McMaster University

Department of Computing and Science

COMPSCI 2XC3
Lab-4 Report

Suhaas Parcha (Mac-ID : Parchas)
Yu Chang (Mac-ID : changy72)
Yuvraj Singh Sandhu (Mac-ID : sandhy1)
Group 20-L01

Professor: Dr. Vincent Maccio

A report on various operations performs related to graphs including traversing algorithms, Minimum vertex covers and Maximum independant set

February 27, 2023

Contents

List of plots

Abstract

This lab report contains various implementations of functionalities of both DFS and BFS.We also conduct experiments to gain a deeper understanding on the properties of graph algorithms, and the corresponding probability of properties holding. Beside that, we also implemented three different approximation algorithms for the vertex cover problem. Also, we implemented the function of counting the MIS for a graph and discovered the relationships between maximum independent set (MIS) and minimum vertex covert (MVC).

Part - 1

0.1 Experiment 1

Description:

In this experiment, we have studied the relationship between the probability that a randomly generated graph contains a cycle and the number of edges for a fixed number of nodes. For each number of edges, we generated a certain number of random graphs and then calculated the proportion of graphs which contains at least 1 cycle.

Procedure:

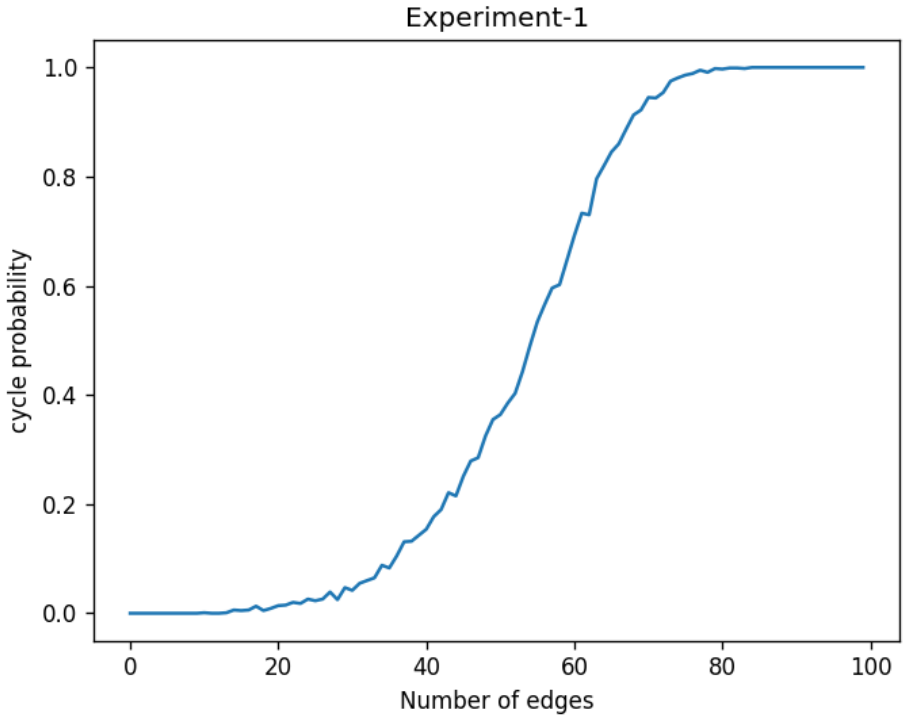

Figure 1: Experiment 1

The graph above shows the probability of a cycle in a graph with 100 nodes and varying numbers of edges which ranges from 0 to 100. We generated 1000 graphs for each number of edges. The randomly generated graphs neither have self-loop edges nor multiples of the same edge.

Conclusion:

As we increase the number of edges, the number of nodes that are connected to each other increases and therefore, the probability that a graph contains a cycle also increases. From 0 to 20, the probability is almost 0. After that, it increases at an exponential rate till 50. From 50 to 70, the probability increases at a linear rate. After that, it reaches 1 at a logarithmic rate till 80. After 80, the probability is constant at 1.0.

0.2 Experiment 2

Description:

In this experiment, we studied the probability that a graph is connected. We analyzed the relationship between the probability that a randomly generated graph is connected and the number of edges for a fixed number of nodes. For each number of edges, we generated a certain number of random graphs and then calculated the proportion of graphs which is connected.

Procedure:

The following graph shows the probability that a randomly generated graph is connected. The graph has 60 nodes and varying numbers of edges which ranges from 0 to 300.

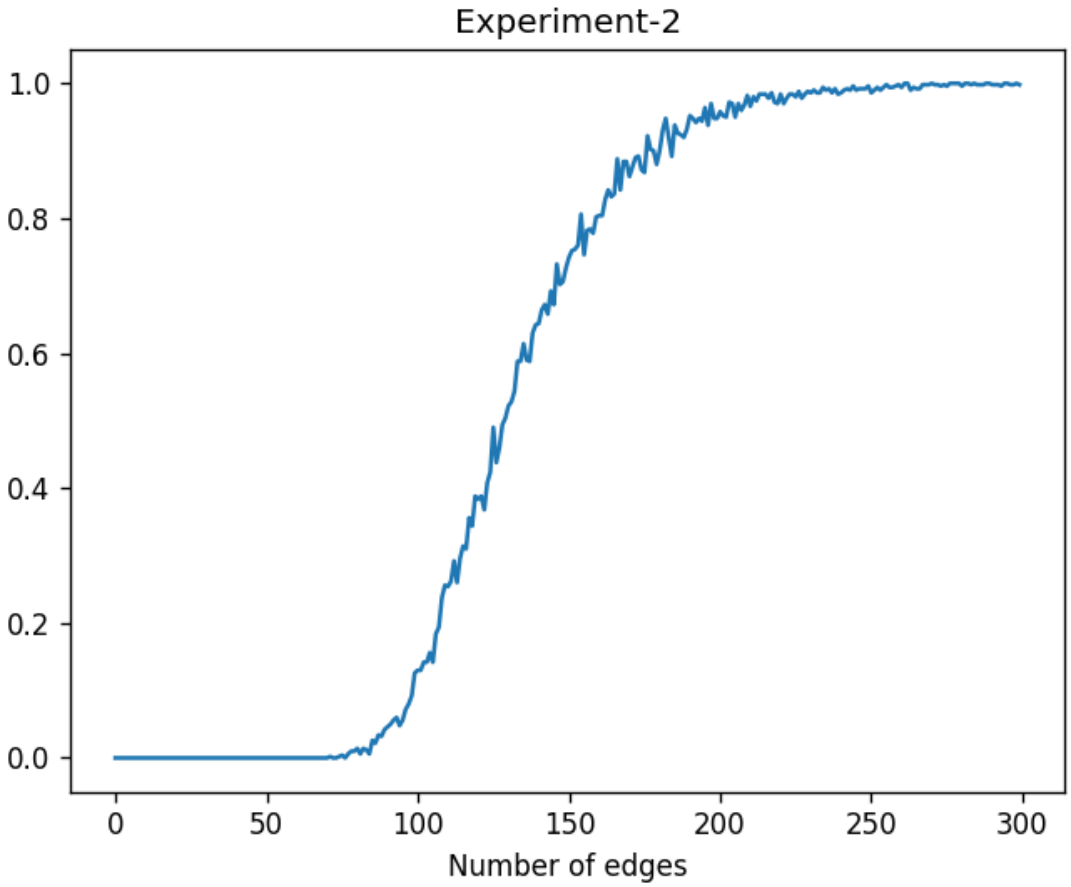

Figure 2: Experiment 2

We generated 500 graphs for each number of edges from 0 to 300 to minimize errors. The randomly generated graphs neither have self-loop edges nor multiples of the same edge.

Conclusion:

After analyzing the graph, we can say that from edges 0 to 70(approx), the probability that there exists a path between any two nodes in a randomly generated graph is almost 0. From 70 to 120, the probability starts to rise at an exponential rate. After that, the probability increases linearly till 160. From 160 to 200, the probability increases at a logarithmic rate and reaches 1. Finally, after 200 edges we can say that there is a very high probability that all nodes in a randomly generated graph of 60 nodes will be connected to each other as the probability reaches 1 and becomes constant in the graph.

Part - 2

0.3 Vertex Cover Problem

Description:

We were given a function "MVC" which returned the Minimum vertex cover of a given graph. Given 3 approximate vertex cover algorithms, we had to compare them and find the most consistent algorithm by calculating and plotting the expected performance of the algorithms.

Procedure:

The MVC function uses a brute force type approach by iterating through the power set of a given graph. This causes the function to not be efficient for graphs with nodes > 26. Hence the following ideas were tested to find the most consistent algorithm for graphs of larger sizes.

- Approx 1 : Finding the largest degree vertex and then appending it to a temporary "cover" variable as well as, removing all incident edges corresponding to the vertex. If "cover" is a vertex cover then we stop, else, we repeated the process until a vertex cover is reached.
- Approx 2 : By randomly selecting a vertex and then adding this vertex to a "cover" variable, then we append this variable to the cover, then repeat until a vertex cover is reached.
- Approx 3 : Instead of taking a single vertex like Approx 2 , we select and edge and append both vertexes corresponding to the edge to cover . Let these 2 vertexes be named u and v , then we remove

incident edges of either u or v. We repeat this process until we reach a vertex cover.

For the first experiment, we generate a random graph with varying edges and 8 nodes, we take the sum of the length of the list returned by the approximation algorithms and divide it by the total sum length of MVC for the same graph, the result of this can be represented as the expected performance of the algorithm. Expected perfomance is plotted against number of edges.

- appr1 = approx1/mvc
- appr2 = approx2/mvc
- appr3 = approx3/mvc

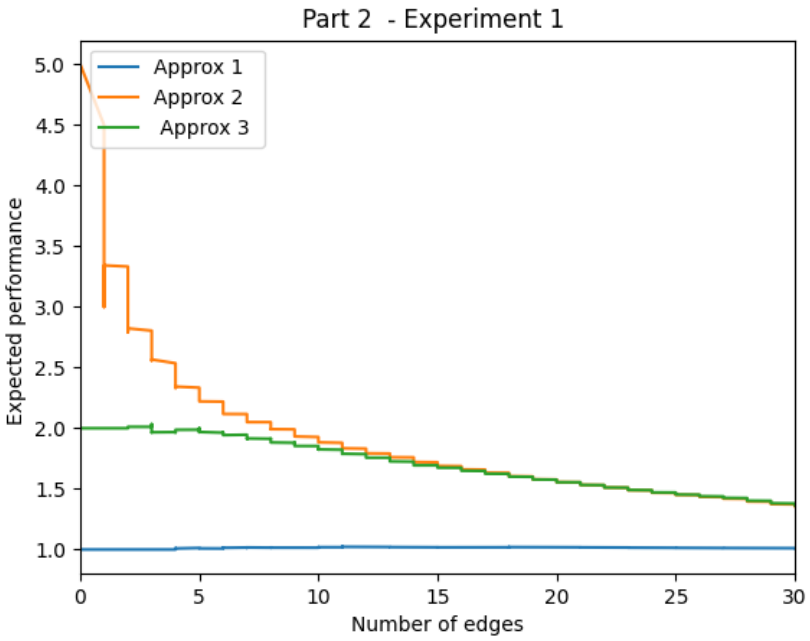

Figure 3: Expected Performance potential graph

Conclusion & Analysis:

From the graph above, it is clear that approx1 has the most consistent performance when compared to MVC, it is closer to 1.0 in expected performance when the number of edges are from 0 to 30. We can also observe that Approx 3 is the worst when the number of edges < 5 , but is

similar to approx2's performance when the nodes are > 10. The reason that both approx2 and approx3 perform worse can be attributed to the randomness factor, since we have no way to distinguish if a node belongs to the minimum vertex cover or not, the probability of finding a node which doesn't belong to the Minimum vertex cover increases. This can be avoided by grouping the known probable nodes in the minimum vertex cover and the ones that don't belong, doing so decreases the chance the probability that a "bad" node is picked.

Experiment 2 Question - " Is there a relationship between how good we would expect an approximation to be and the number of edges in a graph? In general, does the approximation get better/worse as the number of edges increases/decreases? "

We can answer this question by looking at the previous graph, as the number of edges increase approx2 & approx3 close the performance gap when compared to approx1.

We can confirm this with another experiment, by increasing the number of nodes to 17, we know that the maximum number of edges without self loops is n*(n-1)/2 , with this information we know that the maximum no of edges would be 136, and 2000 runs we get the following graph :

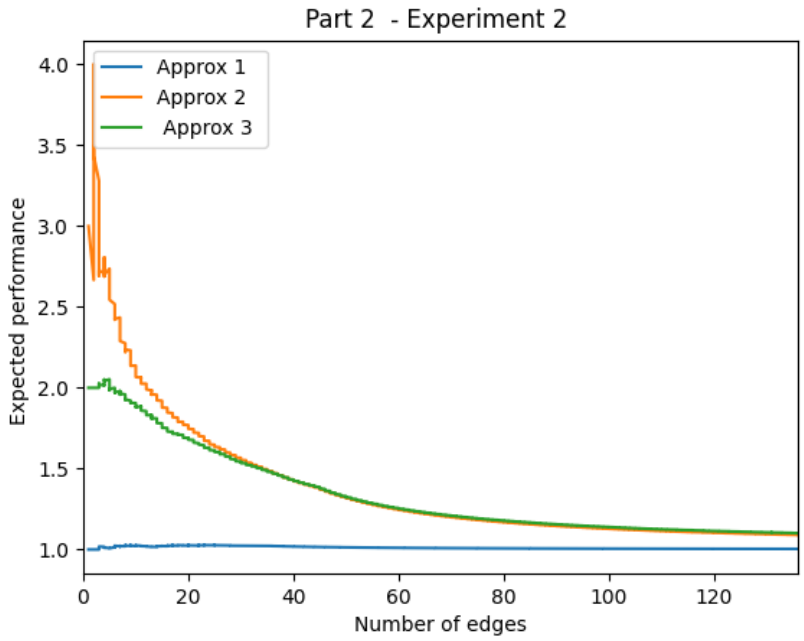

Figure 4: More edges

Conclusion Analysis We observe that the Expected performance decreases across all the approximations when the number of edges increase . This implies that with more edges, the performance becomes similar. Since the randomness factor of approx2 and approx3 becomes obsolete due to the larger amount of edges. So in general, we can say that the approximation becomes better with more number of edges.

Experiment 3 Question - " Is there a relationship between how good we would expect an approximation to be and the number of nodes in a graph? In general, does the approximation get better/worse as the number of nodes increases/decreases? "

Similar to the other experiments we can answer this question by increasing the no of nodes and plotting approximation against the no of increasing nodes.

We can confirm this with another experiment, by taking the number of nodes as 17 , 2000 runs, and keeping the edges constant at 20 to avoid max edges , this following graph is the output:

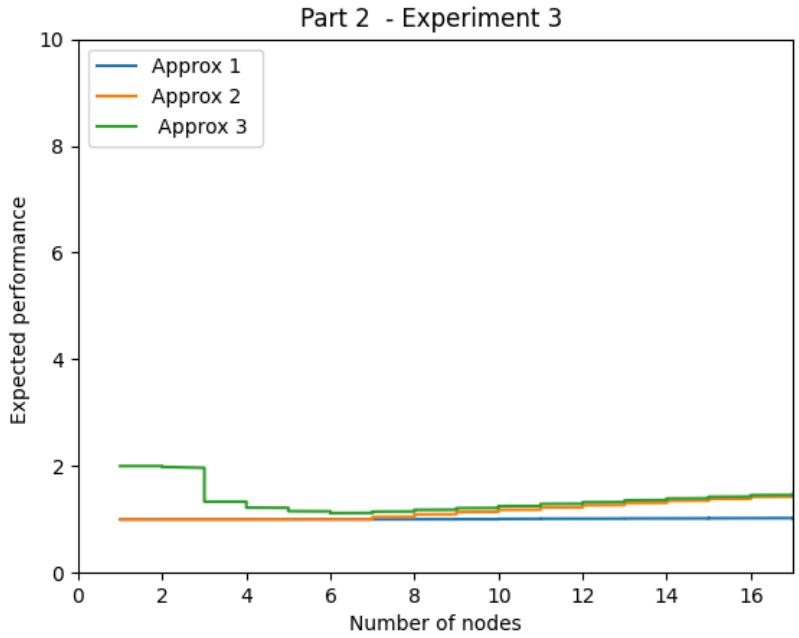

Figure 5: More nodes

Conclusion & Analysis We observe that when the number of nodes increase and the number of edges is kept constant, the expected performance becomes worse for both Approx2 and approx3 while Approx1 almost remains constant. This indicates that when the nodes are increased the approximations perform worse.

Experiment 4 Question - " The approach described in the Potential Experiment is really getting at the average performance of the approximation. What about the worst case of the approximation? To figure that out we would have to test our approximations on every single graph for approx1(). And for the other two the non-deterministic nature of the algorithms makes this even more problematic. However, we may be able to test the worst case for approx1() on very small graphs. How would you generate all graphs of size 5 for example? "

The size of the graph is determined by number of nodes, we can take the limit of the nodes as 5 and plot a graph with increasing edges on x axis and runtime on the y axis. This would show the worst case runtime. we should also consider the fact that maximum number of edges is 10 in this case , by using the formula $n * \frac{n-1}{2}$

if we time approx1 and plot it against the maximum number of edges across 1000 runs we get the following graph:

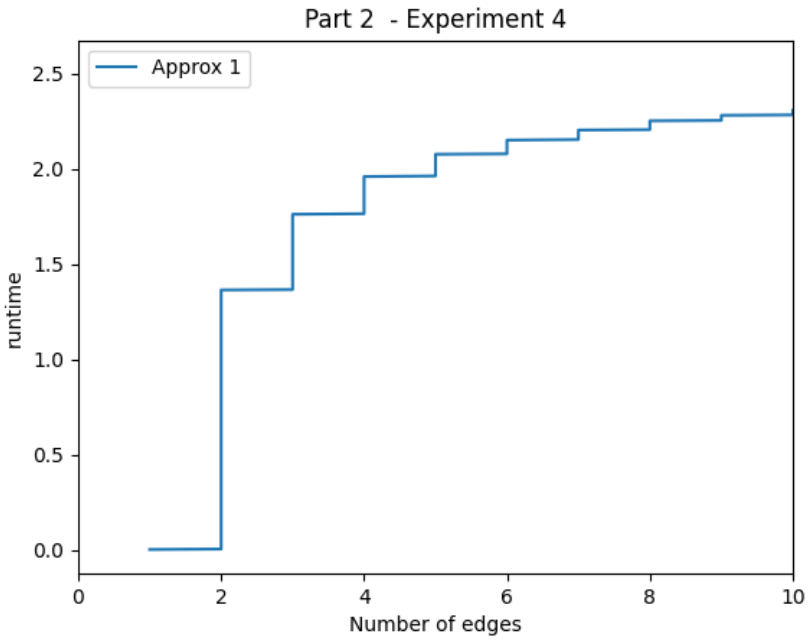

Figure 6: Worst case Approx1

Conclusion Analysis As we can depict from the graph , as the number of edges increase the runtime of Approx1 gets worse, ultimately having its worst runtime when the number of edges is at maximum, with

this experiment we can determine that the worst case runtime for Approx1 is when the number of edges is maximum.

0.4 The Independent set Problem

Description:

The following graph performs the relationship between MIS and MVC.

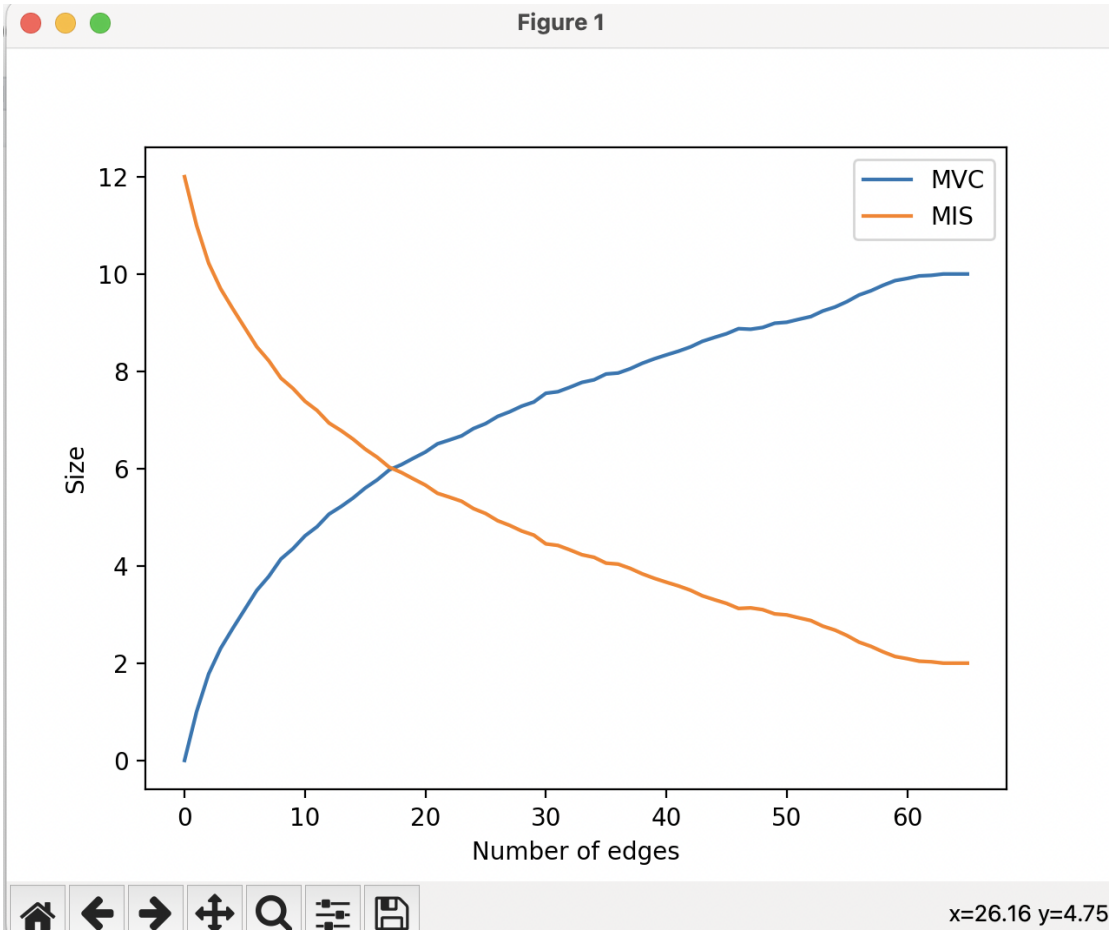

Figure 7: The Independent set Problem

We used the create random graph function from the previous question. We make the number of nodes to be fixed to 12, thus, the maximum possible edges would be "edges", which is 11+ 10+9 +... +1. And for each number of edges(0...edges), we generate 250 random graphs in order to reduce any potential errors. From observing the above graph, we can roughly see that the size of MIS plus the size of MVC is equal to the total number of nodes. And as the number of edges increases, the number of MIS goes down and the number of MVC goes up.

Analysis/Conclusion:

Empirically speaking, the graph makes sense because when the number of edges is zero, there are no nodes connecting to other nodes. So in this case, every node could be in the independent set, so the length of MIS would be the number of total nodes. And based on the definition, since there's no edges, the number of MVC is zero. From the graph, we can observe that the sum of MIS size and MVC size equals to the total number of nodes for any number of edges.

Appendix

0.5 How to navigate the code:

Part1: Experiment1:
Open the file "Part1.py" and uncomment the code you want to run.
Part1: Experiment2:
Open the file "Part1.py" and uncomment the code you wish to run.
Part2: Experiment1:
Open the file name "Part2_MVC.py", the function Exp1() runs the 1st experiment , it doesnt take any inputs.
Part2: Experiment2:
Open the file name "Part2_MVC.py", the function Exp3(number_of_nodes) runs the 2nd experiment , it takes the number of nodes as input ,this calculates the maximum number of edges
Part2: Experiment3:
Open the file name "Part2_MVC.py", the function Exp3(number_of_nodes) runs the 3rd experiment , it takes the number of nodes as input, please input number of nodes as > 7 as the number of edges is 20, doing so will avoid maximum edges.
Part2: Experiment4:
Open the file name "Part2_MVC.py", the function Exp4() runs the 4rd experiment , it takes no parameters.
The Independent set Problem:
Open the file named " Part2_MIS.py", we got everything ready there, run the file or the function called "main()".

End of Lab Report

Group 20, L01.