Minimum Vértex Cover

António Ramos, ajframos@ua.pt, 101193, MEI

***Resumo* – Este trabalho apresenta uma análise de esforço computacional de algoritmos de grafos, nomeadamente o Minimum Vértex Cover.**

Para o estudo do esforço computacional foram feitos testes sequencias de grafos a partir de 10 vertices. Por último é estimado o tempo de execução para grafos maiores.

***Abstract* - This work presents a computational effort analysis of graph algorithms, namely the Minimum Vertex Cover.**

To study the computational effort, sequential graph tests were carried out from 10 vertices. Finally, the execution time for larger graphs is estimated.

I. INTRODUCTION

A graph is a pair *G = (V, E)*, where V is vertices/nodes/pairs and E is edges/lines/links [2].

There are two classifications for graphs direct and undirect. A graph is undirect when the edges doesn't have information regarding two nodes. A graph is direct when the edges have orientation and information connecting two nodes.

Exists to many problems associated with graphs, but in this assignment, it's focused on Minimum Vertex Cover.

A vertex cover of a graph is a set of vertices that includes at least one endpoint of every edge of the graph.

The minimum vertex cover is the optimization problem of finding the smallest vertex cover in a graph. It's a NP-hard problem, that can't be solve using polynomial-time algorithm if P ≠ NP. The figure bellow shows example of applying the minimum vertex cover algorithm [3]

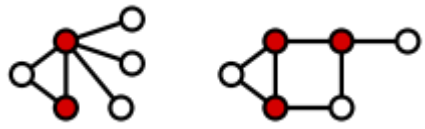

Figure 1 - Example of Minimum Vertex Cover

The weighted of minimum vertex cover can be

$$\begin{aligned} &\text{minimize } \textstyle\sum_{v \in V} c(v) x_v && \text{(minimize the total cost)} \\ &\text{subject to } x_u + x_v \geq 1 \text{ for all } \{u, v\} \in E && \text{(cover every edge of the graph),} \\ &\qquad x_v \in \{0, 1\} \text{ for all } v \in V. && \text{(every vertex is either in the vertex cover or not)} \end{aligned}$$

Figure 2 - Expression of Minimum Vertex Cover

formulated as the following formula. [3]

To solve the problem, it was implemented an exhaustive search algorithm, this tests all possibilities to find a solution to a problem.

II. ALGORITHM ANALYSIS

In this chapter is it's showed how the algorithm was analysed and what is the approach chosen.

A. Graph Generation

The graphs are generated based on the choice of random number between 1, 9 (x, y). The min vertices are ten, due to the implementation of percentage of edges [25, 50, 75]. In this problem doesn't make sense to have 0 and 100, coz exists a need to check the vertices that are connected to all.

After the generation the graph is initialled by full filling the number of points of vertices until the percentage of edges is reach, with are randomly determinate in the set of points.

The graphs are represented as adjacent matrix, due to simplicity, existence of two correlated lists. The value 1 in adjacent matrix means that the vertex x is connect to vertex y, and 0 not connect.

The graph and the adjacent are saved in different files. The graph file is called "output_graph.txt" and adjacent matrix is "adj_matrix.txt".

B. Determination of Minimum Vertex Cover

The determination of Minimum Vertex Cover is based on the use of product method from itertools from python. The product method will receive two sets of all ordered pairs (a, b) where a belongs to A and b belong to B. [4]

All possible products are generated by looping through the points given 0 and 1.

Then it's occurred the verification all the points generated previously if are a cover/linked to all the possible vertices.

To get the set of the minimum vertex, it's used the greedy algorithm to explore all vertices.

The minimum vertex is found using count variable that sums all the vertices from a graph and using min functionality from python to compare with the number of

the vertices. If it's not found the minimum vertex it's assumed that are the number of vertices on graph.

C. Algorithm Complexity Analysis

The algorithm starts to make a matrix of the product of 01 of the number of vertices, and then loop through the same matrix and the adjacent matrix to find the vertex cover and the calculate the min of the operations.
The exhaustive search of algorithm can solve a minimum vertex cover in $2^k n^O$
The best algorithm can achieve the solution in time $O(1.2738^k + (k \cdot n))$ where k is the number of vertices.

III. Tests Sequence Analysis

To test the algorithm developed, was implemented with successive larges instances of min vertices that is 10. In each execution of graph was recorded to a file the time of execution of the creation of the graph that are printed to a console, the execution of the algorithm, the min vertex cover and the number of operations.

Tests were performed to graphs with number of vertices between 10 and 24 (it's assumed that the number of vertices is 23). For each graph number tests were performed with graph with percentage of vertices 25, 50, 75. The max can be the choice of the user, in this tests analysis the arguments are 10 – 24, it's the max number that my computer can handle, before getting the memory error.

A. Execution Time Analysis

The analysis regarding the execution time is given in seconds is in the bellow image.

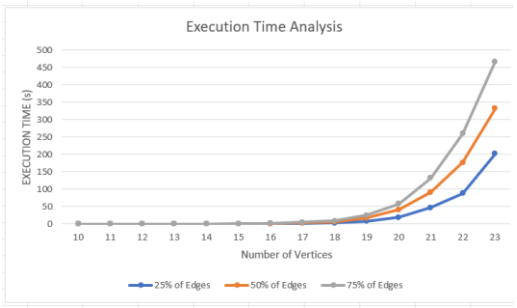

Figure 3 - Execution Time

Analysing the figure, it's possible to see that around the nineteen vertices the graph starts growth exponentially.
The rate growth on every possible percentage it's proportional equal.
In conclusion if the number of vertices increase the execution time will be increase exponentially.

B. Number of Operations Analysis

The results of the number of operations in the algorithm minimum vertex cover to verify if a vertex is a cover is in the figure 4.

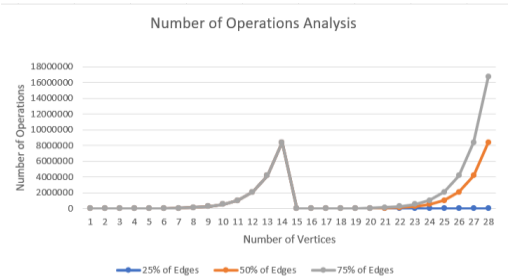

Figura 4 - Number of Operations

By analysing the figure it's possible to see that both of percentage of edges are equal until 22 vertices, but then the 75% growth more than the previous ones.

C. Minimum Vertex Cover Analysis

The results of the Minimum Vertex Cover are in the figure 5.

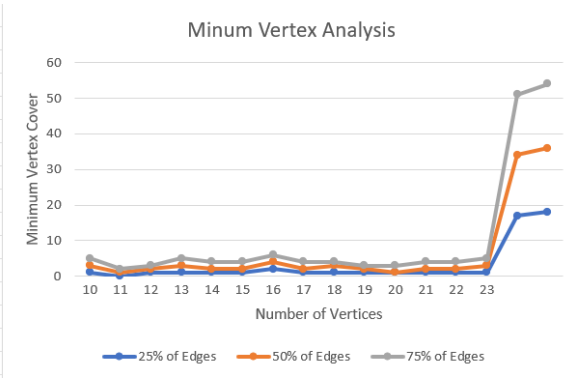

Figure 5 - Minimum Vertex Cover

Analysing the image, it's show that the number of minimum vertex cover doesn't change due to the vertices.

D. Worst and Best Case

The worst and best case based on analysis in the section A, the best case is 25% of edges and worst case is 75%.

IV. EXCEPT TIME ON LARGEST VERTICES

In order to analyse the time to largest vertices, it's used the function by excel growth, due to the computer takes to long to find a solution for bigger vertices. The interval of vertices chosen as 50, 250, 500, 900, note that 1000 the excel can't calculate.

Analysing the figure 6 we can say that the best time was 75 % of edges and the worst 25%.

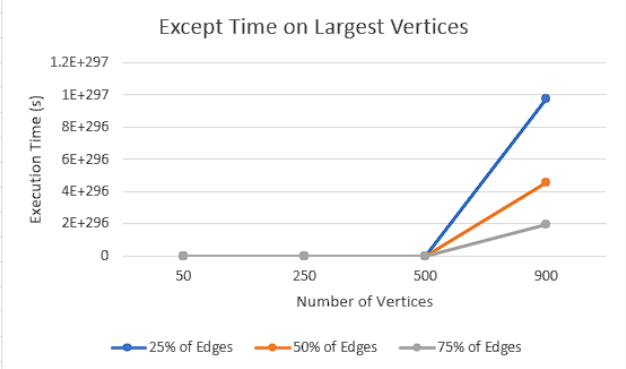

Figure 6 - Except Time

V. CONCLUSION

Through the analysis of various types of the graphs previously showed, it is concluded that the percentage of edges has an impact on the execution on time.

Due to the complexity of using the exhaustive search algorithm developed, it is not possible to run on the personal computer for bigger graphs

REFERENCES

[1] http://dgtlview.blogspot.com/2015/07/vertex-cover-python-implementation.html

[2] https://www.geeksforgeeks.org/vertex-cover-problem-set-1-introduction-approximate-algorithm-2/

[3] https://en.wikipedia.org/wiki/Vertex_cover

[4] https://www.geeksforgeeks.org/python-itertools-product/