Experimento - Problema do Caixeiro Viajante

Ana C. Knop[1], **Carlos S. Mondo**[1], **Leandro M. da Silva**[1]

[1]ENGETEC – Universidade da Região de Joinville (Univille)
Caixa Postal 246 – 89201-972 – Joinville – SC – Brazil

{ana.knop, carlosm, leandrosilva.1}@univille.br

Resumo. *Este artigo se trata de uma resenha descritiva a respeito de um experimento desenvolvido na matéria de introdução a teoria da computação sobre o problema do caixeiro viajante.*

1. O Problema do Caixeiro Viajante

O problema do caixeiro viajante consiste em TODO.

Para nossa experiencia, teremos um conjunto de cidades, interligadas umas com as outras, totalmente conectadas na forma de um grafo. Para fazer o trajeto de uma cidade para outra, temos o custo da distância, um valor numérico diferente de uma para outra. Nesse cenário, devemos visitar todas as cidades, sem repeti-las, porém, um dos problemas que já podemos levantar é o quão custoso é um algoritmo para detectar e nos informar o melhor caminho a ser percorrido se levarmos em consideração que queremos testar todas as possibilidades.

Para iniciar a resolução deste problema, definimos o ponto de saída: O algoritmo sempre irá partir de uma cidade aleatória.

Teremos um conjunto de cidades e uma tabela NxN para "mapear" as distâncias da cidade A para B, para C, D, E e assim por diante, sendo que, quanto for da cidade para a mesma cidade, como por exemplo de A para A o custo é 0.

A ideia para este experimento é executar três algoritmos ao menos 30 vezes e identificar a média, o desvio de padrão e assim qual a melhor solução dentre esses três.

2. Descrição dos Algoritmos

2.1. Algoritmo Aleatório

No algoritmo aleatório, determinamos o caminho entre a cidades de forma aleatória como o próprio nome diz. Neste cenário determinamos uma ordem totalmente aleatória de cidades para visitar, e seguimos ela, não considerando distância ou priorizando algo, apenas seguindo a ordem do vetor.

Listing 1. Algoritmo Aleatório.

```
1  Algoritmo Aleat rio
2  copia das cidades = copia(cidades)
3  ordenacao aleatoria = []
4  Enquanto tamanho(copia das cidades) > 0 fa a:
5    numero aleat rio = inteiro aleat rio(entre 0 e tamanho(copia das cidades) - 1
6    cidade removida = copia das cidades[numero aleat rio]
7    del copia das cidades[numero aleat rio]
8    ordenacao aleatoria.inclue(cidade removida)
9  Fim-Enquanto
10 Imprimir ordenacao aleatoria
11 Fim Algoritmo Aleat rio
```

2.2. Algoritmo Guloso

No algoritmo guloso, a partir de uma cidade aleatória, escolhemos a próxima cidade (ou no caso, a melhor opção) a ser visitada levando em consideração a distância mínima, ou seja, a cidade mais próxima. Entretanto, o algoritmo guloso tende a ficar preso em ótimos locais. Um "ótimo local" é uma solução mediana, ou seja, nem ruim, nem boa.

Listing 2. Algoritmo Guloso.

```
Algoritmo Guloso
copia das cidades = copia(cidades)
numero aleat rio = inteiro aleat rio(entre 0 e tamanho(copia das cidades) - 1
cidade inicial = copia das cidades na posicao(numero aleat rio)
deleta copia das cidades na posicao(numero aleat rio)
ordenacao = [cidade inicial]

Enquanto tamanho de copia das cidades > 0 fa a
  saindo da cidade = cidade de cidades na posicao(ordenacao na posicao(tamano de ordenacao - 1))
  proxima cidade = primeira cidade de copia das cidades
  indice da proxima cidade = 0
  menor distancia = distancia entre cidades(saindo da cidade)(proxima cidade)
  indice = 0

  Para possivel proxima cidade Em copia das cidades Fa a:
    distancia = distancia entre cidades(saindo da cidade)(possivel proxima cidade)
    Se distancia < menor distancia Ent o:
      menor distancia = distancia
      proxima cidade = possivel proxima cidade
      indice da proxima cidade = indice
    indice ++
   Fim-Fa a

  cidade removida = copia das cidade na posicao(indice da proxima cidade)
  remove cidada de copia das cidaes na posicao(indice da proxima cidade)
  adiciona na lista de ordenacao cidade removida
Fim-Enquanto
Imprimir ordenacao
Fim Algoritmo Guloso
```

2.3. Algoritmo Semi-Guloso ou Híbrido

O algoritmo híbrido é uma combinação entre o algoritmo aleatório e o algoritmo guloso. Neste cenário, sorteamos um número inteiro entre 0 e 100, e um novo parâmetro delta (podemos chamar este parâmetro como quisermos), no qual ele seleciona o modo de escolha da próxima cidade que são: Guloso ou Aleatório. Assim, determinamos o valor do parâmetro delta seja 70 (um mero exemplo), e a cada momento de decidir qual a próxima cidade, sorteamos um número aleatório de 0 a 100, caso esse número for menor ou igual a delta (70), escolhemos o modo aleatória, caso contrário, escolhemos o modo guloso.

Listing 3. Algoritmo Hibrido.

```
Algoritmo Hibrido
copia das cidades = copia(cidades)
numero aleat rio = inteiro aleat rio(entre 0 e tamanho(copia das cidades) - 1
cidade inicial = copia das cidades na posicao(numero aleat rio)
deleta copia das cidades na posicao(numero aleat rio)
ordenacao = [cidade inicial]
delta = 75

Enquanto tamanho de copia das cidades > 0 fa a:
  valor randomico = numero aleatorio entre 0 e 100

```

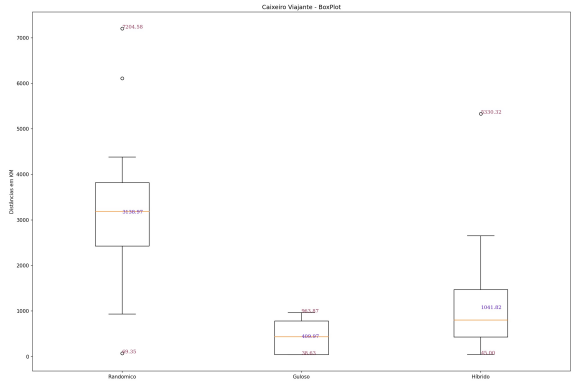
```
    Se valor randomico > delta Entao:
      saindo da cidade = cidade de cidades na posicao(ordenacao na posicao(tamano de ordenacao - 1))
      proxima cidade = primeira cidade de copia das cidades
      indice da proxima cidade = 0
      menor distancia = distancia entre cidades(saindo da cidade)(proxima cidade)
      indice = 0

      Para possivel proxima cidade Em copia das cidades Fa a:
        distancia = distancia entre cidades(saindo da cidade)(possivel proxima cidade)
        Se distancia < menor distancia Ent o:
          menor distancia = distancia
          proxima cidade = possivel proxima cidade
          indice da proxima cidade = indice
        indice ++
      Fim-Fa a

      cidade removida = copia das cidade na posicao(indice da proxima cidade)
      remove cidada de copia das cidaes na posicao(indice da proxima cidade)
      adiciona na lista de ordenacao cidade removida
    Senao:
      numero aleat rio = inteiro aleat rio(entre 0 e tamanho(copia das cidades) - 1
      cidade removida = copia das cidades[numero aleat rio]
      del copia das cidades[numero aleat rio]
      ordenacao.inclue(cidade removida)
    Fim-Senao
  Fim-Enquanto
  Imprimir ordenacao
Fim Algoritmo Hibrido
```

3. Resultados

Executanddo 30 vezes cada um dos algoritmos, obtemos os seguintes resultados a respeito da média aritmética, desvio padrão e diagrama de caixa.

Algoritmo	Média (Distância)	Desvio Padrão (Distância)
Aleatorio	3138.97 KM	1415.68
Guloso	409.97 KM	332.13
Híbrido	1041.82 KM	1042.17

Figura 1. Diagrama de Caixa - Algoritmos



4. Conclusões

Analisando os testes realizados, é possível concluir um item que quase se equivalente aos algoritmos: seu tempo de execução. Podemos relacionar isso com a quantidade de dados

processados, pois não colocamos os algoritmos em testes de estresse, para validar qual algoritmo demoraria mais para encontrar sua solução com grandes quantidades de dados. Por via de regra, o algoritmo aleatório tende a ser o que pode achar uma solução mais rápido devido a sua lógica, se pegar qualquer caminho indiferente da distância, porem isso não analisa o ponto mais relevante deste experimento que se trata de encontrar a melhor solução. Analisando cada algoritmo com os dados levantados, podemos constar que:

Em todas as situações o algoritmo aleatório se mostrou o pior para determinar a rota mais curta. Sua média e desvio padrão ultrapassa a soma dos respectivos valores dos outros dois algoritmos. Em relação ao diagrama de caixa, é o que possui os maiores outliers e também o que possui maior dispersão dos seus pontos, sendo assim ”a maior caixa”. Podemos visualizar que o algoritmo aleatório gera soluções capazes de se igualar ao algoritmo guloso e híbrido porem também acaba por gerar soluções que extrapolam os valores, tornando-se a pior solução.

O algoritmo guloso, surpreendentemente é o que gerou as menores rotas, mesmo levando em consideração o risco dele ter problema nos ”ótimos locais”. Dos três algoritmos foi o que gerou as melhores soluções, tanto analisando os seus valores de média e desvio, quando ao diagrama de caixa, no qual podemos visualizar que grande parte de suas soluções se encontram em valores próximos. Tendo a solução mais longa equivalendo a média das soluções do algoritmo híbrido.

O algoritmo híbrido aparentou ser o mais equilibrado dentre os três quando analisamos o diagrama de caixa. Se analisarmos a sua média e desvio padrão, ele aparenta se aproximar do algoritmo aleatório, o que nos indicaria que não compensa utiliza-lo, porem olhando para o diagrama de caixa, podemos visualizar que o que causou esses valores elevados nas médias foram seus outliers. Se considerarmos que mais de 80 porcento dos casos não entram nesta condição, ele acaba se tornando um algoritmo com soluções bem equilibradas e capaz de contornar o problema de ótimos locais do algoritmo guloso.

Como conclusão, neste experimento, levando em consideração os dados usados, o melhor algoritmo seria o guloso, porem, é necessário levantar a ressalva de que, é possível que acontecem problemas de ótimos locais, o que podem gerar outliers para nós. Podemos especular que, nesta situação o algoritmo guloso se apresentou a melhor solução pela quantidade de dados que o algoritmo precisa consumir, e o numero de tentativas, pois em teoria o algoritmo hibrido que combina os outros dois tendem a ser uma boa solução quando temos uma grande quantidade de dados.