Fontys Hogescholen

# Algorithms 2

Assignment1

Aleksandar Georgiev, Ivaylo Ivanov
Supervisor: Suzana Andova

## Table of Contents

# 1. Explanation

## Main code (GUI)

We decided to use a python library named PySimpleGUI for the GUI of the assignment.

To be able to run the application, please run:

*pip3 install pysimplegui*
*sudo apt-get install python3-tk*

First we start with the GUI generation. We declare all the components that we are going to use and the actual window of the application.

After that we have an event loop with all the event of the application :

- We have the *generate* event which is activated if the user clicks on the generate button. This event is responsible for the adjacency matrix, dot file generation and the visualization of the graph itself.
- Next we have the *connect* event which is responsible for converting the disconnected graph into a connected one and then display the new graph.
- The *Vertex cover* button takes an int from the vertex cover text box and attempts to make a vertex cover with the said number of vertices. If that is possible, it highlights the vertices in the cover red, if it is not, it displays an error message in red.

## Connect graph code:

We implemented a DFS algorithm starting from vertex 0. after we have traversed all the nodes that are connected to 0 we check if the number of visited vertices is the same as the number of all the vertices of the graph and if that is not the case we locate a vertex which is not connected to the traversed graph the connect that vertex to the last visited vertex. Then we continue with the DFS and check again if the number of visited nodes is the same as the number of vertices in the graph.

## Generate dot file code:

This function is really straightforward: we go through the adjacency matrix and generate the dot file base on it. We write everything as a string and then we use the *file.write()* to put the generated string into a file.
Generating a dot file for a graph where the program is asked to create a vertex cover

(when pressing the Vertex cover button) is done in a similar fashion, however the function takes a cover as an argument and highlights the nodes that are contained in the cover in red.

## Kernelization:

As it is stated in the *Labmanual* this function finds all the isolated, pendant and tops vertices. To find all the tops vertices we use the value from the text box *k* and then color them in the graph where every isolated vertex is red, every pendant vertex is green and every tops vertex is blue.

Moreover we added two more buttons - add pendant, and add tops using k:

- Add pendant - Get a non pendant vertex and make it a pendant one by removing connection to that vertex.
- Add tops using k - Get a non tops vertex and make it a tops one by connecting the vertex to more vertices.

## 2. Testing

We decided to manually test our solution so we did an extensive testing on our solution with different numbers of vertices and probability for the edges. We covered all the features of the program and tested every edge case.

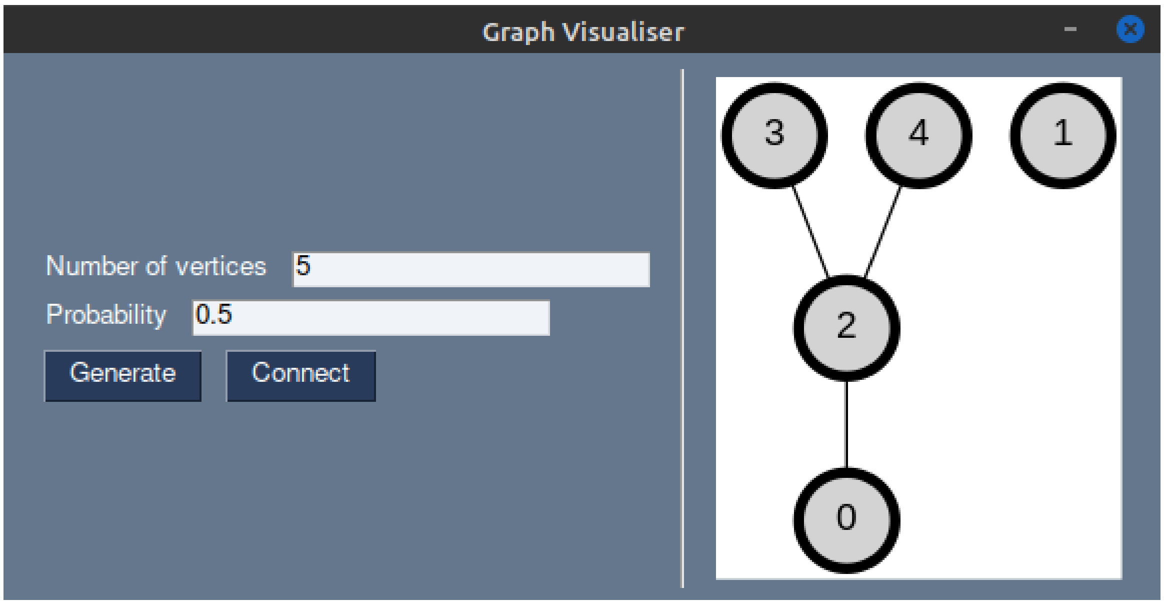
### 2.1. Performance

The performance of the bruteforce vertex cover depends on multiple factors - the size of the graph in which we need to find a vertex cover, the size of the vertex cover and the number of edges in the graph. The larger the graph and the more connected it is (more edges) - the longer it takes to find a cover or the lack thereof. The size of the cover we need to find is also crucial. If n is the number of nodes in the graph and k is the cover that we need to find, then the closer k is to n/2 the longer it takes. Since the number of elements that have to be checked is n choose k, which is calculated $\frac{n!}{k!(n-k)!}$, finding the cover is exponential, however it is extremely slow for covers of size closer to n/2.

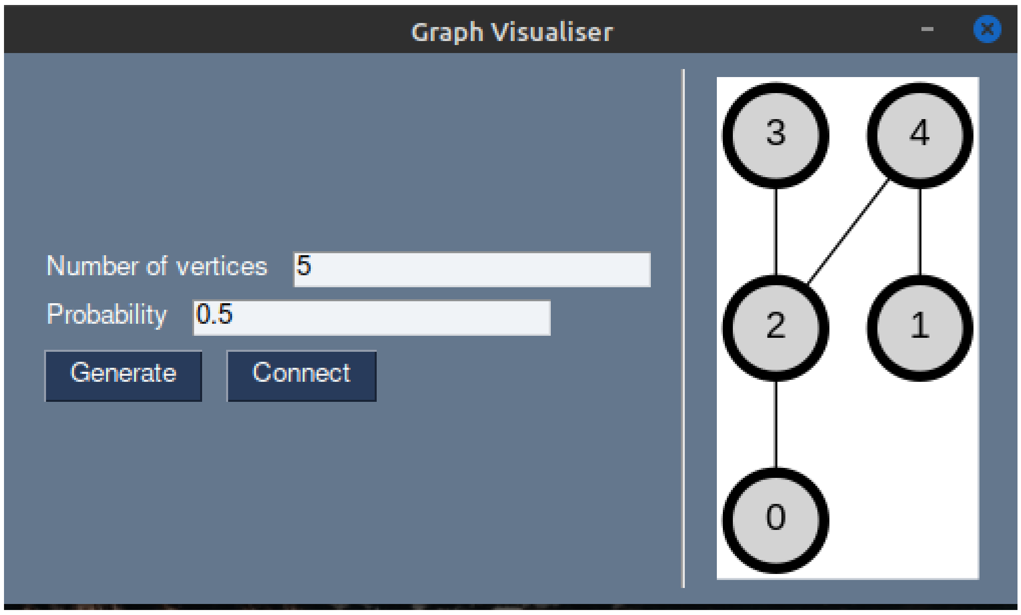
Our program is able to handle a cover of 10 nodes out of a 20-node graph in a reasonable amount of time (a few seconds), and for anything larger than that it freezes. I can do 11 nodes out of a 22-node graph or even 12 nodes out of a 24-node graph if the graph is less connected (lower probability - less edges).

## 3. Result

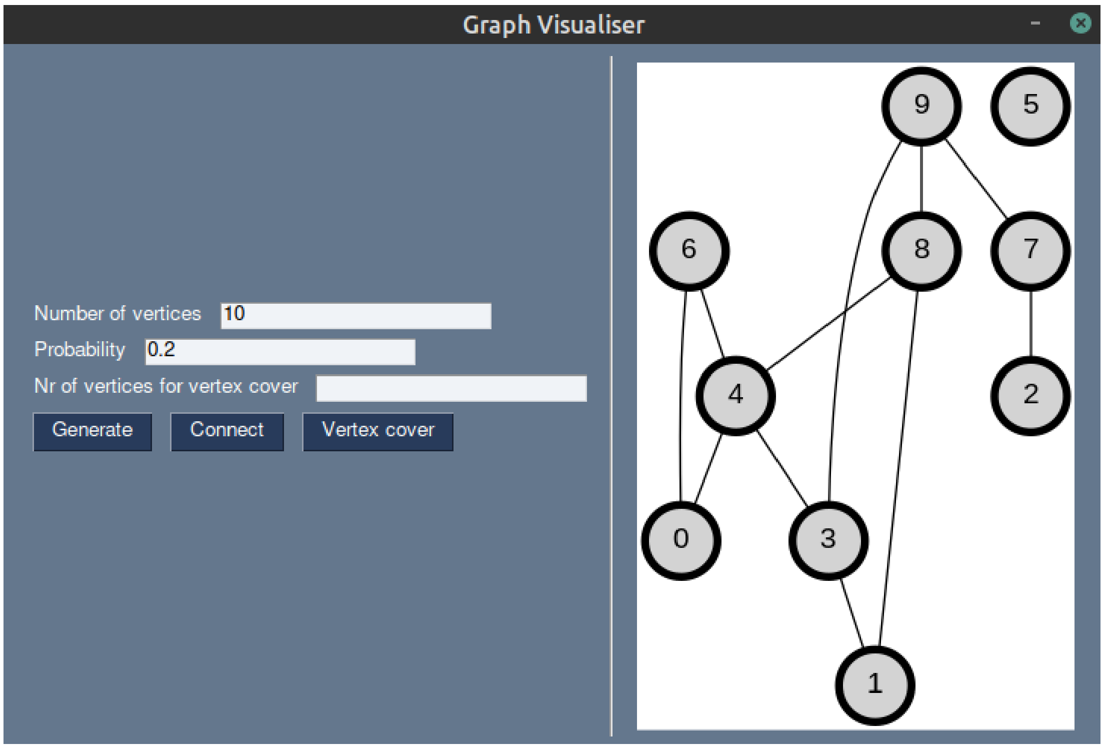
before clicking *Connect*



after *Connect*

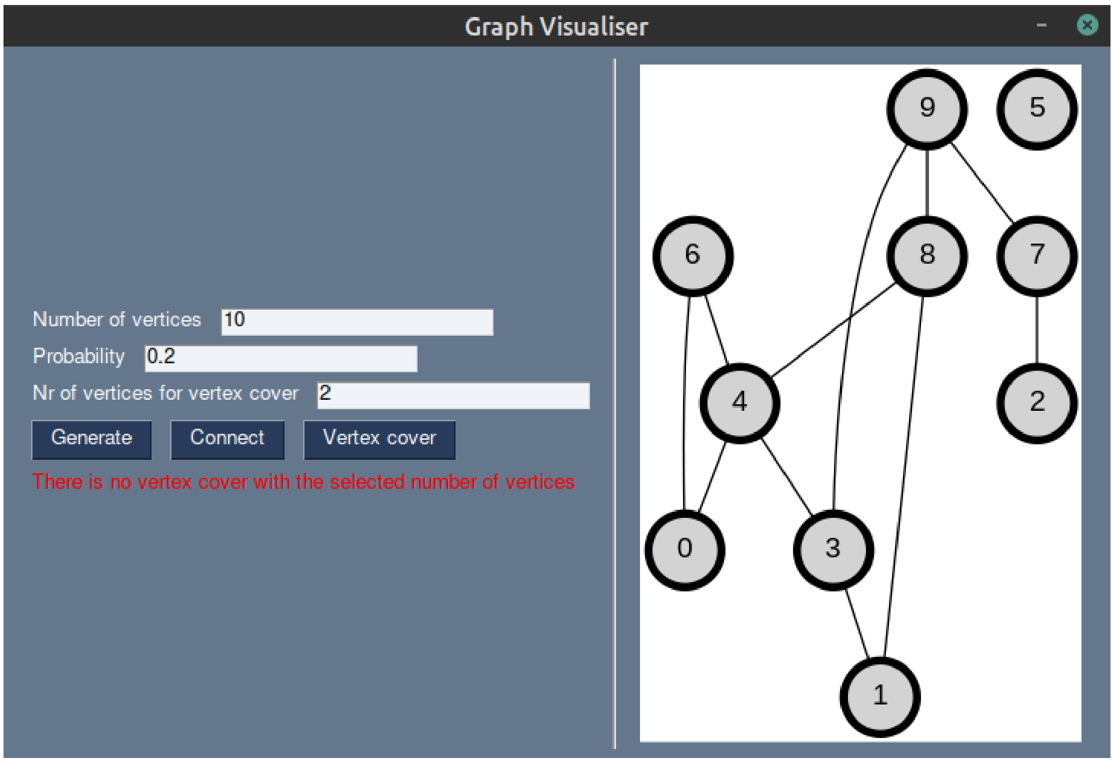


This is a graph with 10 vertices.



Number of vertices 10
Probability 0.2
Nr of vertices for vertex cover

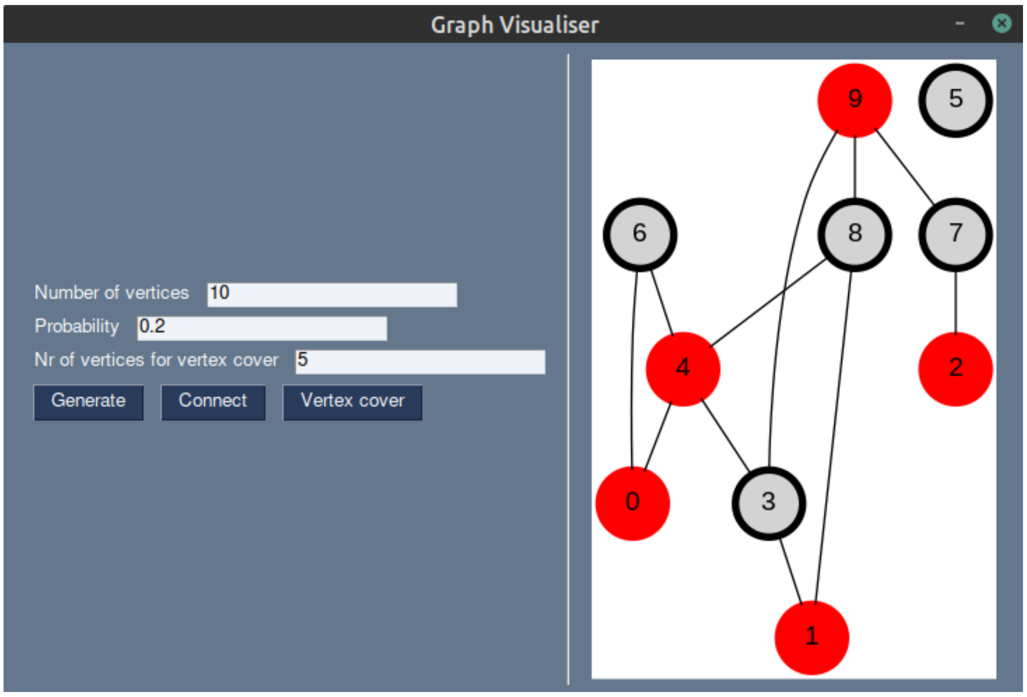
We then ask the program to make a vertex cover with 2 vertices, which is impossible.



Number of vertices 10
Probability 0.2
Nr of vertices for vertex cover 2

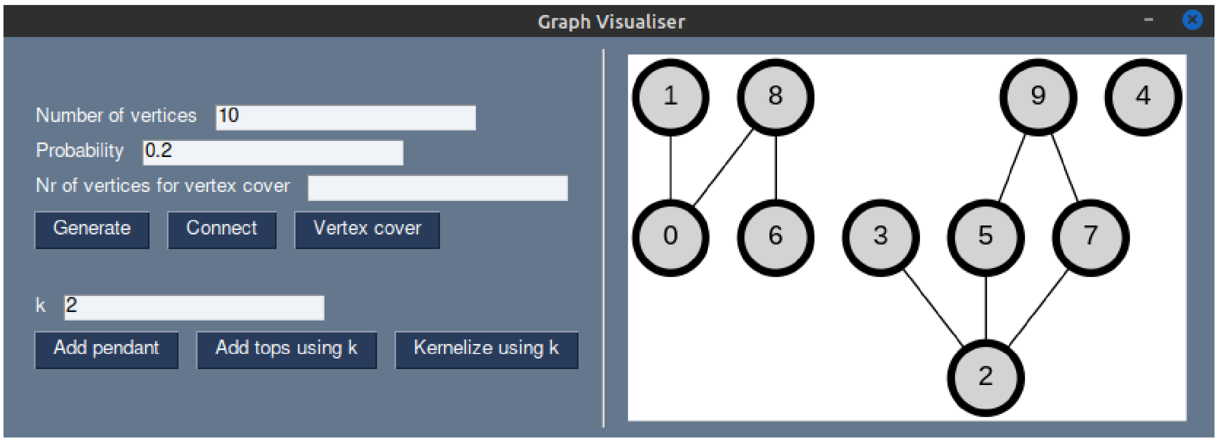
There is no vertex cover with the selected number of vertices

However, with 5 vertices it is, and it shows us the valid cover.



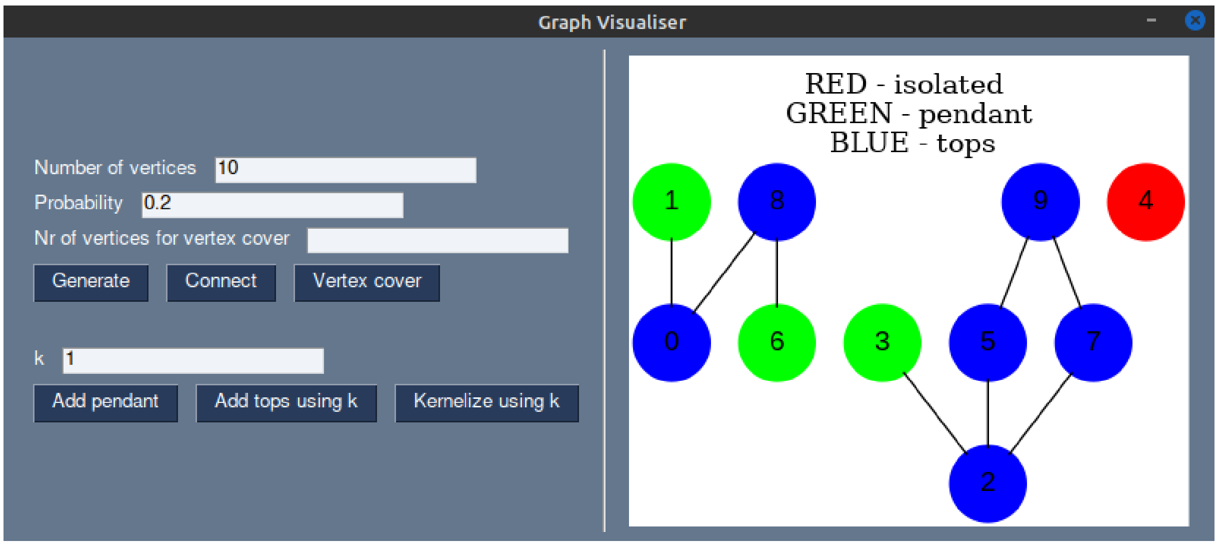
Number of vertices 10
Probability 0.2
Nr of vertices for vertex cover 5

Graph with 10 vertices.



Number of vertices 10
Probability 0.2
Nr of vertices for vertex cover
k 2

After clicking “kernelize using k”.


Graph Visualiser

Number of vertices 10
Probability 0.2
Nr of vertices for vertex cover
k 1

There is a legend at the top and the colored graph below.