Faculdade de Engenharia da Universidade do Porto

Concepção e analise de algoritmos

Relatório

Smart Waste

Ana Santos up200700742@fe.up.pt
Bernardo Ramos up201505092@fe.up.pt
Pedro Reis up201506046@fe.up.pt

Professores
Ana Paula Rocha
Rosaldo Rosseti

April 8, 2017

Índice

1 Introdução

1.1 Descrição do tema

No âmbito da unidade curricular Conceção e Análise de Algoritmos foi-nos proposto a resolução de um problema de atribuição de rotas a veículos de recolha de resíduos. Como pedido consideramos um sistema de coleta inteligente, onde os contentores estão equipados com sensores de volume, que indicam o momento em que o seu conteúdo precisa ser coletado.

Para maximizar a eficiência da operação, os veículos devem sair da garagem mais próxima, seguir o melhor trajeto possível para recolher os resíduos, isto é, recolher o maior número possível de contentores cheios na menor distancia possível, deixando-os para o seu tratamento na estação mais próxima.

1.2 Preliminares

Primeiramente é importante esclarecer algumas das definições usadas ao longo do relatório:

- G : Grafo que simboliza um mapa.
- V : Nós de um mapa, neste contexto o local dos ecopontos, garagens de veículos e centrais de recolha de resíduos.
- E : Arestas de um mapa, neste contexto os caminhos que unem dois ecopontos.

1.3 Identificação do problema

Este trabalho visa responder à necessidade de simplificação da gestão de recolha de resíduos das cidades. Deste modo, os camiões de recolha de resíduos não precisam realizar viagens periódicas, mas apenas realizar o itinerário que inclui contentores cheios, minimizando assim a distancia a percorrer.

No sistema desenvolvido foi considerado a monitorização remota dos contentores a partir de sensores de volume, assim como, a capacidade dos veículos de recolha em função do tipo de resíduo, a fim de determinar o melhor itinerário a executar. O itinerário é composto obrigatoriamente pela garagem de onde saem os veículos, o maior número de contentores cheios de um tipo de resíduo e a estação de tratamento de resíduos. A complexidade deste trabalho reside na possibilidade de existirem múltiplos trajetos com diferentes graus de eficiência.

1.4 Formalização do problema

1.4.1 Dados de entrada

O tipo de dados de entrada para este modelo deveria ser uma representação gráfica de uma rede onde possivelmente isto poderia ser aplicado, porém, era necessário um grande pré-processamento dessa informação. Deste modo, optamos por uma simplificada rede, de forma a simular corretamente a solução implementada.

A informação foi gerada em ficheiros de texto, sendo recolhida e transformada num grafo. Esta estrutura de dados é a mais adequada para representar uma rede de estradas com direções, onde neste caso cada nó simboliza um ecoponto e cada aresta a estrada que une dois ecopontos.

1.4.2 Limitações de aplicação

A maior limitação de aplicação, neste contexto, é a conectividade do grafo utilizado. Deste modo, é necessário avaliar a conectividade do grafo, a fim de evitar que ecopontos se encontrem em zonas inacessíveis a partir do ponto onde se encontra as garagens e as centrais de tratamento de resíduos. Para este trabalho vamos então considerar grafos com uma taxa de conexão de 100%.

Afim de avaliar a conetividade dos três grafos incluídos no trabalho foi incluída uma opção no menu principal que calcula a taxa de conetividade do mapa escolhido. Para este calculo foi realizada um visita em largura (BFS - Breadth-First Search), onde os nós visitados são guardados num vetor, desta forma, é possível estabelecer uma relação entre este número de nós e o número de nós totais do grafo.

Pseudocódigo do cálculo da taxa de conetividade:

```
for all V in G:
        source = vertex[i]
        bfs = graph.bfs(source)
        connectivity += bfs.size()/vertex.size()
end for
average percentage = (connectivity/vertex.size())*100
```

Uma outra limitação possível é a que deriva da existência de arestas de peso negativo, mas no contexto da coleta de resíduos isso não pode ser aplicado, visto que as distâncias entre vértice têm de ser forçosamente positivas.

1.4.3 Situações de contorno

Caso um grafo seja fracamente conexo, não existe nenhuma alternativa que possa eficientemente ultrapassar esta barreira. Se houver um vértice que não consiga ser acedido, mesmo que seja um ecoponto, os resíduos que este acumula nunca poderão ser recolhidos

1.4.4 Resultados esperados

A solução implementada responde aos principais objetivos deste trabalho, desta forma, espera-se que assim os resíduos dos ecopontos sinalizados como cheios foram recolhidos e o percurso realizado pelos veículos o mínimo entre cada ecoponto.

2 Solução

2.1 Técnicas de conceção

Para a resolução do problema optamos por uma estratégia gulosa, onde o tentamos resolver fazendo a escolha localmente ótima (em termos de distancia ao próximo nó a visitar) em cada fase, afim de encontrar um ótimo global. Esta estratégia é atrativa uma vez que permite obter uma solução aproximada em tempo polinomial.

Primeiramente assumimos apenas um tipo de resíduo que tinha de ser recolhido, que existia unicamente uma garagem e uma central de recolha, e não consideramos capacidades quer do veículo quer dos contentores. Para isso, quando era iniciado um caso de teste era gerado de forma aleatória os id's dos contentores (nós) que ficariam cheios. Para a implementação da solução, utilizamos o algoritmo de Dijkstra modificado, que abordaremos nó tópico seguinte. Desta forma, quando esta era executada um veículo saía da garagem, recolhia todos os resíduos, escolhendo sempre o contentor mais próximo do nó onde se encontrava e por fim, dirigia-se para a central de tratamento de resíduos.

Na etapa seguinte foi considerada a recolha seletiva dos resíduos, para isso, utilizamos as cores standard dos 3 principais resíduos recicláveis: Azul (papel), Verde (vidro) e Amarelo (Plástico). A solução implementada nesta etapa utiliza parte da solução da anterior, uma vez que para simplificação do problema, optamos por fazer a recolha ordenada por tipo: primeiro o papel, seguido de vidro e por fim o plástico. Nestas condições, tal como no caso primeiro caso, a solução implementada procura o ecoponto que esteja cheio mais perto da sua atual localização e que seja do seu tipo de resíduo, desta forma, desloca-se até lá pelo caminho mais curto, e esvazia esse contentor. Após isto, repete o processo de pesquisa e recolha e quando não houver mais nenhum contentor desse tipo de resíduo cheio, dirige-se para a central de tratamento. Isto é repetido para cada tipo de resíduo.

Por outro lado, foi também necessário ter em conta a capacidade quer do veículo quer dos contentores. Á solução anterior foi acrescentada esta condição e deste modo, cada vez que é recolhido os resíduos de um contentor e verificado se o veículo já atingiu a sua capacidade máxima, em caso afirmativo este dirige-se para a central, pelo contrario este prossegue a recolha.

Por fim, foi acrescentado a existência de múltiplas centrais e garagens de veículos, que são geradas em numero aleatório (tendo em conta o numero de nós totais) seguidamente ao carregamento de dados no grafo. É importante salientar que nesta nova etapa é tido em consideração qual a garagem que tem mais próximo de si um contentor cheio, e também, é escolhida a central de recolha mais próxima do último contentor esvaziado.

2.2 Algoritmo implementado

2.2.1 Dijkstra modificado

Neste algoritmo, é considerado como ponto inicial o nó onde o veículo se encontra. Ao iniciar o processo, são percorridos todos os vértices adjacentes a ele (unidos por uma aresta). Caso esses pontos nunca tinham sido alcançados, marcam-se como visitados e atualizam-se os valores de distância mínima e o caminho ótimo, para futuras comparações.

Após ter sido feito este processo para todos os seus adjacentes, considera-se o vértice inicial como processado. Repete-se agora o processo para todos os adjacentes dos adjacentes. Caso um dos vértices adjacentes do atual vértice a ser processado tenha um valor de distância mínima superior ao da distância do atual caminho, atualiza-se esse valor e modifica-se o trajeto guardado para o atual.

No final, iremos ter todos vértices com as distâncias e caminhos mínimas entre o ponto inicial e todos os pontos do grafo.

Pseudocódigo Dijkstra modificado:

```
Function dijkstraShortestPath(source)
    Let H be a min heap containing all vertexes yet to be processed
        for all V in G:
                mark path as NULL
                mark dist as infinite
                mark processing as false

        if V.id equals source then
                source->dist = 0
            H.add(V)
        end for

    while H not empty:
        v = H.head
        let A be a adjacent edges of v
        for all E in A:
            w = E.dest;
            if(v->dist + E.weight < w->dist ) then
                w->dist = v->dist + E.weight;
                w->path = v;
                if(not w->processing) then
                    w mark processing as true
                    H.add(w)
                sort heap
        end for
    end while
```

2.3 Análise de complexidade temporal e espacial

Para fazer a análise da complexidade temporal foram utilizados dois algoritmos: o Dijkstra e o Floyd-Warshall.

Primeiramente vamos começar por explicar o algoritmo de Floyd-Warshall, uma vez que este algoritmo, contrariamente ao de Dijkstra, não precisa de ser repetido a cada iteração uma vez que não utiliza um ponto inicial. Ou seja, são criadas duas matrizes com todas as combinações de vértices dois a dois, a qual uma é preenchida com a distancia mínima entre todos os pontos e a outra com o caminho necessário. Desta forma apenas se faz o cálculo inicialmente, e assim, sempre que seja necessário saber alguma distância, basta aceder à respetiva posição do nó de inicio e do nó de destino na respetiva matriz de distancias. Contudo para saber o caminho é utilizada uma função que recursivamente consulta a matriz do caminho e nos devolve num vetor de caminho a percorrer.

Para concluir, como podemos ver o cálculo inicial é bastante complexo, logo o algoritmo Floyd-Warshall é usado apenas em grafos com pesos negativos. Porém, para efeitos de comparação de complexidade temporal com o algoritmo de Dijkstra, resolvemos incluir este algoritmo no nosso trabalho. Os dois algoritmos, por terem metodologias muito diferentes, têm níveis de complexidade temporal e espacial muito distintas. Para simplificação desta analise foi considerado a recolha de um único tipo de resíduo assim como foi ignorado a capacidade dos veículos, uma vez que em grafos muito grandes o tempo de processamento era muito elevado.

2.3.1 Dijkstra modificado

Uma vez que utilizamos a versão modificada deste algoritmo, na qual é usada uma heap. Através desta estrutura, conseguimos obter uma complexidade temporal de $O(\log |V|)$ nas inserções e nas eliminações dos nós. No entanto é preciso aceder a todos os vértices adjacentes ao que está a ser processado, o que significa que temos de multiplicar esta complexidade por $O(|E|)$. Contudo é ainda necessário multiplicar pelo número de vezes que este algoritmo é invocado. O processo é repetido sempre que um veículo sai de uma garagem, sempre que é escolhido o próximo contentor cheio a esvaziar , e também, quando o veículo tem de ir para uma central de tratamento de resíduos. Então, em suma obtemos (com i garagens e j estações) a seguinte de complexidade temporal:

$$O((i+j) \times |E| \times \log |V|)$$

Para a complexidade espacial deste algoritmo, basta observar a existência de três vetores, um deles com o tamanho V, outro com o tamanho dos vértices adjacentes, que no pior caso possível teria o tamanho de V. O terceiro e último vetor é o que contém toda a informação sobre as arestas, de tamanho E. Tendo estas estruturas em conta, a complexidade espacial máxima apresenta-se da forma:

$$O(2 \times V + E)$$

2.3.2 Floyd-Warshall

A complexidade temporal deste algoritmo é bastante simples de verificar, dado que é invocado apenas uma vez no cálculo do caminho mais curto. Como já referido, é criado uma matriz de trajetos ótimos para todos os pontos, bastando depois ser chamado no início da execução do programa. Observando o código, é imediatamente visível a fonte do complexidade, existindo três ciclos juntos para a criação da tal matriz. Como cada iteração é repetida V vezes, ficando a forma final da complexidade:

$$O(V^3)$$

A componente espacial deste é também fácil de se verificar. No início são criados 2 matrizes de tamanho V*V. Seguidamente, numa delas a distancia mínima de cada dois pontos e na outra o caminho necessário. Isto leva a uma complexidade espacial de:

$$O(2 \times V^2)$$

2.3.3 Análise de dados recolhidos

Para efeitos de estudo foram criados três grafos distintos, um com 32 nós, outro com 156 nós e por último um com 300 nós, deste modo é possível fazer uma análise empírica da complexidade temporal dos dois métodos. Para isso, foram gerados 10 simulações, onde o número de contentores cheios, o numero de garagens e o número de centros de recolha era concebidos de forma random. Porém, em ambos os algoritmos foram utilizadas exatamente as mesmas condições de teste. Na figura 1, como podemos confirmar, o algoritmo de Dijkstra é bastante mais eficiente que o de Floyd-Warshall.

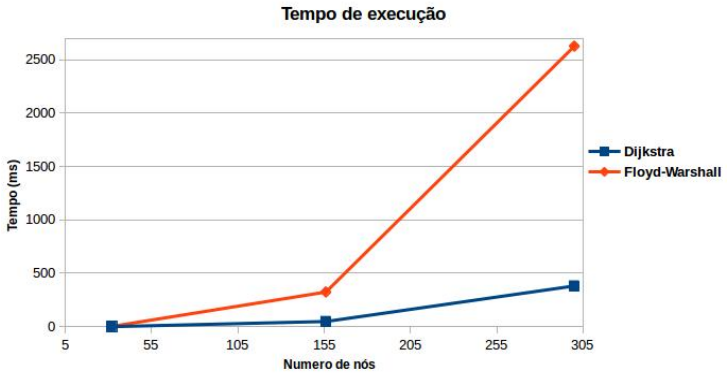

Figure 1: Gráfico dos tempos de execução

Uma outra análise realizada foi o comportamento destes algoritmos tendo em conta o número de contentores cheios, para o efeito, foi considerado apenas o grafo de 156 nós. Verificou-se (figura 2) que o algoritmo de Dijkstra tem um comportamento muito semelhante independentemente no número de nós cheios, tendo apenas um ligeiro crescimento, já o algoritmo de Floyd-Warshall teve um comportamento mais inconstante, contudo, demorou sempre mais tempo a computar uma solução.

Figure 2: Gráfico dos tempos de execução

3 Lista de casos de utilização

Este sistema visa a eficiência de um setor crítico para a manutenção de cidades, por isso não é difícil de imaginar que a existência atual deste tipo de sistemas. Prova disto é a empresa Citibrain, cujo principal foco é a da produção de sistemas inteligentes que solucionam problemas relacionado com a gestão de metrópoles. Um dos seus produtos curiosamente tem o nome de "Smart Waste", em que, recorrendo a sensores de volume implantados em todos os contentores, otimizam a frequência e rotas dos veículos de recolha. Adicionalmente, utilizam nos contentores um acelerómetro para deteção de movimentos, assim como mais alguns alarmes adicionais para segurança, como um detetor de incêndios. A solução dada pela empresa é fundamentalmente semelhante com a apresentada no nosso trabalho.

De forma idêntica a ECube Labs procura também rentabilizar todo o processo de recolha, usando compactadores de lixo para aumentar o espaço que cada contentor pode armazenar, tal com a Citibrain também utilizam sensores de carga para saber quando cada contentor está cheio, e assim sinalizam à empresa de recolha de lixo, na altura mais oportuna, o trajeto ótimo para proceder à coleta de resíduos.

3.1 Principais funcionalidades implementadas

Neste trabalho foi implementado uma solução simplificada para a gestão dos resíduos numa cidade, contudo, após algum desenvolvimento poderia provavelmente ser aplicado na resolução de problemas tal como fazem empresas referidas anteriormente.

4 Dificuldades encontradas

Após estarmos mais familiarizados com os algoritmos, o processo de configurar o IDE Eclipse para este projeto foi algo moroso. Tivemos múltiplos problemas técnicos com alguns ficheiros dados, como exemplo o GraphViewer, que atrasou substancialmente a fase inicial de alguns membros.

Outro aspeto que também provocou algumas dificuldades foi a exportação dos ficheiros de texto derivados do Open Street Maps. Houve problemas com a geração do grafo porque havia vértice que estavam a ser interpretados incorretamente. Para contornar estes obstáculos acabamos por criar novos ficheiros manualmente, mais pequenos que tiveram utilidade na facilidade de teste.

Por fim, existe uma situação em que o nosso algoritmo pode não apresentar a melhor solução. Um exemplo disto é o da figura 3. A solução ótima seria recolher o conteúdo do nó 'R' e seguidamente o de 'A' onde a distancia percorrida seria de 360. Porém a solução implementada, como procura sempre a distancia mínima a cada passo, o que faria era recolher 'A' primeiramente e só depois 'R' onde por sua vez a distancia percorrida seria de 520, uma vez que acabou por de repetir muito caminho desnecessariamente. A única maneira de se resolver este problema é através de uma abordagem de força bruta, em que se teria de calcular todas os percursos, a partir de todos os vértices. Isto implicaria uma complexidade temporal maior, e seria muito menos eficiente que o algoritmo utilizado. Quando falamos de grafos pequenos como este pode compensar, mas vendo num panorama de uma cidade, o cenário pode mudar drasticamente.

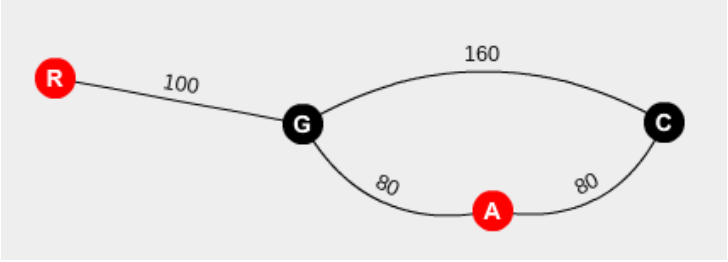

Figure 3: Possível caso

5 Trabalho desenvolvido por cada elemento

Subdividimos o projeto em três componentes distintas: a estrutura principal do código, onde os algoritmos e a interface com o utilizador são implementados, incluindo o visualizador de grafos. Esta parte foi tratada pela Ana Santos, com algum contributo do Pedro Reis.

A organização e elaboração do relatório foi tratada pelo Pedro Reis, com informação adicional fornecida pela Ana Santos, tal como pseudo-código, dados analisados, etc.

A componente que resta foi a conversão do presente relatório para Latex. Sendo que esta foi tratada, em parte, pelo Bernardo Ramos. Distribuímos desta maneira o nosso trabalho uma vez que houve problemas inicialmente com o software utilizado o que impediu a participação ativa de todos os membros na elaboração do código.

Ana Santos - 60 %
Pedro Reis - 30 %
Bernardo Ramos - 10%

6 Conclusão

Com este projeto conseguimos aprender algumas das dificuldades da geração e utilização de algoritmos gulosos complexos, e das suas tangíveis vantagens em setores relevantes da sociedade.

Reconhecemos também que a nossa solução possa não ser a mais eficiente. No entanto, a complexidade temporal e espacial desta compensa comparativamente á solução ótima, uma vez que gera uma solução próxima (e em muitos caso, é efetivamente a melhor), com um tempo de cálculo mínimo.

Concluindo, isto não é apenas um algoritmo, pelo contrário, poderá se traduzir numa valiosa ferramenta para o futuro. Diminuição do consumo de combustíveis, melhoria de qualidade de ar devido à diminuição das emissões dos veículos, melhoria da qualidade de vida dos habitantes e redução de custos são algumas das vantagens associadas.

7 Bibliografia

1. Rodrigues, P., Pereira, P., Sousa, M. Programação em C++ - Conceitos básicos e Algoritmos. FCA.
2. Cplusplus.com. Acedido em 2017. Disponível em :
   http://www.cplusplus.com/
3. citibrain.com. Acedido em 2017. Disponível em:
   http://www.citibrain.com/pt/solutions/smart-waste-pt/
4. ecubelabs.com. Acedido em 2017. Disponível em:
   http://ecubelabs.com/