CS224W Final Report: Study of Communities in Bitcoin Network

Jim Hu
Group 40

December 10, 2013 9AM EST

1. Abstract

Bitcoin is the world's first person-to-person decentralized digital currency allowing financial transactions to be made through the Internet. It offers several compelling advantages to traditional country currency in that it eliminates the bank/clearinghouse to lower fees, it is usable across the globe, accounts cannot be frozen, and there are no arbitrary legal limits. Businesses are able to easily accept this global currency, and individuals can make anonymous payments and transfers anywhere in the world. The Bitcoin network is relatively new with the first proof of concept published in 2009. Since then it has grown to a marketplace of around $1.5 billion USD. The sample data studied contains ~4 million unique nodes (representing parties in the payment transaction) and ~25 million edges (representing payments). The primary algorithm used is described by Clauset et al. in "Finding community structure in very large networks." I use this algorithm to identify communities in the Bitcoin network, and attempt to decipher meaning behind these communities.

2. Introduction

In "Finding Community Structure in Very Large Networks", Clauset et al. defines a heirarchical algorithm to find communities that runs in O(*md log n*) time where *n* is the number of vertices, *m* is number of edges, and *d* is the depth of the dendrogram representing the community. This speed allows the algorithm to be utilized on large networks. Modularity of a graph identifies how well communities are defined by comparing the number of edges within a community to edges between communities. The algorithm takes a "greedy" approach to maximize modularity where each iteration is focused on joining parts of the network that result in maximal modularity increase. Modularity Q is defined as:

$$Q = \frac{1}{2m}\sum_{ij}\left[A_{ij} - \frac{k_i k_j}{2m}\right]\delta(c_i, c_j),$$

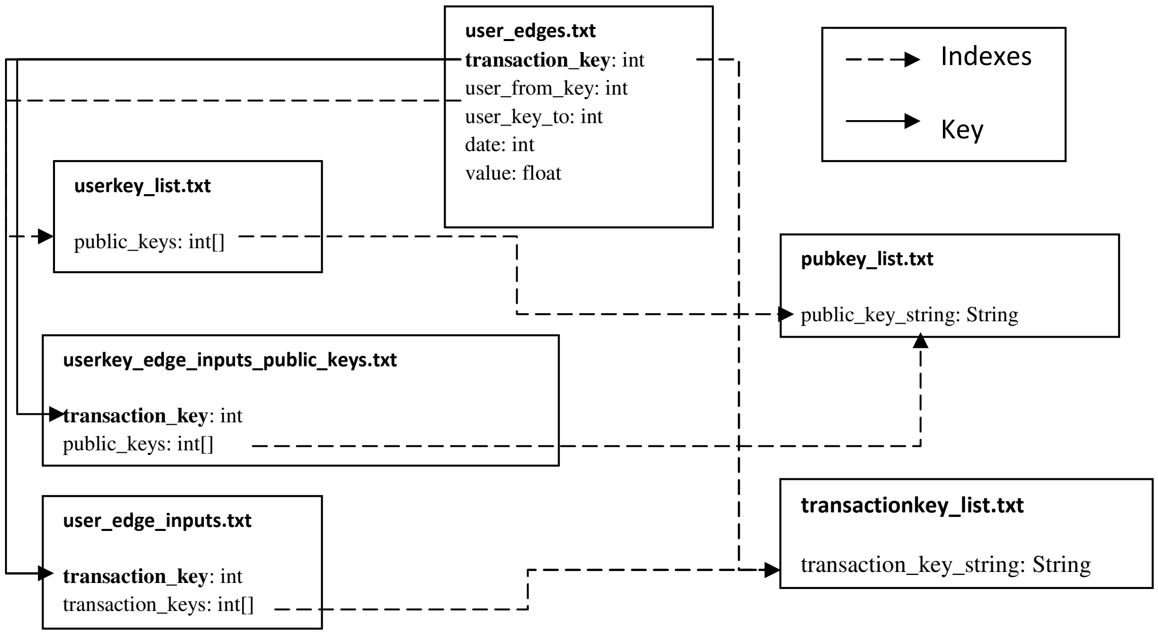
$$where\ m\ is\ number\ of\ edges, A_{ij}\ is\ 1\ if\ node\ i\ and\ j\ are\ connected\ and\ 0\ otherwise;$$
$$k\ is\ the\ degree\ of\ the\ node;$$
$$\delta(i,j) = 1\ \ if\ i = j, meaning\ the\ nodes\ are\ in\ the\ same\ community, and\ 0\ otherwise$$

Typically, a Q value above 0.3 indicates a significant community in the graph. The "greedy" approach is optimized by tracking modularity values in order to eliminate calculations that would have needed to be performed with the adjacency matrix to calculate modularity. This paper is the basis for the algorithm that I implement for community detection on the Bitcoin network. One enhancement to this algorithm that I would like to implement is to weight the edges as described by Newman in "*Analysis of Weighted Networks.*" The modularity equation lends itself well to weights. For the adjacency matrix A, instead of storing 1 to indicate an edge, the edge weight is stored.

3. Method

3.1. Bitcoin Data Set

The Bitcoin Data Set is obtained from the work of Ivan Brugere at the Laboratory for Computational Population Biology, University of Illiois at Chicago (http://compbio.cs.uic.edu/data/bitcoin/). The dataset contains a set of files as shown below to describe the network. The datasets I will use cover two blockchains – which are public ledgers of transactions – from Jan 6, 2013 and Apr 7, 2013. The Jan 6, 2013 blockchain consists of 4.86 million nodes with 25 million edges, and the Apr 7, 2013 blockchain consists of 6.3 million nodes with 37.4 million edges.



The graph edges are obtained from “user_edges.txt” where the user_from_key and user_key_to columns point to line rows in the “userkey_list.txt.” Each line in the “userkey_list.txt” represents a unique graph node. Each line may have multiple public key strings which are the addresses that send and receive Bitcoins. The user_edges file also maps to a transaction key, which can be used to look up the Bitcoin transaction key string from the “user_edge_inputs.txt” file. These public and transaction keys strings can be searched at https://blockchain.info for further details about the chain.

3.2. Algorithm

The data model and algorithm are implemented in Python and using the SNAP.PY library. The network is modeled as a undirected graph. The nodes represent users, and edges are Bitcoin transactions. Node id’s are the line number from the “usekey_list.txt” file. The implementation will also track the number of times a pair of users has conducted transactions which will be the edge weight. There are several key data structures used:

- Q – is a dictionary where each user is mapped to other users where there was a transaction. The *key* represents a user (initially all users are a *key*), and the value is a list of all parties that the user has had a transaction with. The list values are represented as a list of [Q value, node id, neighbor id]. The
$$\text{initial Q value} = 2 * \frac{1}{2*numEdges} - \frac{deg_{user(key)}deg_{neighbor}}{(2*numEdges)^2}$$
- H – is a list that stores the [Q value, key, neighbor] pair with the highest Q value of each key in dictionary Q
- a – is a list that stores $\frac{deg}{2*numEdges}$ for each key in Q
- QResult – a list tracking the overall modularity value. Each index represents an iteration and its value is the modularity value at the end of that iteration. QResult[0] is initialized as $\sum_i -a_i^2$.

After initialization as described above, each user is essentially its own community. Each iteration of the algorithm seeks to greedily increase the modularity value stored in QResult. The implementation runs until there is only one community remaining:

1. Find the largest Q value for items in H of format [Q val, key, neighbor]
2. Retrieve Q[key i] and Q[neighbor j] which returns a heap of neighbors for each of those nodes, and join the two communities. The logic for the join is as follows. For each neighbor k of i and/or j:

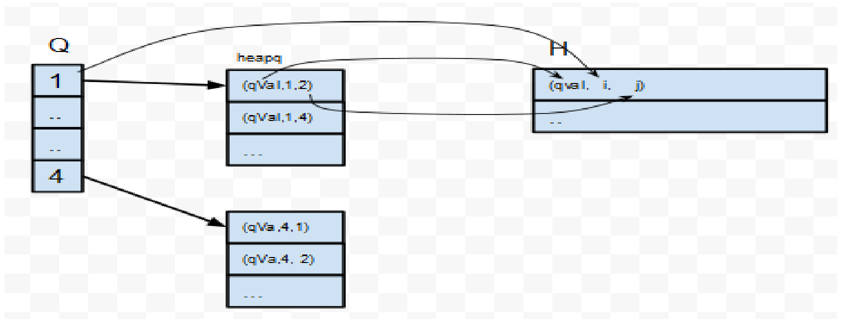
    a. If k is a neighbor to both i and j, then update Q[j][k] += Q[i][k]
    b. If k is a neighbor to i but not j, then update Q[j][k] = Q[i][k] - $2a_ja_k$
    c. If k is a neighbor to j but not i, then update Q[j][k] -= $2a_ia_k$

    We also update the corresponding Q[k][j] entries. These steps merge community i into community j, so we then delete Q[i] and update Q[k][i] entries.
3. Update H to reflect the new max values for changes made to Q in step 2.
4. Update list a, where a_j += a_i and a_i=0.
5. Update QResult[iteration] = QResult[iteration-1] + largest Q value from step 1

3.3. Implementation Details

The main logic described above is contained in "proj.py" which outputs "output.csv" and "consolidated.csv" files. Each line in the output files represents an iteration and "output.csv" shows the modularity value, and "consolidated.csv" shows the two communities being consolidated.

The core data structure, MaxHeap class, used to manage the various heaps is contained in "heap.py" This data structure is used to manage both the community heaps store in Q as well as H. Below is graphical representation of the data structures.

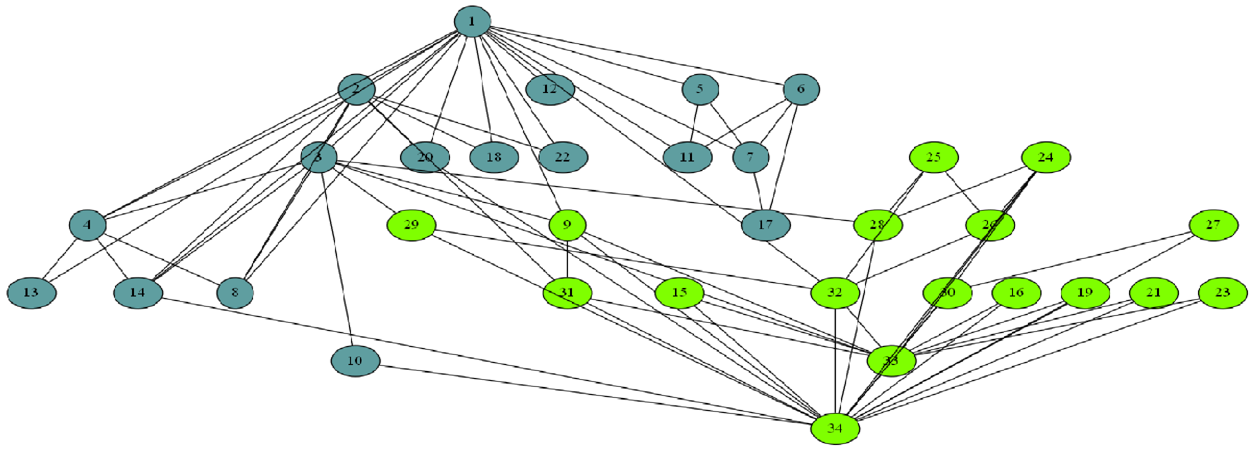


The MaxHeap object uses Python's heapq implementation, but also adds two indexes. One index uses (i,j) as the key to find the value (qval,i,j) data in the heap, where i and j are two nodes. The other index takes i as the key to find corresponding (qval,i,j) in the heap. These indices allow the code to look up the data tuples in O(1) time instead of O(log |i|) or O(log|j|) time, where |i| is the number of neighbors for i and |j| is the number of neighbors to j. An insert into MaxHeap will take O(log n) time, but delete will be O(1) because the code uses the index to find the tuple to 'soft' remove it – it sets the data to a value to indicate it is 'removed' but will remain in memory until it is popped off.

Thus to perform one iteration of step 2 of the algorithm, all of the neighbors of i, designated as |i|, needs to be merged into |j|. All of j's values are updated, and after update python's heapify call is made to rebuild the heap invariant, which takes O(|i|+|j|) time. Additionally, for each k neighbor of i or j, the code inserts the updated values into the k MaxHeap, which takes O(log |k|), and soft removes the old values in O(1) time. Finally, the old values in H are soft removed, and the new values are inserted, which takes at most O(log (|i|+|j|)). While this approach is faster than the sparse matrix, it is not as optimal as the algorithm presented by Clauset et al. I chose ease of implementation using python's heap library instead of a custom implementation.

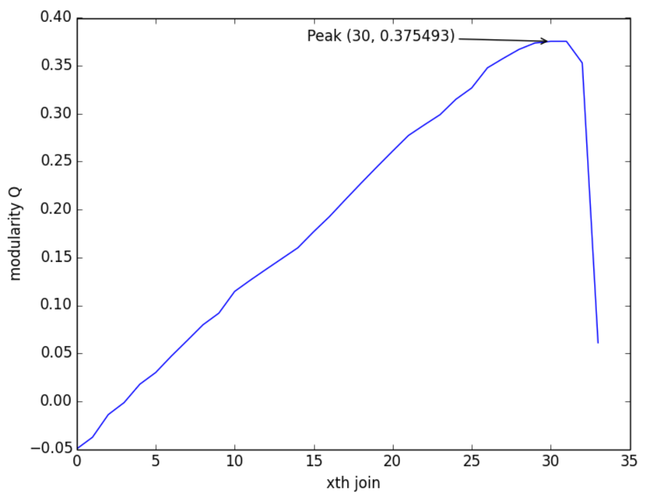
3.4. Verification with Karate Club Network

I used the karate club network to test the accuracy of my implemented algorithm (data obtained from http://networkdata.ics.uci.edu/data.php?id=105). The graph below shows the two actual communities in the club.

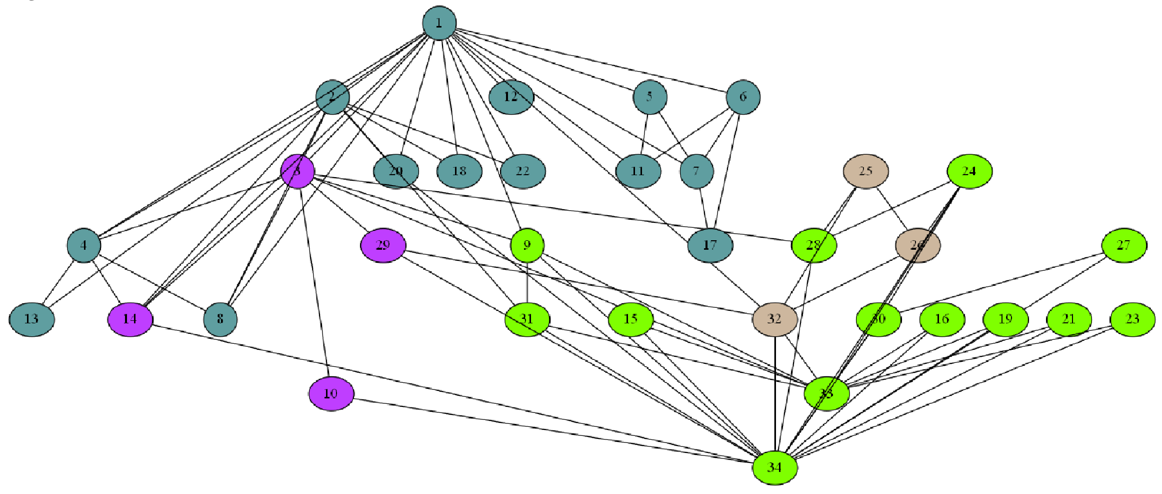


Karate Club

Below is the modularity graph as communities are joined over time. The peak Q of 0.375 is obtained at iteration 30, which is a solid indication of community structure.



At iteration 30, I obtain the following communities, which is very close to the real-life split shown above. We can see that the two main communities center on 1 and 34, however the algorithm misses a few members.



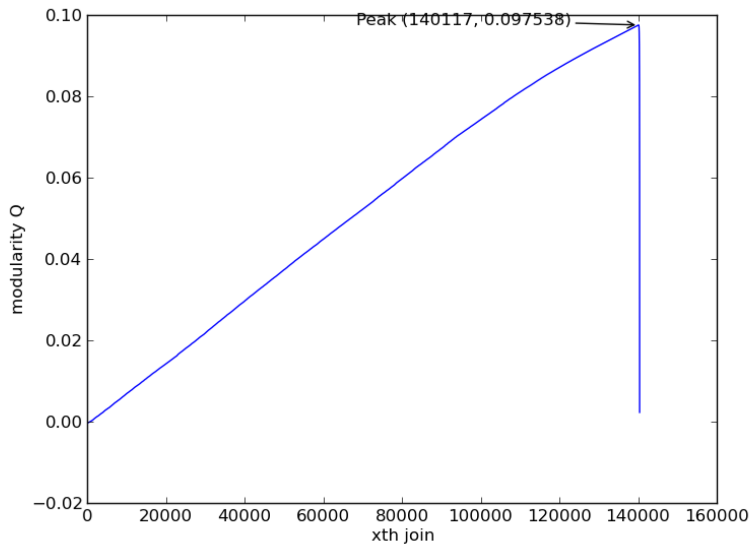
Karate Club

4. Bitcoin Transaction Network

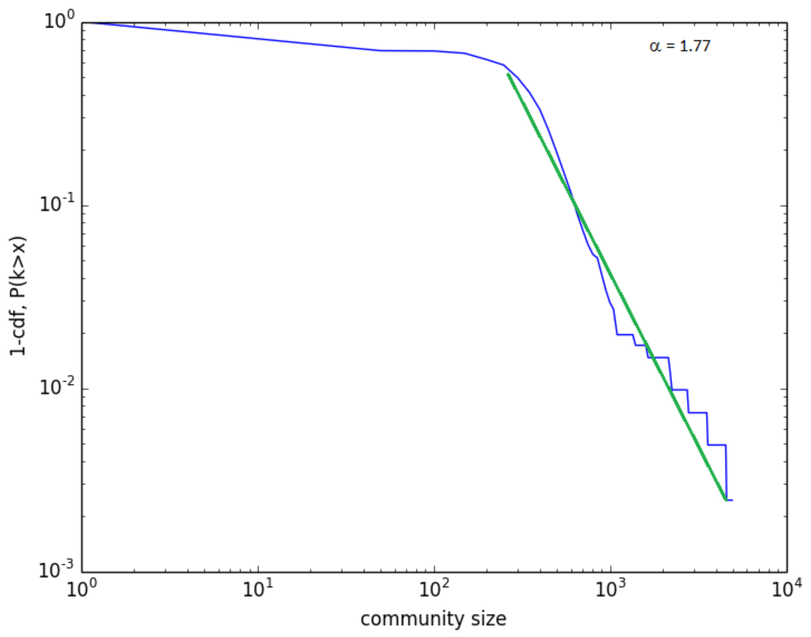
Although the algorithm speed is comparable to that presented by Clauset, et al, processing the entire Bitcoin chain remained a challenge. To obtain results in a reasonable time, I used the Jan 6, 2013 dataset with 4.86 million nodes and reduced it to a manageable size as follows:

- Find the SCC of the original graph
- From the SCC, remove nodes that have less than 10 degrees
- Finally, remove any nodes that may no longer have any neighbors

This reduction gives a graph of ~140,000 nodes and ~1.9 million edges of more frequent Bitcoin users, which is used for further analysis. The graph below shows the results of the community analysis where the peak modularity Q of 0.097 is found on join 140,117. Although Q is less than 0.3, which would be a strong indicator of communities, any positive value indicates a non-random community structure.



As further verification of community structure, Clauset et al. notes that their study of Amazon, as well as other network studies have found that the community size distribution follows a power-law for some portion of the graph. Below I plot the cumulative distribution of community sizes and found a power-law fit with α=1.77 over a significant portion of the graph, which helps to further confirm the validity of the communities.



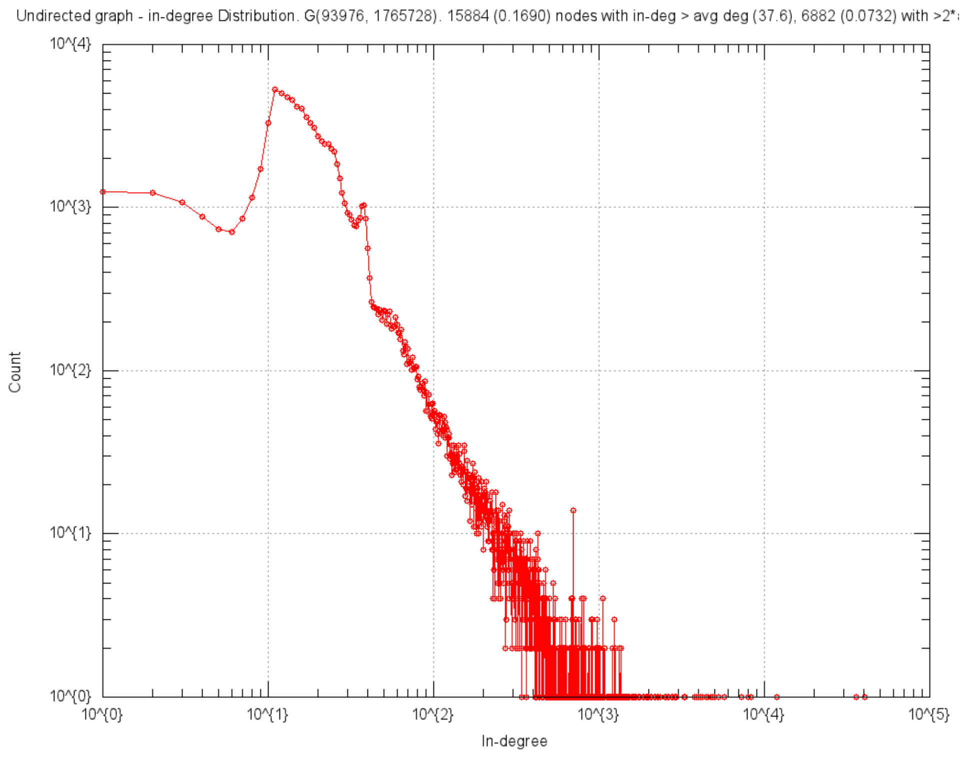
I attempted to improve on the modularity by including edge weights indicating that two parties had multiple transactions. However, I found that of the ~25 million transactions in the dataset, only ~350k parties had more than one transaction with each other. Additionally, none of these users were included in the ~1.9 million edge subset that was analyzed.

While the analysis indicates the presence of communities, it is not clear what the communities represent due to the anonymous nature of Bitcoin. The top 5 communities by size are (degree, community id):

(6822, 30), (4584, 6), (3568, 14), (2786, 3048046), (2208, 13)

One hypothesis I had is that the communities may represent geographic location (i.e. transactions within a country or continent). I took random samples from each of these top communities and looked up the related transaction key (from transactionkey_list.txt mentioned in section 3.1), and then search by these keys on https://blockchain.info. Within a community, the geographic origination varied greatly across continents.

As further analysis, I looked at the degree distribution of the entire bitchain as shown in the graph below. There is almost an exact one-to-one match between the highest degree node and the community found (i.e. each high degree node forms its own community). However, there are a few discrepancies indicating that multiple high degree nodes got merged.



5. Conclusion

The analysis demonstrates there likely exists community structures in the Bitcoin network. However, the graph was greatly reduced down to accommodate time limits. For further work, it would be interesting to look for communities across the entire chains. The meaning of these communities remains a mystery. Another hypothesis I have is that the communities simply represent large businesses that accept Bitcoin, which would explain the one-to-one correlation between communities and high degree nodes. However, my findings show instances where high degree nodes were merged into the same community. It would be interesting to investigate this further.

6. References

[1] M.E.J. Newman, M. Girvan (2003). "Finding and evaluating community structure in networks". *Phys. Rev*. E 69, 026113, 2004.

[2] A. Clauset, M.E.J Newman, C. Moore (2004). "Finding community structure in very large networks". *Phys. Rev*. E 70, 066111, 2004.

[3] M.E.J. Newman (2003). "Fast algorithm for detecting community structure in networks". *Phys. Rev*. E 69, 066133, 2004.

[4] J.-P. Onnela, J. Saramaki, J. Hyvonen, G. Szabo, D. Lazer, K. Kaski, J. Kertesz, A.L. Barabasi. "Structure and tie strengths in mobile communication networks". PNAS, 2007

[5] J.M. Kumpula, J. Saramäki, K. Kaski , and J. Kertész (2007). "Limited resolution in complex network community detection with Potts model approach". *The European Physical Journal B* 56 (1): 41–45

[6] M.E.J Newman, Analysis of weighted networks. Preprint cond-mat /0407503 (2004).

[7] http://compbio.cs.uic.edu/data/bitcoin/