Minimum Vertex Cover

João Pedro Pereira <106346>, pedro.jp@ua.pt

***Resumo* - O principal objetivo deste artigo é analisar o problema de cobertura mínima do vértice de um grafo. Neste artigo, foram analisados dois modelos de cobertura minima do vértice, onde foram ambos comparados para determinar o melhor e mais ótimo algoritmo.**

***Abstract* - The main objective of this article is to analyze the problem of minimum coverage of the vertex of a graph. In this article, two models of minimal vertex coverage were analyzed, where both were compared to determine the best and most optimal algorithm.**

I. INTRODUÇÃO

O objetivo é encontrar uma cobertura mínima de vértices para um dado grafo não direcionado G(V, E), com n vértices em arestas.
Uma cobertura de vértices de G é um conjunto C de vértices, de modo que cada aresta de G incide em, pelo menos, um vértice em C.
Uma cobertura de vértice mínima é uma cobertura de vértices do menor tamanho possível.
Para a resolução do problema, foram analisados dois algoritmos, um algoritmo ganancioso e um algoritmo de aproximação, onde em análise de ambos é será determinado qual deles retorna o melhor resultado.

II. DESCRIÇÃO DO PROBLEMA

A cobertura mínima do vértice é um conjunto de vértices que inclui pelo menos um vértice do grafo.
Dado um grafo não direcionado, o problema é encontrar a cobertura de vértices de tamanho mínimo.
Na ciência da computação, o problema de encontrar uma cobertura mínima de vértices é um problema de ótimização.

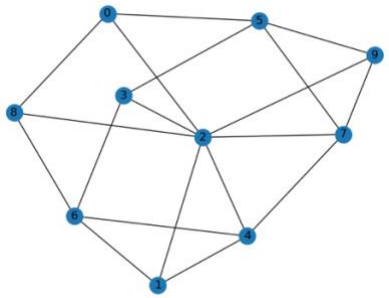

Figura 1- Grafo de exemplo

Na figura 1, poderemos observar um grafo de exemplo onde o objetivo seria determinar o conjunto pertencente à cobertura mínima dos vértices. Onde poderá ter várias soluções ótimas. E quanto o gráfico crescer, mais soluções irá ter.

III. DESCRIÇÃO DAS SOLUÇÕES

Para procurar as melhores soluções ao problema, foram analisados dois algoritmos, um algoritmo ganancioso e outro de aproximação.

Algoritmo ganancioso ou greedy algorithm é um paradigma algorítmico que produz uma solução por passos, vai escolher sempre o próximo passo como se fosse o último, ou seja, o passo que irá seguir seria o melhor no momento. Seguindo o paradigma do o próximo passo sendo o melhor e não analisando os restantes, pode dar a soluções não ótimas. É considerado um NP-complete, *Non-deterministic Turing Machine in Polynomial time.*

Algoritmo de aproximação ou Approximation Algorithm, é um algoritmo para lidar com o NP-complete. Tendo objetivo de chegar o mais perto possível da solução ideal no tempo polinomial, mas não garante a solução ótima.

A. Algoritmo ganancioso ou greedy algorithm

Inicialmente, com os vérices do grafo e as suas arestas criadas, foi ciado um dicionário que funciona como uma tabla de adjacência. As chaves do dicionário corresponde ao vértice referente, onde nos valores tem uma lista com os vértices vizinhos, o grau qu corresponde ao numero dos vértices dos vizinhos e um valor booleano que identifica se já foi visitado ou se não. Na figura 2, é possível ver um pequeno exemplo deste mesmo dicionário.

```
{
'0': {'neighbors': ['1', '2', '5', '6', '9'],
        'degree': 5,
        'visited': False},
'1': {'neighbors': ['4', '5', '7', '9', '0'],
        'degree': 5,
        'visited': False}
}
```

Figura 2 - Dicionário de adjacência

Para este algoritmo foi utilizado o seguinte pseudocódigo.

```
Greedy algorithm
    BEGIN
    Resultado_final <- []
    Adjacencia <- Dicionário de adjacencia
    Enquanto existir vertices não visitados:
        Termo <- termo com grau maior e não visitado
        Marca o Termo da adjacência como visitado
        Adiciona o termo ao Resultado_final
        Lista_para_remover <- [ vizinhos do termo ]
        Para x ∈ Lista_para_remover:
            Remover o termo dos vizinhos do x
            Atualizar o grau no x
            Se o grau do x = 0:
                Colocar x como visitado
    Retornar Resultado_final
    END Greedy algorithm
```

B. Algoritmo de aproximação ou Approximation Algorithm

No algoritmo de aproximação, foi usado um dicionário com conjunto de arestas, a figura 3, demonstra um pequeno usado, onde a chave será o nome da aresta e o valor do item corresponde aos vértices a que essa aresta liga.

```
{'A0': ('0', '1'),
 'A1': ('0', '3')}
```

Figura 3 - Dicionário inicial para algoritmo de aproximação

Neste algoritmo, usou-se o dicionário E, correspondente à figura 3, e C que corresponde ao conjunto final.
Enquanto o dicionário E for diferente de um conjunto vazio, é escolhido aleatoriamente qualquer aresta que esteja no dicionário de E. É unido o conjunto da aresta ao conjunto C e é apagado todas as arestas de E que {u,v} da aresta escolhida sejam incidentes em E. Quando E for um conhunto vazio, a função terminará com o retorno do conjunto C.
A seguir demonstra o pseudocódigo

```
Approximation algorithm
        BEGIN
        C <- []
        E <-  dicionario das arestas
        Enquanto existe E:
            Aresta_escolhida <- Escolher {u,v} ∈ E
            C <- C + Aresta_escolhida
            Apagar de E arestas incidentes na
            Aresta_escolhida
        END Approximation algorithm
```

IV. Modos de utilização

O programa desenvolvido é composto por dois modos de iniciação:

- Same
- Increase

Em qualquer um dos modos, após a geração dos grafos e análise de ambos os algoritmos, são gerados 3 ficheiros de texto. Ficheiro *aprox_solutions.txt* que corresponde ao conjunto de resultados do algoritmo de aproximação de cada grafo. O ficheiro *greedy_solutions.txt* corresponde aos resultados do algoritmo ganancioso e o ficheiro *graph.txt* que corresponde à lista de arestas de cada grafo gerado. Também são geradas imagens de cada grafo, que ficarão armazenadas na pasta *graphs*.

A. Modo Same

Este modo trabalha com o mesmo número de vértices, onde na execução do programa define-se o valor de número de vértices e a quantidade de grafos a serem gerados. Neste caso, ao introduzir a quantidade a serem gerados, esse valor será triplicado, pois por cada grafo, serão criados grafos, onde cada um vai ter 3 probabilidades de arestas do seguinte conjunto $\{0.25,\ 0.50,\ 0.75\}$. Após a análise de imagem

que contém um gráfico para análise de comparação do tamanho da solução de cada algoritmo para cada grafo.

B. Modo Increase

Neste modo, é requisitado dois valores, um número mínimo e um número máximo, significa que vai ser criado um ciclo desde o mínimo até ao máximo, e por cada valor será criado um grafo com valor correspondendo ao número de vértices. Após a análise de ambos algoritmos, serão geradas duas imagens, *times_graphics.png* que vai mostrar os tempos de execução de cada algoritmo para cada grafo e *solutions_graphics.png* que mostra um gráfico com o tamanho da solução de cada algoritmo para cada grafo.

V. Análise de resultados

Aqui vão ser analisados os resultados por cada modo de execução.

A. Modo Same

Na figura 4, é possível analisar o tamanho das soluções para cada grafo, neste caso, foram gerados 10 grafos com 10 vértices cada.

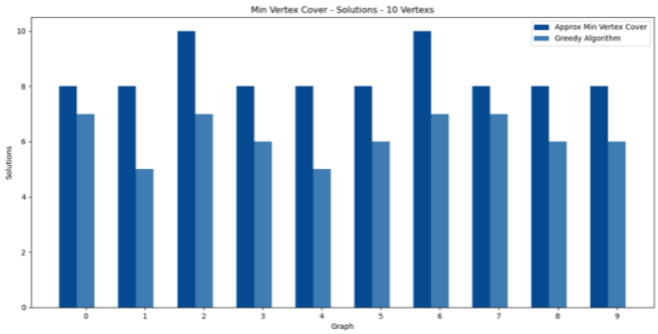

Figura 4 - Tamanho das soluções (Modo Same)

Conseegu-se verificar que para determinar a cobertura mínima de um vértice o Algoritmo ganancioso é considerado o mais ótimo devido ao tamanho da solução ser inferior ao Algoritmo de Aproximação.

B. Modo Increase

Já mencionado anteriormente, este modo gera duas imagens *times_graphics.png e solutions_graphics.png*.
Na figura 5, é possível observa o tamanho de cada solução de cada algoritmo de cada gráfico. Podemos ver que o tamanho da solução aumenta conforme o número de vértices, pois quantos mais vértices, maior o número de arestas.

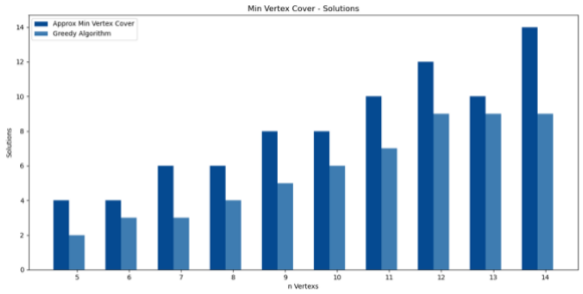

Figura 5 - Tamanho da solução (Modo increase)

Já na figura 6, podemos ver que quanto maior o tamanho do grafo, o tempo de execução normalmente também aumenta. Quando existe uma diminuição de tempo para um grafo com um número de vértices superior ao anterior, significa que o grafo inferior contém mais arestas que o grafo superior.

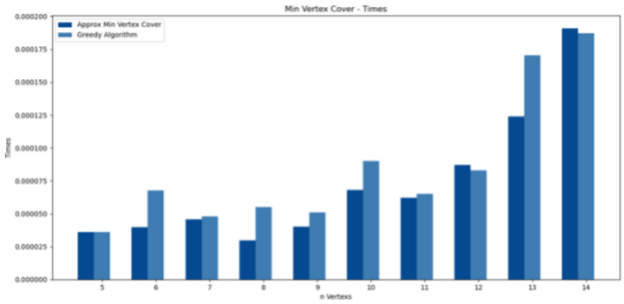

Figura 6 - Tempos de execução (Modo Increase)

VI. Resultados

Para cada modo (same ou increase), o programa vai retornar o resultado por grafo, é possível observar um exemplo de resultado na figura 7.

```
**************************************************
Graph with 6 vertices

Edges:
[('0', '2'), ('0', '4'), ('1', '2'), ('1', '3'), ('2',
'4'), ('3', '4'), ('3', '5')]

Greedy Algorithm:
['2', '3', '0']
Execution Time: 6.604194641113281e-05

Aprox Algorithm:
['2', '4', '5', '3']
Execution Time: 3.910064697265625e-05

**************************************************
```

Figura 7 - Resultado por grafo

Neste exemplo da figura 7, temos um grafo com 6 vértices, nele mostra a lista de vértices.
Onde na figura 8, é possível observar o grafo em questão.

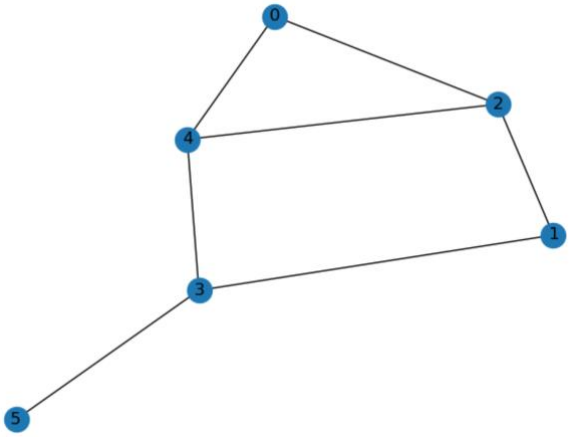

Figura 8 - Grafo do exemplo

Para análise de tempos, foi usado o modo same, onde foi colocado a quantidade do grafo e apenas para gerar um único grafo *(--repeat 1*).

Vértices	Arestas	Tempo ms (Greedy Algorithm)	Tempo ms (Approximation Algorithm)
2	1	0.00001	0.00001
5	5	0.00001	0.00001
10	26	0.00001	0.00001
25	143	0.00038	0.00046
50	608	0.00111	0.00273
100	2479	0.00519	0.02035
1000	249646	2.12340	29.6667
5000	6251470	294.077	5342.343

VII. Conclusões

Em resumo, após verificações manuais em alguns grafos e comparação com os resultados obtidos, verifica-se que nenhum de ambos os algoritmos são ótimos, por vezes o algoritmo ganancioso consegue chegar ao resultado ótimo, mas nem sempre consegue. Verifica-se também que os tempos de execução aumentam, quando aumentamos o grafo.

REFERENCES

[1] JINGRONG CHEN*, LEI KOU, XIAOCHUAN CUI , "AN APPROXIMATION ALGORITHM FOR THE MINIMUM VERTEX COVER PROBLEM", SCHOOL OF MATHEMATICS AND PHYSICS, LANZHOU JIAOTONG UNIVERSITY, LANZHOU 730070, P. R. CHINA

[2] GEEKSFORGEEKS: APPROXIMATION ALGORITHM, 22 JUN, 2021

[3]GEEKSFORGEEKS: GREEDY ALGORITHM, 31 JUL, 2021