Constructing a Social Network Analysis System for SIGMOD 2014 Programming Contest

VIDA Team: Fernando Chirigati[1], Kien Trung Pham[1], and Tuan-Anh Hoang-Vu[1]
Supervised by Huy T. Vo[1,2]
1. NYU Polytechnic School of Engineering | 2. Center for Urban Science and Progress (CUSP)

TASK

- Construct a social network analysis system that can execute a set of queries as quickly as possible and output results to standard output

Queries

- 4 types of queries provided in a text file
- Perform tasks that require graph search

Data

- Datasets (a total of 31) from the LDBC social network benchmark generator: *https://github.com/ldbc/ldbc_socialnet_bm*
- Different social network sizes are tested: small (1K persons), medium (10K persons), large (100K persons), and huge (1M persons)

Evaluation Environment

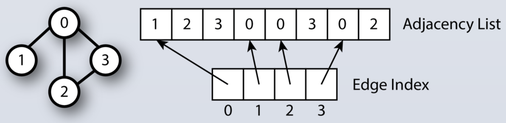
- Processor: 2.67 GHz Intel Xeon E5430
- Configuration: 2 processors (8 cores at total)
- L2 Cache Size: 12 MB
- Main Memory: 15 GB
- Operating System: RHE Linux Server 6.5
- Compiler: GCC 4.4.7

QUERY PRE-PROCESSING

- Input queries may come in any order
- System internally sorts queries by query type
- Repeated queries are detected and processed only once
- Original order is preserved in the output

I/O AND DATA STRUCTURES

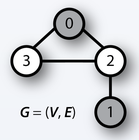
- We use memory mapped files from Boost to improve the I/O performance for large files
- Whenever possible, static allocation, rather than dynamic allocation, is used
  - Size of a data structure is estimated based on the size of the dataset file to pre-allocate memory
- Arrays and vectors are preferred over maps and sets
- Graphs are constructed using a compact adjacency list representation



- Some data structures are used across all query types
  - They are initialized when the system starts and then *shared* among all queries

    E.g.: *Persons* and *Tag* information, *Persons Graph*

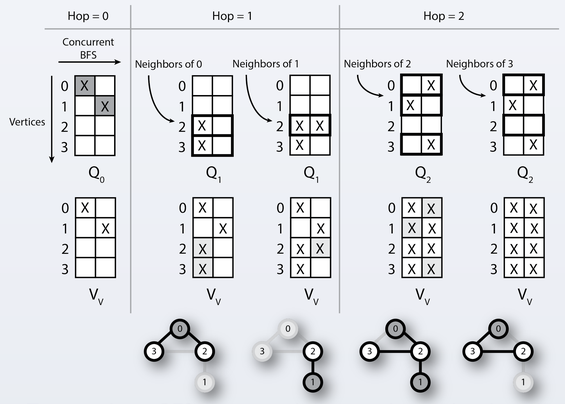
THE MS-BFS TECHNIQUE



Initialize visited vertices list V_V;
For hop h = 1… k (i.e.: until all vertices are visited):
  Initialize queue Q_h;
  For each vertex u in V:
    If $Q_{h-1}[u]$ == 0, continue;
    For each neighbor v of u:
      $Q_h[v]$ |= $Q_{h-1}[u]$ & ~$V_V[v]$;
      $V_V[v]$ |= $Q_h[v]$;

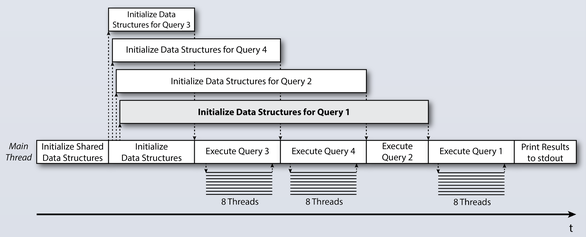
Time Complexity

$O(k * 2 * |E|)$ = **$O(k * |E|)$**



- To speed up BFS computations, we have developed a technique that takes advantage of SIMD instructions and bit operations to efficiently execute multiple BFS concurrently in a single thread – the ***Multiple-Sources BFS***, or simply ***MS-BFS***:
  - Given a fixed graph, multiple BFS can be executed concurrently without the need of a locking mechanism or multiple threads
    - Bit masks and bit operations are used for efficiency
    - 64-bit mask: 64 concurrent BFS
    - Time complexity for 64 BFS is O(|E|) for dense graphs, while the usual algorithm takes O(|V|+|E|) for a single BFS
  - Since queues may overlap, vertices can be "*shared*" and visited only once for multiple concurrent BFS

MULTITHREADING STRATEGY

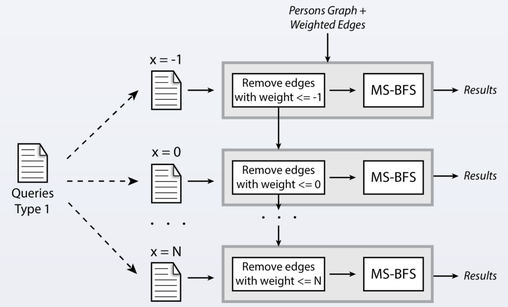


- We use Boost to support multithreading in the system
- We use *inner* multithreading (i.e.: inside each query type) rather than *outer* multithreading (i.e.: across different query types) for executing queries
  - Initializing data structures for query type 1 is a bottleneck, since its I/O is time-consuming
  - It provides a better use of resources
    - Evaluation environment has 8 cores

QUERY TYPE 1

Shortest Distance over Frequent Communication Paths

Given two person ids p1 and p2, and an integer x, find the minimum number of hops between p1 and p2 in the graph induced by persons who: (i) have made more than x comments in reply to each others' comments, and (ii) know each other.

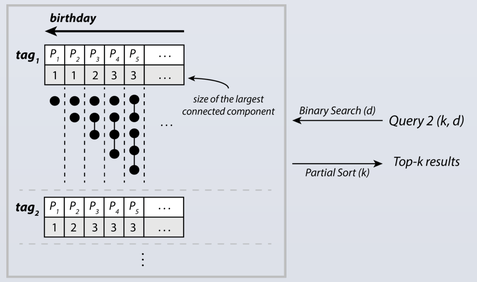


- Run BFS for each query
  - Number of comments between two persons is added as a weight to the corresponding edge in *Persons Graph*
- Group queries by *x* so that the graph can be incrementally reduced, which improves the performance of the BFS
- 8 threads are used for query type 1, and each thread executes one MS-BFS
  - 512 queries are executed concurrently

QUERY TYPE 2

Interests with Large Communities

Given an integer k and a birthday d, find the k interest tags with the largest connected component in the graph induced by persons who: (i) have that interest tag, (ii) were born on d or later, and (iii) know each other.

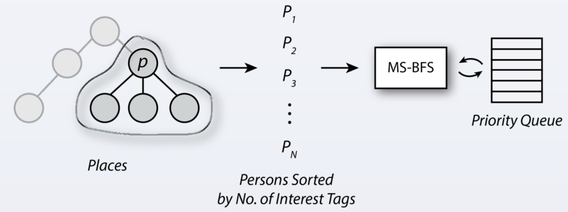


- Pre-compute the size of the largest connected components for each tag *prior* to query execution
  - List of persons who are interested in the tag is sorted in decreasing order of birthday
  - Pre-computation done incrementally and with respect to the sorted list: for each person, calculate the size of the largest connected component up to that person
- For each query, binary search is used to get the size of the largest component for each tag

QUERY TYPE 3

Socialization Suggestion

Given an integer k, a maximum hop count h, and a place name p, find the top-k pairs of persons based on the number of common interest tags. For each of the k pairs, the two persons must be located in p, or study or work at organizations in p. Furthermore, these two persons must be no more than h hops away from each other.

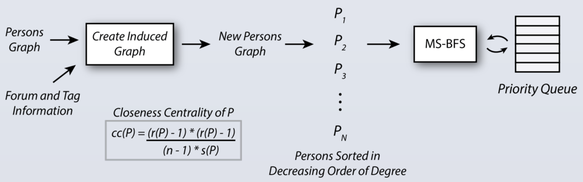


- For each person in *p*, run a BFS to get pairs of persons, respecting the constraints of *p* and *h*; a priority queue of size *k* maintains the top-k pairs
- A graph of places is used to find persons in *p*
- Persons are read in decreasing order of number of tags
- **Early termination:** stop when number of tags of the upcoming person is less than the current minimum of the queue
- Queries are processed sequentially, and 8 threads are used for each query – each thread has its own queue
  - Queues are merged at the end of the execution

QUERY TYPE 4

Most Central People

Given an integer k and a tag name t, find the k persons who have the highest closeness centrality values in the graph induced by persons who: (i) are members of forums that have tag name t, and (ii) know each other.



- Run one BFS per person in the induced graph to compute the closeness centrality
  - Persons are sorted in decreasing order of their degree
  - MS-BFS is executed respecting this order and maintaining a priority queue for the top *k* results
  - **Early termination:** stop BFS when the current accumulated *s(P)* is greater than the current maximum *s(P)* maintained in the queue
  - Intuition: persons with higher degree will have smaller *s(P)*, helping early-terminate a higher number of BFS
- Threading mechanism similar to query type 3
  - 8 threads per query, and each thread has its queue