CENTRO DE INFORMÁTICA - UFPB

Etapa 2 - Parte 2

Relatório de Representação da Solução

Yuri da Costa Gouveia

11328072

João Pessoa, Abril de 2019

1. Descrição do Problema

Este relatório visa explicar a heurística construída para resolução do problema do caixeiro viajante. Na seção 2, será abordado o método utilizado para a construção inicial da solução. Já na seção 3, será descrita lógica utilizada para alcançar uma solução de otimização utilizando movimentos de vizinhança.

2. Construção

Para a construção inicial do problema foi utilizado o método do vizinho mais próximo. Dado um ponto partida (inicialmente a posição 0 da matriz, no método desenvolvido) o de chegada é escolhido com base na distância entre o primeiro e os seus vizinhos, de modo que o ponto de chegada escolhido será o seu vizinho mais próximo. Esse procedimento termina quando todas os vértices são visitados e no fim, retorna-se para o ponto inicial.

procedimento $ConstrucaoGulosa(g(.), s)$;
1 $s \leftarrow \emptyset$;
2 Inicialize o conjunto C de elementos candidatos;
3 enquanto $(C \neq \emptyset)$ faça
4 $\quad g(t_{melhor}) = melhor\{g(t) \mid t \in C\}$;
5 $\quad s \leftarrow s \cup \{t_{melhor}\}$;
6 $\quad$ Atualize o conjunto C de elementos candidatos;
7 fim-enquanto;
8 Retorne s;
fim $ConstrucaoGulosa$;

Figura 1: Pseudo-código da solução gulosa do vizinho mais próximo.

```
def vizinho_mais_proximo(pontoPartida, numNos, matriz, visitados, caminho, distancia):
    minimo = sys.maxsize

    for noChegada in range(0, numNos):                    # Loop de escolha do vizinho mais próximo
        if matriz[pontoPartida][noChegada] > 0 and matriz[pontoPartida][noChegada] < minimo and visitados[noChegada]==False:
            minimo = matriz[pontoPartida][noChegada]

    distancia += minimo
    visitados[matriz[pontoPartida].index(minimo)] = True    # Marca ponto como visitado
    caminho.append(matriz[pontoPartida].index(minimo))      # Adiciona ponto aos caminhos

    if len(caminho) < numNos:                               # Recursividade para achar o caminho do vizinho mais próximo
        return vizinho_mais_proximo(matriz[pontoPartida].index(minimo), numNos, matriz, visitados, caminho, distancia)
    else:
        return visitados, caminho, distancia
```

Figura 2: Implementação do vizinho mais próximo.

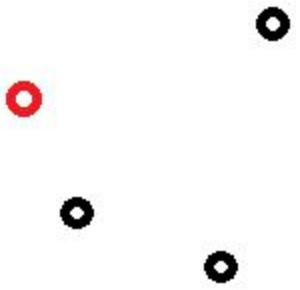

Figura 3: Representação de pontos a se percorrer pelo PCV.

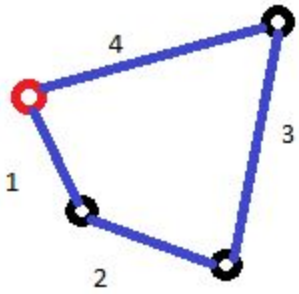

Figura 4: Representação da solução do vizinho mais próximo..

3. Multi Start

A metaheurística utilizada para gerar uma possível otimização da construção inicial foi o Multi Start.

Após a fase de construção, comum caminho já construído, faz-se a criação de um percurso de maneira aleatória. Com um caminho aleatório gerado é realizada uma busca local para realizar troca de posições, ligando cidades mais próximas e ao final retorna-se o percurso otimizado. Caso esse percurso otimizado tenha um menor custo para se percorrer do que o gerado inicialmente, esse valor de otimização é escolhido. Um critério de parada foi definido, para evitar que o código fique executando por muito tempo (o critério escolhido permite n laços, onde n é igual ao número de nós), retornando o valor inicial de construção.

```python
def metaheuristica(numNos, matriz):
    visitados, caminho, distancia = construcao_gulosa(n, matriz)
    criterioParada = 0

    print("A distância da construção inicial foi: " + str(distancia))

    while criterioParada < 5000:
        caminhoAleatorio, distanciaAleatoria = gera_solucao_aleatoria(numNos, matriz)
        novoCaminho = busca_local(caminhoAleatorio, matriz)
        novaDist = custo(matriz, novoCaminho)

        if novoCaminho != None and novaDist < distancia:
            caminho = novoCaminho.copy()
            distancia = novaDist
            return caminho.copy(), distancia
        else:
            criterioParada += 1
    return caminho.copy(), distancia
```

Figura 6: Implementação doMulti Start.

Os resultados apresentados na heurística de construção apresentam uma mesma distância percorrida para uma determinada instância sempre que executado, logo isso explica o porquê do GAP apresentar um valor 0.

Heurística de Construção

Instâncias	Média	Melhor	Tempo medio (seg.)	GAP
tsp1.txt	251.508	251.508	0,38	0
tsp2.txt	69.630	69.630	0,39	0
tsp3.txt	24.000	24.000	0,32	0

Metaheurística de Otimização

Instâncias	**Média**	**Melhor**	**Tempo medio (seg.)**	**GAP**
tsp1.txt	237.584,33	235.673	5,73	0,8
tsp2.txt	69.630	69.630	2096.80	0
tsp3.txt	24.000	24.000	2049.19	0