Projeto #1: Minimum Weight Cut Algoritmos Avançados

João Bernardo Tavares Farias
joaobernardo0@ua.pt

***Resumo*—Este documento apresenta um breve estudo, realizado no âmbito da disciplina de Algoritmos Avançados, entre a Pesquisa Exaustiva e a Pesquisa Gulosa, bem como os resultados recolhidos.**

I. Introdução

Este relatório apresenta o trabalho realizado no âmbito do primeiro projeto da disciplina de Algorimos Avançados. São comparadas os dois tipos de soluções desenvolvidas: a **pesquisa exaustiva** e a **pesquisa gulosa** para o problema *Minimum Weight Cut*.

II. Apresentação do Problema

Dado um grafo não orientado **G**, com **N** vértices e **E** arestas, o objetivo deste problema é encontrar um corte em que os vértices fiquem divididos em dois subconjuntos **S** e **T**, não vazios, de maneira que o peso das arestas que atravessam os dois subconjuntos (cujos vértices estão em subconjuntos diferentes) seja o menor possível.

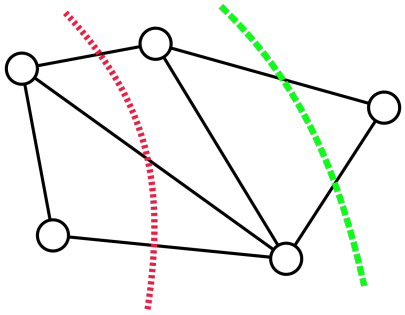

Figura 1. Minimum Cut

Como é possível observar na figura acima representada, o corte mínimo está representado a verde, mas para além deste existe outro corte mínimo que pode ser adotado se forem cortadas as arestas que ligam o vértice situado no canto inferior esquerdo às outras duas arestas. De notar que, o exemplo representa um grafo sem peso nas arestas, o que difere do problema *Minimum Weight Cut Problem* uma vez que neste problema o peso das arestas representa a distância euclidiana entre dois pontos. Para resolver este problema, foram usadas duas soluções:

Pesquisa exaustiva é uma técnica usada para resolver problemas combinatórios, uma vez que são gereados todos os elementos do domínio do problema e selecionado o melhor. Esta estratégia encontra sempre a solução ótima.

Pesquisa gulosa é uma técnica usada para encontrar a solução ótima localmente, esperando que esta solução possa vir a ser a solução global do problema. Ao contrário do algoritmo anterior, este pode ou não encontrar a solução ótima para o problema. A grande diferença é que este algoritmo é computacionalmente mais rápido que o anterior, devido a utilizar a heurística como parâmetro de avaliação.

III. Análise Computacional

A. Pesquisa Exaustiva

O problema apresentado, ao ser resolvido utilizando a pesquisa exaustiva, gera todas as combinações de vértices **exceto** o conjunto **vazio** e o **conjunto completo**. Estes subconjuntos são iterados e para cada um, inicialmente é calculado o conjunto complementar ao que está a ser iterado no momento, isto é, o conjunto que tem os nós que não estão no subconjunto do momento, e são verificadas as arestas que pertencem aos dois subconjuntos. Para cada uma dessas arestas é calculado o custo de corte e se for menor ao anteriormente calculado o valor é atualizado. O pseudocódigo pode ser analisado a seguir:

Algorithm 1 Minimum cost

```
Initialize list allCumbinations
Initialize minCost
for subset in allCumbinations do
    Get complementar subset
    Calculate cost associated with current subset
    if cost < minCost then
        minCost = cost
    end if
end for
return minCost
```

A complexidade computacional utilizada para calcular todas as combinações de vértices é O(2^n) e pode ser representada dada pela seguinte fórmula:

$$\sum_{k=2}^{n} \binom{n}{k} = 2^n - 2$$

onde **n** representa o número de nós. Como referi, os dois subconjuntos não podem ser vazios nem conter todos os elementos, pelo que é necessário exclui-los. Para remover programaticamente estes dois subconjuntos, a capacidade computacional é **O(1)** enquanto calcular o custo associado a cada

subconjunto tem complexidade computacional **O(n)**. No final, e combinando as duas complexidades computacionais, surge **O(n)** $\times$**O**$(2^n) =$ **O(n**2^n**)**.

B. Pesquisa Gulosa

A pesquisa gulosa está diretamente associada com uma heurística que leva a que a solução seja encontrada mais rapidamente. No caso do problema apresentado, a heurística utilizada levou a uma ordenação crescente da lista de adjacências, uma vez que os vértices com menor número de arestas associadas, têm uma maior probabilidade de terem um menor custo de corte e consequentemente serem solução do problema.

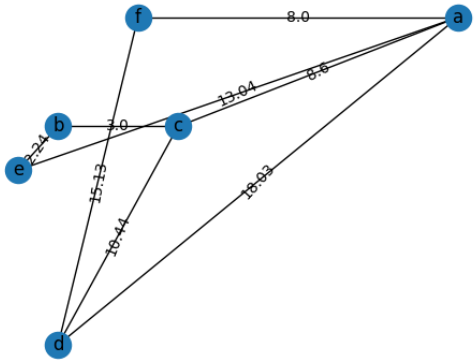

Figura 2. Graph with 6 nodes and 8 edges

Como é possível observar a partir da figura, o grafo apresentado tem a seguinte lista de adjacência:

a	$\rightarrow$ c $\rightarrow$ d $\rightarrow$ e $\rightarrow$ f
b	$\rightarrow$ c $\rightarrow$ e
c	$\rightarrow$ a $\rightarrow$ b $\rightarrow$ d
d	$\rightarrow$ a $\rightarrow$ c $\rightarrow$ f
e	$\rightarrow$ a $\rightarrow$ b
f	$\rightarrow$ a $\rightarrow$ d

Neste grafo o custo da solução é **5.24** e está associado com o corte das arestas **BE** e **BC**. Assim, o subconjunto **S** ficaria apenas com o vértice **B** enquanto o subconjunto **[T]** continha o resto dos vértices **[A,C,D,E,F]**.

Fazendo a análise no ponto de vista da Pesquisa Gulosa, se a lista de adjacência estivesse ordenada pelo tamanho do número de nós conectados, a solução seria encontrada nos primeiros instantes. Deste modo, existem subconjuntos que não precisam de ser calculados e consequentemente a solução é encontrada mais rapidamente. Neste contexto foram tidas em conta as listas de adjacência mais pequenas e com o mesmo tamanho, sendo as outras descartadas. No exemplo da **Figura 2** apenas seriam tidos em conta as seguintes listas de adjacência:

b	$\rightarrow$ c $\rightarrow$ e
e	$\rightarrow$ a $\rightarrow$ b
f	$\rightarrow$ a $\rightarrow$ d

Tal como descrito, existem duas operações: a **ordenação** das lista de adjacência, seguido do **cálculo do custo**, **O(n)**, com complexidade computacional combinada de **O(nlog(n))**, onde **N** representa o número de vértices.

Mais uma vez, é importante referir que, este algoritmo pode não encontrar a solução ótima para o problema.

IV. Geração de Grafos

Os grafos utilizados para a resolução do problema foram gerados aleatoriamente, com uma *seed* definida. Para cada número de vértices foram gerados, regra geral, 4 grafos com a seguinte densidade de arestas **12.5%, 25%, 50% e 75%** do número máximo de vértices possíveis, tendo sido associadas coordenadas a cada vértice, entre 1 e 20. O peso das arestas, tal como já foi referido é a distância entre dois pontos. Caso os grafos não tenham o número mínimo de arestas **N-1**, onde N representa o número de vértices, então esse grafo não é gerado, e uma nova densidade de arestas é tida em conta.

V. Resultados

Neste problema foram gerados grafos com número vértices entre **4** e **20**. A solução inclui o tempo de execução, o custo, os subconjuntos gerados e o número de operações básicas executadas.

A. Tempo de execução

Quanto aos tempos de execução é possível observar no gráfico seguinte a comparação entre o algoritmo exaustivo e o algoritmo guloso:

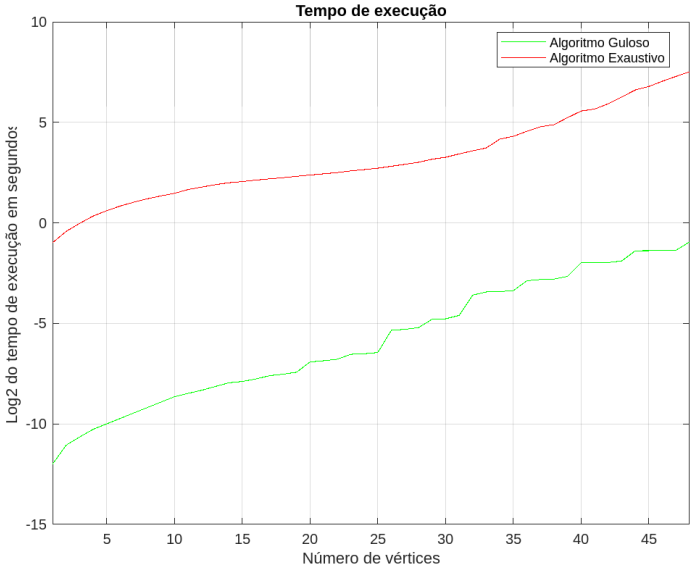

Figura 3. Gráfico comparativo entre as duas soluções

Para normalizar a curva do gráfico, os valores do eixo do **Y** são obtidos aplicando o logaritmo em base 2 aos resultados originais. O algoritmo guloso é computacionalmente mais

rápido comparativamente ao algoritmo exaustivo, mas em termos de eficiência a encontrar a solução ótima, o algoritmo exaustivo é mais eficaz, uma vez que encontra sempre a solução ótima, quando esta existe.

B. Número de operações básicas

Quanto ao número de operações básicas, que no caso do problema corresponde a cada iteração na procura da solução, os resultados podem ser observados na figura seguinte.

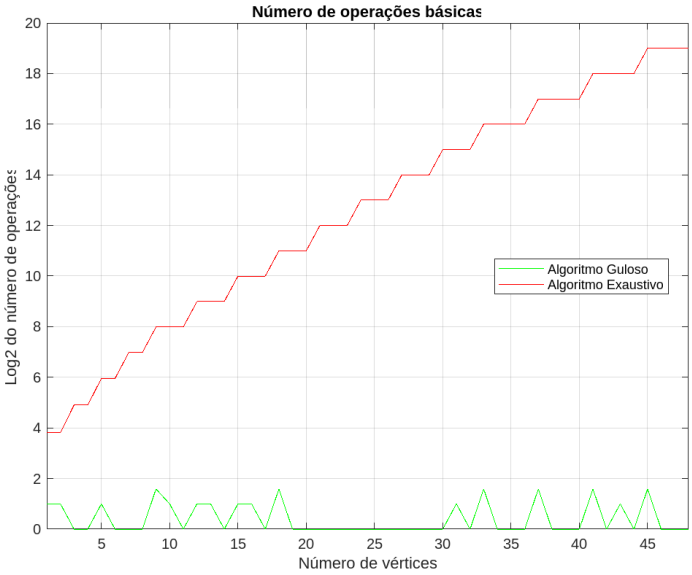

Figura 4. Gráfico comparativo entre as duas soluções

À semelhança do gráfico anterior, os valores do eixo do **Y** são obtidos depois de aplicar o logaritmo na base 2 aos resultados.

C. Número de soluções encontradas

Quanto ao número de soluções encontradas, não houve muitas situações em que fossem obtidas mais do que uma, isto porque o peso das arestas é a distância entre dois pontos. Sendo assim, é muito improvável gerar aleatoriamente vértices com o mesmo número de arestas e à mesma distância. Mas houve alguns casos em que foi possível encontrar mais do que uma solução.

Na pesquisa exaustiva, de entre os 20 grafos gerados, apenas **dois** tiveram mais do que uma solução encontrada. Esses grafos encontram-se representados a seguir:

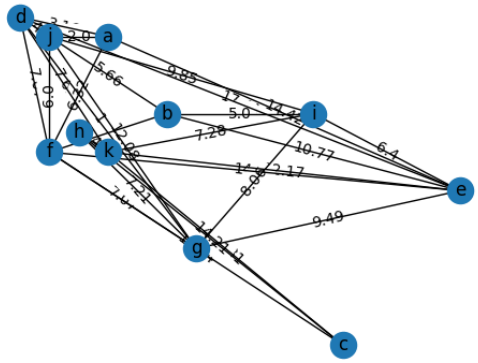

Figura 5. Grafo com 2 soluções possíveis

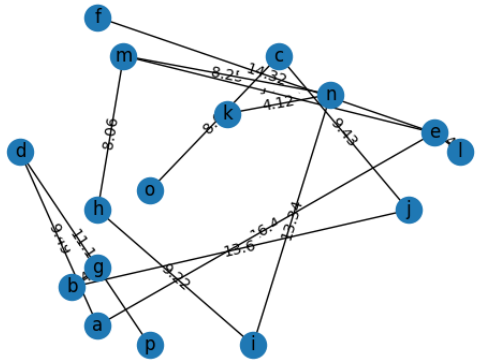

Figura 6. Grafo com 3 soluções possíveis

Na pesquisa gulosa, como não são calculados todos os subconjuntos de solução possível, apenas uma solução é encontrada.

D. Problemas de maior escala

Para 20 vértices, o tempo de execução do algoritmo *Greedy* é cerca 600 vezes mais rápido do que o algoritmo *Exhaustive*. Para problemas de dimensão superior, o tempo de execução dos algoritmos pode ser aproximado pela seguinte fórmula:

$$\frac{n * 2^n}{20 * 2^{20}} * 145$$

para o **algoritmo exaustivo**, onde 145 representa o tempo de execução para 20 vértices, e por:

$$\frac{n * log(n)}{20 * log(20)} * 0.2$$

para o **algoritmo guloso** onde 0.2 representa o tempo de execução para 20 vértices.

VI. Estrutura do código

O ficheiro *models.py* tem as principais classes para a resolução deste problema. A primeira a ser chamada é a class **Problema** onde é criado um grafo com um número incremental de vértices, começando em 4. A classe **Grafo** é responsável por criar os vértices e associar-lhes uma posição e é responsável também por criar as arestas e associar-lhes o respetivo peso. Quando as arestas são construídas, existem várias verificações feitas para não incluir várias arestas entre os mesmos vértices e para não incluir também arestas entre o mesmo vértice. É também calculada a lista de adjacência para cada grafo.

Após o grafo estar construído, o método *solveProblem* da classe **Problem** é chamado e a solução começa a ser procurada. No final são escritos para disco ficheiros associados a cada grafo que contém cada um dos subconjuntos, o custo da solução, o número de soluções encontradas, o tempo de execução, a lista de adjacência e o número de operações básicas executadas.

VII. Conclusão

A pesquisa gulosa e a pesquisa exaustiva são duas boas maneiras de encontrar a solução de um problema, dependendo da sua dimensão. Para problemas mais pequenos, a solução exaustiva é a mais indicada para resolver o problema, mas para problemas de dimensão superior é a pesquisa gulosa a mais indicada, por ser mais rápida, ainda que por vezes possa não devolver a solução correta.

Referências

[1] Stanford Algorithms, “Graphs and Minimum Cuts” URL: https://www.youtube.com/watch?v=4Ih3UhVuEtw

[2] Michel X. Goemans, Combinatorial Optimization, May 10th, 2007, URL: https://math.mit.edu/ goemans/18433S07/mincut.pdf

[3] Minimum cut, URL: https://en.wikipedia.org/wiki/Minimumcut

[4] NX Graph, URL: https://nx.dev/nx/dep-graph

[5] Get OPT, URL: https://docs.python.org/3/library/getopt.html