Segundo Trabalho de Algoritmos Avançados - Corte mínimo de um grafo

João Reis, 98474

***Resumo* - Este documento tem como intuito documentar a implementação e a análise realizada sobre o algoritmo de Karger-Stein para resolver o problema de grafos do corte mínimo. Ao longo deste documento é explicado o raciocínio e implementação do algortimo. No final, é realizada uma análise dos resultados obtidos.**

***Abstract* - This paper is intended to document the implementation and analysis performed on the Karger-Stein algorithm for solving the minimum cut graph problem. Throughout this paper the reasoning and implementation of the algorithm is explained. At the end, an analysis of the results obtained is performed.**

***Key words* - Python, Grafo, Corte mínimo, Algoritmo de Karger, Algoritmo de Karger-Stein, Tempo de Execução, Número de Iterações, Números de Soluções Encontradas**

I. INTRODUÇÃO

Para o segundo trabalho da unidade curricular Algoritmos Avançados, é proposto a resolução de um pequeno problema envolvendo grafos, utilizando um algoritmo que utilize a aleatoriedade como base.

Para testar este algoritmo podem ser utilizados grafos gerados aleatoriamente, tal como no primeiro projeto desta unidade curricular, ou grafos em ficheiros *txt* de *Sedgewick & Wayne*, ou até mesmo, utilizando estes 2 métodos.

Para resolver o problema foi implementado o algoritmo de Karger-Stein, uma versão melhorada do algoritmo de Karger. Após o teste deste algoritmo, foram recolhidos dados e feita uma análise da complexidade computacional deste algoritmo.

Para executar o código, basta executar um dos comandos indicados abaixo, dependendo dos grafos que são pretendidos testar. Caso seja apenas os grafos gerados aleatoriamente, execute o seguinte comando.

```
$ python3 karger-stein_algorithm.py -g
```

Caso seja apenas os grafos de *Sedgewick & Wayne*, execute o seguinte comando, escolhendo qual dos ficheiros pretende testar.

```
$ python3 karger-stein_algorithm.py -f <one
of: [ tiny, medium, large, all ]>
```

Caso seja ambos os grafos, execute o seguinte comando, escolhendo qual dos ficheiros pretende testar.

```
$ python3 karger-stein_algorithm.py -g -f
<one of: [ tiny, medium, large, all ]>
```

Este relatório é acompanhado com todos os ficheiros python produzidos para implementar este algoritmo, bem como a geração aleatória de grafos.

II. PROBLEMA

Antes de explicar o problema em si, é necessário compreender a estrutura criada para representar os grafos não direcionados e sem pesos nas arestas associadas.

Um grafo será definido por um dicionário, onde a chave será um vértice e o seu valor será uma lista de todos os vértices à qual esse está ligado, isto é, uma lista de todas as arestas associadas a esse vértice. Um exemplo de um grafo poderia ser o seguinte:

```
{'a': ['d'], 'b': ['d', 'c'], 'c': ['b',
'd'], 'd': ['b', 'a', 'c']}
```

Os vértices são representados também utilizando um dicionário, onde a chave é uma representação alfabética e o seu valor será as coordenadas onde se situa esse vértice, tal como indica a seguinte representação:

```
{'a': (9, 2), 'b': (13, 2), 'c': (2, 14),
'd': (3, 17)}
```

Tendo estes conhecimentos, o problema que é proposto é encontrar um corte mínimo para um dado grafo não direcionado G(V, E), com n vértices e m arestas. O corte mínimo de G é uma divisão dos vértices do gráfico em dois conjuntos complementares, S e T, tais que o número de arestas entre o conjunto S e o conjunto T é o menor possível, ou seja, o número de arestas que separam esses conjuntos deve ser o menor.

Os próximos dois tópicos deste relatório explicarão como foi implementada a solução para este problema, utilizando os dois algoritmos pedidos.

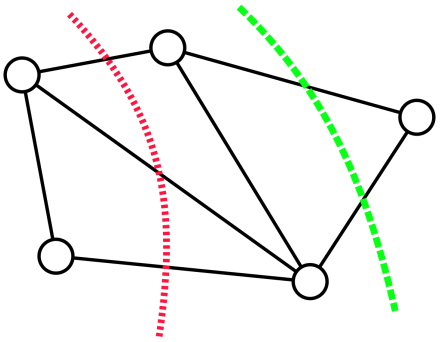


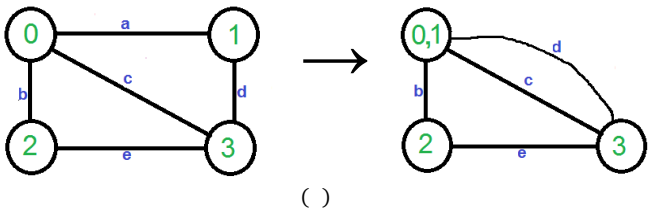
Fig. 1 - Representação gráfica do problema (o corte verde é uma solução)

III. ALGORITMO DE KARGER

Antes de explicar o algoritmo utilizado, é necessário explicar o seu algoritmo base, já que o implementado é uma extensão e/ou melhoria do algoritmo de Karger. ESte algoritmo foi desenhado para resolver o problema de corte mínimo.

A ideia deste algoritmo é simples. Primeiramente, escolhe-se aleatoriamente uma aresta do grafo. A aresta escolhida é removida e os dois vértices que formam essa aresta são então contraídos. De seguida, remove-se, se houver, todas as arestas que se liguem ao mesmo vértice, ao que se chama de laço.

Por exemplo, no seguinte grafo, o da esquerda, a aresta escolhida foi a aresta 'a', portanto remove-se essa aresta e contrai-se os vértices.



( )

Fig. 2 - Representação de uma contração de dois vértices

Repetimos este processo até ficarem apenas dois vértices. Quando esta condição se verificar, o número de arestas que ligam os dois vértices restantes será uma possível solução para o problema do corte mínimo.

Continuando o exemplo anterior, após ser escolhida aleatoriamente e removida a aresta 'd', contraindo os seus vértices e removendo o laço criado, a aresta 'c', obtemos o seguinte resultado.

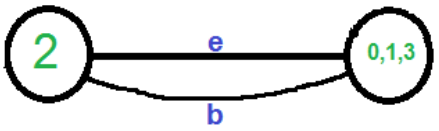


Fig. 3 - Representação final do grafo da Fig.2, após a aplicação deste algoritmo

A solução do corte mínimo para este exemplo é então duas (2) arestas.

É de frisar que é possível que a solução encontrada não seja a melhor das soluções.

IV. ALGORITMO DE KARGER-STEIN

O algoritmo explicado no tópico anterior tem algumas desvantagens, como por exemplo, o tempo de execução e o facto deste algoritmo só falhar se contrair uma aresta que faz parte da melhor solução.

Para melhorar estes pontos, foi então criado uma extensão ao algoritmos de Karger, o algoritmo de Karger-Stein.

A probabilidade de contrair a aresta errada aumenta à medida que o número de vértices diminui. Como a escolha de uma aresta errada é mais provável mais tarde no algoritmo, a estratégia passa por repetir o algoritmo de Karger em fases onde o grafo é suficientemente grande.

Se o tamanho do grafo, após algumas contrações devidas ao algoritmo de karger, for menor ou igual a seis (6) vértices, aplica-se então pesquisa exaustiva para assim aumentar a probabilidade de encontrar a melhor solução.

Se contrairmos n vértices até $\frac{n}{\sqrt{2}}$ vértices, a probabilidade de não contrairmos uma aresta errada é cerca de 50%.

O pseudocódigo que serviu de **inspiração** para a implementação deste algoritmo é o seguinte:

- **KargerStein**(G = (V,E)):
  - n ← |V|
  - if n < 4:
    - find a min-cut by brute force \\ time O(1)
  - Run Karger's algorithm on G with independent repetitions until $\lfloor \frac{n}{\sqrt{2}} \rfloor$ nodes remain.
  - G_1, G_2 ← copies of what's left of G
  - S_1 = **KargerStein**(G_1)
  - S_2 = **KargerStein**(G_2)
  - **return** whichever of S_1, S_2 is the smaller cut.

Fig. 4 - Pseudocódigo do algoritmo de Karger-Stein

Ao contrário do explicado até agora e do pseudocódigo indicado, quando o número de vértices for menor ou igual a 6, aplica-se o algoritmo de Karger **nC2 (combinações de n 2 a 2) vezes** em vez de se aplicar **pesquisa exaustiva**. Isto porque o que é pedido para este trabalho é a implementação de um randomized algorithm, ou seja, não seria coerente aplicar pesquisa exaustiva nesta situação. Por outro lado, ao aplicar desta maneira, a probabilidade de não encontrar o corte mínimo é de $\frac{1}{n}$.

Caso o número de vértices do grafo for maior que seis (6), aplicamos o algoritmo de Karger até originar um grafo com $\frac{n}{\sqrt{2}}$ vértices. De seguida, de forma recursiva, a função *search* é chamada mas desta vez para os grafos resultantes do algoritmos de Karger, como está indicado no pseudocódigo.

No final é retornada a melhor solução até então encontrada.

V. RESULTADOS

Neste tópico, serão analisados os resultados para o algoritmo de Karger-Stein, de acordo com os seguintes parâmetros:

- Tempo de Execução;
- Número de Iterações/Operações Básicas;
- Número de Soluções Encontradas.

Para fazer esta análise, foram gerados 168 grafos com 4 a 49 vértices. Os grafos disponíveis no E-Learning não fazem parte desta análise, dado que destoam dos grafos gerados e o grafo presente no ficheiro *SWlargeG.txt* demora demasiado tempo a interpretar e executar.

Em relação ao tempo de execução, de acordo com o gráfico presente na figura 5, o algoritmo é rápido comparando com os resultados obtidos no projeto anterior, uma vez que consegue computar a solução de uma gráfico com 49 vértices à volta de 4 segundos.

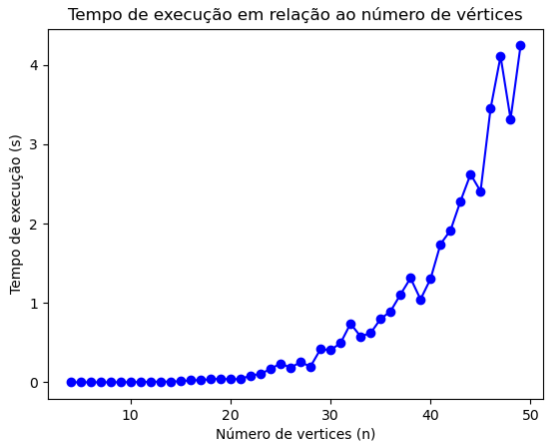


Fig. 5 - Tempo de execução de acordo com o número de vértices

Verifica-se, pela análise do gráfico acima, que à medida que o número de vértices aumenta, o tempo de execução aumenta exponencialmente.

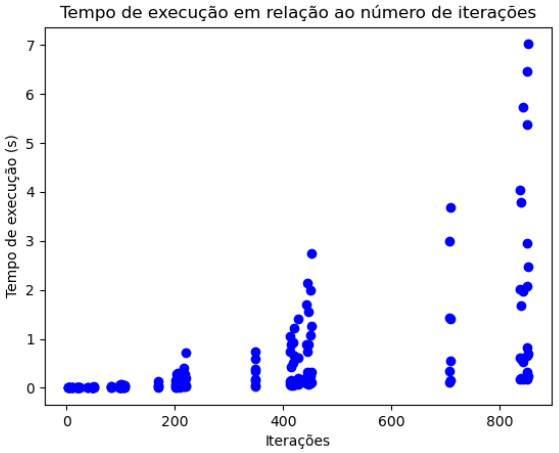


Fig. 6 - Tempo de execução de acordo com o número de iterações

O tempo de execução também aumenta à medida que o número de iterações aumenta. É possível identificar a forma de um gráfico exponencial. Mas como é possível por exemplo haver mais de 800 iterações e o tempo de execução ser inferior a um (1) segundo? Isto porque o grafo testado tem menos arestas para serem analisadas do que outros grafos com mais arestas.

O gráfico da figura 7 descreve a relação entre o número de vértices com o número de iterações.

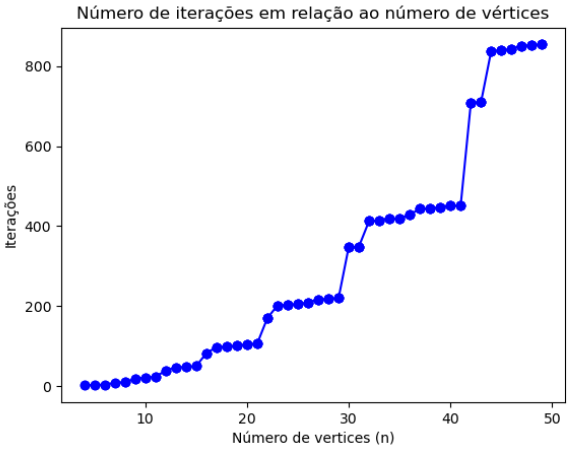


Fig. 7 - Número de vértice em relação ao número de iterações

Quanto maior o número de vértices, maior o número de iterações, uma vez que terá que percorrer mais arestas.

Em relação ao número de soluções encontradas, este número aumenta à medida que o tamanho dos grafos aumenta, como demonstra a figura seguinte.

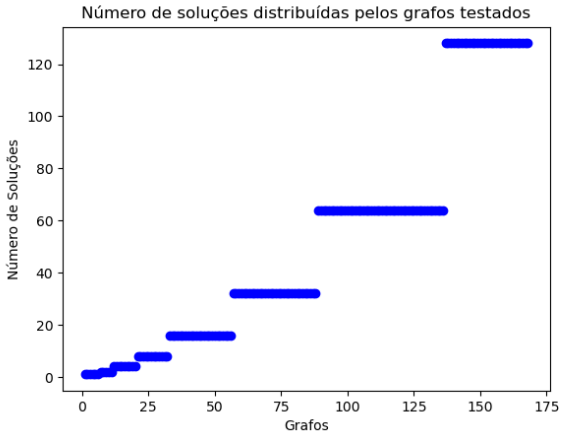


Fig. 8 - Número de soluções distribuídas pelos grafos testados

O tempo de execução total, ou seja, o tempo de execução para resolver o problema para cada um dos 168 grafos foi de, aproximadamente, 90 segundos, isto é, 1 minuto e 30 segundos.

Também há conclusões a tirar da comparação entre os resultados do algoritmo de Karger-Stein com os resultados obtidos no projeto anterior, onde foram implementados dois algoritmos: pesquisa exaustiva e pesquisa gulosa para o mesmo problema de grafos.

Podemos dizer que o algoritmo de Karger-Stein resolve os pontos fracos da pesquisa exaustiva e da pesquisa gulosa. Por um lado, consegue solucionar o problema com com um bom tempo de execução, ao contrário da pesquisa exaustiva, onde o tempo de execução era enorme mas encontrava a melhor solução. Por outro lado, encontra na maioria das vezes a melhor solução, ao contrário da pesquisa gulosa que, apesar de ter um ótimo tempo de execução, não encontrava a melhor solução na maioria das vezes.

VI. CONCLUSÃO

Em suma, podemos concluir que o algoritmo de Karger-Stein é um "meio termo" entre os algoritmos implementados no projeto anterior: pesquisa exaustiva e pesquisa gulosa. Tem um bom tempo de execução, e a probabilidade de encontrar a melhor solução é grande.

VII. REFERÊNCIAS

[1] Randomized Algorithms
https://web.stanford.edu/class/archive/cs/cs161/cs161.1138/lectures/11/Small11.pdf

[2] Karger's algorithm for Minimum Cut | Set 1 (Introduction and Implementation) - GeeksforGeeks
https://www.geeksforgeeks.org/kargers-algorithm-for-minimum-cut-set-1-introduction-and-implementation/

[3] Karger's algorithm - Wikipedia
https://en.wikipedia.org/wiki/Karger%27s_algorithm

[4] Min Cut and Karger's Algorithm - ppt download
https://slideplayer.com/slide/15146715/