SpeedMail: Correio Prioritário

(tema 9)

Concepção e Análise de Algoritmos

Grupo 4 Turma 7
Luís Fernandes - up201706910@fe.up.pt
Nelson Gregório - up200900303@fe.up.pt
Paulo Moutinho - up201704710@fe.up.pt

18 de Abril de 2019

Descrição do tema

Uma empresa pretende estabelecer um sistema inteligente, onde encomendas são recolhidas num ponto de origem indicado pelo cliente e transportadas, por veículos da empresa, até aos seus respectivos pontos de destino.

Sendo o número de veículos limitado é preciso encontrar uma rota que minimize o tempo necessário para transportar as encomendas, tendo em conta possíveis obstáculos.

Identificação do problema

Dada a localização do centro de apoio, o número e respetivas capacidades de veículos disponíveis, a dispersão dos pontos e precedência de recolha face à entrega, é preciso configurar rota(s) de forma a minimizar o tempo necessário para satisfazer todas as encomendas. Existe também uma restrição que obriga a fazer a recolha de uma encomenda antes da sua entrega.

Vamos então modelar a rede de estradas através de um grafo, onde as arestas serão as ruas e os vértices as junções entre elas. As arestas podem ser unidirecionais ou bidirecionais de forma a representar o sentido do trânsito das ruas. Os pontos de recolha, de entrega e os centros de apoios são também representados por vértices no grafo.

Minimizar o custo da rota é redutível ao problema do **Caixeiro Viajante (TSP),** sendo portanto **NP-difícil**, mas não **NP**. Isto é, é tão difícil quanto qualquer outro problema NP, no entanto não é possível encontrar a solução óptima (rota mais curta neste caso) em tempo polinomial.

Embora seja fácil calcular todas as rotas possíveis e encontrar a mais curta, a natureza combinatória do problema faz com que ao aumentar o número de vértices, a quantidade de rotas a verificar cresça com o fatorial desse número (vértices). É importante notar que certas restrições específicas ao nosso problema, diminuem o número máximo de rotas possíveis a verificar.

Soluções sub-óptimas são calculáveis através de algoritmos gananciosos com complexidade temporal aceitável face ao número de vértices.

Formalização do problema

Dados de entrada

- ➔ V - conjunto de vértices (cruzamentos de ruas e pontos de interesse).
  Cada ponto terá os seguintes parâmetros associados:
  - ◆ Nome (termo identificativo)
  - ◆ Coordenadas (localização geográfica)
- ➔ E - conjuntos de arestas (ruas)
  - ◆ Peso W - cada estrada terá associado um peso (depende do comprimento da estrada, grau de acessibilidade, etc)
- P(Vi, Vf) - tuplo que define os vértices de origem e destino de uma encomenda.
- ➔ G(V,E) - grafo dirigido pesado, em que os vértices representam os vários pontos no mapa e as arestas representam as estradas (unidirecionais ou bidirecionais) que ligam os vários vértices.
- ➔ N - Conjunto de veículos representados pela sua capacidade
- ➔ C - Vértice que define o centro de apoio

Dados de saída

- R - Conjuntos de vértices (ordenados) que representa a(s) melhor(es) rota(s) entre todos os pontos de recolha e entrega de encomendas
  - T - Soma dos pesos de todas as arestas de uma rota em **R**

Restrições

- $\forall e \in E : W(e) > 0$
- $|N| > 0 \land \forall n \in N : n > 0$
- $T > 0$, pois é uma soma entre pesos (W) também positivos
- Uma encomenda só pode ser entregue se já foi recolhida
- O primeiro e último elementos de R são o centro de apoio
- Só é possível recolher uma encomenda se houver espaço para ela no veículo.

Função objetivo

A solução deste problema é obtida minimizando o peso total T das arestas percorridas em cada rota de R satisfazendo todas as encomendas pendentes. Assim sendo, função objetivo será a seguinte:

$$h = Min(\sum_{i=1}^{|R|} T(r)),\ r \in R$$

Perspectiva de solução

Divisão do problema em subproblemas:

1. Um veículo com espaço ilimitado.
2. Um veículo com espaço limitado.
3. Um veículo com espaço limitado e um centro de apoio
4. Múltiplos veículos com espaço limitado e um centro de apoio

Pré-processamento

Calcular distâncias de todos os vértices para todos os outros vértices usando o algoritmo **Floyd-warshall**

Procedimento a cada fase

Subproblema 1)

1. Determinar rota com menor custo (força bruta VS algoritmo ganancioso)
2. Minimizar custo (tempo de viagem/distância percorrida) da rota

Subproblema 2)

1. Determinar rotas (semelhante ao subproblema 1) tendo em atenção a capacidade máxima do veículo
2. Minimizar custo

Subproblema 3)

1. Determinar rotas (semelhante ao subproblema 2) tirando partido do centro de apoio.
2. Minimizar custo

Subproblema 4)

1. Determinar configuração de rotas, utilizando múltiplos veículos, minimizando o custo total destas
2. Minimizar custo (tempo de viagem/distância percorrida) de cada rota

Casos de utilização

O software que propomos criar vai, potencialmente, criar um grafo a partir de pontos (guardados num ficheiro em disco).

Vai calcular as distâncias entre todos esses pontos (este cálculo pode ser feito apenas uma vez, com o seu resultado também guardado em disco).

Irá apresentar um menu com opções para:

- ➔ introduzir/alterar/remover encomendas
- ➔ definir a localização do centro de apoio
- ➔ definir o tamanho e capacidades da frota de veículos disponíveis
- ➔ calcular rotas baseadas em diferentes critérios.

A visualização do grafo será feita recorrendo a material de apoio fornecido, inicialmente ao GraphViewer, e posteriormente ao OpenStreetMap

Considerações e situações a averiguar

Com o intuito de avaliar qualitativamente as rotas calculadas, tencionamos colecionar métricas (obtidas através de geração aleatória de encomendas) do custo, sob a forma de tempo. Para isso, pensamos em definir uma velocidade média para o veículo que junto com a distância entre pontos permite calcular esse tempo.

Será também contabilizado um tempo constante para simular o ato de recolher/entregar uma encomenda. É assumido que a configuração de encomendas no veículos é feito de acordo com a sua ordem de entrega, daí ser constante.

Tencionamos também estudar a utilidade de um algoritmo (pesquisa em profundidade/largura) para avaliar a acessibilidade a um determinado ponto, contra tentar calcular imediatamente o caminho para ele, uma vez que o grafo representa uma zona urbana, com nós bastante interligados (possivelmente poucos casos de reduzida acessibilidade).

Outra ideia é, mediante uma avaliação de acessibilidade ao centro de apoio, um valor é atribuído a cada ponto e é usado como peso para definir prioridade de visita.

Mais uma vez, com um centro de apoio localizado no centro urbano, é esperado que a acessibilidade diminua radialmente a partir desse centro.

Na fase de múltiplos veículos, avaliar métodos que permitam atos de recolha e entrega na mesma rota contra rotas dedicadas a recolha ou entrega. A noção pré-implementação é que a variação de tempo do processo todo (colocar pedido até entregar encomenda) será menor com rotas dedicadas, algo que pode ser desejável pois mostra consistência do serviço. No processo de escolha de encomendas a serem entregues, é importante controlar o número de vezes que uma encomenda é adiada, de forma a manter “constante” o tempo de entrega entre todas.

Estruturas de Dados utilizadas

Para o grafo é utilizado o código das aulas práticas, (composto por três classes: Graph, Edge e Vertex) definido em graph.h. Classe Vertex foi alterada para manter um registo das arestas a entrar e as a sair (outgoing e incoming). Classe Edge foi alterada para manter um ID da aresta a usar no graphviewer e foram adicionados getters para os nós. Foi adicionado código também para facilitar a interação entre as estruturas do grafo e a sua representação gráfica (graphviewer).

Para o conteúdo dos nós, utilizamos a struct Node, com o ID do nó e as respectivas coordenadas.

As encomendas são representadas por Package que guarda um ID, os nós de recolha e entrega, o ID da aresta que os liga e a distância final da rota.

As rotas são representadas por Route, que guarda um ID, vetores de nós e arestas do ponto inicial até ao final e a distância total da rota que é a soma dos pesos de todas as arestas.

Conectividade do Grafo

Em relação à conectividade do gráfico, aplicou se o algoritmo BFS (breadth first search) falado nas aula teóricas, de modo a encontrar o nó com o maior número de ligações, sendo que a partir desse, aproximadamente 20% do grafo se encontra acessível.
Após várias tentativas de processamento, verificou-se também quais dos nós eram sem saída (sinks), o que dificultou a criação de encomendas aleatórias.
Deste modo, para efeitos de análise mais eficiente, decidiu-se transformar todas as arestas em bidirecionais, tornando efetivamente o grafo num não dirigido.
Foram também identificados e eliminados os nós sem arestas ligadas.

Casos Implementados

O único caso implementado é a situação de um veículo com um número à escolha de encomendas que calcula uma rota com início e fim no Centro de Apoio, passando por todos os pontos de recolha e os respetivos pontos de entrega.

Apresentação e análise dos algoritmos

- Vi - Conjunto de pontos de recolha
- Vf - Conjunto de pontos de entrega
- Res - Conjunto ordenado da rota
- Packs - Conjunto de pacotes (P)

findSubOptimalDeliveryRoute:

```
L← Vi
s ← CENTRO_APOIO
While L ≠ Ø do
    bDist ← INF
    DIJKSTRA(G, s)                          // O[(|V|+|E|)*log(|V|)]
    for each p ∈ L do                       // O(N)
        if dist(p) < bDist do
                bDist ← dist(p)
                s ← p
        if bDist = INF do
                break
        if s ≠ CENTRO_APOIO do              // O(N)
            if s ∈ Vi do
                REPLACE(s, dest(s)) in L
            if s ∈ Vf do
                REMOVE(s) in L
        INSERT(s, Res)
```

```
DIJKSTRA(G,s):
for each v  V do                                        //O(|V|)
    dist(v)   ← INF
    path(v) ←  nil
dist(s) ← 0
Q ← Ø
INSERT(Q, (s, 0))                                       // O(1+1)
    while Q ≠ Ø   do
    v  ← EXTRACT_MIN(Q)                                         // O(LOG(V))
    for each w ∈ Adj(v) do                              // O(E)
        if dist(w) > dist(v) + weight(v,w) then
            dist(w) ←  dist(v) + weight(v,w)
            path(w) ←  v
            if w ∉  Q then // old dist(w) was INF
                    INSERT(Q, (w, dist(w)))             //  O(1)
            else
                    DECREASE-KEY(Q, (w, dist(w)))               //O(LOG(V))
```

$O(1) + O(|V|)*O(\log|V|) + O(|E|)*[O(1)+O(\log|V|)]$

$O(|V|)*O(\log|V|) + O(|E|)*O(\log|V|)$

$O(\log|V|) * [O(|E|)+O(|V|)]$

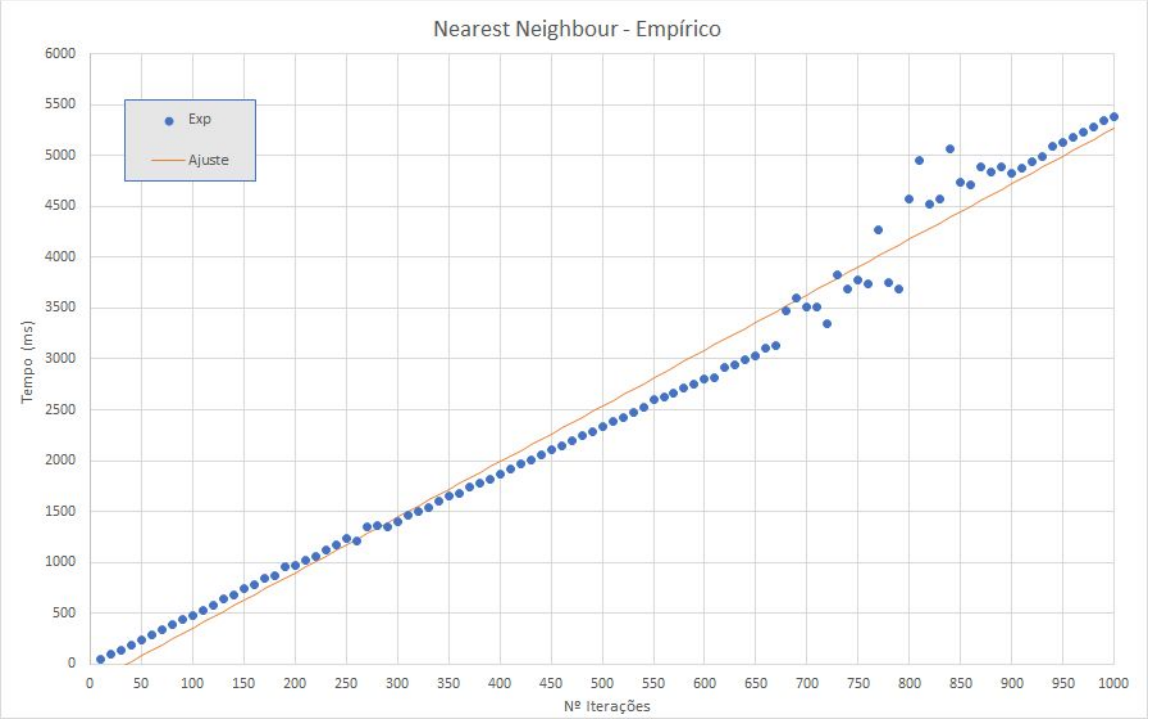
```
generateRandomPackages(G, k):
Shuffle(V)
i ← 0
while |Packs| < k do
    if i >= |V| do break
        DIJKSTRA(G, CENTRO_APOIO)
        i++
        O ← V[i]
        if O ≠ CENTRO_APOIO do
            if path(O) ≠ Ø do
                DIJKSTRA(G, O)
                    while i < |V| do
                        i++
                        D ← V[i]
                        if D ≠ CENTRO_APOIO do
                            if if path(D) ≠ Ø do
                                INSERT(Packs, {O, D})
                                break
```

Uso de Dijkstra é propositadamente repetido na fase de desenhar a rota, em vez de aproveitar o cálculo feito em **findSubOptimalDeliveryRoute**.
Desta forma o código para calcular a rota é somente usado na rota, e o tempo que o algoritmo demora é representativo disso.

Análise de complexidade dos algoritmos

Para calcular uma rota que passa por todos os pontos de recolha e entrega, utilizou-se um algoritmo de aproximação denominado por “Nearest Neighbour”, o qual é ganancioso, pois para cada nó da rota, escolhe o nó mais próximo desde que este não seja um ponto de entrega sem se ter feito previamente a sua recolha.

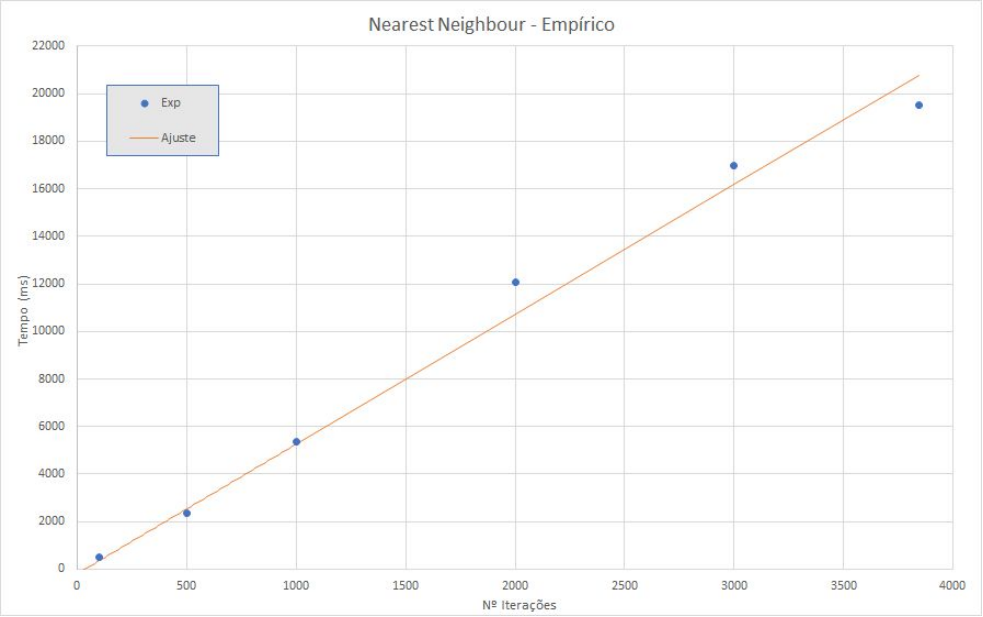
O próximo gráfico mostra a linearidade do algoritmo em questão.



Como se pode observar, o tempo em microsegundos cresce de maneira linear em relação ao número de iterações. A zona com maior irregularidades é causada pela utilização do computador em outras tarefas executadas em simultâneo que podem ter afetado o tempo de execução do algoritmo.

Neste gráfico, a gama do número de iterações é limitada pelo número máximo de pacotes possíveis no mapa do Porto começando no nó definido como centro de apoio.

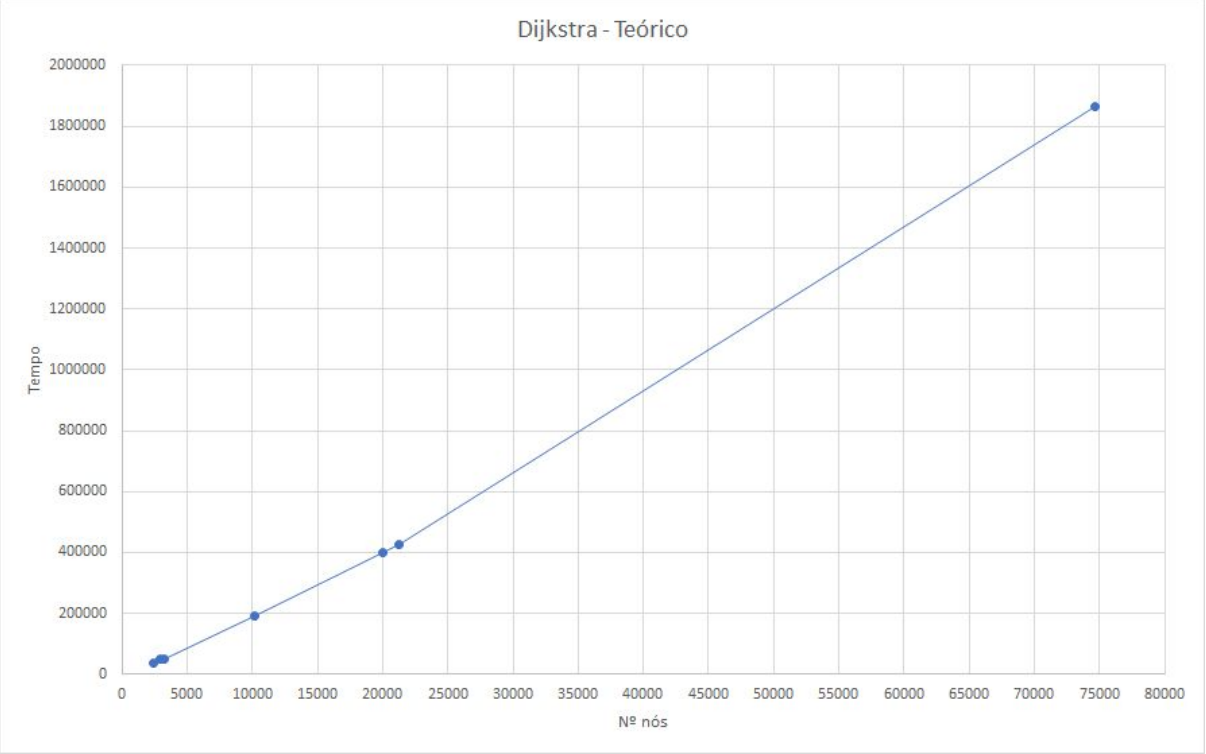
Com efeito, continua a ser visível a linearidade do gráfico anterior mesmo até ao limite estabelecido pelo mapa.



Para calcular as distâncias utilizamos o algoritmo de Dijkstra, pois os grafos implementados no programa são pesados e os pesos das suas arestas são sempre positivos devido a serem representações das distâncias euclidianas de ruas.

A complexidade teóricas espacial e temporal deste algoritmo é O((|V| + |E|)*log(|V|)) e O(N^2) respetivamente. Podemos observar o crescimento da complexidade temporal no seguinte gráfico:

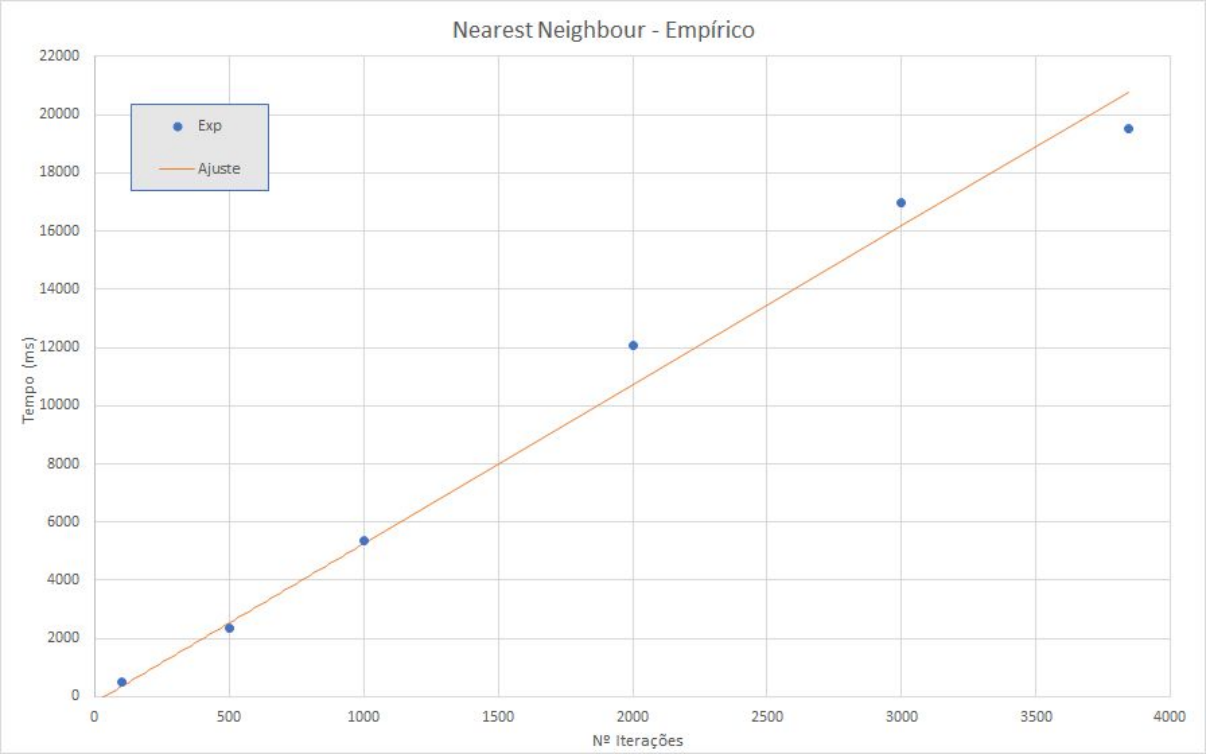
O número de nós de cada ponto foi escolhido de acordo com o número de nós dos mapas de Aveiro, Fafe, Coimbra, Porto, Braga e Lisboa.



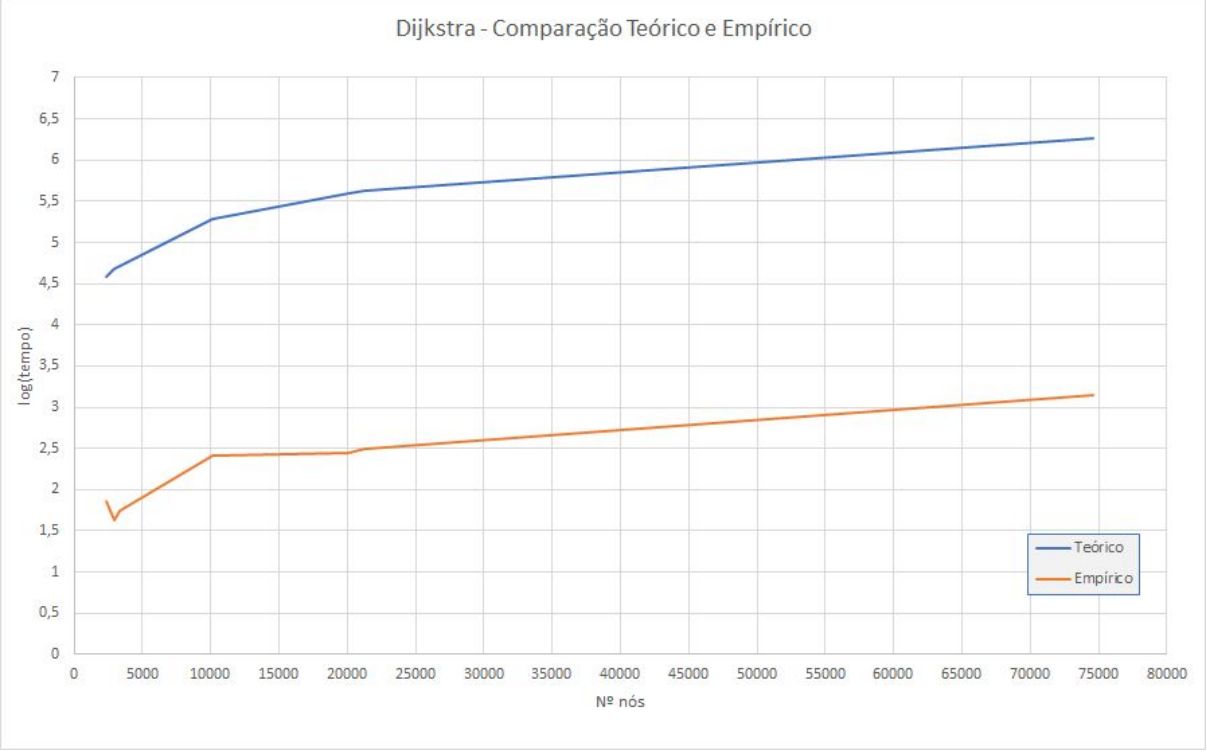
Foi feita uma média para cada mapa, mas foram observadas variações não desprezáveis entre iterações (Mapa de Coimbra - 247 a 565 µs).

Além disso, com o hardware disponível não conseguimos testar o mapa de Portugal, o que possivelmente mostraria o comportamento teórico esperado.

Testando o algoritmo no mapas fornecidos (indicados em cima), os dados obtidos permitiram corroborar a análise teórica.



A comparação dos dois conjuntos de pontos numa escala logarítmica permite observar a semelhança no comportamento das duas curvas.



Conclusão

No decorrer deste trabalho, deparamo-nos com várias dificuldades, sendo a principal a diferença entre os mapas fornecidos e o nível de conectividade esperada de um contexto urbano.

O nível de detalhe conseguido ficou aquém das expectativas iniciais, sendo que muitas das considerações tidas não foram devidamente analisadas e os casos de utilização não foram todos implementados

O trabalho foi realizado em conjunto por todos os membros do grupo, não havendo particular distinção na participação individual de cada membro.