On the Connectivity of the Reddit Community

Sean Li
Cornell University
sxl6@cornell.edu

Feb. 7, 2013

The main statistical result of this paper is that of the reddit users who have upvoted at least one post, 31.4% of them have upvoted only one post and, moreover, have been the *only* person to upvote it. Conversely, 22.6% of posts which have been voted on at all have received only downvotes and no upvotes. In addition, 67.4% of users are connected through mutually upvoting posts, while 91.4% of posts are connected by having mutually been upvoted by users.

1 Introduction

In many large graphs that are connected by some rule of the form "two vertices are connected if and only if they have property X," there exist a giant connected component and a scattering of tiny ones. For instance, in chemistry we can form a graph with proteins as vertices and protein interactions as edges, where a protein interaction is said to occur if two amino acids bind to each other for a function. The graph has 2735 vertices and 3602 edges, and the results are as follows [H1]:

Size of Component	Number of Components
1	38
2	179
3	50
4	25
5	14
6	6
7	4
8	6
9	1
10	1
11	1
16	1
1851	1

The same phenomenon occurs looking at words in the English language as vertices and using the rule that an edge between two words implies they are listed as synonms in the Moby Thesaurus [H1]:

Size of Component	Number of Components
1	2
2	1
3	1
4	1
14	1
16	1
18	1
48	1
117	1
125	1
128	1
30242	1

Note that this seems somewhat absurd. Certainly 30242 words are not synonymns of each other, but it could be that given 3 words

w_1, w_2, w_3 and letting $\approx$ denote synonym, we could have $w_1 \approx w_2$ and $w_2 \approx w_3$ but $w_1 \not\approx w_3$. This is to say that the synonym relation is not transitive. It also means that given two arbitrary words x and y, there is a very good chance you can find a chain of words $x = w_1, w_2, \ldots, w_k = y$ such that $w_i \approx w_{i+1}$ at each step, even if x and y mean totally different things.

2 The Reddit Data

We can think of a similar experiment for the reddit community. If we say that users are vertices and an edge exists between two users if they upvoted the same post, can we expect to find a giant group of people all connected this way? It turns out the answer is yes. From some 2010 data [H1], which I didn't realize it was old data until I had already processed it, my algorithm obtained the following results:

Size of Component	**Number of Components**
1	9839
2	53
3	5
4	2
6	3
10	1
21322	1

I used a very large data set consisting of vote data of users on the site `reddit.com`. The original data, from a torrent to a 29 MB gzip compressed .csv file, is found at `http://www.reddit.com/r/redditdev/comments/bubhl/csv_dump_of_reddit_voting_data/` Keep in mind that this data is two years old, and might not represent the current data accurately.

The file contains 7,405,561 lines of data,* where each entry is a 3-tuple of the form

`username`, `post_id`, `vote`.

The first two are arbitrary strings, while `vote` is always 1 or -1, corresponding to an upvote or a downvote respectively. In total there are 31,553 unique users and 2,046,401 unique links voted upon.

Now, of the total 7,405,561 entries, 5,597,222 of them were upvotes. There were 31,318 users with upvotes, meaning $31553 - 31318 = 235$ users did not upvote a single post. There were also 1,583,267 links with upvotes, which implies that **given a post has been voted upon (up or down) by at least one user**, there is a $1 - (1583267/2046401) = 22.6\%$ chance that it **did not receive a single upvote**.

I also tried flipping the procedure, so that the vertices are posts instead of users, and two posts are connected if the same user upvoted both of them. It turns out we do indeed have a giant component, but also many components of various sizes. This is because we just have so many links compared to users.

*This file will actually not fully open on Microsoft Excel, as the default spreadsheet size is limited to 2^{20} =1,048,576 rows. The alphabetical list will be cut off after "c".

Size of Component	Number of Components
1	4012
2	1511
3	804
4	591
5	367
6	292
7	257
⋮	⋮
591	1
601	1
620	1
625	1
704	1
1000	1
1490103	1

Note that this giant component consists of 1490103/1583267 = 94.1% of the upvoted links.

So the links are shared to a pretty high degree, but the users themselves are a bit disjointed. Out of the 31,318 users who had upvoted at least one link, 21,122 or 67.4% of them are connected in a giant component. Of the remainder, 9839 or 31.4% of the total are completely disjoint, meaning that while they upvoted one post, nobody else upvoted it.

3 Code and Algorithm

I wrote a Java class to compute this. The algorithm iterates through the entries of the csv file once. Note that the code within the `while` loop is constant with the exception of the `getRoot()` call, which takes $O(\log n)$ as one can show that the tree induced by the

nodes is sufficiently balanced. Hence the runtime of the algorithm is $O(n \log n)$.

The quickness is based on the idea of putting our vertices into trees with a fast way of determining whether two vertices belong in the same tree. The first thought may be to hash each vertex directly to a tree, but this can be cumbersome when we need to combine two trees when two disjoint components are connected. The worst case for this scenario is $O(n^2)$ time, as each of n mappings could be changed on the order of n times. Our method is based on an efficient implementation of Kruskal's algorithm that uses a union-find approach, actually running in $O(n\alpha(n))$ where $\alpha(n)$ is the inverse of Ackermann's function and is effectively bounded by 4 [K1]. Hence this algorithm may actually run with time $O(n\alpha(n))$, which is slightly faster than $O(n \log n)$.

```
1  import java.io.*;
2  import java.util.ArrayList;
3  import java.util.HashMap;
4
5  public class DataAnalyzer {
6
7    public void read() throws Exception {
8      File f = new File("publicvotes.csv");
9      FileReader fr = new FileReader(f);
10     BufferedReader br = new BufferedReader(fr);
11     String[] strings = new String[3];
12
13     String person;
14     String post;
15     HashMap<String, SNode> map1 = new HashMap<String, SNode>();
16     HashMap<String, SNode> map2 = new HashMap<String, SNode>();
17
18     SNode sn1, sn2;
19     String s = br.readLine();
20     int lineCount = 1;
21     while (s != null && !"".equals(s)) {
22       strings = s.split(",");
23       if ("1".equals(strings[2])) {
24         person = strings[0];
25         post = strings[1];
26         if (!map1.containsKey(person) && !map2.containsKey(post)) {
27           SNode sn = new SNode(null);
28           map1.put(person, sn);
29           map2.put(post, sn);
30         } else if (!map1.containsKey(person) && map2.containsKey(
               post)) {
31           map1.put(person, map2.get(post).getRoot());
32         } else if (map1.containsKey(person) && !map2.containsKey(
               post)) {
```

```
33            map2.put(post, map1.get(person).getRoot());
34          } else { //If both are in
35            sn1 = map1.get(person).getRoot();
36            sn2 = map2.get(post).getRoot();
37            if (sn1 == sn2) {
38              //Do nothing
39            } else {
40              if (sn1.depth < sn2.depth) {
41                sn2.depth = Math.max(sn2.depth, sn1.depth + 1);
42                sn1.parent = sn2;
43              } else {
44                sn2.parent = sn1;
45                sn1.depth = Math.max(sn1.depth, sn2.depth + 1);
46              }
47            }
48          }
49        }
50        s = br.readLine();
51        lineCount++;
52      }
53
54      ArrayList<Integer> counts = new ArrayList<Integer>(map1.size())
            ;
55      HashMap<SNode, Integer> map = new HashMap<SNode, Integer>();
56      int count = 0;
57      for (SNode sn : map1.values()) {
58        sn = sn.getRoot();
59        if (!map.containsKey(sn)) {
60          map.put(sn, count);
61          counts.add(1);
62          count++;
63        } else {
64          counts.set(map.get(sn), counts.get(map.get(sn)) + 1);
65        }
66      }
67
68      int max = 0;
69
70      for (int i = 0; i < count; i++) {
71        if (max < counts.get(i))
72          max = counts.get(i);
73      }
74      int[] finalArray = new int[max + 1];
75      for (int i = 0; i < count; i++) {
76        finalArray[counts.get(i)]++;
77      }
78
79      for (int i = 0; i < finalArray.length; i++) {
80        if (finalArray[i] != 0) {
81          System.out.println("Components of size " + i + ": " +
                finalArray[i]);
82        }
83      }
84
85      br.close();
86    }
87  }
```

The utility class `SNode` is given by

```
1  public class SNode {
2    public SNode parent;
3    public int depth;
4
5    public SNode(SNode p) {
6      parent = p;
7      depth = 1;
8    }
9
10   public SNode getRoot() {
11     if (this.parent == null) {
12       return this;
13     }
14     return this.parent.getRoot();
15   }
16
17 }
```

Finally, the invoking of the class is trivial:

```
1 DataAnalyzer da = new DataAnalyzer();
2 try {
3   da.read();
4 } catch (Exception e) {}
```

The program took 30 seconds to run through the 29 MB file on a Dell Studio XPS laptop with an Intel Core i5 M 430 @ 2.27GHz.

4 Further Inquiries

A first thing to do is repeat the analysis with more recent data, to see if there has been any major shift in the reddit community's connectivity since 2010.

Second, it would be interesting to note the average "degrees of separation" between two connected reddit users based on upvoting common posts, sort of like the six degrees of separation in the real world.

And third, it would be great to know how the data was selected. Certain numbers may be inaccurate due to possible sampling error.

5 About

I've been a lurker on reddit.com for a few months now. In a computer science class that I am taking, CS 4850, "Mathematical Foundations for the Information Age," but which I always think of as "that class where you become enlightened for 50 minutes 3 times a week by mind-blowing math talks given by genius, superman, and Turing award winner John Hopcroft," we were assigned a problem having to do with large data sets that can be represented as graphs. This paper is a more fleshed out version of what I wrote for that assignment. In fact, for the original assignment I had made an error in my code and wrongly concluded that there was no giant component!

References

[H1] Hopcroft, J. and Kannan, R. *Computer Science Theory for the Information Age*. Spring 2013 Edition. Cornell University, 2012.

[K1] Kozen, D. Lectures on Algorithms, Cornell University, Ithaca, NY, 2013.

[R1] `reddit.com/r/redditdev/comments/bubhl/csv_dump_of_reddit_voting_data/`. r/redditdev, April 21, 2010. Accessed 2/6/13.