Minimum Vértex Cover

António Ramos, [ajframos@ua.pt](mailto:ajframos@ua.pt), 101193, MEI

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

*Abstract* - Must be in English.

# 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]



Fig. 1 - Example of Minimum Vertex Cover

The weighted of minimum vertex cover can be formulated as the following formula. [3]

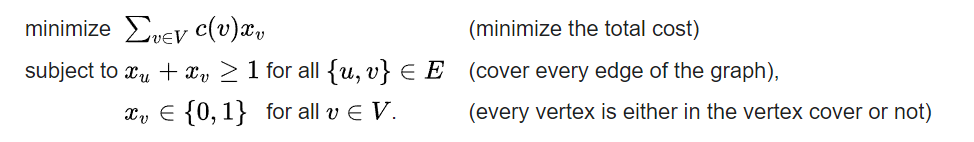


Fig. 2 - Expression of Minimum Vertex Cover

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

# 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*

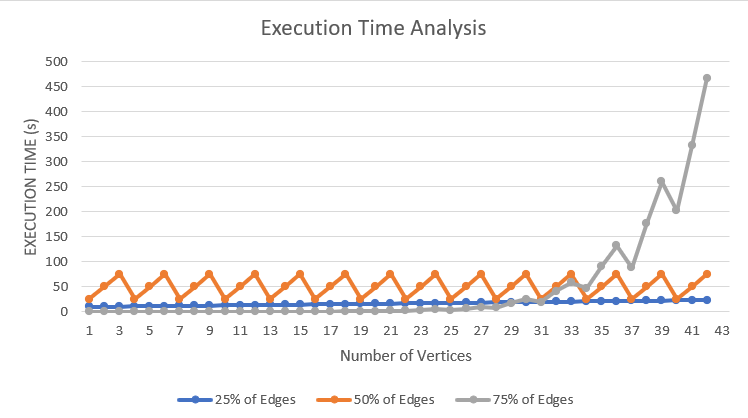
# Tests Sequence Analysis

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 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 – 23, it’s the max number that my computer can handle, before getting the memory error.

## Execution Time Analysis

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



# 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/>