Finding Hidden Communities in HackerNews

Graph Mining - WS 2017/18 - Project Report

Willi Gierke
Fabian Paul

ABSTRACT

HackerNews[1] is a news site focusing on entrepreneurship and computer science. Users are able to post stories or comment on these. Unlike social networks, users can not add friends or join groups. We were interested in whether it is still possible to find communities just based on the information which stories users commented.

ACM Reference Format:
Willi Gierke and Fabian Paul. 2018. Finding Hidden Communities in HackerNews. In *Proceedings of Graph Mining Project, Potsdam, February, 18 – 2018 (GM 2017),* 4 pages.
https://doi.org/

1 DATA PREPARATION

The dataset we used was published on reddit[2] and contains a complete dump of all stories and comments of HackerNews between its launch in 2006 and July 2017 in CSV format. The dataset contains 9.9M comments and 2.1M stories posted by 400k users.

Due to the amount of data, we could not clean and preprocess all CSV files every time we wanted to build a graph. We decided to read all the files once and store them in a SQLite[3] database on disk. With this approach, we also avoided mixed column types because we specified our types first in a pandas Dataframe and saved it as CSV file afterwards. Unfortunately, not all of the CSV files shared the same schema so we had to select our important columns first before inserting them into the database.

It was important for us to select a subset of comments because of the increasing complexity of building the graph and computation time of all used algorithms. Especially the SQL projection to select just the columns we needed to reduce the amount of main memory we were supposed to allocate drastically.

2 GRAPH BUILDING

We have built a tool to quickly extract all the needed information from the database and build the graph based on it. It is possible to build graphs on any arbitrary subset of the data. Our graph building approach is iterating over the extracted data, looking for every comment for its parent comment or story. Since we have to traverse the whole subset in every line, the resulting runtime is in $O(n^2)$ where n is the number of rows.

In the end, each node represents a user. An edge between two nodes is drawn if they have commented the same story. In the case of users sharing multiple commented stories we just draw one edge to avoid duplicates. The graph, therefore, is undirected and unweighted. In the following, we will use two different graphs based on different subsets.

3 DESCRIPTION OF THE ALGORITHMS

The following section gives a brief overview what kind of community detection algorithms we used. We will also give a short explanation why we decided to use these algorithms and how they work. Allowing a good comparison between those we chose three algorithms which try to optimize the modularity while applying their procedures. Our choices were also strongly influenced by the implementation given by the igraph[4] framework because it is one of the few frameworks which offers all implementation in fast C code. Fortunately, we could use the provided python bindings to keep our analysis simpler. Specifically, we looked for algorithms which offer quick results and we do not have to wait multiple hours to run the community detection algorithms on our graphs.

3.1 Fast Greedy

The first algorithm we selected was developed by Clauset et al. [2] in 2004. This algorithm was specially developed for the physics community due to its need for analyzing big networks. It provides a run time of $O(m * d * log(n))$ hereby is m the number of edges, n the nodes and d is the depth of the dendrogram describing the community structure. Clauset et al. took the assumption that many real-world graphs are sparse and hierarchical therefore $m \sim n$ and $d \sim log(n)$ which leads to an almost linear run time.

In summary, the algorithm tries to assign nodes to certain communities maximizing the modularity greedily. If there are no improvements to the modularity possible, which can be done through a node reassignment, the algorithm terminates.

3.2 Leading Eigenvector

The idea of the Leading Eigenvector[4] approach is to split the graph successively into densely connected communities based on the eigenvector of the biggest eigenvalue of the modularity matrix. The modularity matrix is defined as the difference between the symmetric adjacency matrix of the graph and P. $P[i, j]$ contains the probability that node i shares an edge with node j. Obviously, P is symmetric as well. After calculating the eigenvector of the biggest eigenvalue of the modularity matrix, the graph is split based on the sign of the corresponding node in the eigenvector. If all entries in the eigenvector share the same sign, the network is assumed to have no underlying community structure. As described by Newman[4], since the proposed method runs in polynomial time and the problem

[1] https://news.ycombinator.com/
[2] https://www.reddit.com/r/datasets/comments/6v685o/complete_hacker_news_ycombinator_data_dump/
[3] https://www.sqlite.org/
[4] http://igraph.org/

GM 2017, February, 18 – 2018, Potsdam
2018. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/

of modularity optimization is NP-hard, one can be certain that the algorithm does not yield the best result in all cases.

3.3 Louvain

In comparison to the first described algorithm, this one tries to optimize the modularity similarly. The algorithm got its name from its authors Blondel et al. [1] because the main part of the work was done within the Université catholique de Louvain. Unfortunately, there is no existing proof for the runtime but it is presumably considered as $O(nlog(n))$.

Each vertex will be assigned to a community initially rather than a node as described in Subsection 3.1. Iteratively, the algorithm moves vertices between communities to optimize the modularity every round. After every round, all found communities are shrunk to a single vertex and the process continues until the modularity of the graph cannot be further optimized by moving vertices.

4 NETWORK ANALYSIS

Even though there is not a direct relation between users on HackerNews, we wanted to see if users can be grouped by their opinions. Hence we might find them commenting on the same stories.

First, we tried building the full graph on all years of posts. Unfortunately, we quickly discovered that building the graph would take a long time and a lot of RAM, more than a normal laptop could provide. With the method described in Section 2 we started using specific user groups. Figure 1 indicates that the majority of users is not commenting very often rather just having a few comments at all.

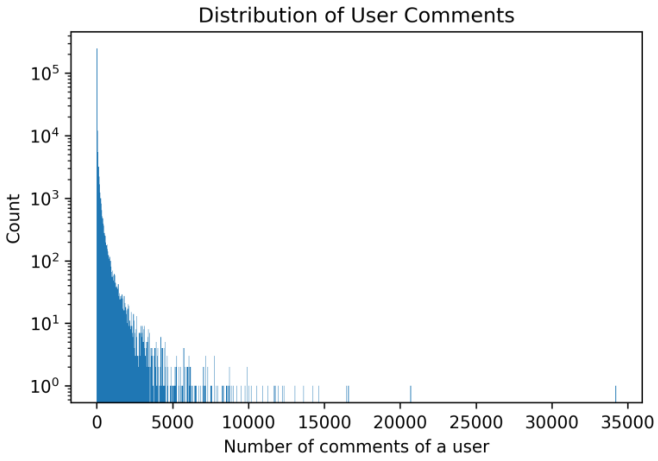

Figure 1: User comments histogramm extracted from the full dataset

In the following section, we will mainly work with two subsets of user groups and its resulting graphs. Table 1 should give a short overview what kind of graphs we are analyzing.

	Middle 10k	Top 10k
Edge Count	38914	170204
Average Degree	5.959	17.348
Diameter	11	8

Table 1: Graph summary

4.1 Top 10k Users

With this knowledge, we created our first graph by selecting the top 10000 most commenting users. We still noticed that the graph was growing too fast especially the number of edges when working on the full dataset. We filtered the underlying data by only considering comments posted in January 2017. Our assumption was to find good communities with a high modularity due to the high number of comments. This might have been a misconception which can be seen in Figure 2

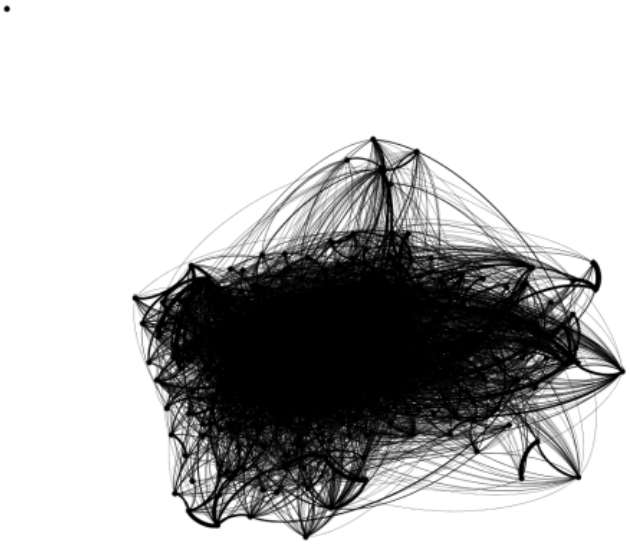

Figure 2: Graph subsampled with the top 10k users and January 2017

The resulting graph is highly connected and does not imply any communities while looking at it. That is why we assumed that the top 10000 users are not very representative for the all users on HackerNews.

4.2 Middle 10k Users

Our next idea was building the graph with a user group somewhere in the middle of the histogram in Figure 1. This might lead to a better user distribution based on their opinions since we have limited the number of overall comments. The graph was built with the first

Algorithm	Time	Modularity
Fast Greedy	1s	0.46
Louvain	1.1s	0.46
Leading Eigenvector	3.8s	0.41

Table 2: Community detection performance on full graph

10000 users ordered by the comment count per user having more than 50 comments.

In Table 2 one can see that all community algorithms find significant community structures due to $0.4 < modularity < 0.7$ which usually indicates how good the detection algorithm performs. Although the first two algorithms running almost the same time and performing equally, the Leading Eigenvector algorithm is more than three times slower and also performing worse. In general, we would have expected a better result from the Leading Eigenvector algorithm since the approach itself tries to directly optimize the community splits depending on the modularity matrix and not on moving edges or nodes. A reason might be that the computed graph is degenerated which leads to an unstable solution of the eigenvector solver which is needed for the algorithm.

To further improve the modularity of our partitions we investigated in removing very tiny communities which are just increasing the complexity. Therefore we built the biconnected components of the graph as shown in Figure 3.

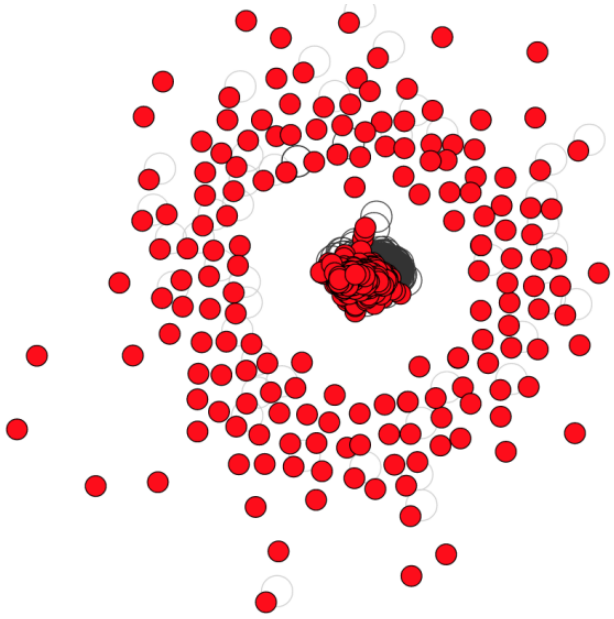

Figure 3: Biconnected components of full graph

The newly formed graph has a node for every detected component and an edge if the components are connected. Hereby we are not showing the articulation nodes which would not belong to one component only to get a better overview of the overall structure. Henceforth we will describe the inner components as *Core*.

Following the idea mentioned before we are just concentrating on the *Core* and removing the surrounding components to build a new graph. The reduced graph has roughly 700 nodes and edges less than the one looked at before.

Algorithm	Time	Modularity
Fast Greedy	408ms	0.45
Louvain	780ms	0.45
Leading Eigenvector	2.95s	0.37

Table 3: Community detection performance on core graph

In comparison to the results on the full graph, nothing significant has changed. All algorithms are faster but slightly less accurate. This means even with the removal of small components the algorithms cannot find better community structures. Nevertheless, the number of communities was drastically reduced from 204 to 38 in the *Core*.

Given a more structured view seen in Figure 4 it is noticeable that the formed communities closely relate to the found biconnected components shown in Figure 3.

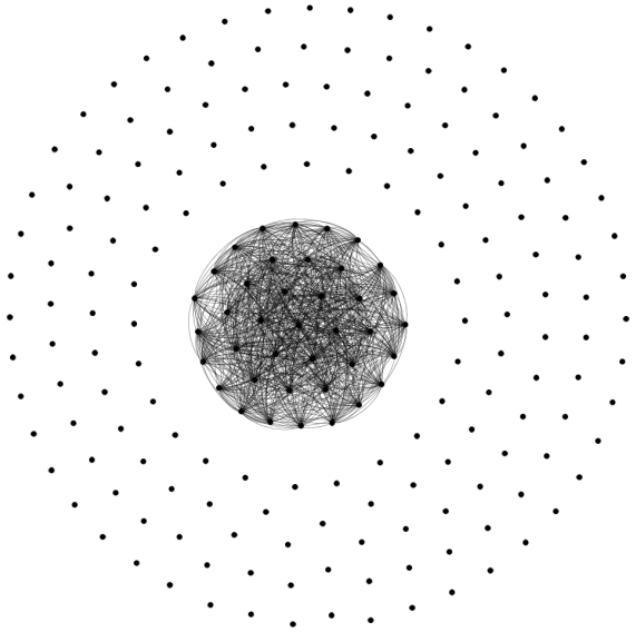

Figure 4: Extracted communities with the Louvain algorithm

One reason for the dense core could be that a usual user of HackerNews is commenting a lot of stories but without a meaningful or predictable background.

5 LIMITATIONS AND INTERESTING FINDINGS

When analyzing the underlying dataset, we noted several power law relationships. As an example, we already presented the number

of user comments in Figure 1. The number of comments and the scores of a story also follow this distribution which peaks at zero. Since this expresses a high skewness of the underlying data, the results of the community detection algorithms might be improved if one filters the considered comments more extensively. This could, for example, be done by building the graph based on comments that belong to stories with a score above a certain threshold. An other possibility would be to only consider stories that are connected with a minimum number of comments to better select popular stories.

An other limitation is that the communities are barely interpretable and there is no ground truth to compare the results with. To overcome these issues, one could apply topic modeling based on the comments of the users. This way, one can not only find common topics users talk about but by aggregating the comments of a user one can find his "favorite topic". This way, each user can be labeled with his favorite topic and the found communities can be compared with the topic labels using e.g. mutual information[3]. The idea behind this approach is that users who comment the same stories are expected to also comment on the same topics.

In our presented approach we assumed the graph to be unweighted. However, it might make sense to consider weighted edges since initially we deduplicated edges which may be used for weighting.

ACKNOWLEDGEMENT

We would like to thank Anton Tsitsulin for providing the data which we used to build the graphs.

REFERENCES

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

[2] Aaron Clauset, Mark EJ Newman, and Cristopher Moore. 2004. Finding community structure in very large networks. *Physical review E* 70, 6 (2004), 066111.

[3] Thomas M. Cover and Joy A. Thomas. 1991. *Elements of Information Theory.* Wiley-Interscience, New York, NY, USA.

[4] Mark EJ Newman. 2006. Finding community structure in networks using the eigenvectors of matrices. *Physical review E* 74, 3 (2006), 036104.