ID2222 – Data Mining

Homework 3: Mining Data Streams

Group 10

Xin Ren

Nikhil Dattatreya Nadig

2018-11-17

Introduction

In this assignment, we implement the Flajolet-Martin algorithm called as HyperLogLog in the paper we selected and the graph algorithm called as HyperBall presented in the paper that makes use of the HyperLogLog algorithm to calculate centrality.

Hyperloglog.scala

Hyperloglog is a class that counts the number of distinct items in a stream and it contains following functions:

- Add - Modify the counter for a new item
- Size - Calculate the number of distinct items received so far
- Union - Merge the counters of two itemsets, i.e. compare the counter values of M and N and return the maximum
- Copy - Produce a copy of the counter
- isSame - Check if the counters of two itemsets are the same and return a Boolean value

HyperBall.scala

In this class, We keep track of one Hyperloglog counter for each node; at the t-th iteration of the main loop, the counter c(Int, Hyperloglog) is in the same state as if it would have been fed with BG(v, t), and so its expected value is |BG(v, t)| which is the estimated number of nodes with distance t from v. During the execution of the loop, when we have finished examining node v the counter a is in the same state as if it would have been fed with BG(v, t + 1), and so its value will be |BG(v, t + 1)| in expectation. So we can present harmonic centrality as:

$$\sum_{y \neq x} \frac{1}{d(y,x)} = \sum_{t>0} \frac{1}{t} |\{\, y \mid d(y,x) = t \,\}|$$
$$= \sum_{t>0} \frac{1}{t} \left(|\mathscr{B}_{G^T}(x,t)| - |\mathscr{B}_{G^T}(x,t-1)| \right).$$

The class takes an array contains all the nodes' Ids and their neighbors' Ids as input parameter. The Hyperloglog counter for each node is initiated by adding the node itself. Function execute() starts the iteration and runs it until there is no changes in the counters. Parameter distances stores the centrality of all the nodes and is returned by function execute().

Main.scala

In this class, we call the Hyperloglog and find the number of distinct items of a seq to verify the class. We also load the graph and compute the closeness using HyperBall class' execute method and print the top 10 nodes with the highest harmonic centrality.

Bonus Questions

1. What were the challenges you have faced when implementing the algorithm?
   The implementation of the algorithms is quite straight forward. What challenged us is the hash function in Hyperloglog. At beginning we used direct hash as the input and output of the hash function are both int. However it gave very bad performance as all the nodes were added into the same position of the counter. So instead we changed to byteshash function in scala lib MurmurHash3 and it improved the performance a lot.

2. Can the algorithm be easily parallelized? If yes, how? If not, why? Explain.

   This algorithm can be parallelized. We can employ the method of Gather Apply and Scatter (GAS) paradigm that is used in graph processors such as GraphX. This is done by dividing the graph into subgraphs by vertex cutting them, these subgraphs can then be used as input to compute and perform the union with the other computations to obtain the final result.

3. Does the algorithm work for unbounded graph streams? Explain.

   The algorithm will not be able to work for unbounded graph streams. Following are the scenarios we need to consider for supporting unbounded graph streams:
   - When there is no connection between two nodes, we then receive data about an edge between these two nodes
   - There is a certain shortest distance computed between two nodes, adding another edge creates a new shortest path

   Both these scenarios some shortest paths would no longer be shortest. We would need modify the centrality of affected nodes by removing the contribution of the nodes on the old shortest paths and adding the contribution of the nodes on the new shortest paths. But the algorithm does not store the shortest path info of the graph and the contribution of the algorithm is to store Hyperloglog counter instead of the shortest paths to reduce memory usage. Thus the algorithm does not work for unbounded graph streams.

4. Does the algorithm support edge deletions? If not, what modification would it need? Explain.

   No, the algorithm does not support edge deletion unless we recompute the entire graph. If we assume that the computation of the node centrality is already completed when the edge deletion takes place. Lets try to recalculate the centrality for the affected nodes. For example, we have a simple graph contains a path look like this:

   A-->B-->C-->D

   The edge C-->D is deleted. The affected node here is D. We will have to remove the contribution of C from centrality of D. However we are not sure if we need to do the same for A and B as we are not sure if the shortest paths from A and B to D contain edge C-->D (if B-->D exists or not) since we do not have the info of all the shortest paths in the graph in the algorithm. And storing the Hyperloglog counter instead of all the shortest paths is the whole point of the algorithm.

Thus we can not reduce the centrality of the affected nodes correctly. Combining with our answer for question 3, we conclude that the algorithm does not work for mutable graph.

How to run

sbt run

Dataset

http://konect.uni-koblenz.de/networks/facebook-wosn-links

Result