Analise do Problema do Caixeiro Viajante (PCV) Versão de decisão

Jorge Nery[a,1] **and Iarley Moraes**[a,2]

[a] *UFBA - Universidade Federal da Bahia*

Professor/George Marconi de Araujo Lima

Conteúdo

1. Motivação do trabalho

1.1. Objetivo

O Problema do Caixeiro Viajante (PCV) é um dos problemas mais estudados na área de teoria da computação, conhecido por sua complexidade e relevância prática em diversas áreas. [4]

Podendo ser enunciado da seguinte forma: *Dada uma lista de cidades e as distâncias entre todos os pares de cidades, encontrar uma rota mais curta (menor distância percorrida) que passa por cada uma das cidades exatamente uma vez e retorna para a cidade inicial. Sabe-se que este problema pertence classe de problemas conhecidos como NP-Difícil. [4] Na sua versão de decisão, que pertence à classe NP-Completo, o problema passa a ser determinar se há uma rota que passa por todas as cidades exatamente uma vez cujo tamanho não ultrapasse um dado valor L. PVC e sua versão de decisão está ainda fortemente relacionada com o problema de encontrar ciclos hamiltonianos em grafos.*

1.2. Motivação Pessoal/Acadêmica

Disciplina: IC0004 - Algoritmo e Grafos

1.3. Proposta do Trabalho

- Uma demonstração que a versão de decisão de PVC é NP-Completo.
- O uso de abordagens exatas e aproximadas para resolver PVC.
- Análise experimental de pelos menos três abordagens distintas.

2. Trabalho

A primeira etapa do trabalho consiste em provar que a versão de decisão do PCV é NP-Completo. Para isso, inicialmente, demonstramos que o PCV de decisão pertence à classe NP, ou seja, que, dado um certificado (uma rota) e resolvida em tempo polinomial se o comprimento da rota é menor ou igual a um valor L dado. Essa verificação é simples, pois envolve apenas a soma das distâncias ao longo da rota e a comparação com L

Em seguida, abordamos a redução de um problema NP-Completo conhecido, o problema do Ciclo Hamiltoniano (HC) [4], para o PCV. O HC consiste em verificar se existe um ciclo em um grafo que visite cada vértice exatamente uma vez. A redução é feita ao construir um grafo completo a partir do grafo do HC, atribuindo pesos baixos (por exemplo, 1) às arestas que fazem parte do ciclo Hamiltoniano e pesos muito altos (por exemplo, 9) às demais arestas. Se houver um ciclo Hamiltoniano, a solução do PCV no grafo modificado terá um comprimento igual ao número de vértices do grafo. Se essa solução existir e for menor ou igual ao número de vértices, então existe um ciclo Hamiltoniano no grafo original.

2.1. Prova de NP-Completo

2.1.1. Verificação de Certificado

Dado um conjunto de cidades e uma rota como certificado, podemos calcular a soma das distâncias entre as cidades na rota em tempo polinomial. Se essa soma for menor ou igual a L, o certificado é aceito, provando que o problema está em NP.

```python
def calcular_distancia(rota, distancias):
    return (
        sum(distancias[rota[i]][rota[i + 1]] for i in range(len(rota) - 1))
        + distancias[rota[-1]][rota[0]]
    )
```

Code 1. Distancia

Complexidade Computacional

$O(n)$

2.1.2. Redução de um Problema NP-Completo Conhecido (Ciclo Hamiltoniano)[3] para o PCV

Para mostrar que o problema do Ciclo Hamiltoniano é redutível ao problema do PCV - Caixeiro Viajante, transformamos uma instância do problema do Ciclo Hamiltoniano em uma instância do PCV de tal forma que a solução de um resolva o outro.

Dado um grafo $G = (V, E)$ onde queremos verificar se existe um Ciclo Hamiltoniano:

Criamos uma matriz de distâncias D para o PCV. A matriz D terá tamanho VXV, onde V é o numero de vértices no grafo G

$$D[i][j] = \begin{cases} 0, & \text{se } i = j \\ 1, & \text{se } (i, j) \in E \\ \infty, & \text{se } (i, j) \notin E \end{cases}$$

Definimos o valor $C = V$ como o custo que desejamos verificar no PCV. Este custo corresponde ao número de vértices, que é o número de arestas em um Ciclo Hamiltoniano.

A redução do problema do Ciclo Hamiltoniano para o PCV mostra que qualquer instância do problema do Ciclo Hamiltoniano pode ser resolvida usando um algoritmo para o PCV. Como o Ciclo Hamiltoniano é um problema NP-Completo, essa redução implica que o PCV é pelo menos tão difícil quanto o Ciclo Hamiltoniano, confirmando que o PCV também é NP-Completo.

2.1.3. Conclusão Prova

Essa redução, realizada em tempo polinomial, juntamente com a verificação em tempo polinomial do certificado e qualquer instância do problema NP-Completo pode ser transformada em uma instância do PCV em tempo polinomial, o PCV na versão de decisão é NP-Completo.

2.2. Implementação de Abordagens de Solução

Na segunda parte do trabalho, implementamos três algoritmos distintos para resolver o PCV: a força bruta, a programação dinâmica (algoritmo de Held-Karp) [1] e vizinhos mais próximo [2].

2.2.1. Algoritmo de força bruta

O algoritmo de força bruta gera todas as permutações possíveis das cidades e calcula a distância total para cada uma dessas rotas, retornando a rota de menor distância. Embora este método seja exato, sua complexidade $O(n!)$, tornando-o impraticável para um número grande de cidades [4].

```
from itertools import permutations


def calcular_distancia(rota, distancias):
    return (
        sum(distancias[rota[i]][rota[i + 1]] for i in range(len(rota) - 1))
        + distancias[rota[-1]][rota[0]]
    )


def forca_bruta(distancias, distancia_maxima):
    cidades = list(range(len(distancias)))
    melhor_rota = None
    menor_distancia = float("inf")

    for rota in permutations(cidades):
        distancia_atual = calcular_distancia(rota, distancias)
        if distancia_atual <= distancia_maxima and distancia_atual < menor_distancia:
            menor_distancia = distancia_atual
            melhor_rota = rota

    return melhor_rota, menor_distancia if melhor_rota else None
```

Code 2. Força Bruta

> **Complexidade Computacional**
>
> $O(n!)$

2.2.2. Programação dinâmica

A programação dinâmica, por sua vez, é implementada através do algoritmo de Held-Karp. Este algoritmo resolve o PCV de maneira mais eficiente que a força bruta, explorando subproblemas para evitar a recalculação de distâncias. A complexidade é $O(n^2 * 2^n)$, ainda exponencial, mas muito mais eficiente que a força bruta [5]

```
import itertools


def held_karp(distances, max_distance):
    n = len(distances)
    C = {}

    for k in range(1, n):
        C[(1 << k, k)] = (distances[0][k], 0)

    for subset_size in range(2, n):
        for subset in itertools.combinations(range(1, n), subset_size):
            bits = 0
            for bit in subset:
                bits |= 1 << bit
            for k in subset:
                prev_bits = bits & ~(1 << k)
                res = []
                for m in subset:
                    if m == 0 or m == k:
                        continue
                    res.append((C[(prev_bits, m)][0] + distances[m][k], m))
                C[(bits, k)] = min(res)

    bits = (2**n - 1) - 1
    res = []
    routes = []
    for k in range(1, n):
        res.append((C[(bits, k)][0] + distances[k][0], k))
        routes.append(C[(bits, k)][1])
    opt, parent = min(res)

    return routes, opt
```

Code 3. Held Karp

> **Complexidade Computacional**
>
> $O(n^2 * 2^n)$

2.2.3. Vizinhos mais Próximos

A abordagem Vizinhos mais Próximos na busca de alimentos para encontrar uma rota próxima ao ideal para o PCV - Problema do Caixeiro Viajante. Esse algoritmo pode ficar presa em mínimos locais, ou seja, uma escolha localmente ótima pode não resultar na melhor solução global[2]

```
def vizinho_mais_proximo(
    matriz_distancias, cidade_inicial=0, distancia_maxima=float("inf")
):
    n = len(matriz_distancias)
    visitadas = [False] * n
    rota = [cidade_inicial]
    distancia_total = 0

    cidade_atual = cidade_inicial
    visitadas[cidade_atual] = True

    for _ in range(n - 1):
        # Encontra a pr xima cidade com a menor dist ncia, n o visitada
        proxima_cidade = np.argmin(
            [
                matriz_distancias[cidade_atual][j] if not visitadas[j] else float("inf")
                for j in range(n)
            ]
        )
        distancia_para_proxima = matriz_distancias[cidade_atual][proxima_cidade]

        # Verifica se adicionar a pr xima cidade excede a dist ncia m xima
        if distancia_total + distancia_para_proxima > distancia_maxima:
            return None, None

        distancia_total += distancia_para_proxima
        rota.append(proxima_cidade)
        visitadas[proxima_cidade] = True
        cidade_atual = proxima_cidade

    # Adiciona a dist ncia de volta para a cidade inicial
    distancia_final = matriz_distancias[cidade_atual][cidade_inicial]

    if distancia_total + distancia_final > distancia_maxima:
        return None, None

    distancia_total += distancia_final
    rota.append(cidade_inicial)

    return rota, distancia_total
```

Code 4. Vizinhos mais Próximos

> **Complexidade Computacional**
>
> $O(n^2)$

3. Análise Experimental

A última parte do trabalho é dedicada à análise experimental das abordagens implementadas. Para isso, utilizamos conjuntos de cidades de tamanhos variados, com distâncias entre elas geradas aleatoriamente. Medimos o tempo de execução de cada algoritmo e comparamos a qualidade das soluções encontradas.

Os resultados mostram que o algoritmo de força bruta, como esperado, é impraticável para mais de 10 cidades, devido ao crescimento exponencial do tempo de execução. A programação dinâmica, embora mais eficiente, ainda enfrenta dificuldades para grandes conjuntos de dados, mas consegue encontrar a solução ótima. A Colônia de Formigas, por outro lado, oferece uma solução em tempo muito mais curto, mesmo que a rota encontrada não seja sempre a ótima.

Os gráficos resultantes ilustram claramente as diferenças de desempenho. O tempo de execução dos algoritmos de força bruta e de programação dinâmica cresce exponencialmente com o número de cidades, enquanto o de Vizinhos mais Próximos mantém um crescimento mais linear, permitindo a resolução de problemas maiores. Em termos de qualidade da solução, tanto a força bruta quanto a programação dinâmica garantem a solução ótima, enquanto a Vizinhos mais Próximos encontra uma solução próxima, mas em muito menos tempo.

Informação

A matriz de Distancias entre cidades utilizada foi a do site Melhores Rotas[6]
(https://www.melhoresrotas.com/tabela-de-distancias-entre-cidades/br-bahia/)

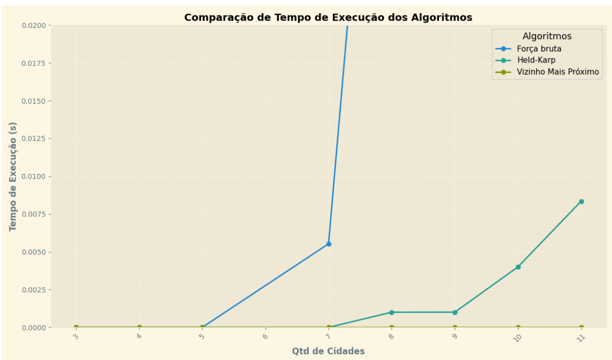

Figura 1. Gráfico Experimentos Comparativo de Tempo Execução

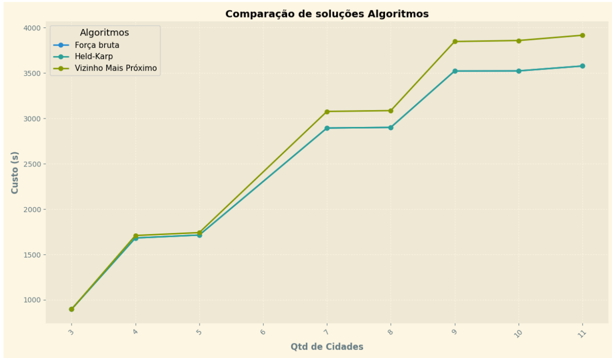

Figura 2. Gráfico Experimentos Comparativo de Soluções

4. Conclusão:

O estudo comparativo demonstra a importância de escolher o algoritmo adequado dependendo do tamanho e das restrições do problema. Para pequenos conjuntos de dados, métodos exatos são preferíveis, mas para grandes conjuntos, as meta-heurísticas são mais viáveis.

A análise conclui que, embora os métodos exatos sejam necessários para instâncias pequenas do PCV, para grandes instâncias, as meta-heurísticas como a Colônia de Formigas se mostram mais viáveis, equilibrando tempo de execução e qualidade da solução

Referências

[1] D. S. Johnson, L. A. McGeoch e E. E. Rothberg, «Asymptotic experimental analysis for the Held-Karp traveling salesman bound», em *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms*, ACM Press San Francisco, vol. 341, 1996, p. 350.

[2] G. Jäger e P. Molitor, «Algorithms and experimental study for the traveling salesman problem of second order», em *Combinatorial Optimization and Applications: Second International Conference, COCOA 2008, St. John's, NL, Canada, August 21-24, 2008. Proceedings 2*, Springer, 2008, pp. 211–224.

[3] A. C. Gusmão, L. R. Bueno e R. A. Hausen, «Um algoritmo paralelo para o problema de caminhos hamiltonianos em grafos Kneser», *XLV. Simpósio Brasileiro de Pesquisa Operacional (SBPO 2013), Natal-RN*, pp. 3102–3113, 2013.

[4] T. H. Cormen, *Introduction to Algorithms* (Fourth Edition). MIT Press, 2022, ISBN: 978-0262046305.

[5] Wikipedia. «Held-Karp Algorithm». Accessed: 2024-08-20. (2023), URL: https://en.wikipedia.org/wiki/Held%E2%80%93Karp_algorithm.

[6] Balaiocientífico. «Matriz de distância dos municípios brasileiros». Acesso em: 30 ago. 2024. (2024), URL: https://www.balaiocientifico.com/r/matriz-de-distancia-dos-municipios-brasileiros/.

Agosto 2024 ☻