Universidade de Pernambuco – UPE
Campus Garanhuns, PE
Disciplina de Inteligência Artificial
Adjailson Ferreira de Melo

Projeto I - Busca

A proposta de usar o algoritmo A* para encontrar a menor rota da cidade de Garanhuns-PE até Caruaru-PE. De acordo com o site do Google Maps a distância entre essas duas cidades é de 96,80 km. O tempo estimado do percurso da viagem entre as duas cidades é de aproximadamente 1 h 35 min. Já em linha reta entre essas duas cidades a distância é de 88,41 km, como mostra a Figura 1.

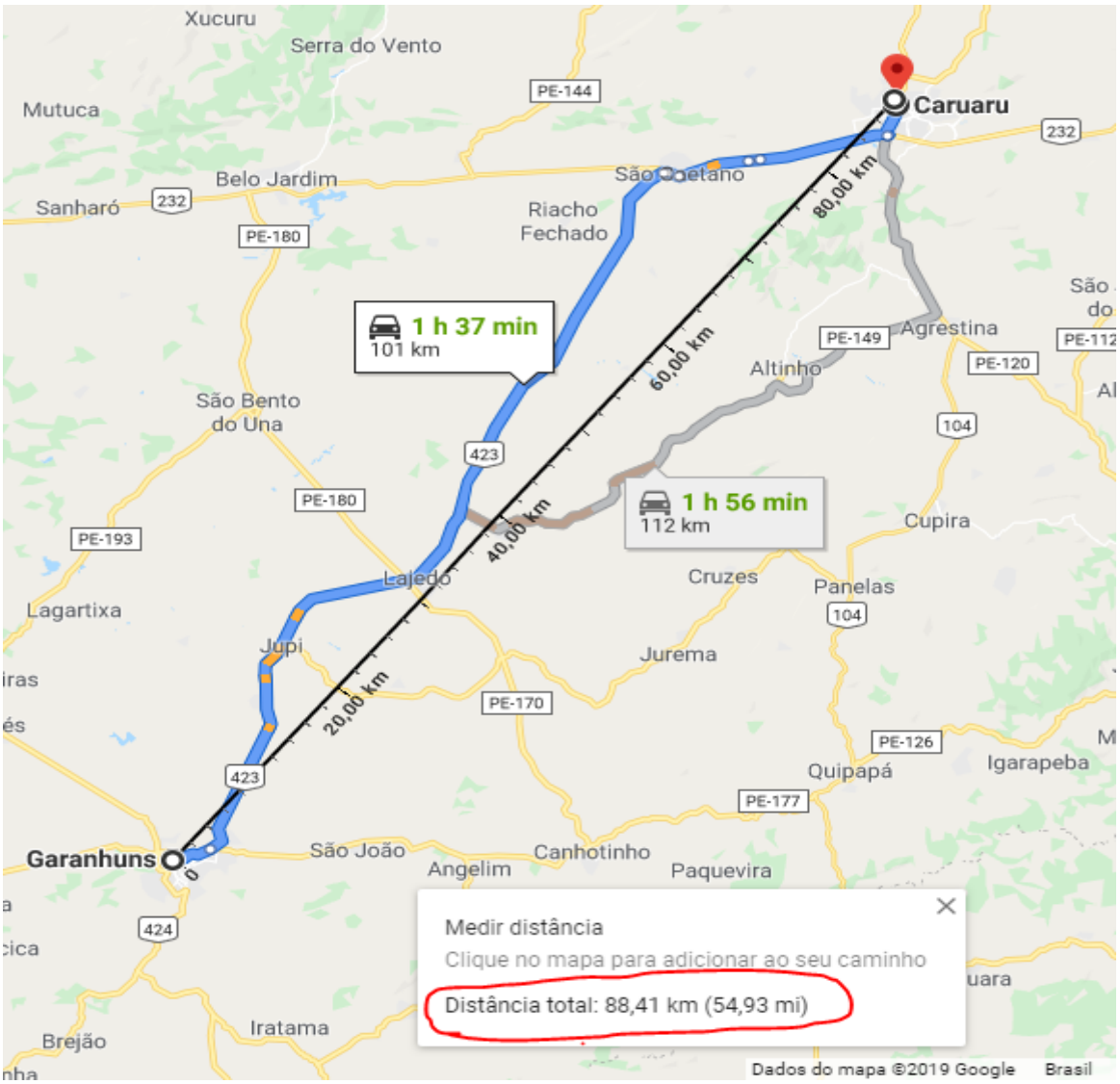

Figura 1: Rota entre Garanhuns e Caruaru de acordo com Google.

Ao usar o algoritmo para encontrar a melhor rota para chegar entre uma cidade e outra, considerando o menor custo em Km e a condição do caminho (Figura 3). Para isso foi usado outras cidades com outras possíveis rotas para chegar ao mesmo destino, Figura 2.

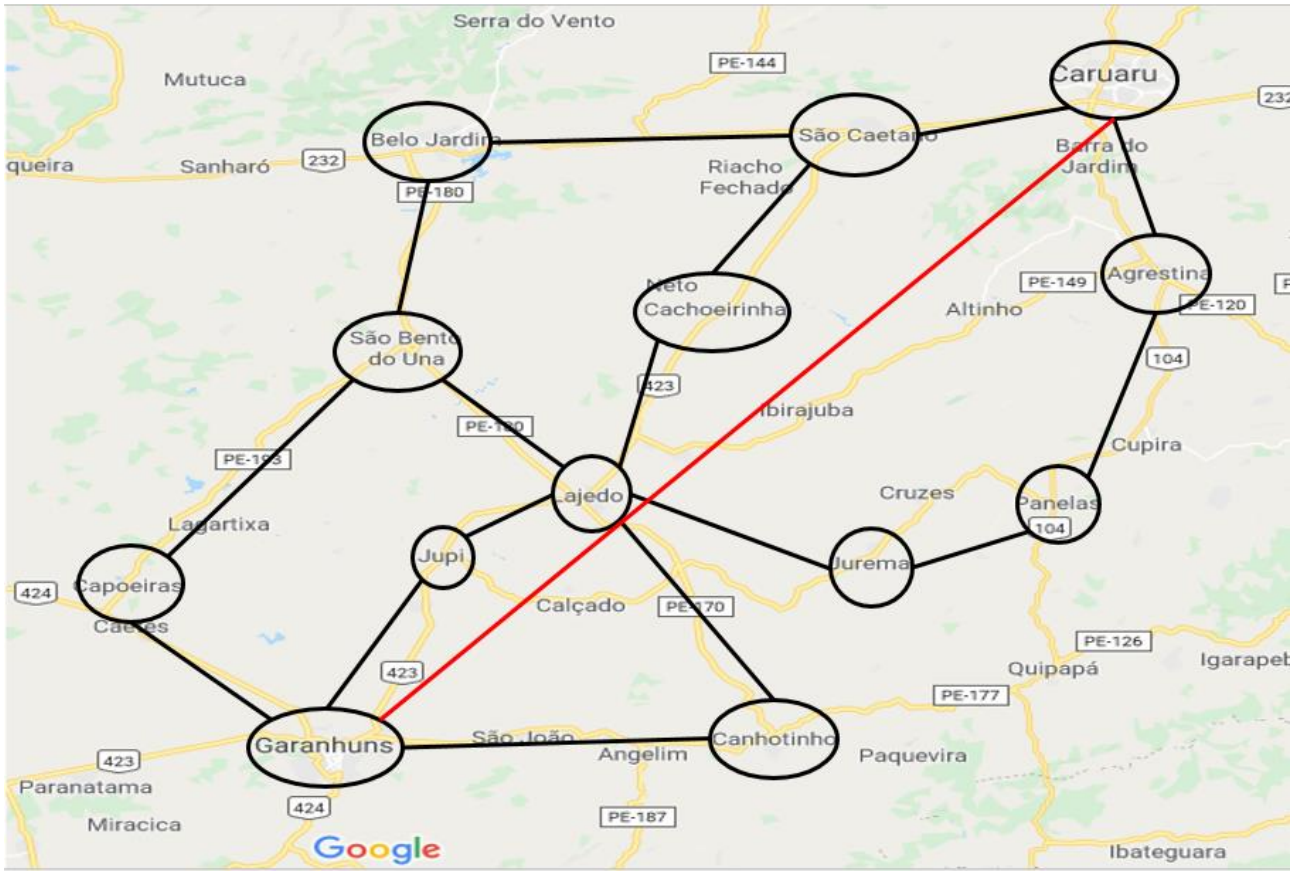

Figura 2: Algumas cidades vizinhas de Garanhuns a Caruaru.

Para encontrar os valores usei o Google Maps para calcular as distâncias real entre uma cidade e outra, também os valores heurísticos que é a distância em linha reta entre cada cidade até ao destino, uma outra forma também seria usando uma escala sobre a imagem capturada e transformando nos valores verdadeiros, mas isso pode não ter tanta precisão.

Na Figura 3, temos todas as informações dos valores reais (cor preta) e heurístico (cor roxo) mais duas observações: a aresta de cor amarela (Capoeiras a São Bento do Una) leva mais tempo devido à grande quantidade de buracos, assim também para a aresta laranja (Lajedo a São Bento do Una) devido às 22 lombadas físicas existentes. O tempo da aresta amarela é de 25 e a laranja 20, esses valores são somados sobre os pesos ou distâncias reais de suas respectivas arestas.

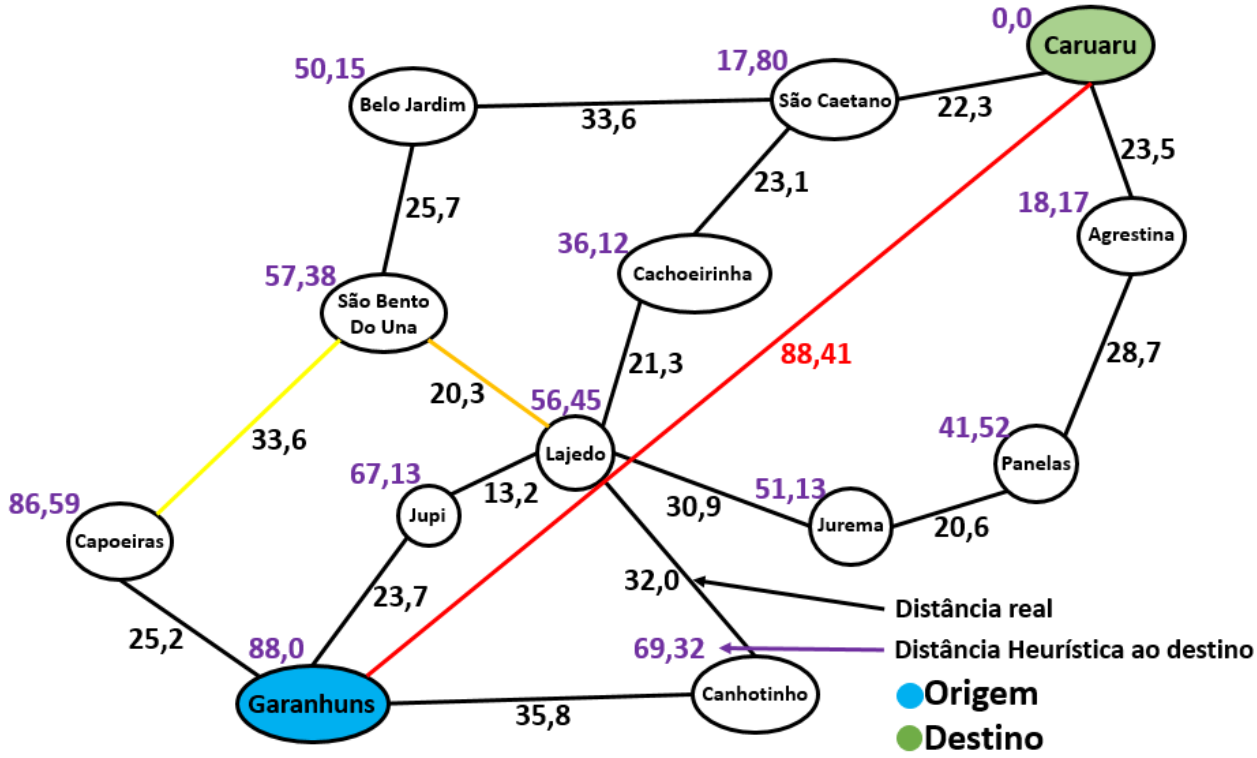

Figura 3: Mapa completo com todas as informações detalhadas.

O algoritmo foi construído na linguagem Python na versão 3.7 usando o conceito de orientação a objetos. Para desenvolver o algoritmo de busca A* foi modelado o problema através de um grafo não direcionado com seus respectivos vértices e arestas.

Três classes foram criadas para representar o grafo, a classe Vertice, a classe Aresta e a classe Grafo, onde contêm todos os métodos do grafo e o algoritmo A*.

Classe Vertice:

```
# coding: utf-8
class Vertice:

    BRANCO = 0 #public
    CINZA = 1 #public
    PRETO = 2 #public

    def __init__(self, nome, heuristica):
        #Declaração variáveis private
        self.nome = nome
        self.heuristica = heuristica #Valor heurístico
        self.custo = 0 #Calcular os somatórios
        self.cor = self.BRANCO #Inicia todos branco
        self.predecessor = None #Apenas para testes fora do projeto

    def getNome(self):
```

```
      return self.nome
    def setNome(self, nome):
      self.nome = nome

    def getHeuristica(self):
      return self.heuristica
    def setHeuristica(self, heuristica):
      self.heuristica = heuristica

    def getCusto(self):
        return self.custo
    def setCusto(self, custo):
        self.custo = custo

    def getCor(self):
        return self.cor
    def setCor(self, cor):
        self.cor = cor

    def getPredecessor(self):
        return self.predecessor
    def setPredecessor(self, predecessor):
        self.predecessor = predecessor
```

Classe Aresta:

```
# coding: utf-8
class Aresta:

    def __init__(self):
        #Declaração variáveis private
        self.origem = None
        self.destino = None
        self.peso = None #Km reais
        self.tempo = None #Tempo percuso

    def getOrigem(self):
        return self.origem
    def setOrigem(self, origem):
        self.origem = origem

    def getDestino(self):
        return self.destino
    def setDestino(self, destino):
        self.destino = destino

    def getPeso(self):
        return self.peso
```

```
    def setPeso(self, peso):
        self.peso = peso

    def getTempo(self):
        return self.tempo
    def setTempo(self, tempo):
        self.tempo = tempo
```

Na classe Grafo, temos uma outra diversidade de métodos (além de outros como busca Gulosa) necessários e incluindo o método de busca A*.

Parte Classe Grafo:

```
class Grafo:

    def __init__(self):
        #Declaração variáveis private
        self.vertices = [] #lista de vertices
        self.arestas = [] #lista de arestas

    # Métodos padrões do grafo
    def addVertice(self, nome, heuristica):
        if not self.contem(nome):
            self.vertices.append(Vertice(nome, heuristica))

    def addAresta(self, origem, destino, peso, tempo):
        vOrigem = self.obterVertice(origem)
        vDestino = self.obterVertice(destino)
        aresta_x = Aresta()
        aresta_x.setOrigem(vOrigem)
        aresta_x.setDestino(vDestino)
        aresta_x.setPeso(peso)
        aresta_x.setTempo(tempo)
        self.arestas.append(aresta_x)

        aresta_y = Aresta()
        aresta_y.setOrigem(vDestino)
        aresta_y.setDestino(vOrigem)
        aresta_y.setPeso(peso)
        aresta_y.setTempo(tempo)
        self.arestas.append(aresta_y)

'''

BUSCA HEURISTICA A*
'''
def buscaAestrela(self, origem, destino):
    try:
```

```
        caminhos = []
        aux = []

        v_inicial = self.obterVertice(origem)
        v_inicial.setCusto(0)
        v_inicial.setCor(Vertice.CINZA)

        aux.append(v_inicial)
        while(len(aux) != 0):
            u = aux.pop(0)
            caminhos.append(u.getNome())#Armazenar esse caminho
            if u == self.obterVertice(destino):#Se é objetivo
                caminhos.append(None)#guardar total custo
                caminhos[-1] = "Total = "+str(u.getCusto())
                return caminhos#break;

            menor = 2147483647 #MAX_VALUE
            menorVertice = None

            for v in self.adjacentes(u.getNome()):

                if v.getCor() != Vertice.PRETO:
                    #          G(n):
                    v.setCusto(u.getCusto() + self.obterAresta(u, v).getPeso())
                    #       G(n) + a condição que é o tempo:
                    total = v.getCusto() + self.obterAresta(u, v).getTempo()
                    #h <= h*:
                    if ((total + v.getHeuristica()) <= menor and v.getCor() ==
Vertice.BRANCO):
                        menor = total + v.getHeuristica() #G(n) + H(n)
                        menorVertice = v
                        menorVertice.setCusto(total)
                    v.setCor(Vertice.CINZA)

            menorVertice.setCor(Vertice.PRETO)
            aux.append(menorVertice)
            u.setCor(Vertice.PRETO)
    except:
        return "Erro!\nVerifique os valores Heurísticos!"
```

Agora temos mais um arquivo Python chamado de Main, que contém toda representação do mapa da Figura 3.

Arquivo de código Main:

```
# coding: utf-8
```

```
from Grafo import *

g = Grafo()
'''
1º paramentro, nome do vertice
2º paramentro, valor heurístico
'''
g.addVertice("Garanhuns",88.41)
g.addVertice("Capoeiras",86.59)
g.addVertice("Jupi",67.13)
g.addVertice("Canhotinho",69.32)
g.addVertice("São Bento do Una",57.38)
g.addVertice("Lajedo",56.45)
g.addVertice("Jurema",51.13)
g.addVertice("Belo Jardim",50.15)
g.addVertice("Cachoeirinha",36.12)
g.addVertice("Panelas",41.52)
g.addVertice("São Caetano",17.80)
g.addVertice("Agrestina",18.17)
g.addVertice("Caruaru",0)
'''
Grafo não direcionado

1º paramentro, vertice origem
2º paramentro, vertice destino
3º paramentro, peso real da aresta
4º paramentro, tempo percuso
'''
g.addAresta("Garanhuns","Capoeiras",25.2, 0)
g.addAresta("Garanhuns","Jupi",23.7, 0)
g.addAresta("Garanhuns","Canhotinho",35.8, 0)

g.addAresta("Capoeiras","São Bento do Una",33.6, 25)#aresta amarela + 25
g.addAresta("Jupi","Lajedo",13.2, 0)
g.addAresta("Canhotinho","Lajedo",32, 0)

g.addAresta("São Bento do Una","Belo Jardim",25.7, 20)
g.addAresta("São Bento do Una","Lajedo",20.3, 0)#aresta laranja + 20
g.addAresta("Lajedo","Cachoeirinha",21.3, 0)
g.addAresta("Lajedo","Jurema",30.9, 0)

g.addAresta("Belo Jardim","São Caetano",33.6, 0)
g.addAresta("Cachoeirinha","São Caetano",23.1, 0)
g.addAresta("Panelas","Jurema",20.6, 0)
g.addAresta("Panelas","Agrestina",28.7, 0)

g.addAresta("São Caetano","Caruaru",22.3, 0)
g.addAresta("Agrestina","Caruaru",23.5, 0)
```

```
print(g.buscaAestrela("Garanhuns", "Caruaru"))
```

Na saída do algoritmo obtivemos o seguinte resultado, Figura 4:

```
>>>
 RESTART: .\Main.py

['Garanhuns', 'Jupi', 'Lajedo', 'Cachoeirinha', 'São Caetano', 'Caruaru',
'Total = 103.60000000000001']
>>>
```

Figura 4: Execução da busca saindo de Garanhuns até Caruaru.

Mas, porque o resultado total é 103,60 Km se na Figura 1 mostra 101 Km? – esse valor maior um pouco é devido aos caminhos reais informados pelo Google Maps entre uma cidade e outra, por ter calculado um ponto de origem do centro da cidade ao outro centro da cidade destino.

Referências

COPPIN, Bem. **Inteligência Artificial**. Edição: 1. Editora: LTC; Edição: 1 (27 de junho de 2017).

NASCIMENTO, Serafim. **Busca Heurística (Parte 1): Busca Gulosa.** (2017). Disponível em: <https://www.youtube.com/watch?v=fJGmyfV9zPs>. Acesso em 09 de setembro de 2019.

NASCIMENTO, Serafim. **Busca Heurística (Parte 2): Busca A* (A estrela).** (2017). Disponível em: <https://www.youtube.com/watch?v=FU6JQaRMMDM>. Acesso em 09 de setembro de 2019.

Anexos

- Código fonte completo do projeto: <https://github.com/Adjailson/AlgoritmoBusca>. Esse código do Git pode estar atualizado em relação ao postado aqui.
- Tabela com os valores heurísticos:

Nome da cidade	Heurística
Garanhuns (origem)	88,41
Capoeiras	86,59
Jupi	67,13
Canhotinho	69,32
São Bento do Una	57,38
Lajedo	56,45
Jurema	51,13

Belo Jardim	50,15
Cachoeirinha	36,12
Panelas	41,52
São Caetano	17,80
Agrestina	18,17
Caruaru (destino)	0