# Midterm Report

# SIGMOD programming contest 2014

## Progress Report

*We have got the implementations of all the queries working.*

We use the neo4j graph database system, and the py2neo library which exposes its Python interface.

Our code can be found at <https://github.com/guptha-/sigmod> (Private repository - will add as collaborator if required)

We have got 4 queries working, out of which two have been optimized to a fair extent. The queries have been tested against the small dataset supplied by the organizers of the programming contest. Timing results\* have been collected.

We execute queries over two phases. The first phase involves loading the database. This is done once for each query type. The time taken for this case can be considered as amortized over the entire run time of the queries. After this, we directly execute the queries for each set of test case parameters.

#### Query Type 1 (Shortest Distance Over Frequent Communication Paths)

**Unoptimized approach**: For the given start node and end node, find paths between them (in the person\_knows\_person graph) with minimum edges in between. Verify if the path found is indeed a frequent communication path. If yes, the path is the solution. If not, look for longer paths between those nodes that are frequent communication paths. The approach took prohibitively long amounts of time for some of the test cases, for instance where there is no frequent communication path between the given nodes.

**More-optimized (current) approach:** Do a BFS traversal on the person\_knows\_person graph, starting from the given start node, until the given end node is not encountered. The BFS traversal has to be such that only those outgoing edges from a node are considered that satisfy the frequent communication condition between the two nodes across that edge. If the BFS queue is empty at some point, no such path can be said to exist between the given start/end nodes.

#### Query Type 2 (Interests with Large Communities)

**Optimized (current) approach:** Index Person nodes in the graph by using a Lucene Index for range queries, since people with birthday after a certain day are desired.

Get the Persons satisfying the given birthday condition by querying the Lucene index. For every Interest tag, find the people among these that are associated with this particular Interest tag. Do a BFS traversal on the Person nodes associated with each of the Interest tags, to find all the connected components in the graph induced by Person nodes with that Interest tag and edges from person\_knows\_person. Store the connected component with the largest size for each Interest node. Return the top k Interest tags with the largest connected components.

#### Query Type 3 (Socialization Suggestion)

**Unoptimized approach:** Query for all the Person nodes that lived, studied or worked in the given place. For each of the persons, get their list of Interest tag nodes. With each Person as a start node, do a BFS traversal on the person\_knows\_person graph, up to a depth of h (given parameter), to find the Person node satisfying the place condition and that has maximum overlapping interest tags with the starting node. From all the BFS traversal results, find the top k (given parameter) node couples that had maximum overlap in their interest tags.

**Optimized (current) approach:** Instead doing a BFS traversal with each Person (satisfying place condition) as the start node, find the number of matching Interest tags for each pair of persons. Sort the pairs in decreasing order of the number of overlapping interests. Traverse this list of Person pairs checking if a path less than or equal to length h exists for that pair. Enumerate k such pairs.

#### Query Type 4 (Most Central People)

**Unoptimized (current) approach:** Query all the forums that are associated with the given Interest tag. Query for all the Person nodes that are members of those forums. With each Person node as a starting point, do a BFS traversal on the person\_knows\_person graph, considering only those nodes that satisfy the Forum-Interest tag condition, until the BFS queue is empty. This would tell what all nodes are reachable from a given Person node and at what distance are they present. Using this information, find the centrality measure of every Person node that satisfies the Forum-Interest tag condition. Find the top k nodes among these that have the maximum value for this measure.

## Refined Project Milestones

#### April 1 (Mid-term report)

Done: Got all 4 queries working, with two of them optimized

#### April 12

Optimize the remaining two queries

#### April 24

Try further optimizing all the queries individually

#### April 29 (Project presentation)

Try to bunch optimizations such that all of them run well together.

#### May 8

Finish the project report

## Timing Results (all times in s)

**\*Important note on the timing results:**

The timing results below were obtained on a 1.7 GHz core i5 (2 physical cores, 4 logical cores, 512 KB L2 cache), 6 GB RAM machine with a conventional HDD, which is far less powerful than the system the SIGMOD organizers provide. We cannot use their system because our code is in Python, which they don’t support.

#### Query 1

Load time 7354.18300009

Individual runs:

576 400 -1 time 38.6770000458

58 402 0 time 34.5739998817

266 106 -1 time 33.6689999104

313 523 -1 time 34.3120000362

858 587 1 time 54.8840000629

155 355 -1 time 34.9470000267

947 771 -1 time 28.3529999256

105 608 3 time 82.4440000057

128 751 -1 time 31.8029999733

814 641 0 time 46.3350000381

Total q1 time 420.001000166

Load time of query 1 is quite high. We will aim to optimize that first.

#### Query 2

Load time 32.6449999809

Individual runs:

3 722846 time 80.8550000191

4 723249 time 61.5350000858

3 723633 time 56.0460000038

3 724039 time 60.3880000114

5 724459 time 41.0669999123

3 724792 time 34.0409998894

3 725171 time 38.2730000019

7 725546 time 23.4509999752

3 726051 time 14.3300001621

4 726492 time 8.10899996758

Total q2 time 418.09800005

We would want to push some common optimization into the load phase so that the cost is amortized over the runs.

#### Query 3

Load time 27.6159999371

Individual runs:

3 2 Asia time 473.381000042

4 3 Indonesia time 1.94899988174

3 2 Egypt time 0.444999933243

3 2 Italy time 0.311000108719

5 4 Chengdu time 0.137000083923

3 2 Peru time 0.157999992371

3 2 Democratic\_Republic\_of\_the\_Congo time 0.108999967575

7 6 Ankara time 0.0980000495911

3 2 Luoyang time 0.0899999141693

4 3 Taiwan time 0.131000041962

Total q3 time 476.811999798

We need to optimize the case where there are a lot of people present in a place.

#### Query 4

Load time 440.994999886

Individual runs:

3 Bill\_Clinton time 9849.704

4 Napoleon time 6724.16999984

As the runs of the 4th query are taking a long time, we are unable to produce timing results for the rest. We are already working on optimizing it.