Overview

The principal objective of this project is to detect sub-communities in any social network using Girvan–Newman’s algorithm for community detection, using Brandes algorithm to calculate betweenness centrality unweighted graphs. This algorithm and their subtly different versions are limited in the data sets they are capable of processing without the problem becoming intractable though, and while it can be generalized to tackle weighted graphs, the performance takes serious hits in the process.

Data

Three Facebook data sets were chosen for this project, the first one being the UCSD Facebook data provided by the specialization developers themselves[[1]](#footnote-1), which consists of about 15k vertices and 450k edges, the second one being a smaller, hence less demanding to process data set[[2]](#footnote-2) with around 4k nodes and 90k vertices, and the last one being a much larger data set with more than two million nodes[[3]](#footnote-3), which I do not expect to successfully process at all, but I feel compelled to try anyways and simply see what happens.

Questions

The easy task that will be addressed is finding the list of second-level friends of a given user ‘x’, with second-level friends being the *direct Facebook friends of my friends to whom the user ‘x’ is not directly connected*.

The hard problem, and the main theme of the project, is to implement Brandes algorithm for edge-betweenness centrality inside of Girvan-Newman’s algorithm for community detection. After doing some humble research, I’ve learned of the progress made in approximately the last two decades in the study of community-detection algorithms, and I’m now fascinated with the possibilities. The practical implications of being able to detect sub-communities in large and relatively large networks are instinctively huge, but several applications are not obvious, or at least they weren’t to me this past couple of weeks when I first started learning about the topic.

What astonished me the most was the realization that one could separate one community into its different components in many different ways, approaching the overall coPammunity with different definitions of what the nodes in the graph would come to represent, as well as the edges. This idea implies that a single community can be deconstructed differently depending on what the analysis is focused and interested in. For example, my first idea for the capstone project, and one with which I couldn't move forward because of the difficulty implied, was to take one month of Reddit comments, where the nodes would be the users and the edges or relationships between them their similar activity on Reddit, hence allowing me to identify, or at least to try to identify, the existence of sub-communities in Reddit that would span different subreddits; that is, the sub-communities wouldn’t be defined simply by subreddit, but by user interactions, which could imply that there were groups of users interacting with each other in different subreddits, showing a different view of how the Reddit ecosystem is conformed. At the end I’ll elaborate on this idea because I intend to revisit this when I am capable of overcoming the difficulties, and hence I’d like it to be written in this project summary what the first approach was, why I failed, what I tried to do not to fail in the first place, and my ideas for the future on this topic.

Algorithms and Data Structures

For the **easy problem**, I’ll simply iterate over the said node’s neighbors v, and for each neighbor I’ll iterate over its neighbors w: if w is already a direct neighbor of v, it’s ignored, otherwise it’s added to a set. Also, for each neighbor w, I’ll have to check not only if it’s a direct neighbor of v, but also if it has already been added to the set. The point of using a set instead of a list is to make this last check quicker than using another data structure which, to check if an object is contained, many other data structures would have to iterate over their entire content until it was found, with a worst-case performance of O(n), with n being their size.

For the **hard** **problem**, first of all, I think it would a good idea to mention that my first approach was to resort to the Bron-Kerbosch algorithm, which was the first relevant thing I found when researching community-detection algorithms, which scales exponentially with the size of the graph in the worst-case scenario; I decided to move away from it because, as I understand it, cliques tend not to be unique, and the assignment of one vertex to one community or another seems to me to be quite arbitrary.

*Girvan-Newman algorithm for community detection*

1. The betweenness of all existing edges in the network is calculated.
2. The edge with the highest betweenness is removed.
3. The betweenness of all remaining edges is recalculated.
4. Steps 2 and 3 are repeated until no edges remain or until a threshold –to be defined only if necessary– is met.

If a network contains communities loosely connected by edges, meaning less connected than a sub-community itself, then all shortest paths between said communities must go through very few edges, and hence this bridges between communities will have a high betweenness score, implying that by removing such edges we are able to separate communities in an effort to detect them/differentiate them.

*Modified version of Brandes algorithm to compute edge-betweenness centrality instead of node-betweenness*

1. Initialize the betweenness centrality score of each vertex to zero
2. For each vertex in the graph:
   1. Initialize a Stack, a Queue and three Arrays: one to count the number of shortest paths from each vertex to the root of the current iteration, one to measure the distance of each vertex from the root, which equals the minimum number of edges between the vertex and the root (initially set to infinity), and an array of linked lists, where each vertex has a linked list with all the vertices that precede it in the BFS (that is, the ‘node’s parents’).
   2. Enqueue a vertex.
   3. While the queue is not empty:
      1. Dequeue a node from the queue and push the node to the stack, proceeding to perform a BFS traversal from the given root to find the shortest path to all other vertices. While performing the traversal, the distance from the root to each vertex is computed.
   4. Compute the betweenness centrality using Brandes’s dependency accumulation technique, which means that while the stack is not empty:
      1. Pop a vertex from the stack, which contains the vertices in order of non-increasing distance from the root, and for all of vertices in its linked list:
         1. While in the original version, the algorithm known as Algorithm 1 used Brandes’s technique to sum all pair-dependencies without having to do the sum explicitly because of the recursive relation of the partial sums, a slight tweak allows us to calculate edge betweenness instead of node-betweenness (Brandes, 2007: 10-11).
         2. If the node popped from the stack is not equal to the root, then the betweenness centrality score of the node popped from the stack is its score plus its delta, as defined by Brandes.

The algorithm designed by Brandes basically allows one to discover the most *important* nodes in a graph. Using Girvan and Newman’s incredibly clear explanation in layman’s terms, “if a network contains communities or groups that are only loosely connected by a few inter-group edges, then all shortest paths between different communities must go along one of these few edges. Thus, the edges connecting communities will have high edge betweenness. By removing these edges, we separate groups from one another and so reveal the underlying community structure of the graph. (Girvan and Newman, 2001: p. 3)”.

I am, however, considering implementing this version as well as one where I use Dijkstra’s algorithm, regardless of its poorer performance. Regardless, the algorithm doesn’t really change that much that I would feel compelled to detail it separately; at its heart, the concept remains the same.

It seems reasonable to mention that the difference of implementing Brandes algorithm with Dijkstra’s algorithm and a BFS traversal of the graph is the not at all minimal difference of working with weighted and unweighted edges/graph. This is obviously relevant because performance is not the only concern, and one might choose to sacrifice a piece of analysis and conclusions/answers that he or she was looking for in their work, while another person might be looking for something simpler and therefore a drop in performance this big would not only be unwarranted but unreasonable.

Algorithm Analysis, Limitations and Risks

The easy problem can be solved in polynomial time, taking O(n\*m) as I iterate over the neighbors of the central node’s neighbors, with n being the size of the central node’s neighbor list and m being the average size of the neighbor’s neighbors list.

As for the hard problem, it was already mentioned that, for the version of the problem dealing with unweighted edges, the time complexity of the overall algorithm is O(VE), with V being the number of vertices in the graph and E being the number of edges (Brandes, 2001; Bader, 2012: 13). However, when working with weighted graphs, the need for a priority queue instead of a normal queue worsens the runtime to O(n\*m+n^2\*log n) (Brandes, 2007). Now, that means that, inside the Girvan-Newman algorithm, each iteration takes O(VE), and so the worst running time of the algorithm is O(V\*E^2) for a dense graph, or O(V^3) for a sparse graph.

Besides the runtime analysis, the first obstacle is not something to be careful about in the future but rather something already encountered and, in a way, surpassed, and that is the already mentioned problem with the data set consisting of a month of comments from Reddit. Firstly, I haven’t been able to figure out how to deal with such a large data set, which becomes even scarier considering how costly these algorithms are. Whenever I chose to tackle that challenge and modify my code as I gain more experience, I might do more research and look for approximations that improve the runtime analysis.

It is hard to tell right away, before any coding and testing is done, how this costly algorithms will perform for the Facebook data, although it’s clear that it’s going to be better than with the Reddit data…this might prove more challenging than initially thought though since the Facebook data isn’t small either. For this last reason is that I have chosen to work with three data sets, two of which have a decent size but are within the recommendations of Girvan and Newman, and one which is absolutely massive, at least in comparison, and which I’m not even sure I’ll be able to work with in the end –its but a tempting idea which was probably doomed to fail from the get go.

A second thing to look for is not necessarily a potential pitfall, but rather simply the fact that these algorithms seem pretty complicated –without a doubt they are much more advanced than anything I’ve even read so far. But this challenge makes it just as exciting as it does scary.

There may be modifications to some of the data structures used to store the data as the algorithms are being written, given that I might encounter the circumstance in which a different data set allows me to improve the performance of an algorithm greatly. For example, hash sets are much more efficient than an arraylist to store data if I need to ask if an object is contained in it, a hash maps allows me to access a value associated with a known key really fast, and linked lists are really efficient when the list is expected to grow greatly and I don’t need to access the elements by index. This is being acknowledged from the beginning because of, again, how costly these algorithms are. Choosing the data structures properly is hence pivotal for performance, and a change in the algorithm as written by the authors, while it might imply having repeated data, having it stores differently might be a great way to better performance –again, even if this is at the cost of having repeated data.

One other thing is that, at this moment, the data is in a JSON file format, which is not very efficient for big files, and also the data sets I’ve used in the warm-up exercises where I used the parser made by the specialization developers were .txt files. I have to see how to go about that; basically I have to revisit the parser and see that the data will be properly loaded.

Finally, I’m sure that it will prove hard to check the correctness of the algorithms given that, as already explained, they are seriously ambitious for someone who has started programming very recently. Regardless, I expect to find a way to be rigorous, if not by myself, asking for help in the community, be that Coursera’s discussion forum for this capstone project or in other forums, such as Reddit, Stack Overflow, etc.

A comment on my attempt to work with Reddit data

* Data

The data chosen was a set of Reddit comments chosen randomly in the form of JSON file format. The comments corresponded to one month of any year after 2005; I wanted to pick a relatively recent month to work with data that was as relevant as possible, given that the Reddit community activity has grown at an impressive speed, giving me reasons to infer that by choosing a more recent month I would get more people that are seriously involved/active in Reddit, hence implying the existence of more relationships between users. The first problem encountered here was that I have a limited space for storage in my personal computer, on which I am doing the project, and secondly that the algorithms are extremely costly, and the papers referenced made it clear that a data set with more than 10k nodes would make the problem intractable (Newman, 2003: p. 14).

The data set came from the hard work of the well known Reddit user –in the Reddit community, that is–Stuck\_In\_the\_Matrix, who, with help from other members of the Reddit community according to his own statements in several comments on his own posts, consistently works to gather such comments, upload them and make them readily available to anyone who might be interested, probably for research purposes just like the one motivating this project. The link –at the moment of writing this project summary– to the data is the following: <https://files.pushshift.io/reddit/>.

Each node would represent a user, and there would be a relationship between two users if both posted a comment in the same subreddit, and the weight of the edge would increase by one by each pair of comments that this two users post in the same subreddit, here called a *match*. To be clear, if both users posted one comment in two different subreddits, then the weight of their relationship would be two; this is being repeated to make it clear that the idea was for the relationship to span across subreddits.

These data sets are more than substantial, consisting generally of dozens of millions of input lines (<https://files.pushshift.io/reddit/comments/monthlyCount.txt>). Hence, separating them to know how many of them would be nodes and how many of them would be edges is basically impossible before writing any code to process the data.

* First idea for the project’s hard problem

The idea was to implement Brandes algorithm for betweenness centrality inside of Girvan-Newman algorithm for community detection. Initially, the idea was to analyze the existence of sub-communities in Reddit, which are defined here as a *set of users that tend to be similarly active in the same subreddits*. To do this, I’d go for a variation in Brandes classical approach –obviously not a modification of mine but one researched and referenced below…–, recurring to Dijkstra’s algorithm instead of breath-first search, which would allow me to use Brandes algorithm for weighted graphs.

Instead of separating Reddit sub-communities by subreddits, as might be the first impulse of anyone, I’d attempt to see if there are sub-communities of users with shared interests that span several subreddits. Surely, since I have no way of knowing if this is the case beforehand, it is to be defined how demanding the algorithms will be in branching communities; the point is neither to be really demanding nor the opposite, but rather to be as demanding as possible without ending up with no sub-communities being found at all.

This last uncertainty shouldn’t be a concern to the reader, since it will be available information when running the algorithms on the data set how strongly connected this users that are being grouped in the same sub-community really are, so one will be able to see if the sub-communities that are found to span across several subreddits can indeed be considered to be such, or if the program had to be very undemanding.

However, the idea does depend on the existence of a lot of people that are extremely active. Because of this, the nature of how a relationship is weighed might have to change as the project progresses to make the algorithms work for a data set that contains with people that are extremely active and a lot of people that are maybe present but not really active.

* Algorithms and Data Structures

By having to use Dijkstra’s algorithm instead of BFS, with the difference consisting basically on using a priority queue instead of a queue, the running time worsens by quite a bit, from O(nm) to O(nm+(n^2) \* log n) to be specific (Brandes, 2007: 16). I won’t bore the reader with the details of the variations needed in the algorithm in the form of pseudo code. If I modify this project to be able to use such data at a later date, I’ll edit this document at the end, adding these details I’m here omitting.

I did attempt to read on some newer algorithms on the subject, which imply optimizations on the *old* algorithms or optimizations on other, newer algorithms that weren’t discussed here, finding things like Louvain’s Algorithm. I tried to read on it, starting with the original paper, but right away I noticed I was highly out of my league, at least for the time being. I do wish to revisit this, as I already said, and attempt this or another algorithm that allows me to handle such a massive data set.

References

* Brandes, Ulrik, A faster algorithm for betweenness centrality; Journal of Mathematical Sociology 25, 163–177 (2001).
* Brandes, Ulrik, On Variants of Shortest-Path Betweenness Centrality and their Generic Computation; Social Networks 30(2): 136-145 (2008).
* Clauset, A., Newman, M. E. J. and C. Moore, Finding community structure in very large networks; Physical Review E, 70:066111 (2004).
* Gjoka, Minas, Kurant, Maciej, Butts Carter T. and Athina Markopoulou, Walking in Facebook: A Case Study of Unbiased Sampling of OSNs; Proceedings of IEEE INFOCOM '10; San Diego, CA (2010).
* Girvan, M. and M. E. J. Newman, Community structure in social and biological networks, Proc. Natl. Acad. Sci. USA 99, 7821–7826 (2002).
* Girvan, M. and M. E. J. Newman, Finding and evaluating community structure in networks, Phys. Rev. E 69, 026113 (2004).
* Green, O., McColl, R. and D.A. Bader, A fast algorithm for streaming betweenness centrality; in *Privacy, Security, Risk and Trust (PASSAT), International Conference on and 2012 International Confernece on Social Computing (SocialCom)* (2012).
* Newman, M. E. J., Fast algorithm for detecting community structure in networks; Department of Physics and Center for the Study of Complex Systems, University of Michigan; Ann Arbor, MI (2003).
* Newman, M. E. J., Scientific collaboration networks: II. Shortest paths, weighted networks, and centrality; Phys. Rev. E 64, 016132 (2001).

1. See Amanda L. Traud, Peter J. Mucha, and Mason A. Porter, Social Structure of Facebook Networks, arXiv: 1102.2166. [↑](#footnote-ref-1)
2. See J. McAuley and J. Leskovec. Learning to Discover Social Circles in Ego Networks. NIPS, (2012). Data downloaded from: <https://snap.stanford.edu/data/egonets-Facebook.html> on July 29th, 2018. [↑](#footnote-ref-2)
3. See:

   @inproceedings{gjoka10\_walkingfb,

   author= {Minas Gjoka and Maciej Kurant and Carter T. Butts and Athina Markopoulou},

   title= { {Walking in Facebook: A Case Study of Unbiased Sampling of OSNs} },

   booktitle = {Proceedings of IEEE INFOCOM '10},

   address = {San Diego, CA},

   month = {March},

   year = {2010}

   }

   Downloaded from: <http://odysseas.calit2.uci.edu/wiki/doku.php/public:osn_datasets#facebook_social_graph_-_breadth_first_search> on July 25th, 2018. [↑](#footnote-ref-3)