# Predicting Bitcoin Transactions from Blockchain Records Through Recursive Clustering

**Crystal Qian**
Princeton University
`cqian@princeton.edu`

**Karen Ouyang**
Princeton University
`kouyang@princeton.edu`

## Abstract

Bitcoin, a virtual cryptocurrency, is an attractive payment system in part due to the strong built-in anonymity of transactions. However, Bitcoin transactions are publicly recorded in the blockchain, offering a potential means to derive relationships between addresses. Training on a year's worth of public blockchain data, we present a method of predicting transaction histories using recursive clustering and a hyperparameterized threshold. Our clustering approach reveals latent communities in the blockchain transactions with reasonable modularity, showing a need for improved practices in keeping transactions anonymous.

## 1 Introduction

Bitcoin transactions between addresses are recorded in a public blockchain. Because these transactions strive for anonymity, there should be no latent patterns or structures within the blockchain. Given this, finding relationships in the blockchain could indicate security and anonymity weaknesses in the Bitcoin infrastructure as well as in user practices.

We train a model on Bitcoin transactions in the blockchain from March 2012 to 2013, containing $55,000$ blocks and ~3 million interactions between unique address pairs (found in `txTripletsCount.txt`), and test our predictions on $10,000$ potential sender-receiver transactions (found in `testTriplets.txt`).

We present a model for predicting transaction histories through recursive clustering on a hyperparametrized threshold, evaluated to be significantly better than random guessing based on a robust set of metrics. The formulation of this model suggests that additional practices and techniques are needed to improve Bitcoin anonymity.

## 2 Methods

### 2.1 Evaluation metrics

**Motivation for robust metrics:** Sparsity is a substantial challenge with this training dataset; our training data contains $444,075$ addresses with $3,348,026$ pairs of transactions; this is a sparsity rate of $1.7 \cdot 10^{-5}$ in relation to the $444,075^2$ unique sender-receiver address combinations. As a result of the limited instances of positive (existent) training transactions, this data is unbalanced and largely skewed towards classification of negative (non-existent) transactions. Taking advantage of this sparsity, we can naively predict 0 for all transactions in the dataset. This results in an accuracy paradox, since the 0.9 accuracy is a result of unbalanced class representation. Rather than use accuracy, we use the area under the ROC curve and the $F_1$ score (a measure of precision and recall) to evaluate the relative success of our solution.

**ROC area under the curve (AUC) calculation:** The ROC (receiver operating characteristic) curve plots the false positive rate on the $x$-axis against the true positive rate on the $y$-positive rate at various thresholds. The area under the curve is the probability that the model will assign a higher rank to a randomly-selected positive sample than a randomly-selected negative sample. A perfect model has an AUC of 1, and a model that is equivalent to random guessing has an AUC of 0.5.

**$\mathbf{F_1}$ score calculation:** Let $TP$ be the true positive count (predicted = 1 and actual = 1), $FP$ be the false positive count (predicted = 1 and actual = 0), $TN$ be the true negative count (predicted = 0 and actual = 0), and $FN$ be the false negative count (predicted = 0 and actual = 1). We calculate precision $p = \frac{TP}{TP+FP}$ and recall $r = \frac{TP}{TP+FN}$. The $F_1$ score is defined as $F_1 = 2 \cdot \frac{pr}{p+r}$.

## 2.2 Approach

Our algorithm for predicting transactions has three main parts, as described below.

**Community clustering:** We partition the $444,075$ addresses into community clusters of varying size, using Louvain's algorithm for community detection on an undirected graph (address-to-address, not sender-to-receiver) with edge weights equal the number of transactions between any two addresses. Our heuristic predicts a transaction to occur if the sender and receiver address are in the same cluster. Community clustering performs significantly better than random guessing.

**Grid search for clustering threshold:** Since the Louvain method is agglomerative, that is, "bottom-up," with each node starting in its own cluster and clusters being merged to form larger clusters, we cannot reduce the size of a cluster once it is formed. Our heuristic predicts transactions between addresses within the same cluster to occur, so large cluster sizes will result in false positives and poor precision. We improve this by recursively clustering the $t$ largest clusters, where $t$ is a hyperparameterized threshold found through cross-validation and grid search. Using the number of subclustered clusters is a more robust threshold than minimal cluster size, since the latter is prone to overfitting to specific cluster sizes.

**Recursive subclustering:** We then run another instance of the Louvain algorithm on $t$ of the largest clusters to divide them into more manageable and informative subclusters. We wish to minimize the number of subclusters; if these subclusters would have made significant differences initially, they would already be in separate clusters with cohesive modularity. We thus select the final partitioning (fewest clusters) generated by Louvain's algorithm. Note that in the initial clustering, we take the algorithm's initial partitioning to maximize the number of clusters, because we want the largest number of clusters with reasonable modularity. (More on Louvain algorithm follows in Section 2.3.)

## 2.3 Spotlight: Clustering

**Motivation:** An intuitive way to predict transactions is to perform a matrix completion (perhaps a variant of the Netflix recommendation system) on an adjacency matrix of addresses (where the value at $i, j$ is the number of transactions from sender $i$ to receiver $j$), and use a hyperparameterized threshold on the completed matrix to determine if transactions occur. However, the dataset is so large that an adjacency matrix mapping all addresses to each other is not computationally feasible: a $440,075^2 = 197,202,605,625$ cell grid, even if each entry is a single bit to encode an unweighted edge, would still take at least 24 GB to store, excluding metadata; if we store the edge weights in a single byte, we would require almost 200 GBs of memory. We must cluster our data to make data manipulation a computationally-realistic goal and to prevent overfitting.

**Approach:** One method to clustering data is to maximize *modularity*. Modularity is a measure of the partitioning of graph nodes into separate clusters, also known as *modules* [4]. A cluster contains densely interconnected nodes, and nodes in different clusters are sparsely connected. By modeling the training dataset as a network graph, we may perform clustering algorithms on our data. Each address is a separate node, and each transaction is a directed edge from the sender to the receiver.

We investigated two possible clustering algorithms. The Girvan-Newman algorithm, calculates the *betweenness* of an edge, indicating the number of shortest paths between pairs of nodes that contain this edge. Edges between clusters have higher betweenness; the algorithm iteratively removes these edges to separate the clusters. This algorithm, however, runs in $O(n^3)$ time for sparse networks of $n$ vertices [3], making it impractical for datasets of our scale.

We decided to implement the Louvain method for modularity clustering [1], which takes linearithmic time and is more appropriate for large datasets. Each node is initialized in its own cluster of size 1; the algorithm then aggregates each node with its neighbors while maximizing modularity gain. These groupings are then transformed into single nodes, yielding another graph. This process is repeated until the modularity reaches a local maximum. Note that each iteration generates a partitioning with fewer clusters than the previous one. The partitions at each iteration can be represented using a dendrogram, which is a tree diagram that illustrates the arrangement of clustered nodes.

As an example, consider the three-layer dendrogram in Figure 1. The 0th layer represents each node in separate clusters (*a*, *b*, *c*, *d*, *e*, *f*, *g*), the first layer represents three clusters (*ab*, *cde*, *fg*), the second layer represents two clusters (*abcde*, *fg*), and the third layer represents a single cluster (*abcdefg*).

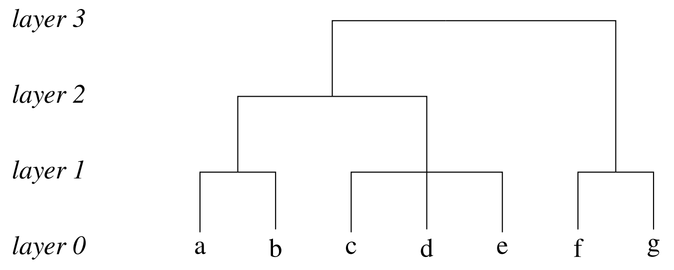


Figure 1: Example dendrogram with three layers.

To run the Louviain method on our dataset, we read the raw data as a `scipy.sparse.coo_matrix` (as storing the entire 2D matrix is computationally inefficient), convert the sparse matrix into a `networkx` graph, and use `python-louvain` to generate a dendrogram. We consider edges undirected, thus losing the relational nature of our data, because the `python-louvain` module does not support directed edges, and re-implementing the Louvain method for modularity clustering in the case of directed edges is outside the scope of our project. However, it is worth noting that preserving the directional sender-receiver relationships could potentially result in better prediction. The output dendrogram contains four layers, each of which represent a smaller (fewer clusters with higher modularity) partitioning of the network than the previous layer.

We evaluate our clustering by calculating the modularity of each partition. Modularity measures the fraction of edges that connect nodes within the same community minus the expected number of such edges in a graph with the same nodes but randomly-generated edges; thus, a modularity of 0 indicates that the number of within-cluster edges is no better than random [4]. The value of modularity lies in the range $[-\frac{1}{2}, 1)$ [2]. As shown in Table 1, our cluster sizes decrease dramatically as we increase in layer depth, but modularity improves only slightly. This indicates that the first partition clusters the address space reasonably; additional filtering/refinement of the clusters will not significantly strengthen cluster interconnections.

| **Partition** | **Clusters** | **Modularity** |
|---|---|---|
| 1 | 7116 | 0.4245 |
| 2 | 254 | 0.4400 |
| 3 | 125 | 0.4406 |
| 4 | 115 | 0.4406 |

Table 1: Layers of the generated dendrogram using the Louvain clustering method.

The “best partition” is generally considered the partition with the highest modularity, as this is the optimization metric in the Louvain method. This is the last partition in Table 1, with 115 clusters. In the last partition, however, the largest three clusters (of 115 total) contained over 75% of the addresses (see Figure 2b), resulting in very skewed cluster sizes. Furthermore, this partitioning aggressively groups nodes without significantly improving modularity, as indicated above.

Because decreasing the number of clusters only marginally improves modularity, we also take the number of clusters into account when choosing the ideal partition for our use case. In particular, the graph is very sparse, so we should be cautious of clustering all ~440, 000 addresses into a small amount of large clusters. To this end, we decided to use the topmost layer, with 7116 clusters, for a wider distribution of addresses across clusters (see Figure 2a). Though the largest cluster is consistent between the two partitions, the remainder of the addresses are more uniformly divided across smaller clusters. This suggests a closer relationship between nodes in the same cluster.

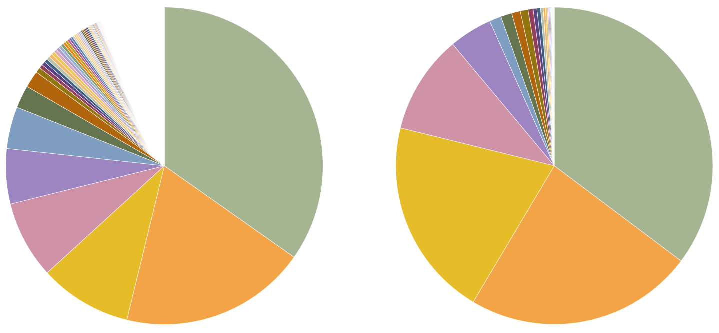

(a) partition = 1, clusters = 7116 (b) partition = 4, clusters = 115

Figure 2: Sizes of clusters for different partition levels.

## 3 Results

### 3.1 Community clustering

Table 3 shows that clustering significantly improves accuracy, AUC, and $F_1$ score over random guessing and improves as the number of clusters increase. We decide to use partition 1 (containing 7116 clusters) moving forward as our initial partitioning.

| **Strategy** | **Accuracy** | **AUC** | **$F_1$ Score** |
|---|---|---|---|
| Predict randomly | 0.5090 | 0.5170 | 0.1770 |
| Community cluster (partition 4) | 0.4666 | 0.4663 | 0.1487 |
| Community cluster (partition 3) | 0.4666 | 0.4663 | 0.1487 |
| Community cluster (partition 2) | 0.6434 | 0.5392 | 0.1866 |
| Community cluster (partition 1) | 0.7495 | 0.5770 | 0.2240 |

Table 2: Clustering strategies and their performance.

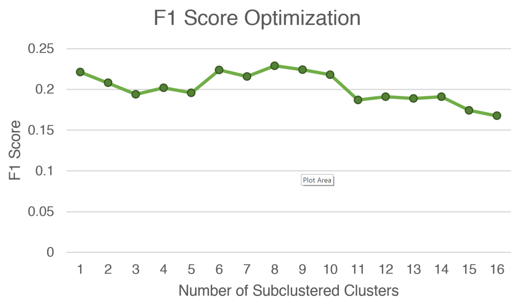
### 3.2 Grid search for cluster size threshold

To increase precision, we further subcluster the $n$ largest clusters, as these clusters contain too many addresses to be informative. We hyperparameterize $n$, running a grid search to determine the optimal value: first, we use `sklearn.model_selection.train_test_split` to repeatedly split `testTriplets.txt` into a validation and test set, each of size 5000; next, we determine the optimal cluster cutoff based on the validation set, and use that parameter to predict on the test set. We sample 20 validation/test sets for robustness; the difference across each iteration is around a thousandth of a score point, which makes sense given that the sparsity of the test set makes it relatively resistant to overfitting in cross validation and our heuristic.

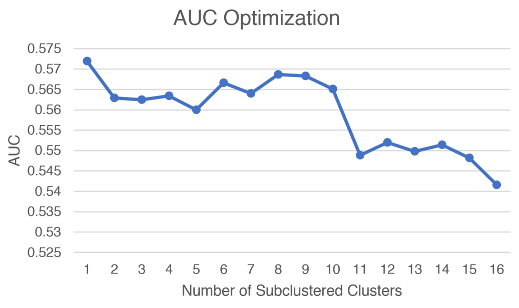
As shown in Figure 3, we find a local maximum for both $F_1$ score and AUC at around $n = 8$. We also observed the change in accuracy over $n$, but due to the accuracy paradox the accuracy stayed largely steady and was thus not very informative.

### 3.3 Recursive subclustering

The results shown for recursive subclustering in Table 3 are calculated by averaging the scores over 40 iterations of the cross-validation step described above. We stop subclustering at recursion depth



(a) $F_1$ score



(b) Area under the curve

Figure 3: Different metrics over the number of clusters to subcluster.

2 because we want to preserve a reasonable modularity value at each clustering step; the heuristic of predicting transactions within clusters will decrease in precision as clusters lose coherency.

| **Strategy** | **Accuracy** | **AUC** | **$F_1$ Score** |
|---|---|---|---|
| Predict 0 | 0.9 | 0.5 | 0 |
| Predict randomly | 0.509 | 0.5174 | 0.177 |
| Community cluster on partition 1 | 0.7495 | 0.5770 | 0.2240 |
| Recursive subclustering on partition 1 | 0.8598 | 0.5891 | 0.2289 |

Table 3: Baseline strategies and their performance.

## 4 Discussion and Conclusion

We trained and tuned a model on blockchain data to predict potential sender-receiver Bitcoin transactions. If Bitcoin interactions were truly anonymous, such predictions should have performance comparable to a random guesser at best. We were, however, able to achieve an accuracy of 0.8598 (which is not an entirely useful metric given the accuracy paradox), area under the curve of 0.5891, and $F_1$ score of 0.2289. Each of these metrics perform significantly better than a random guesser, exposing a latent community structure within the blockchain transactions.

Our approach partitioned the addresses into different community clusters based on shared interaction. Since Bitcoin recommends that users use separate addresses for each transaction, an ideal blockchain would have few occurences of each address, resulting in a sparse clustering. However, we were able to partition addresses based on their transactions with a significant clustering modularity of ~0.44. This suggests widespread improper usage of Bitcoin, such as storing bitcoins on paper wallets. Only $\frac{4092}{\text{\textasciitilde}3{,}000{,}000}$ transactions involve unique addresses (proper usage); the majority of addresses (about 63%) reside in one of three large clusters (see Figure 2).

Another explanation for these clusters is the use of Bitcoin mining pools, where Bitcoin miners pool their resources, usually at a single address (though decentralized pools do exist). This would indicate an interesting usage pattern of Bitcoin, where the vast majority of users primarily participate in mining pools using the same address, while transactions between users are the minority.

## 5 Future Work

We present this relatively successful prediction method with interesting implications on Bitcoin security; however, it is worth noting that our results are based on a test set of size $10,000$, which is significantly smaller than the space of all possible transactions. We were not able to split the given training data into additional validation and testing sets for more robust, cross-validated results without knowledge of the time series. Our validation set would tune more accurately if exposed to more transactions; additional work using our approach would benefit from verification of our results on a much larger test set.

## References

[1] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008.

[2] Ulrik Brandes, Daniel Delling, Marco Gaertler, Robert Gorke, Martin Hoefer, Zoran Nikoloski, and Dorothea Wagner. On modularity clustering. *IEEE Trans. on Knowl. and Data Eng.*, 20(2):172–188, February 2008.

[3] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, 2002.

[4] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69:026113, Feb 2004.