# Microloan Mania: Community Detection in Kiva

CS 224W Project Milestone, Fall 2018

Yachen Sun, Yasmeen Alfouzan, Aaron Levett

GitHub: https://github.com/levett/cs224w_kiva

## Introduction

Real-world networks are inhomogeneous and far from random. Often, groups of nodes are found such that they have a denser connection between one another than with the rest of the network. This feature of networks, known as community structure, allows us to detect communities and assume certain qualitative features about nodes that are clustered.

Founded in 2005, Kiva.org is a online microloan network allowing individuals and organizations to loan money to low-income entrepreneurs and students from around the world. As of October 2017, over one billion dollars worth of loans have been provided through Kiva's network of 1.6 million lenders and 2.6 million borrowers. A few years after Kiva's creation, Kiva opened up a "Teams" feature. Kiva teams are groups of lenders which are, as Kiva describes, 'self-organized groups built around common interests, school affiliation or location'. As of November 4, 2018, there were a total of 36,405 Kiva teams.

Kiva, however, does not automatically form or suggest teams. There may be large groups of people donating to similar causes who are not yet linked. Linking individuals making similar donations can help introduce individuals to more loan requests they may be interested in contributing to, increasing overall net philanthropy. The goal of our project is to come up with a model for community detection suitable for our Kiva dataset so that we can identify potential new teams that should form. Overall, we use the Leiden Algorithm and Trawling to perform community detection. We qualitatively evaluate bipartite cores generated from the trawling and learn that the trawling reveals groups of users with very similar lending habits, in essence discovering that trawling may be useful tool for Kiva to improve user connection and engagement. We quantitatively evaluate Leiden-detected communities against a null model and see that team structure is significantly captured relative to the null model, indicating that teams do promote similar lending behaviors.

## Review of Relevant Prior Work

A major piece of feedback from our initial project proposal was to focus more on community detection algorithms instead of lender-borrower networks. We address this by reviewing some methods below (at a high level with a bit of mathematical background for this milestone) for exploring **both the bipartite network, and a folded unipartite representation**.

### *1. Leiden Algorithm (Traag et. al)*

In initial exploration of our data, we attempted using builtin SNAP community detection methods, such as CNM on the folded network. Unfortunately, these used up way too much memory and were taking far too long to run. We needed to find an algorithm scalable to our folded network with over 25 million edges. We came across the Leiden algorithm, which seems well suited for this. The leiden algorithm is similar to the Louvain algorithm discussed in class, but it adds additional refining phases between the movement of nodes and aggregation to ensure the connectivity within communities. In addition to merging communities like Louvain, Leiden is also able to split communities, and find subset optimal clusters; in practice, it has been shown to work better than Louvain on many networks (Traag et al.)

Some initial findings using the Leiden algorithm are discussed later in the milestone report. An important note is that the Leiden algorithm is obviously limited for our folded network. Given how we fold the network, we form cliques consisting of lenders for a given loan. These cliques will likely be the detected communities. This leads us to need a hidden community detection method to find communities outside of the unipartite cliques. one of which is discussed below.

*2. HICODE (He et al.)*

HICODE (Hidden Community Detection) is an algorithm to detect communities that are weaker than those in the dominant community structure. HICODE starts by applying an existing community detection algorithm to a network. It then repeatedly weakened the structure of the detected communities in the network, allowing increased visibility of the hidden networks.

To weaken the predominant community structure, three approaches can be taken:

1. Removing intra-community edges
2. Approximating community layers as a stochastic blockmodel, and other edges as noise. After making the approximation, this method removes certain edges within each community block until the block's edge probability is equal to the background edge probability of the block.

   Mathematical Description:

   For each community $C_k$ to be reduced:
   $n_k$ = number of nodes in $C_{k,}$ $e_{kk}$ = number of edges in $C_k$ ,$d_k$ = sum of degrees of nodes in $C_k$, $p_k$ = Observed edge probability = $e_{kk}$ / $(0.5n_k(n_k\text{-}1))$, $q_k = (d_k - 2e_{kk})/[n_k(n\text{-}n_k)]$ = the average outgoing edge density of the community block, and $q_k' = q_k/p_k$

   For each edge in a detected community $C_k$ , we remove the edge with probability $(1\text{-}\ q_k')$

3. Reducing intra-community edge weights

The Louvain algorithm was one of the two most successful base algorithms the authors tested. Given the similarity of the Louvain to the Leiden algorithm it seems like the Leiden algorithm could be well suited for use in HICODE. However, initial interesting results from our Trawling (method described below) experimentation (discussed later in the paper) led us to decide that the use of HICODE would likely not benefit our analysis.

*3. Trawling (Kumar 1999)*

Trawling (Kumar 1999) allows for overlapping community detection in bipartite graphs, making it relevant to our lender-loan dataset. Trawling enumerates (*i,j*) bipartite subgraphs (bipartite cliques) with *i* nodes in one partition and *j* nodes in the other. For the Kiva dataset, we believe that detecting complete bipartite subgraphs would be a good way to detect or recommend teams, given how lenders in complete bipartite subgraphs are contributing to similar loans. We can experiment with different *i* and *j* values to explore how the number of cores detected and perform qualitative analysis on generated cores. One limitation is that Kiva teams do not have every single member contributing to a given loan. We cannot detect entire existing teams through trawling, but nonetheless could detect groups of similar lenders who may be good to connect based off of loan interests.

## Data Collection and Processing

We have completed data retrieval from the Kiva network. First, we downloaded Kiva's full data snapshot, which contains information about all loan requests (whether fully funded or not), and all lenders who have contributed to the loan. The snapshot data is from April 2006 but only up to January 2018. We decided to limiting the loans to those that were posted in 2017 in order to limit network size but still be able to detect communities. We followed that with some data processing to combine lender and loan data into a unified format for downstream use. After filtering and reshaping the data, we mapped usernames (string) to generated user ids (int), in order to be able to construct the network in SNAP.PY that consists of 205k loans, 504k lenders, and 3.19 million edges.

In addition to the snapshots containing loan/lender information, Kiva offers APIs to provide team information. However, there is no API to get a list of all teams and their members. Instead, we had to use a series of APIs. First, we used the general team search API to retrieve all team IDs (team ID numbers are not consecutive). Then, after saving the list of team IDs, we used an API to retrieve lists of team members by team ID. API calls were rate limited to 60 per minute. We wrote a Python script to make API calls 1x/second for the team data. It took about 10 hours to retrieve the list of team members for all 36,128 teams. For our early stage exploratory analysis and community detection methods, we used a 3-month snippet in order to reduce computation time, which we then extended to the full 12-month data.

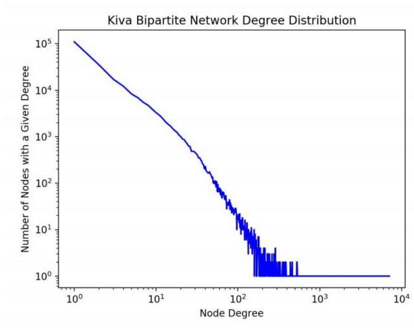
## Initial Graph Creation

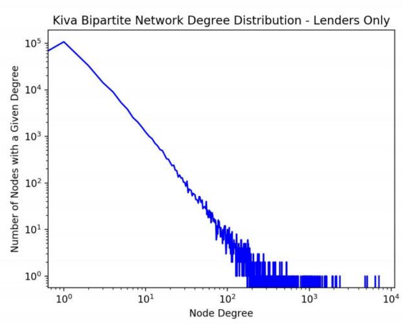
We first created a bipartite graph in SNAP.py out of the three month time period defined above, with one partition containing nodes that represent loan requests, and the other partition containing nodes that represent lenders. Edges between a lender and loan request node indicate that the lender contributed to the loan request. In this bipartite graph, there are 49,043 loan nodes, 189,718 lender nodes and 747,807 edges. We folded the bipartite graph in Snap.py such that nodes represented lenders and edges between lenders indicated that the lenders had contributed to at least one of the same loans. For the three month time period, there were 189,718 nodes and 26,534,564 edges in the folded network.

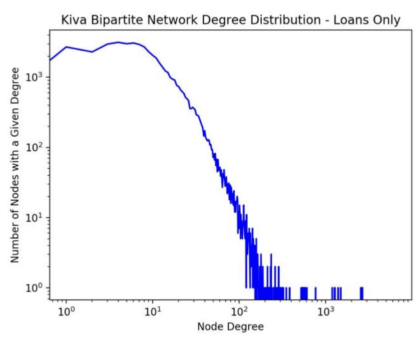
## Summary Statistics

In our project proposal, the reviewer suggested we perform some exploratory analysis on the loan-lender network, as there is not a thorough discussion of this in the literature. For the bipartite graph, the average node degree was 6.26. For lender nodes, the average degree was 3.81, indicating that on average a lender contributes to 3.81 loans. For the loan nodes, the average degree was 15.25, indicating on average that 15.25 lenders contribute to a loan.

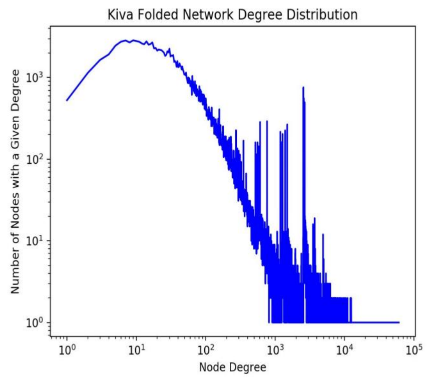
***Degree Distribution: Bipartite Network***





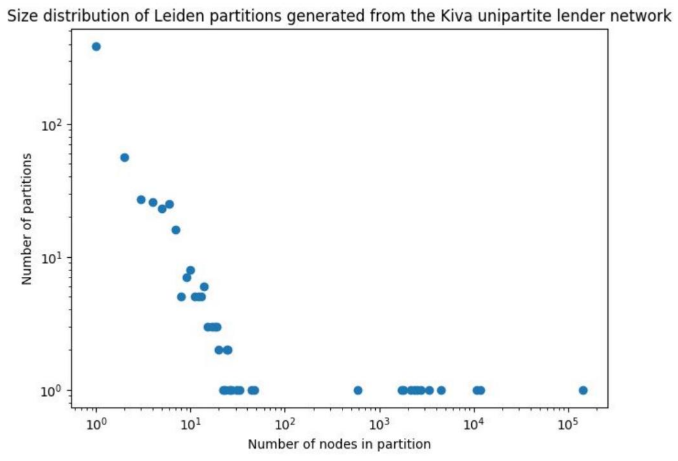


For the folded lender-loan network, the average degree distribution was 280.30, meaning on average, a lender has co-lended with 280.30 other people on at least one loan. The degree distribution of the folded network, shown, appears to follow a power law distribution except at very low node degrees (≤ 10). This is because - as seen through our bipartite graph analysis - multiple people usually contribute to a single loan, so we would not expect very many nodes in the folded network to have an extremely low degree. Next, we used SNAP to calculate the average clustering coefficient (0.793) out of a random sample of 25,000 nodes in the folded network.



## Community Detection via Leiden Algorithm

### *Leiden Analysis of the Unipartite Lender Network*



We discovered the Leiden Algorithm to be a fast and scalable way to perform non-overlapping community detection for our dataset. Using the "leidenalg" package, we obtained preliminary community partitioning results of 644 clusters from the unipartite lender network. As shown below on the left, the distribution of partition sizes are very uneven, with a few huge clusters and a large number of very small clusters.

Specifically, the largest partition contains 75.5% of the total nodes.

We suspected that this distribution of partitions may be a result of a handful of "super loans" with a lot of lenders generating huge cliques in the folded unipartite network. We then evaluated if users who donated to the largest loans are more likely to appear in the same large cluster together. As shown below, for the loans with the highest degrees in the bipartite network, their lenders are very likely to all be partitioned into the same cluster/partition. These clusters are also very likely to be one of the medium-large range clusters. This means that these super loans has a significant role in generating medium-large sized partitions in the Leiden result. However, this does not quite explain the existence of the one giant cluster that comprises of 75.5% of all the nodes; none of the top 20 loans has significant loan-related clique existence inside that cluster, and out of the top 50 loans, only 15 loans have more than 20% of their clique in that cluster, which means that its size is likely not related to cliques generated by super loans, but a result of noisy data and resolution issues associated with the Leiden algorithm.

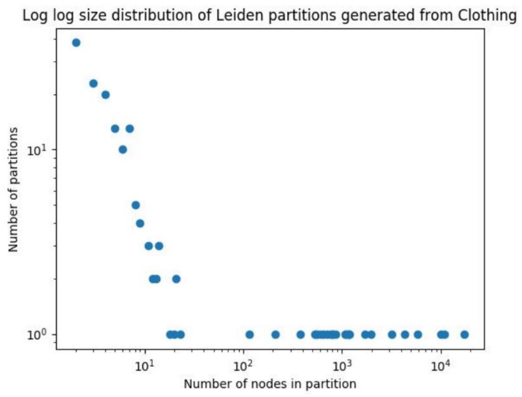
| Largest Loan | Percentage of Lenders in Cluster | Largest Cluster | Cluster Node Percentage |
|---:|---:|---:|---:|
| 1 | 0.804 | 8 | 0.0125 |
| 2 | 0.893 | 6 | 0.0144 |
| 3 | 0.939 | 7 | 0.0134 |
| 4 | 0.6 | 9 | 0.0114 |
| 5 | N/A | N/A | N/A |
| 6 | 0.582 | 9 | 0.0114 |
| 7 | 0.598 | 10 | 0.00949 |
| 8 | 0.727 | 4 | 0.024 |
| 9 | 0.717 | 11 | 0.0091 |
| 10 | 0.865 | 2 | 0.0618 |

***Splitting the Data by Loan Category and Noise Reduction by Trimming***

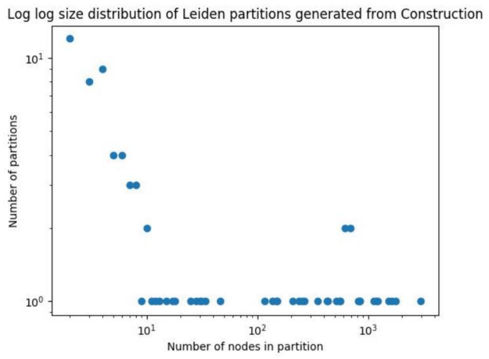
Because the Leiden analysis on the unipartite network did not have enough resolution, we decided to change our approach to graph creation by including two more restrictions (as well as expanding it to the full 12-month data): First, we set a threshold: lenders only share an edge in the folded unipartite network if they contribute to at least three of the same loans, and that is to address the problem of the presence of the giant cluster. Second, since there are fifteen categories of loans, we decided to process the data to output several graphs where lenders contribute to the same category of loan. The categories and their relative sizes (by loan volume) are: Agriculture (24.4%), Food (22.7%), Retail (20.2%), Services (7.3%), Clothing (5.7%), Housing (4.2%), Personal Use (3.6%), Education (3.2%), Transportation (2.8%), Arts (2.0%), Construction (1.3%), Health (1.2%), Manufacturing (1.1%), Entertainment (0.1%), and Wholesale (0.1%).

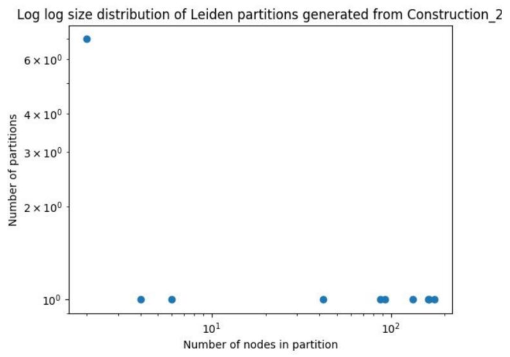
## *Leiden Analysis of Category Networks and Trimmed Networks*

We ran Leiden analysis on all of the category graphs, and their trimmed subgraphs with shared loans ≥ 3. We made the following observations on the communities generated:



1. For the untrimmed category networks, the sizes of small communities (under 50 lenders) tend to follow a power law distribution, as shown on the right.
2. In the untrimmed networks, the size distributions of large communities differ from category to category. Some categories have one giant community that includes most of the lenders in the network, such as Agriculture and Health, whose largest communities consist of 83.2% and 70.7% of all the lenders in the networks, respectively. Other categories show much more even community distributions, such as Food (top community has 29.7% of all lenders), Arts (27.9%) and Housing (9.2%).
3. As expected, compared to untrimmed networks, the trimmed networks tend to display a lot fewer and smaller communities detected, as shown below. Noticeably, even though the overall community sizes shrunk, the trimmed networks actually don't have a lot of super small (under 10 lenders) communities, and the size distributions no longer has any power law shape to them. By excluding low-quality connections, we obtained communities that are more medium sized and of higher quality.





*The graph on the left is the log-log community size distribution of the untrimmed Construction category network. The one on the left is that of the Construction network trimmed with shared loans ≥ 3.*

## *Comparison between Large Partitions and the WCC*

As mentioned in the section above, some of our category networks has large communities that contained most of the lenders in the network. Since our unipartite lender networks are likely not connected networks, we were concerned that the Leiden communities may coincide with the connected components in these networks. To test this idea, we computed the Jaccard index between the largest Leiden community and the largest weakly connected component (WCC) in all of the untrimmed category networks.

The Jaccard indices' average came out to be 0.312 with a standard deviation of 0.199, which suggests that overall, **while there is some reasonable overlap between the largest WCC and the largest Leiden community, in most communities they're still pretty different**.

However, two category networks, **Health and Manufacturing**, stood out as their top Leiden partitions are very similar to their largest WCCs (Jaccard Indices 0.707 and 0.591). Both of these are very small categories (1.2% and 1.1% of the entire network) with most of their lenders belonging to one giant partition. This suggests that **lenders in these two categories tend to share more common loan interests and form larger communities**.

***Leiden - Modularity Evaluation***

To evaluate the quality of the Leiden results, we calculated the modularity of the Leiden partitions for the category networks.

| *File Name* | *Modularity* | *Modularity for Trimmed Network* |
|---|---|---|
| Arts | 0.5847108624 | 0.3739684848 |
| Clothing | 0.4888333958 | 0.3197973851 |
| Construction | 0.6222456764 | 0.4114191771 |
| Education | 0.5140466356 | 0.378236955 |
| Entertainment | 0.7891241025 | 0.4457894737 |
| Health | 0.5082542732 | 0.5190710255 |
| Housing | 0.5123627199 | 0.3910463045 |
| Manufacturing | 0.5715018017 | 0.4093520677 |
| Personal Use | 0.5237333012 | 0.5288473014 |
| Retail | 0.4839447485 | 0.3591253641 |
| Services | 0.5476153036 | 0.4452818697 |
| Transportation | 0.6223509787 | 0.432623442 |
| Wholesale | 0.07446472501 | 0.5417187283 |
| **Average** | **0.5263991173** | **0.4274059676** |
| **Standard Deviation** | **0.158494631** | **0.06822970247** |

As shown above, the modularity for all of the network partitions, except untrimmed Wholesale and untrimmed Entertainment, are in the 0.3 to 0.7 range. Wholesale and Entertainment partitions also achieved that range after trimming. This shows that **our Leiden community detection has discovered significant community structures in the category unipartite networks**.

***Leiden - Null Model Comparative Analysis***

We wanted to determine how well the communities we detected via Leiden analysis reflected team structure compared to a null model. To perform this analysis, we did the following:

1. Take communities detected via Leiden for a subcategory, and calculate the Jaccard Coefficient between the communities and the teams. Record maximum Jaccard score.
2. Randomize communities detected via Leiden analysis while maintaining the overall number of lenders in each community. Do this 100 times.
3. For each of the 100 randomizations, calculate the maximum Jaccard Similarity score as done for the Leiden-detected communities.
4. Let the p-value equal the fraction of the iterations where the maximum Jaccard score from the randomized communities is greater than the maximized Jaccard score from the Leiden-detected communities.

Originally, when running the analysis, p-values for different categories hovered around 0.50. Upon closer inspection, we noticed that in both the Leiden-detected communities and randomly detected communities, there would be some extremely high Jaccard scores stemming from, for example, teams and communities with only 2 donors in which there was one shared neighbor. We felt that this resulted in outlier high-Jaccard scores, and that the metric was not necessarily ideal for comparing detected communities to randomize ones. To address this, we re-ran our analysis only considering Jaccard Similarity scores for teams/communities where there were *at least 2 shared neighbors*. We felt that threshold applied to both the detected and randomized communities was the least invasive way to improve the analysis quality. Closely inspecting the results, we noticed that the small team/community issue was mitigated, and that many more shared neighbors existed in the communities/team pairings that results in the highest Jaccard similarity scores.

With this threshold, for all loan subcategories, the detected communities actually *always* had a higher max jaccard score (p-value = 0.00). This indicates that **our Leiden community aggregation did a significantly better job of detecting communities similar to true Kiva teams than a random community detection method would.**

**Emerging Community Detection via Trawling Bipartite Core Generation**

In their groundbreaking paper, 'Trawling the Web for Emerging Cyber-Communities', *Kumar et al.* search for *internet* communities, noting that the 'chaotic nature of content-creation on the Web has resulted in many more implicitly defined communities. Such implicitly defined communities often focus on a **level of detail that is typically far too fine to attract the current interest (and resources) of large portals to develop long lists of resource pages for them**.' [emphasis added] (Kumar et al, 1999). The Kiva network presents a similar challenge. Many existing teams on Kiva are quite broad (i.e. employees of a specific company, or members of a certain religion), and are sparse (few members actually make loans).

We implemented a modified version of the trawling algorithm described in the paper (our method described below). Mainly, we stop our trawling after the inclusion-exclusion core generation step, without proceeding to the *apriori* algorithm, which finds weaker cores than the complete bipartite subcores generated during inclusion-exclusion core generation. This decision was made in part due to both computational considerations and the nature of Kiva. Furthermore, we were satisfied with the cores generated from the inclusion-exclusion step.

***Trawling Methodology + Results***

Our trawling implementation parallels that of Kumar et al. with some modifications. Below is pseudocode detailing our algorithm.

```
For i in range(3,11),                                         (1)
        For j in range (3,9):                                 (2)
                iterative prune                               (3)
                Inclusion exclusion pruning and core generation (4)
```

During iterative pruning, for a given (i, j) pair, we iteratively remove lender nodes with a degree < j (lenders who donate to fewer than j loans), and loans with degree < i (loans contributed to by fewer than i lenders) until no nodes are removed. During inclusion-exclusion pruning, we find all lenders with degree j. For each loan the lender contributes to, we get the corresponding lenders, and the intersection if size of the lender sets' intersection ≥ i, we record it as an (i,j) core (note that this means that for all k > i, a (k, j) core are recorded as an (i, j) core; simple arithmetic can be used to calculate unique core counts for those interested). As j increases, nodes that were pruned would be pruned again, so we use the existing network, which resulted in drastic runtime improvements.

Below, we show the table of core counts one category of loans (housing). Core counts for other categories can be found in the appendix.

| **Total Nodes: 75750** | **Edges: 135769** | |
|---|---|---|
| ***I (lenders)*** | ***j (loans)*** | ***Count*** |
| 3 | 3 | 106 |
| 3 | 4 | 17 |
| 3 | 5 | 6 |
| 3 | 7 | 1 |
| 4 | 3 | 53 |
| 4 | 4 | 6 |
| 4 | 5 | 2 |
| 4 | 7 | 1 |
| 5 | 3 | 25 |
| 5 | 4 | 3 |
| 6 | 3 | 10 |
| 6 | 4 | 1 |
| 7 | 3 | 5 |
| 7 | 4 | 1 |
| 8 | 3 | 3 |
| 9 | 3 | 2 |
| 10 | 3 | 1 |

Given the size of ground truth Kiva teams and the size of cores and a discussion with a TA, we decided that quantitative analysis of cores did not seem conducive to practical insights. So, our next step was to perform manual, qualitative inspection and evaluation of randomly chosen cores from each category, to see if any apparent similarities between loans were detected. In fact, this is also the approach taken by Kumar et al.

Some of the loan partitions in the sampled cores were seemingly random (i.e. three housing loans that did not seem to have any unifying attributes other than being housing related). Many, however, seemed to reflect non-random groupings of loans, especially when there were high j-values. One notable core was the (4,7) core from the housing category. All seven loans were for different women in Vietnam who were hoping to install new toilets to improve the sanitation of their homes (loan IDs 1378531, 1378534, 1378536, 1378540, 1378554, 1378576, 1378596 can all be searched and examined directly on Kiva). Another notable core was a (6,7) core from the services category network. All 7 loans in the partition pertained to different small, all female-run beauty salons in Paraguay (loan IDs 1224121, 1224476, 1225448, 1228571, 1228612, 1228746, 1228751). Qualitative inspection detected similarities between intraloan cores such as the two described above much more frequently for high j-value cores ($j \geq 6$). The cores files for each loan category are publicly available on our GitHub (linked at the end of the report).

Given these results, we believe that NLP methods (beyond the scope of this class) could be used to analyze cores and do a fully comprehensive analysis of intracore loan similarity. This could help Kiva automatically recommend loans to lenders or displaying similar loans. Kiva could take cores whose loans have semantic similarity, further expand these cores to include all donors who have donated to any of the loans in the core, and auto-create teams of lenders with like-minded attitudes towards microloans. These teams would be distinct from existing teams in that they are mainly mission-based, as opposed to identity based.

## Conclusion

We have explored methods for community detection, including the Leiden Algorithm and Trawling. Trawling seems to effectively capture groups of users who provide loans to very similar loan requests, with notable examples described earlier in the report. Overall, it appears that trawling could be quite beneficial; combined with NLP models, it has the potential for Kiva to connect users with niche lending behaviors. The Leiden Algorithm produces communities that more closely capture Kiva team structure than a random model; Leiden communities can vary greatly in sizes and can reveal structural properties of different category networks. Overall, we believe that these results indicate that Kiva can be doing better to connect similar users based off of loan history: trawling may be a good place for Kiva to start looking for improvements.

**Citations**

Traag, V., Waltman, L., & van Eck, N. J. (2018). From Louvain to Leiden: guaranteeing well-connected communities. *arXiv preprint arXiv:1810.08473*.

He, K., Li, Y., Soundarajan, S., & Hopcroft, J. E. (2018). Hidden community detection in social networks. *Information Sciences*, *425*, 92-106.

Kumar, R., Raghavan, P., Rajagopalan, S., & Tomkins, A. (1999). Trawling the Web for emerging cyber-communities. *Computer networks*, *31*(11-16), 1481-1493.

**Appendix A:** Trawling Core Counts By Category (Representative Subset of Categories, full sets of cores available on GitHub)

**Health**

| **Total Nodes: 50494** | **Edges: 80987** | |
|---|---|---|
| ***I (lenders)*** | ***j (loans)*** | ***Count*** |
| 3 | 3 | 99 |
| 3 | 4 | 19 |
| 3 | 6 | 5 |
| 4 | 3 | 42 |
| 4 | 4 | 8 |
| 5 | 3 | 23 |
| 5 | 4 | 6 |
| 6 | 3 | 11 |
| 6 | 4 | 6 |
| 7 | 3 | 7 |
| 7 | 4 | 4 |
| 8 | 3 | 5 |
| 8 | 4 | 4 |
| 9 | 3 | 4 |
| 9 | 4 | 1 |
| 10 | 3 | 5 |
| 10 | 4 | 1 |

**Education:**

| **Total Nodes: 107713** | **Edges: 215441** | |
|---|---|---|
| ***i(lenders)*** | ***j(loans)*** | ***Count*** |
| 3 | 3 | 177 |
| 3 | 4 | 37 |
| 3 | 5 | 9 |
| 3 | 6 | 2 |
| 3 | 7 | 1 |
| 3 | 8 | 1 |
| 4 | 3 | 78 |
| 4 | 4 | 10 |
| 4 | 5 | 3 |
| 4 | 6 | 1 |
| 4 | 8 | 1 |
| 5 | 3 | 40 |
| 5 | 4 | 3 |
| 5 | 8 | 1 |
| 6 | 3 | 23 |
| 6 | 4 | 2 |
| 7 | 3 | 14 |
| 7 | 4 | 1 |
| 8 | 3 | 9 |
| 8 | 4 | 1 |
| 9 | 3 | 8 |
| 9 | 4 | 2 |
| 10 | 3 | 6 |
| 10 | 4 | 2 |

## Personal Use

| Total Nodes: 74657 | Edges:130643 | |
|---|---|---|
| *i(lenders)* | *j(loans)* | *Count* |
| 3 | 3 | 104 |
| 3 | 4 | 40 |
| 3 | 5 | 6 |
| 3 | 6 | 6 |
| 3 | 7 | 2 |
| 4 | 3 | 42 |
| 4 | 4 | 15 |
| 4 | 5 | 3 |
| 4 | 6 | 3 |
| 5 | 3 | 22 |
| 5 | 4 | 6 |
| 5 | 5 | 2 |
| 5 | 6 | 2 |
| 6 | 3 | 16 |
| 6 | 4 | 7 |
| 6 | 5 | 2 |
| 6 | 6 | 1 |
| 7 | 3 | 14 |
| 7 | 4 | 6 |
| 7 | 5 | 2 |
| 7 | 6 | 1 |
| 8 | 3 | 13 |
| 8 | 4 | 5 |
| 8 | 5 | 1 |
| 9 | 3 | 11 |
| 9 | 4 | 2 |
| 9 | 5 | 1 |
| 10 | 3 | 6 |
| 10 | 5 | 1 |

## Agriculture

| Total Nodes: 302488 | Edges: 921837 | |
|---|---|---|
| *i(lenders)* | *j(loans)* | *Count* |
| 3 | 3 | 1031 |
| 3 | 4 | 202 |
| 3 | 5 | 55 |
| 3 | 6 | 19 |
| 3 | 7 | 12 |
| 3 | 8 | 10 |
| 4 | 3 | 404 |
| 4 | 4 | 78 |
| 4 | 5 | 15 |
| 4 | 6 | 8 |
| 4 | 7 | 3 |
| 4 | 8 | 1 |
| 5 | 3 | 215 |
| 5 | 4 | 45 |
| 5 | 5 | 10 |
| 5 | 6 | 5 |
| 5 | 8 | 1 |
| 6 | 3 | 133 |
| 6 | 4 | 35 |
| 6 | 5 | 5 |
| 6 | 6 | 4 |
| 7 | 3 | 89 |
| 7 | 4 | 25 |
| 7 | 5 | 3 |
| 7 | 6 | 4 |
| 8 | 3 | 65 |
| 8 | 4 | 17 |
| 8 | 5 | 2 |
| 8 | 6 | 3 |
| 9 | 3 | 50 |
| 9 | 4 | 13 |
| 9 | 5 | 3 |
| 9 | 6 | 3 |
| 10 | 3 | 45 |
| 10 | 4 | 12 |
| 10 | 5 | 2 |
| 10 | 6 | 3 |