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.

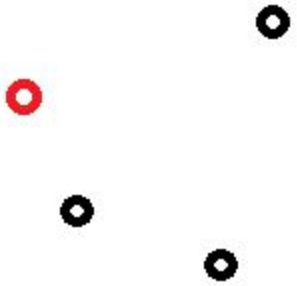

Figura 3: Representação de pontos a se percorrer pelo PCV.

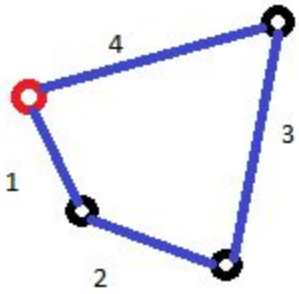

Figura 4: Representação da solução do vizinho mais próximo..

3. Self Annealing

A metaheurística utilizada para gerar uma possível otimização da construção inicial foi o Self Annealing.

O funcionamento desse metaheurística se dá através de condições iniciais (temperatura inicial, temperatura mínima e a taxa de esfriamento). A otimização é verificada em um laço, que se repete enquanto a temperatura atual for maior que a mínima.

Com uma rota inicial, calcula-se uma rota adjacente aleatória e a partir desta verifica-se a sua aceitação. Essa aceitação se dá por meio da fórmula probabilística:

```
probabilidadeAceitacao = np.exp((-1)*(distanciaAdjacente - distanciaAtual) / temperatura)
```

Se esse valor da probabilidade for maior do que um número aleatório gerado, então a rota é aceita.

```python
def self_anealing(temperatura, caminhoAtual, matriz, numNos):
    caminhoMinimo = caminhoAtual
    caminhoAdjacente = []*numNos

    while temperatura > TEMPERATURA_MINIMA:
        #print(str(custo(matriz, caminhoAtual)) + " | " + str(round(temperatura,2)))
        caminhoAdjacente = obter_caminho_adjacente(caminhoAtual, numNos)

        distanciaAtual = custo(matriz, caminhoAtual)            # Calculo dos custos das distancias
        distanciaMinima = custo(matriz, caminhoMinimo)
        distanciaAdjacente = custo(matriz, caminhoAdjacente)

        if distanciaAtual < distanciaMinima:        # Se a distancia do caminho atual for menor que a minima, ...
            caminhoMinimo = caminhoAtual                                        # ... a minima como sendo a atual

        if aceita_rota(distanciaAtual, distanciaAdjacente, temperatura):
            caminhoAtual = caminhoAdjacente

        temperatura *= 1-TAXA_ESFRIAMENTO                                   # Formula de decrescimo da temperat

    return caminhoMinimo
```

Figura 6: Implementação do Self Annealing.

Os resultados apresentados do Simulated Annealing sempre mostraram uma distância final maior do que a da heurística de construção e os valores nunca mudavam, logo isso explica o porquê do GAP sempre apresentar um valor 0.

Heurística de Construção

Instâncias	Média	Melhor	Tempo medio (seg.)	GAP (%)
tsp1.txt	2074443	2074443	0,368	0
tsp2.txt	106404	106404	0,395	0
tsp3.txt	30930	30930	0,323	0

Metaheurística de Otimização

Instâncias	Média	Melhor	Tempo medio (seg.)	GAP (%)
tsp1.txt	2095963	2095963	423,34	0
tsp2.txt	109469	109469	426,57	0
tsp3.txt	31880	31880	431,09	0