Distributed Graph Matching

Anmol Anand
Department of Computer Science
Texas A&M University
aanand@tamu.edu

1 Introduction

In an undirected graph, a matching refers to a collection of edges that do not share any vertices among any pair of edges in that matching. A maximum matching specifically has the largest possible matching size. Often, locating a maximal matching, which is a 2-approximation to the maximum matching, is easier. A matching is maximal if it would stop being a matching on the addition of any other edge.

For large graphs, a distributed algorithm exists to find a maximal matching while equally distributing edges on all available machines. In my project, I've conducted experiments to analyse how the latency for the distributed matching algorithm varies with the number of machines involved.

Definitions

- Let $\epsilon > 0$ be a constant which indicates the size of machines
- Let the algorithm run for rounds $r \in \{1, ..., R\}$ such that $G = (V, E_r)$ is the remaining graph after r rounds where
  - $G_0 = G_0(V, E)$ will be the input graph, and
  - $G_R = G_R(V, \phi)$ will be the subgraph with no edges.

  We perform the following steps in each round $r \in \{1, 2, ..., R\}$ of the algorithm:

Algorithm 1: Round r of the Distributed Matching Algorithm

```
1 $\forall\ i \in \{1, 2, ..., M\}$ machine $i$ marks each edge with probability $\frac{n^{1+\epsilon}}{2|E_r|}$
2 $\forall\ i \in \{1, 2, ..., M\}$ machine $i$ sends marked edges to the master machine
3 The master machine computes maximal matching on the marked edges and sends the newly
    marked vertices to all the machines $i \in \{1, 2, ..., M\}$
4 $\forall\ i \in \{1, 2, ..., M\}$ machine $i$ discards all local edges incident on the marked vertices
```

2 Code

The code can be found in this Github repo github.com/anmol-anand/distributed-graph-matching. The instructions to run the code are mentioned in the README.

3 Sample Generation

I've generated four graphs, each comprising 1000 nodes, with edge counts of 5,000, 10,000, 20,000, and 50,000, respectively. The generation process involves an iterative edge generation process, ensuring that each new edge isn't a self-loop and doesn't already exist as a duplicate. For each graph, and across epsilon values of 0.01, 0.05, 0.1, 0.2, and 0.5, I generate a plot illustrating how the latency changes with the number of machines used for computing the maximal matching.

3.1 Measuring latency

In my code, I have performed all the operations on one machine, in one thread, without any parallelism. The computations for each machine, are made sequentially. However, to measure the equivalent latency of a proper distributed system where each machine runs parallely, we compute the effective latency of the algorithm as follows:

$$Latency(Master) + max_{i=1}^{M} Latency(Machine_i)$$

4 Analysing the Results

Each of the following four figures represents the experiment results for four differently sized graphs respectively. For each graph, the plots show how the algorithm latency varies with the number of machines. Additionally, the plots show how these plots are different when we use different values of ϵ.

The observations are as follows:

- As we would expect, the latency is higher for larger graphs.
- The latency reduces when we increase the number of machines.
- The latency increases on increasing the value of ϵ. This one is not readily obvious. The reason for this is that every round r of the algorithm is performed on larger sets of edges for greater values of ϵ.

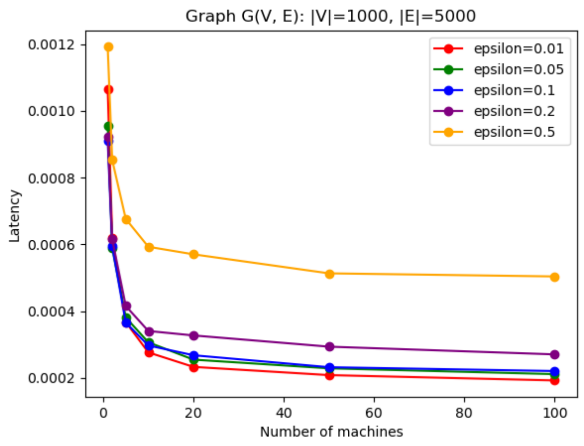

Figure 1: Latency v/s number of machines for Graph 1

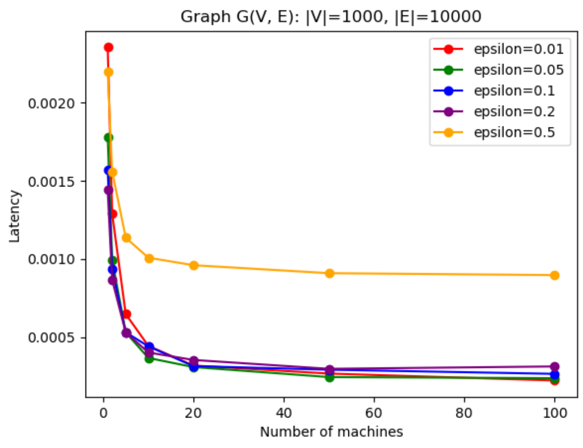

Figure 2: Latency v/s number of machines for Graph 2

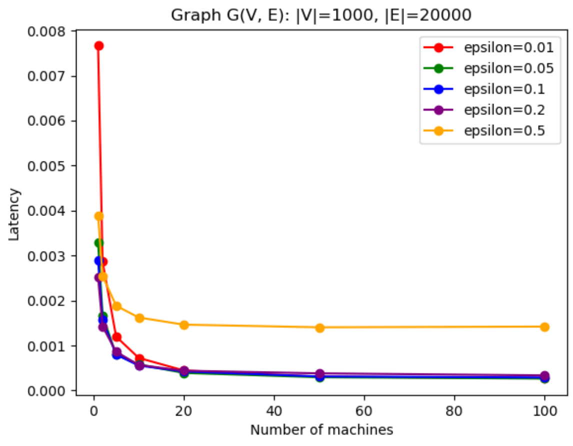

Figure 3: Latency v/s number of machines for Graph 3

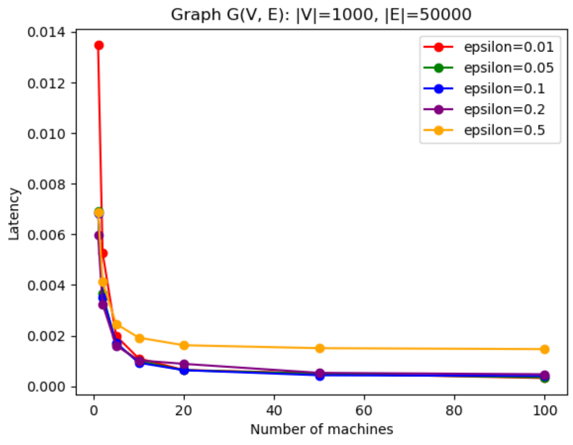

Figure 4: Latency v/s number of machines for Graph 4