COSC6339: Graph Analytics
Identifying the K Shortest Paths for a Vertex Pair Browsing Transitive Closure G^+

1 Introduction

In this project you will create a Java/Vertica and Scala/Spark programs to find the shortest path between a pair of vertices in a directed graph. Such pair will be chosen based after the user has a good understqanding of transitive closure. A graph is defined as $G = \{V, E\}$ where V is a set of vertices or nodes and E is a set of edges between the nodes. Each edge has a distance v (weight/cost) associated with it.

2 Program and output specification

- Phase 1: compute transitive closure G^+ and store it on a table. Use the recursive query program. This table will be queried in Phase 2.
- Phase 2: Compute shortest paths of some chosen vertex by browsing G^+. The main program should be called "topkpath", after being compiled, we must be able call the program by typing in the command below. It is acceptable to develop SQL queries in scripts if you cannot modify the Java program.

  Syntax:

```
topkpath sourcevertex=<id>;destinationvertex=<id>;k=<max rec depth>;K=<# paths>
```

and the output shown on screen are the K (uppercase) shortest path lengths between a chosen vertex pair (source, destination). This pair will be chosen on Phase 2 after $G+$ is ready (chosen by TA). Notice that you should not confuse this K with the maximum recursion depth k (lowercase) defined in the paper.

3 Requirements

- The input will be a list of edges and corresponding cost in a csv file with three values on each line: i, j and v. We will provide the data set to test your program.
- Storage
  - In Vertica: G will be represented by a table with three columns: i, j, and v.
  - In Spark: G will be stored as an RDD file cached across the cluster (with Vertex cut partitioning). You can program it in Scala with matrix multiplication or call functions in GraphX.
- Programming: you must use Java (or C++) for Vertica, and Scala (or Java) for Spark. You are not allowed to export the data set outside the system and process it in the Java/C++ and Scala programs in main memory (e.g. with arrays storing E). That is, processing must be done inside each system. In other words, Vertica and Spark do the main job. ONLY the result is sent to the Java and Scala program to be displayed on the screen.

  SQL: It is acceptable to solve Phase 2 with your own SQL queries, not automatically generated by the Java program. It is acceptable to create temp tables provided your program does not become too slow.

- it is unacceptable to precompute all potential paths at any recursion depth since that approach is very inefficient.
- The problem will be solved by calculating Transitive Closure of the graph G*, i.e. iteratively calculate an adjacency matrix multiplication as explained in [1, 2].
- You can assume the maximum path length up to $k \leq 10$ edges (max. recursion depth).
- The program should not halt when encountering errors. It should just send a message to the log file and continue with the next line. The only error that is unrecoverable is a missing input file or a missing output file.
- Deliverables: source code and a README file showing how to compile and run the programs from the Unix terminal. Run the programs a few times and record differences between the two systems. Which one processes the query faster? You are expected to show and explain your program to the TA.

4 Extra credit: highly ecnouraged

- 30 points for any teams showing actual paths. Note: the path should be stored INSIDE each system, only the final path to be retrieved from
- Any team reprogramming the system in C++/ODBC will also get 30 points.

References

[1] C. Ordonez. Optimization of linear recursive queries in SQL. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 22(2):264–277, 2010.

[2] C. Ordonez, W. Cabrera, and A. Gurram. Comparing columnar, row and array dbmss to process recursive queries on graphs. *Information Systems*, 2016.