Aplicação do Problema do Caixeiro Viajante na Otimização de Coleta de Amostras de Solo na Ride da Grande Teresina

Maria Eva Clemencia Fonseca de Castro Silva[1] , Juan Morysson Viana Marciano[1] , Guilherme Amaral Avelino[1]

[1]Programa de Pós-Graduação da Ciência da Computação - Universidade Federal do Piauí (UFPI)

clemencia182017@gmail.com, desaribeiro.guilherme@gmail.com, juan.morysson@ifpi.edu.br

Abstract. *This paper addresses the traveling salesman problem (TSP) and its application in optimizing soil sample collection routes in the Greater Teresina RIDE. Using the Hamilton cycle concept, the aim is to minimize the time and costs associated with collection, ensuring efficiency. In the adopted model, each of the 15 municipalities is represented by a vertex, and the 27 roads mapped between them are the edges, with the weight corresponding to the distance in km. The resulting graph is non-directional, since the roads can be traveled in any direction. Brute force algorithms and a heuristic algorithm, the nearest neighbor, were implemented to compare the efficiency and accuracy of the solutions. The analysis of the results demonstrated that, although the heuristic algorithm does not guarantee the optimal solution, it offers reasonably good solutions in a feasible time, being a practical alternative for large-scale problems where resource optimization is crucial.*

Resumo. *Este trabalho aborda o problema do caixeiro viajante (PCV) e sua aplicação na otimização de rotas de coleta de amostras de solo na Ride da Grande Teresina. Utilizando o conceito do ciclo de Hamilton, busca-se minimizar o tempo e os custos associados à coleta, garantindo eficiência. No modelo adotado, cada um dos 15 municípios é representado por um vértice, e as 27 estradas mapeadas entre eles são as arestas, com o peso correspondente à distância em km. O grafo resultante é não direcional, pois as estradas podem ser percorridas em qualquer sentido. Foram implementados algoritmos de força bruta e um algoritmo heurístico, o do vizinho mais próximo, para comparar a eficiência e a precisão das soluções. A análise dos resultados demonstrou que, embora o algoritmo heurístico não garante a solução ótima, ele oferece soluções razoavelmente boas em tempo viável, sendo alternativa prática para problemas de grande escala onde a otimização de recursos é crucial.*

1. Introdução

O problema do caixeiro viajante (PCV) é um dos problemas clássicos mais notáveis nas áreas de programação e matemática. Sua origem está ligada à antiga ocupação do "Caixeiro Viajante", um comerciante que distribuía mercadorias em localidades desprovidas de sua produção, desempenhando um papel crucial em épocas de transporte limitado. Esse problema consiste em determinar a trajetória de menor custo para um comerciante que deseja visitar um conjunto limitado de cidades, começando e terminando na mesma cidade, e passando por cada cidade exatamente uma vez (APPLEGATE et al., 2006).

Goldbarg e Luna (2000) descrevem o PCV, ou *Travelling Salesman Problem* (TSP), em inglês, como um problema de roteamento que lida na maioria das vezes com passeios ou *tours* sobre pontos de demanda, que podem representar cidades, postos de trabalho, depósitos, entre outros. Um dos passeios mais importantes é o denominado hamiltoniano, em homenagem a Willian Rowan Hamilton. Em 1857, Hamilton propôs o jogo "*Around the World",* que envolvia um dodecaedro onde cada vértice representa uma cidade. O objetivo do jogo era encontrar uma rota que iniciasse e terminasse na mesma cidade, sem nunca repetir uma visita, como mostrado na Figura 01a. A solução do jogo de Hamilton, em sua homenagem, passou a se denominar um ciclo hamiltoniano. Um exemplo de solução do jogo está presente na Figura 01b.

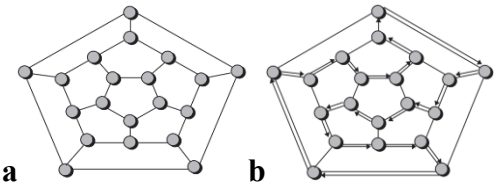


Figura 01: (a) Jogo de Hamilton (b) Uma solução do jogo de Hamilton

Fonte: Goldbarg e Luna (2005).

Ao associar o jogo de Hamilton a um grafo G = (V, E), onde V = {1,..,n} é o conjunto de nós/vértices e E = {1,...,m} é o conjunto de arestas de G, com cij, representando os custos associados a cada aresta ligando os vértices i e j, o problema consiste em localizar o menor ciclo hamiltoniano do grafo G. Os vértices do grafo representam as cidades, e o objetivo é visitar todas as cidades apenas uma vez por cada cidade, retornando ao ponto de origem, minimizando a distância total percorrida. O PCV é um problema de otimização associado ao da determinação dos caminhos hamiltonianos em um grafo (GOLDBARG E LUNA, 2000).

1.1. Formulação Matemática do PCV

De acordo Cormen et al. (2012) o PCV está intimamente relacionado ao problema do ciclo hamiltoniano. Um vendedor deve visitar n cidades, modelando o problema como um grafo completo com n vértices. Um vendedor deseja fazer um percurso, ou um ciclo hamiltoniano, visitando cada cidade exatamente uma vez e terminando na cidade de origem. O vendedor incorre em um custo inteiro não negativo c(i, j) para viajar da cidade i para a cidade j, e deseja fazer o percurso cujo custo total seja mínimo, onde o custo total é a soma dos custos individuais ao longo das arestas do percurso. A linguagem formal para o problema de decisão correspondente é dada por:

PCV= $\{\langle G, c, k\rangle: G = (V, E)$ é um grafo completo; c é uma função de V x $V \rightarrow N; k \in N$ e G tem um percurso de caixeiro-viajante com custo menor ou igual a $k\}$.

Em relação à formulação matemática, existem várias formulações para o PCV. No presente artigo, será utilizada a formulação de Dantzig, Fulkerson e Johnson, presente em Goldbarg e Luna (2000) visto que a mesma é frequentemente utilizada na literatura. É dada por:

$$Minimizar\ z = \sum_{j=1}^{n}\sum_{i=1}^{n} c_{ij}x_{ij}$$

Sujeito a:

$\sum_{i=1}^{n} x_{ij} = 1, \qquad \forall j \in V$ (1)

$\sum_{j=1}^{n} x_{ij} = 1, \qquad \forall i \in V$ (2)

$$\sum_{i,j \in S} x_{ij} \leq |S| - 1, \quad \forall S \in V \tag{3}$$

$$x_{ij} \in \{0,1\}, \quad \forall i,j \in V \tag{4}$$

Onde:

c_{ij}: Custo de ir da cidade i a cidade j (distância); $x_{ij} = 1$, se aresta $(i,j) \in A$, ou seja, for escolhido o caminho da cidade i até a cidade j para integrar a solução; $x_{ij} = 0$, caso contrário; S: é um subgrafo de G; $|S|$: números de vértices do subgrafo S.

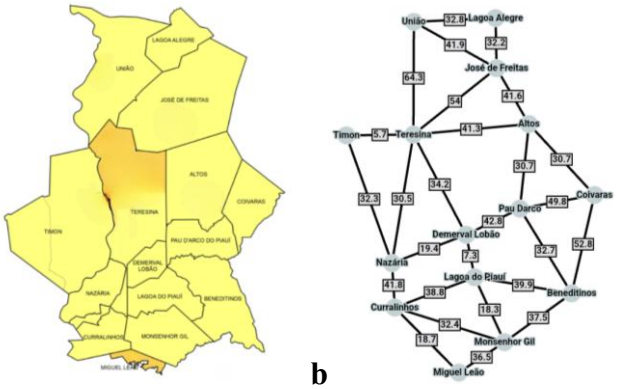
As restrições indicadas em (1), determinam que o fluxo de chegada em cada cidade j deve ser 1. As restrições indicadas por (2) determinam que o fluxo de saída de cada cidade j deve ser 1. As restrições indicadas por (3) impõem a eliminação de circuitos pré-hamiltonianos, ou seja, evitam subciclos. As restrições indicadas por (4) determinam que as variáveis sejam binárias, ou seja, podem assumir apenas os valores 0 ou 1.

2. Descrição do Problema Prático

Suponha que um pesquisador esteja conduzindo um estudo sobre a qualidade do solo nas 15 cidades da Região Integrada de Desenvolvimento (Ride) da Grande Teresina, com restrições de recursos. Para realizar suas análises, o pesquisador está limitado em termos de tempo e recursos financeiros para a coleta das amostras de solo destas cidades. O desafio é encontrar uma rota eficiente entre essas cidades, minimizando tanto o tempo gasto quanto os custos associados à coleta, garantindo que cada local seja visitado uma única vez e que o pesquisador retorne ao ponto de partida após concluir todas as coletas.

É crucial para o pesquisador otimizar o uso de seus recursos limitados, evitando trajetos desnecessários e reduzindo os custos associados à coleta de amostras de solo. Isso se torna especialmente importante dada a natureza restrita de seus recursos disponíveis. Portanto, a aplicação do conceito do Ciclo de Hamilton nesta situação é de grande relevância prática. Ao utilizar essa abordagem, é possível economizar tanto tempo quanto recursos financeiros.

A área de coleta das amostras é a Ride da Grande Teresina, que compreende uma área de mais de 11.000 km² e é composta por 15 municípios, conforme mostrado na Figura 02a abaixo.



a b

Figura 02: (a) Ride da Grande Teresina (b) Grafo da Ride da Grande Teresina com as distâncias entre as cidades

Fonte: (a) Instituto de Pesquisa Econômica Aplicada. Imagem modificada. (b) Autores, 2024

A Figura 02b ilustra o grafo da Ride da Grande Teresina. Neste contexto, os vértices representam as cidades, enquanto as arestas denotam os caminhos entre elas, com

o custo associado sendo a distância em quilômetros. O grafo caracteriza-se como conexo e esparso e não direcional, sendo viável para um ciclo hamiltoniano.

3. Metodologia

Este trabalho adota uma abordagem metodológica que inclui pesquisa bibliográfica, análise quantitativa e experimentação prática. O estudo foi dividido em duas partes principais: pesquisa sobre o PCV para compreensão do problema e identificação de soluções existentes, seguida pela implementação e análise dos algoritmos de força bruta e uma heurística baseada no vizinho mais próximo, aplicadas a um caso real. Inicialmente, o problema foi modelado como um grafo, onde cada uma das 15 cidades da Ride da Grande Teresina é representada por um vértice, e as estradas entre essas cidades são representadas por arestas, com o peso correspondendo à distância em quilômetros. Este grafo foi representado por uma matriz de adjacência.

Foram implementados dois algoritmos: o método de força bruta, que explora todas as possíveis rotas para encontrar a solução ótima em termos de custos, e a heurística do vizinho mais próximo, que oferece soluções aproximadas com menor custo computacional. Ambos os métodos utilizaram o algoritmo de Dijkstra para calcular o caminho de menor custo de retorno ao ponto de partida. A pesquisa teve como principal objetivo compreender o PCV, investigar abordagens existentes de soluções viáveis e suas particularidades. Além disso, realizou-se uma análise detalhada dos benefícios e limitações de cada abordagem, essencial para embasar a construção da solução para PCV.

A análise empírica foi realizada com base no problema real (15 vértices e 27 arestas) e instâncias fictícias variadas entre 3 e 8 vértices, onde para cada tamanho de vértices foi associado uma quantidade de arestas em pelo menos três tamanhos diferentes, geradas aleatoriamente, no intuito de simular variedade de grafos esparsos e densos. Todos os testes foram conduzidos utilizando a linguagem de programação Python 3.10, no ambiente de desenvolvimento PyCharm, no sistema operacional Windows 10 Professional. O hardware utilizado para os testes consistiu em um processador Intel Core i7-9700K de 3.60GHz e 8GB de memória RAM DDR3 a 1600 MHz. Os resultados dos dois métodos foram comparados em termos de custo total da rota e tempo de execução, destacando a eficiência da heurística em relação ao método de força bruta. A análise permitiu avaliar a viabilidade computacional da heurística para instâncias maiores do problema.

4. Implementação

Neste problema da coleta de amostras do pesquisador nos 15 municípios da Ride da Grande Teresina, cada município é considerado um vértice, enquanto as 27 estradas mapeadas entre esses municípios são consideradas como as arestas do grafo, com o peso de cada aresta representando a distância a ser percorrida em quilômetros.

4.1. Instância do Problema

A instância do problema é modelada utilizando um grafo representado por uma matriz de adjacência, visando reduzir os custos de acesso aos pesos das arestas. Cada vértice, aresta e o próprio grafo foram considerados como parâmetros do problema. A matriz n×n é empregada, onde cada elemento G(i,j) corresponde ao valor do peso (distância em quilômetros) entre as cidades. As posições na matriz que contêm valor zero indicam a ausência de aresta entre os vértices. Além disso, foram adicionados métodos úteis ao algoritmo, como a lista de vértices adjacentes, que recebe o grafo, o vértice e o caminho

atual para excluir os vértices já visitados. Também é implementado o algoritmo de Dijkstra (Dijkstra, 1959), que retorna a menor distância entre dois pontos no grafo. Para construir o grafo do problema, o algoritmo recebe como entrada a lista de municípios em tabela, com seus índices correspondendo aos vértices do grafo, e a lista de distâncias entre os pontos (arestas). O grafo é então preenchido percorrendo os *n* vértices e as *m* arestas.

4.2. Força Bruta

O algoritmo de força bruta desenvolvido para resolver o PCV, foi implementado com o objetivo de solucionar o problema de forma geral, não se restringindo apenas à instância específica da Ride da Grande Teresina. Isso significa que a solução é aplicável a quaisquer grafos que representam problemas semelhantes, independentemente do número de vértices e arestas envolvidos. Como abordagem inicial (baseline), foi desenvolvido um algoritmo de força bruta, que percorre todas as possibilidades de combinações de vértices, assinalando qual caminho é válido, e comparando os custos para identificar o melhor. Como resultado teremos um conjunto de caminhos avaliados pela limitação do problema e pelos custos. O algoritmo busca determinar a trajetória de menor custo, partindo de um vértice inicial e percorrendo todos os vértices, retornando ao ponto inicial sem repetir o caminho (PAPADIMITRIOU & STEIGLITZ, 1998).

O algoritmo de força bruta seleciona apenas os caminhos que partem de um ponto inicial predefinido e explora essas combinações. Em cada iteração, é necessário validar se é um caminho que contém todas as cidades do grafo. Um dos métodos mais simples para resolver o PCV é a abordagem de força bruta, que envolve enumerar todas as permutações e combinações possíveis das cidades e selecionar a mais curta. Embora este método seja impraticável para grandes instâncias devido à sua complexidade fatorial de tempo, ele serve como base para comparação de algoritmos mais sofisticados (APPLEGATE et al., 2006).

A verificação inicial para garantir que cada cidade aparece exatamente uma vez requer um tempo de O(n). A soma total das distâncias do caminho e a distância para fechar o ciclo também podem ser calculadas em um tempo de O(n), assumindo que o acesso às distâncias entre as cidades seja O(1). A comparação entre as soluções para apontar a de menor distância requer um tempo de O(m), onde m é o conjunto de soluções encontradas. A estratégia utilizada para construir o caminho válido envolve buscar os vértices adjacentes disponíveis em cada recursão, deixando o vértice já visitado indisponível para os próximos passos. A cada recursão, o conjunto de vértices possíveis é decrementado de n para n−1, tornando a complexidade da escolha do próximo vértice O(n!), pois para cada possível vértice é criada uma cópia da solução. Assim, a complexidade total do algoritmo de força bruta é O(n!), uma vez que ele considera todas as permutações possíveis dos n vértices e realiza verificações e cálculos de custos para cada permutação (KARP, 1972).

Nesse sentido, o baseline foi definido em escolher o ponto inicial, e listar todas as possibilidades de combinação possível de vértices. Em seguida, o algoritmo compara todos os caminhos e seleciona apenas os que têm caminho válido, ou seja, que passam por aresta que exista. Na implementação com matriz, arestas que não existem são arestas que possuem valor zero na posição *E(i,j)*. Cada caminho selecionado é calculado o custo total, ou seja, a distância percorrida até o último vértice, adicionado o custo do caminho de volta para o vértice inicial. O caminho de volta é previamente calculado pelo algoritmo de Dijkstra, que retorna um mapa do menor caminho entre o vértice inicial até qualquer outro vértice. Assim, o menor caminho é a soma do custo total do algoritmo de força bruta com o custo de retorno disponível no mapa gerado pelo Dijkstra (CORMEN et al., 2012).

A seguir, o pseudocódigo que descreve a implementação do algoritmo de força bruta, baseado na descrição de Applegate et al. (2006).

Pseudocódigo Força Bruta

```
1.  Input: G, v
2.     cidades = VERTICES(G); caminhos = PRODUTO_VERTICES(cidades)
3.     menor_distancia = +∞;    menor_caminho = Null
4.     for i=0 to len(caminhos) do
5.        if caminho[0] = v then
6.           distancia_caminho = 0; caminho = caminhos[i]
7.           completo = True
8.           for k=0 to len(cidades) do
9.              if not caminho.contem(cidades[k]) then
10.                completo = False
11.          if completo == True then
12.             for j=0 to len(caminho)  do
13.                custo = G[caminho[j], caminho[j+1]
14.                if custo > 0 then
15.                   distancia_caminho += custo
16.               else:
17.                  distancia_caminho += +∞
18.          distancia_caminho += DIJKSTRA(primeiro_vertice, ultimo_vertice)
19.          if distancia_caminho < menor_distancia then
20.             menor_distancia = distancia_caminho;  menor_caminho = caminho
21. return menor_distancia, menor_caminho
```

5. Prova de que o Problema do Caixeiro Viajante é NP-Completo

A prova de que o PCV é NP-completo é crucial, pois mostra que ele pertence a uma classe de problemas sem soluções conhecidas em tempo polinomial, até o momento. Isso torna os métodos exatos inviáveis para problemas de grande escala, devido ao seu tempo de execução exponencial. Portanto, a prova de que o PCV é NP-completo não é apenas um exercício teórico, mas uma base fundamental que justifica a busca e o desenvolvimento de alternativas heurísticas. Essas abordagens são essenciais para lidar com a complexidade do PCV em situações reais, onde a otimização e a eficiência são essenciais. Para provar que o PCV é NP-completo, deve-se mostrar que ele é tanto NP quanto NP-difícil. A classe NP (Nondeterministic Polynomial time) é composta por problemas cujas soluções podem ser verificadas em tempo polinomial por uma máquina de Turing não determinística. Um problema é considerado NP-difícil se qualquer problema em NP pode ser reduzido a ele em tempo polinomial. Esta redução significa que uma solução para o problema NP-difícil pode ser usada para resolver qualquer problema em NP com eficiência semelhante. A prova de que o PCV é NP-difícil frequentemente envolve a redução de um problema já conhecido como NP-completo ao PCV. A redução a partir do problema do ciclo hamiltoniano, um problema conhecido por ser NP-completo, pode ser usada para mostrar que o PCV é NP-difícil (CORMEN et al., 2012). Segundo Karp (1972), "NP é o conjunto de linguagens derivadas de elementos de P (classe polinomial) por quantificação existencial limitada por polinômios".

Primeiramente, demonstra-se que o PCV pertence à classe NP. Um problema está em NP se, dada uma solução candidata, pode-se verificar a sua validade em tempo polinomial. No caso do PCV, o certificado é uma sequência de n vértices que representa

um percurso. O algoritmo de verificação deve confirmar que essa sequência contém cada vértice exatamente uma vez, realizando a soma dos custos das arestas no percurso e verificando se a soma dos custos é no máximo k. Esse processo de verificação pode ser realizado em tempo polinomial, pois envolve apenas a verificação de uma sequência de vértices e a soma dos custos das arestas, operações que são computacionalmente eficientes (CORMEN et al., 2012). Isso demonstra que o PCV pertence à classe NP, uma vez que sua solução pode ser verificada de forma polinomial.

Para demonstrar que o PCV é NP-difícil é necessário mostrar que um problema conhecido como NP-completo pode ser reduzido ao PCV em tempo polinomial. Utilizou-se neste estudo o problema do ciclo hamiltoniano para essa redução. Dada uma instância do ciclo hamiltoniano, representada por um grafo G=(V,E), com V representando o conjunto de vértices e E o conjunto de arestas, pode-se construir uma instância equivalente do PCV. Para isso, cria-se um grafo completo G'=(V,E'), onde V é o mesmo conjunto de vértices de G, e as arestas E' são definidas por uma função de custo c. Nessa função, o custo de uma aresta é 0 se ela pertencer a E e 1 caso contrário. Essa construção pode ser feita em tempo polinomial.

Resolver o PCV no grafo G' determinará se há um ciclo hamiltoniano em G. Mostra-se que o grafo G tem um ciclo hamiltoniano se e somente se o grafo G′ tem um percurso cujo custo é igual a 0. Caso contrário, não há um ciclo hamiltoniano em G. Se G possui um ciclo hamiltoniano, então existe um percurso em G′com custo total 0, pois todas as arestas do ciclo estarão no conjunto original E. Se não houver um ciclo hamiltoniano em G, qualquer percurso em G′ terá custo maior que 0, pois incluirá arestas não presentes em E (CORMEN et al., 2012). Portanto, o ciclo hamiltoniano pode ser reduzido ao PCV, demonstrando que o PCV é NP-difícil.

Com base na demonstração de que o PCV pertence à classe NP e que ele é NP-difícil, pode-se concluir que o PCV é um problema NP-completo. Esta conclusão tem implicações importantes para a solução de problemas práticos, como o pesquisador na Ride da Grande Teresina, onde a otimização dos recursos é essencial. Na literatura, o PCV é amplamente explorado e discutido. Os autores atuais ainda o categorizam como NP-completo, pois não há, até o momento, nenhuma solução matemática que reduza sua complexidade para tempo polinomial.

6. Heurística

A implementação de algoritmos baseados em heurísticas surge como uma alternativa à abordagem de força bruta, que pode demandar recursos computacionais indesejáveis devido à sua complexidade exponencial. Embora não garantam a solução ótima, as heurísticas proporcionam uma solução suficientemente boa dentro dos recursos computacionais disponíveis, resolvendo o problema mais rapidamente ou com menor uso de recursos, baseando-se em regras empíricas ou no conhecimento do problema específico (CORMEN et al., 2012). Para esse estudo, foram implementadas uma heurística e comparada com o desempenho de força bruta. A heurística consiste em um método guloso que seleciona o vértice mais próximo como ótimo local, ou seja, como ponto de partida ideal.

A metodologia da heurística emprega a recursividade no código, onde os vértices disponíveis são sempre os adjacentes que ainda não faziam parte do caminho. Assim como o algoritmo de força bruta, o cálculo do caminho de volta é realizado utilizando o algoritmo de Dijkstra, que retorna o menor caminho entre o vértice inicial e qualquer outro vértice (DIJKSTRA, 1959).

6.1 Heurística do Vizinho Mais Próximo

A heurística de Bellmore e Nemhauser (heurística do vizinho mais próximo) é uma abordagem clássica para resolver o PCV. Esta heurística consiste em partir de um vértice qualquer do grafo e, a cada passo, dirigir-se ao vértice mais próximo do vértice anterior, formando assim um caminho Hamiltoniano. Este método é caracterizado por sua estratégia gulosa, onde, a cada passo, o algoritmo toma a decisão de menor custo, ou seja, escolhe o vértice mais próximo ao vértice da extremidade do caminho parcial que está sendo formado (BELLMORE & NEMHAUSER, 1968).

A complexidade dessa heurística é $O(n^2)$, onde n é o número de vértices no grafo. Esta complexidade se deve ao fato de que, para cada vértice, o algoritmo deve percorrer a lista de vértices adjacentes para encontrar aquele com a menor aresta, e essa operação é repetida para todos os vértices do grafo. Uma variação desta heurística repete o algoritmo para todos os nós ou possíveis cidades para minimizar o efeito da escolha da cidade inicial, aumentando a complexidade para $O(n^3)$ (GOLDBARG & LUNA, 2000).

A heurística do vizinho mais próximo é eficiente em termos de tempo de execução comparada com métodos mais exaustivos como a força bruta. A sua complexidade $O(n^2)$ a torna prática para grafos de tamanho moderado. No entanto, assim como outras heurísticas gulosas, ela não garante uma solução ótima para o PCV. Em casos onde a estrutura do grafo é irregular ou há grande variabilidade nos pesos das arestas, a solução obtida pode estar longe da ótima. Goldbarg e Luna (2000) destacam que uma variação desta heurística, que repete o processo para todos os vértices como ponto inicial, pode ajudar a minimizar a influência da escolha do ponto de partida, embora aumente a complexidade para $O(n^3)$. Esta abordagem permite explorar múltiplas soluções iniciais e selecionar a melhor entre elas, melhorando a qualidade da solução final. A seguir, apresentamos o pseudocódigo que descreve a implementação do algoritmo da heurística do vizinho mais próximo apresentado por Bellmore & Nemhauser (1968).

Pseudocódigo da Heurística do Vizinho Mais Próximo

```
1.  Input: G, v
2.      caminho.add(v);
3.      while len(caminho) < n do
4.          v = caminho[-1]; disponiveis = DISPONIVEIS(v, caminho);
5.          menor_custo = +∞;   vertice_menor = Null;
6.          for vertice in disponiveis do
7.              if G[v, vertice] < menor_custo then
8.                  menor_custo = G[v, vertice];  vertice_menor = vertice;
9.          caminho.add(vertice_menor);
10.         if len(caminho) == n then
11.             Fechar o ciclo voltando ao início;
12. return menor_custo, caminho
```

7. Resultados

Os testes mostraram que o algoritmo de força bruta, embora garanta a solução ótima, tem um tempo de execução exponencialmente maior com o aumento do número de vértices e arestas. O tempo médio de execução atinge um pico de 1.792,22 segundos com 8 vértices, e não foi possível executar com maior quantidade por limitações de memória, evidenciando sua escalabilidade extremamente limitada. Em contraste, a heurística do vizinho mais próximo é significativamente mais rápida, com tempo de execução

consistentemente baixo, independentemente do tamanho do grafo, demonstrando alta eficiência e excelente escalabilidade, como ilustrado na Figura 03. Corroborando com esses resultados, a média geral do tempo de execução para a força bruta foi de 299,28 segundos (desvio padrão de 687,35), refletindo grande variação e ineficiência para grafos maiores. Enquanto a heurística do vizinho mais próximo tem uma média de 0,0011 segundos (desvio padrão de 0,0024), demonstrando rapidez e consistência.

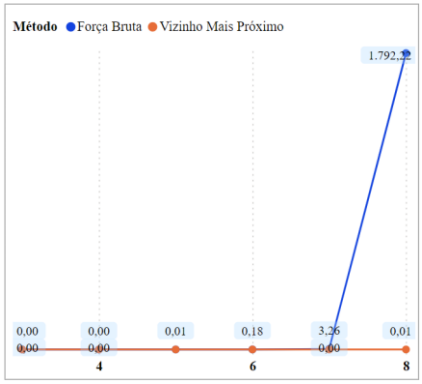


Figura 03: Comparação do tempo médio de execução (s) entre os algoritmos de força bruta e vizinho mais próximo em relação ao número de vértices.

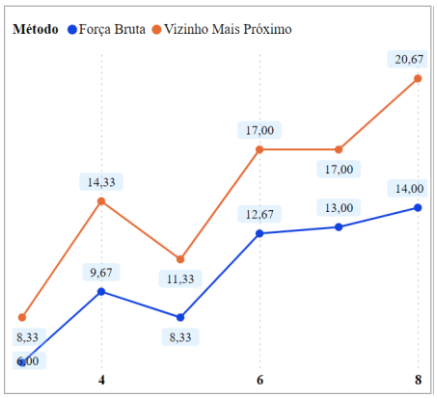


Figura 04: Comparação da distância média entre os algoritmos de força bruta e vizinho mais próximo em relação ao número de vértices.

O algoritmo de força bruta encontra distâncias menores, variando de 5 a 18 unidades, indicando soluções mais próximas do ótimo. Em contraste, a heurística do vizinho mais próximo mostra distâncias maiores, variando de 4 a 27 unidades, como ilustrado na Figura 04. Corroborando esses resultados, a média geral das distâncias para a força bruta é de 10,61 unidades (desvio padrão de 3,76), indicando menor variação e soluções mais otimizadas. Enquanto para o vizinho mais próximo é de 14,78 unidades (desvio padrão de 5,41), mostrando uma maior variabilidade e soluções menos otimizadas.

8. Conclusão

O estudo comprovou que, em problemas de otimização como o problema do caixeiro viajante, a heurística ofereceu uma solução prática e eficiente comparada ao método de força bruta, que é impraticável para grandes grafos. No caso do pesquisador analisar a qualidade do solo nas 15 cidades da Ride da Grande Teresina, a heurística mostrou-se viável, otimizando recursos limitados.

Para trabalhos futuros, pretende-se realizar a aplicação de meta-heurísticas como algoritmos genéticos, colônias de formigas e exame de abelha com o objetivo de melhorar

ainda mais a qualidade das soluções, equilibrando rapidez e precisão. Essas técnicas oferecem potencial significativo para resolver problemas de roteamento em diversas áreas práticas.

9. Referências

Applegate, D. L., Bixby, R. E., Chvátal, V., & Cook, W. J. (2006). *The Traveling Salesman Problem: A Computational Study*. Vol. 17. Princeton University Press.

Bellmore, M., & Nemhauser, G. L. (1968). *The Traveling Salesman Problem: A Survey*. Operations Research, 16(3).

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2012). *Algoritmos*. 3ª edição. Rio de Janeiro: Elsevier.

Dijkstra, E.W. (1959). *A note on two problems in connexion with graphs*. Numer. Math. 1, 269–271. Disponível em: https://doi-org.ez14.periodicos.capes.gov.br/10.1007/BF01386390, acessado em 29 de julho de 2024.

Goldbarg, M. C. e Luna, H. P. L. (2005). *Otimização combinatória e programação linear: modelos e algoritmos.* 2. ed. Rio de Janeiro: Elsevier.

Instituto De Pesquisa Econômica Aplicada. *Região Integrada de Desenvolvimento (Ride) da Grande Teresina*. Disponível em: https://portalantigo.ipea.gov.br/agencia/images/stories/PDFs/livros/livros/171208_atlas_idhm_desenvolvimento_humano_rm_teresina.pdf, acessado em 29 de julho de 2024.

Karp, R. M. (1972). Reducibility among Combinatorial Problems. *Complexity of Computer Computations*, 85–103. doi:10.1007/978-1-4684-2001-2_9.

Papadimitriou, C. H., & Steiglitz, K. (1998). *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications.