UNIVERSIDADE FEDERAL DO VALE DO SÃO FRANCISCO
COLEGIADO DE CIÊNCIA DA COMPUTAÇÃO
CAMPUS SALGUEIRO-PE
Av. Antônio Angelim, 570, Santo Antonio, Salgueiro - PE, 56000-000
cicomp.salgueiro@univasf.edu.br

Relatório de Análise Algorítmica do Problema do Caixeiro Viajante

---

Integrantes

Emanuel Flávio
Gabriel Santos Garcez
Matheus Barros Rosa
João Paulo Brandão
Janyson Alves

Docente

Valdigleis da Silva Costa

Salgueiro-PE, 2024

1. INTRODUÇÃO

O problema do caixeiro viajante (PCV) consiste em um dos entraves mais difíceis e importantes da era contemporânea na ciência da computação, em particular na análise de complexidade de problemas e teoria dos grafos. O problema pode ser, em síntese, caracterizado pela busca de um ciclo hamiltoniano de menor custo. Apesar da simplicidade por trás do enunciado, tal problema é ainda classificado como NP-completo e NP-difícil, existindo, portanto, soluções exatas não satisfatórias, abordagens aproximadas que eventualmente impõem numerosas restrições ou métodos heurísticos sem garantia de obterem soluções suficientemente boas.

Sob esse prisma, este relatório analisa um subconjunto de abordagens para o PCV, suas limitações, potenciais pontos de melhoria encontrados na literatura e, por fim, discorre a implementação de soluções notáveis nas linguagens de programação C e Python, seguida da análise de complexidade temporal e espacial das mesmas.

2. FORMALIZAÇÃO DO PROBLEMA

Seja $G = \langle V, E, \varphi \rangle$ um grafo não direcionado, positivamente ponderado e completo tal que:

- V corresponde a um conjunto enumerável de vértices (ou cidades)
- $E = V \times V$ corresponde a um conjunto enumerável de arestas (ou caminhos)
- φ: $E \to R^*_+$ é uma função total que determina o peso das arestas do grafo. Em outros termos, dados três vértices arbitrários $u, v, w \in V$, a função φ representa o custo do deslocamento a partir de u diretamente em direção a v sem passar por qualquer outro vértice w ao longo do caminho.

Uma vez munido das definições acima, agora é possível estender φ para o custo de rotas com passagens únicas por vértices intermediários e com retorno direto do destino final. Nesse sentido, dados $i, j, k \in N^*$, defina
$H = \{ (v_1,v_2,...,v_k,v_1) \in V^{k+1} \mid$ existe $k \geq 2$ tal que para todo $i \leq j \leq k$, se $i \neq j$, então $v_i \neq v_j \}$, onde v_1 representa o vértice de partida. Em seguida, tome uma função total ψ: $H \to R^*_+$. Logo, o custo total de uma rota $\pi \in H$, denotado $\psi(\pi)$, resulta do somatório dos pesos individuais das arestas ao longo do caminho. Em outras palavras, tem-se que:

$$\psi(\pi) = \varphi(v_i, v_{i+1}) + \varphi(v_{i+1}, v_{i+2}) + \ldots + \varphi(v_{k-1}, v_k) + \varphi(v_k,v_1)$$

completando a definição de um ciclo dito hamiltoniano e o custo de uma trajetória qualquer sobre o mesmo.

Com isso, os esforços direcionados ao PCV são para encontrar o $x = \min(\psi(\pi_1), \psi(\pi_2), \ldots, \psi(\pi_n))$ em complexidade de tempo polinomial, onde $n \in N^*$ representa o total de combinações possíveis de rotas para uma instância do problema e $\pi_i \neq \pi_j$, sempre que $i \neq j$. Apresentados os pressupostos teóricos, a seção subsequente introduz o conceito de solução exata, aproximada, heurística e discorre sobre alguns algoritmos notáveis.

3. ANÁLISE E DISCUSSÃO

Antes de prosseguir com a apresentação das soluções contempladas neste relatório, é imperiosa uma definição das classificações das mesmas quanto à obtenção do resultado ótimo. Nesse sentido, uma solução para o PCV é dita exata se, e somente se, o custo total da rota é ótimo e determinístico para instâncias iguais do problema. Por outro lado, uma solução aproximada é aquela que objetiva minimizar o custo da rota, mas sem garantias de encontrar a solução de custo ótimo, mantendo um coeficiente de qualidade que indica o seu resultado no pior caso em relação à solução ótima para uma instância do problema.

O resultado do custo para instâncias iguais do problema pode ser diferente. Desse modo, as soluções aproximadas se desdobram em algoritmos heurísticos, gulosos e metaheurísticos e serão abordados logo em seguida nesta seção. A vantagem destas em detrimento do primeiro tipo de solução citado é que já existem algoritmos de complexidade de tempo polinomial.

3.1 MÉTODOS GULOSOS

São algoritmos que tomam decisões locais ótimas em cada passo, com a expectativa de que isso leve a uma solução globalmente boa. Entre as soluções aproximadas que utilizam métodos gulosos, tem-se aquela apresentada por (CORMEN et al., 2009) no capítulo 35, que recorre à formação de uma árvore geradora mínima e, em seguida, a realização de um percurso em pré-ordem sobre tal árvore. Para que a solução seja aplicável, no entanto, algumas imposições se fazem necessárias sobre o grafo. Por exemplo, a desigualdade triangular deve ser satisfeita, o grafo ser completo, positivamente ponderado e não direcionado.

Com tais restrições, o fator de qualidade do algoritmo corresponde a 2, indicando que o custo da solução obtida é, no pior caso, o dobro do custo de se percorrer a árvore geradora mínima. Uma forma eficiente de se implementar esse algoritmo é utilizando uma fila prioridade implementada sobre uma *heap* de Fibonacci estruturada em um arranjo unidimensional no algoritmo de Prim para árvores geradoras mínimas. Pseudocódigo para o algoritmo:

1. Defina um nó raiz r
2. Construa uma árvore geradora mínima T enraizada em r
3. Percorra T em pré-ordem
4. Vá diretamente do último nó v visitado até o nó raiz r
5. Retorne o percurso obtido

3.2 MÉTODOS HEURÍSTICOS

3.2.1 2-OPT

O algoritmo 2-opt é uma heurística simples e eficiente que funciona eliminando cruzamentos em rotas, ajustando localmente. Ele melhora iterativamente uma solução inicial, reduzindo o comprimento das rotas para encontrar um caminho melhor.

O algoritmo busca melhorar a solução inicial realizando trocas entre dois segmentos de uma rota, e em seguida verificando se isso resulta em uma redução do custo(caminho). Logo, a ideia central é reestruturar a rota para torná-la mais curta, semelhante ao algoritmo da subida da encosta, o algoritmo funciona da seguinte forma:

É utilizado a busca local para percorrer todas as combinações possíveis de dois segmentos consecutivos na rota. Para cada par de segmentos, realiza uma inversão da ordem dos vértices nesse intervalo e calcula o novo custo da rota.
Se o custo total da nova rota for menor que o custo da rota atual, a mudança é aceita. Caso contrário, ela é descartada. E esse processo continua até que nenhuma troca resulte em uma melhoria. A melhor solução encontrada ao longo das iterações é retornada como resultado.

2-OPT(Grafo, Rota)

```
melhor_rota ← Rota
melhor_tamanho ← CALCULAR-TAMANHO-ROTA(Grafo, melhor_rota)
melhor_encontrado ← VERDADEIRO

Enquanto melhor_encontrado faça
    melhor_encontrado ← FALSO

    Para i ← 1 até |melhor_rota| - 2 faça
        Para j ← i + 1 até |melhor_rota| - 1 faça
            nova_rota ← CÓPIA(melhor_rota)
            INVERTER-SUBROTA(nova_rota, i, j)

            novo_tamanho ← CALCULAR-TAMANHO-ROTA(Grafo, nova_rota)

            Se novo_tamanho < melhor_tamanho então
                melhor_rota ← nova_rota
                melhor_tamanho ← novo_tamanho
                melhor_encontrado ← VERDADEIRO
                Interrompa o laço interno (j)
        Se melhor_encontrado então
            Interrompa o laço externo (i)
Retorne melhor_rota
```

CALCULAR-TAMANHO-ROTA(Grafo, Rota)

```
tamanho ← 0
Para k ← 1 até |Rota| - 1 faça
    tamanho ← tamanho + Grafo[Rota[k], Rota[k+1]]
tamanho ← tamanho + Grafo[Rota[|Rota|], Rota[1]]
Retorne tamanho
```

3.2.1 MÉTODOS METAHEURÍSTICOS

As metaheurísticas são procedimentos que guiam outras heurísticas, ou seja, procedimentos computacionais, usualmente de busca de local, explorando o espaço de soluções além do ótimo local. As metaheurísticas consideram boas características das soluções encontradas para explorar novas regiões promissoras. Com isso podemos dizer que as metaheurísticas podem encontrar a solução ótima de um determinado modelo, mas não temos a garantia que a solução seja a ótima, logo podemos dizer que as metaheurísticas são usadas normalmente quando o problema é tão grande que o tempo de processamento do simplex torna-se infinito em relação à aplicação desejada. Problemas deste tipo são ditos serem NP-difícil ou NP-completo.

Entre os algoritmos metaheurísticos, a otimização da fisiologia da árvore (TPO) recorre ao método metaheurístico para encontrar soluções ótimas para problemas complexos de otimização, representando um “crescimento adaptativo” da árvore em busca de uma solução ideal. Este algoritmo pertence à categoria de algoritmos de otimização bio inspirados, que são modelos matemáticos que são baseados em comportamentos naturais, como o algoritmo genético e o enxame de partículas.

TPO: Início
Definir função objetivo f(S), onde S é a rota e f(S) é a distância total
Determinar a população inicial com:
- shoot Si (i = 1, 2, ..., n), representando as rotas iniciais
- root ri (i = 1, 2, ..., n), representando o custo da rota
- carbon Ci (i = 1, 2, ..., n), representando ajustes locais
- nutrient Ni (i = 1, 2, ..., n), representando mutações
Definir α, β, e o número de ramificações (branches)

Para t = 1 até número de iterações faça:
Para cada j = 1 até número de ramificações faça:
Avaliar a melhor solução global da ramificação atual, Spopbest
Comparar Spopbest com a melhor solução global encontrada, Sbest
Fim para

Para cada solução Si na população faça:
// Atualizar cada componente da população
Si^t ← Si^(t-1) + α * (Sbest - Si^(t-1)) + β * Ni^(t-1)
ri^t ← f(Si^t)
Ci^t ← distância entre Si^t e Si^(t-1)
Ni^t ← perturbação(Si^t)
Fim para
Fim para

Retornar a melhor rota Sbest e sua distância total f(Sbest)
Fim

3.2.1.1 VANTAGENS

1. Exploração e intensificação balanceadas:

A combinação de exploração (movimentos aleatórios) e intensificação (uso da melhor solução global) permite uma boa busca pelo espaço de soluções.

2. Flexibilidade:

O TPO pode ser adaptado para diferentes problemas de otimização, incluindo o TSP.

3. Escapando de ótimos locais:

Movimentos como 2−opt e atualizações globais ajudam a evitar o travamento em ótimos locais.

3.2.1.2 DESVANTAGENS

1. Complexidade computacional:

Com uma grande população ou muitas iterações, o algoritmo pode se tornar caro em termos de tempo computacional.

2. Dependência da população inicial:

Uma população inicial ruim pode levar a uma convergência mais lenta.

3. Qualidade subótima:

Em alguns casos, o TPO pode não ser competitivo com algoritmos mais especializados para o TSP, como o algoritmo genético.

4. CÁLCULO DA COMPLEXIDADE

4.1 CONSTRUÇÃO DO GRAFO

Descrição:
Inicializa o grafo, alocando memória para os vértices e listas de adjacência. Cada vértice é configurado com uma lista vazia e um ponteiro para o último vizinho.

Complexidade:

- **Tempo:** O(n), já que percorre todos os n vértices para inicializar as estruturas.
- **Espaço:** O(n), para armazenar as informações dos vértices.

4.2 INSERÇÃO DE ARESTAS

Descrição:
Adiciona uma aresta entre dois vértices (u, v), verificando previamente se já existe.

Complexidade:

- Para verificar se uma aresta já existe, percorre a lista de adjacência do vértice u, com custo proporcional ao grau do vértice. Isso é O(d), onde d é o número de vizinhos de u.
- Inserir a aresta é O(1), utilizando um ponteiro que aponta para o último vizinho.
- Para todas as arestas do grafo, o custo total é O(E), onde E é o número de arestas.
- **Espaço:** O(E), pois são armazenadas todas as arestas.

4.3 ALGORITMO DE PRIM

Descrição:
Constrói a Árvore Geradora Mínima (MST) utilizando uma fila de prioridade para selecionar vértices e ajustar os custos.

Operações no Algoritmo:

1. Inicializa as chaves dos vértices como infinito e insere todos na fila. Custo: O(n).
2. Extrai o vértice com menor chave da fila. Essa operação é repetida n vezes e tem custo O(n log n) para uma fila baseada em heap.
3. Atualiza as chaves dos vértices adjacentes. Para cada aresta adjacente, a atualização tem custo O(log n), resultando em O(E log n) para todas as arestas.

Complexidade Total: O((E + n) log n).
Espaço: O(n + E), para armazenar o grafo e a fila de prioridade.

4.4 PERCURSO EM PRÉ-ORDEM

Descrição:
Realiza um percurso na Árvore Geradora Mínima (MST) para gerar um ciclo aproximado.

Complexidade:

- O percurso visita todos os n vértices e percorre as listas de adjacência, com custo O(V + E).
- **Espaço:** O(n), para armazenar o ciclo.

4.5 FUNÇÃO PRINCIPAL

Descrição:
Orquestra o programa: constrói o grafo, calcula a MST, realiza o percurso em pré-ordem e exibe os resultados.

Complexidade Total:

- **Tempo:** O((E + n) log n).
- **Espaço:** O(n + E).

5. CASO DENSO vs. ESPARSO

5.1 CASO ESPARSO:

Se o grafo é esparso, o número de arestas é proporcional ao número de vértices (E aproximadamente O(n)).

- **Complexidade Temporal:** O(n log n).
- **Complexidade Espacial:** O(n).

5.2 CASO DENSO:

Se o grafo é denso, o número de arestas é proporcional ao quadrado do número de vértices (E aproximadamente $O(n^2)$).

- **Complexidade Temporal:** $O(n^2 \log n)$.
- **Complexidade Espacial:** $O(n^2)$.

6. QUADRO-RESUMO DAS COMPLEXIDADES

Operação	Complexidade (Tempo)	Complexidade (Espaço)
Construção do Grafo	O(n)	O(n)
Inserção de Arestas	O(E)	O(E)
Algoritmo de Prim	O((E + n) log n)	O(n + E)
Percurso em Pré-Ordem	O(V + E)	O(n)
Total (Esparso)	O(n log n)	O(n)
Total (Denso)	$O(n^2 \log n)$	$O(n^2)$