Finding a Minimum Edge Dominating Set for a graph

João Carvalho, 89059

***Resumo* – Neste relatório é abordado e analisado o problema do Edge Dominating Set de um determinado grafo. O objetivo deste problema é encontrar o Edge Dominating Set de menor tamanho possível. Para isso foram desenhados e testados dois algoritmos: *Exhaustive search* e algoritmo com *greedy heuristics*. Como conclusão são analisados e comparados os resultados obtidos.**

***Abstract* - This report analyzes the Minimum Edge Dominating Set problem for a graph, which means, finding an Edge Dominating Set of smallest possible size using two algorithms: *Exhaustive search* and *Greedy search*.**

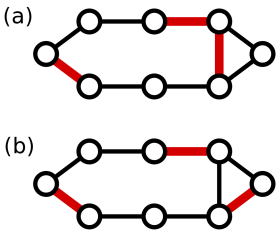
I. Introduction

Os *dominating sets* dos grafos é um dos conceitos da teoria dos grafos que atraiu muitos pesquisadores a trabalhar nele por causa de suas muitas e variadas aplicações em campos como álgebra linear e otimização.

Neste trabalho foi tratado o *dominating set* de arestas, especificamente, o problema do *edge dominating set* de menor tamanho, um problema, quanto à complexidade computacional, NP-completo.

II. Problem Analysis

Um Edge dominating set de um grafo $G = (V, E)$ é um subconjunto $D \subseteq E$, tal que qualquer aresta que não está em D é adjacente a pelo menos uma aresta em D. O *minimum edge dominating set* (*MEDS*) é o subconjunto D de menor tamanho, isto é, com o menor número de arestas possível.



Acima podemos observar dois exemplos de um *minimum edge dominating set* de um grafo com oito vértices e nove arestas. Nestes casos o *MEDS* possui uma tamanho de 3.

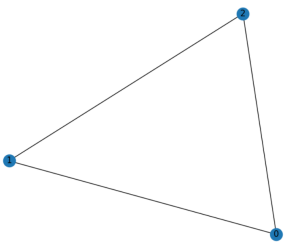
Para grafos com dimensões muito reduzidas é relativamente fácil descobrir uma solução apenas analisando a sua estrutura, contudo, no caso de grafos continuamente maiores em número de vértices e arestas a resolução do problema apenas analisando a estrutura diretamente torna-se impossível. Assim, iremos analisar a resolução do problema com base nos algoritmos já mencionados.

III. Solution Analysis

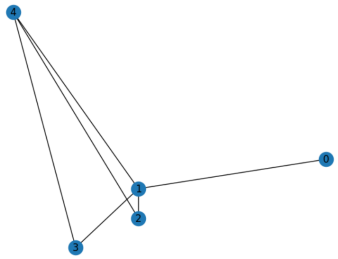
Primeiramente o programa, desenvolvido em python3, implementa um grafo com n vértices (número dado pelo utilizador) e com um número aleatório de arestas.

Os vértices do grafo são pontos 2D com coordenadas *random* entre 1 e 9. O número de arestas por vértice é determinado aleatoriamente também.

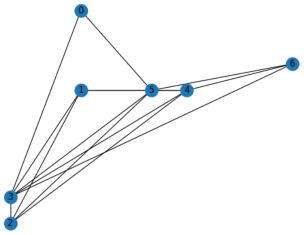
Abaixo estão alguns exemplos de grafos gerados com três, cinco e sete vértices respectivamente.



Grafo 1



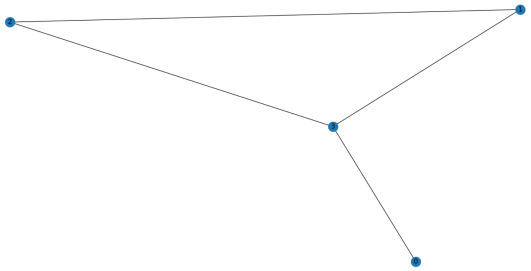
Grafo 2



Grafo 3

O grafo está representado por uma matriz de incidência uma vez que, como a análise é sobre as arestas, a matriz de incidência dá-nos a relação direta dos vértices e suas arestas incidentes.

Para essa representação e implementação da resolução é usada uma classe "Graph" que contém os métodos para esses efeitos. O método "*add_edge(v1,v2)"* é usado para criar a matriz recebendo os dois vértices e adicionando o valor 1 às *rows* dos vértices v1 e v2 e 0 às restantes formando, assim, uma nova aresta como *column*.



Acima está representado um exemplo de um grafo criado com quatro vértices. Abaixo encontra-se a sua matriz de incidência.

```
[1 0 0 0]
[0 1 1 0]
[0 1 0 1]
[1 0 1 1]
```

O método *getEdgesAdjacency()* cria, com base na matriz de incidência, um dicionário com as arestas como *keys* e uma lista de arestas adjacentes à *key* como *value.* Dicionário este que posteriormente irá ser usado para encontrar a solução do problema. Abaixo encontra-se o dicionário de adjacência do grafo exemplo.

{0: [2, 3], 1:[2, 3], 2:[0, 1, 3], 3:[0, 1, 2]}

Neste ponto analisamos a implementação dos algoritmos usados para a resolução do problema.

1. *Exhaustive search*
2. *Greedy search*

Exhaustive search

Para a implementação da pesquisa exaustiva é usado o método *findExhaustiveSolution()* da classe *Graph.* O algoritmo consiste em iterar o range do número de arestas do grafo para calcular todas as combinações possíveis (com tamanho entre 1 e o número de arestas) de conjuntos de arestas que constituem, assim, possíveis soluções. Iterando os conjuntos de arestas calculados verificamos as arestas adjacentes a cada uma pertencente ao conjunto, se as arestas do conjunto juntamente com as adjacentes formarem todo o domínio de arestas do grafo então encontramos a combinação que reflete o *dominating set* de arestas. O *MEDS* é o/um conjunto solução de arestas com menor tamanho.

Search with Greedy Heuristics

Para a implementação da pesquisa com *greedy heuristics* é usado o método *findGreedySolution()* da classe *Graph.* O algoritmo consiste, numa primeira fase, em ordenar o dicionário de adjacência por tamanho da lista de arestas adjacentes, isto é, ficará ordenado por ordem decrescente das arestas com maior número de arestas adjacentes. Com o dicionário ordenado adiciona-se à solução final a primeira aresta do dicionário e seguidamente elimina-se do próprio dicionário a aresta e suas arestas adjacentes. Repetindo estes passos anteriores tem-se uma solução final (*MEDS*) assim que o dicionário ficar vazio.

IV. RESULTS AND DISCUSSION

Neste ponto irá analisar-se e discutir-se os resultados obtidos das experiências realizadas na resolução do problema. Numa primeira instância, encontra-se abaixo a solução, isto é, o *MEDS* dos três grafos apresentados anteriormente(Grafo 1 <3 vértices>, Grafo 2 <5 vértices>, Grafo 3 <7 vértices>) com os dois algoritmos bem como o tempo gasto e o número de *basic operations* usadas em cada.

-$ python3 src/main.py 3

```
carvalho@LAPTOP-S01N1QNU:/mnt/c/Users/joaoc/OneDrive/Ambiente de Trab
Graph created with 3 vertices and 3 edges! Time elapsed: 0.0665 (ms)
Minimum edge dominating set (exhaustive algorithm): (0,)
Time elapsed: 0.0815 (ms) | Num basic ops 12
Minimum edge dominating set (greedy algorithm): [0]
Time elapsed: 0.0219 (ms) | Num basic ops 2
```

-$ python3 src/main.py 5

```
carvalho@LAPTOP-S01N1QNU:/mnt/c/Users/joaoc/OneDrive/Ambiente de Tra
Graph created with 5 vertices and 6 edges! Time elapsed: 3.6511 (ms)
Minimum edge dominating set (exhaustive algorithm): (1,)
Time elapsed: 0.1614 (ms) | Num basic ops 192
Minimum edge dominating set (greedy algorithm): [1]
Time elapsed: 0.011 (ms) | Num basic ops 5
```

-$ python3 src/main.py 7

```
carvalho@LAPTOP-S01N1QNU:/mnt/c/Users/joaoc/OneDrive/Ambiente de Traba
Graph created with 7 vertices and 15 edges! Time elapsed: 1.7316 (ms)
Minimum edge dominating set (exhaustive algorithm): (2, 7)
Time elapsed: 400.701 (ms) | Num basic ops 245760
Minimum edge dominating set (greedy algorithm): [8, 3]
Time elapsed: 0.0584 (ms) | Num basic ops 17
```

Para uma melhor visualização dos resultados obtidos os dados estão dispostos numa tabela que se pode consultar abaixo.

	#Vertices	#Edges	#Basic operations	Time (ms)
Exhaustive	2	1	1	0.1726
Greedy	2	1	1	0.0303
Exhaustive	3	2	12	0.1264
Greedy	3	2	2	0.0174
Exhaustive	5	8	1024	2.1849
Greedy	5	8	10	0.0999
Exhaustive	8	12	24576	19.552
Greedy	8	12	14	0.0966
Exhaustive	11	23	96468992	106681.5076
Greedy	11	23	25	0.536
Exhaustive	25	151	--	--
Greedy	25	151	237	0.5145
Exhaustive	200	10131	--	--
Greedy	200	10131	19714	14.3182
Exhaustive	200	63205	--	--
Greedy	500	63205	124095	2676.0232

Como podemos verificar os testes foram realizados iterativamente com um número crescente de vértices e consequentemente de arestas.

No caso da ***Exhaustive search***, como previsível, o tempo gasto assim como o número de *basic operations* cresce de forma exponencial(complexidade O(2^n)) à medida que os grafos vão ficando mais complexos pois o número de possibilidades combinadas aumenta da mesma forma. Neste caso para grafos com mais de 12/13 vértices e, principalmente com mais de 25/27 arestas, este tipo de pesquisa já se torna bastante demorosa e ineficiente.

Usando a expressão exponencial *"2^num_vertices"* podemos, mesmo que pouco precisamente, prever um possível valor de tempo gasto para encontrar uma solução. É pouco preciso pois o número de arestas é determinado aleatoriamente. Por exemplo um grafo com 20 vértices demoraria à volta de 1048576 ms para resolver o problema, cerca de 20 min. Para a previsão do número de *basic operations* podemos usar a expressão *"5^num_vertices"* como aproximação. No exemplo de 20 vértices o processo usaria 95367431640625 operações.

Pelo contrário, no algoritmo usando ***Greedy heuristics***, o tempo gasto e o número de *basic operations usadas* para a resolução parece crescer linearmente, pois apenas depende do dicionário de adjacência das arestas.

V. CONCLUSION

Por fim pode-se concluir que os resultados obtidos foram ao encontro das expectativas prévias. Algoritmos usando *Greedy heuristics* aumentam, de facto, consideravelmente, a eficiência de um algoritmo que realiza uma pesquisa apenas por *brute force.*

A implementação e posteriores testes refletem com alguma precisão essa diferença de complexidade e apresentam valores de tempo e *basic operations* razoáveis para o problema em questão.

REFERENCES

[1] https://www.hindawi.com/journals/isrn/2014/975812/

[2] https://core.ac.uk/download/pdf/82784436.pdf