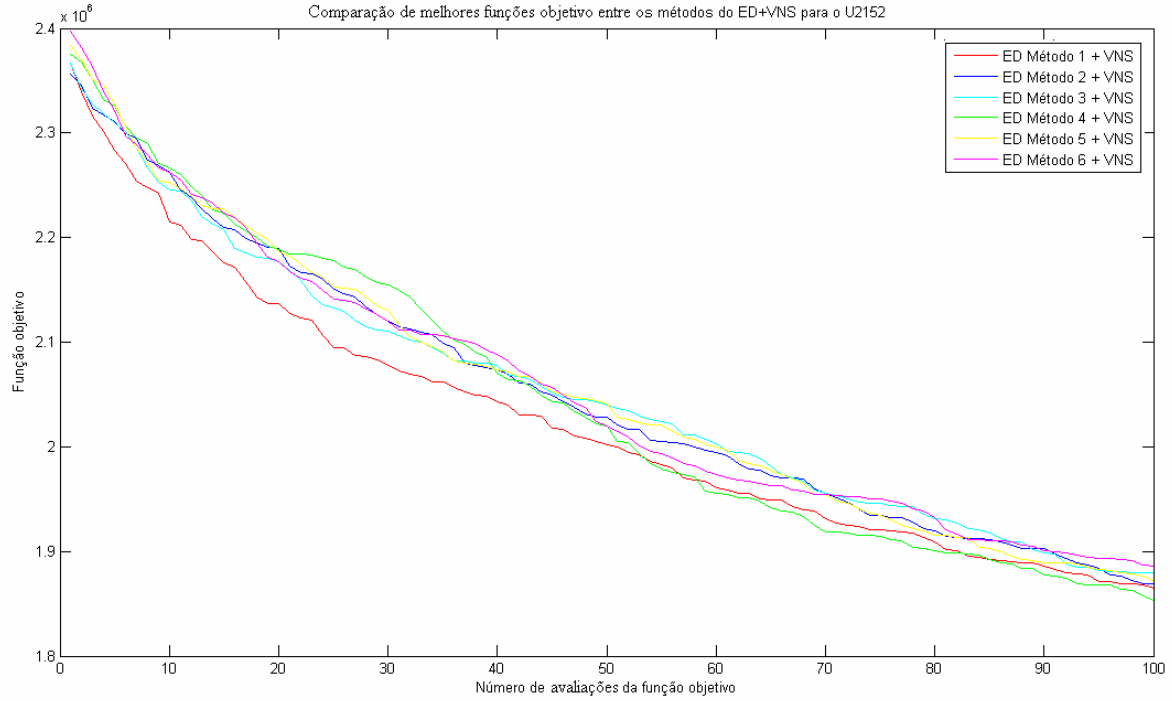

Figura 33: Comparação das melhores funções objetivo entre os métodos do ED+VNS para o U2152

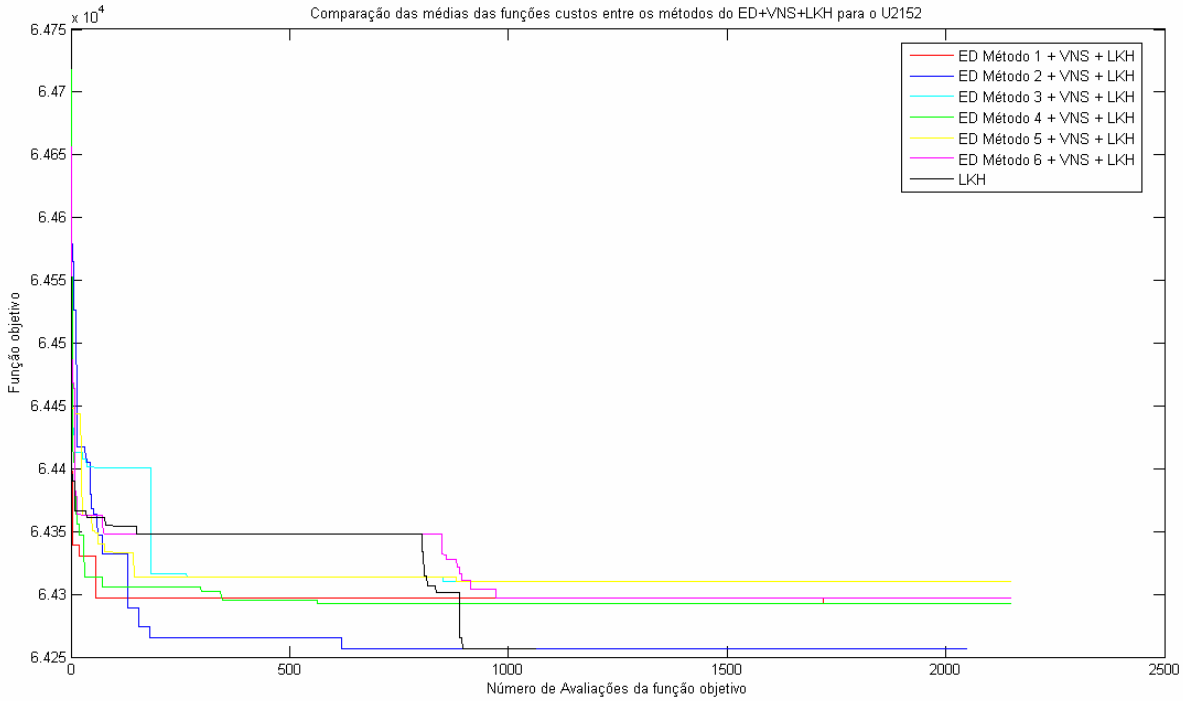

Figura 34: Comparação das médias das funções objetivo entre os métodos do ED+VNS+LKH para o U2152

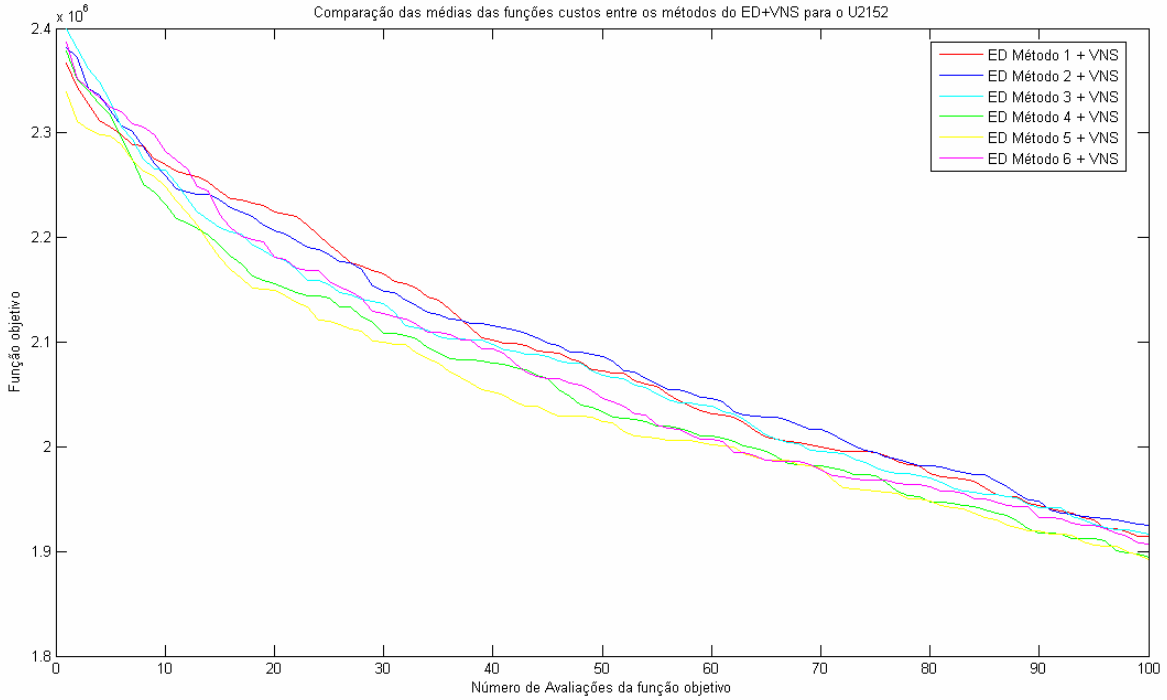

Figura 35: Comparação das piores funções objetivo entre os métodos do ED+VNS para o U2152

Nos problemas de grandes quantidade de cidades, como no caso do PCB3038, pode-se notar que o método 1 conseguiu um ganho de velocidade de aproximadamente 15%. Mas este método não foi o que obteve a melhor média, pois novamente o método 3 apresentou uma média superior em relação a todos os outros métodos. Comparando-se o resultado obtido com [TSA04] , temos que no caso do ED+VNS+LKH não conseguiu-se obter uma média geral com resultado ótimo, já usando os algoritmos evolucionários, conseguimos obter tal performance. Porém, a média de tempo para obter tal resultado é de aproximadamente 600 segundos, quase o dobro do que se obteve com o método aqui proposto. Já em [DUA06], novamente temos um resultado aproximadamente 5% melhor e com um tempo de processamento de quase 1000% maior.

Tabela 8: Resultados para o problema PCB3038 – Resultado ótimo: 137694

Método	Função Objetivo						Desvio Padrão	Tempo(s)
	Mínimo/(%)		Média/(%)		Máximo/(%)			
ED – Método 1	5133870	02,68	5170664	02,66	5202107	02,64	16979,24	59,10
ED – Método 2	5154884	02,67	5199352	02,64	5223287	02,63	15276,77	61,67
ED – Método 3	5139092	02,67	5182142	02,65	5206576	02,64	16300,69	59,77
ED – Método 4	5138957	02,67	5195580	02,65	5222748	02,63	17421,35	59,03
ED – Método 5	5155603	02,67	5192121	02,65	5218324	02,63	13496,61	60,90
ED – Método 6	5175461	02,66	5213000	02,64	5233804	02,63	12913,58	59,87
ED – Método 1 + VNS	4251942	03,23	4302541	03,20	4356353	03,16	26434,48	17,06
ED – Método 2 + VNS	4225168	03,25	4297696	03,20	4380753	03,14	31020,83	4,04
ED – Método 3 + VNS	4242237	03,24	4297930	03,20	4367477	03,15	32827,17	4,15
ED – Método 4 + VNS	4242011	03,24	4300797	03,20	4358760	03,15	25975,14	6,35
ED – Método 5 + VNS	4251926	03,23	4301107	03,20	4364110	03,15	26206,70	6,41
ED – Método 6 + VNS	4242011	03,24	4293957	03,20	4358760	03,15	30215,83	12,77
ED – Método 1 + VNS + LKH	137694	100	137704	0,99	137753	0,99	20,78	369,34
ED – Método 2 + VNS + LKH	137694	100	137704	0,99	137753	0,99	20,81	86,3
ED – Método 3 + VNS + LKH	137694	100	137704	0,99	137753	0,99	19,89	96,92
ED – Método 4 + VNS + LKH	137694	100	137707	0,99	137753	0,99	23,69	156,60
ED – Método 5 + VNS + LKH	137694	100	137704	0,99	137753	0,99	19,89	130,72
ED – Método 6 + VNS + LKH	137694	100	137701	0,99	137753	0,99	18,48	233,08
ED – Método 1 + LKH	137694	100	137706	0,99	137753	0,99	23,34	123,10
ED – Método 2 + LKH	137694	100	137705	0,99	137753	0,99	21,72	135,24
ED – Método 3 + LKH	137694	100	137704	0,99	137760	0,99	21,70	128,60
ED – Método 4 + LKH	137694	100	137708	0,99	137760	0,99	23,12	140,08
ED – Método 5 + LKH	137694	100	137705	0,99	137753	0,99	21,40	157,80
ED – Método 6 + LKH	137694	100	137710	0,99	137760	0,99	21,80	136,34
LKH	137694	100	137710	0,99	137757	0,99	22,55	142,64

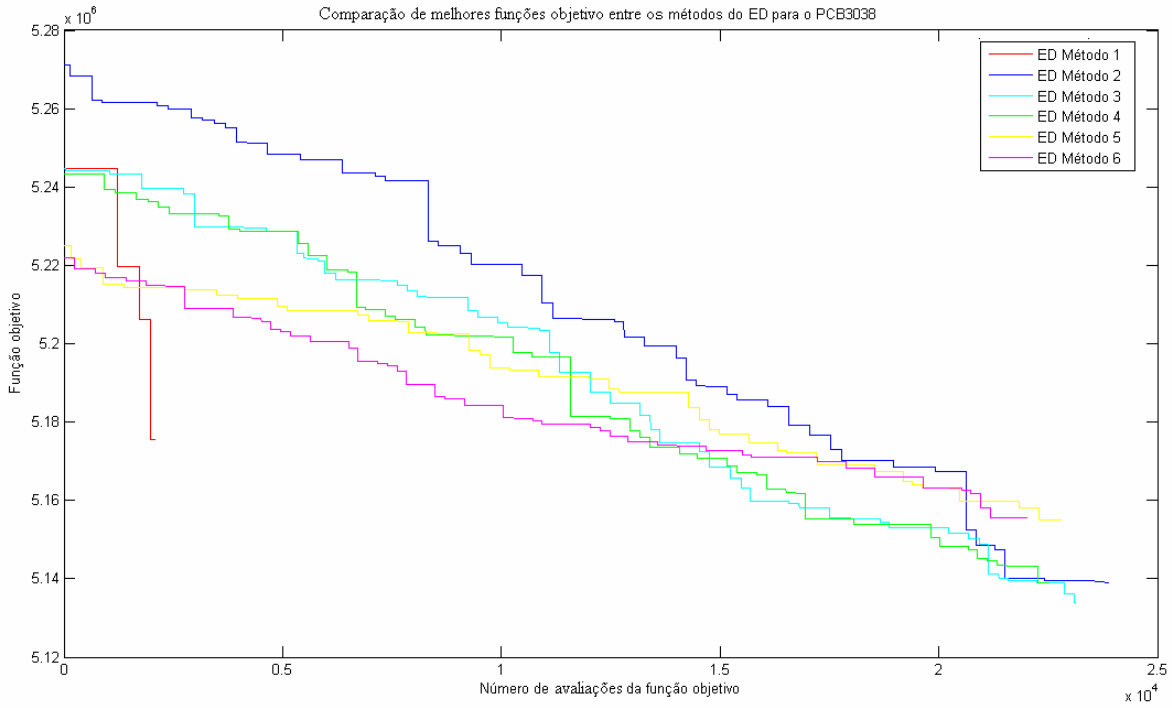

Figura 36: Comparação de melhores funções objetivo entre os métodos do ED para o PCB3038

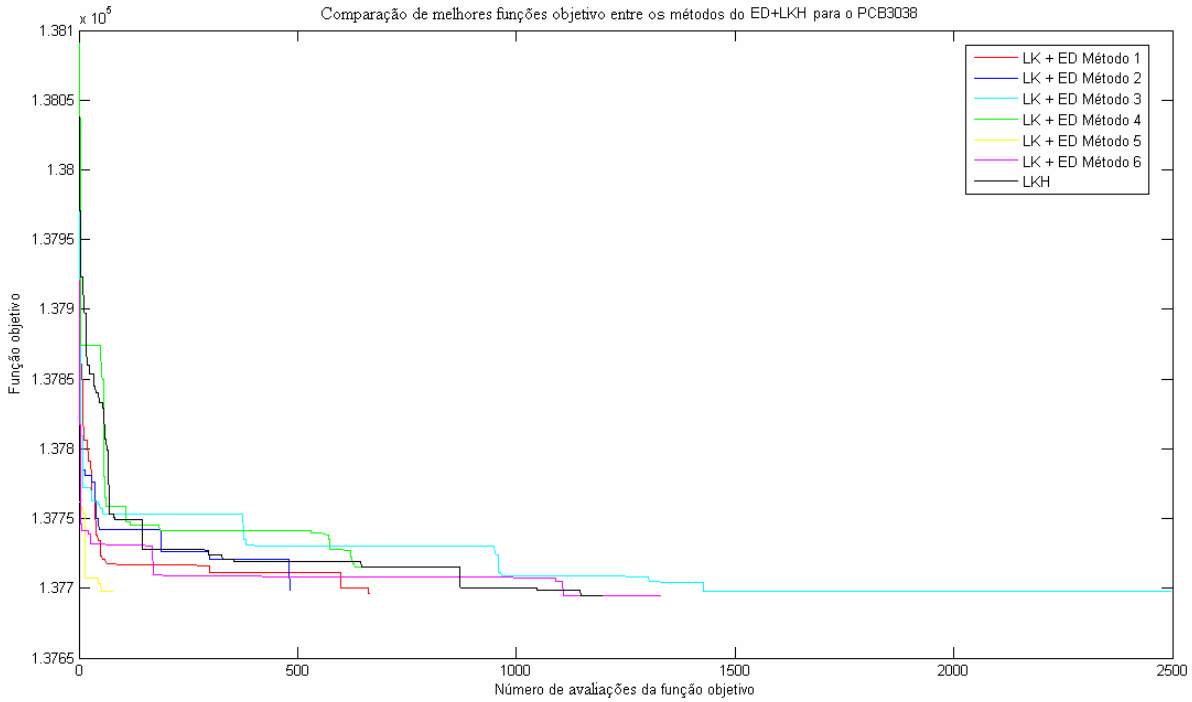

Figura 37: Comparação de melhores funções objetivo entre os métodos do LKH+ED para o PCB3038

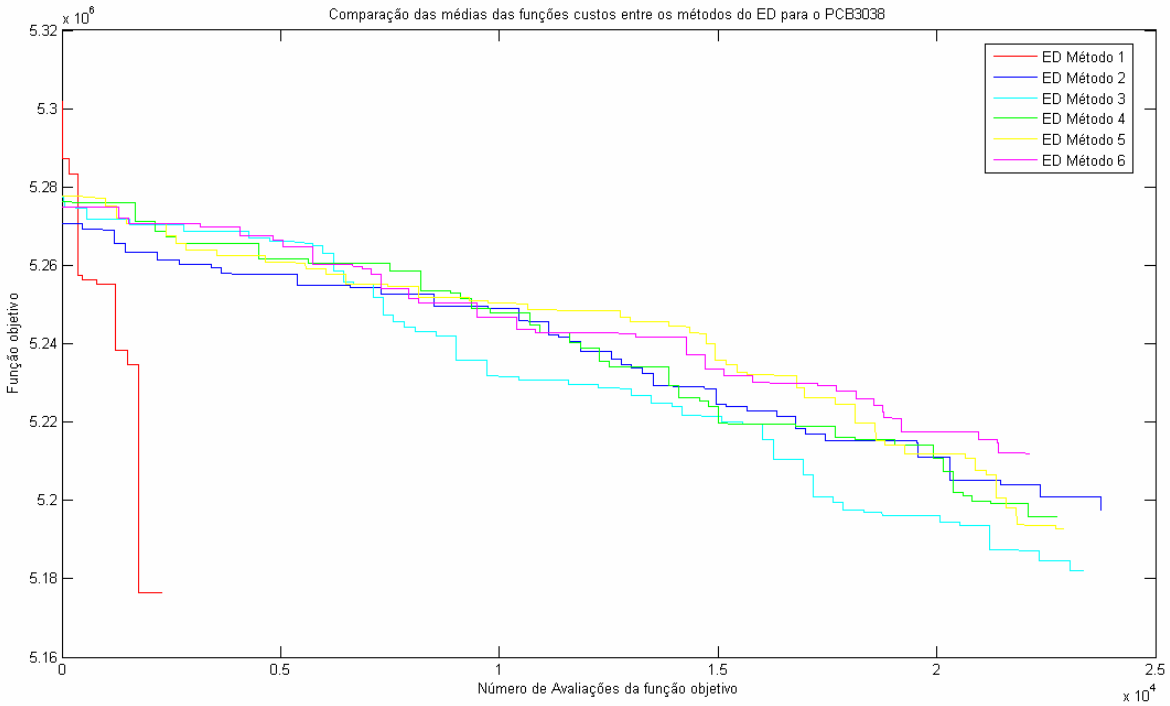

Figura 38: Comparação das médias das funções objetivo entre os métodos do ED para o PCB3038

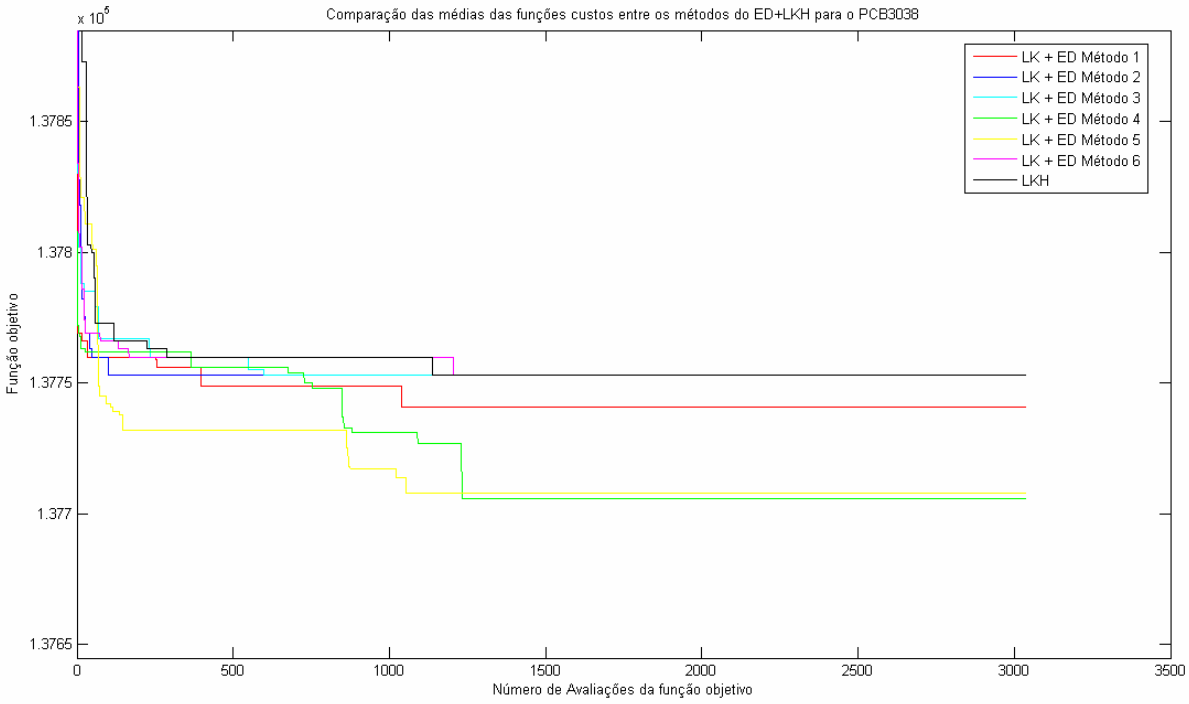

Figura 39: Comparação das médias das funções objetivo entre os métodos do LKH+ED para o PCB3038

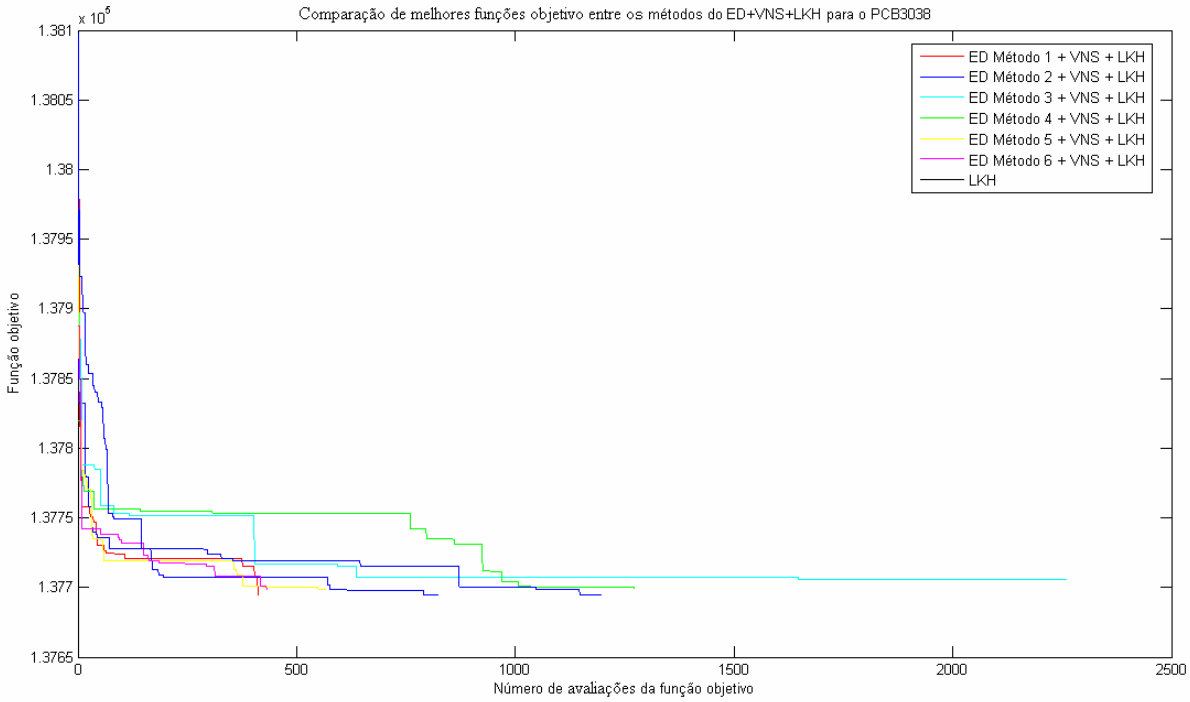

Figura 40: Comparação das melhores funções objetivo entre os métodos do ED+VNS+LKH para o PCB3038

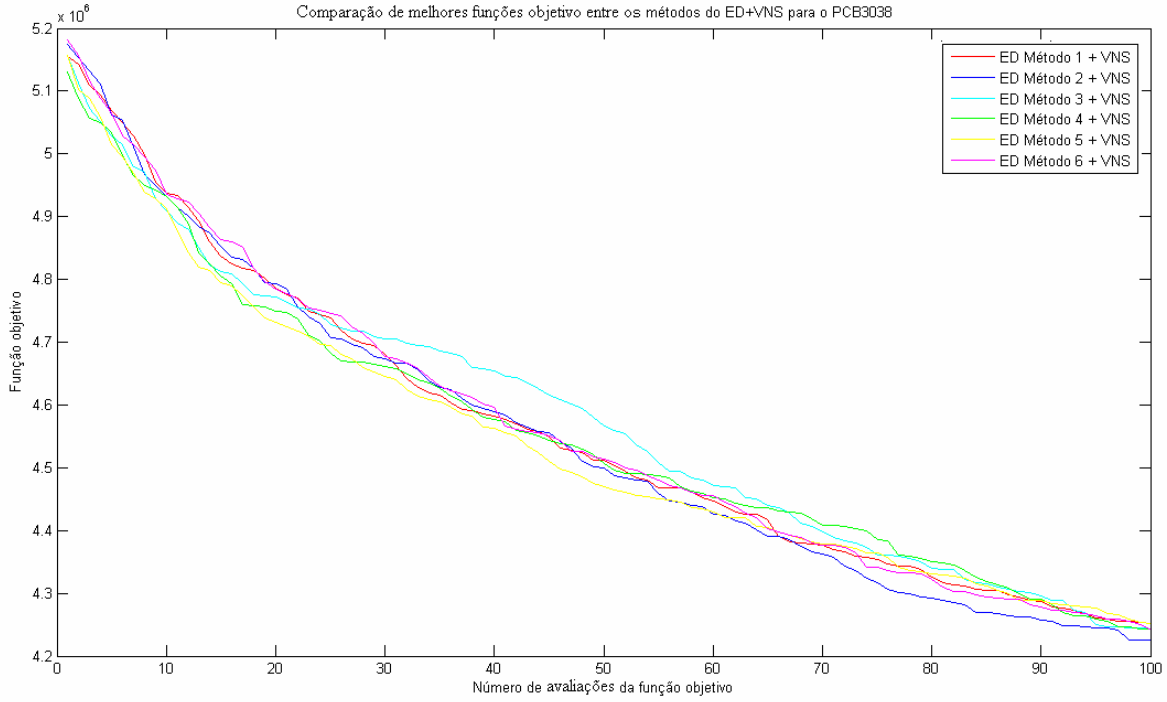

Figura 41: Comparação das melhores funções objetivo entre os métodos do ED+VNS para o PCB3038

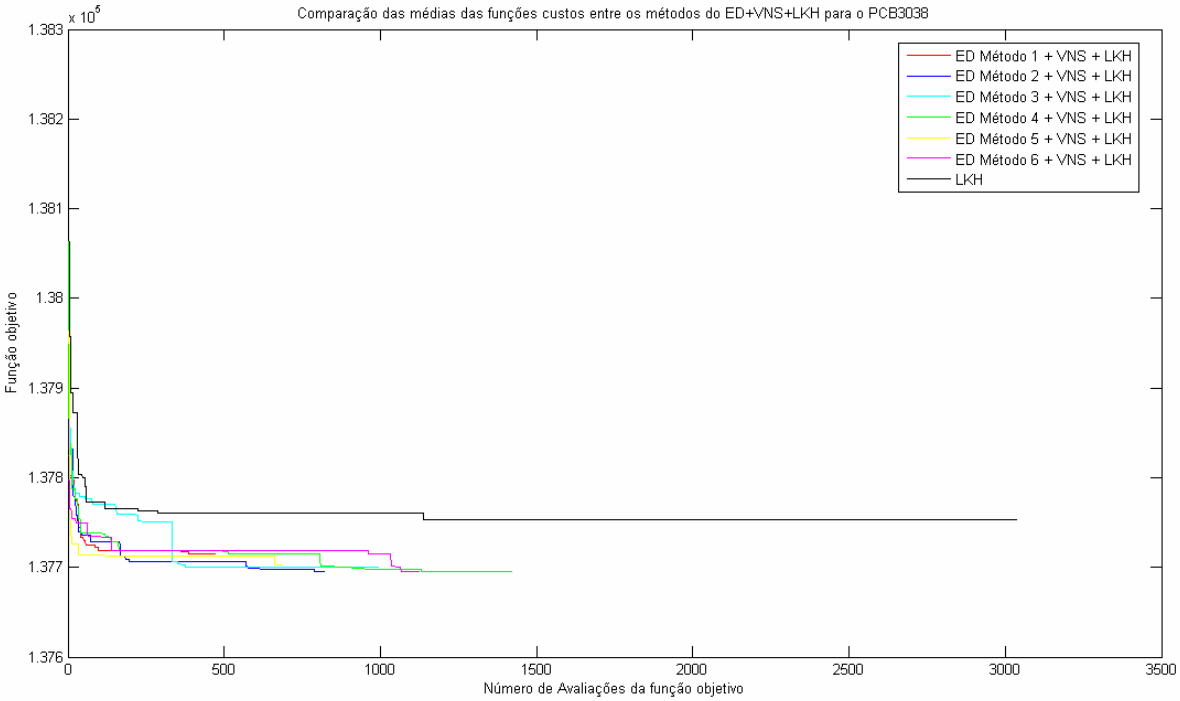

Figura 42: Comparação das médias das funções objetivo entre os métodos do ED+VNS+LKH para o PCB3038

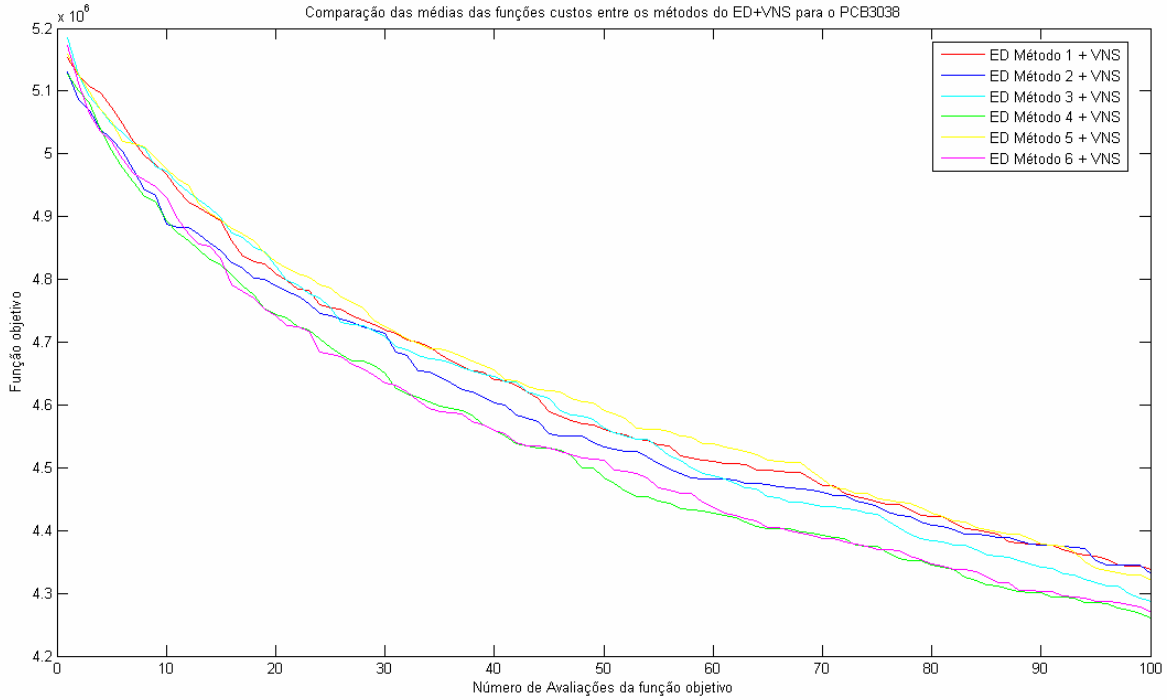

Figura 43: Comparação das piores funções objetivo entre os métodos do ED+VNS para o PCB3038

Por fim, no PLA7397, pode-se notar que o tempo de execução do ED+LKH demonstrou ter uma função objetivo e uma taxa de convergência melhor do que o LKH com entrada aleatória. Todos os métodos fizeram com que o problema fosse resolvido em aproximadamente 15% mais rápido. Mas nota-se também que os *tours* obtidos na média foram piores que o LKH com entrada aleatória. Porém, no desvio padrão nota-se que realmente houve uma melhora da qualidade dos *tours* gerados, havendo apenas 2 métodos que não obtiveram tal melhora. Em [NGU07], temos um método que usa algoritmos genéticos que consegue obter uma média melhor do que o método ED+VNS+LKH. Porém, o que realmente mostra a capacidade deste novo método é o tempo, pois ele consegue executar quase que 400% mais rápido do que o método proposto. Mas neste caso, o tempo rápido acarretou em uma função custo maior do que a ótima, pois o método só obteve uma resposta ótima em um método que demorou aproximadamente 3000 segundos, o que acaba fazendo com ele fique 50% mais rápido do que o método apresentado.

Tabela 9: Resultados para o problema PLA7397 – Resultado ótimo: 23260728

Método	Função Objetivo						Desvio Padrão	Tempo(s)
	Mínimo (%)		Média (%)		Máximo (%)			
ED – Método 1	2693563800	0,86	2710405086	0,85	2723186572	0,85	6252729,91	193,23
ED – Método 2	2706759730	0,85	2719171271	0,85	2730232783	0,85	5568919,45	197,26
ED – Método 3	2691174593	0,86	2712482460	0,85	2722793128	0,85	7238381,75	189,83
ED – Método 4	2705170509	0,85	2716715530	0,85	2727591661	0,85	5606255,50	191,81
ED – Método 5	2707787934	0,85	2717029035	0,85	2724217850	0,85	5672061,92	197,06
ED – Método 6	2701923188	0,86	2713618645	0,85	2721716054	0,85	4675237,14	192,03
ED – Método 1 + VNS	2282677680	1,01	2292893726	1,01	2305226654	1,00	7921297,27	131,94
ED – Método 2 + VNS	2269501429	1,02	2292552029	1,01	2318960463	1,00	11845233,31	09,92
ED – Método 3 + VNS	2266112209	1,02	2290785431	1,01	2312022575	1,00	12685775,23	10,21
ED – Método 4 + VNS	2266162503	1,02	2292755791	1,01	2312022985	1,00	11887071,57	13,63
ED – Método 5 + VNS	2263346968	1,02	2294155912	1,01	2319104083	1,00	11938846,14	22,57
ED – Método 6 + VNS	2272165485	1,02	2292705087	1,01	2316343362	1,00	9487591,19	14,45
ED – Método 1 + VNS + LKH	23260728	100	23261170	0,99	23264711	0,99	1327,66	13636,55
ED – Método 2 + VNS + LKH	23260728	100	23261301	0,99	23264711	0,99	1191,31	4229,96
ED – Método 3 + VNS + LKH	23260728	100	23261452	0,99	23264747	0,99	1289,89	4476,02
ED – Método 4 + VNS + LKH	23260728	100	23261287	0,99	23264798	0,99	1270,12	4709,73
ED – Método 5 + VNS + LKH	23260728	100	23261395	0,99	23264911	0,99	1276,74	7412,34
ED – Método 6 + VNS + LKH	23260728	100	23261263	0,99	23263960	0,99	1076,37	6021,17
ED – Método 1 + LKH	23260728	100	23261533	0,99	23265152	0,99	1346,24	6771,52
ED – Método 2 + LKH	23260728	100	23261590	0,99	23264911	0,99	1420,34	7954,14
ED – Método 3 + LKH	23260728	100	23261383	0,99	23264196	0,99	1236,25	6276,08
ED – Método 4 + LKH	23260728	100	23261253	0,99	23264074	0,99	1089,09	6542,52
ED – Método 5 + LKH	23260728	100	23261115	0,99	23263874	0,99	955,36	6484,14
ED – Método 6 + LKH	23260728	100	23261458	0,99	23263960	0,99	1235,85	7004,67
LKH	23260728	100	23261000	0,99	23264711	0,99	1294,50	7441,50

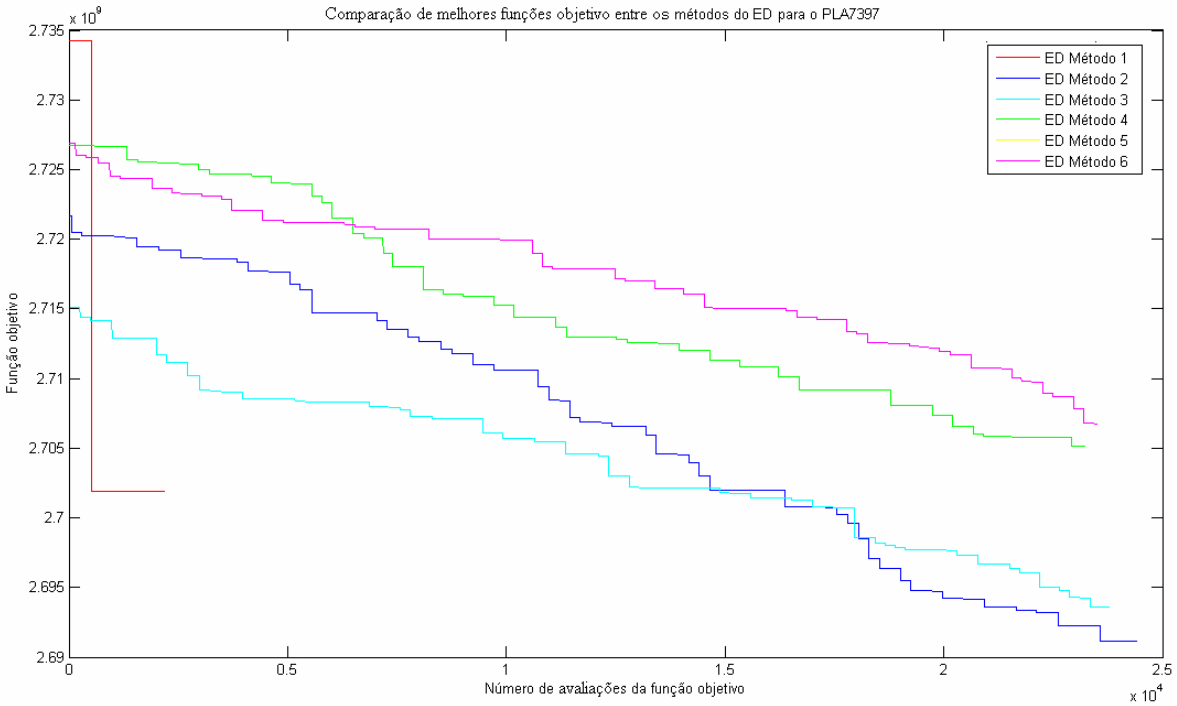

Figura 44: Comparação de melhores funções objetivo entre os métodos do ED para o PLA7397

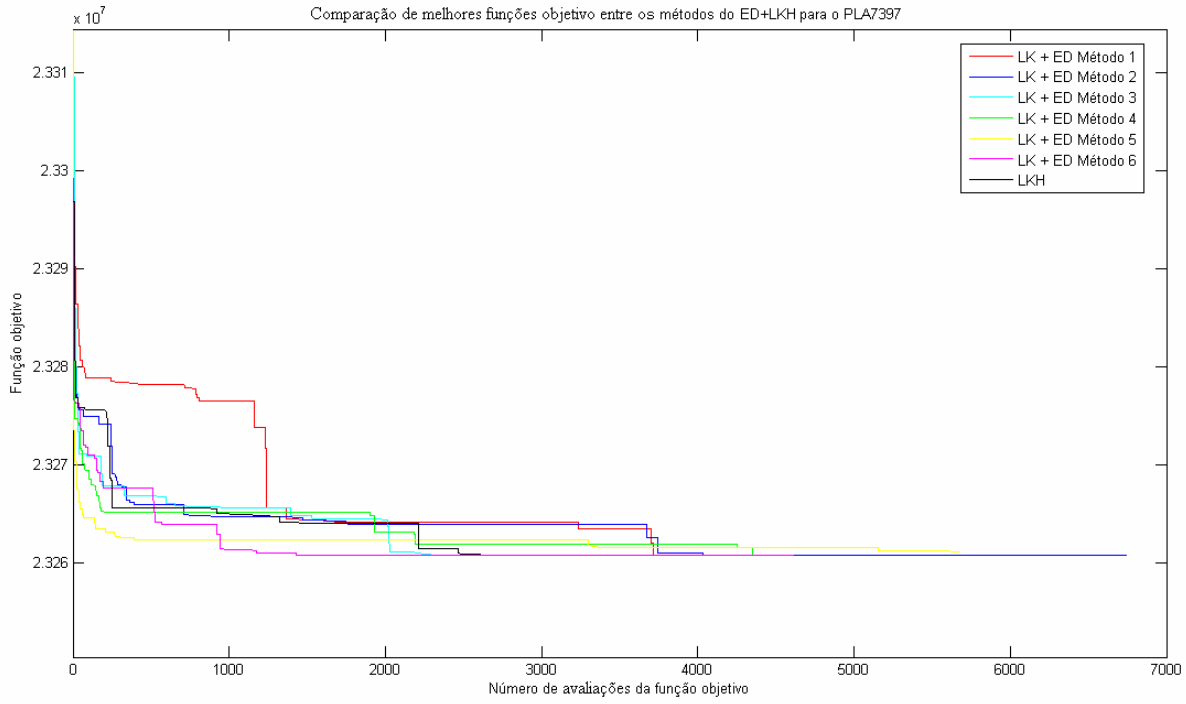

Figura 45: Comparação de melhores funções objetivo entre os métodos do LKH+ED para o PLA7397

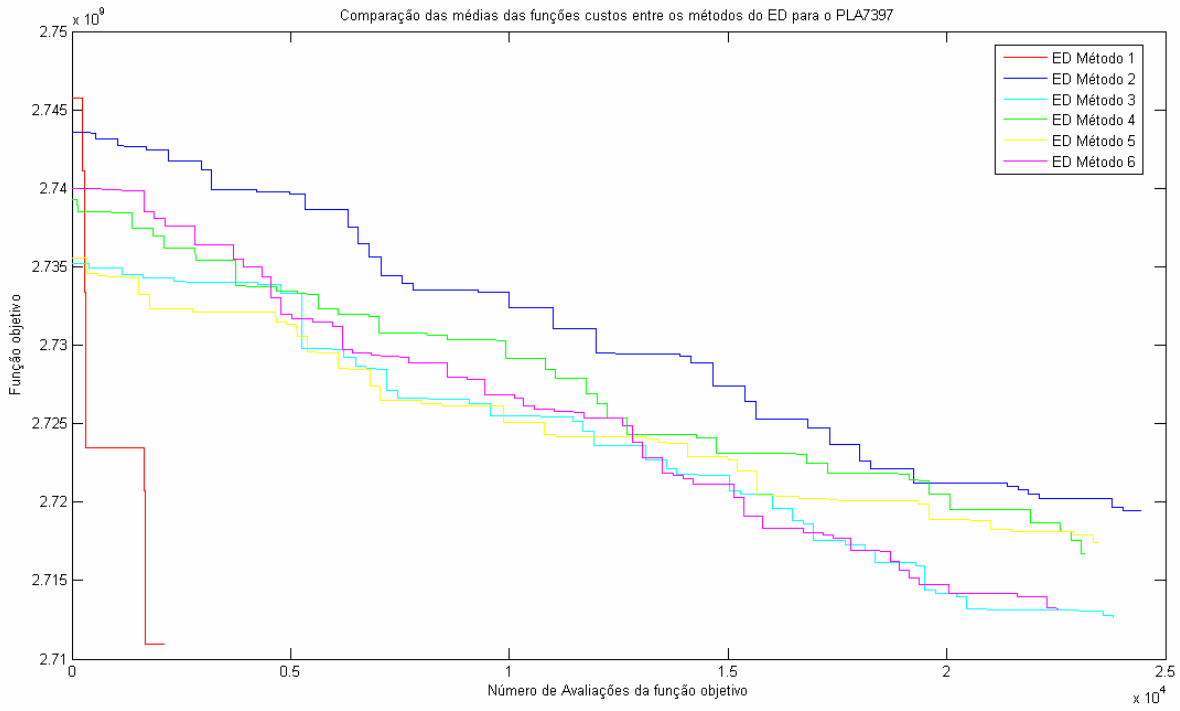

Figura 46: Comparação das médias das funções objetivo entre os métodos do ED para o PLA7397

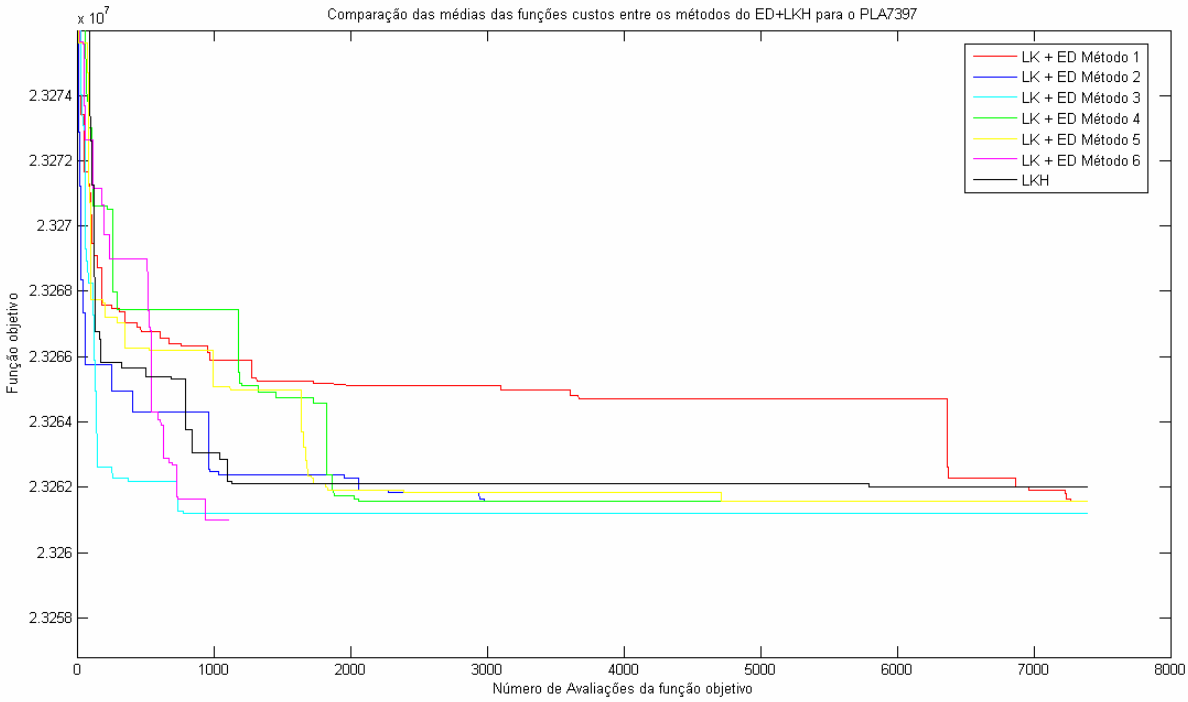

Figura 47: Comparação das médias das funções objetivo entre os métodos do LKH+ED para o PLA7397

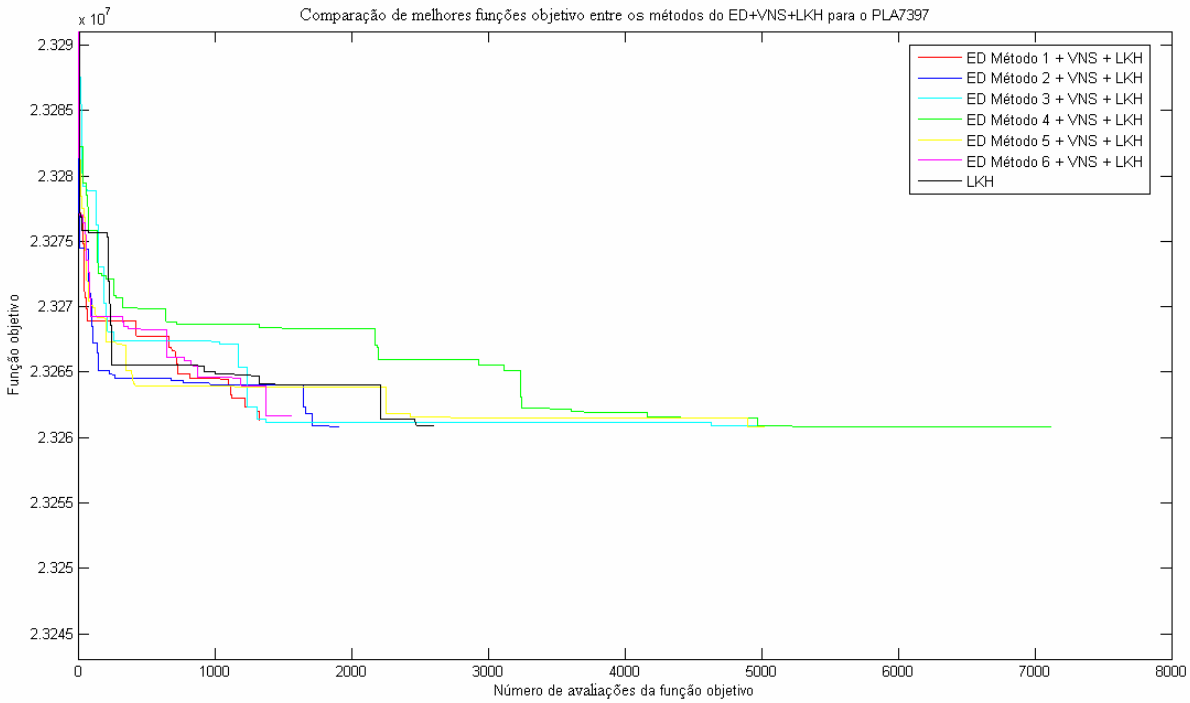

Figura 48: Comparação das melhores funções objetivo entre os métodos do ED+VNS+LKH para o PLA7397

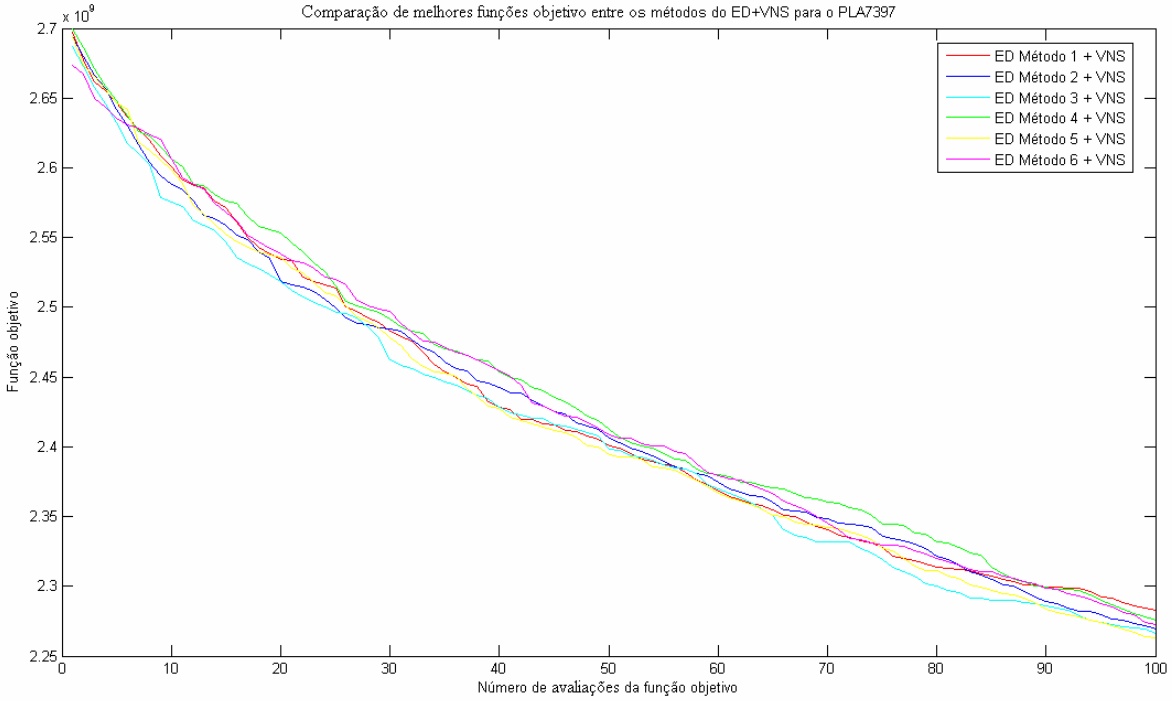

Figura 49: Comparação das melhores funções objetivo entre os métodos do ED+VNS para o PLA7397

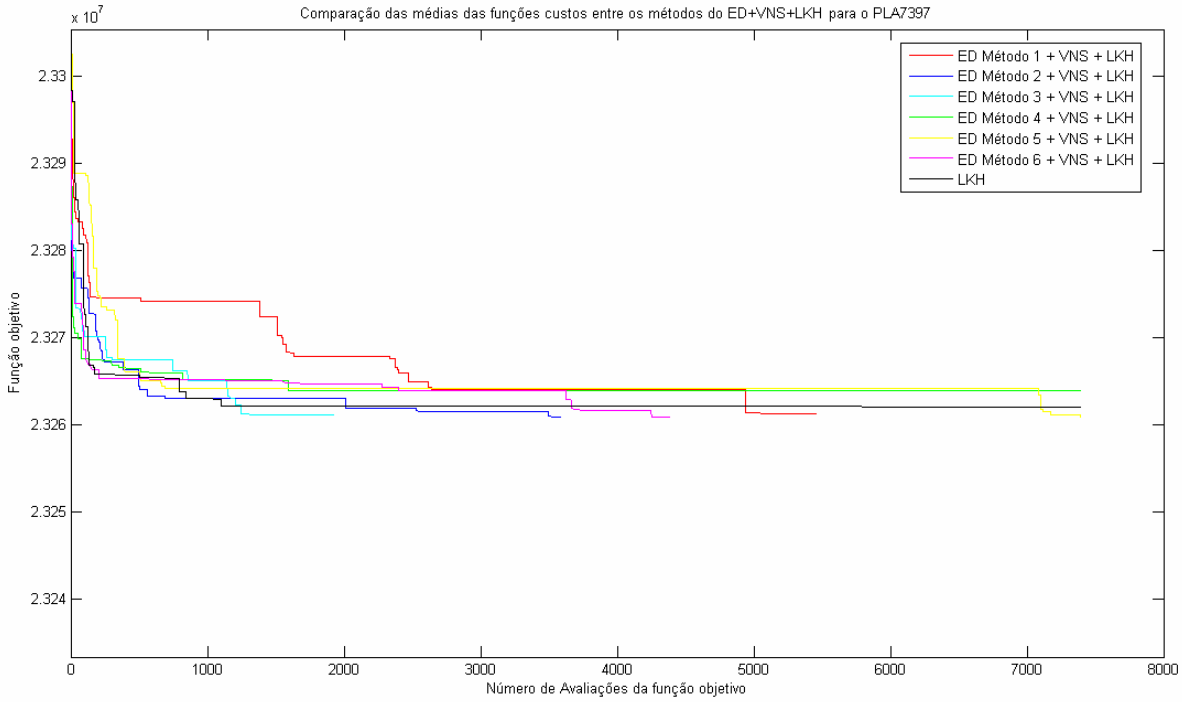

Figura 50: Comparação das médias das funções objetivo entre os métodos do ED+VNS+LKH para o PLA7397

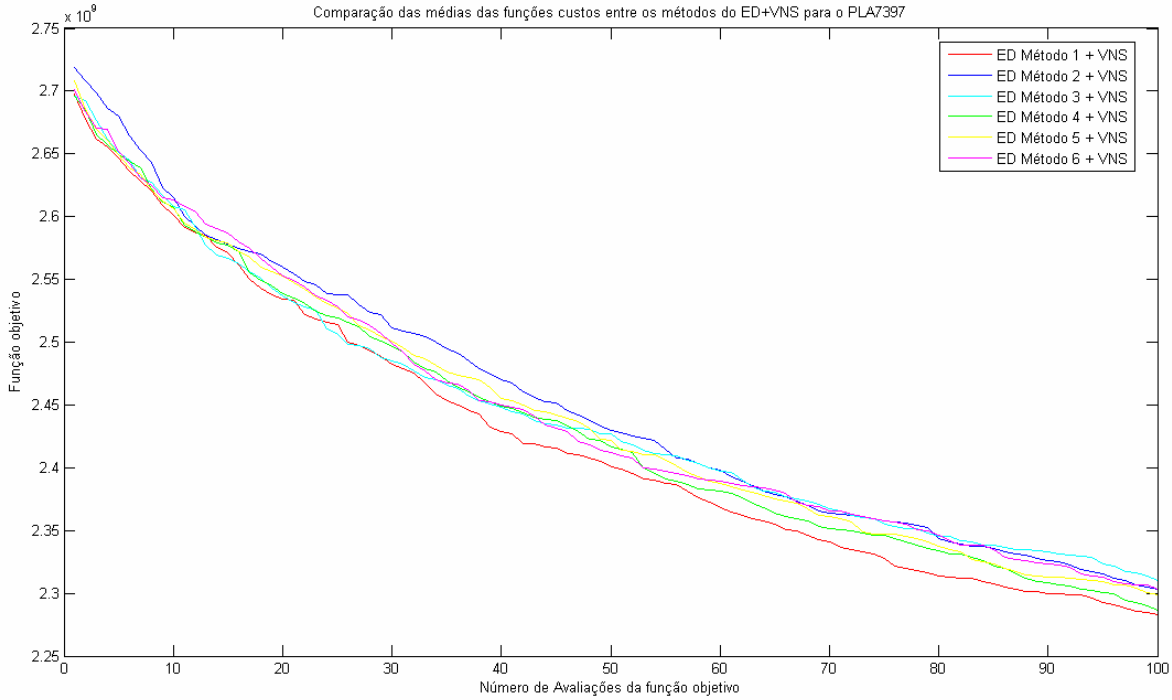

Figura 51: Comparação das médias das funções objetivo entre os métodos do ED+VNS para o PLA7397

Capítulo 5

Conclusões e Sugestões para Trabalhos Futuros

Como visto, o PCV é um problema do tipo NP-Difícil, pois sua dificuldade cresce de maneira exponencial a cada novo nó(cidades) inserido no problema. Computacionalmente é viável resolver problemas que sejam do tipo polinomial, mas o PCV é um problema de complexidade exponencial, o que dificulta a obtenção de uma solução "ótima" para o PCV.

Para o PCV, astécnicas que resolvam problemas de forma polinomial são praticamente inviáveis de se utilizar por causa do seu custo computacional, o que acarreta a sua utilização apenas em problemas pequenos, que sejam menores que 2000 nós. Nestes casos, uma abordagem diferente que utilize metaheurísticas, fornece condições de se obter um resultado próximo do ótimo em um tempo aceitável para um problema deste tipo.

A heurística do Lin-Kernighan [LIN73], ainda é considerada um dos melhores métodos para resolver tais problemas, porém, sozinha demora para se conseguir obter um bom *tour* quando executada de modo isolada, pois sua geração de modo aleatório do *tour* inicial não conseguem, às vezes, chegar ao resultado esperado.

A metaheurística Evolução Diferencial, abordada nesta dissertação, demonstrou que sozinha ela não é uma boa ferramenta para fazer uma busca pelo melhor *tour*, porém, quando se cria um sistema híbrido ela auxilia na obtenção do resultado. Mesmo não apresentando um resultado satisfatório para problemas com poucos nós, a mesma demonstrou que para problemas maiores, consegue melhorar os resultados, fazendo com que ocorra uma otimização do tempo e da função objetivo quando utilizada.

Já o VNS foi uma solução encontrada para o melhoramento do resultado do ED que fosse rápido e que apresentassem melhoras significativas. O ED, por sua vez, utiliza um sistema de busca global de uma solução ótima. Já o VNS faz uma busca local, mas começando com poucos nós e crescendo a quantidade de vizinhos conforme vai conseguindo

melhores *tours*. Isso faz com que se melhore o resultado a ser passado para o LKH e que o mesmo consegui-se obter uma função objetivo e um tempo menor do que o LKH executado de maneira isolada.

O que se nota no capítulo 4 é que o LKH sozinho consegue alcançar resultados ótimos em pequenas instâncias, menores que 2000 cidades, porém, para instâncias maiores ele não consegue achar os resultados rapidamente e em alguns casos ele não consegue achar a função objetivo ótima. A ED aplicada com o LKH demonstrou uma melhora em relação ao tempo em todas as instâncias, porém, quando somado os tempos da ED com o LKH, nota-se que para pequenas instâncias não é uma apropriada estratégia usá-la, pois os resultados da função objetivo obtidos foram iguais e apenas houve um acréscimo de tempo por causa do tempo de execução da ED. Já para instâncias maiores, nota-se um ganho de performance e maior precisão quando se usa ED com LKH. Em alguns casos, chegou-se a ter aproximadamente 30% de melhoramento no tempo demandado para encontrar a função objetivo, que foi o caso do problema PLA7397.

Quando aplicado a metaheurística VNS, que consegue fazer uma busca local em grupos menores, conseguiu-se melhorar a resposta obtida pela ED e com um tempo de aproximadamente 60% mais rápido. Porém, nota-se que ela fez com que todos os métodos da ED convergissem para uma função objetivo com resultados iguais, fazendo com que não houvesse melhoras significativas nos resultados obtidos através do LKH.

De modo geral, o modelo híbrido ED+VNS conseguiu melhorar o *tour* inicial, chegando a ficar aproximadamente 20% menor do que o original criado pela ED. Porém, tal melhora não foi notada ao se rodar o modelo híbrido ED+VNS+LKH. Isto pode ocorrer porque o resultado obtido pelo ED+VNS está convergindo para um *tour* que impossibilite o LKH a conseguir chegar à função objetivo ótima.

O ED possui vários parâmetros de configuração que podem ser ajustados para se obter um melhor resultado. Um dos parâmetros foi utilizado para se tentar alcançar um melhor resultado, que no caso são os métodos usados para obtenção do próximo resultado. Ao todo, foram abordados os 6 primeiros métodos do ED nesta dissertação. Porém, existem ainda mais parâmetros que podem ser modificados para que se consiga chegar um resultado melhor, além de se poder usar os outros 4 métodos que foram criados depois.

Futuras investigações podem ainda ser realizada no VNS, como a mudança do tipo da busca local, que foi usada apenas a 2-*opt*, mas pode-se tentar fazer um sistema que varie conforme a repetição de resultados, como feito no LKH. Pode-se ainda modificar o tamanho

da quantidade de divisão de cidades, que no momento é apenas de 6 cidades. Isso é pode ser usado para um número pequeno de cidades, mas para um número grande cidades esse valor deveria ser aumentado também, pois melhoria a busca local implementada.

Referências

[ALV04] ALVARENGA, G. B., MATEUS, A. (2004). "Two-Phase Genetic and Set Partitioning Approach for the Vehicle Routing Problem with Time Windows" Fourth International Conference on Hybrid Intelligent Systems, Kitakyushu, Japan.

[APP95] APPLEGATE, D., BIXBY, R. E., CHVÁTAL, V., COOK, W. (1995). "Finding Cuts in the TSP: a Preliminary Report", Report 95-05, DIMACS, Rutgers University, New Brunswick, NJ, USA.

[APP07] APPLEGATE, D., BIXBY, R. E., CHVÁTAL, V., COOK, W. (2007). "Traveling Salesman Problem", Disponível em: http://www.tsp.gatech.edu. Acesso em: Março de 2007.

[ARA06] ARANTES, M. B., OLIVEIRA, T. S., SARAMANGO, S. F. (2006). "Evolução Diferencial Aplicada à Solução de Alguns Problemas de Engenharia de Produção". FAMAC em revista, vol. 1, no. 6, pp. 46-61.

[BAR01] BARAGLIA, R. *et al.* (2001). "A Hybrid Heuristic for the Traveling Salesman Problem". IEEE Transactions on Evolutionary Computation, vol. 5, no. 6, pp. 613-622.

[BRE06] BREST, J. GREINER, S., *et al*. (2006). "Self-Adapting Control Parameters in Differential Evolution: A Comparative Study on Numerical Benchmark Problems". IEEE Transactions on Evolutionary Computation. vol. 10, no. 6, pp. 646-657

[BUR91] BURKARD, R. E., SANDHOLZER, W. (1991) "Efficiently Solvable Special Cases of Botteneck Travelling Salesman Problems". Discrete Applied Mathematic. vol. 32. pp. 61-76.

[CHA05] CHAVES, A. A. (2005). "Heurísticas Híbridas com Busca através de Agrupamentos para o Problema do Caixeiro Viajante com Coleta de Prêmios". Dissertação de Mestrado, INPE, São José dos Campos, SP.

[CHR72] CHRISTOFIDES, N., EILON, S. (1972). "Algorithms for Large-scale Traveling Salesman Problems", Operations Research, vol. 23, pp. 511-518.

[CHR75] CHRISTOFIDES, N. (1975). "Graph Theory: An Algorithm Approach". Academic Press. Inc. Burlington, MA, EUA.

[COR01] CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L., STEIN, C. (2001). "Introduction to Algorithms." Second Edition, The MIT Press, Cabridge - MA, USA.

[COT95] COTTA C., ADLANA J. F., NEBRO A. J., TROYA J. M. (1995). "Hybridizing Genetic Algorithms with Branch and Bound Techniques for the Resolution of the TSP, Artificial Neural Nets and Genetic Algorithms", NY, USA, pp. 278-280.

[CRO58] CROES, G. (1958). "A Method for Solving Traveling Salesman Problems", Operations Research , vol. 6, pp. 791-8112.

[CRO80] CROWDER, H., PADBERG, M. W. (1980) "Solving Large-scale Symmetric Traveling Salesman Problems to Optimality", Management Science, vol. 26, pp. 495-509.

[CUN97] CUNHA, C.B. (1997). "Uma Contribuição para o Problema de Roteirização de Veículos com Restrições Operacionais". EPUSP, Departamento de Engenharia de Transportes. São Paulo, SP. (Tese de Doutorado).

[DAN54] DANTZIG, G. B., FULKERSON, D. R., JOHNSON, S. M. (1954), "Solution of a Large-scale Traveling Salesman Problem", Operations Research, vol. 2, pp. 393-410.

[DAV87] DAVIS, L. (1987). "Genetic Algorithms and Simulated Annealing". Morgan Kaufmann Publishers. San Francisco, CA, EUA.

[DOR96] DORIGO, M. GAMBARDELLA L. M. (1996). “Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem”, BioSystem. vol. 1, no. 1, pp. 53-66.

[DOR97] DORIGO M.; GAMBARDELLA L. M. (1997). “Ant Colonies for the Traveling Salesman Problem”, BioSystem. vol. 1, no. 1, pp. 73-81.

[DUA06] DUARTE, H. M., (2006), “Um Estudo Algorítmico de Problemas Logísticos na Industria de Petróleo e Gás Natural”, Mestrado em Sistemas e Computação, Universidade Federal do Rio Grande do Norte, Natal - RN.

[FIS88] FISCHETTI, M., TOTH, P. (1988). “An Additive Approach for the Optimal Solution of the Prize-Collecting Traveling Salesman Problem”, Vehicle Routing Methods and Studies. North-Holland, Amsterdam, pp.319-343.

[GAR79] GAREY, M. R., JOHNSON, D. S. (1979), “Computers and Intractability. A Guide to the Theory of NP-Completeness”, Freemann, New York, EUA.

[GLO77] GLOVER, F. (1977). “Heuristics for Integer Programming Using Surrogate Constraints”, Decision Science, vol. 8, pp. 156-166.

[GLO89-a] GLOVER, F. (1989). “Tabu Search – Part I”. ORSA Journal on Computing, vol. 1, pp. 190-206.

[GLO89-b] GLOVER, F. (1989). “Tabu Search – Part II”. ORSA Journal on Computing, vol. 1, pp. 207-225.

[GRO87] GRÖTSCHELAND, M., HOLLAND, O. (1987). “A Cutting Plane Algorithm for Minimum Perfect” Matching. Computing, vol. 39. pp. 327–344.

[GRO88] GRÖTSCHEL, M., LOVÁSZ, L. e SCHRIJVER, A. (1988). "Geometric Algorithms and Combinatorial Optimization", Society for Industrial and Applied Mathematics. London, UK, vol. 1, no. 2, pp. 332-233.

[HEL98] HELSGAUN, K. (1998). "An Effective Implementation of the Lin Kernighan Traveling Salesman Heuristic", DATALOGISKE SKRIFTER (Writings on Computer Science), no. 81, Roskilde University. Roskilde, Dinamarca.

[HO03] HO, L. T. W.; SAMUEL, L. G., PITTS, J. M. (2003). "Applying Emergent Self-Organizing Behavior for the Coordenation of 4G Networks using Complexity Metrics", Bell Labs Technical Journal, vol. 8, no. 1, pp. 5-25.

[HOL75] HOLLAND, J. H. (1975). "Adaptation in Natural and Artificial Systems". Ann Arbor: University of Michigan Press. Michigan, EUA.

[JOH90] JOHNSON, D. S. (1990). "Local Optimization and the Traveling Salesman Problem", Lecture Notes in Computer Science, vol. 442, Springer, London, UK, pp. 446-461.

[JOH97] JOHNSON, D. S., McGEOCH, L. A. (1997). "The Traveling Salesman Problem: A Case Study in Local Optimization" em E. H. L. Aarts and J. K. Lenstra (eds.), Local Search in Combinatorial Optimization, Wiley, NY, USA.

[JUN02] JUNG, S., MOON, B. R. (2002). "Toward Minimal Restriction of Genetic Encoding and Crossovers for the Two-Dimensional Euclidean TSP", IEEE Transactions on Evolutionary Computation, vol. 6, no. 6, pp. 215-310.

[KAO78] KAO, E. P. C. (1978). "A Preference Order Dynamic Program for the Stochastic Traveling Salesman Problem". Optimum Research. vol. 26, pp. 1033-1045.

[KIR83] KIRKPATRICK, S., GELATT, C. D., VECCHI, M. P. (1983). "Optimization by Simulated Annealing", Science, vol. 220, pp. 671-680.

[KIT99] KITA, N., DAVISON, U., *et al* (1999). “Mobile Sensing Robots for Nuclear Power Plant Inspection.” Advanced Robotics, vol. 13, no. 3, pp. 355-356.

[LAP87] LAPORT, G., NORBERT, Y. (1987). “Exact Algorithms for the Vehiche Routing Problem”. Discrette Mathematic, vol. 31, pp. 147-184.

[LAP92] LAPORT, G. (1992). “The Traveling Salesman Problem: An Overview of Exact and Approximate Algorithms”, European Journal Operational Research, vol. 59, pp. 231-247.

[LAW85] LAWLER, E. L., LENSTRA, *et al.* (1985). “The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization”, Wiley, NY, USA.

[LIN65] LIN, S. (1965). “Computer Solutions for the Traveling Salesman Problem”. Bell System Computing Journal, vol. 44, pp. 2245-2269.

[LIN73] LIN, S., KERNIGHAN, B. W. (1973). “An Effective Heuristic Algorithm for the Traveling Salesman Problem”. Operations Research, vol. 21, pp. 498- 516.

[MIL91] MILLER, D. L., PEKNY, J. F. (1991). “Exact Solution of Large Asymmetric Traveling Salesman Problems”, Science, vol. 251, pp. 754-761.

[MLA95] MLADENOVIC, N. (1995). “A Variable Neighborhood Algorithm – a New Metaheuristic for Combinatorial Optimization”. Abstracts of Papers at Optimization Days, Montreal, Canadá. vol. 112.

[MLA97] MLADEVONIC, N., HANSEN P. (1997). “Variable Neighbourhood Search”. Computers and Operations Research, vol. 24, pp. 1097-1100.

[MOR97] MORAIS, M. J. C. (1997). “PCV: Um Velho Problema Revisitado Estatísticamente”. V Conferência do CEMAPRE (Centre for Applied Mathematics and Economics), Lisboa, Portugal.

[MOR01] MOREIRA, N. (2001). “Máquinas de Turing: Uma Introdução”, Technical Report, Departamento de Ciência de Computadores da Faculdade de Ciências da Universidade do Porto, Porto, Portugal.

[NOV01] NOVAES, A. G. (2001). “Logística e Gerenciamento da Cadeia de Distribuição”, Editora Campus, Rio de Janeiro, RJ.

[ONG82] ONG, H. L. (1982). “Approximate Algorithms for the Traveling Purchaser Problem” Operations Research Letters vol. 1, no. 5, pp. 201-205.

[OSM96] OSMAN, I. H., LAPORTE, G. (1996). “Metaheuristics: a Bibliography”, Annals of Operations Research, vol. 63, pp. 513-623.

[PAD91] PADBERG, M.W. e RINALDI, G., (1991) “A Branch and Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems”, SIAM (Society for Industrial and Applied Mathematics) Review, vol. 33, pp. 60-100.

[RAM05] RAMOS, I. C. O., (2005) “Metodologia Estatística na Solução do Problema do Caixeiro Viajante e na Avaliação de Algoritmos: Um Estudo Aplicado a Transgenética Computacional”, Programa de Pós-graduação em Engenharia Elétrica da Universidade Federal do Rio Grande do Norte, RN.

[REE96] REEVES, C. R., RAYWARDSMITH, *et al*. (1996). “Modern Heuristic Techniques”. Modern Heuristic Search Methods, John Wiley & Sons, pp. 1-25.

[REI91] REINELT, G. (1991) “TSPLIB: a Traveling Salesman Problem Library”. ORSA Journal on Computing, vol. 4, pp. 86-384.

[REI94] REINELT, G. (1994). “The Traveling Salesman: Computational Solutions for TSP Applications”, Lecture Notes in Computer Science, Springer, vol. 840. London, UK.

[RIB96] RIBEIRO, C. C. (1996) “Metaheuristics and Applications”. Advanced School on Artificial Intelligence, Estoril, Portugal.

[ROD00] RODRIGUES, M. A., (2000). “Problema do Caixeiro Viajante – Um Algoritmo para Resolução de Problemas de Grande Porte Baseado em Busca Local Dirigida”. Dissertação de Mestrado, Engenharia de Produção, UFSC. Florianópolis, SC.

[ROM04] ROMERO, R.; MONTOVANI, J. R. S. (2004). “Introdução à Metaheurísticas”. Anais do III Congresso Temático de Dinâmica e Controle da SBMAC (Sociedade Brasileira de Matemática Aplicada e Computacional), Universidade Estadual Paulista, Ilha Solteira, SP, Brasil.

[ROS77] ROSENKRANTZ, D. E., STREAMS, R. E., LEWIS II, P. M. (1977). “An Analysis of Several Heuristics for the Traveling Salesman Problem”, SIAM (Society for Industrial and Applied Mathematics) J. Computer., vol.6, pp.563-581.

[SIM98] SIMONETTI, N. (1998). “Applications of a Dynamic Programming Approach to the Traveling Salesman Problem”. Carnegie Mellon University, Pittsburgh, PA, USA

[STO95] STORN, R., PRICE, K., (1995). “Differential Evolution: a Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces”, Technical Report TR-95-012, International Computer Science Institute, Berkeley, CA, USA.

[STO97] STORN, R., PRICE, K., (1997). “Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces”, Journal of Global Optimization, vol. 11, no.1, pp. 341–359.

[TAN00] TAN, K. C., *et al.* (2000) “Heuristic Methods for Vehicle Routing Problem with Time Windows”. Elsevier Science: Artificial Intelligence in Engineering, vol. 15. pp. 281-295.

[TSA04] TSAI, C. F., TSAI, C. W., TSENG, C. C. (2004). “A New Hybrid Heuristic Approach for Solving Large Traveling Salesman Problem”. Information Sciences, vol. 166, no. 1-4, pp. 67-81.

[TUR36] TURING, A. (1936), “On Computable Numbers, with an Application to the Entscgeidungsproblem”. Proceedings of the London Mathematical Society, Raven Press. London, UK.

[WAG03] WAGNER, S.; AFFENZELLER, M.; IBRAHIM, I. K. (2003). “Agent-based Problem Solving: The Ant colonies Metaphor.”. Proceedings of the 5 International Conference on Information Integration and Web-Based Applications & Services, pp. 317-323. Österreichische Computer Gesellschaft. Áustria.