Friendster Network Analysis

The Da Vinci Code
Xi Chen, Yangyang Dai, Rose Gao, Liqiang Yu

Agenda

1. Data
2. Shortest Path & Degrees
3. Community Detection
4. Centrality Measurements
5. Friends Recommender
6. Most Well Connected Person (Graph Radius)
7. NetworkX (Visualizations, Centrality Measurements, Friends Recommender)

Data

Friendster Dataset

- Total dataset is 30 GB
- We used a subset of 140 MB for most of our analysis

Sample data subset:

```
170:101,168,173,175,187,202,204,209,210,217,222,298,341,349,691,806,807,91770
171:notfound
172:134
173:170,198692,709856,778213,2433982,22016441,26999667
174:notfound
175:private
```

Friends Recommender

Motivation: Recommend new friends to a user based on mutual friends

MapReduce Algorithm:

1. For each pair of people who aren't friends with each other, but share at least one mutual friend, yield the count of their mutual friends.
    - E.g. Alice is friends with Brian and Claire, who are not friends with each other:
        - yield Brian, (Claire, 1)

2. Group by key (user name)
    - E.g. Brian, [(Claire, 5), (Stuart, 1), (Michael, 3), (Amy, 4), ...]

3. Use heapq.nlargest to yield 5 people with whom a user shares the most mutual friends, and thus is likely to add as new friends.
    - E.g. Brian, [(Tom, 7), (Claire, 5), (Amy, 4), (Michael, 3), (Amanda, 3)]

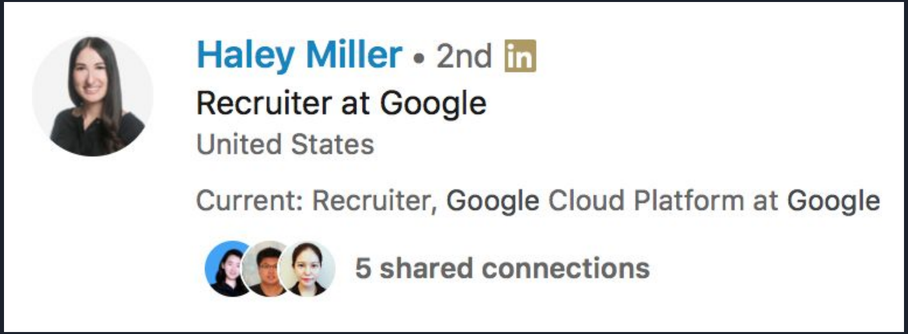
Runtime:

To process 1,000 users, we used 1 local machine => 6.2 seconds

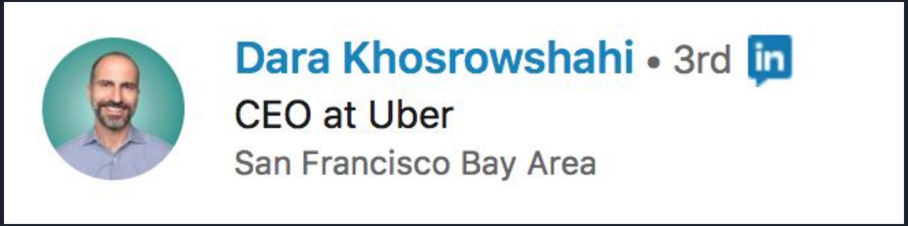
To process 1,000,000 users, we used 3 machines on GCP dataproc => 12 hours

Shortest Path & Degrees

Motivation: How you could be introduced to someone you want to know? How far are you from this person?



Haley Miller • 2nd
Recruiter at Google
United States
Current: Recruiter, Google Cloud Platform at Google
5 shared connections

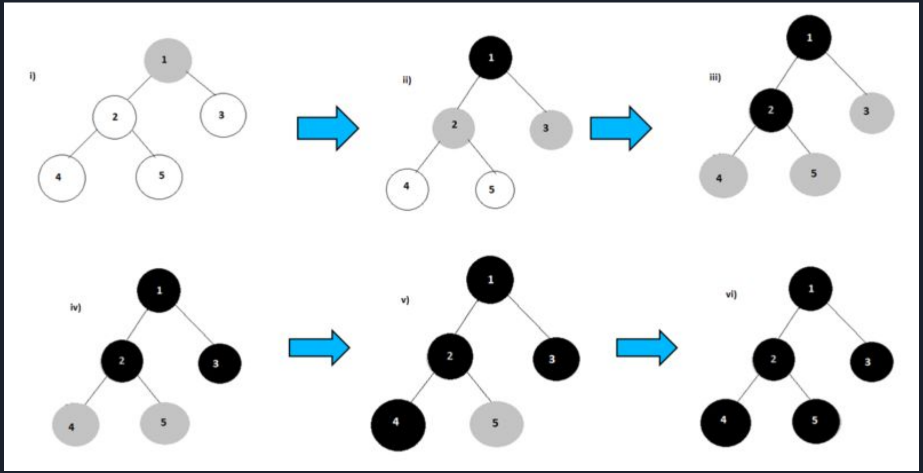


Dara Khosrowshahi • 3rd
CEO at Uber
San Francisco Bay Area

Shortest Path & Degrees

Breadth-first search in MapReduce

- BFS - O(|V| + |E|)
- Iterative Mapreduce



Shortest Path & Degrees

Breadth-First Search in MapReduce

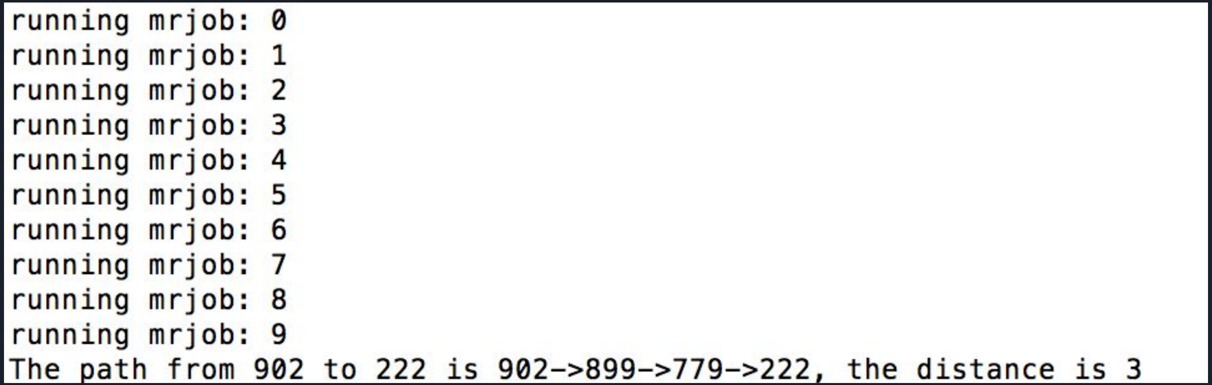
1. Create a temporary file to store the graph
   1.1. ID | Children | Distance | Path | Color (a Node)

2. Iterative Mapreduce Algorithm
   2.1. Mark the beginning node as ‘gray’
   2.2. Expand each child of it
   2.3. Combiner - yield Child_ID, Child_ID | Null | Distance + 1 | Path + Parent_ID | ‘gray’ (a Node)
   2.4. Reducer - Reduce to the shortest path, save the darkest color, store adjacent nodes

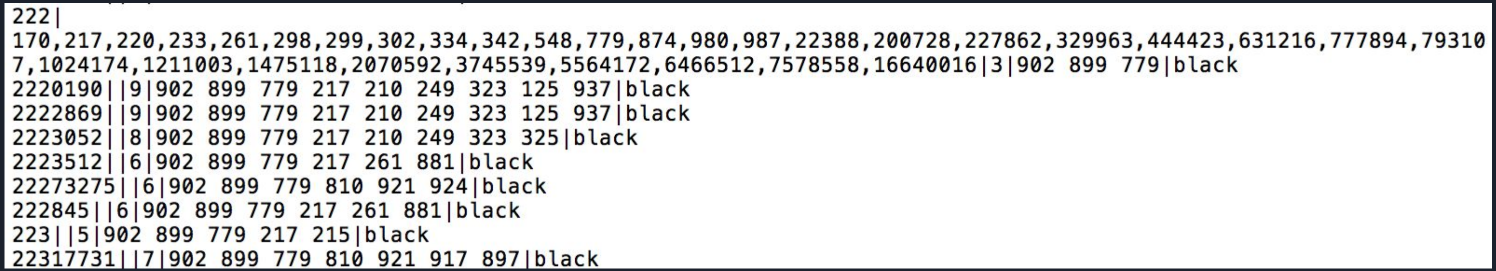
3. Iterate 10 times
   3.1. for i in range(N)
   3.2. with mr_job.make_runner() as runner:
   3.3. runner.run()

Shortest Path & Degrees

Results



```
running mrjob: 0
running mrjob: 1
running mrjob: 2
running mrjob: 3
running mrjob: 4
running mrjob: 5
running mrjob: 6
running mrjob: 7
running mrjob: 8
running mrjob: 9
The path from 902 to 222 is 902->899->779->222, the distance is 3
```



```
222|
170,217,220,233,261,298,299,302,334,342,548,779,874,980,987,22388,200728,227862,329963,444423,631216,777894,79310
7,1024174,1211003,1475118,2070592,3745539,5564172,6466512,7578558,16640016|3|902 899 779|black
2220190||9|902 899 779 217 210 249 323 125 937|black
2222869||9|902 899 779 217 210 249 323 125 937|black
2223052||8|902 899 779 217 210 249 323 325|black
2223512||6|902 899 779 217 261 881|black
22273275||6|902 899 779 810 921 924|black
222845||6|902 899 779 217 261 881|black
223||5|902 899 779 217 215|black
22317731||7|902 899 779 810 921 917 897|black
```

Motivation for Community Analysis

- The world is big
- We live in communities
- Most important or well-connected people in the community

Community Detection

MapReduce Algorithm:

- Construct the adjacency vector for each user

  E.g., user 1: user 3, user 4 -> [1, 1, 0]

  user 2: user 3, user 5 -> [1, 0, 1]

- Calculate the Euclidean distance for each pair of users/vectors
- Rank the distance and choose the first 50 people to be in the same community

Centrality Measurement (I)

- **Betweenness Centrality**

$$C_b(v_i) = \sum_{s \neq t \neq v_i} \frac{\sigma_{st}(v_i)}{\sigma_{st}},$$

  - Given (pair, shortest path)
  - for each community
    - calculate total number of shortest path that passes through every node
    - The sum divided by the product of the scale and the length of the community.
    - The most important person gets the highest score.

- **Closeness Centrality** (new approximation)

$$C(x) = \frac{N}{\sum_y d(y,x)}$$

  - Traditionally based on the shortest path
  - New design: based on the distance between members from community detection
  - For each community (using MapReduce):
    - Calculate each member’s total distance to other members
    - The “closest” or most central person is the one with the shortest total distance

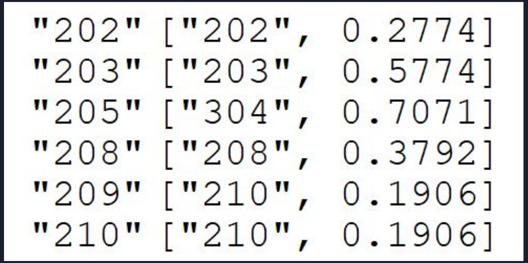
Centrality Measurement (II)

- **Degree Centrality** $$C_d^{\text{norm}}(v_i) = \frac{d_i}{n-1},$$
    - The more connections/friends, the more important.
    - For each community (Using MapReduce):
        - The number of friends each member has in this community
        - Normalization by the scale of the community
        - The most important person gets the longest list of friends

- **Eigenvector Centrality** $$c_e(v_i) = \frac{1}{\lambda}\sum_{j=1}^{n} A_{j,i}c_e(v_j),$$
    - Having more important friends provides a stronger signal
    - For each community (using MapReduce):
        - Construct adjacency matrix of the graph
        - Select the largest eigenvalue
        - Compute the corresponding eigenvector
        - The most central person is the one with the highest element in the eigenvector

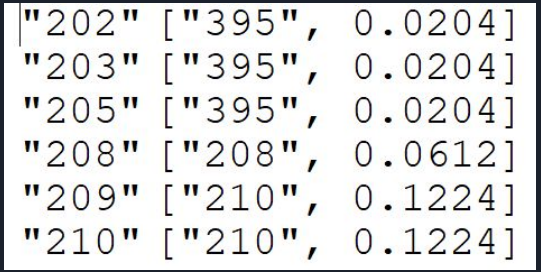
Comparison Between Centrality Measurements

Closeness centrality:



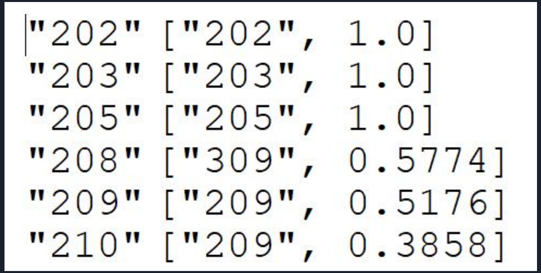
```
"202" ["202", 0.2774]
"203" ["203", 0.5774]
"205" ["304", 0.7071]
"208" ["208", 0.3792]
"209" ["210", 0.1906]
"210" ["210", 0.1906]
```

Degree Centrality:



```
"202" ["395", 0.0204]
"203" ["395", 0.0204]
"205" ["395", 0.0204]
"208" ["208", 0.0612]
"209" ["210", 0.1224]
"210" ["210", 0.1224]
```

Eigenvector Centrality:



```
"202" ["202", 1.0]
"203" ["203", 1.0]
"205" ["205", 1.0]
"208" ["309", 0.5774]
"209" ["209", 0.5176]
"210" ["209", 0.3858]
```

Most Well Connected Person (Graph Radius)

Motivation: Determine the most well connected person in the network

MapReduce Algorithm:

1. Run our MapReduce breadth-first search algorithm on each user to yield shortest paths from that user to every other user (either possible within 10 degrees or not).

2. Find the distance (in degrees of separation) between a user and the farthest person in their network, call it max_distance.

3. Find the user with the smallest max_distance, otherwise known as the graph radius.

Results:

Our results were unexpected - users who were friends with 1 other user exclusively yielded a minimum max_distance of 1.

Future work will be to implement our algorithm on the largest connected component of the network to yield more meaningful results.

NetworkX

Communities

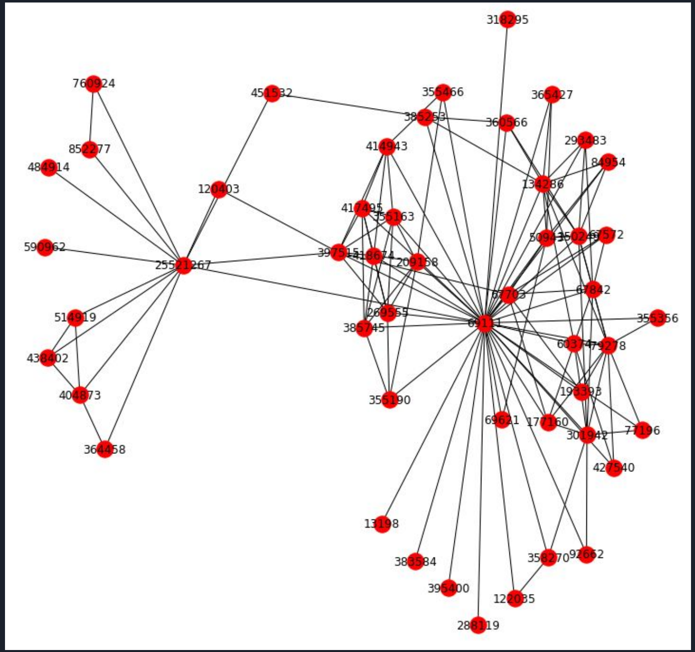
- randomly chosen centers
- 50 closest people to a center

Centralities:

- Closeness*
- Betweenness
- Degree*

Friends Recommender

*rewrote NetworkX methods


318295
760924
451532
355466
365427
385253
360566
414943
293483
852277
84954
484914
134286
120403
417495
355163
590962
25521267
200158
67703
67842
514919
269555
355356
385745
79278
438402
193393
404873
355190
69621
177160
301942
77196
364458
427540
13198
358270
92652
383584
396400
122035
288119