

# EasyPilot: Sistema de navegação

Fabiola Silva, Hugo Drumond e Susana Santos

Conceção e Análise de Algoritmos

Faculdade de Engenharia da Universidade do Porto

Rua Dr Roberto Frias, 4200-65 Porto, Portugal

{fabiola.silva, hugo.drumond, santos.susana}@fe.up.pt

**Abstract**—Neste trabalho prático, pretende-se determinar a eficácia de algoritmos na aplicação do GPS, com o objetivo de serem evitados custos adicionais de deslocamento e perda de tempo. No fundo, que o percurso em consideração seja o mais otimizado possível.

## I. INTRODUÇÃO

Os algoritmos de pesquisa em grafos vão especificar regras para exploração desse mesmo grafo.

O caso de estudo neste trabalho prático é desenvolver um software que nos permita identificar qual o caminho a seguir, qual o critério para escolher o algoritmo utilizado, que dados precisamos guardar e como o fazemos. Tendo em conta a imensidão de rotas disponíveis para ir de um determinado ponto para outro, a tarefa será escolher o melhor percurso, o mais rápido. Este poderá depender de vários fatores, nomeadamente: trânsito, velocidades máximas, probabilidade de acidentes num troço, menor distância entre pontos. Só considerámos o fator distância entre pontos uma vez que as outras variáveis iriam tornar a implementação muita mais complexa. Por exemplo teríamos de fazer uma query ao openstreetmap que devolve-se essa informação, e desenvolver um novo parser. Por esta razão, o algoritmo adotado foi o algoritmo de Dijkstra.

O algoritmo Uniform Cost Search (Branch and Bound) é um caso mais genérico do Dijkstra, em que se define um objetivo e se termina a execução quando já se alcançou o objetivo e os custos na lista de exploração são superiores. Isto é, a diferença é que o Dijkstra encontra o caminho curto do início para todos os nós. Não são utilizadas técnicas gananciosas puras, como é o caso dos algoritmos Steep Ascent e Hill climbing porque estes podem levar a "becos" sem saída. Porque as escolhas que estes algoritmos fazem são definitivas.

Este trabalho tem como objetivo sugerir o caminho mais curto mostrando o resultado num mapa.

## II. ALGORITMOS E COMPLEXIDADES

### A. Algoritmo de Dijkstra

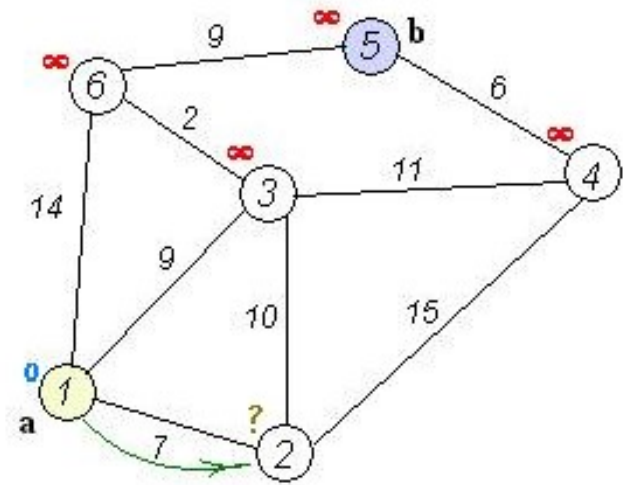


Fig. 1. Exemplo de um grafo percorrido pelo algoritmo Dijkstra

O algoritmo de Dijkstra, utilizado no nosso trabalho prático, soluciona o problema do caminho mais curto num grafo dirigido com arestas de peso não negativo. A nossa implementação apresenta um tempo computacional  $O([E + V] \log V)$ , onde  $E$  é o número de arestas e  $V$  é o número de vértices. Visto termos usado uma binary heap. Poderíamos ter melhorado o algoritmo fazendo a implementação de uma Fibonacci heap, mas tal não foi feito por falta de tempo.

## III. DIJKSTRA'S SHORTEST PATH

No início todos os nós são colocados a infinito, exceto o start node que se inicializa a 0. É usada uma heap onde se coloca o nó inicial. Até a heap ficar vazia explora os nós do vértice selecionado (o que se encontra no topo da heap). Calcula a distância do vértice selecionado para o novo de acordo com a aresta, e se esse custo for inferior ao outro vértice altera o custo e referência a aresta que o atualizou. Caso este não tenha sido visitado é adicionado à heap e altera-se o estado para visitado. É feito o decrease.key de maneira a reordenar a heap.

#### IV. FORMALIZAÇÃO DO PROBLEMA

Foi criada uma interface de linha de comandos que suporta certas operações. É feito o parse e análise desses campos. Posteriormente, faz-se o parse dos ficheiros nodes.txt, roads.txt e subroads.txt de vários mapas. Gera-se os vértices a partir da lista de nós, e as arestas da lista de subroads. É corrido o algoritmo de dijkstra. O resultado é devolvido em formato Overpass QL de maneira a poder-se visualizar a solução num mapa real. Os vértices guardam a informação dos nós, e os edges a das subroads. O caminho do objetivo até o começo é percorrido pela aresta que levou ao menor custo do nó.

[illegible]

Fig. 3. Formato da query devolvida pelo script executado

União de um caminho limitado por dois nós.

```
(node(410697669);node(1131134002);way(106811317);)out;
```

Fig. 4. Formato da query com dois nós

## V. DIAGRAMA DE CLASSES (UML)

```

template <typename T>
void Graph<T>::dijkstra(const T& in) {
    for (auto& vertex_ptr : this->vertexSet) {
        vertex_ptr->cost_edge = NULL;
        vertex_ptr->cost = std::numeric_limits<double>::max();
        vertex_ptr->visited = false;
    }

    Vertex<T>* selected_vertex = this->getVertex(in);
    if (selected_vertex == nullptr) {
        return;
    }
    selected_vertex->cost = 0;

    vector<Vertex<T>*> priority_queue;
    priority_queue.push_back(selected_vertex);

    make_heap(priority_queue.begin(), priority_queue.end());

    while (!priority_queue.empty()) {
        selected_vertex = priority_queue.front();
        pop_heap(priority_queue.begin(), priority_queue.end());
        priority_queue.pop_back();

        for (auto& edge : selected_vertex->adjacency_list) {
            Vertex<T>* target = edge.target;

            double new_target_cost = selected_vertex->cost + edge.weight;
            if (new_target_cost < target->cost) {
                target->cost = new_target_cost;
                target->cost_edge = &edge;

                if (!target->visited) {
                    target->visited = true;
                    priority_queue.push_back(target);
                }

                make_heap(priority_queue.begin(), priority_queue.end(),
                           VertexCostGreater Than<T>());
            }
        }
    }
}

```

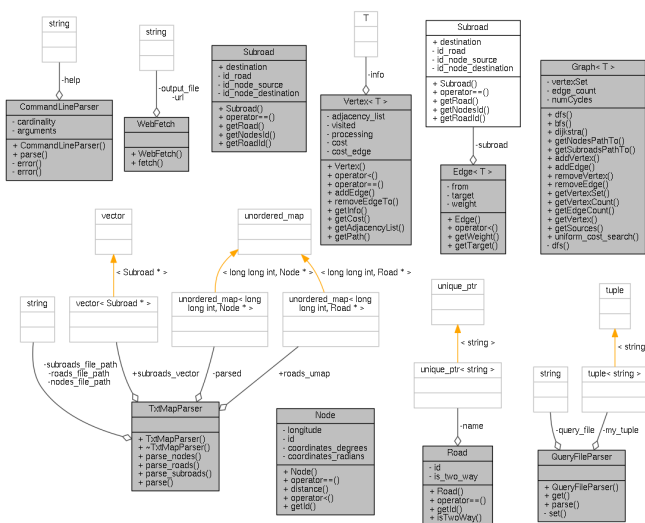


Fig. 5. Diagrama de classes

## VI. ROTAS DEVOLVIDAS PELO ALGORITMO

### A. Mapa do Porto

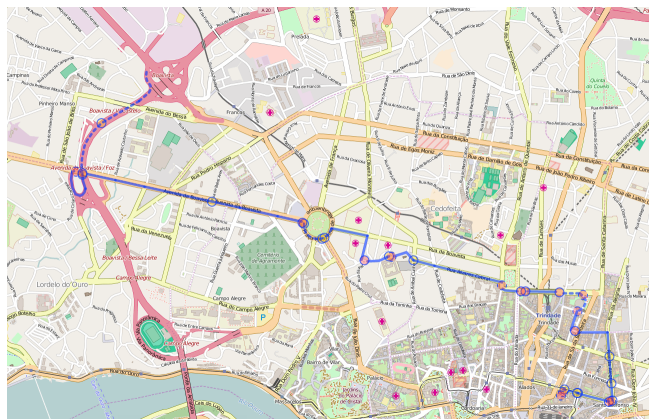


Fig. 6. Percurso mais curto selecionado pelo algoritmo

Fig. 2. Algoritmo Dijkstra

### A. Formação dos dados

Os dados de saída são imprimidos na consola e armazenados em ficheiro. Estes estão em formato Overpass QL de modo a poderem ser introduzidos num caixa de texto, em <https://overpass-turbo.eu/>. Possibilitando a visualização do percurso mais curto no mapa.

## B. Mapa de Lisboa

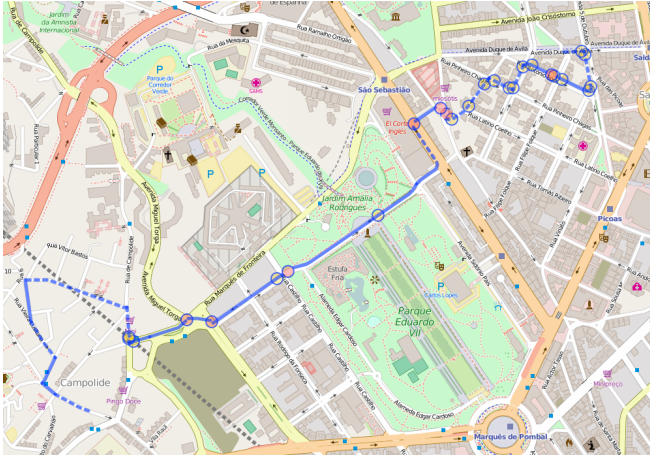


Fig. 7. Percurso mais curto selecionado pelo algoritmo

## C. Mapa de Lisboa

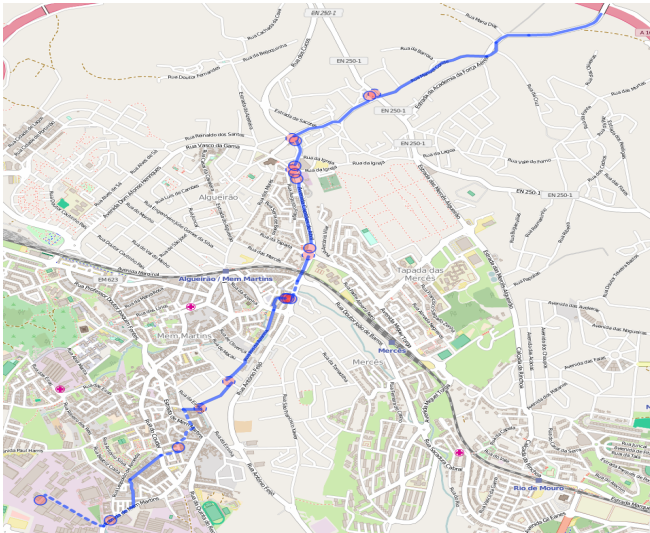


Fig. 8. Percurso mais curto selecionado pelo algoritmo

## D. Mapa de São Paulo (Brasil)

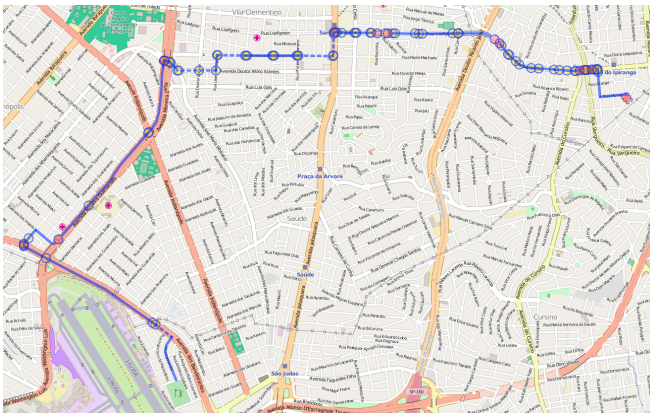


Fig. 9. Percurso mais curto selecionado pelo algoritmo

## VII. INSTRUÇÕES

### A. Procedimento

- Instalar ubuntu 16.04
- Executar o script de inicialização `./scripts/initUbuntu.sh`
- Chamar cada um dos exemplos em `./code/run/*.sh` (O script faz make). Por exemplo `./code/run/porto.sh`
- Dar três cliques de modo a copiar a query Overpass QL, resultado da execução de um exemplo
  - Ir a <https://overpass-turbo.eu/>
  - Fazer paste da query para a caixa de texto
  - Clicar em Run no menu
  - Clicar na lupa de magnificação (zoom to data) no menu do mapa
  - Comparar com a imagem respetiva imagem em `./code/run/*.png`

### B. Sistema Operativo

Ubuntu 16.04 `./scripts/initUbuntu.sh` - script de instalação de dependências e ferramentas

### C. Estrutura do Sistema

- `./scripts/` - Scripts úteis para configurar a máquina para o desenvolvimento
- `./code/`
  - **src** - código fonte
  - **makefile** - ficheiro de compilação, targets: debug, all -i default
  - **external** - ferramentas externas ao nosso projeto
  - **run** - scripts de casos de uso da nossa aplicação e respetivos resultados
  - **data** - mapas (nodes, roads, subroads) para -mtf e ficheiros json para -qf
  - **bin** - onde são colocados os ficheiros binários
  - **buildtemp** - ficheiros temporários de compilação
  - **Doxyfile** - Config de geração de ficheiros de documentação
  - **Doxygen** - Pasta com a geração da documentação doxygen
  - **CAL.workspace** - Ficheiro de projeto codelite

### D. Dependências

- Bibliotecas
  - Boost
  - Curl
  - Curlpp
- Compilador C++14
- Command Line Options `((-queryfile|-qf) queryfile [outputfile] ) | ( (--maptxtfiles|-mtf) innodesfile inroadsfile inwaysfile instartnodeid ingoalnodeid outshortestfile)`
  - qf** faz um pedido get a um servidor de acordo com uma localização e query. query file em json.
  - mtf** a partir de um conjunto de ficheiros de um mapa (nodes, roads, ways) encontra o caminho mais curto

entre um nó e um outro(start, goal). Imprime o resultado e escreve para um ficheiro( `shortest_file`) ambos em formato query Overpass QL de modo a visualizar a solução.

## VIII. CONCLUSÃO

Com a ajuda do algoritmo de Dijkstra, conseguiu-se obter uma solução inteligente ao nosso caso de estudo. Desta forma, a partir do ponto inicial até ao ponto de chegada, foi gerado um percurso ideal. Este foi um trabalho bastante exigente, pois teve que haver uma pesquisa exaustiva e só depois começar a pôr em prática tudo o que foi lecionado nas aulas teóricas. Contudo, foi um trabalho desafiante que permitiu a pesquisa de assuntos desconhecidos e foi, portanto, enriquecedor. Uma vez que a aplicação ficou com boas bases poderíamos implementar algoritmos informados, por exemplo, o A\*. Este resulta da combinação do algoritmo greedy(h) e uniform cost search(g), ou seja, estimativas de custos posteriores e custos acumulados.  $F(n)=h(n)+g(n)$ . Em que se define uma heurística que no caso da pesquisa de grafo tem de ser consistente e em árvore admissível de maneira a poder encontrar a solução ótima.

## REFERENCES

- [1] "How to Query", OpenAlfa Blog. <http://blog-en.openalfa.com/how-to-query-openstreetmap-using-the-overpass-api>.
- [2] "Overpass API/Language Guide", Wikipedia. [http://wiki.openstreetmap.org/wiki/Overpass\\_API/Language\\_Guide](http://wiki.openstreetmap.org/wiki/Overpass_API/Language_Guide).
- [3] "Overpass API/Overpass QL", Wikipedia. [http://wiki.openstreetmap.org/wiki/Overpass\\_API/Overpass\\_QL](http://wiki.openstreetmap.org/wiki/Overpass_API/Overpass_QL).
- [4] "How to convert OSM file into a Graph Representation", OpenStreetMap Help <https://help.openstreetmap.org/questions/19213/how-can-i-convert-an-osm-xml-file-into-a-graph-representation>.
- [5] "Graph Model", Mapbox. <https://www.mapbox.com/blog/smart-directions-with-osrm-graph-model/>.
- [6] "Elements", Wikipedia. <http://wiki.openstreetmap.org/wiki/Elements>.
- [7] "Algoritmos em Grafos: Caminho mais curto", Moodle da FEUP. [https://moodle.up.pt/pluginfile.php/112223/mod\\_label/intro/06.grafos2\\_a.pdf](https://moodle.up.pt/pluginfile.php/112223/mod_label/intro/06.grafos2_a.pdf).
- [8] "Algoritmo de Dijkstra", Wikipedia, ultima atualização 18h30 de 25 de julho de 2014. [https://pt.wikipedia.org/wiki/Algoritmo\\_de\\_Dijkstra](https://pt.wikipedia.org/wiki/Algoritmo_de_Dijkstra).
- [9] "Shortest Path Algorithm". <http://www.csc.villanova.edu/~helwig/grafpage.html>
- [10] "Dijkstra's Algorithm", YouTube. <https://www.youtube.com/watch?v=gdmfOwyQlcI>
- [11] "Dijkstra's Algorithm, example", YouTube. <https://www.youtube.com/watch?v=5GT5hYzjNoo>