CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO
LABORATÓRIO DE ALGORITMOS E ESTRUTURAS DE DADOS II
ALGORITMOS E ESTRUTURAS DE DADOS II

CAIXEIRO VIAJANTE

CRISTHIAN SALA MINOVES
DIEGO SANTOS GONÇALVES
MARIANA BULGARELLI ALVES DOS SANTOS

Professores: Amadeu Almeida e Thiago de Souza Rodrigues

BELO HORIZONTE
DEZEMBRO DE 2019

CRISTHIAN SALA MINOVES 20183005167
DIEGO SANTOS GONÇALVES 20183012537
MARIANA BULGARELLI ALVES DOS SANTOS 20183000330

CAIXEIRO VIAJANTE

CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO
LABORATÓRIO DE ALGORITMOS E ESTRUTURAS DE DADOS II
ALGORITMOS E ESTRUTURAS DE DADOS II
BELO HORIZONTE
DEZEMBRO DE 2019

Lista de Figuras

Lista de Tabelas

Sumário

1 Introdução

O problema do Caixeiro Viajante consiste em, dado um conjunto de cidades por onde existe um caminho entre cada par de cidades, encontrar um caminho que, a partir de uma cidade, visita-se todas as cidades e retorna à inicial percorrendo a menor distância possível.

O Problema do Caixeiro Viajante tem sido muito utilizado em diversos métodos de otimização por ser um problema facilmente descrito e compreendido, de larga aplicabilidade e grande dificuldade de solução (Karp, 1975, apud Prestes (2006)).

No Problema do Caixeiro viajante, o tamanho do espaço de procura aumenta exponencialmente dependendo de n, o número de cidades, posto que dada uma cidade inicial, tem-se n-1 opções para a segunda cidade, n-2 para a terceira cidade, e assim por diante, sendo, por conseguinte, sua ordem de complexidade O (c^n), com c maior que 1.

Foram implementados um algoritmo de força bruta e duas heurísticas para o problema do Caixeiro Viajante: algoritmo genético e guloso. O algoritmo Força bruta (ou busca exaustiva) é um tipo de algoritmo de uso geral que consiste em enumerar todos os possíveis candidatos de uma solução e verificar se cada um satisfaz o problema. Já o algoritmo Guloso tenta resolver o problema fazendo a escolha localmente ótima em cada fase buscando encontrar um ótimo global, enquanto que o algoritmo genético é uma classe particular de algoritmos evolutivos que usam técnicas inspiradas pela biologia evolutiva.

1.1 Objetivo

O objetivo deste trabalho prático é a avaliação do desempenho na resolução do Problema do Caixeiro Viajante utilizando a implementação de um algoritmo de força bruta e duas outras heurísticas. Optou-se por implementar, além do algoritmo de Força Bruta, os algoritmos Guloso e Genético.

1.2 Organização do trabalho

Este trabalho apresenta no Segundo Capítulo um apanhado geral sobre o Problema do Caixeiro Viajante. No Capítulo Três é gerada uma discussão acerca da complexidade do problema, enquanto que no Quarto Capítulo os métodos de solução aplicados são explanados. Por fim, são apresentadas a Metodologia aplicada e análises dos resultados.

2 O Problema do Caixeiro Viajante

Este capítulo trata do problema do caixeiro viajante ao qual serão aplicados os algoritmos Força Bruta, Guloso e Genético. O Problema do Caixeiro Viajante (Traveling Salesman Problem) é um dos problemas mais intensamente estudados na área de computação e é fundamental para inúmeras aplicações, como mapeamento genético e projeto de circuitos (WATERLOO, 2019).

2.1 O Problema do Caixeiro Viajante

O Problema do Caixeiro Viajante tem sido muito utilizado em diversos métodos de otimização por ser um problema facilmente descrito e compreendido, de larga aplicabilidade e grande dificuldade de solução, posto que é NP-difícil (Karp, 1975, apud Prestes (2006)).

O Problema do Caixeiro Viajante consiste em, dada uma coleção de cidades e o custo da viagem entre cada um deles, deve-se encontrar o caminho de menor custo para visitar todas as cidades e retornar ao seu ponto de partida (passando uma única vez por cada cidade), sendo que viajar da cidade X para a cidade Y pode ter o mesmo custo que viajar de Y para X (Problema do caixeiro viajante simétrico) ou não (Problema do caixeiro viajante assimétrico) (WATERLOO, 2019).

Em outras palavras, dado um grafo G = (N,E), onde N = é o conjunto de nós e E = {1, ..., m} é o conjunto de arestas de G, sendo c os custos associados com cada aresta ligando os vértices i e j, o problema consiste em determinar o menor ciclo Hamiltoniano do grafo G que minimize o custo total (PRESTES, 2006). O custo total é dado pelo somatório dos custos das arestas que formam o ciclo.

Um ciclo hamiltoniano em um grafo é um ciclo que passa (uma única vez) por cada vértice deste grafo. Um grafo é hamiltoniano se admite um ciclo hamiltoniano (FEOFILOFF, 2000).

Dada uma cidade inicial, para calcular o número de diferentes caminhos pelas n cidades, tem-se n-1 opções para a segunda cidade, n-2 para a terceira cidade, e assim por diante.

Segundo Prestes (2006), em 1990, foi criado o TSPLIB, uma biblioteca de instâncias de amostras para o Problema do Caixeiro Viajante (e problemas relacionados) de várias fontes e de vários tipos (HEIDELBERG, 2019). Esta biblioteca contém mais de 100 exemplos com tamanhos que variam de 14 até 85900 cidades.

Desde abril de 2004, restam apenas três instâncias da TSPLIB que ainda não

foram resolvidas, contendo 18512, 33810 e 85900 cidades (PRESTES, 2006). A Tabela 1 apresenta os melhores resultados para as instâncias ainda não resolvidas do TSPLIB.

Tabela 1 – Melhores resultados para as instâncias ainda não resolvidas. Fonte: (PRESTES, 2006)

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

O Problema do caixeiro viajante é um problema de otimização combinatória extremamente pesado (NP-difícil), tanto que para encontrar uma viagem de ida e volta mais curta para os grandes problemas de hoje consome tempo, exigindo anos para serem resolvidos em computadores paralelos, apesar do hardware e software modernos (HEIDELBERG, 2019). A Figura 1 mostra o caminho mais curto por 15112 cidades (HEIDELBERG, 2019).

Figura 1 – Caminho mais curto por 15112 cidades. Fonte: (HEIDELBERG, 2019)

2.2 Origem do Problema

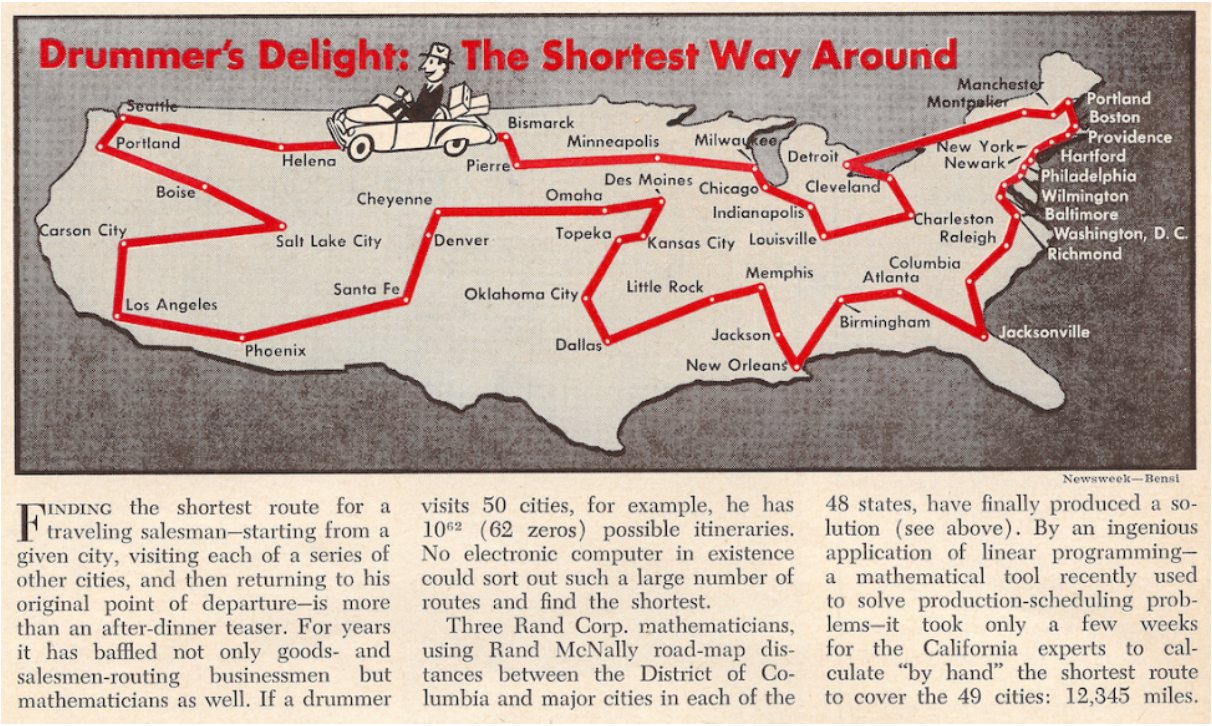
O matemático irlandês Sir William Rowan Hamilton e o matemático britânico Thomas Penyngton Kirkman trataram no século XIX problemas relacionados ao Problema do Caixeiro Viajante. No entanto, a origem do termo "caixeiro viajante"não é bem definida. Segundo Applegate Robert E. Bixby (2006), aparentemente, não há documentação oficial que indique o criador do nome.

Ainda de acordo com Applegate Robert E. Bixby (2006), um dos primeiros e mais influentes pesquisadores do problema, Merrill Flood, da Universidade de Princeton, em uma entrevista comentou: "não sei quem cunhou o nome "problema do caixeiro viajante"para o problema de Hassler Whitney [1], mas certamente esse nome pegou e o problema acabou sendo de uma importância fundamental"(APPLEGATE ROBERT E. BIXBY, 2006).

A forma geral do TSP foi estudada pela primeira vez por matemáticos a partir da década de 1930 por Karl Menger em Viena e Harvard e posteriormente por Hassler Whitney e Merrill Flood em Princeton (WATERLOO, 2019). Hassler Whitney descreveu o problema como "o desafio de encontrar um caminho mais curto para visitar uma cidade em cada um dos estados dos EUA".

O primeiro avanço na solução do Problema do caixeiro viajante foi promovido por George Dantzig, Ray Fulkerson e Selmer Johnson (1954) quando publicaram uma descrição de um método de resolução para uma instância com 49 cidades (DANTZIG42). Eles criaram essa instância escolhendo uma cidade de cada um dos 48 estados dos EUA na época e Washington, DC. Os custos de viagem entre essas cidades foram definidos pelas distâncias rodoviárias(WATERLOO, 2019). Os cálculos, segundo Waterloo (2019), foram realizados manualmente e utilizaram programação linear (Figura 2).

[1] O "48-states problem"de Hassler Whitney



FINDING the shortest route for a traveling salesman—starting from a given city, visiting each of a series of other cities, and then returning to his original point of departure—is more than an after-dinner teaser. For years it has baffled not only goods- and salesmen-routing businessmen but mathematicians as well. If a drummer visits 50 cities, for example, he has 10^{62} (62 zeros) possible itineraries. No electronic computer in existence could sort out such a large number of routes and find the shortest.

Three Rand Corp. mathematicians, using Rand McNally road-map distances between the District of Columbia and major cities in each of the 48 states, have finally produced a solution (see above). By an ingenious application of linear programming—a mathematical tool recently used to solve production-scheduling problems—it took only a few weeks for the California experts to calculate "by hand" the shortest route to cover the 49 cities: 12,345 miles.

Figura 2 – Reportagem da Newsweek sobre a resolução para uma instância com 49 cidades. Fonte: (WATERLOO, 2019)

Utilizaram uma abstração utilizando 42 cidades, removendo Baltimore, Wilmington, Filadélfia, Newark, Nova York, Hartford e Providence. Um caminho ideal pelas 42 cidades usou a "fronteira que unia Washington, D.C. a Boston e como a rota mais curta entre essas duas cidades passa pelas sete cidades removidas, essa solução do problema de 42 cidades produz uma solução para o problema de 49 cidades"(WATERLOO, 2019).

Em 1962, Proctor e Gamble lançou um concurso (Figura 3) para a resolução de um Problema do Caixeiro Viajante para 33 cidades especificadas oferecendo um prêmio de $10000 dólares pra quem conseguisse. Um dos primeiros pesquisadores do assunto,"o professor Gerald Thompson, da Carnegie Mellon University, foi um dos vencedores"(WATERLOO, 2019).

Figura 3 – Concurso da Proctor and Gamble. Fonte: (WATERLOO, 2019)

Em 1971, Michael Held e Richard Karp [2] solucionaram o problema para 64 cidades aleatórias. Em palestra no Prêmio Turing, Richard Karp fez o seguinte comentário acerca da solução: "Eu não acho que nenhum dos meus resultados teóricos tenha proporcionado uma emoção tão grande quanto a visão dos números saindo do computador na noite em que Held e eu primeiro testei nosso método de delimitação"(WATERLOO, 2019).

Em 1977, Martin Groetschel encontrou o caminho ideal para 120 cidades [3] do que era então a Alemanha Ocidental (GR120).

H. Crowder e M. W. Padberg resolveram em 1980 a instância LIN318 que consiste em 318 pontos que surgiram em um aplicativo de perfuração (fornecido por R. Habermann), onde a broca era um laser pulsado (WATERLOO, 2019). M. Padberg e G. Rinaldi, em 1987,

[2] "O tema unificador no trabalho de Karp tem sido o estudo de algoritmos combinatórios. Seu artigo de 1972, Redutibilidade entre problemas combinatórios, mostrou que muitos dos problemas combinatórios mais comumente estudados são NP-completos e, portanto, provavelmente intratáveis". Fonte: https://simons.berkeley.edu/people/richard-karp

[3] Listadas na tabela da edição de 1967/68 do Deutscher General Atlas (WATERLOO, 2019).

resolveram a instância ATT532, um conjunto de dados com 532 cidades continentais dos Estados Unidos (Figuras 4 e 5).

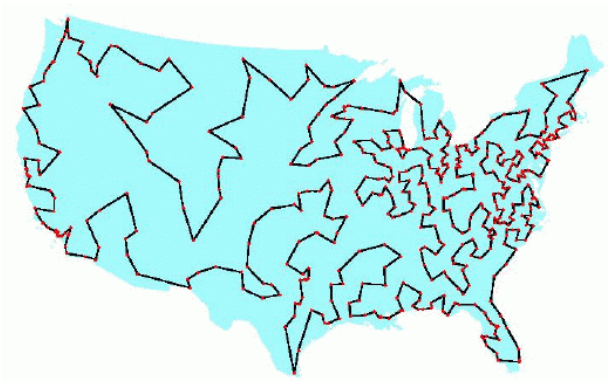


Figura 4 – ATT532. Fonte: (WATERLOO, 2019)

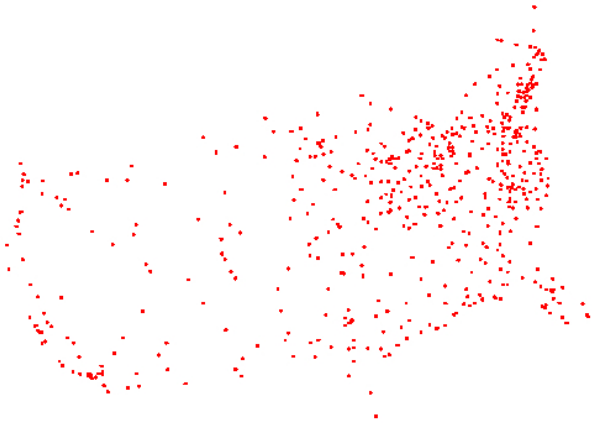


Figura 5 – ATT532. Fonte: (WATERLOO, 2019)

Ainda em 1987, Olaf Holland e M. Groetschel solucionaram o GR666, conjunto de dados com 666 cidades interessantes de todo o mundo cujas distâncias são especificadas por uma função que aproxima as grandes distâncias circulares. No mesmo ano, um TSPLIB de tamanho médio (PR2392) foi resolvido por M. Padberg e G. Rinaldi e "constava de um layout de 2.392 pontos obtidos da Tektronics Incorporated"(WATERLOO, 2019)

Segundo Waterloo (2019), Applegate, Bixby, Chvátal e Cook, em 1994, encontraram o caminho ideal para um TSP de 7.397 cidades (PLA7397) que surgiu em um aplicativo de matriz lógica programável nos laboratórios da AT&T Bell. Em 1998 encontraram o caminho ideal das 13.509 cidades nos EUA com populações superiores a 500 (Figura 6).

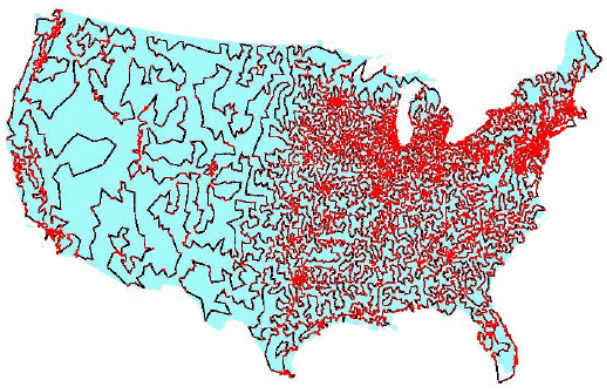


Figura 6 – Caminho ideal das 13.509 cidades nos EUA com populações superiores a 500. Fonte: (WATERLOO, 2019)

O mesmo grupo de pesquisadores, em 2001, encontraram o caminho ideal por 15.112 cidades (D15112) na Alemanha (Figura 1). Com a ajuda de Helsgaun, este grupo de pesquisadores, em maio de 2004, solucionou o sw24798, que continha todas as 24.978 cidades da Suécia (Figura 7), derivadas de dados do banco de dados da Agência Nacional de Imagens e Mapeamento de nomes de recursos geográficos (WATERLOO, 2019). Foi encontrado um caminho "de 855.597 unidades TSPLIB (aproximadamente 72.500 quilômetros) e ficou provado que não existe um caminho mais curto"(WATERLOO, 2019).

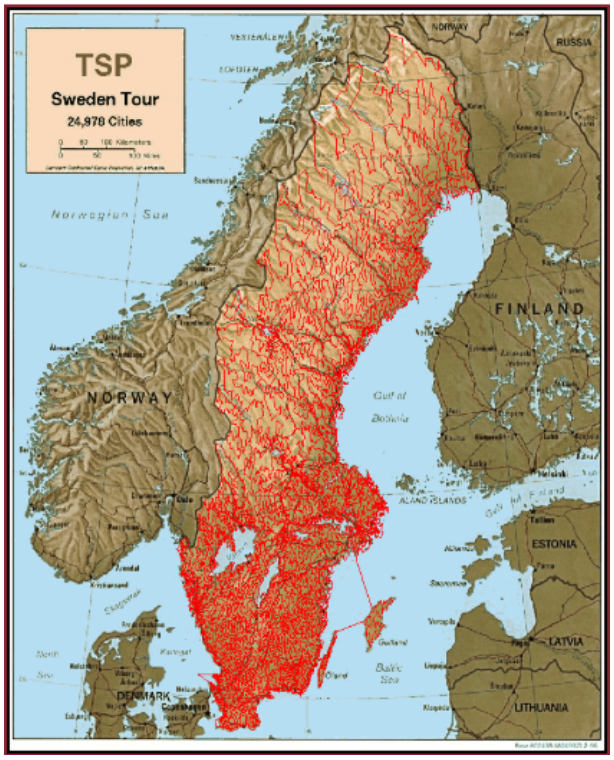


Figura 7 – Caminho ideal das 13.509 cidades nos EUA com populações superiores a 500. Fonte: (WATERLOO, 2019)

Os algoritmos para tratar o Problema do Caixeiro Viajante tornaram-se cada vez mais sofisticados ao longo dos anos, permitindo a solução de instâncias cada vez maiores, "passando da solução de Dantzig, Fulkerson e Johnson para um problema de 49 cidades em 1954 até a solução de um problema de 24.978 cidades 50 anos depois"(WATERLOO, 2019). A Tabela 2 apresenta os avanços na solução das instâncias do Problema do Caixeiro Viajante.

Tabela 2 – Avanço na solução das instâncias do Problema do Caixeiro Viajante. Fonte: (WATERLOO, 2019)

Ano	**Pesquisadores**	**Tamanho**	**Instância**
1954	G. Dantzig, R. Fulkerson, and S. Johnson	49	dantzig42
1971	M. Held and R.M. Karp	64	64 random points
1975	P.M. Camerini, L. Fratta, and F. Maffioli	67	67 random points
1977	M. Grötschel	120	gr120
1980	H. Crowder and M.W. Padberg	318	lin318
1987	M. Padberg and G. Rinaldi	532	att532
1987	M. Grötschel and O. Holland	666	gr666
1987	M. Padberg and G. Rinaldi	2,392	pr2392
1994	D. Applegate, R. Bixby, V. Chvátal, and W. Cook	7,397	pla7397
1998	D. Applegate, R. Bixby, V. Chvátal, and W. Cook	13,509	usa13509
2001	D. Applegate, R. Bixby, V. Chvátal, and W. Cook	15,112	d15112
2004	D. Applegate, R. Bixby, V. Chvátal, W. Cook,and K. Helsgaun	24,978	sw24798

3 Complexidade

Apesar da simplicidade da enunciação do problema do caixeiro viajante, trata-se de um dos problemas mais intensamente estudados em matemática computacional (WATERLOO, 2019).

Ainda não se tem conhecimento de nenhum método de solução eficaz no caso geral, o que solucionaria o problema P versus NP. Mesmo assim, "seu estudo abriu caminho para métodos de solução aprimorados em muitas áreas da otimização matemática"(WATERLOO, 2019).

No Problema do Caixeiro viajante, o tamanho do espaço de procura aumenta **exponencialmente** dependendo de n, o número de cidades, posto que dada uma cidade inicial, tem-se n-1 opções para a segunda cidade, n-2 para a terceira cidade, e assim por diante (vide equação a seguir). Trata-se de um problema NP-difícil (ANDRETTA, 2015)(PRESTES, 2006).

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

A ordem de complexidade do Problema do Caixeiro Viajante é exponencial, cuja notação O é O ((n!)) - O (c^n), com c maior que 1 (Figura 8).

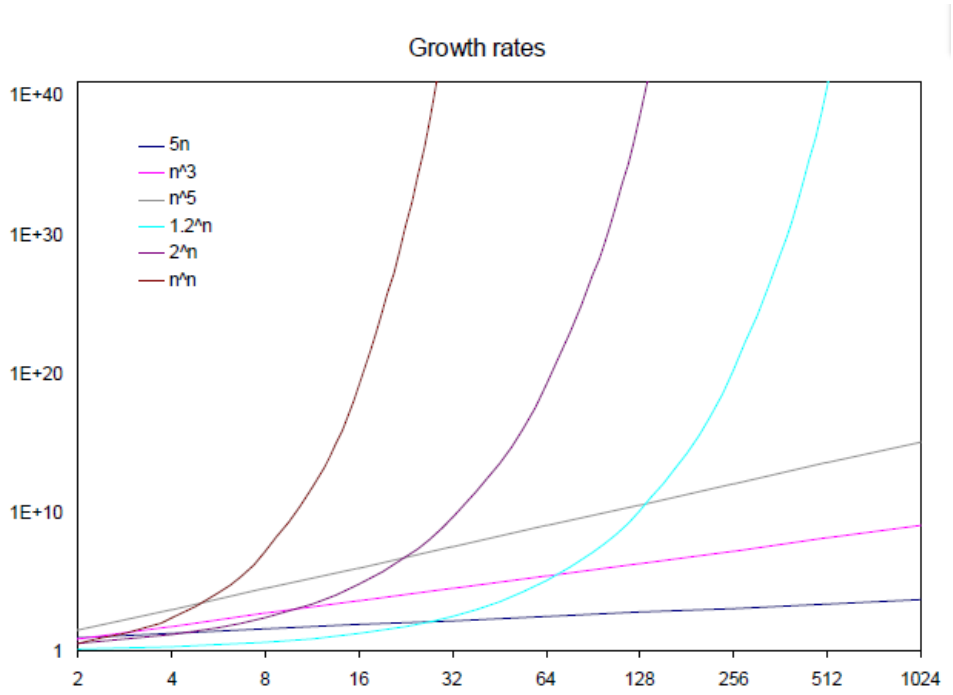


Figura 8 – Complexidades. Fonte: (RODRIGUES, 2019)

"Para provar se um problema pode ser resolvido por um algoritmo, basta dar uma descrição detalhada de um procedimento finito capaz de resolver o problema computacional em consideração"(OLIVEIRA, 2010). A Teoria da Complexidade particiona os problemas que podem ser resolvidos por algoritmos de acordo com a quantidade de recursos computacionais necessários e suficientes para resolver cada problema. A Teoria da Complexidade visa "estimar os recursos computacionais necessários para solucionar problemas [...], identificar e definir a tratabilidade dos [mesmos] [...], desenvolver métodos para classificá-los [...] e comparar a eficiência de diversos modelos computacionais"(OLIVEIRA, 2010).

Dentro dessa classificação tem-se os problemas P e NP. O conjunto de problemas computacionais com soluções que podem ser **encontradas** de forma eficiente são ditos P, enquanto na classe NP [1] estão os aqueles com soluções que podem ser **verificadas** de forma eficiente (OLIVEIRA, 2010).

Problemas NP são aqueles cuja melhor solução conhecida demanda algoritmos com complexidade (assintótica) de ordem exponencial. A classe NP de problemas é constituída pelo conjunto dos problemas polinomialmente verificáveis, ou seja, aqueles em que é possível verificar, em tempo polinomial, se uma suposta solução de uma instância de X é de fato uma solução [2](FEOFILOFF, 2018).

NP-completo é um subconjunto de NP. Um problema de decisão C é NP-completo se C está em NP e se todo problema em NP é redutível para C em tempo polinomial. Um problema H é NP-difícil (ou NP-complexo) se, e somente se, existe um problema NP-completo L que é Turing-redutível em tempo polinomial para H, ou seja, L pode ser resolvido em tempo polinomial por uma Máquina de Turing não determinística com um oráculo para H. Um problema é NP-completo se for NP-difícil e estiver em NP (FEOFILOFF, 2018).

"Um problema de decisão que seja NP-difícil pode ser mostrado ser NP-completo exibindo um algoritmo não-determinista polinomial"para o mesmo (RODRIGUES, 2019). Além disso, um problema NP-difícil não é menos difícil do que um problema NP-completo.

[1] Nondeterministic polynomial.

[2] Um problema é da classe NP "se existe um algoritmo que, ao receber uma instância I de X e uma suposta solução S de I, responde sim ou não"(FEOFILOFF, 2018).

4 Métodos de Solução para o Problema do Caixeiro Viajante

Neste capítulo serão apresentadas, de maneira simplificada, duas maneiras de se abordar o Problema do Caixeiro Viajante: utilizando abordagens exatas e heurísticas. O desenvolvimento do presente trabalho foi focado na utilização dos algoritmos de força bruta, genético e guloso.

4.1 Métodos exatos

De maneira simplista, métodos exatos são aqueles que analisam todas as alternativas possíveis. No entanto, para problemas de otimização combinatória esta abordagem é impraticável devido ao crescimento exponencial das possibilidades em relação ao tamanho do problema (PRESTES, 2006). Pode-se utilizar estratégias baseadas em Programação Inteira, que provam que uma solução viável não existe ou garantem a obtenção da solução ótima e provam a otimalidade da solução obtida num tempo de execução finito (PRESTES, 2006). Os métodos mais usados são os do tipo ”branch-and-cut” e ”branch-and-bound” (consistem em expandir os nós e cortar caminhos de pesquisa que não são promissores, Relaxação Lagrangeana e Programação Dinâmica. Muitas das soluções dos últimos 30 anos das instâncias do Problema do caixeiro viajante ocorreram com a utilização de métodos Branch and Cut (Figura 9).

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

Figura 9 – Histórico das instâncias Resolvidas com Branch and cut. Fonte: (PRESTES, 2006)

Como vantagens, este tipo de método pode provar que soluções ótimas podem ser

obtidas se o algoritmo tiver sucesso na execução e fornecer informações sobre os limites inferior e superior em relação a solução ótima, mesmo se o algoritmo for terminado antes de completar sua execução (PRESTES, 2006). Assim, métodos exatos podem ser utilizados como aproximações caso seja definido um critério de parada que anteceda a conclusão do algoritmo.

Apesar disso, para muitos problemas o tamanho das instâncias que podem ser solucionadas é limitado pelo custo computacional muito elevado onde o tempo de processamento cresce exponencialmente com o tamanho da instância (PRESTES, 2006). Não obstante, outra limitação é o consumo de memória que pode ser muito alto levando ao término prematuro do programa (PRESTES, 2006).

4.1.1 Força Bruta

O algoritmo Força bruta (ou busca exaustiva) é um tipo de algoritmo de uso geral que consiste em enumerar todos os possíveis candidatos de uma solução e verificar se cada um satisfaz o problema.

Geralmente possui uma implementação simples e sempre encontrará uma solução se ela existir, a depender de seu custo computacional, que é proporcional ao número de candidatos a solução (DIAS, 2015).

Deste modo, é utilizado em problemas cujo tamanho é limitado, quando não se conhece um algoritmo mais eficiente ou quando a simplicidade da implementação é mais importante do que a velocidade de execução (DIAS, 2015). Assim, o algoritmo pode ser visto como a mais simples meta-heurística. O método Força bruta consiste em:

- **Listar** -> Listar todas as soluções potenciais para o problema.
- **Avaliar** -> Avaliar cada solução, mantendo a melhor encontrada até o momento.
- **retornar** -> Quando a busca terminar, retornar a solução encontrada.

Como vantagens, apresenta simplicidade e ampla aplicabilidade. No entanto, a principal desvantagem do método por força bruta é que, para muitos problemas reais, o numero de combinações cresce exponencialmente, tornando o processo muito lento.

4.2 Métodos heurísticos

Heurísticas são modificações no tratamento de um problema, produzindo algoritmos polinomiais para problemas exponenciais a custo de otimalidade. Neste sentido, um algoritmo heurístico pode não produzir uma solução ótima, podendo nem mesmo produzir uma solução.

Tendo em vista a grande dificuldade na solução exata do Problema do Caixeiro Viajante, por ter complexidade exponencial, várias abordagens heurísticas têm sido aplica-

das proporcionando soluções aproximadas de boa qualidade (PRESTES, 2006). Segundo Prestes (2006), um exemplo de abordagem heurística aplicada ao Problema do Caixeiro Viajante é a Busca Tabu que foi introduzida por Hansen (1986) e Glover (1990), no qual é criado um mecanismo para auxiliar o processo de busca, evitando que o mesmo espaço de soluções seja percorrido repetidas vezes. Além desse, pode-se citar o algoritmo guloso e o genético, focos deste trabalho.

4.2.1 Guloso

Segundo Roman (2019), Algoritmos gulosos (míopes) são aqueles que, a cada decisão sempre escolhem a alternativa que parece mais promissora naquele instante, sendo que esta escolha nunca é reconsiderada ou revista, ou seja, tenta resolver o problema fazendo a escolha localmente ótima em cada fase buscando encontrar um ótimo global. Além disso, podem ocorrer cálculos repetitivos, nem sempre produz a melhor solução e não leva em consideração as consequências das decisões.

Para construir a solução ótima existe um conjunto ou lista de candidatos. Um conjunto de candidatos é escolhido e acumulado, outros, rejeitados. Algoritmos Gulosos são comumente utilizados para resolver problemas de otimização que funcionem através de uma sequência de passos. É um algoritmo simples, de fácil implementação, rápida execução e que pode fornecer a melhor solução.

(ROMAN, 2019)

4.2.2 Genético

Um algoritmo genético é uma técnica de busca para soluções aproximadas em problemas de otimização e busca, fundamentado por John Henry Holland. São uma classe particular de algoritmos evolutivos que usam técnicas inspiradas pela biologia evolutiva como hereditariedade, mutação, seleção natural e recombinação (ou crossing over).

Algoritmos Genéticos são muito eficientes para busca de soluções ótimas, operando sobre uma população de candidatos em paralelo, de modo a fazer a busca em diferentes áreas do espaço de solução, alocando um número de membros apropriado para a busca em várias regiões (CARVALHO, 2009).

Segundo Carvalho (2009), os Algoritmos Genéticos (AGs) diferem dos métodos tradicionais de busca e otimização, trabalharem com uma codificação do conjunto de parâmetros e não com os próprios parâmetros; trabalharem com uma população e não com um único ponto; por utilizarem informações de custo ou recompensa além de regras de transição probabilísticas e não determinísticas.

Em algoritmos genéticos, a população de representações abstratas de solução é selecionada em busca de soluções melhores e a evolução se inicia a partir de um conjunto

de soluções criado aleatoriamente, sendo realizada por meio de gerações (CARVALHO, 2009). "Durante o processo evolutivo, esta população é avaliada: para cada indivíduo é dado [...] um índice, refletindo sua habilidade de adaptação a determinado ambiente. Uma porcentagem dos mais adaptados são mantidos, enquanto os outros são descartados (darwinismo)"(CARVALHO, 2009). Os membros mantidos, podem sofrer modificações em suas características fundamentais (mutações, cruzamento - crossover ou recombinação genética), gerando descendentes para a próxima geração. Este processo, chamado de reprodução, é repetido até que uma solução satisfatória seja encontrada (CARVALHO, 2009).

5 Metodologia e Procedimentos

Nesta seção será discutido o processo de execução da implementação dos algoritmos Força Bruta, Guloso e Genético com vistas à resolução do Problema do Caixeiro Viajante.

O trabalho é composto por três etapas:

- **Aplicação do algoritmo de força bruta** -> aplicação em instâncias do problema do caixeiro viajante.
- **Desenvolvimento das heurísticas Gulosa e Genética** -> desenvolvimento de duas heurísticas para solucionar três instâncias do mesmo problema.
- **Análise** -> avaliação dos resultados experimentais.

5.1 Premissas

O Problema do caixeiro viajante é um problema de complexidade exponencial cujo objetivo é, partindo de uma cidade inicial percorrer uma determinada quantidade de cidades, passando uma única vez por cada uma delas, e retornar ao local de partida. Cada caminho entre as cidades possui um determinado valor.

Como premissa do trabalho, este valor foi definido de maneira randômica. Ademais, ressalta-se que para os algoritmos nos quais era necessário definir um ponto de partida, optou-se pela cidade 0.

5.2 Dados Utilizados

Foram utilizadas três instâncias no formato .tsp, fornecidas pelos professores: si535, instância com 535 cidades; pa561, instância com 561; e si1032, instância com 1032 cidades.

5.3 Parte 1

Esta etapa consistiu na implementação do método de força bruta para solucionar o problema, ou seja, um algoritmo que determina todas as possíveis rotas e escolhe a menor.

Em seguida, instâncias de tamanho 2 à n foram geradas e aplicou-se o método Força Bruta para solucioná-las. O tempo de execução durante a aplicação do algoritmo foi computado.

Para a representação das cidades e os caminhos entre elas foi utilizado um grafo não direcionado implementado por meio de uma matriz adjacência através da classe Grafos. Com ela foi possível inserir os caminhos entre as cidades, verificar se um caminho existe além de retornar a distância entre duas cidades.

Para procurarmos o menor caminho que passe por todas as cidades e retorne para a cidade inicial, foi utilizada a classe ForcaBruta. Ela possui os campos que armazenam todos os caminhos possíveis que passam por todos as cidades, o menor caminho dentre eles e a menor distância deste caminho. Para encontrar todos os caminhos possíveis foi utilizado um algoritmo de permutação baseado em um algoritmo de permutação de Strings das notas do curso de Física Computacional da Universidade de Exeter.

Após encontrar todos os caminhos possíveis, é feito o cálculo da distância total desse caminho até encontrarmos o menor. Como mencionado anteriormente, o método de permutação foi implementado na classe Permutações. O vetor inicialmente não possui o vértice inicial, que será adicionado a cada caminho no início e no fim, para indicar o ciclo fechado. O método de permutação é feito utilizando rotações de forma recursiva a fim de se obter todas as possibilidades.

5.4 Parte 2

Esta etapa consistiu na implementação de duas heurísticas que têm como objetivo encontrar soluções para o problema do caixeiro viajante: uma de programação gulosa e uma utilizando algoritmos genéticos.

Em seguida, aplicou-se os métodos Guloso e Genético nas instâncias si535, pa561 e si1032 para solucioná-las. O tempo de execução durante a aplicação do algoritmo foi computado.

5.4.1 Algoritmo Guloso

Essa resolução funciona de uma maneira muito simples, buscando usar mínimo de processamento possível. Após a leitura do arquivo de texto, passa todos os dados para uma matriz de adjacência, e, escolhido o nó de partida, o algoritmo busca na linha da matriz de adjacência da cidade atual o próximo destino que gere um menor caminho.

Após achar o próximo destino e adicioná-lo no vetor de caminho, essa nova cidade passa a ser o novo ponto de partida, e a cidade anterior tem a sua coluna zerada na matriz de adjacência, a fim de evitar loopings. O algoritmo continua realizando os mesmos passos até percorrer todas as cidades. Por fim, é adicionado o caminho do último destino até o destino final: a cidade inicial.

5.4.2 Algoritmo Genético

Essa resolução se baseia na premissa do algoritmo genético, no qual ele interpreta o caminho total a ser percorrido como um vetor, aqui chamado de cromossomo. No início, é retirado da matriz de adjacência a linha e a coluna de index 0, pois ela será somente o ponto de partida e de chegada.

Feito isso, são gerados cromossomos de maneira aleatória, dando início à geração zero. Para as próximas gerações, será analisado quais são os cromossomos mais evoluídos, ou seja, que geram menor caminho, para serem os pais de uma nova geração.

Os pais são recombinados e analisados se eles sofrerão mutação, ou seja, realiza-se uma troca arbitrária de maneira aleatória, para gerarem novos filhos, os quais passarão pelo mesmo processo dos pais. Essa recombinação será realizada até a chegada de gerações limites, o qual foi fixada em dez mil para o caso em questão do caixeiro viajante, tentando buscar o equilíbrio entre tempo de execução e precisão numérica.

6 Análise e Discussão dos Resultados

Nesta seção serão avaliados os resultados experimentais da execução dos algoritmos Força Bruta, Guloso e Genético. Foi computado o tempo de execução durante a aplicação dos algoritmos e elaborados gráficos que avaliam o número de cidades em relação ao tempo de execução para cada um dos algoritmos avaliados.

6.1 Força Bruta

Para o algoritmo Força Bruta foram avaliados de 2 a 11 pontos apenas. Para n = 12 cidades ocorreu estouro de memória. Observa-se o crescimento exponencial do tempo necessário para resolver o problema do caixeiro viajante pelo crescimento do tamanho do problema (de 2 a 11). Os resultados contendo os tempos de execução para cada valor de n (cidades), bem como a menor distância calculada e o caminho originado são apresentados na Figura 10.

FORÇA BRUTA

Número de Cidades	Tempo de Execução (ns)	Menor distância	Caminho
2	5355900	59	0->1->0->
3	215500	96	0->2->1->0->
4	317900	72	0->1->3->2->0->
5	571200	109	0->1->2->3->4->0->
6	1255400	184	0->5->2->1->4->3->0->
7	4971200	152	0->1->3->2->5->4->6->0->
8	6967500	143	0->5->2->3->7->1->6->4->0->
9	24027600	144	0->4->3->8->7->1->5->2->6->0->
10	86865900	146	0->6->7->9->1->5->4->8->2->3->0->
11	3072548000	137	0->6->1->7->4->8->3->2->10->5->9->0->

Figura 10 – Tabela com os resultados do Força Bruta

O Gráfico Força Bruta apresentado na Figura 11 mostra com mais detalhes o crescimento exponencial do problema. Observa-se que entre os pontos 10 e 11 houve um crescimento alto no tempo de execução do algoritmo. Além disso, é notório que a distância das cidades diminuía e aumentava para cada n. Isto deve-se ao fato dos valores terem sido gerados de maneira randômica. Outro ponto a ser observado é que o número de cidades igual a dois apresenta um tempo de execução superior do que aqueles com mais cidades. Este erro de resultado é em função à JVM e ao Garbage Collector, sendo inclusive, algo esperado dada a linguagem e aplicação utilizadas (Java e Netbeans).

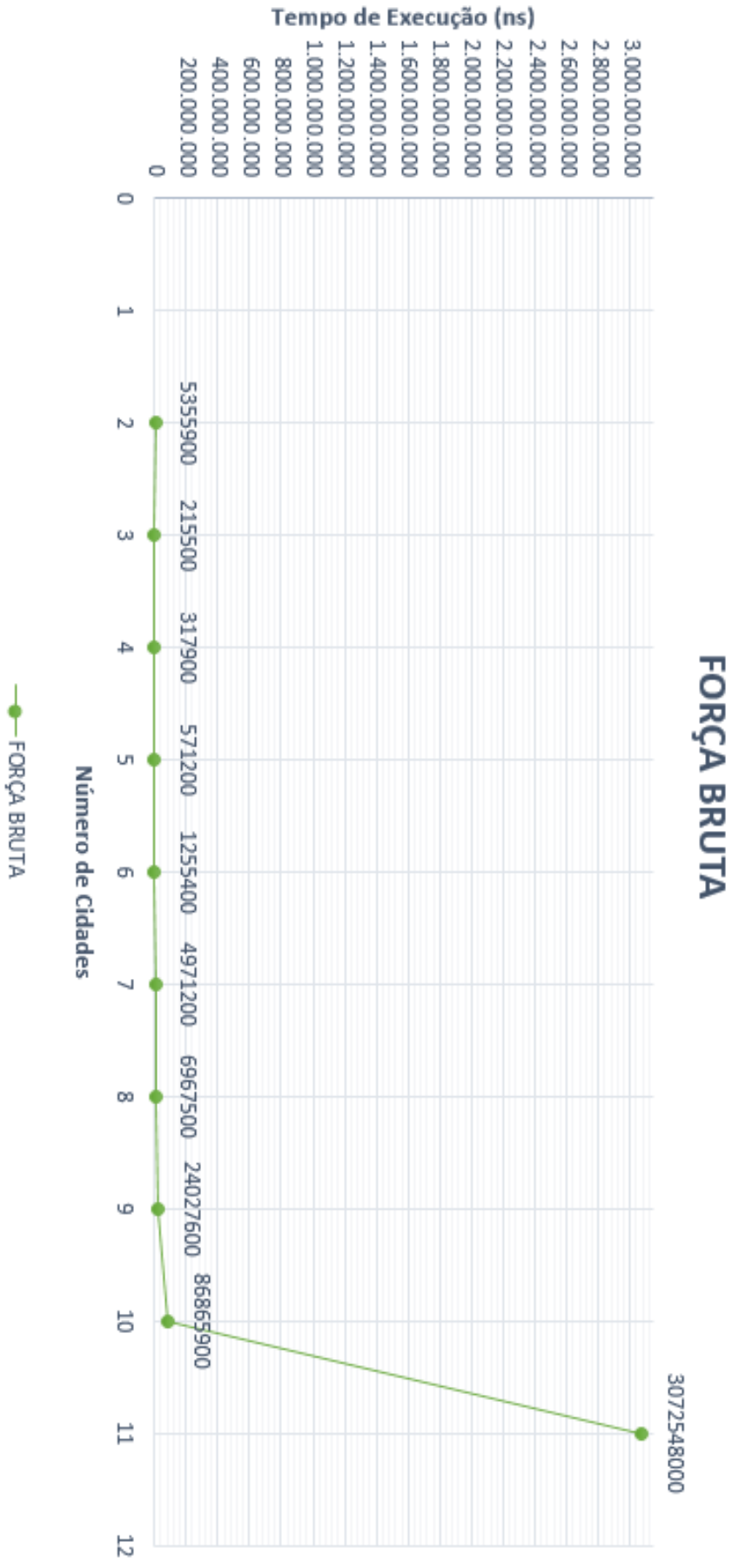


Figura 11 – Variação do tempo de execução em função do número de cidades do Força Bruta.

6.2 Algoritmos Genético e Guloso

A segunda parte experimental consistiu na avaliação de desempenho dos algoritmos Guloso e Genético para resolução do Problema do Caixeiro Viajante para diferentes instâncias, a saber:

- **si535** -> instância com 535 cidades.
- **pa561** -> instância com 561.
- **si1032** -> instância com 1032 cidades.

Durante a execução dos algoritmos foram coletados dados referentes ao tempo de execução para cada uma das instâncias e elaborados a tabela (Figura 12) e os gráficos (Figura 13, Figura 14 e Figura 15) a seguir. Para uma melhor visualização, os gráficos dos tempos de execução foram separados.

TEMPOS DE EXECUÇÃO (ns)			
	SI535	PA561	SI1032
Guloso	22253000	25411801	37008500
Genético	538991734499	558840085299	1170129031400

Figura 12 – Variação do tempo de execução em função do número de cidades.

Depreende-se da avaliação da Tabela da Figura 12 que os tempos necessários à execução do algoritmo genético foram mais que duas mil vezes maiores que os tempos de execução do algoritmo Guloso, considerando a mesma quantidade de cidades.

Conforme explicado no Capítulo anterior, no algoritmo Genético são gerados cromossomos de maneira aleatória, dando início à geração zero. Para as próximas gerações, são analisados quais são os cromossomos mais evoluídos (que geram o menor caminho) para serem os pais de uma nova geração.Deste modo, o algoritmo genético para quando chega ao limite do numero de evoluções. No caso deste trabalho, foi adotada a quantidade de 10000 evoluções, sendo que cada recombinação é uma evolução. Neste sentido, a grande variação nos tempos de execução são esperadas. Se considerássemos uma quantidade de evoluções igual a 1, teríamos valores de tempos de execução com ordem de grandeza próxima (Tabela 3).

Tabela 3 – Tabelas de tempos de execução.

TEMPOS DE EXECUÇÃO (ns)			
	SI535	**PA561**	**SI1032**
Guloso	22253000	25411801	37008500
Genético	53899173	55884009	117012903

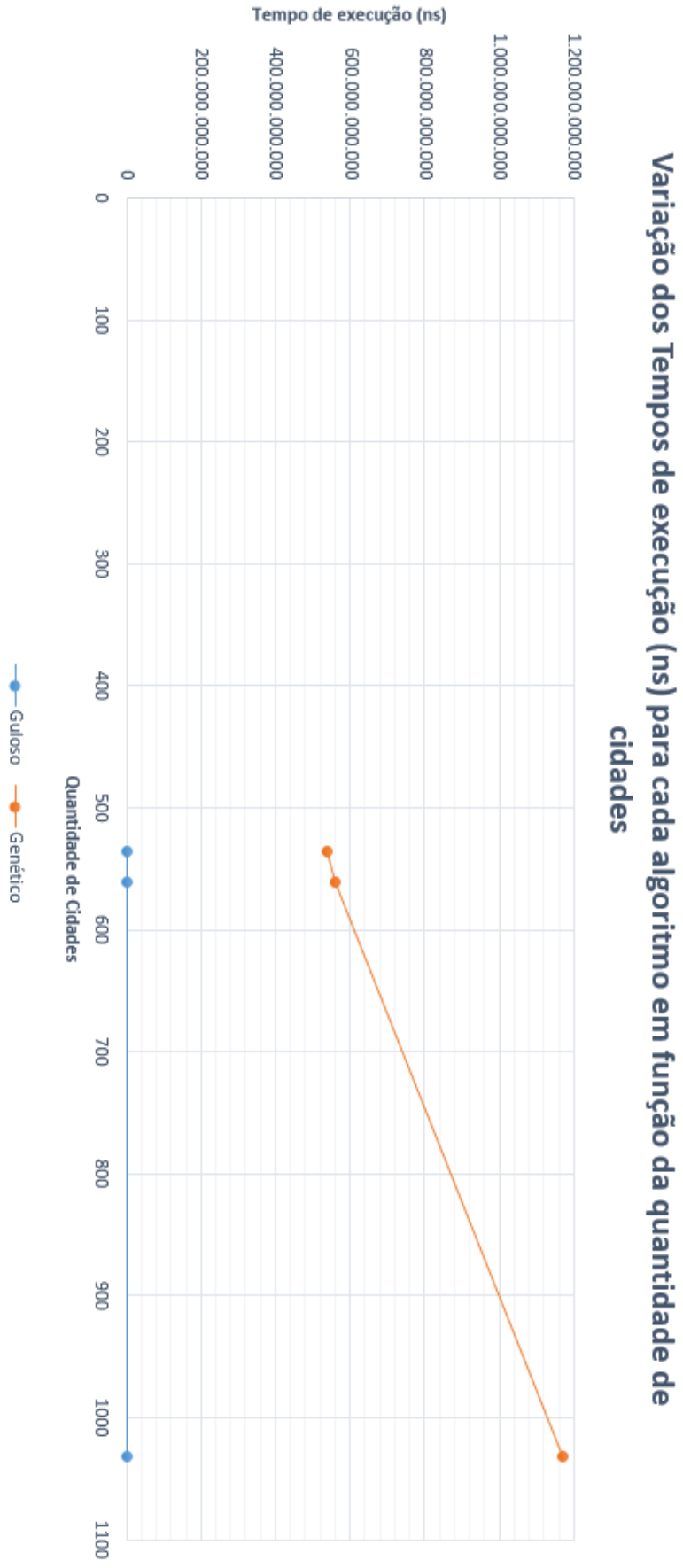


Figura 13 – Variação do tempo de execução em função do número de cidades do dos Algoritmos Guloso e Genético.

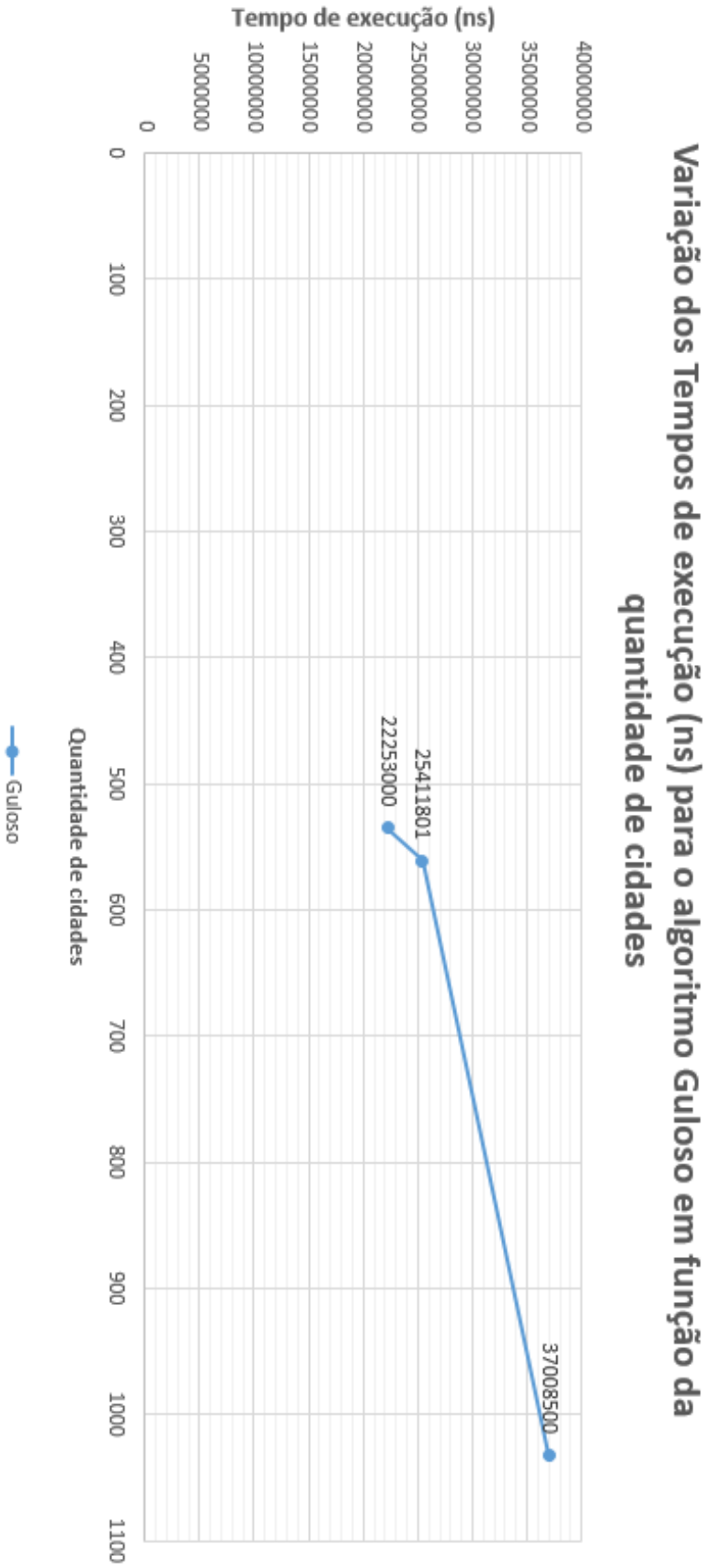


Figura 14 – Variação do tempo de execução em função do número de cidades do Algoritmo Guloso.

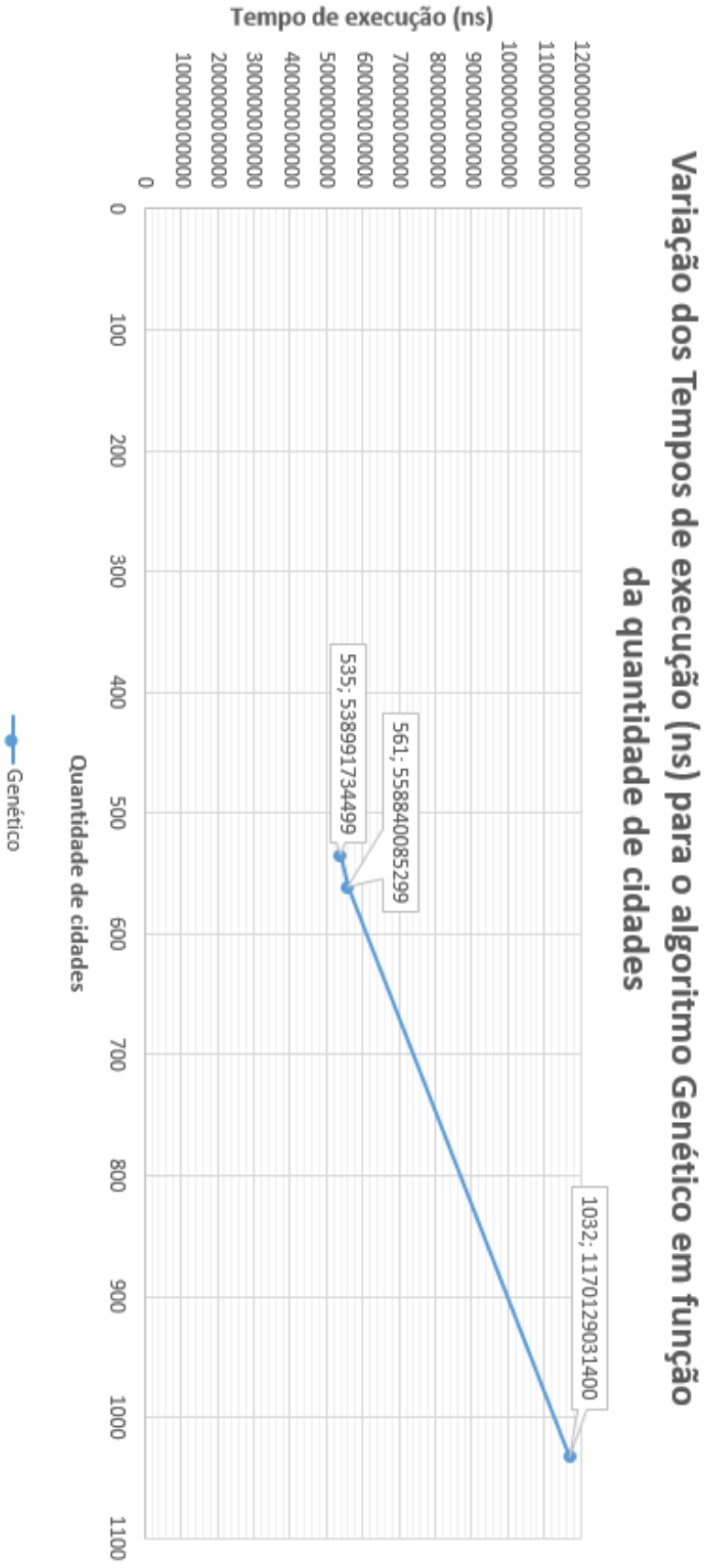


Figura 15 – Variação do tempo de execução em função do número de cidades do Algoritmo Genético.

O algoritmo Guloso tende a executar o Problema do Caixeiro Viajante em tempos menores do que o algoritmo Genético devido ao fato de que nenhuma das escolhas de menor caminho realizadas a cada iteração é reavaliada. Assim, ao escolher uma das opções de menor caminho de um ponto a outro, esta passa a ser a decisão. final. Já o algoritmo Genético revê as escolhas a cada evolução, mantendo apenas os cromossomos mais evoluídos. Isto contribui bastante para a variação dos tempos de execução.

Da análise dos gráficos dos tempos de execução necessários à resolução do Problema do Caixeiro Viajante para os algoritmos genético e Guloso, que o crescimento deste tempo em função da quantidade de cidades é de ordem polinomial. Em contraposição ao Gráfico referente ao tempo de execução do algoritmo Força Bruta, cujo crescimento é exponencial em função da quantidade de cidades visitadas, a diferença é muito grande, posto que o Força Bruta não realiza cálculos maiores que 11 cidades (tendo em vista a capacidade computacional do notebook utilizado).

Neste sentido, ao se utilizar um algoritmo Força Bruta, a complexidade exponencial inicial do Problema do Caixeiro viajante é mantida, limitando a conclusão dos cálculos para um número reduzido de cidades. O problema continua a pertencer à classe NP-difícil.

Os algoritmos Guloso e Genético transformam o problema do caixeiro viajante para uma complexidade polinomial por meio de Transformação Polinomial. Deste modo, a execução é mais simplificada. Além disso, o algoritmo genético realiza algumas aproximações, rendendo resultados de boa qualidade.

7 Conclusão

O intuito do presente trabalho foi exercitar os conceitos relacionados à grafos e complexidade de algoritmos por meio da avaliação do desempenho na resolução do Problema do Caixeiro Viajante utilizando a implementação dos algoritmos de força bruta, Guloso e Genético.

Tendo em vista que Problema do Caixeiro Viajante é um problema de complexidade exponencial pertencente à classe NP-difícil, a avaliação dos tempos de execução na sua resolução atende aos objetivos propostos pela prática.

Neste sentido, ao se utilizar um algoritmo Força Bruta, a complexidade exponencial inicial do Problema do Caixeiro viajante é mantida, limitando a conclusão dos cálculos para um número reduzido de cidades. Ao se utilizar das heurísticas Gulosa e Genética o panorama é diferente. Estes algoritmos, por meio de Transformação Polinomial, alteram a complexidade do problema para polinomial, facilitando sua execução para um número elevado de cidades. Ressalta-se aqui que os valores obtidos nem sempre correspondem ao valor de caminho mínimo ótimo posto que são resultados aproximados, diferentemente do algoritmo de força bruta.

Apesar do trabalho atender aos seus propósitos, sugerimos que análises de mais instâncias possam compor o cenário para que os gráficos fiquem visualmente mais interessantes.

Após a realização dos experimentos anteriormente detalhados, observamos que os resultados obtidos são condizentes com o propósito do trabalho prático.

Referências

ANDRETTA, M. **Notas de aula de Tópicos de Otimização Combinatória - Algoritmos de aproximação - Problema do caixeiro viajante**. [S.l.], 2015. Disponível em: <http://conteudo.icmc.usp.br/pessoas/andretta/ensino/aulas/sme0216-5826-2-15/aula6-caixeiro.pdf>. Acesso em: 29 de novembro de 2019. Citado na página 10.

APPLEGATE ROBERT E. BIXBY, V. C. W. J. C. D. L. **The travelling salesman problem: a computational Study**. [S.l.], 2006. Disponível em: <https://books.google.com.br/books?id=zfIm94nNqPoC&printsec=frontcover&hl=pt-BR#v=onepage&q&f=false>. Acesso em: 29 de novembro de 2019. Citado na página 4.

CARVALHO, A. P. de Leon F. de. **Algoritmos Genéticos**. [S.l.], 2009. Disponível em: <http://conteudo.icmc.usp.br/pessoas/andre/research/genetic/>. Acesso em: 29 de novembro de 2019. Citado 2 vezes nas páginas 14 e 15.

DIAS, Z. **Notas de aula de Algoritmos e Programação de Computadores - Força Bruta, Backtracking e Branch and Bound**. [S.l.], 2015. Disponível em: <http://www.ic.unicamp.br/~zanoni/mc102/2013-1s/aulas/aula22.pdf>. Acesso em: 29 de novembro de 2019. Citado na página 13.

FEOFILOFF, P. **Notas de aula do Curso de Teoria dos Grafos**. [S.l.], 2000. Disponível em: <https://www.ime.usp.br/~pf/mac5827/aulas/hamilton.html>. Acesso em: 29 de novembro de 2019. Citado na página 2.

FEOFILOFF, P. **Complexidade: problemas NP-completos**. [S.l.], 2018. Disponível em: <https://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/NPcompleto.html>. Acesso em: 29 de novembro de 2019. Citado na página 11.

HEIDELBERG, U. **TSPLIB - Discrete and Combinatorial Optimization**. [S.l.], 2019. Disponível em: <http://comopt.ifi.uni-heidelberg.de/>. Acesso em: 29 de novembro de 2019. Citado 3 vezes nas páginas ii, 2 e 3.

OLIVEIRA, I. C. **Complexidade Computacional e o Problema P vs NP**. 2010. 125 f. Dissertação (Mestrado em Ciência da Computação) — UNICAMP, 2010. Disponível em: <http://taurus.unicamp.br/bitstream/REPOSIP/275804/1/Oliveira_IgorCarboni_M.pdf>. Acesso em: 29 de novembro de 2019. Citado na página 11.

PRESTES Álvaro N. **Uma análise Experimental de abordagens Heurísticas Aplicadas ao Problema do Caixeiro Viajante**. Julho 2006. 85 f. Dissertação (Mestrado em Sistemas e Computação) — Universidade Federal do Rio Grande do Norte, 2006. Disponível em: <https://repositorio.ufrn.br/jspui/bitstream/123456789/17962/1/AlvaroNP.pdf>. Acesso em: 29 de novembro de 2019. Citado 9 vezes nas páginas ii, iii, 1, 2, 3, 10, 12, 13 e 14.

RODRIGUES, T. **Notas de Aula - Problemas NP-Completos e Algoritmos Aproximados**. [S.l.], 2019. Acesso em: 29 de novembro de 2019. Citado 3 vezes nas páginas ii, 10 e 11.

ROMAN, N. T. **Internet Research Laboratory**. 2019. Disponível em: <http://www.each.usp.br/digiampietri/SIN5013/13-algoritmosGulosos.pdf>. Acesso em: 29 de novembro de 2019. Citado na página 14.

WATERLOO, U. of. **The Traveling Salesman Problem**: The traveling salesman problem. [S.l.], 2019. Disponível em: <http://www.math.uwaterloo.ca/tsp/>. Acesso em: 29 de novembro de 2019. Citado 10 vezes nas páginas ii, iii, 2, 4, 5, 6, 7, 8, 9 e 10.