Universidade Federal Fluminense

Inteligência Artificial

Grupo: Paulo Roberto Mann Marques Júnior e Juan Lucas Vieira
Trabalho: Roteamento com Algorítmos de Busca

Introdução

Utilizamos a ferramenta de exportação de dados de mapas do www.openstreetmap.org , onde dada uma área específica (escolhida por nós), temos todas as informações que o Open Street Map pode nos oferecer sobre o local, como **nodes** e **ways** (nós e vias), que é uma abstração para as ruas, onde cada rua é uma lista ordenada de nós. Os dados estão organizados em um arquivo XML. O programa realiza a importação das informações desse XML, inserindo todos os elementos na memória do servidor, o que **pode demorar alguns minutos**, dependendo do tamanho do mapa.

Primeiro importam-se os ways, para saber quais serão os nodes necessários, para que não ocorra a importação de nodes desnecessários. É durante o processo de importação dos ways que criamos o grafo efetivamente.

As informações são organizadas da seguinte forma: todos os nós relevantes do XML são representados pela nossa classe **GeoNode**, e as vias são representadas como uma cadeia de nós que são ligados par-a-par, como vizinhos.

Quando todas as informações estiverem importadas, a classe **Main** terá um HashMap, contendo um par <Long, GeoNode> onde o Long é a chave que identifica o GeoNode, e uma lista de todos os GeoNodes utilizados pelo mapa importado.

Observação 1: Utilizamos uma estrutura de dados chamada KD-Tree (K-Dimensional Tree) para o cálculo do nó mais próximo ao clique do usuário (uma vez que o usuário poderá clicar em localizações onde não existam informações no arquivo XML). Essa estrutura é uma generalização da árvore binária que guarda pontos em um espaço de dimensão *k* e consegue calcular o nó mais próximo de um dado ponto, com custo de tempo linear.

Observação 2: O código foi baseado nas referências [3] e [5].

Instruções de execução

Para executar o código, serão necessárias as seguintes ferramentas: NetBeans (versão 8.0.2 ou maior) e o servidor Glassfish Server 4.1 no NetBeans. Ambos poderão ser obtidos através do link https://netbeans.org/downloads/, tanto a distribuição **"Java EE"** quanto a "**Tudo"**, podem ser utilizadas.

Representação do Estado

Os estados são representados por meio de nós (GeoNodes). Cada nó está relacionado a um ponto do mapa. Neles estão contidos latitude, longitude, sua função de avaliação, a distância entre o nó e a raiz (o custo do caminho até o nó atual) e outras informações extras da rua onde este nó está contido, que serão usadas em uma das heurísticas.

Representação das Ações

As ações de um nó são representadas pela vizinhança desse nó. Essa vizinhança é definida pela verificação de nós próximos ao nó atual que estão contidos em uma via. Ou seja, se temos uma via sendo representada pelo nós n1, n2 e n3, teremos uma cadeia onde n1 é vizinho de n2 e n2 é vizinho de n3.

Função Heurística

São utilizadas duas funções heurísticas:

- Heurísica de Distância em Linha Reta - Essa heurística é definida pela distância em linha reta do nó atual até o nó destino. Ela é calculada usando a fórmula de Haversine presente no método distanceFrom da classe GeoNode.

- Heurísticas de Informações da Via - Essa heurística é baseada nas informações adicionais sobre uma via a que o nó está contido, elas são: velocidade máxima, tipo de estrada e superfície. A falta de informação de uma via foi tratada usando um valor médio para essa informação.

Algoritmos

Os algorítmos implementados foram os seguintes:

- Algorítmos de Buscas Não Informadas
    - Busca em Largura
    - Busca de Custo Uniforme
- Algorítmos de Buscas Informadas
    - Busca A*
    - Busca Gulosa
- Busca Local
    - Hill Climbing Random Restart

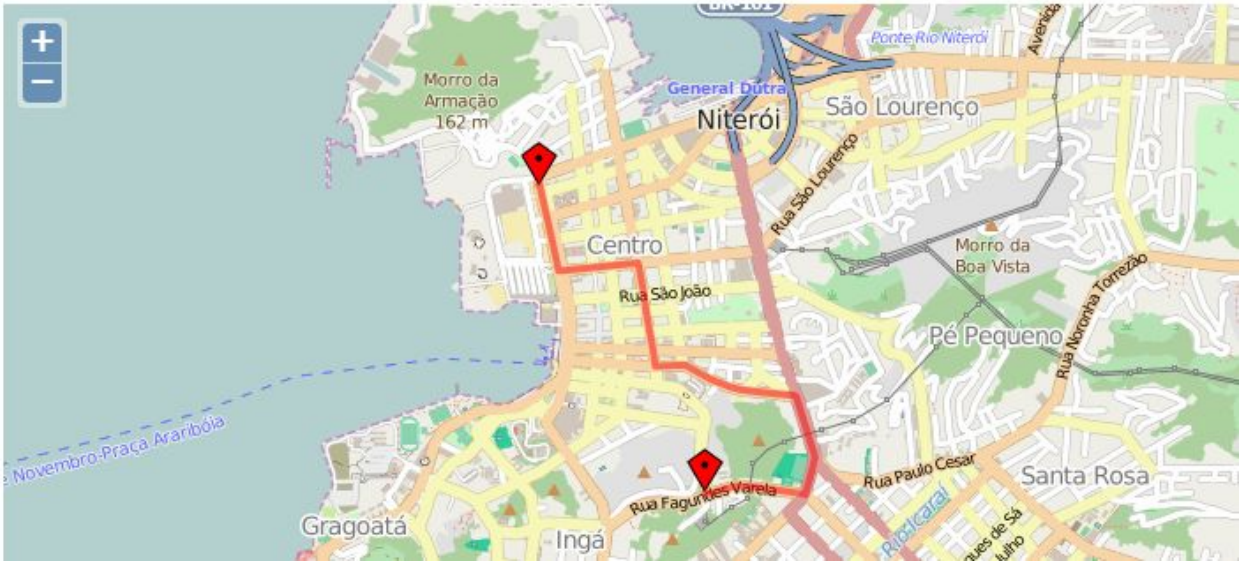
Todos os outros algorítmos são semelhantes aos vistos em sala de aula, o único que ficou um pouco diferente do normal foi o Hill Climbing, por isso será mencionado com mais detalhes.

O Hill Climbing possui as variáveis **path**, um HashMap que associa uma chave GeoNode com um valor GeoNode(uma abstração para reconstruir a solução), e **explored**, uma lista em que mantemos todos os nós que foram explorados. O algorítmo pega um vizinho aleatório, calcula a avaliação (que é um valor baseado na distância em linha reta até o destino) e tenta melhorar o seu desempenho no máximo **kMax** vezes. O Hill Climbing retorna NULL caso não existam vizinhos (beco sem saida) ou todos os vizinhos foram explorados, nesse caso, realizamos um “backtracking” e tentamos explorar um nó alternativo, com avaliação menor, caso não existam alternativas possíveis até o pai do nó raíz (que é NULL), o algorítmo falha. Basicamente existem duas funções, uma fica a par de realizar o backtracking e a outra de realizar o Hill Climbing para cada nó encontrado, que são adicionados ao caminho enquanto o destino não for encontrado ou o nó atual seja NULL (no caso, um erro).

Comparativos

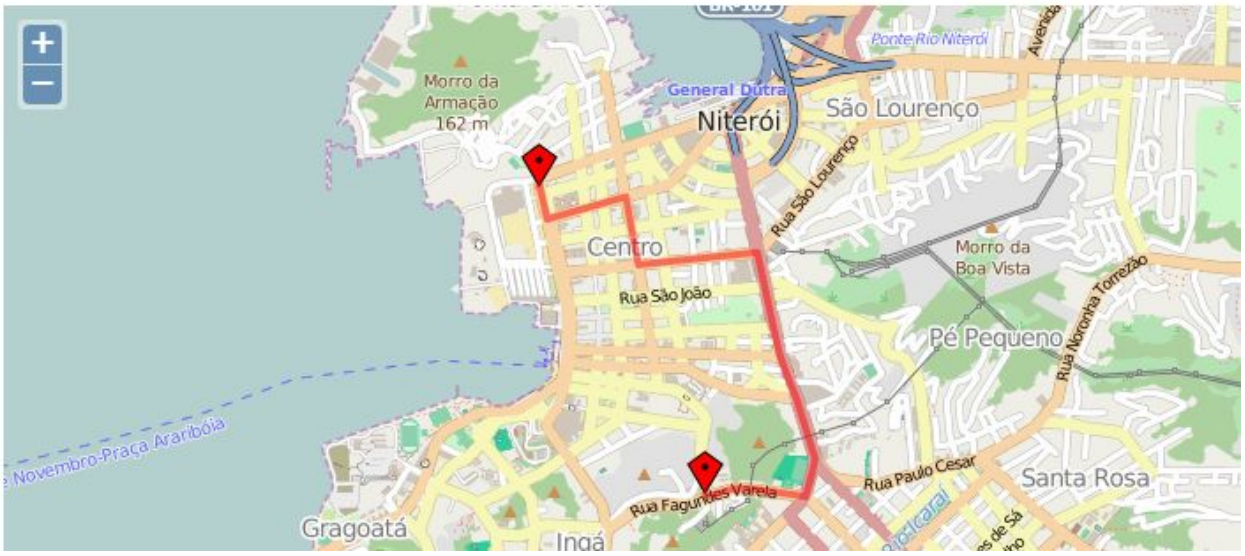
Nas imagens abaixo, pode-se comparar os algorítmos e heurísticas implementados. Podemos perceber que a heurística de informações da via gera uma quantidade de nós bem maior que a de distância em linha reta. É interessante que, apesar de a heurística de informações da via geralmente ser pior que a de distância em linha reta, nos exemplos da busca gulosa, apresentou uma distância menor que a de distância em linha reta, mostrando-se mais eficaz.

Execution time(ms) : 3, Distance(km) : 3.0979737193465833, Generated Nodes : 902, Path Nodes : 44



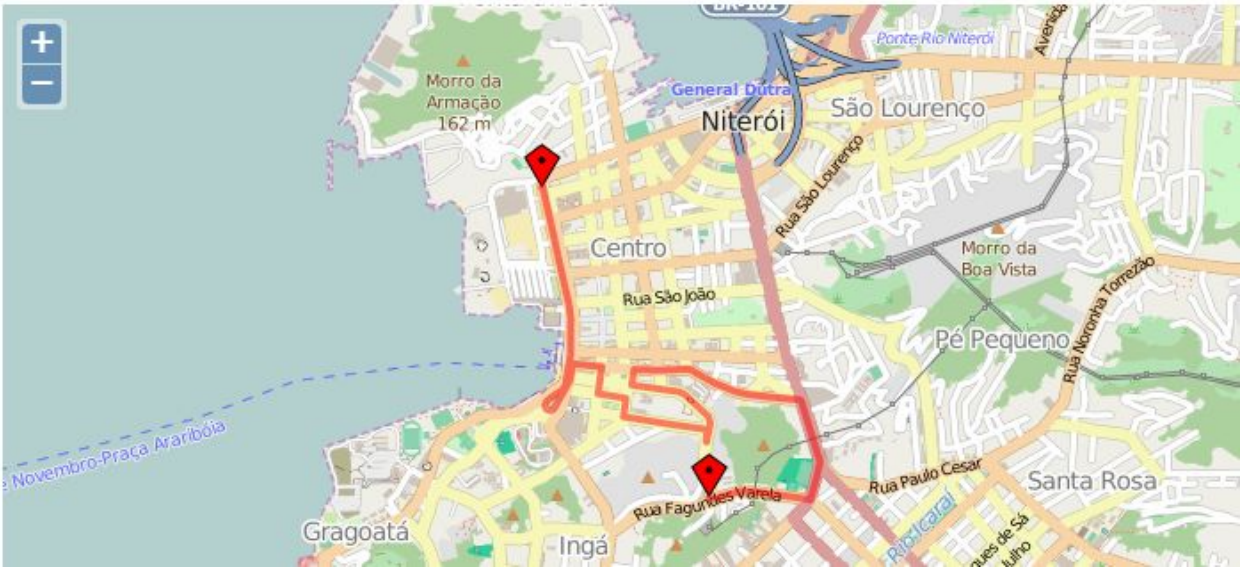
A* Algorithm with Straight-Line Distance Heuristic

Execution time(ms) : 9, Distance(km) : 3.276285056927533, Generated Nodes : 2253, Path Nodes : 37



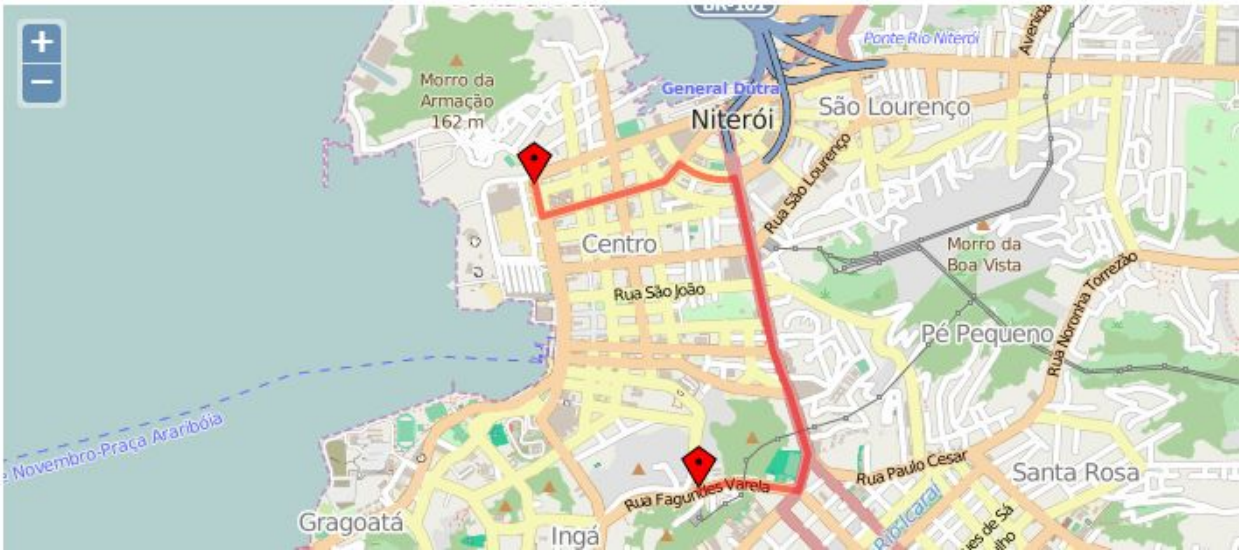
A* Algorithm with Highway Information Heuristic

Execution time(ms) : 0, Distance(km) : 4.93045449940858, Generated Nodes : 167, Path Nodes : 104



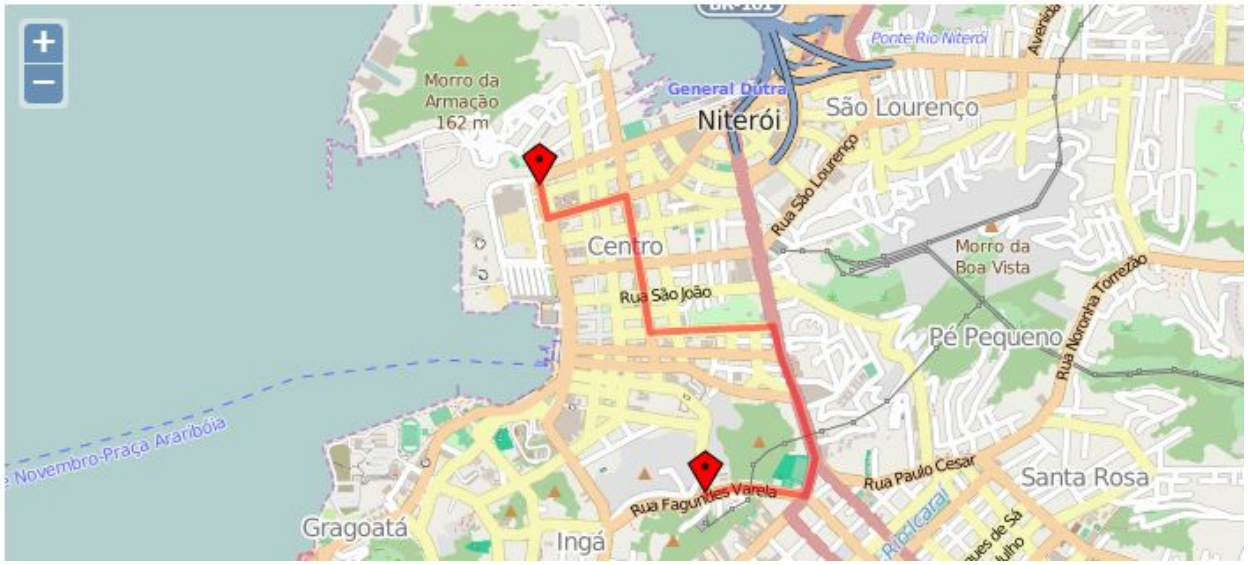
Greedy Search with Straight-Line Distance Heuristic

Execution time(ms) : 10, Distance(km) : 3.349258571516682, Generated Nodes : 1179, Path Nodes : 43



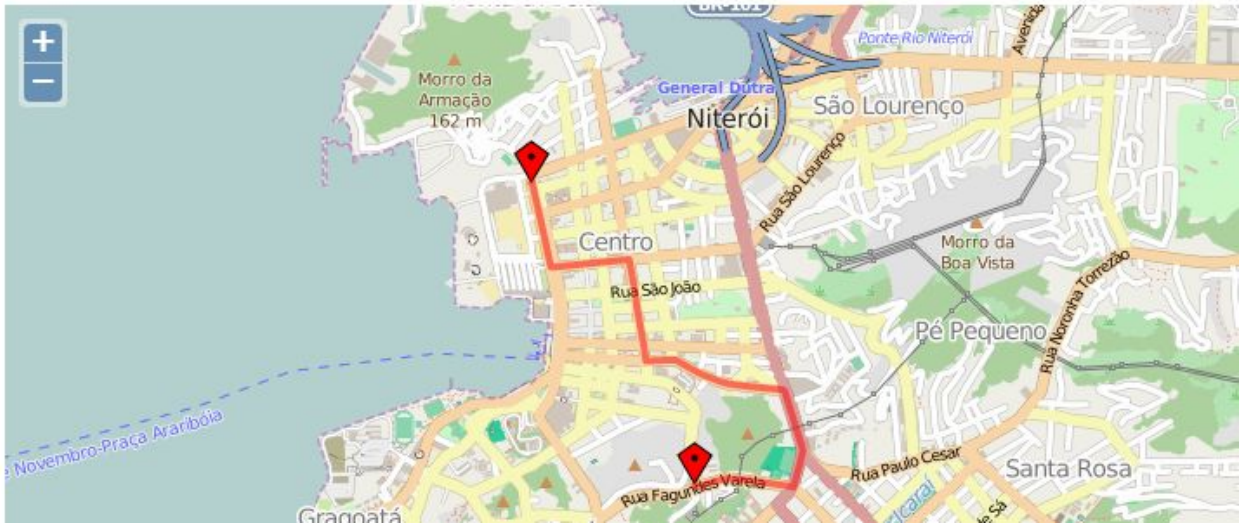
Greedy Search with Highway Information Heuristic

Execution time(ms) : 17, Distance(km) : 3.262035517532703, Generated Nodes : 1764, Path Nodes : 34



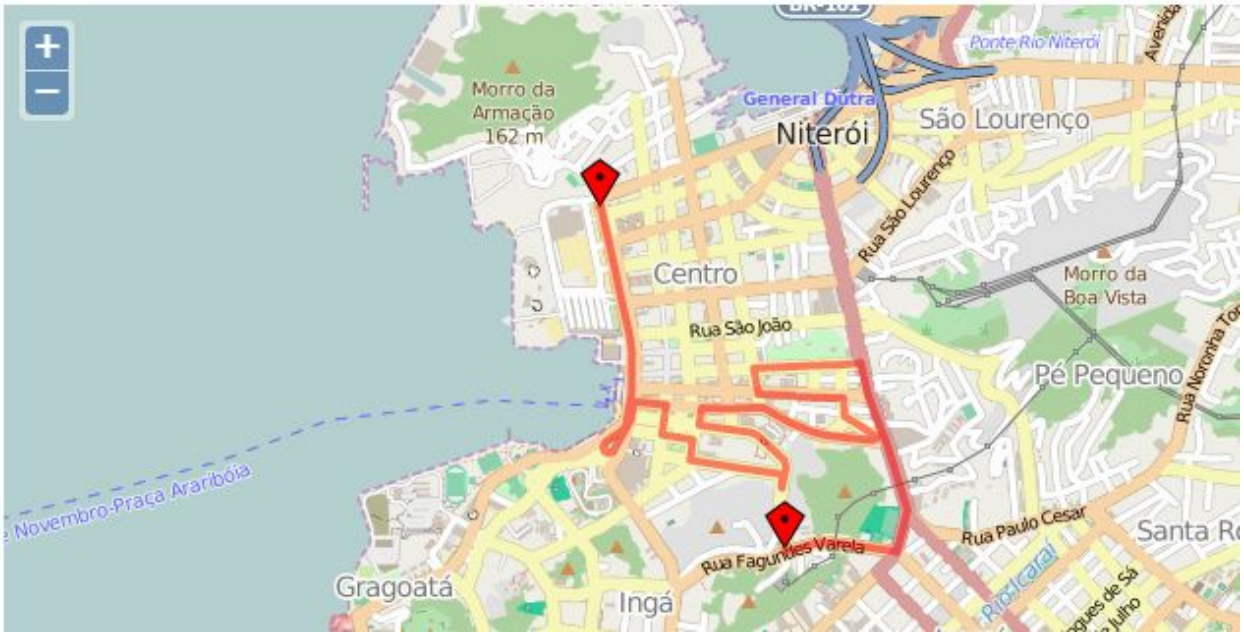
Breadth First Search Algorithm

Execution time(ms) : 50, Distance(km) : 3.0979737193465833, Generated Nodes : 2612, Path Nodes : 44



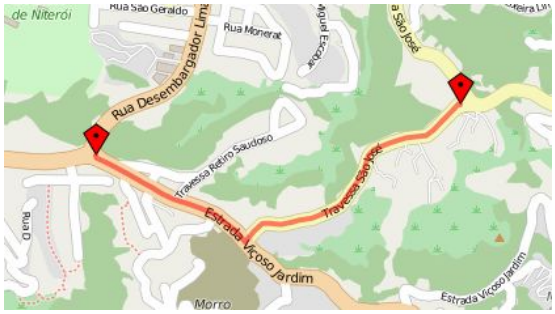
Uniform Cost Search Algorithm

Execution time(ms) : 3, Path Nodes : 126



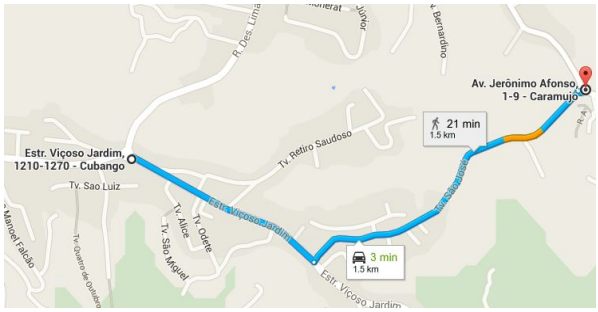
Hill Climbing

A* Algorithm with Straight-Line Distance



Distância: **1.49 km**

Google Maps



Distância: **1.5 km**

Referências

[1]http://wiki.openstreetmap.org/wiki/Overpass_API/Language_Guide#Background_and_concepts
[2]http://en.wikipedia.org/wiki/Haversine_formula
[3]http://ds.cs.ut.ee/Members/hadachi/dss-fall-2014/Heiki-Parn-Report.pdf
[4]http://web.stanford.edu/class/cs106l/handouts/assignment-3-kdtree.pdf
[5]https://bitbucket.org/heiki112/ds_shortestpath_v2