# Problema do Caixeiro Viajante Aplicado ao Roteamento de Veículos numa Malha Viária

***José Luiz Machado Morais***

UNIFESP
UNIVERSIDADE FEDERAL DE SÃO PAULO
1933

# Problema do Caixeiro Viajante Aplicado ao Roteamento de Veículos numa Malha Viária

José Luiz Machado Morais

Trabalho de conclusão de curso apresentado ao Instituto de Ciência e Tecnologia – UNIFESP, como parte das atividades para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Luis Augusto Angelotti Meira

São José dos Campos – SP
Dezembro, 2010

# Problema do Caixeiro Viajante Aplicado ao Roteamento de Veículos numa Malha Viária

José Luiz Machado Morais

Trabalho de conclusão de curso apresentado ao Instituto de Ciência e Tecnologia – UNIFESP, como parte das atividades para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Luis Augusto Angelotti Meira

Banca Examinadora:

_____

Prof. Dr. Luis Augusto Angelotti Meira

_____

Profa. Dra. Kelly Cristina Poldi

_____

Prof. Dr. Arlindo Flavio da Conceição

Aprovado em:

*Aos meus pais.*

# Agradecimentos

Agradeço primeiramente a Deus, que me inspirou e me deu força e coragem a cada dia para transpor os obstáculos encontrados durante esta importante etapa da minha vida.

Ao meu anjo da guarda, por ter me guiado e protegido no caminho percorrido até aqui.

Aos meus pais, Lafaiete e Elisiara, por todo o amor, doação e apoio incondicional que sempre me deram.

Ao meu irmão João Vitor, amigo de todas as horas, pelo companheirismo, amizade e por todos os momentos de felicidade que passamos juntos.

Ao meu orientador, o professor Meira, pelo empenho, prestatividade, confiança e conhecimento compartilhado ao longo deste trabalho.

Aos professores, que transmitiram seu conhecimento e contribuíram de forma indispensável para minha formação profissional e pessoal.

Aos amigos e companheiros de classe, pelo companheirismo, pelas árduas horas de estudos e pelos momentos divertidos e descontraídos que compartilhamos ao longo desses anos.

A todos que, direta ou indiretamente, contribuíram para a realização deste trabalho.

# Resumo

O Problema do Caixeiro Viajante (TSP) é um problema da classe NP-Difícil que modela diversas aplicações práticas. Este documento contém uma monografia de Trabalho de Graduação que objetiva estudar, implementar e comparar soluções para a variante do TSP-Métrico aplicada ao roteamento de veículos numa malha viária. Aqui estão contidas a revisão bibliográfica utilizada como base para este trabalho, a modelagem dos problemas, definições importantes relacionadas a grafos e às classes dos problemas P, NP, NP-Completos e NP-Difíceis, a apresentação das técnicas propostas para resolver o problema, os resultados dos experimentos feitos no decorrer do ano e a análise comparativa entre as técnicas.

**Palavras-chave**: Caixeiro Viajante, Caixeiro Viajante Métrico, Otimização.

# Abstract

The Traveling-Salesman Problem (TSP) is a NP-Hard problem that models many practical applications. This document contains a undergraduation final work which aims to study, implement, and compare solutions to TSP-Metric applied to the routing of vehicles on a road network. The dissertation contains the literature review used as the basis for this work, the modeling of problems, graphs definitions, classes of problems P, NP, NP-Complete and NP-Hard, presentation of the techniques that were addressed to resolve the problem, as well as results of the experiments that were made during this year and comparative analysis between the techniques.

**Keywords**: Traveling-Salesman Problem, Optimization, TSP, TSP-Metric.

# Sumário

CAPÍTULO

1

# Introdução

Este trabalho aborda uma variante do Problema do Caixeiro Viajante, um problema clássico na área de otimização. O caso específico de interesse é o roteamento de veículos em uma malha viária, que recai no Problema do Caixeiro Viajante Métrico. A Figura 1.1 exemplifica uma solução válida.

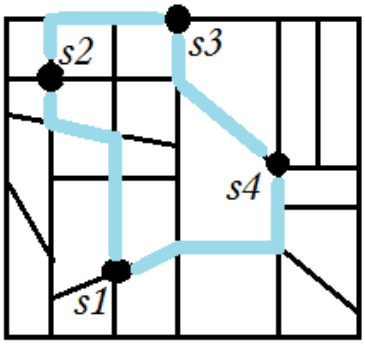



**Figura 1.1:** Exemplo de permutação (*s1, s2, s3, s4*) para um conjunto de vértices.

Dada uma instância representada por um grafo, o objetivo é encontrar, a partir de um vértice inicial, um ciclo que passe por todos os demais vértices do grafo, retorne ao vértice inicial e que tenha custo mínimo.

Esse problema foi resolvido por três técnicas distintas: solução heurística, algoritmo de aproximação e força bruta. Na parte experimental, foram resolvidas diversas instâncias de

tamanhos variados, incluindo instâncias reais, com o intuito de compreender e comparar diferentes métodos de solução através da análise dos resultados obtidos.

Esta monografia trata também de classes de problemas, a fim de ajudar na compreensão e descrição do trabalho, bem como na fundamentação das técnicas para resolvê-lo. As classes de problemas NP-Completo e NP-Difícil foram enfatizadas, uma vez que estão diretamente relacionadas ao problema aqui tratado.

## 1.1 Objetivos

O principal objetivo deste trabalho é analisar diferentes técnicas de resolução para o Problema do Caixeiro Viajante, implementá-las e compará-las, dando atenção tanto ao tempo de execução quanto à qualidade da solução obtida por cada uma.

## 1.2 Organização

Este trabalho está organizado da seguinte forma: o Capítulo 2 descreve toda a base teórica para a implementação dos algoritmos e mostra uma série de definições importantes para a compreensão do tema abordado. O Capítulo 3 traz detalhes sobre os algoritmos utilizados e suas respectivas implementações. Por fim, o Capítulo 4 mostra os resultados empíricos obtidos através dos testes, e é seguido pelas considerações finais e pela conclusão do trabalho.

CAPÍTULO

2

# Conceitos e definições

Este capítulo contém a revisão bibliográfica que serviu como base para o trabalho. Nele estão contidas as definições utilizadas ao longo do estudo, bem como os modelos teóricos utilizados. Dentre os tópicos abordados, estão as classes de problemas P, NP, NP-Completos e NP-Difíceis, algumas definições importantes em grafos, a definição do Problema do Caixeiro Viajante (TSP, do inglês *traveling-salesman problem*) e do Problema do Caixeiro Viajante Aplicado ao Roteamento de Veículos numa Malha Viária, um caso particular do TSP-Métrico.

## 2.1 Classes dos problemas

Para compreender o problema abordado neste trabalho, é necessário conhecer a classe de problemas NP-Completos. Para os problemas dessa classe ainda não foi descoberto nenhum algoritmo que pudesse resolvê-los em tempo polinomial, isto é, com tempo de execução limitado por $O(n^k)$, sendo *n* o tamanho da instância de entrada e *k* uma constante qualquer. Em outras palavras, não há expectativas de algoritmos eficientes para problemas NP-Completos. Para melhor compreender essa classe e também as classes P e NP, segue uma definição básica de cada uma delas.

### 2.1.1 A classe P

A classe P engloba todos os problemas decisórios que podem ser resolvidos por um algoritmo de tempo polinomial. Tais problemas são chamados problemas tratáveis, ou problemas fáceis. Um problema cujo algoritmo é de ordem $O(n^k)$ com uma constante *k* muito grande é considerado eficiente, pois é provável, pelo que se vê ao longo da história dos algoritmos, que logo sejam encontrados novos algoritmos mais eficientes que resolvam tal problema com uma complexidade menor. Os problemas em P, que podem ser resolvidos em tempo polinomial, têm características muito parecidas. Problemas mais complexos, compostos por vários problemas polinomiais, também pertencem à classe P.

### 2.1.2 A classe NP

A classe NP engloba os problemas decisórios cuja resposta "sim" pode ser verificada em tempo polinomial. Isto é, dada uma solução "sim" para um determinado problema, pode-se testar um certificado de que essa solução é correta em tempo polinomial. Um bom exemplo, que é utilizado nesse trabalho, é verificar se uma sequência de vértices $(v_1, v_2, \ldots, v_{|V|})$ forma um ciclo hamiltoniano em um grafo $G = (V, E)$. Tal sequência é um certificado curto para "sim", existe um ciclo hamiltoniano no grafo. É importante lembrar que todo problema pertencente à classe P também pertence à classe NP, uma vez que se um problema pode ser resolvido em tempo polinomial, ele pode ser igualmente verificado em tempo polinomial. Uma das grandes discussões acerca desse tema é descobrir se P = NP ou P $\neq$ NP. É sabido que P é um subconjunto de NP, como foi dito acima, mas ainda não se chegou a uma prova de que P é um subconjunto próprio de NP, ou que P = NP.

### 2.1.3 A classe NP-Completos

Esta classe de problemas é alvo de longos estudos e pesquisa durante as últimas décadas, pois ela está no centro de um dos impasses mais famosos dentro da computação. Um problema é NP-Completo se ele pertence à classe NP e, caso seja encontrado um algoritmo que seja capaz de resolvê-lo em tempo polinomial, então estará provado que P = NP. A maioria dos pesquisadores acredita que P $\neq$ NP, pois já se tem várias décadas de estudos acerca desse tema e nunca foi encontrado nenhum algoritmo polinomial para qualquer problema NP-Completo. Contudo, os estudos também não levaram a uma conclusão que permite afirmar que esse algoritmo não existe.

Para formalizar a definição desta classe é preciso definir o conceito de redução em tempo polinomial. Com esse intuito, é interessante utilizar uma estrutura de linguagem formal, uma

vez que problemas de decisão tornam viável o uso desse mecanismo. Segue a definição de redução polinomial através da utilização da teoria da linguagem formal:

> Uma linguagem (problema) $L_1$ é redutível em tempo polinomial a outra linguagem (problema) $L_2$, o que se escreve como $L_1 \leq_p L_2$, se existir uma função calculável de tempo polinomial $f : \{0,1\}^* \rightarrow \{0,1\}^*$, tal que para todo $x \in \{0,1\}^*, x \in L_1$ se e somente se $f(x) \in L_2$ (Cormen *et al.*, 2001).

Formalmente, ainda segundo Cormen, a NP-Completude é definida da seguinte forma:

Uma linguagem (problema) $L \subseteq \{0,1\}^*$ é NP-Completa se:

- 1. $L \in NP$, e
- 2. $L^{'} \leq_p L$ para todo $L^{'} \in NP$.

A redução polinomial possibilita mostrar que um problema é tão difícil quanto um outro. É possível reduzir todos os problemas contidos em NP para qualquer problema NP-Completo. Por esse motivo, é dito que a partir de um problema NP-Completo é possível se chegar a todos os demais problemas dessa mesma classe.

Em termos mais simples, um problema L em NP é NP-Completo se um algoritmo polinomial para L implica P = NP.

A Figura 2.1 mostra o cenário mais aceito atualmente para as classes de problemas descritas.

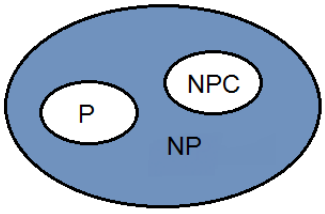



**Figura 2.1:** Cenário das classes de problemas mais aceito atualmente.

### 2.1.4 A classe NP-Difícil

Esta é a classe na qual está contido o problema aqui abordado. Os problemas NP-Difíceis são problemas geralmente relacionados à otimização, e eles são definidos formalmente, segundo Cormen, da seguinte forma:

Uma linguagem $L \subseteq \{0,1\}^*$ é NP-Difícil se:

- $L^{'} \leq_p L$ para todo $L^{'} \in NP$.

Contudo, um problema NP-Difícil pode também estar em NP, mas não necessariamente. Os problemas contidos nesta classe são ditos ao menos tão difíceis quanto os problemas mais difíceis de NP e não é possível resolvê-los na exatidão, a menos que P = NP. Ou seja, um algoritmo polinomial que encontra a solução exata para um problema NP-Difícil implica P = NP.

## 2.2 Definições em grafos

Esta seção contém definições obtidas no livro *Introduction to Algorithms*, de Thomas Cormen, e que se fazem necessárias para a compreensão dos conceitos e técnicas aqui utilizados.

**Grafo -** um grafo *G* é um par ordenado $(V(G), E(G))$, onde $V(G)$ é um conjunto de vértices, $E(G)$ é um conjunto de arestas, e uma função de incidência $\psi_G$ que associa cada aresta $e \in E(G)$ a um par de vértices *u*, $v \in V(G)$ que são denominados os extremos de $e$.

**Grafo Completo -** um grafo completo é um grafo simples $G(V, E)$ tal que $\forall\ u,\ v \in V$, $u \neq v$, $\exists\ (u, v) \in E$.

**Grafo Conexo -** um grafo $G(V, E)$ é conexo se é possível chegar a qualquer vértice $v \in V$ a partir de um vértice inicial qualquer $v_i$, passando pelas arestas de $E$.

**Subgrafo -** dado um grafo $G(V, E)$, $H(V', E')$ é um subgrafo de *G* se *H* é um grafo e $V' \subseteq V$ e $E' \subseteq E$.

**Custo de um Grafo (*c(G)*) -** dada uma função $c : E \to \mathbb{Q}^+$, um grafo $G(V, E)$ e o custo de um grafo, $c(G)$ é $\sum_{e \in E} c(e)$.

**Passeio -** dado um grafo $G(V, E)$, um passeio em $G$ é uma sequência alternada de vértices e arestas.

**Trilha -** uma trilha é um passeio no qual não há repetições de arestas.

**Caminho -** um caminho é uma trilha na qual não há repetição de vértices.

**Ponte -** dado um grafo conexo $G(V, E)$, uma ponte é uma aresta $e \in E$ que torna o grafo desconexo caso seja retirada de $E$.

**Ciclo -** dado um grafo $G(V, E)$, um ciclo é uma sequência de vértices $(v_1, \ldots, v_n)$ tal que $(v_1, v_{i+1},) \in E$ e $(v_n, v_1,) \in E$.

**Ciclo Euleriano -** dado um grafo $G(V, E)$, um ciclo Euleriano é um ciclo que passa por todas as arestas de *E* uma única vez.

**Grafo Euleriano -** um grafo $G(V, E)$ é Euleriano se possui um ciclo Euleriano.

**Árvore -** Uma árvore é um grafo conexo e acíclico.

**Árvore Geradora -** $T(V', E')$ é uma árvore geradora de $G(V, E)$ se $V' = V$, $E' \subseteq E$ e $T$ é uma árvore.

**Floresta -** é uma coleção de árvores $F$ em um grafo $G(V, E)$ sem cíclos.

**Árvore Geradora de Custo Mínimo (MST, do inglês *minimum-cost spanning tree*) -** uma árvore geradora de custo mínimo de um grafo $G(V, E)$ é uma árvore geradora $T$ de $G$ tal que $c(T)$ é mínimo.

**Aresta Segura -** dada uma MST $T(V', E')$ e $A$ um subconjunto de arestas de $T$, uma aresta segura é uma aresta $uv$ que pode ser adicionada a $A$ sem formar ciclos ou violar a propriedade $A \cup \{uv\} \subseteq E'$.

**Corte -** dado um grafo não orientado $G(V, E)$, um corte $[S, \overline{S}]$ de $G$ é uma bipartição de $V$. Uma aresta $uv \in E$ cruza o corte se $u \in S$ e $v \in \overline{S}$.

**Aresta Leve -** dado um corte $[S, \overline{S}]$, uma aresta é leve se ela cruza o corte e tem o menor custo.

## 2.3 O problema do caixeiro viajante (TSP)

O Problema do Caixeiro Viajante (TSP, do inglês *Traveling-Salesman Problem*) é um problema NP-Difícil da área de otimização que possui inúmeras aplicações práticas. O TSP modela diversas situações reais, como o roteamento de veículos para atender chamados ou ocorrências. Por pertencer à classe dos problemas NP-Difíceis, não há algoritmos eficientes para resolvê-lo na exatidão. O TSP também é encontrado como subproblema de modelagens maiores, que envolvem muitas vezes vários problemas pertencentes às classes NP-Completa e NP-Difícil, como o Problema da Mochila (Martello e Toth, 1990). O Problema da Mochila pode ser usado em conjunto com o TSP para modelar problemas de empacotamento e entrega da mercadoria de um depósito. Primeiro empacotaria-se dados elementos dentro de um caminhão segundo restrições e critérios definidos. Depois a entrega seria feita no menor ciclo hamiltoniano partindo-se do depósito. A função objetivo, nesse caso, é minimizar o custo da soma de todos os ciclos hamiltonianos no processo.

O TSP consiste em, dado um grafo completo $G(V, E)$, com *n* vértices, obter um ciclo hamiltoniano de custo mínimo, isto é, deseja-se, a partir de um vértice inicial, passar por todos os demais vértices do grafo uma única vez e então retornar ao vértice inicial. Cada aresta que liga um par de vértices do grafo possui um custo $c(i, j)$ que determina o quanto se gasta para

ir de *i* até *j*. Esse custo pode ter diversos significados, de acordo com a aplicação desejada. Ele pode representar, por exemplo, a distância, o tempo ou mesmo o preço para se deslocar entre duas cidades *i* e *j*. Considerando esses custos, o objetivo do problema é obter um ciclo hamiltoniano de custo total mínimo a partir de um dado ponto inicial. Esse custo total é dado pela soma dos pesos de todas as arestas contidas no ciclo.

A definição de um ciclo hamiltoniano é dada da seguinte forma:

Dado um grafo $G(V, E)$ com *n* vértices, deseja-se obter um ciclo $(v_1, \ldots, v_n)$ que parta de um vértice inicial $v_1$, passe por todos os demais *n-1* vértices de *V* uma única vez e retorne a $v_1$.

A definição formal pode ser escrita como se segue:

Dados um grafo completo $G(V, E)$ e uma função custo $c : V$ x $V \rightarrow \mathbb{Q}^+$, encontrar uma permutação $(v_1, v_2, \ldots, v_n)$ de *V* tal que

$$c(v_n, v_1) + \sum_{i=1}^{n-1} c(v_i, v_{i+1}) \tag{2.1}$$

seja mínimo.

Na Seção 2.4 será feita a particularização do TSP para o caso específico do roteamento de veículos numa malha viária, que é um caso particular do TSP-Métrico. O TSP-Métrico se enquadra na classe dos problemas NP-Difíceis e, portanto, não há algoritmos exatos para ele. Apesar de não haver algoritmos aproximados para o TSP, a menos que $P = NP$ (Vazirani, 2003), é possível fazer aproximações para o TSP-Métrico. No Capítulo 3 serão mostradas técnicas para uma resolução heurística, aproximada e exata do problema, visando encontrar soluções em tempo viável de acordo com as instâncias utilizadas. No caso exato, os algoritmos não serão polinomiais.

## 2.4 Caixeiro viajante aplicado ao roteamento de veículos numa malha viária

Realizar o roteamento de veículos em uma malha viária de maneira eficiente é um problema enfrentado por diversas empresas que fazem entregas de mercadorias ou prestação de serviços, por exemplo. Estas empresas necessitam enviar um carro para atender a uma demanda de clientes que estão localizados em diversos pontos distintos, sejam eles ruas e bairros, no caso de uma empresa local, ou cidades e estados, considerando empresas com maior área de atuação. Visando otimizar o serviço, é interessante para essas empresas encontrar a ordem de atendimento às demandas de modo a minimizar o custo para o deslocamento entre os pontos de demanda. Tal tarefa não possui soluções simples.

Ao modelar-se o caso real, os pontos de demanda são associados a vértices de um grafo. Neste grafo completo $G(V, E)$, o custo $c(i, j)$ é o valor numérico que representa o caminho de menor custo na malha formada por ruas, avenidas e rodovias. Este custo $c(i, j)$ pode representar o caminho de menor comprimento, de menor tempo, ou mesmo de menor custo financeiro para realizar o percurso. Minimizar o custo do trajeto do carro que sai da empresa, atende todos os pontos de demanda e retorna após o serviço pode ser modelado como uma instância do TSP, que é um problema NP-Difícil.

O problema estudado neste trabalho de graduação é a particularização do TSP para o roteamento de veículos numa malha viária. Este caso particular recai no Problema do Caixeiro Viajante Métrico (TSP-Métrico). Neste caso pode-se assumir que a desigualdade triangular é válida. Considerando-se três vértices quaisquer de *V*, por exemplo *a*, *b* e *c*, o custo para ir de *a* para *b* é menor do que o custo para se deslocar de *a* para *c* e de *c* para *b*. Em outras palavras, $c(a, b) \leq c(a, c) + c(c, b)$, $\forall a$, $b$ e $c \in V$.

## 2.5 Trabalhos relacionados

Esta seção traz alguns trabalhos da área de otimização que estão relacionados à abordagem aqui proposta.

Existem trabalhos na literatura que propõem técnicas heurísticas baseadas em colônias de formigas para a resolução do TSP, tanto para o TSP-Métrico quanto para a modelagem generalizada. Este método de solução é influenciado pelas áreas de redes neurais artificiais e computação evolutiva, que estão fortemente relacionadas a problemas da área de otimização (Dorigo e Gambardella, 1996), (Dorigo e Stutzle, 1999).

Heurísticas simples, como a que será descrita na Seção 3.2, já foram estudadas e analisadas anteriormente, focalizando a importância de uma boa solução inicial e da escolha correta dos critérios de parada do algoritmo (da Cunha *et al.*, 2002).

Também existem modelagens exatas e heurísticas para uma outra variação do TSP, denominada TSP com coleta de prêmios. Esta variação do problema inclui, além do custo das viagens, a atribuição de um valor positivo a cada cidade. O caixeiro recebe um prêmio que é o valor atribuído à cidade, caso a visite, e paga uma penalidade para cada cidade não visitada. A função objetivo desse problema é minimizar o custo da total da viagem e das penalidades, enquanto é necessário se obter um valor mínimo dado pela soma dos prêmios coletados (Chaves *et al.*, 2004).

É possível encontrar na literatura diversas técnicas para a resolução do TSP, como metaheurísticas, heurísticas de mínimo local, algoritmos genéticos e algoritmos com base em redes

neurais artificiais (Johnson e McGeoch, 1995). Algoritmos meméticos também podem ser utilizados para resolver algumas variantes do TSP (Moscato e Norman, 1991).

CAPÍTULO

3

# Abordagens de resolução

Neste capítulo estão descritas as técnicas utilizadas para resolver o TSP-Métrico. Foram utilizadas três técnicas diferentes, Força Bruta, Heurística e Algoritmos de Aproximação. Este capítulo também traz uma quarta técnica de abordagem, o *Branch & Bound*, um método exato para resolver o TSP-Métrico, mas que não foi implementada. Essas quatro técnicas serão tratadas em seções individuais neste capítulo.

## 3.1 Algoritmos de força bruta

Algoritmos de força bruta são uma forma de se encontrar a solução ótima para um problema de otimização através do teste exaustivo de todos os elementos de seu domínio. Considerando o TSP, que é o problema estudado neste trabalho, um algoritmo de força bruta consiste em testar todas as permutações possíveis entre os vértices de uma instância e escolher o ciclo hamiltoniano de menor custo.

Formalmente, a abordagem através da força bruta para o TSP Métrico é dada da seguinte forma:

Dados um grafo completo $G(V, E)$ e uma função custo $c : V$ x $V \rightarrow \mathbb{Q}^+$, encontrar e testar todas as permutações $(v_1, v_2, \ldots, v_n)$, com $n = |V|$, de *V* e escolher a que minimize $c(v_n, v_1)$ + $\sum_{i=1}^{n-1} c(v_i, v_{i+1})$.

Apesar de devolver a solução ótima para o problema, uma vez que todas as possibilidades são testadas, o algoritmo de força bruta tem um custo computacional muito elevado. Mesmo para instâncias pequenas é esperado que o algoritmo não consiga resolver o problema em tempo viável, pois o custo computacional é $\Theta(n!)$ (Cormen *et al.*, 2001).

### 3.1.1 Implementação

Para implementar o algoritmo de força bruta foi utilizado um algoritmo de permutação que constrói todas as permutações existentes entre os vértices. O custo de cada aresta é obtido através de uma matriz $M_{n \times n}$. Desta forma, o valor contido na posição $M_{i,j}$ corresponde ao custo para ir do ponto *i* ao ponto *j*. Com esses dados, são testadas todas as permutações possíveis, e é retornada a solução ótima.

A seguir, o pseudocódigo que descreve a implementação feita para o algoritmo de força bruta.

**Força Bruta(*Ciclo[n], int pos*)**

1. SE *pos* $==$ *n*
2. Calcule o custo do ciclo
3. SE *custo* $<$ *custo mínimo*
4. ENTÃO *custo mínimo* $=$ *custo*
5. SENÃO FAÇA
6. PARA *j* $=$ *pos* até *n*
7. Troca (*ciclo[], pos, j*)
8. Força Bruta (*ciclo[], pos + 1*)
9. Troca (*ciclo[], pos, j*)

Como já foi descrito, o grafo da instância foi representado por uma matriz bidimensional $M_{n \times n}$, sendo *n* o número de vértices da instância. O vetor *ciclo* passado como parâmetro corresponde à sequência de vértices que serão permutados. O laço descrito entre os passos 6 e 9 faz o teste de todo o domínio de forma recursiva, através da fixação de uma posição do vetor por vez e da permutação das demais.

Os cálculos realizados no passo 2, com relação ao custo da instância, podem ser descritos da seguinte forma:

**Custo**($M[n][n]$, $ciclo[n]$)

1. double *custo* = 0
2. PARA *i* = 1 até *n-1*
3. FAÇA *custo* = *custo* + *M[ciclo[i]][ciclo[i + 1]]*
4. *custo* = *custo* + *M[ciclo[n]][ciclo[1]]*
5. RETORNE *custo*.

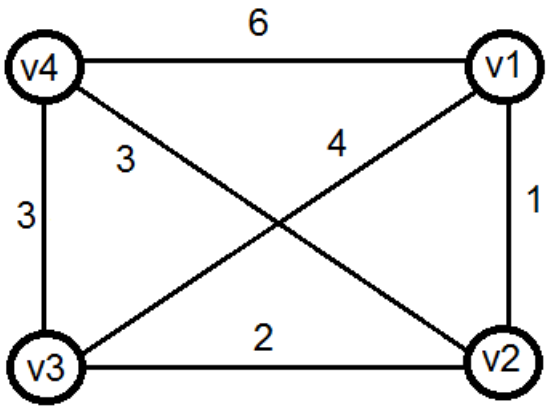
O laço descrito nos passos 2 e 3 do algoritmo faz o cálculo do custo do ciclo passado como parâmetro, através da matriz de distâncias. Já o passo 4 tem como objetivo calcular o custo do último vértice visitado até o vértice inicial.

Voltando ao algoritmo principal, os passos 7 e 9 são responsáveis pela permutação de fato dos vértices no ciclo. Para isso, foi utilizado um algoritmo simples de troca de posições em um vetor.

**Troca**(*ciclo*$[n]$, *int i, int j*)

1. int *aux* = *ciclo [i]*
2. *ciclo[i]* = *ciclo [j]*
3. *ciclo[j]* = *aux*

Desta forma, é possível realizar todas as trocas possíveis entre os vértices do ciclo, e assim testar todas as permutações possíveis para a instância de entrada. Através da Figura 3.1, pode-se observar um grafo que representa uma instância do TSP-Métrico e todas as permutações possíveis para a obtenção da solução.




v1-v2-v3-v4-v1 - custo = 12
v1-v2-v4-v3-v1 - custo = 11
v1-v3-v2-v4-v1 - custo = 15
v1-v3-v4-v2-v1 - custo = 11
v1-v4-v2-v3-v1 - custo = 15
v1-v4-v3-v2-v1 - custo = 12

**Figura 3.1:** Exemplo de abordagem por força bruta para o TSP-Métrico.

Como foi descrito no início dessa seção, o algoritmo de força bruta retorna sempre a solução ótima para o problema do TSP-Métrico, mas tem um custo computacional muito elevado, da

ordem de $\Theta(n!)$. Isso pode ser observado através da recursão apresentada na implementação acima descrita, que é necessária para a análise exaustiva de todas as soluções viáveis para o problema. No Capítulo 4, que traz os resultados obtidos pelo algoritmo nas instâncias testadas, é possível verificar de forma prática a limitação desse tipo de algoritmo.

As próximas duas seções deste capítulo trazem as propostas de algoritmos não exatos para a resolução do TSP-Métrico que, apesar de não garantirem a solução ótima, não possuem a limitação do custo computacional imposta pelo algoritmo de força bruta e encontram soluções viáveis em tempo aceitável mesmo para instâncias de tamanho grande.

## 3.2 Heurísticas de mínimo local

Uma heurística é uma estratégia que busca uma solução boa, mas sem um comprometimento com o ótimo ou garantia de desvio limitado. Qualquer estratégia pode ser considerada uma heurística, como soluções montadas aleatoriamente ou criadas de maneira arbitrária. Gerar um grande número de soluções e escolher a de menor custo é um exemplo de heurística. A cada momento, guarda-se o mínimo até que seja gerado um valor ainda menor. Outra possibilidade é varrer o domínio ao redor do melhor mínimo até o momento, buscando por uma solução na vizinhança do mínimo corrente.

### 3.2.1 Implementação

Neste trabalho foi usada uma estratégia heurística simples denominada *Single Swap*. Inicialmente, foi criada uma solução viável que consiste em um caminho formado por todos os vértices da instância em sequência, considerando a numeração a eles atribuída por meio das linhas da matriz de distâncias $M_{n\times n}$, que corresponde ao primeiro mínimo. A partir dessa solução, é feita uma troca randômica de duas arestas do ciclo. Se tal troca melhorar a solução, a troca é feita, caso contrário, é descartada. A condição de parada adotada foi $10n^2$ iterações sem que alguma troca de duas arestas trouxesse melhora ao mínimo corrente. Esta heurística pode ser melhorada com a troca de três, quatro ou mais arestas a cada passo. Estas alternativas, entretanto, podem aumentar muito o custo computacional da estratégia.

O pseudocódigo da implementação pode ser observado a seguir:

**Single Swap**($M[n][n]$)

1. Crie um vetor *ciclo* de tamanho *n*
2. PARA *i* = 1 até *n*

3. FAÇA *ciclo*[$i$] = *i*
4. int *it* = 0
5. ENQUANTO *it* $< 10n^2$
6. Gere dois valores inteiros aleatórios *x* e *y*
7. SE *x* > *y*
8. $x \leftrightarrow y$
9. SE *x* e *y* não forem iguais ou consecutivos
10. Verifique o custo da troca de arestas
11. SE custo da troca $<$ custo sem troca
12. Efetue a troca
13. *it* = 0
14. SENÃO *it* = *it* + 1
15. FIM ENQUANTO
16. Calcule o custo final de *ciclo* e guarde-o em *custo mínimo*
17. RETORNE o *ciclo mínimo*.

Nos primeiros passos desse algoritmo, uma solução inicial é obtida formando o ciclo com os vértices na ordem em que aparecem na matriz de entrada. Nesse caso, não há necessidade de calcular previamente o custo total do ciclo, uma vez que as comparações feitas para analisar se houve melhora na troca das arestas são locais, isto é, apenas entre as arestas trocadas. Para controlar o laço principal de execução é utilizada uma variável contadora, que é incrementada sempre que uma troca não é eficiente e zerada sempre que uma troca é feita. Desta forma, garante-se que o algoritmo só irá parar sua execução após $10n^2$ iterações sem melhora na solução encontrada.

Dentro do laço principal, são gerados dois valores inteiros aleatórios, *x* e *y*, a cada iteração. Esses valores determinam dois vértices do ciclo que serão usados como base para a troca das arestas. Os passos de verificação e suas eventuais ações resultantes são necessários para a troca efetiva das arestas, que é feita da seguinte forma:

**Troca arestas**(*ciclo*$C(V, E)$, *int x, int y*)

1. Remova as arestas $(v_x, v_{x+1})$ e $(v_y, v_{y+1})$ de *C*
2. Insira as arestas $(v_x, v_y)$ e $(v_{x+1}, v_{y+1})$ em *C*
3. Inverta a orientação das arestas que vão de $v_{x+1}$ até $v_y$

Este algoritmo faz a troca de duas arestas a partir de dois vértices não consecutivos, $v_x$ e $v_y$. O vértice $v_x$ passa a ter como sucessor o vértice $v_y$, enquanto é adicionada uma aresta entre os

vértices $v_{x+1}$ e $v_{y+1}$ e as arestas que anteriormente ligavam $v_x$ a $v_{x+1}$ e $v_y$ a $v_{y+1}$ são removidas. Desta forma, o ciclo continuará sendo hamiltoniano, mesmo considerando a orientação das arestas. O passo 3 é necessário pois o ciclo é guardado em uma estrutura do tipo vetor, onde existe implicitamente uma ordem no ciclo. Caso não fosse feita a inversão proposta não seria possível formar um ciclo no grafo, pois as arestas ficariam orientadas no sentido contrário ao esperado para sua formação. A Figura 3.2 ilustra como ficaria uma troca de duas arestas sem a inversão proposta.

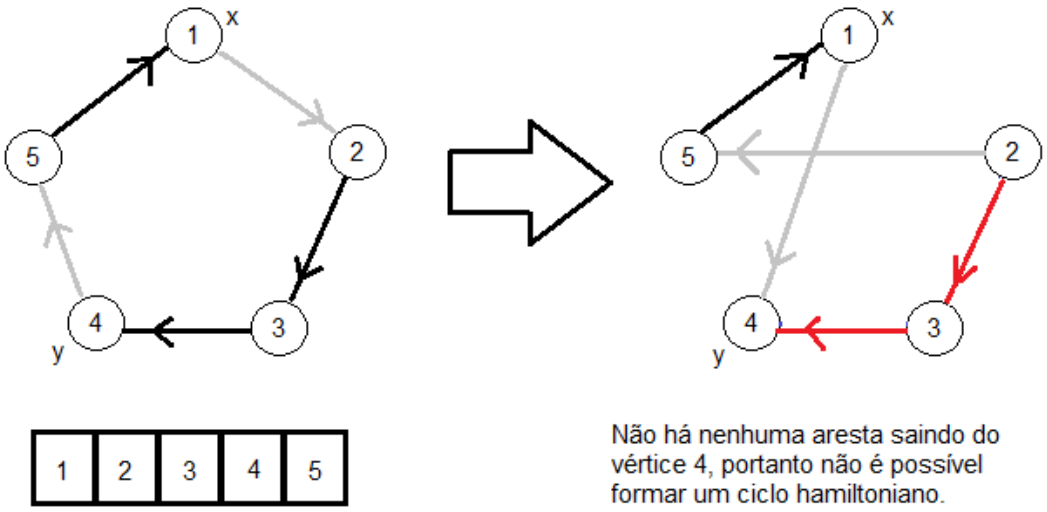



**Figura 3.2:** Exemplo de troca de arestas sem inversão, resultando em uma solução inválida.

Contudo, antes de realizar a troca entre as arestas, é necessário verificar, através da matriz de distâncias, se a soma dos custos das arestas $(v_x, v_y)$ e $(v_{x+1}, v_{y+1})$ é menor que a soma dos custos das arestas $(v_x, v_{x+1})$ e $(v_y, v_{y+1})$. Caso a troca proposta não diminua o custo da solução, isto é, a soma atual seja maior que a anterior, ela não é realizada, economizando assim tempo de processamento. A Figura 3.3 exibe um caso bem sucedido de troca de arestas.

Também é importante salientar que a troca de duas arestas consecutivas, analisada no passo 9 do algoritmo através dos valores de *x* e *y*, que são posições no vetor de inteiros que representa o ciclo, não resulta em alteração alguma na atual solução. A Figura 3.4 expressa o resultado da tentativa de trocar duas arestas consecutivas.

Essa heurística encontra uma solução viável para o TSP-Métrico sem nenhum comprometimento com o ótimo ou com tempo de convergência polinomial. A execução pode ficar presa em um mínimo local, isto é, fazer sucessivas trocas que melhoram a solução apenas localmente, sem uma preocupação com as eventuais trocas posteriores, e por isso chegar em uma situação

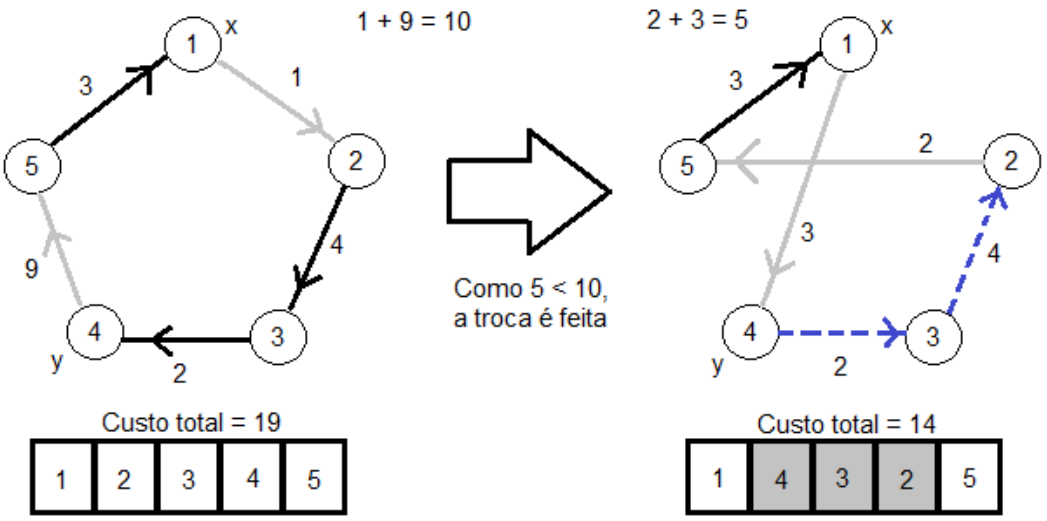



**Figura 3.3:** Exemplo de troca de arestas bem sucedida.

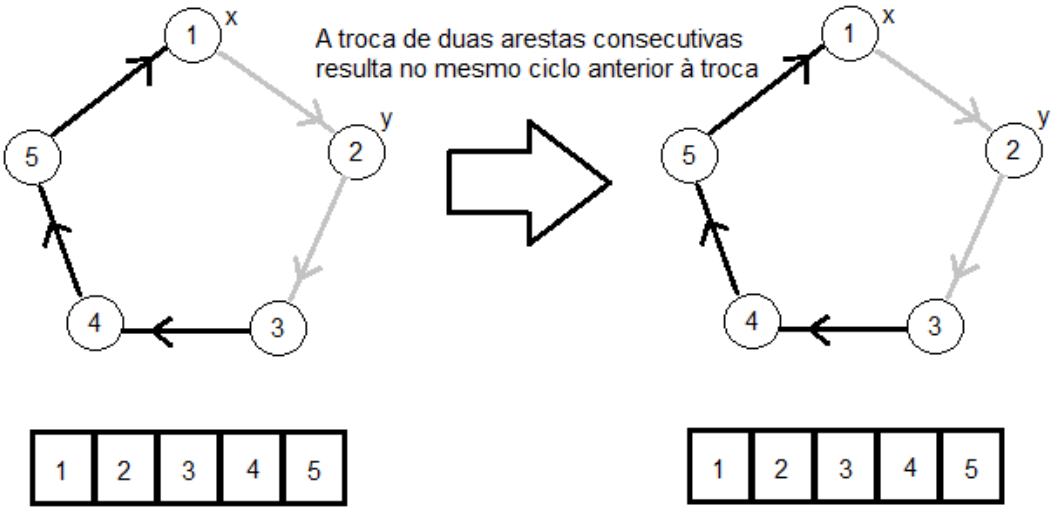



**Figura 3.4:** Exemplo de troca de arestas consecutivas.

na qual as trocas não conseguem melhorar a atual solução. Isso acontece porque essa técnica é uma heurística de mínimo local, e a única condição considerada em uma troca de arestas é a minimização imediata do custo atual. Também não há uma garantia que a heurística pare em tempo polinomial. Em tese, pode haver uma instância que permita um número exponencial de trocas entre o início e o mínimo local.

O tempo de execução do algoritmo é da ordem de $\Omega(n^2)$, pois o laço principal, compreendido entre os passos 5 e 15 do algoritmo, para após pelo menos $10n^2$ trocas. A troca acontece toda vez que as condições expressas pelos passos 9 e 11 forem satisfeitas, e seu tempo de execução é da ordem de $O(n)$, pois a inversão, que pode ser visualizada na Figura 3.3, depende do tamanho *n* do ciclo e necessita, em seu pior caso, de $n - 3$ iterações.

No Capítulo 4 é possível analisar de maneira prática, através da qualidade da solução obtida e do tempo de execução gasto para obtê-la, a eficiência dessa heurística e compará-la às demais técnicas propostas.

## 3.3 Algoritmos de aproximação

Algoritmos de aproximação são uma alternativa para se encontrar soluções com erro limitado para problemas para os quais não existem algoritmos polinomiais exatos. Esses algoritmos garantem um erro máximo provado matematicamente (Vazirani, 2003). Formalmente, um algoritmo é dito uma $\alpha$-Aproximação se retorna uma solução no máximo $\alpha$ vezes pior que o ótimo, para toda instância do problema. Sendo assim, uma 2-Aproximação garante um erro máximo de 100% em relação ao ótimo, enquanto uma 1,5-Aproximação garante um erro máximo de 50%, por exemplo, independentemente da instância.

Encontrar boas aproximações para problemas clássicos de otimização é um dos grandes desafios da área. Acredita-se que é possível chegar a uma 1,33-Aproximação para o TSP-Métrico, porém a melhor aproximação encontrada até hoje foi uma 1,5-Aproximação (Vazirani, 2003).

### 3.3.1 2-Aproximação para o TSP

Neste trabalho foi utilizada uma 2-Aproximação para o TSP Métrico. Para entender esta abordagem, é necessário conhecer alguns conceitos utilizados por esse método, como árvore geradora de custo mínimo (MST, do inglês *minimum-cost spanning tree*), ciclo e grafo Euleriano, definidos na Seção 2.2.

Dadas essas definições, é possível descrever o algoritmo 2-Aproximado para o TSP-Métrico, proposto em (Vazirani, 2003):

**2-Aproximação para o TSP-Métrico**

1. Encontrar uma MST, $T$ de $G$.
2. Dobrar todas as arestas da MST para obter um grafo Euleriano.
3. Encontrar um ciclo Euleriano, $\tau$, nesse grafo.
4. Retornar a trilha $C$ que visita os vértices de $G$ uma única vez na ordem em que eles aparecem em $\tau$.

Esse algoritmo garante um erro máximo de 100% em relação ao ótimo para o TSP Métrico (de Carvalho *et al.*, 2001). É utilizado, no passo 4 do algoritmo, um método semelhante ao *"short-cutting"* (Vazirani, 2003), que cria atalhos entre os vértices da trilha obtida a partir da MST. O custo dessa trilha não é aumentado, uma vez que para o TSP Métrico vale a desigualdade triangular. Se esse algoritmo fosse utilizado para solucionar um TSP geral, para o qual não é válida a desigualdade triangular, não seria possível garantir que o custo da trilha não seria aumentado durante esse processo.

O problema de encontrar uma MST é polinomial e pode ser resolvido com os algoritmos de Prim ou Kruskal em *O(|E| log |V|)* (Cormen *et al.*, 2001). Ambos são baseados no algoritmo *Generic MST*:

**Generic-MST ($G$ conexo)**

1. $A \leftarrow \emptyset$
2. ENQUANTO $A$ não forma uma MST
3. encontre uma aresta segura $uv$
4. $A \leftarrow A \cup \{uv\}$
5. RETORNE $A$.

$A$ é um subconjunto de arestas que pertencem a alguma árvore geradora mínima $T$.

A seguir, está descrito o algoritmo de Kruskal, que foi utilizado no algoritmo implementado neste trabalho.

**Algoritmo de Kruskal**

O algoritmo de Kruskal (Cormen *et al.*, 2001) é um algoritmo guloso que consiste em, dado um grafo $G(V, E)$, adicionar, a cada iteração, a aresta segura $uv$ de menor peso que conecta duas árvores de $G$ dentre todas as arestas possíveis. Desta forma, é possível que haja mais de uma árvore em $G$ durante a execução do algoritmo, mas no final do procedimento, graças à inserção de arestas seguras, haverá apenas uma árvore, que será a MST.

Segue abaixo a assinatura do algoritmo de conjuntos disjuntos (*Union-Find*) necessário ao algoritmo de Kruskal.

**MakeSet(x) -** função que cria um conjunto com o elemento $x$ e faz $x$ ser o representante desse conjunto em *O(1)*.

**Union(x, y) -** função que une os conjuntos $S_x$ e $S_y$ que contém os vértices $x$ e $y$ respectivamente. Isto é, cria um conjunto $S_x \cup S_y$ e elege um elemento, $x$ ou $y$, para ser o representante desse conjunto. Tal algoritmo tem complexidade *O(log n)*, com *n* = $|S_x|$ + $|S_y|$.

**FindSet(x) -** função que retorna o representante do conjunto ao qual o vértice $x$ pertence com complexidade *O(log n)*, onde *n* é o tamanho do conjunto.

**MST-Kruskal** ($G(V, E)$**,** $C$**)**

1. $A \leftarrow \emptyset$
2. PARA cada $v \in V$
3. FAÇA MakeSet($v$)
4. Ordene as arestas de $E$ em ordem não decrescente de pesos
5. PARA cada aresta $(u, v) \in E$, respeitando a ordenação
6. FAÇA SE FindSet($u$) $\neq$ FindSet($v$)
7. ENTÃO $A \leftarrow A \cup \{(uv)\}$
8. Union($u, v$)
9. RETORNE $A$.

Com esse algoritmo, é possível resolver o passo 1 do algoritmo 2-Aproximado para o TSP-Métrico.

A resolução dos passos 2 e 3 do algoritmo envolvem o problema de encontrar um ciclo Euleriano em um grafo conexo $G(V, E)$. Para isso, é necessário seguir o seguinte procedimento:

- A partir de um vértice $v \in V$, escolher um vértice adjacente $u$ a $v$ tal que a aresta $vu \in E$ que os conecta não seja uma ponte.
- Isso deve ser feito até que se chegue em um vértice que não possui arestas de saída que ainda não foram percorridas.

Desta forma, obtém-se o ciclo Euleriano $\tau$ utilizado no algoritmo 2-Aproximado para resolver o TSP-Métrico.

### 3.3.2 Método do atalho

O passo 4 do algoritmo da Subseção 3.3.1 é similar ao método do Atalho (do inglês *short-cutting*) citado em (Vazirani, 2003). Ele pode ser descrito da seguinte forma:

- percorrendo o ciclo Euleriano obtido, é possível obter um ciclo Hamiltoniano. Isso é feito através do método do atalho, que consiste em desprezar os caminhos que passam por vértices que já foram visitados, através da inserção de novas arestas (atalhos) que levem a vértices que ainda não foram adicionados ao ciclo.

A Figura 3.5 ilustra, de forma geral, o princípio do algoritmo 2-Aproximado aplicado ao TSP-Métrico para uma instância hipotética pequena de quatro vértices.

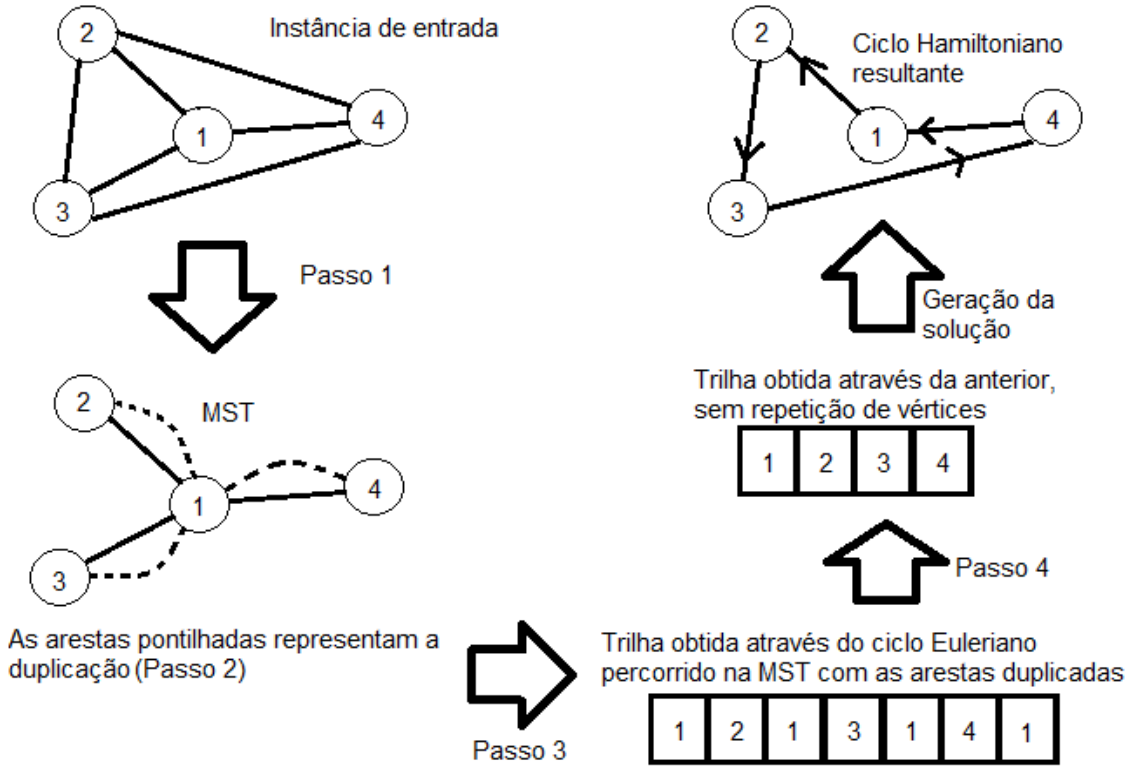



**Figura 3.5:** Princípio geral do algoritmo 2-Aproximado.

### 3.3.3 Implementação

A implementação do algoritmo aproximado feita neste trabalho segue o algoritmo descrito na Subseção 3.3.1. Para encontrar a MST referente à instância de entrada foi utilizado o algoritmo de Kruskal, também descrito anteriormente. As funções *MakeSet*, *Union* e *FindSet* foram implementadas da seguinte forma:

**MakeSet**(Vértice $v$, int $id$)

1. $v$.id = *id*
2. $v$.tamanho = 1
3. $v$.pai = *NULL*

**FindSet**(Vértice $v$)

1. SE $v$.pai == *NULL*
2. RETORNE $v$.id
3. SENÃO RETORNE **FindSet**($v$.pai)

**Union**(Vértice $v_1$, Vértice $v_2$)

1. Vértice $u_1$ = *find*($v_1$)
1. Vértice $u_2$ = *find*($v_2$)
1. SE $u_1$.tamanho < $u_2$.tamanho
2. $u_1$.pai = $u_2$
3. $u_2$.tamanho = $u_2$.tamanho + $u_1$.tamanho
4. $u_1$.tamanho = 0
5. SENÃO
6. $u_2$.pai = $u_1$
7. $u_1$.tamanho = $u_1$.tamanho + $u_2$.tamanho
8. $u_2$.tamanho = 0

Desta forma, é possível obter uma MST para a instância do TSP-Métrico de entrada. O algoritmo de Kruskal é da ordem de *O(|E| log |V|)* (Cormen *et al.*, 2001), sendo *E* o número de arestas e *V* o número de vértices. Nesta implementação, os atributos *id*, *tamanho* e o apontador *pai* são utilizados, respectivamente, para identificar o nó, definir o tamanho do conjunto ao qual o nó pertence, e qual o nó para o qual ele aponta. Inicialmente, todos os nós fazem parte de um conjunto formado apenas por eles mesmos e, por isso, apontam para *NULL*. Conforme as uniões ocorrem, esses apontadores passam a apontar para o nó chamado pai. Este pai, por sua vez, pode apontar para um outro nó, ou para *NULL*. Caso ele aponte para *NULL*, então ele é chamado pai do conjunto, que é o nó cujo *id* representa todo o conjunto. Utilizando esta estrutura, a união de dois conjuntos é feita em $O(log\ n)$, e a inserção também é feita em $O(log\ n)$. Com isso, o passo 1 do algoritmo 2-Aproximado está resolvido.

Para realizar os próximos dois passos, que consistem na criação e no percorrimento do ciclo Euleriano, foram utilizadas duas estruturas de *HashSet*, uma contendo as arestas da MST que ainda não foram visitadas, e outra contendo as arestas que já foram visitadas. Inicialmente, todas

as arestas estão contidas no *HashSet* correspondente às arestas que ainda não foram visitadas, ao passo que a outra estrutura é inicializada sem nenhuma aresta. Uma trilha é iniciada em um vértice da MST escolhido de forma arbitrária, que será chamado de estado inicial, e a partir deste vértice são procuradas arestas no *HashSet* de arestas ainda não visitadas que o tenham como extremidade.

Caso a busca retorne um resultado positivo, esta aresta é percorrida e o estado é atualizado, sendo a ele atribuído o vértice que se encontra na extremidade oposta da aresta e seu antecessor colocado na trilha percorrida. Essa operação é realizada nos *HashSet* através da remoção da aresta na estrutura que representa as não visitadas e da adição da mesma à estrutura das visitadas.

Caso contrário, a busca no *HashSet* das arestas não visitadas não encontra a aresta desejada, o que significa que ela já foi percorrida uma vez, e portanto está na segunda estrutura de *HashSet*. Neste caso, esta aresta é percorrida pela segunda vez e o estado é atualizado da mesma forma, sendo a ele atribuído o vértice que se encontra na extremidade oposta da aresta e seu antecessor colocado na trilha percorrida. Porém, essa operação representa a segunda visita à aresta em questão, e portanto ela não poderá mais ser visitada. Para garantir isso, ela é retirada também do *HashSet* das visitadas.

A trilha formada pelos vértices em ordem de visitação na MST tem o tamanho dado por $1 + 2|T|$, sendo $|T|$ o número de arestas da MST. Cada vez que um vértice representado pela variável *estado* é adicionado à trilha, um contador é incrementado. Quando esse contador atingir o tamanho da trilha, significa que foi encontrado um ciclo Euleriano, isto é, todas as arestas já foram visitadas exatas duas vezes, e as estruturas de *HashSet* estão ambas vazias.

O algoritmo abaixo descreve esse procedimento.

**Euleriano**($T(V, E)$)

1. Crie duas estruturas de *HashSet*, *naovisitadas* e *visitadas*
2. Adicione todas as arestas *e* de *E* em *naovisitadas*
3. int *estado = 0*
4. *visitadas = NULL*
5. A partir de um vértice arbitrário $v \in V$
6. ENQUANTO *naovisitadas* e *visitadas* não estiverem vazias
7. Procure *e* em *naovisitadas* que tenha $v$ como extremidade
8. SE encontrar
9. FAÇA *euleriano [estado] = e*
10. Remova *e* de *naovisitadas*
11. Adicione *e* a *visitadas*

12. FAÇA $v$ = extremidade oposta de *e*
13. *estado* = *estado* + 1
14. SENÃO
15. Procure *e* em *visitadas* que tenha $v$ como extremidade
16. SE encontrar
17. FAÇA *euleriano [estado]* = *e*
18. Remova *e* de *visitadas*
19. FAÇA $v$ = extremidade oposta de *e*
20. *estado* = *estado* + 1

Feito isso, já se tem a trilha $\tau$ utilizada no quarto passo do algoritmo inicialmente descrito, que nada mais é do que uma lista contendo os vértices na ordem em que foram visitados durante a formação do ciclo Euleriano. Agora, basta percorrê-la em ordem sequencial e adicionar os vértices, conforme forem aparecendo, em uma nova lista sem repetição de vértices que representa o ciclo hamiltoniano retornado como solução para o TSP-Métrico. Este passo, como já foi dito, é semelhante ao Método do Atalho (Vazirani, 2003).

O algoritmo abaixo representa uma implementação equivalente ao Método do Atalho para esse caso específico.

**Atalho**(Trilha $\tau[n]$)

1. int *it* = 0
2. Crie um vetor *hamiltoniano* com o tamanho da instância
3. PARA *i* = 0 até *n*
4. SE $\tau[i] \notin$ *hamiltoniano*
5. *hamiltoniano[it]* = $\tau[i]$
6. *it* = *it* + 1

A complexidade do algoritmo 2-Aproximado é da ordem de *O($n^2$ log $n$)*, sendo *n* o número de cidades da instância. Essa complexidade é devida ao algoritmo de Kruskal, uma vez que os demais passos acima descritos são todos da ordem *O(n)*, já que consistem apenas em percorrer uma lista de vértices de tamanho *n* por vez.

No Capítulo 4 é possível observar os resultados obtidos a partir do algoritmo 2-Aproximado e compará-los aos demais métodos de resolução do TSP, principalmente à heurística *Single Swap*, já que ambos não garantem a solução ótima.

## 3.4 Algoritmo *Branch & Bound*

Esta seção descreve um método exato que não foi implementado neste trabalho.

Um algoritmo de *Branch & Bound* retorna sempre a solução ótima. Trata-se, portanto, de um algoritmo não polinomial. Todas as possibilidades de solução são organizadas em uma árvore de enumeração. No caso aqui abordado, a árvore teria profundidade *"n"* e, partindo-se da raiz até as folhas, cada caminho formaria uma permutação de *"V"*. A Figura 3.6 mostra um exemplo de árvore de enumeração.

A estratégia de *Branch & Bound* consiste em varrer todo o domínio organizado em forma de árvore e detectar prematuramente que um ramo não precisa ser percorrido, pois este não melhorará o mínimo atual. Ao não percorrer um ramo, faz-se uma poda (*bound*). A poda é feita a partir de limitantes específicos do problema. Neste caso, pode-se considerar dois limitantes:

**Limitante Superior -** o limitante superior é a melhor solução encontrada até o momento pelo algoritmo.

**Limitante Inferior -** o limitante inferior é obtido através da resolução de subproblemas do TSP dentro do domínio da instância. Primeiramente fixa-se um caminho $(u, \ldots, v)$ entre dois vértices $u$ e $v$ da instância. Então é resolvido um subproblema denominado $TSP_{parcial}$, que consiste na permutação de vértices que não estão no caminho entre $u$ e $v$ para formar um ciclo hamiltoniano que contém o caminho fixo $(u, \ldots, v)$.

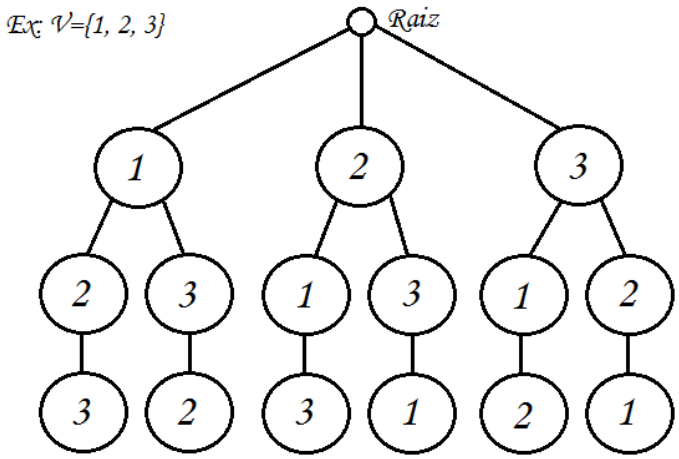



**Figura 3.6:** Exemplo de árvore de enumeração.

Desta forma, se é encontrada uma solução com custo maior que o limitante superior, o ramo dessa solução é podado. Se a soma dos custos do caminho fixo $(u, \ldots, v)$ e da solução obtida no $TSP_{parcial}$ for maior do que a melhor solução obtida, então é feita a poda pelo limitante inferior,

uma vez que o mínimo que se pode obter através desta ramificação já seria mais custoso do que a atual solução.

Esta técnica é interessante para resolver o TSP-Métrico, pois garante a solução ótima e consegue resolver instâncias maiores do que as resolvidas pelo algoritmo de força bruta. Contudo, a abordagem aqui proposta é apenas uma forma de fazer uso desta técnica. O algoritmo *Branch & Bound* pode ser implementado de diversas formas para resolver diversos problemas de otimização combinatória (Lawler e Wood., 1966), (Boyd e Mattingley, 2007) e já foi estudado durante décadas com o intuito de buscar otimizações cada vez mais específicas para minimizar o custo de encontrar a solução ótima para problemas NP-Difíceis, mesmo sendo em tempo não polinomial. Também foi utilizado o método *Branch & Bound* para resolver um caso particular do TSP-Métrico relacionado à sistemas de comunicação (Grimm, 1984).

CAPÍTULO

4

# Resultados

Neste capítulo serão apresentados os resultados obtidos a partir dos experimentos feitos com as técnicas propostas no Capítulo 3, através da comparação entre o tempo de execução e a qualidade da solução obtida. Nele também serão descritas as instâncias utilizadas e será feita uma análise sobre o TSP-Métrico aplicado à instâncias reais contendo as capitais brasileiras e o Distrito Federal.

## 4.1 Ambiente de execução

As técnicas propostas foram implementadas na linguagem Java e o computador no qual foram realizados os testes possui as seguintes configurações:

- Processador: AMD Athlon X2 Dual-Core QL-64 (2.1 GHz)
- Memória principal: 4.00 GB RAM
- Sistema Operacional: Windows 7 Ultimate 64-bits

## 4.2 Instâncias utilizadas

As instâncias do TSP-Métrico utilizadas para os testes foram obtidas em repositórios disponíveis na Internet (Heidelberg, 2008) (Cook, 2010). Foram escolhidas instâncias cujo ótimo é conhecido, para que houvesse um parâmetro de comparação para a qualidade das soluções obtidas pelos algoritmos utilizados.

Para uma análise progressiva do algoritmo de força bruta, foram derivadas instâncias menores a partir de outras obtidas nos repositórios acima citados. Isso foi feito através da escolha aleatória de alguns vértices das mesmas.

As instâncias escolhidas foram divididas em quatro grupos, de acordo com seu tamanho. A divisão foi feita da seguinte forma:

**Grupo 1 -** instâncias de até 20 vértices.

**Grupo 2 -** instâncias de 21 até 100 vértices.

**Grupo 3 -** instâncias de 101 até 1000 vértices.

**Grupo 4 -** instâncias com mais de 1000 vértices.

Também foi criado um quinto grupo, contendo as instâncias reais do TSP-Métrico aplicado às capitais brasileiras, que será tratado de forma mais detalhada na Seção 4.4.

A nomenclatura das instâncias segue o modelo:

*exemplo2010*

sendo *exemplo* o nome e 2010 o tamanho da instância. Geralmente o nome escolhido descreve algo relacionado à instância, como por exemplo a região geográfica de onde ela foi tirada, o problema que ela representa ou ainda as iniciais do nome de quem a propôs.

A seguir, estão descritas as 30 instâncias utilizadas neste trabalho, listadas em ordem de tamanho:

***teste4* -** instância obtida através da escolha aleatória de 4 vértices da instância *pr1002*, uma instância proposta por Padberg e Rinaldi que também é descrita nesta lista.

***teste5* -** instância obtida através da escolha aleatória de 5 vértices da instância *pr1002*.

***teste6* -** instância obtida através da escolha aleatória de 6 vértices da instância *burma14*.

***teste7* -** instância obtida através da escolha aleatória de 7 vértices da instância *pr1002*.

***amazonia7* -** instância que representa 7 capitais brasileiras localizadas na macro-região da Amazônia.

***teste8* -** instância obtida através da escolha aleatória de 8 vértices da instância *pr1002*.

***teste9* -** instância obtida através da escolha aleatória de 9 vértices da instância *pr1002*.

***nordeste9* -** instância que representa as 9 capitais brasileiras localizadas na macro-região Nordeste.

***teste10* -** instância obtida através da escolha aleatória de 10 vértices da instância *pr1002*.

***centrosul10* -** instância que representa 10 capitais brasileiras localizadas na macro-região Centro-Sul.

***teste11* -** instância obtida através da escolha aleatória de 11 vértices da instância *pr1002*.

***teste12* -** instância obtida através da escolha aleatória de 12 vértices da instância *pr1002*.

***teste13* -** instância obtida através da escolha aleatória de 13 vértices da instância *pr1002*.

***burma14* -** instância que representa 14 cidades da República da União de Myanmar, chamada localmente de Burma.

***brasil26* -** instância que representa 25 capitais brasileiras e o Distrito Federal.

***att48* -** instância proposta por Padberg e Rinaldi que representa 48 capitais dos Estados Unidos.

***berlin52* -** instância proposta por Groetschel que representa 52 localidades de Berlin.

***st70* -** instância proposta por Smith e Thompson que representa 70 cidades.

***pr76* -** instância proposta por Padberg e Rinaldi que representa 76 cidades.

***rd100* -** instância proposta por Gerhard Reinelt que representa 100 cidades.

***kroA100* -** instância proposta por Krolak, Felts e Nelson que representa 100 cidades.

***lin105* -** instância que representa um subproblema da instância *lin318*, que consiste em 318 pontos que surgiram em um aplicativo de perfuração.

***ch150* -** instância proposta por Churritz que representa 150 cidades.

***rd400* -** instância proposta por Gerhard Reinelt que representa 400 cidades.

***pcb442* -** instância proposta por Groetschel, Juenger e Gerhard Reinelt que representa um aplicativo de perfuração com 442 pontos.

***rat575* -** instância proposta por Pulleyblank que representa um *rattled grid* com 575 nós.

***rat783* -** instância proposta por Pulleyblank que representa um *rattled grid* com 783 nós.

***pr1002* -** instância proposta por Padberg e Rinaldi que representa 1002 cidades.

***u2319* -** instância proposta por Gerhard Reinelt que representa um aplicativo de perfuração com 2319 pontos.

***pr2392* -** instância proposta por Padberg e Rinaldi que representa 2392 cidades.

Estas foram as instâncias utilizadas neste trabalho. A Seção 4.3 traz os testes realizados com cada algoritmo implementado.

## 4.3 Resultados obtidos

Esta seção mostra os resultados obtidos por cada algoritmo para resolver as 30 instâncias descritas na Seção 4.2 através de tabelas e gráficos.

Para cada instância, foram realizadas 20 execuções com cada algoritmo capaz de resolvê-las. Como foi descrito no Capítulo 3, o algoritmo de força bruta não é polinomial e por isso não é capaz de resolver instâncias de tamanho grande em tempo viável. Portanto, este algoritmo só foi executado para as instâncias do Grupo 1, sendo seu tempo de execução estimado para os demais grupos. As soluções ótimas foram consideradas levando em conta o erro de aproximação do tipo *double* em Java.

Para as tabelas contidas nessa seção, deve-se considerar a seguinte legenda:

- FB: Força Bruta;
- SS: *Single Swap*;
- 2-Aprox: 2-Aproximado.

**Observação:** os valores indicados com (*) nas tabelas representam valores estimados.

### 4.3.1 Instâncias do Grupo 1

Os resultados obtidos para as instâncias do Grupo 1 estão descritos nesta subseção.

A Tabela 4.1 mostra o tempo de execução médio, em segundos, de cada algoritmo para cada instância do Grupo 1. Através da análise desses dados é possível observar como o algoritmo de força bruta se torna ineficiente mesmo para instâncias pequenas, como é o caso da instância *burma14*. Também pode-se observar que, a partir da instância de tamanho 12, o tempo de execução do algoritmo exato começa a se distanciar significativamente do tempo dos demais algoritmos. A Figura 4.1 traz um gráfico no qual é possível observar claramente a curva de crescimento do tempo de execução para o algoritmo de força bruta, que leva aproximadamente 15,8 minutos para resolver uma instância de tamanho 14, ao passo que os algoritmos não exatos levam 1 milissegundo.

As Tabelas 4.2 e 4.3 mostram que o algoritmo *Single Swap* foi capaz de encontrar a solução ótima para todas as instâncias do Grupo 1 de forma praticamente instantânea. Isso mostra que, mesmo para instâncias bem pequenas, o algoritmo de força bruta não é sempre a melhor opção, apesar de garantir a solução ótima.

**Tabela 4.1:** Tempo médio de execução para instâncias do Grupo 1.

| Tempo de Execução Médio (s) | | | |
|---|---|---|---|
| Instâncias | FB | SS | 2-Aprox |
| teste4 | 0,001 | 0,001 | 0,001 |
| teste5 | 0,001 | 0,001 | 0,001 |
| teste6 | 0,001 | 0,001 | 0,001 |
| teste7 | 0,001 | 0,001 | 0,001 |
| teste8 | 0,001 | 0,001 | 0,001 |
| teste9 | 0,005 | 0,001 | 0,001 |
| teste10 | 0,046 | 0,001 | 0,001 |
| teste11 | 0,475 | 0,001 | 0,001 |
| teste12 | 5,892 | 0,001 | 0,001 |
| teste13 | 71,565 | 0,001 | 0,001 |
| burma14 | 948,242 | 0,001 | 0,001 |

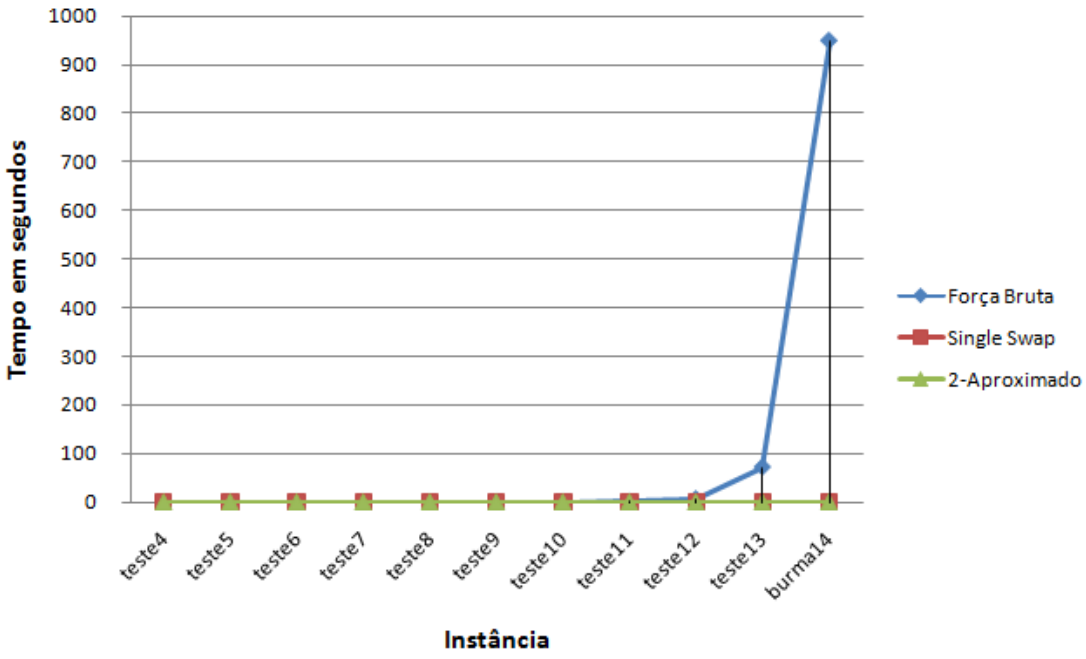



**Figura 4.1:** Tempos de execução para as instâncias do Grupo 1.

**Tabela 4.2:** Custo das soluções para as instâncias do Grupo 1.

| Custo da Solução Obtida | | | | | | |
|---|---|---|---|---|---|---|
| Instâncias | OPT | FB | SS Melhor | SS Médio | SS Pior | 2-Aprox |
| teste4 | **3940,065** | **3940,065** | **3940,065** | **3940,065** | **3940,065** | **3940,065** |
| teste5 | **8237,726** | **8237,726** | **8237,726** | **8237,726** | **8237,726** | **8237,726** |
| teste6 | **21,966** | **21,966** | **21,966** | **21,966** | **21,966** | 23,178 |
| teste7 | **17076,239** | **17076,239** | **17076,239** | **17076,239** | **17076,239** | 18854,002 |
| teste8 | **28649,993** | **28649,993** | **28649,993** | **28649,993** | **28649,993** | 32605,940 |
| teste9 | **34108,544** | **34108,544** | **34108,544** | 34386,125 | 34538,527 | 42390,893 |
| teste10 | **37265,547** | **37265,547** | **37265,547** | **37265,547** | **37265,547** | 46318,234 |
| teste11 | **37298,239** | **37298,239** | **37298,239** | 37616,285 | 39527,808 | 46061,899 |
| teste12 | **46153,424** | **46153,424** | **46153,424** | 46356,050 | 47166,553 | **46153,424** |
| teste13 | **47571,628** | **47571,628** | **47571,628** | 49484,895 | 50958,268 | 65719,626 |
| burma14 | **30,879** | **30,879** | **30,879** | 31,457 | 32,177 | 37,969 |

**Tabela 4.3:** Desvio das soluções obtidas para as instâncias do Grupo 1.

| Desvio da Solução Obtida em Relação ao Ótimo | | | | | | |
|---|---|---|---|---|---|---|
| Instâncias | OPT | FB | SS Melhor | SS Médio | SS Pior | 2-Aprox |
| teste4 | **3940,065** | **0,00%** | **0,00%** | **0,00%** | **0,00%** | **0,00%** |
| teste5 | **8237,726** | **0,00%** | **0,00%** | **0,00%** | **0,00%** | **0,00%** |
| teste6 | **21,966** | **0,00%** | **0,00%** | **0,00%** | **0,00%** | 5,52% |
| teste7 | **17076,239** | **0,00%** | **0,00%** | **0,00%** | **0,00%** | 10,41% |
| teste8 | **28649,993** | **0,00%** | **0,00%** | **0,00%** | **0,00%** | 13,81% |
| teste9 | **34108,544** | **0,00%** | **0,00%** | 0,81% | 1,26% | 24,28% |
| teste10 | **37265,547** | **0,00%** | **0,00%** | **0,00%** | **0,00%** | 24,29% |
| teste11 | **37298,239** | **0,00%** | **0,00%** | 0,85% | 5,98% | 23,50% |
| teste12 | **46153,424** | **0,00%** | **0,00%** | 0,44% | 2,20% | **0,00%** |
| teste13 | **47571,628** | **0,00%** | **0,00%** | 4,02% | 7,12% | 38,15% |
| burma14 | **30,879** | **0,00%** | **0,00%** | 1,88% | 4,20% | 22,96% |
| Média Total | | **0,00%** | **0,00%** | 0,73% | 1,89% | 14,81% |

De acordo com o gráfico da Figura 4.2 é possível verificar que mesmo considerando o desvio médio, e não só o desvio da melhor solução encontrada, o algoritmo *Single Swap* se mostrou mais competitivo que o 2-Aproximado, uma vez que o primeiro obteve resultados significativamente melhores no mesmo tempo de execução que o segundo. É interessante notar que o algoritmo 2-Aproximado foi capaz de encontrar a solução ótima em 3 instâncias, sendo uma delas uma das maiores deste grupo, o que mostra um bom desempenho considerando o erro máximo garantido de 100% (de Carvalho *et al.*, 2001).

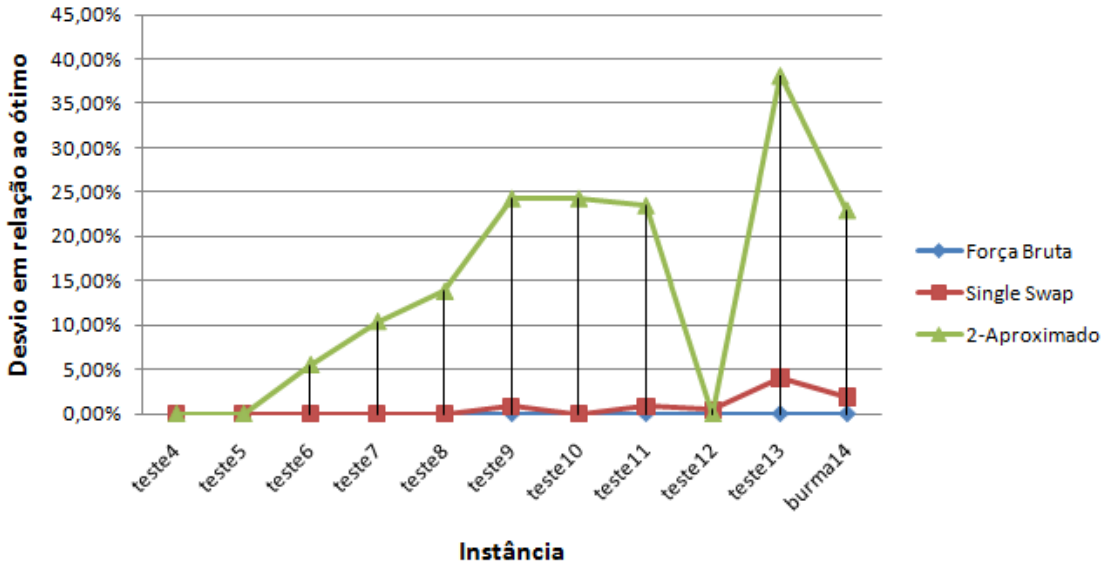



**Figura 4.2:** Desvio médio das soluções obtidas para as instâncias do Grupo 1.

### 4.3.2 Instâncias do Grupo 2

Os resultados obtidos para as instâncias do Grupo 2 estão descritos nesta subseção.

Através da análise dos dados da Tabela 4.4 é possível observar como o algoritmo de força bruta é inviável para as instâncias do Grupo 2. Mesmo para a instância *att48*, que é a menor do conjunto, o algoritmo levaria cerca de $10^{53}$ segundos para obter a solução ótima. Para expressar o quanto esse tempo é elevado, $10^{53}$ segundos equivalem a aproximadamente $10^{44}$ séculos. Também pode-se observar que os algoritmos *Single Swap* e 2-Aproximado já apresentam uma pequena diferença entre eles no tempo de execução, apesar desta ser insignificante na prática, já que é da ordem de milissegundos. A Figura 4.3 traz um gráfico no qual é possível observar essa diferença em uma escala mais detalhada. Os tempos estimados para o algoritmo de força bruta

**Tabela 4.4:** Tempo médio de execução para instâncias do Grupo 2.

| Tempo de Execução Médio (s) | | | |
|---|---|---|---|
| Instâncias | FB | SS | 2-Aprox |
| att48 | $10^{53}$* | 0,008 | 0,001 |
| berlin52 | $10^{59}$* | 0,009 | 0,002 |
| st70 | $10^{92}$* | 0,015 | 0,002 |
| pr76 | $10^{103}$* | 0,021 | 0,003 |
| rd100 | $10^{149}$* | 0,033 | 0,005 |
| kroA100 | $10^{149}$* | 0,033 | 0,004 |

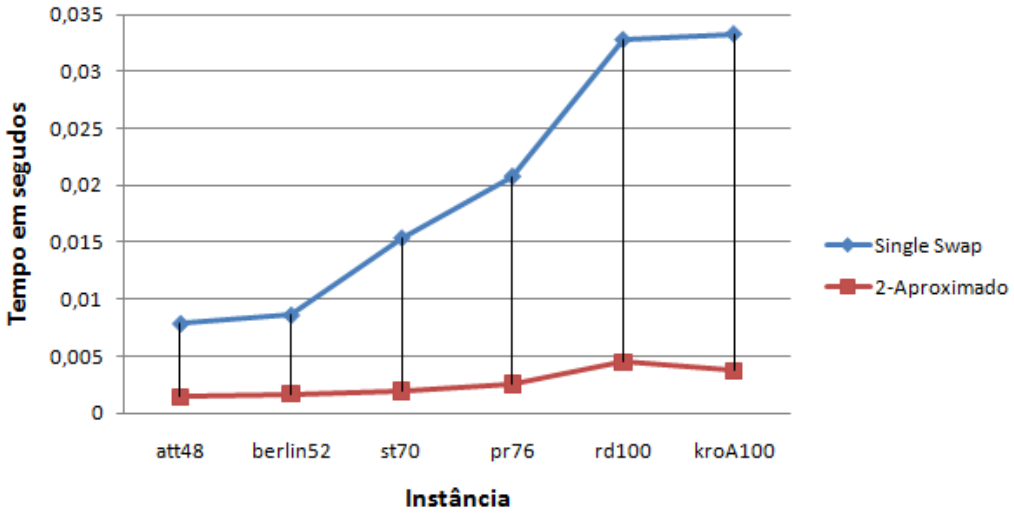



**Figura 4.3:** Tempos de execução para as instâncias do Grupo 2.

não foram exibidos no gráfico porque são infinitamente maiores que os tempos dos algoritmos não exatos, sendo impossível compará-los na mesma escala.

**Tabela 4.5:** Custo das soluções para as instâncias do Grupo 2.

| Custo da Solução Obtida | | | | | |
|---|---|---|---|---|---|
| Instâncias | OPT | SS Melhor | SS Médio | SS Pior | 2-Aprox |
| att48 | **33523,7085** | 34453,65 | 35426,93 | 37175,46 | 42727,9 |
| berlin52 | **7544,3659** | 7807,509 | 8289,948 | 8787,612 | 10056,02 |
| st70 | **678,597452** | 709,4464 | 734,2063 | 765,9785 | 855,2513 |
| pr76 | **108159,438** | 115065,5 | 122856,6 | 133930,4 | 140847,4 |
| rd100 | **7910,39621** | 7998,107 | 8783,962 | 9339,464 | 10753,31 |
| kroA100 | **21282** | 22031,67 | 22931,04 | 24684,02 | 28555,89 |

**Tabela 4.6:** Desvio das soluções obtidas para as instâncias do Grupo 2.

| Desvio da Solução Obtida em Relação ao Ótimo | | | | | |
|---|---|---|---|---|---|
| Instâncias | OPT | SS Melhor | SS Médio | SS Pior | 2-Aprox |
| att48 | 33523,709 | 2,77% | 5,68% | 10,89% | 27,46% |
| berlin52 | 7544,366 | 3,49% | 9,88% | 16,48% | 33,29% |
| st70 | 678,597 | 4,55% | 8,19% | 12,88% | 26,03% |
| pr76 | 108159,438 | 6,39% | 13,59% | 23,83% | 30,22% |
| rd100 | 7910,396 | 1,11% | 11,04% | 18,07% | 35,94% |
| kroA100 | 21282,000 | 3,52% | 7,75% | 15,99% | 34,18% |
| Média Total | | 3,64% | 9,36% | 16,35% | 31,19% |

As Tabelas 4.5 e 4.6 mostram que os algoritmos não exatos foram capazes de encontrar soluções viáveis para todas as instâncias, apesar de não obterem a solução ótima para nenhuma delas. O algoritmo *Single Swap* obteve bons resultados considerando a melhor solução por ele encontrada. O desvio médio de 9,36%, verificado ao longo das 20 execuções, reflete o fator aleatório do algoritmo, uma vez que o desvio médio para as melhores soluções encontradas foi de 3,64%, ao passo que o das piores foi 16,35%. Mais uma vez, o algoritmo 2-Aproximado não se mostrou competitivo se comparado ao *Single Swap*, obtendo 31,19% de desvio médio em relação às soluções ótimas conhecidas. Contudo, este ainda é um valor muito abaixo do erro máximo de 100% por ele garantido (de Carvalho *et al.*, 2001).

No gráfico da Figura 4.4 é possível verificar de forma mais clara a diferença da qualidade das soluções obtidas para cada instância entre os algoritmos *Single Swap* e 2-Aproximado.

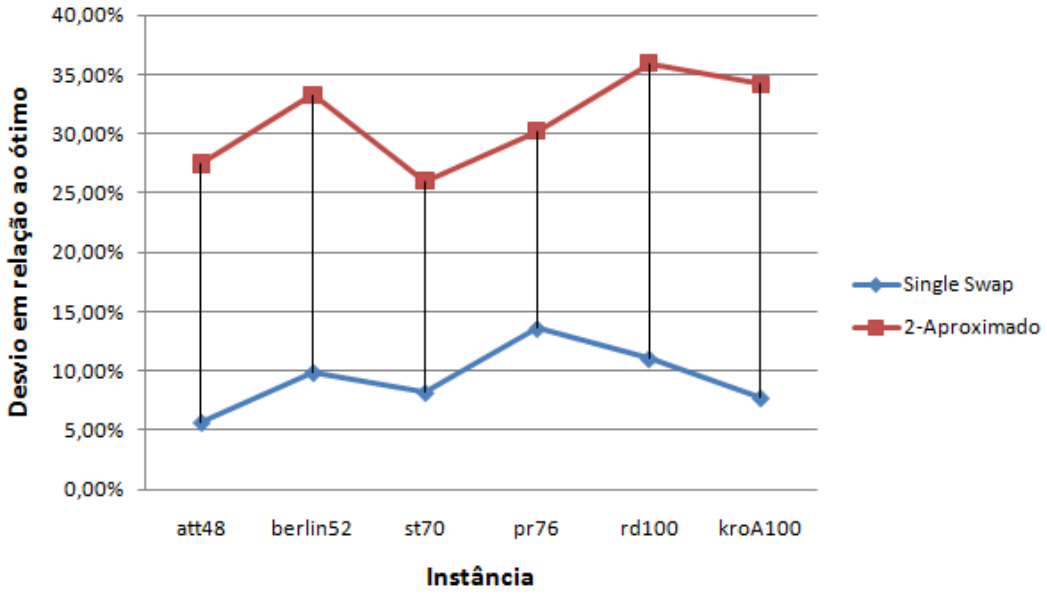



**Figura 4.4:** Desvio médio das soluções obtidas para as instâncias do Grupo 2.

### 4.3.3 Instâncias do Grupo 3

Os resultados obtidos para as instâncias do Grupo 3 estão descritos nesta subseção.

**Tabela 4.7:** Tempo médio de execução para instâncias do Grupo 3.

| Tempo de Execução Médio (s) | | | |
|---|---|---|---|
| Instâncias | FB | SS | 2-Aprox |
| lin105 | $10^{160}$* | 0,035 | 0,003 |
| ch150 | $10^{254}$* | 0,075 | 0,007 |
| rd400 | - | 0,769 | 0,088 |
| pcb442 | - | 0,869 | 0,112 |
| rat575 | - | 1,594 | 0,212 |
| rat783 | - | 3,403 | 0,396 |

Através da análise dos dados da Tabela 4.7 é possível observar que, para as instâncias do Grupo 3, é difícil até estimar o tempo de execução para o algoritmo de força bruta. A maior es-

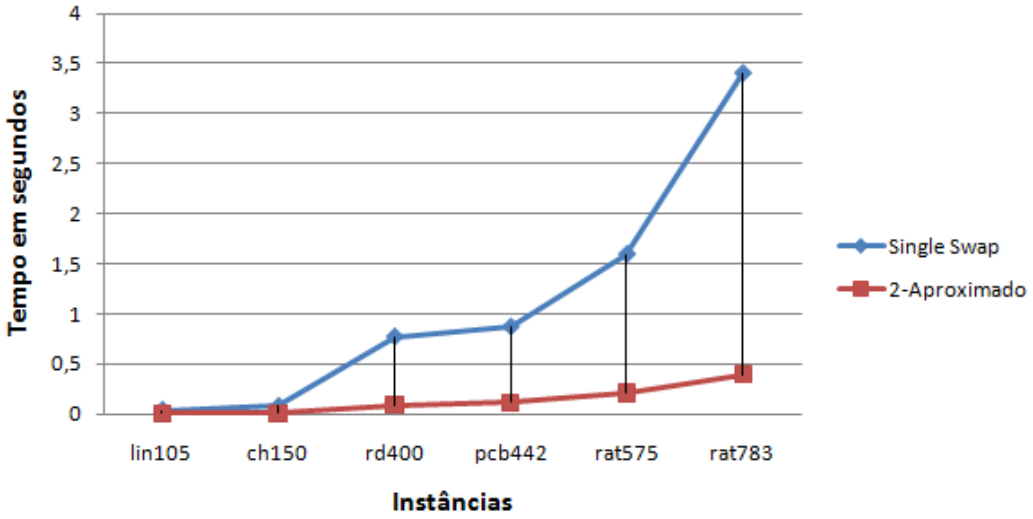



**Figura 4.5:** Tempos de execução para as instâncias do Grupo 3.

timativa feita foi para a instância *ch150*, que é apenas a segunda menor do conjunto, e foi obtido um valor extremamente elevado, da ordem de $10^{254}$ segundos. A dificuldade de se calcular o fatorial para as instâncias maiores torna inviável o cálculo do tempo estimado. Neste grupo de instâncias é possível observar que os algoritmos *Single Swap* e 2-Aproximado começam a apresentar uma diferença mais acentuada entre seus tempos de execução, apesar desta ainda não ser muito sensível na prática, já que é da ordem de 3 segundos para a maior instância do conjunto. A Figura 4.5 traz um gráfico no qual é possível observar essa diferença em uma escala mais detalhada. A partir da instância *rat575* a diferença entre os tempos começa a ficar mais notável.

**Tabela 4.8:** Custo das soluções para as instâncias do Grupo 3.

| | Custo da Solução Obtida | | | | |
|---|---|---|---|---|---|
| Instâncias | OPT | SS Melhor | SS Médio | SS Pior | 2-Aprox |
| lin105 | **14382,996** | 14739,554 | 15867,494 | 16550,699 | 21845,762 |
| ch150 | **6532,281** | 6899,363 | 7302,431 | 7610,452 | 9206,781 |
| rd400 | **15281,000** | 16738,461 | 17173,740 | 17786,126 | 21162,745 |
| pcb442 | **50783,548** | 56153,981 | 57811,613 | 59186,308 | 71734,875 |
| rat575 | **6773,000** | 7471,791 | 7623,431 | 7762,588 | 9568,653 |
| rat783 | **8806,000** | 9803,498 | 10012,728 | 10225,226 | 12551,389 |

As Tabelas 4.8 e 4.9 mostram que os algoritmos não exatos foram capazes de encontrar soluções viáveis para todas as instâncias, apesar de não obterem a solução ótima para ne-

nhuma delas. O algoritmo *Single Swap* obteve resultados satisfatórios para as instâncias do conjunto, principalmente para as duas menores, *lin105* e *ch150*. O maior desvio para o algoritmo considerando as melhores soluções foi de 11,33% para a instância *rat783*, que é a maior instância do conjunto. Como também foi observado nos conjuntos anteriores, o algoritmo 2-Aproximado não se mostrou competitivo se comparado ao *Single Swap*, obtendo um desvio médio de 42,73%. Uma observação sobre este conjunto é que, pela primeira vez, o algoritmo 2-Aproximado obteve uma solução com erro superior a 50% para uma instância (51,89% para *lin105*), que ainda é praticamente metade do erro máximo garantido.

No gráfico da Figura 4.6 é possível verificar a diferença da qualidade das soluções obtidas para cada instância entre os algoritmos *Single Swap* e 2-Aproximado. Além disso, pode-se observar o aumento do desvio médio obtido pelo *Single Swap* com o aumento do tamanho das instâncias deste grupo em particular.

**Tabela 4.9:** Desvio das soluções obtidas para as instâncias do Grupo 3.

| Desvio da Solução Obtida em Relação ao Ótimo | | | | | |
|---|---|---|---|---|---|
| Instâncias | OPT | SS Melhor | SS Médio | SS Pior | 2-Aprox |
| lin105 | 14382,996 | 2,48% | 10,32% | 15,07% | 51,89% |
| ch150 | 6532,281 | 5,62% | 11,79% | 16,51% | 40,94% |
| rd400 | 15281,000 | 9,54% | 12,39% | 16,39% | 38,49% |
| pcb442 | 50783,548 | 10,58% | 13,84% | 16,55% | 41,26% |
| rat575 | 6773,000 | 10,32% | 12,56% | 14,61% | 41,28% |
| rat783 | 8806,000 | 11,33% | 13,70% | 16,12% | 42,53% |
| Média Total | | 8,31% | 12,43% | 15,87% | 42,73% |

### 4.3.4 Instâncias do Grupo 4

Os resultados obtidos para as instâncias do Grupo 4 estão descritos nesta subseção.

A análise dos dados da Tabela 4.10 permite observar que, para as instâncias do Grupo 4, a diferença entre os tempos de execução dos algoritmos *Single Swap* e 2-Aproximado se torna significante. Da instância *pr1002* para a instância *u2319*, que possuem uma diferença de tamanho superior a 1000 vértices, o tempo de execução do algoritmo *Single Swap* aumentou consideravelmente, tornando-se aproximadamente 9 vezes maior que o tempo gasto pelo 2-Aproximado para obter a solução para a mesma instância. Desta forma, é possível verificar que, apesar de não encontrar soluções com um desvio competitivo em relação ao ótimo, o algo-

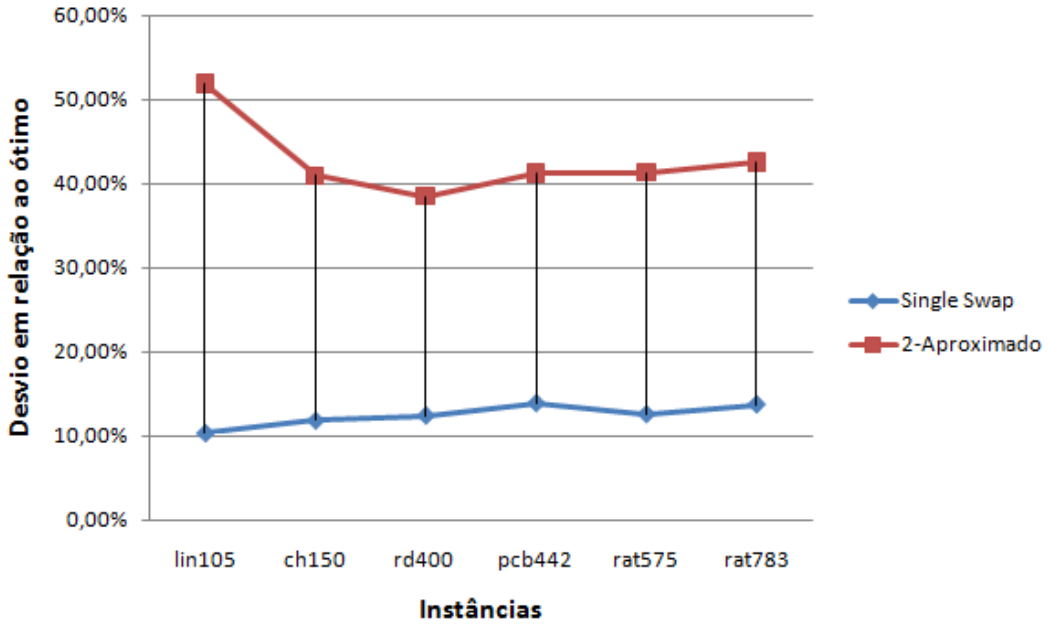



**Figura 4.6:** Desvio médio das soluções obtidas para as instâncias do Grupo 3.

**Tabela 4.10:** Tempo médio de execução para instâncias do Grupo 4.

| Tempo de Execução Médio (s) | | |
|---|---|---|
| Instâncias | SS | 2-Aprox |
| pr1002 | 5,486 | 0,664 |
| u2319 | 37,239 | 4,743 |
| pr2392 | 30,904 | 5,123 |

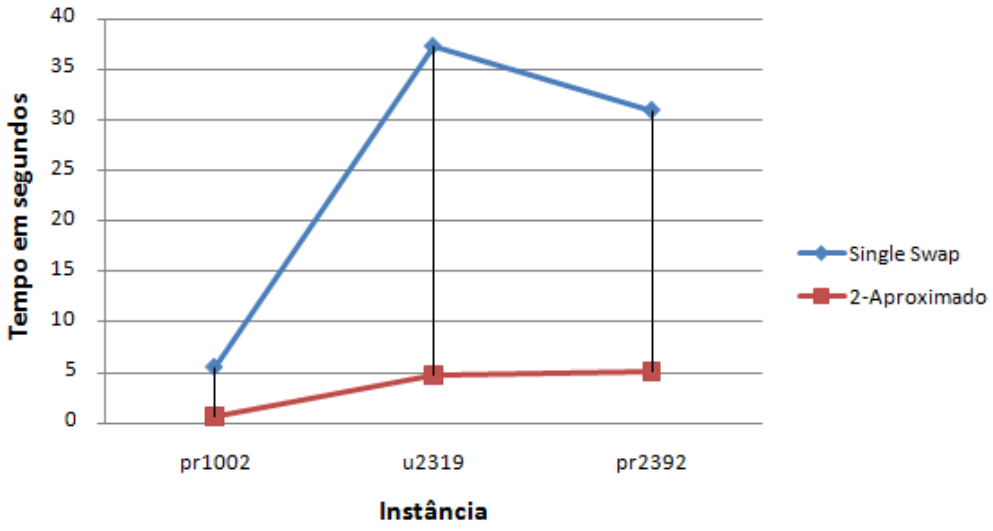



**Figura 4.7:** Tempos de execução para as instâncias do Grupo 4.

ritmo aproximado é executado em um tempo muito inferior se comparado à heurística, sendo uma boa alternativa principalmente para instâncias grandes e para situações nas quais a preocupação com o tempo de execução é maior que a preocupação com a qualidade da solução.

Outro fato que merece destaque nesse conjunto é a diferença entre o tempo de execução do *Single Swap* para as instâncias *u2319* e *pr2392*. Apesar da segunda instância ser maior, o tempo levado pelo algoritmo para encontrar sua solução foi cerca de 6,3 segundos menor que o tempo levado para a obtenção da solução para a primeira. Isso aconteceu porque a solução ótima $S_{OPT}$ da instância *pr2392* consiste no percorrimento de seus vértices na ordem de numeração, isto é, $S_{OPT} = (v_1, v_2, \ldots, v_{2391}, v_{2392})$. Desta forma, nenhuma troca de arestas foi feita, pois a solução inicial coincidentemente já era a solução ótima. Isso também ressalta como a escolha da solução inicial pode fazer a diferença na construção de uma heurística. A Figura 4.7 traz um gráfico no qual é possível observar a diferença considerável entre os tempos de execução dos dois algoritmos comparados.

As Tabelas 4.11 e 4.12 mostram que, assim como para os dois conjuntos anteriores, os algoritmos não exatos foram capazes de encontrar soluções viáveis para todas as instâncias, mas desta vez a heurística foi capaz de encontrar o ótimo para uma instância, como já foi citado no parágrafo anterior. O algoritmo *Single Swap* também obteve resultados satisfatórios para as demais instâncias do conjunto, tendo um desvio médio de 10,70% para a instância *pr1002* e de 9,15% para a instância *u2319*. Como também foi observado nos conjuntos anteriores, o algoritmo 2-Aproximado não foi capaz de obter resultados competitivos se comparado ao *Single*

**Tabela 4.11:** Custo das soluções para as instâncias do Grupo 4.

| Custo da Solução Obtida | | | | | |
|---|---|---|---|---|---|
| Instâncias | OPT | SS Melhor | SS Médio | SS Pior | 2-Aprox |
| pr1002 | **259066,663** | 282943,778 | 286791,988 | 293604,533 | 354522,671 |
| u2319 | **234256,000** | 254948,061 | 255681,205 | 256292,021 | 346178,598 |
| pr2392 | **378062,826** | **378062,826** | **378062,826** | **378062,826** | 536123,554 |

**Tabela 4.12:** Desvio das soluções obtidas para as instâncias do Grupo 4.

| Desvio da Solução Obtida em Relação ao Ótimo | | | | | |
|---|---|---|---|---|---|
| Instâncias | OPT | SS Melhor | SS Médio | SS Pior | 2-Aprox |
| pr1002 | **259066,663** | 9,22% | 10,70% | 13,33% | 36,85% |
| u2319 | **234256,000** | 8,83% | 9,15% | 9,41% | 47,78% |
| pr2392 | **378062,826** | **0,00%** | **0,00%** | **0,00%** | 41,81% |
| Média Total | | 6,02% | 6,62% | 7,58% | 42,14% |

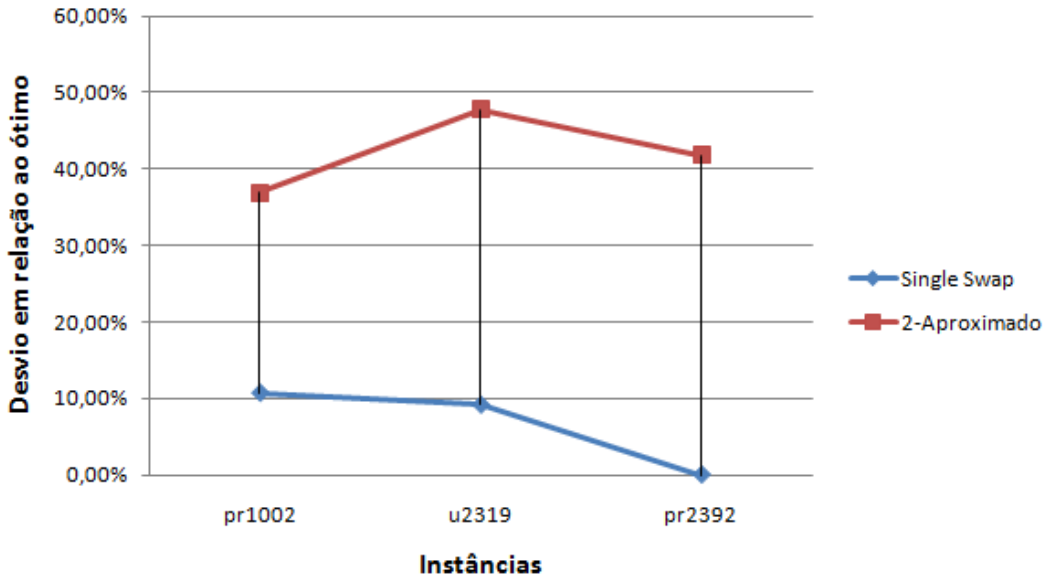



**Figura 4.8:** Desvio médio das soluções obtidas para as instâncias do Grupo 4.

*Swap*, obtendo um desvio médio de 42,14% em relação ao ótimo. Mais uma vez, destaca-se a distância do erro médio comparado ao erro máximo garantido pelo algoritmo mesmo para instâncias grandes, com mais de 2000 vértices.

No gráfico da Figura 4.8 é possível verificar a diferença da qualidade das soluções obtidas para cada instância entre os algoritmos *Single Swap* e 2-Aproximado. Além disso, pode-se observar a obtenção da solução ótima para a instância *pr2392*.

## 4.4 Caixeiro viajante entres as capitais brasileiras

Esta seção trata do grupo de instâncias brasileiras e dos resultados obtidos com os experimentos para o mesmo. Os dados das instâncias deste grupo foram obtidos através do site do Departamento Nacional de Infraestrutura de Transportes (DNIT) (DNIT, 2010), que fornece uma tabela com as menores distâncias de centro a centro entre as principais cidades brasileiras, considerando rodovias federais, estaduais e municipais pavimentadas. Esse grupo de instâncias é bastante interessante para a década que se inicia, uma vez que o Brasil irá receber uma grande quantidade de turistas em função dos eventos esportivos que serão sediados no país, como a Copa do Mundo FIFA em 2014 e as Olimpíadas de 2016. A Figura 4.9 mostra o mapa político do Brasil com a localização das capitais brasileiras e do distrito federal.


**Figura 4.9:** Mapa Político do Brasil.

Nas instâncias desse grupo, foram consideradas 26 cidades brasileiras, sendo 25 das 26 capitais estaduais e o Distrito Federal. A cidade de Macapá, capital do Amapá, não foi considerada nas instâncias, uma vez que não há como chegar até ela a partir das demais cidades considerando apenas rodovias federais, estaduais e municipais pavimentadas devido ao grande número de rios que cortam o estado. Para as instâncias *amazonia7*, *nordeste9* e *centrosul10*, a divisão de cidades foi feita segundo a divisão geoeconômica brasileira em macro-regiões, como mostrado na Figura 4.10. O critério de escolha para a divisão das cidades entre as instâncias considerando a capital de um estado que está localizado em mais de uma macro-região não foi a localização geográfica da cidade, mas sim sua localização geoeconômica. Desta forma, a divisão das cidades pelas macro-regiões representadas pelas instâncias foi feita da seguinte forma:

***amazonia7* -** Belém, Boa Vista, Cuiabá, Manaus, Palmas, Porto Velho e Rio Branco.

***nordeste9* -** Aracaju, Fortaleza, João Pessoa, Maceió, Natal, Recife, Salvador, São Luís e Teresina.

***centrosul10* -** Belo Horizonte, Brasília, Campo Grande, Curitiba, Florianópolis, Goiânia, Porto Alegre, Rio de Janeiro, São Paulo e Vitória.

***brasil26* -** união das 3 instâncias anteriores.

### 4.4.1 Resultados para as instâncias do Brasil

Os resultados obtidos para as instâncias brasileiras estão descritos nesta subseção.

**Tabela 4.13:** Tempo médio de execução para instâncias brasileiras.

| Tempo de Execução Médio (s) | | | |
|---|---|---|---|
| Instâncias | FB | SS | 2-Aprox |
| amazonia7 | 0,001 | 0,001 | 0,001 |
| nordeste9 | 0,005 | 0,001 | 0,001 |
| centrosul10 | 0,043 | 0,001 | 0,001 |
| brasil26 | $10^{18}$* | 0,002 | 0,001 |

A análise dos dados da Tabela 4.13 mostra que o algoritmo de força bruta é capaz de resolver a maioria das instâncias deste grupo em tempo viável. Os subproblemas *amazonia7*, *nordeste9* e *centrosul10*, derivados de *brasil26*, foram resolvidos de forma exata na ordem de milissegundos. Já a instância maior, que representa as capitais brasileiras, levaria cerca de $10^{18}$ segundos para ser resolvida, o que equivale a aproximadamente $10^9$ séculos. Assim como nos


**Figura 4.10:** Divisão do Brasil em macro-regiões.

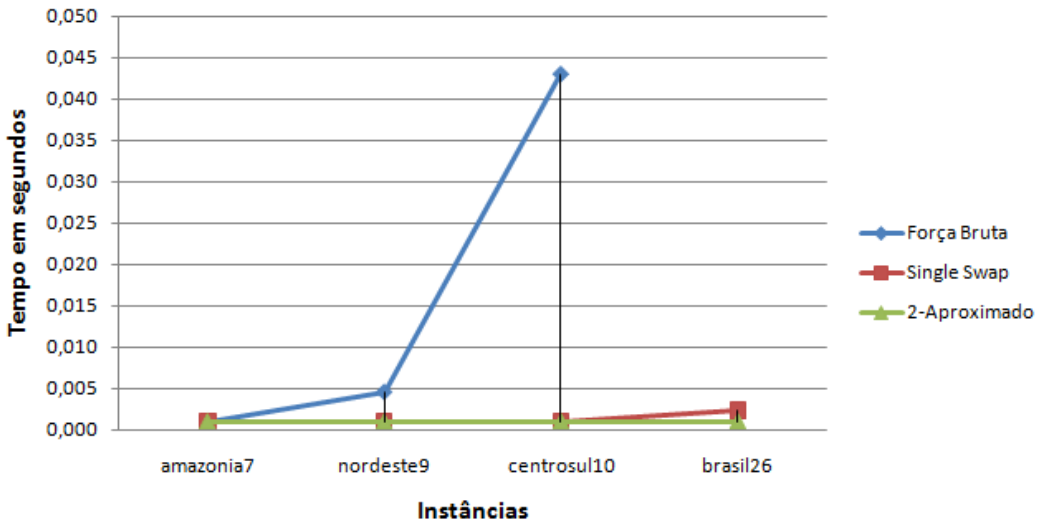



**Figura 4.11:** Tempos de execução para as instâncias brasileiras.

grupos abordados anteriormente, os algoritmos não exatos resolveram as instâncias de forma praticamente instantânea.

A Figura 4.11 traz um gráfico no qual é possível observar a diferença entre os tempos de execução do algoritmo de força bruta e das abordagens não exatas. O tempo de execução do Força Bruta para *brasil26* não foi colocado no gráfico, pois não é possível compará-lo aos demais tempos em uma mesma escala.

**Tabela 4.14:** Custo das soluções para as instâncias brasileiras

| | Custo da Solução Obtida | | | | | |
|---|---|---|---|---|---|---|
| Instâncias | OPT | FB | SS Melhor | SS Médio | SS Pior | 2-Aprox |
| amazonia7 | **13360** | **13360** | **13360** | 13363 | 13370 | 13380 |
| nordeste9 | **4456** | **4456** | **4456** | **4456** | **4456** | 5259 |
| centrosul10 | **6036** | **6036** | **6036** | **6036** | **6036** | 8286 |
| brasil26 | **-** | **-** | 19885 | 20931,1 | 21400 | 24724 |

As Tabelas 4.14 e 4.15 mostram que o algoritmo *Single Swap* foi capaz de encontrar a solução ótima para três das quatro instâncias brasileiras de forma praticamente instantânea. Mais uma vez é possível observar que, mesmo para instâncias bem pequenas, o algoritmo de força bruta não é sempre a melhor opção, apesar de garantir a solução ótima. É interessante ressaltar também que o método heurístico encontrou a solução ótima em todas as 20 execuções para duas instâncias do conjunto.

A instância *brasil26* não possui solução ótima conhecida. Por isso, foi utilizada como parâmetro de comparação a melhor solução obtida pelo algoritmo *Single Swap*. Para a instância *amazonia7*, o algoritmo 2-Aproximado obteve uma solução muito próxima ao ótimo, mas não foi capaz de encontrar soluções competitivas para as demais instâncias se comparado à abordagem heurística que, uma vez mais, obteve resultados muito mais próximos do ótimo quando não o atingiu.

A Figura 4.12 mostra no mapa a melhor solução obtida pelo algoritmo *Single Swap* para a instância *brasil26*. As linhas retas traçadas entre as cidades têm fim meramente ilustrativo, uma vez que as instâncias desse grupo consideram apenas rodovias federais, estaduais e municipais pavimentadas.

Da mesma forma, as Figuras 4.13, 4.15 e 4.14 mostram as soluções ótimas para as instâncias *amazonia7*, *nordeste9* e *centrosul10*. Apesar das figuras não apresentarem o caminho percorrido através das rodovias, elas mostram a ordem de visitação das cidades que corresponde à solução obtida.

**Tabela 4.15:** Desvio das soluções obtidas para as instâncias brasileiras em relação ao melhor valor encontrado.

| Desvio da Solução Obtida em Relação a Melhor Solução Encontrada | | | | | | |
|---|---|---|---|---|---|---|
| Instâncias | Melhor | FB | SS Melhor | SS Médio | SS Pior | 2-Aprox |
| amazonia7 | **13360 - OPT** | **0,00%** | **0,00%** | 0,02% | 0,07% | 0,15% |
| nordeste9 | **4456 - OPT** | **0,00%** | **0,00%** | **0,00%** | **0,00%** | 18,02% |
| centrosul10 | **6036 - OPT** | **0,00%** | **0,00%** | **0,00%** | **0,00%** | 37,28% |
| brasil26 | 19885 | **-** | **0,00%** | 5,26% | 7,62% | 24,33% |
| Média Total | | **0,00%** | **0,00%** | 1,32% | 1,92% | 19,95% |

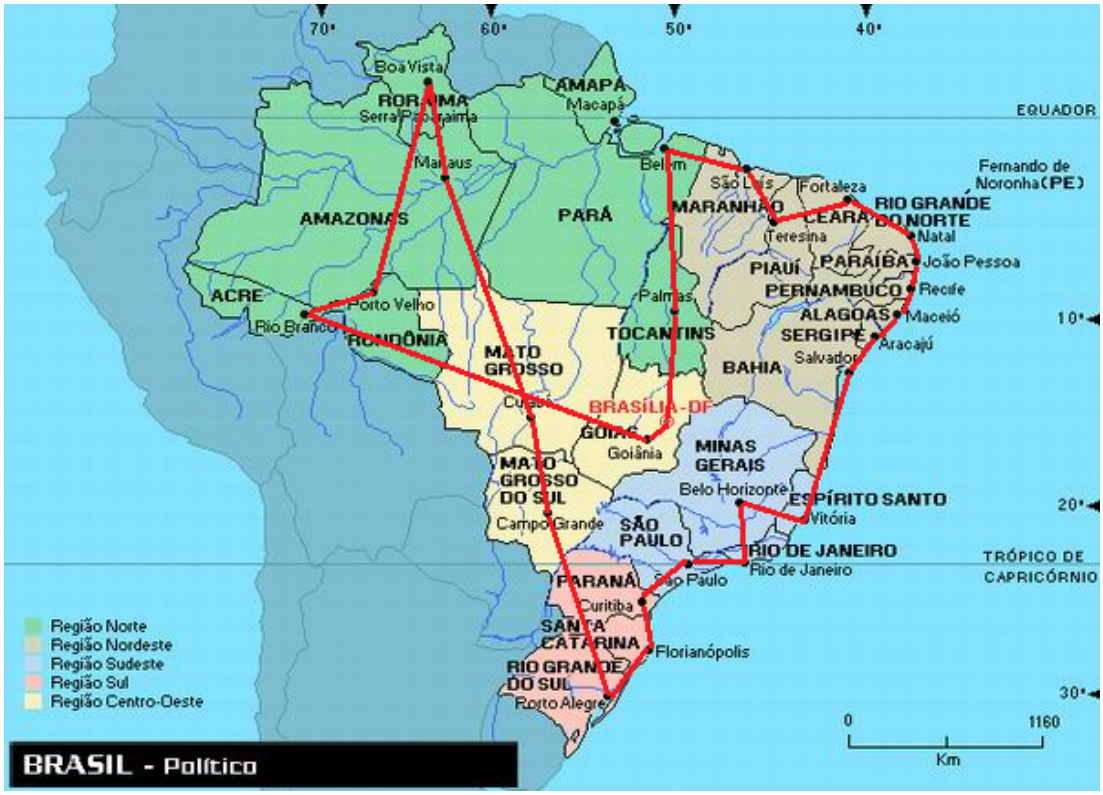



**Figura 4.12:** Melhor solução obtida pelo *Single Swap* para a instância *brasil26*.


**Figura 4.13:** Solução ótima para a instância *amazonia7*.


**Figura 4.14:** Solução ótima para a instância *centrosul10*.

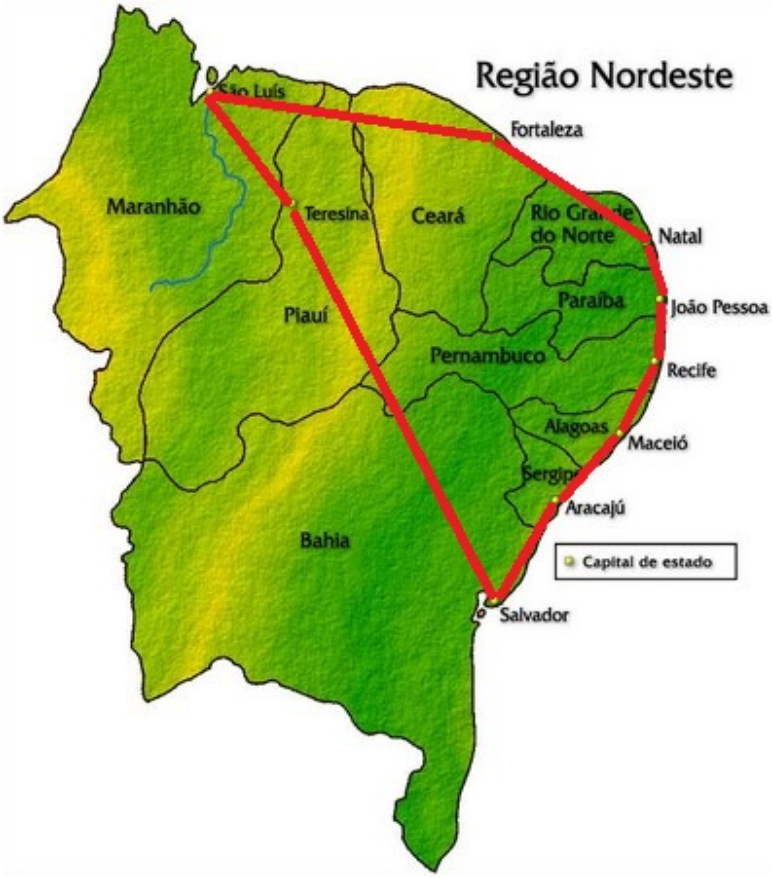



**Figura 4.15:** Solução ótima para a instância *nordeste9*.

CAPÍTULO

5

# Conclusão

Este capítulo contém as considerações finais deste Trabalho de Graduação, as contribuições realizadas na área de estudo e os trabalhos futuros que servirão de complemento para o que aqui foi feito.

## 5.1 Considerações finais

O TSP-Métrico, estudado neste trabalho, é um problema que modela diversas situações reais, tais como o roteamento de veículos numa malha viária, caso particular aqui abordado. Por ser da classe dos problemas NP-Completos, e portanto não existir um algoritmo polinomial para resolvê-lo na exatidão, o TSP é alvo de vários estudos que objetivam desenvolver métodos não exatos que encontrem boas soluções em tempo viável.

As abordagens implementadas neste trabalho, Força Bruta, heurística *Single Swap* e algoritmo 2-Aproximado, se mostraram capazes de resolver o TSP-Métrico de formas diferentes, apresentando vantagens e desvantagens particulares, como descrito a seguir.

O algoritmo de força bruta sempre encontra a solução ótima para o problema, mas não é capaz de fazer isso em tempo viável para instâncias maiores que 14 vértices, como foi averiguado nos testes realizados. Para a instância *burma14*, por exemplo, o algoritmo levou em média 15,8 minutos para encontrar a solução ótima, sendo que para instâncias maiores o tempo de execução

foi apenas estimado. Uma situação prática e próxima da realidade que mostra a limitação desta técnica é resolver o TSP para as capitais brasileiras, representado pela instância *brasil26*. Esse problema é constituído de 26 cidades, e o algoritmo de força bruta levaria aproximadamente $10^9$ séculos para resolvê-lo.

A heurística *Single Swap* se mostrou competitiva principalmente para instâncias pequenas, encontrando o ótimo para todas que constituem o Grupo 1 com tempo de execução da ordem de milissegundos. Para instâncias maiores a heurística também obteve bons resultados em um tempo de execução baixo, mantendo um desvio com relação ao ótimo sempre inferior a 12% para suas melhores soluções obtidas.

O algoritmo 2-Aproximado, que garante um erro máximo de 100% em relação ao ótimo, não se mostrou muito competitivo para a maioria das instâncias, sendo sempre superado no quesito qualidade pelo *Single Swap*. Contudo, esta foi a abordagem que obteve seus resultados com o menor tempo médio de execução, encontrando sempre uma solução viável com margem de erro máximo garantida. Um bom exemplo foi seu desempenho para a instância *u2319*, para a qual obteve uma solução com desvio de 47,78% em relação ao ótimo em aproximadamente 5 segundos.

Com esses dados, é possível concluir que a escolha da técnica de abordagem adequada deve ser feita de acordo com o problema a ser resolvido. Para instâncias muito pequenas, de até 14 vértices, como foi visto neste trabalho, quando houver uma grande preocupação com a qualidade da solução, a melhor opção é o algoritmo de força bruta, que garante o ótimo. Para as mesmas instâncias, quando a preocupação maior é com o tempo de execução, a heurística *Single Swap* pode ser mais interessante, ainda mais se executada várias vezes, como foi feito nos testes. Já para instâncias maiores, de até 1000 vértices, o *Single Swap* se mostrou a abordagem mais vantajosa, pois encontra boas soluções em um tempo de execução baixo. Porém, para instâncias grandes, com mais de 1000 vértices, o algoritmo 2-Aproximado é uma boa escolha se a maior preocupação é o tempo, uma vez que, apesar de não devolver soluções tão boas quanto o *Single Swap*, ele é executado em tempo muito inferior ao obtido através da heurística.

## 5.2 Contribuições

Este trabalho de graduação deixa como contribuição para a área de otimização combinatória a implementação e a análise de três técnicas de abordagem distintas para o TSP-Métrico, considerando instâncias de tamanhos variados e, consequentemente, auxiliando na escolha da metodologia que melhor serve à aplicações práticas diversas. Para o caso particular do roteamento de veículos numa malha viária, este trabalho contribui com a resolução e a análise de-

talhada de instâncias reais que representam as capitais dos estados brasileiros, utilizando dados oficiais do governo. Esta situação em particular é interessante no sentido de contextualizar a importância e a utilidade prática do TSP.

## 5.3 Trabalhos futuros

Os trabalhos futuros impulsionados a partir deste trabalho de graduação consistem no aprimoramento das abordagens aqui implementadas e no desenvolvimento de outras técnicas para a resolução do TSP. Como foi descrito no Capítulo 3, mais precisamente na Seção 3.4, o algoritmo *Branch & Bound* é uma abordagem muito interessante a ser implementada, pois é uma técnica exata que é capaz de resolver o TSP para instâncias maiores que as resolvidas através da força bruta em tempo viável.

Outra metodologia interessante, e que utiliza as técnicas aqui implementadas, é a mescla entre o algoritmo 2-Aproximado e a heurística *Single Swap*, sendo a solução obtida pelo primeiro utilizada como entrada para a heurística, provendo assim uma solução inicial otimizada. Ainda considerando o que foi descrito neste trabalho, é interessante estudar e implementar o algoritmo 1,5-Aproximado para o TSP, que pode melhorar o nível médio de qualidade das soluções obtidas pela abordagem 2-Aproximada.

# Referências Bibliográficas


BOYD, S.; MATTINGLEY, J. Branch and bound methods. *Stanford University*, 2007.

DE CARVALHO, M. H.; CERIOLI, M. R.; DAHAB, R.; FEOFILOFF, P.; FERNANDES, C. G.; FERREIRA, C. E.; GUIMARÃES, K. S.; MIYAZAWA, F. K.; JÚNIOR, J. C. P.; SOARES, J. A. R.; WAKABAYASHI, Y. *Uma introdução sucinta a algoritmos de aproximação*. IMPA, 2001.

CHAVES, A. A.; BIAJOLI, F. L.; MINE, O. M.; SOUZA, M. J. F. Modelagem exata e heurística para resolução de uma generalização do problema do caixeiro viajante. *XXXV - SBPO*, 2004.

COOK, W. TSP - Traveling Salesman Problem. `http://people.brunel.ac.uk/~mastjjb/jeb/info.html`, 2010.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. *Introduction to algorithms*. Segunda ed. The MIT Press, 2001.

DA CUNHA, C. B.; DE OLIVEIRA BONASSER, U.; ABRAHÃO, F. T. M. Experimentos computacionais com heurísticas de melhorias para o problema do caixeiro viajante. *XVI Congresso da Anpet*, 2002.

DNIT Site do departamento nacional de infraestrutura de transportes. `http://www1.dnit.gov.br/rodovias/distancias/distancias.asp`, 2010.

DORIGO, M.; GAMBARDELLA, L. M. Ant colonies for the travelling salesman problem. *Elsevier Science Ireland Ltd*, 1996.

DORIGO, M.; STUTZLE, T. ACO algorithms for the travelling salesman problem. *Elsevier Science Ireland Ltd*, 1999.

GRIMM, M. J. A simple algorithm for the metric traveling salesman problem. *Communications Systems Research Section*, 1984.

HEIDELBERG, U. TSPLIB - a library of sample instances for the TSP. `http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/index.html`, 2008.

JOHNSON, D. S.; MCGEOCH, L. A. The traveling salesman problem: A case study in local optimization. *Local Search in Combinatorial Optimization*, 1995.

LAWLER, E. L.; WOOD., D. E. Branch-and-bound methods: A survey. *Operations Research,*, 1966.

MARTELLO, S.; TOTH, P. *Knapsack problems: Algorithms and computer implementations*. John Wiley & Sons, 1990.

MOSCATO, P.; NORMAN, M. G. A "memetic" aproach for the traveling salesman problem implementation of a computational ecology for combinatorial optimization on message-passing systems. *20th Joint Conference on Informatic and Operations Research*, 1991.

VAZIRANI, V. V. *Approximation algorithms*. Segunda ed. Springer, 2003.