**CSCE 689 : Algorithms for Big Data**

**Programming Assignment 3 Report**

**Due: Dec 11th** **Name: Vasu Naman Verma**

## Generation of Graph and Distributed Machine Setup:

The graph is generated using the Erdős-Rényi model for creating random graphs. The probability of edges being present or absent is set to 0.2 to create a less connected graph. This is done in order to have a higher number of edges present in the matching.

The simulation of Distributed Machines is carried out by using a 2D array called "edge_sets," which has as many rows as there are machines in the setup. Each row in the "edge_sets" represents a machine. The edges are distributed among the rows after shuffling them, and then they are allocated evenly among the rows. The latency for marking and discarding edges is calculated by calculating the total time taken by all machines serially and then dividing it by the number of machines in the distributed setup to simulate parallel processing.

The distributed maximal matching algorithm has two stopping conditions in implementation. First, if there are no more edges left on any of the machines. Second, is if the total number of edges on all of the machines is less than or equal to memory of a single machine (eta).

## Experiment:

1. In the first case, we generate various graphs using a list of vertices in increasing order.

   For each vertex value in the list, we calculate the number of machines using following formula:

   $$Number\ of\ Machines = \frac{\text{Number of Edges}}{\text{Number of Vertices}}$$

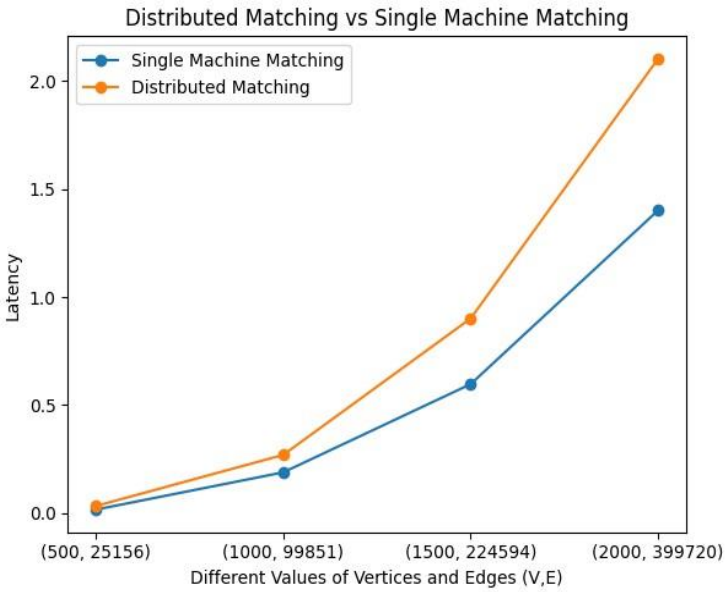
   Epsilon is set at 0.2 and we calculate the memory per machine (eta) using the following formula:

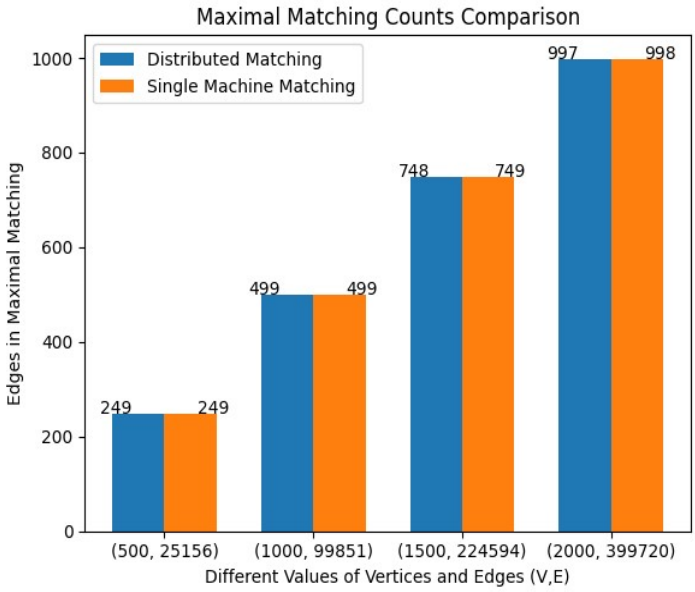
   $$eta = (number\ of\ vertices)^{(1\ +\ \text{epsilon})}$$

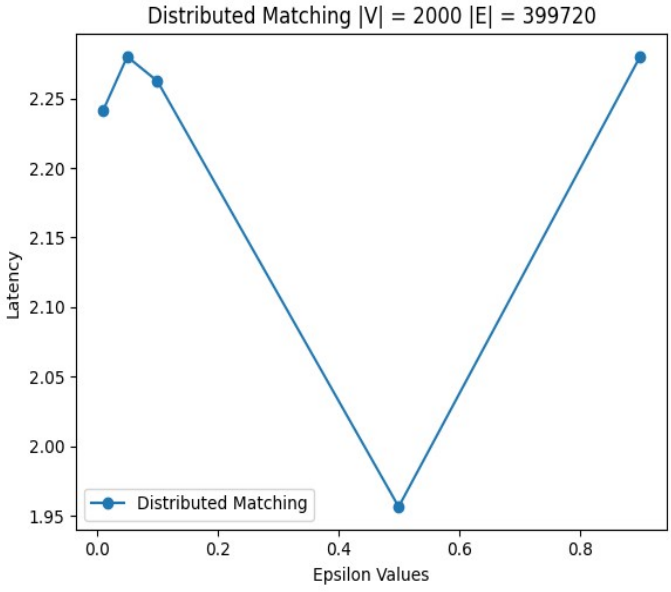
   Subsequently, we compute the latency and maximal matching count using the Distributed Maximal Matching Algorithm. Additionally, we determine the latency and maximal matching count for a single machine capable of accommodating the entire graph.

2. In the second case, we keep the number of vertices fixed at 2000 and vary the values of epsilon. For each epsilon value, we calculate the latency of the Distributed Maximal Matching Algorithm.

**Result:**


Distributed Matching vs Single Machine Matching
Single Machine Matching
Distributed Matching
Latency
2.0
1.5
1.0
0.5
0.0
(500, 25156)
(1000, 99851)
(1500, 224594)
(2000, 399720)
Different Values of Vertices and Edges (V,E)


Maximal Matching Counts Comparison
Distributed Matching
Single Machine Matching
Edges in Maximal Matching
1000
800
600
400
200
0
249
249
499
499
748
749
997
998
(500, 25156)
(1000, 99851)
(1500, 224594)
(2000, 399720)
Different Values of Vertices and Edges (V,E)


Distributed Matching |V| = 2000 |E| = 399720
Latency
2.25
2.20
2.15
2.10
2.05
2.00
1.95
Distributed Matching
0.0
0.2
0.4
0.6
0.8
Epsilon Values

**Analysis:**

In Figure 1, we compare the latency of the Distributed Maximal Matching Algorithm with the Maximal Matching Algorithm on a Single Machine. The figure clearly illustrates that when the entire graph can fit on a single machine, the time is significantly lower compared to the distributed setup. As the graph becomes larger and more complex, the time difference between the two methods increases rapidly. Despite achieving a larger time, we can use machines with lower memory size.

In Figure 2, we compare the count of edges in the maximal matching between the Single and Distributed Approaches. The results show similarity in both cases. However, the values in the Distributed Approach vary on the same graph across multiple runs. This variability is due to the probability of marking an edge and sending it to the main machine, resulting in the count by the Distributed Maximal Matching Algorithm sometimes being smaller or larger than the single machine approach.

In Figure 3, we examine the latency of the Distributed Maximal Matching Algorithm with varying values of epsilon. We have kept the number of vertices fixed at 2000. The figure illustrates that latency varies significantly with different epsilon values. We observe different results on the same graph in multiple runs, again attributed to the probabilistic nature of the algorithm.

**Running the Program:**

To run the program, please make sure to install all the required libraries before proceeding. You can use the following installation commands:

```
pip install networkx[default]
pip install numpy
```

After installing the necessary libraries, you can run the program using the following command:

```
python Assignment3_VasuVerma.py
```

Please note that it will take several seconds to print out the results after running the program as the tests are going over multiple graphs with very high edge count.