Abner Gomes Guimarães e Felipe Correia Araujo

Heurísticas

Alfenas
Maio, 2024

RESUMO

Este relatório apresenta uma análise das heurísticas aplicadas em diferentes contextos. O estudo inclui métodos e resultados relevantes obtidos durante a pesquisa. Vamos apresentar e demonstrar a aplicação de heurísticas e meta-heurísticas. Nosso principal problema é o clássico Problema do Caixeiro Viajante. Vamos demonstrar soluções e a análise dessas soluções utilizando algoritmos de busca local e processo evolutivo em heurísticas.

SUMÁRIO

1 INTRODUÇÃO

O problema principal que vamos analisar e definir suas soluções é o problema do caixeiro viajante (PCV). O caixeiro viajante é um problema de otimização combinatória que é intensamente investigado em matemática computacional devido à sua vasta área de aplicação e complexidade de obtenção de uma solução real.

O caixeiro viajante pode ser representado como um ciclo hamiltoniano em um grafo de n vértices, onde todos os vértices têm que ser visitados apenas uma vez. Após passar por todos os vértices, no final devemos chegar no vértice de origem, ou seja, de onde começamos. Essa descrição pode ser representada como um ciclo hamiltoniano.

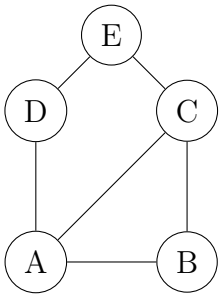

Figura 1 – Representação de um ciclo hamiltoniano.

Podemos observar na figura 1 que podemos começar em qualquer vértice e voltar para o vértice origem, ou seja, suponha que A, B, C, D e E são cidades. Se o caixeiro viajante começar na cidade A, ele pode passar por todas as cidades e voltar para a cidade original de onde ele começou, que é A.

Com essa descrição do problema conseguimos observar que existem diversas soluções possíveis. Na Figura 1, se começamos na cidade A, podemos ir para B ou para D, mas não podemos ir para C pelo fato de não conseguir passar por todas as cidades apenas uma vez. Temos diferentes caminhos possíveis para escolher, e esses caminhos têm distâncias diferentes, pesos diferentes. Veja a Figura 2 abaixo de um grafo ponderado.

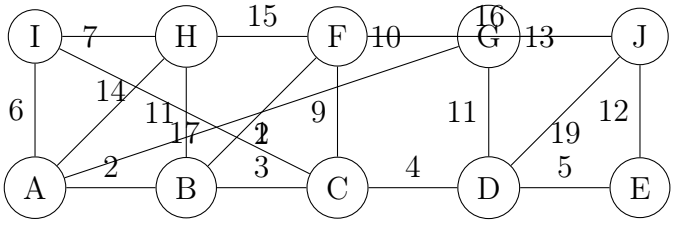

Figura 2 – Grafo ponderado com vértices de A a J.

Na Figura 2, tem a representação de um grafo ponderado de vértices de A a J, podemos observar que temos vários caminhos, 10 vértices, então podemos ter várias soluções.

Como vamos encontrar a melhor opção possível para o caixeiro viajante? Qual o melhor vértice para começar? Qual o melhor caminho para escolher? Será que o melhor é sempre ir pelo o menor caminho utilizando um algoritmo guloso? Podemos responder todas essas perguntas explicando a funcionalidade de Heurísticas e Meta-heurísticas.

1.1 Heurísticas

Heurísticas são estratégias ou métodos baseados em experiências práticas e intuição para resolver problemas complexos de maneira eficiente, embora não garantam uma solução ótima. No contexto de problemas de otimização combinatória, como o Problema do Caixeiro Viajante (PCV), heurísticas são utilizadas para encontrar soluções boas o suficiente em um tempo razoável, especialmente quando métodos exatos são impraticáveis devido à complexidade do problema.

1.1.1 Algoritmos Gulosos

Um algoritmo guloso (ou ganancioso) é um tipo de heurística que constrói uma solução peça por peça, escolhendo a opção localmente ótima em cada etapa, na esperança de encontrar uma solução globalmente ótima. No caso do PCV, um algoritmo guloso poderia funcionar da seguinte maneira:

1. Começa em um vértice arbitrário.
2. Em cada passo, escolhe a próxima cidade não visitada que está mais próxima da atual.
3. Repete até que todas as cidades sejam visitadas.
4. Retorna ao ponto de partida.

Embora simples e rápido, o algoritmo guloso nem sempre encontra a solução ótima para o PCV, mas pode fornecer uma boa aproximação.

1.1.2 Outras Heurísticas Comuns para o PCV

- **Nearest Neighbor (Vizinho Mais Próximo)**: Similar ao algoritmo guloso descrito acima, sempre escolhe o vértice mais próximo que ainda não foi visitado.
- **Minimum Spanning Tree (Árvore Geradora Mínima)**: Constrói uma árvore geradora mínima para o grafo e usa essa árvore para criar um ciclo Hamiltoniano.
- **Christofides' Algorithm**: Um algoritmo mais sofisticado que combina uma árvore geradora mínima com emparelhamento perfeito em um grafo de vértices de grau

ímpar, garantindo uma solução com um limite superior de 1.5 vezes o valor da solução ótima.

- **Algoritmos de Busca Local**:
  - **2-opt, 3-opt**: Modificam a solução atual trocando dois ou três arestas para obter uma nova solução com um caminho menor.
  - **Simulated Annealing (Anelamento Simulado)**: Uma técnica inspirada no processo de recozimento em metalurgia, que permite piorar a solução atual ocasionalmente para escapar de mínimos locais.
  - **Tabu Search (Busca Tabu)**: Utiliza uma lista tabu para evitar revisitar soluções recentemente exploradas.

1.1.3 Algoritmos de Busca Local

Os algoritmos de busca local são estratégias que exploram o espaço de soluções vizinhas para encontrar uma solução melhor. A seguir, apresentamos dois exemplos populares de algoritmos de busca local: VND e Simulated Annealing.

1.1.3.1 Variable Neighborhood Descent (VND)

O algoritmo VND é uma abordagem que explora diferentes vizinhanças de uma solução atual, buscando evitar mínimos locais e encontrar soluções de melhor qualidade.

- **Passo 1:** Começa com uma solução inicial s e define um conjunto de estruturas de vizinhança N_k.
- **Passo 2:** Para cada k no conjunto de vizinhanças, aplique uma busca local usando a vizinhança N_k.
- **Passo 3:** Se uma melhoria é encontrada, a solução atual é atualizada e o processo reinicia a partir da primeira vizinhança.
- **Passo 4:** Se nenhuma melhoria é encontrada, passa para a próxima vizinhança.
- **Passo 5:** Repete o processo até que todas as vizinhanças tenham sido exploradas sem melhorias.

```
Pseudocódigo:
1. Inicializar solução s
2. Definir conjunto de vizinhanças {N1, N2, ..., Nk}
3. k = 1
4. Enquanto k <= K faça
5.     s' = Melhor solução na vizinhança Nk(s)
6.     Se f(s') < f(s) então
7.         s = s'
8.         k = 1
9.     Senão
10.        k = k + 1
11. Fim enquanto
```

1.1.4 Meta-heurísticas

Além das heurísticas, temos as meta-heurísticas, que são métodos de alto nível que guiam outras heurísticas para explorar o espaço de soluções de maneira mais eficiente:

- **Genetic Algorithms (Algoritmos Genéticos)**: Inspirados pela evolução biológica, usam operações como seleção, crossover e mutação para evoluir um conjunto de soluções.
- **Ant Colony Optimization (Otimização por Colônia de Formigas)**: Baseada no comportamento das formigas reais, que depositam feromônios para encontrar o caminho mais curto entre a colônia e a comida.

O uso de heurísticas é comum em problemas de otimização devido à sua capacidade de encontrar soluções aproximadas em tempo razoável. Em problemas de complexidade alta ou NP-difíceis, onde encontrar a solução ótima é impraticável, as heurísticas oferecem uma abordagem viável para encontrar soluções aceitáveis. Algumas situações em que é benéfico usar heurísticas incluem:

- Problemas de Otimização Combinatória
- Complexidade NP-difícil
- Limitações de Recursos
- Aplicabilidade Prática

Essas heurísticas permitem encontrar soluções próximas o suficiente para serem úteis em uma variedade de contextos do mundo real, como logística, planejamento de rotas, alocação de recursos e muito mais. No entanto, é importante ressaltar que as soluções

encontradas por heurísticas podem não ser ótimas, mas muitas vezes são suficientes para atender às necessidades práticas.

2 DESENVOLVIMENTO

Neste capítulo, explicaremos as heurísticas e processos evolutivos utilizados para resolver o Problema do Caixeiro Viajante (PCV). Utilizamos uma abordagem mista que combina heurísticas construtivas, heurísticas de busca local e meta-heurísticas. Começaremos explicando a geração de grafos completos aleatórios, seguida pela descrição do método de Descida em Vizinhança Variável (VND) com a técnica de 2-opt, a heurística construtiva aleatória, o algoritmo guloso aleatório, e o algoritmo genético com seleção por torneio.

2.1 Geração de Grafos Completos Aleatórios

Para nossos experimentos, utilizamos um algoritmo para criar grafos completos aleatórios. Em um grafo completo, todos os vértices estão conectados entre si. Os pesos das arestas são atribuídos aleatoriamente dentro de um intervalo especificado.

2.1.1 Pseudocódigo para Geração de Grafos Completos Aleatórios

```
1. Inicializar lista de vértices V com n vértices
2. Para cada par de vértices (i, j) em V
3.     Atribuir um peso w_ij aleatório à aresta (i, j)
4. Retornar o grafo completo com os pesos atribuídos
```

A geração de grafos completos aleatórios nos permite testar os algoritmos de heurísticas e meta-heurísticas em diferentes cenários com estruturas variadas de grafos.

2.2 Heurística Construtiva Aleatória

A heurística construtiva aleatória que utilizamos começa selecionando vértices aleatórios do grafo de forma que eles não se repitam até que todos os vértices sejam visitados. Esta abordagem simples gera uma solução inicial que pode ser melhorada posteriormente com técnicas de busca local e meta-heurísticas.

2.2.1 Pseudocódigo da Heurística Construtiva Aleatória

```
1. Inicializar lista de vértices não visitados V
2. Escolher um vértice inicial s aleatoriamente de V
3. Enquanto V não estiver vazio
4.     Escolher um vértice v aleatoriamente de V
5.     Adicionar v à solução
6.     Remover v de V
7. Adicionar o vértice inicial ao final da solução para completar o ciclo
```

2.3 Algoritmo Guloso Aleatório

O algoritmo guloso aleatório seleciona vértices aleatoriamente, mas de forma que o próximo vértice escolhido seja um dos mais próximos do vértice atual, mantendo a propriedade de não repetição de vértices até que todos sejam visitados.

2.3.1 Pseudocódigo do Algoritmo Guloso Aleatório

```
1. Inicializar lista de vértices não visitados V
2. Escolher um vértice inicial s aleatoriamente de V
3. Enquanto V não estiver vazio
4.     Escolher um vértice v aleatoriamente entre os mais próximos de s
5.     Adicionar v à solução
6.     Remover v de V
7. Adicionar o vértice inicial ao final da solução para completar o ciclo
```

2.4 Descida em Vizinhança Variável (VND) com 2-opt

A VND é uma técnica que explora múltiplas vizinhanças para evitar mínimos locais. Usamos a técnica de 2-opt, que envolve trocar duas arestas em uma solução atual para obter uma nova solução com um caminho potencialmente menor.

2.4.1 2-opt

O algoritmo 2-opt remove duas arestas e reconecta os dois caminhos resultantes de uma forma diferente para reduzir o comprimento total do caminho. Isso é repetido até que nenhuma melhora adicional seja encontrada.

2.4.2 Pseudocódigo do VND com 2-opt

```
1. Inicializar solução s
2. Definir conjunto de vizinhanças {N1, N2, ..., Nk}
3. k = 1
4. Enquanto k <= K faça
5.     s' = Melhor solução na vizinhança Nk(s) usando 2-opt
6.     Se f(s') < f(s) então
7.         s = s'
8.         k = 1
9.     Senão
10.        k = k + 1
11. Fim enquanto
```

2.5 Algoritmo Genético

Os algoritmos genéticos (AG) são inspirados pela evolução biológica. Usamos um método de seleção por torneio, onde um subconjunto da população é escolhido aleatoriamente e o melhor indivíduo do subconjunto é selecionado para a reprodução.

2.5.1 Pseudocódigo do Algoritmo Genético com Seleção por Torneio

```
1. Inicializar uma população P de soluções
2. Avaliar a aptidão de cada solução em P
3. Enquanto não atingir critério de parada
4.     Selecionar pais usando torneio
5.     Aplicar crossover para gerar novos indivíduos
6.     Aplicar mutação nos novos indivíduos
7.     Avaliar a aptidão dos novos indivíduos
8.     Substituir a população antiga com os novos indivíduos
9. Retornar a melhor solução encontrada
```

Com estas abordagens combinadas, esperamos encontrar soluções eficientes para o Problema do Caixeiro Viajante, equilibrando qualidade da solução e tempo de execução.

3 DESCRIÇÃO DE INSTÂNCIA

Para ilustrar a aplicação das heurísticas e meta-heurísticas discutidas, vamos definir uma instância específica do Problema do Caixeiro Viajante. Considere um grafo ponderado onde os vértices representam cidades e as arestas representam as distâncias entre elas. A matriz de distâncias entre as cidades é dada por:

	A	B	C	D
A	0	2	9	10
B	1	0	6	4
C	15	7	0	8
D	6	3	12	0

Esta matriz representa as distâncias diretas entre as cidades A, B, C e D. O objetivo é encontrar a rota de menor custo que visite cada cidade exatamente uma vez e retorne à cidade de origem.

3.1 Implementação e Resultados

Nesta seção, vamos descrever a implementação das heurísticas e meta-heurísticas mencionadas, utilizando a linguagem Python. Um dos componentes essenciais é a função de cálculo de distância, que computa a distância total de uma rota específica.

3.2 Função calcDist

A função `calcDist` recebe uma rota (uma permutação de cidades) e a matriz de distâncias, e retorna a distância total percorrida pela rota. Abaixo está a implementação em C:

```
Função calcDist(route, dist_matrix)
    total_distance = 0
    num_cities = tamanho de route
    Para cada cidade i de 0 até num_cities-1
        from_city = route[i]
        to_city = route[(i + 1) % num_cities]
        total_distance += dist_matrix[from_city][to_city]
    Retorne total_distance

Rota = [0, 1, 2, 3] // Representa a rota A -> B -> C -> D -> A
```

```
Matriz_de_distância:
]
    [0, 2, 9, 10],
    [1, 0, 6, 4],
    [15, 7, 0, 8],
    [6, 3, 12, 0]
]
Escreva calcDist(Rota, Matriz_de_distância)
```

Nesta função, `route` é uma lista de índices que representam a ordem das cidades a serem visitadas, e `dist_matrix` é a matriz de distâncias. A função soma as distâncias entre cidades consecutivas na rota e inclui a distância de retorno à cidade de origem.

4 CONCLUSÃO

Concluímos que as heurísticas e meta-heurísticas são ferramentas valiosas para resolver o Problema do Caixeiro Viajante (PCV). Durante nossa análise, observamos que as heurísticas construtivas, como algoritmos gulosos e heurísticas aleatórias, são eficazes para gerar soluções iniciais, enquanto as meta-heurísticas, como algoritmos genéticos e busca local variável, são úteis para refinar essas soluções e encontrar soluções de melhor qualidade. Embora as soluções encontradas por heurísticas e meta-heurísticas possam não ser ótimas em todos os casos, elas oferecem uma abordagem prática e eficiente para resolver o PCV em tempo razoável, especialmente para instâncias de tamanho moderado. No entanto, é importante ressaltar que a escolha da heurística ou meta-heurística mais adequada depende do contexto do problema e dos requisitos específicos do usuário. Portanto, recomenda-se explorar diferentes técnicas e ajustar os parâmetros conforme necessário para obter os melhores resultados em cada situação. Em resumo, este estudo destaca a importância das heurísticas e meta-heurísticas como ferramentas poderosas para resolver problemas complexos de otimização, como o PCV, e destaca a necessidade contínua de pesquisa e desenvolvimento nessas áreas para enfrentar desafios futuros.