HELLENIC REPUBLIC
National and Kapodistrian University of Athens

DEPARTMENT OF INFORMATICS & TELECOMMUNICATIONS

MADIK Management of Data, Information & Knowledge Group

ACM SIGMOD 2016 Programming Contest (uoa_team)

Athanasios-Michail Karampatsis, Nikos Dimakopoulos, Georgios Alexandropoulos, Nikos Tzamos
Yannis Foufoulas
Department of Informatics and Telecommunications, University of Athens, Greece
MaDgIK LAB, http://www.madgik.di.uoa.gr

Task description

- Calculate shortest path on a dynamic directed unweighted graph.
- An initial graph is given during the preprocessing stages.
- When the main execution starts, batches of queries are processed containing either updates or shortest path queries.
- Updates may concern insertions or deletions of edges.
- The results must be printed at the end of each batch.

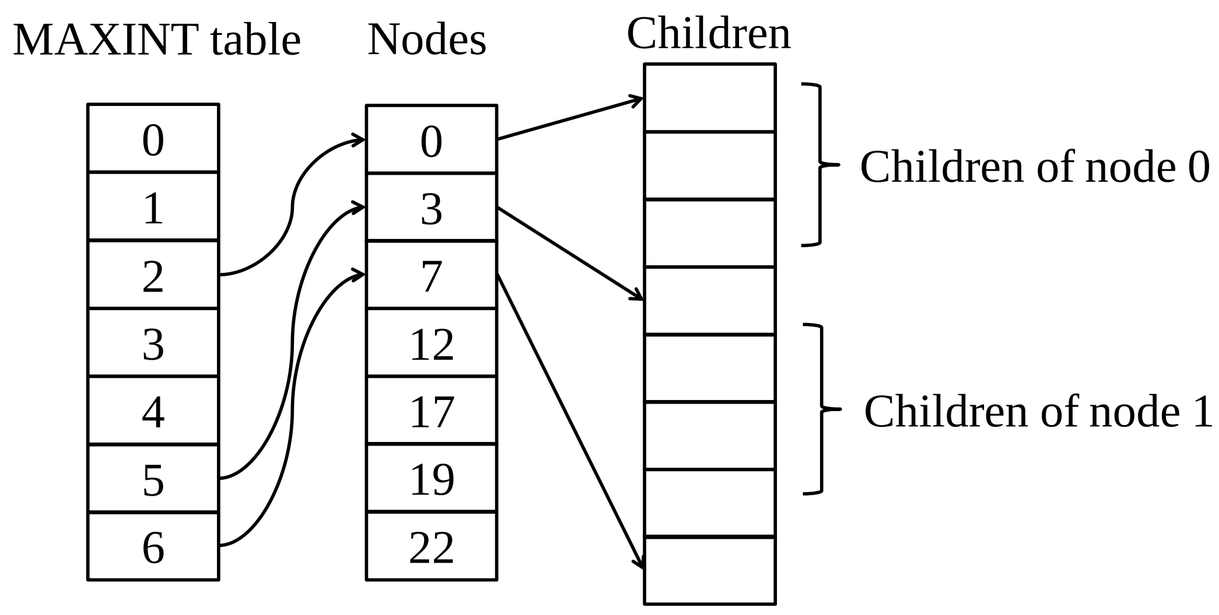
Strategy

- Use an Enhanced Bidirectional BFS to calculate single-source shortest path.
- Parallelize shortest path queries to multiple threads using multiversioning data structures.
- Use heuristics to optimize multi-threading and Bidirectional BFS.

Data Structures

- A hash map is used to create a dense adjacency list.
- A variation of an adjacency list that achieves higher cache locality is used to represent the whole graph.
- Two adjacency lists that keep the node names, along with their updates (insertions or deletions) and a version id are used to support multiversioning.

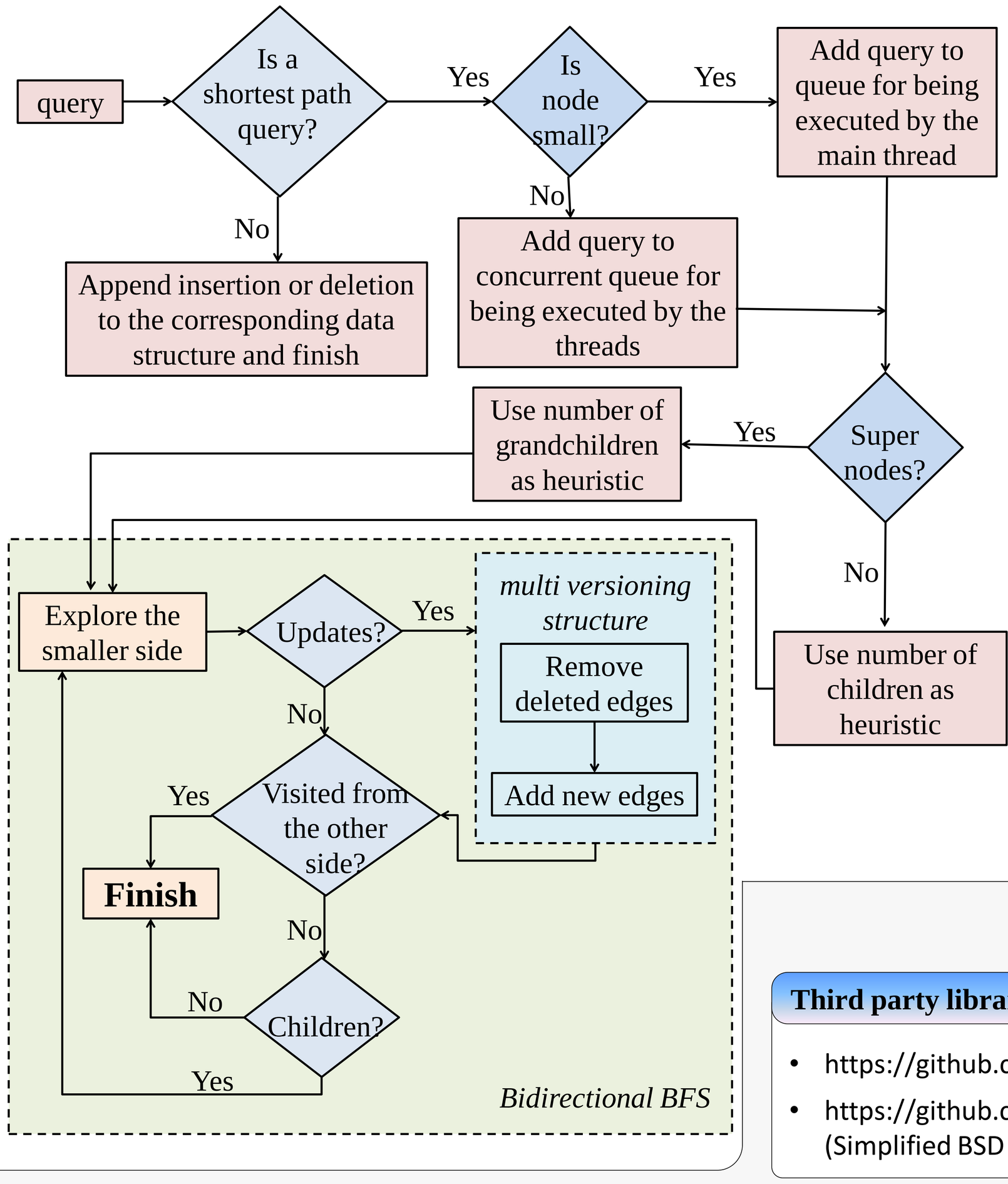
Graph Representation



Preprocessing

- Load graph
- Mark as small, nodes with spanning trees that consist of less than 5 nodes.
- Search for super nodes (nodes with much more connections than average). If there are super nodes in the graph calculate also the number of grand children of every node, to use as heuristic.

Validation Processing



Multithreading strategy

We addressed the following problems :

Skew

A major problem when running queries in parallel is skew. In order to eliminate it, a single concurrent queue is used and all threads consume jobs from this queue.

Locks

The concurrent queues are slower when there are many locks. This happens when there are many light queries and the threads request new jobs continuously.

We solved this problem by running light queries in single thread mode. Queries on small nodes are considered as light. Small nodes are marked during the preprocessing, and remain small if no insertions have been applied to their spanning trees.

Third party libraries used

- https://github.com/tghosgor/threadpool11 (LGPL v3.0)
- https://github.com/cameron314/concurrentqueue (Simplified BSD License)