Community Detection in Bipartite Networks: Application on Movie-User Data from Letterboxd

Zeynep Sema Aydin: Evaluation of Algorithm

Hannah Portmann: Implementation of Algorithm, Network Visualization

Diogo Rocha: Data Preprocessing, Data Enrichment, Network Exploration

Introduction

Community detection is a particularly intriguing application of network science, which aims to identify groups of closely related nodes within a network. This report focuses on community detection in bipartite graphs using data from Letterboxd, a popular social cataloging service where users rate and review movies. Letterboxd provides a rich dataset for exploring community structures because it involves two distinct types of nodes—users and movies—forming a bipartite network. By examining how users rate various movies, we can uncover hidden patterns and clusters that may indicate shared preferences or common characteristics among groups of users or movies. We aimed to implement a community detection algorithm tailored for bipartite networks and apply it to the Letterboxd dataset to reveal insightful patterns.

In the following sections, we detail the data used for our analysis, the preprocessing steps taken to ensure data quality, the enrichment processes applied to augment the dataset, and the implementation of the Bilouvain algorithm for community detection. We then present our findings, evaluate the performance of our implementation, and discuss potential improvements for future work.

Data

We have chosen a dataset called “Letterboxd ratings (1.4M)” for our analyses [1]. It contains data from a social cataloging service called Letterboxd. On Letterboxd, users can log and rate movies they have watched. Our dataset includes 1.4 M ratings from 557 users on 108’275 movies. The usernames have been anonymized and are represented by numbers. Each row in the dataset contains the user number, movie title, and rating from 0 to 5.

It could be interesting to do community detection on this kind of dataset, as we expect similar movies, e.g., movies of the same genre, and users that like similar movies to cluster together.

Data Quality & Preprocessing

The dataset was obtained in a CSV file and analyzed in Python with pandas. The original file contained 22 rows with missing titles. Since there is no movie ID and users are anonymized, their titles were impossible to infer, so these rows were dropped from the data frame. By doing so, we lost none of the original 557 users or 108’275 movies.

To ensure consistency across the dataset, all columns were converted to their coherent data types (user and rating as numeric and title as string).

An incoherence was found while inspecting the rating column. There were ratings of 0, while the lowest rating that can be given in letterboxd is a half star (0.5). These ratings of 0 likely mean that movies were logged as watched by the users but not rated. Since we are not planning on making movie recommendations and do not use the rating in our algorithm, we decided to keep those ratings without inferring the values.

Algorithms based on networks can be quite computationally expensive and have long runtimes depending on the size of the graph. In our case, we have over a million ratings (which, by our standards, is a lot). Thus, the network would benefit from being sampled to different sizes to improve the runtime of our community detection algorithm.

Sampling is a complex field in graph theory. Decreasing network size while keeping its structure similar to the original is hard. However, while studying our dataset, we found that many movies inside of it are only rated by a few of the users. Therefore, we decided to sample the dataset by iteratively keeping only movies with ratings of more than x. By doing so, we are not randomly removing movies; we are just discarding movies that may have less effect on communities since they affect fewer users. This allows density to be kept in the graph, most disconnected subgraphs to be removed, and overall ratings to be decreased while keeping information about user preferences.

To check the impact of sampling on the ratings, we plotted the evolution over sampling of 1) Number of movies. 2) Number of users. 3) Number of ratings. 4) Percentage of ratings (Fig. 1). We observed:

- Number of movies decreases significantly during the first 50 sets.
- Users stay constant until 330 sets and then decrease by one (which is promising as we keep most information from users).
- Number of ratings decreases steadily and ends up at 69261 in set 400 (only movies with more than 400 ratings).

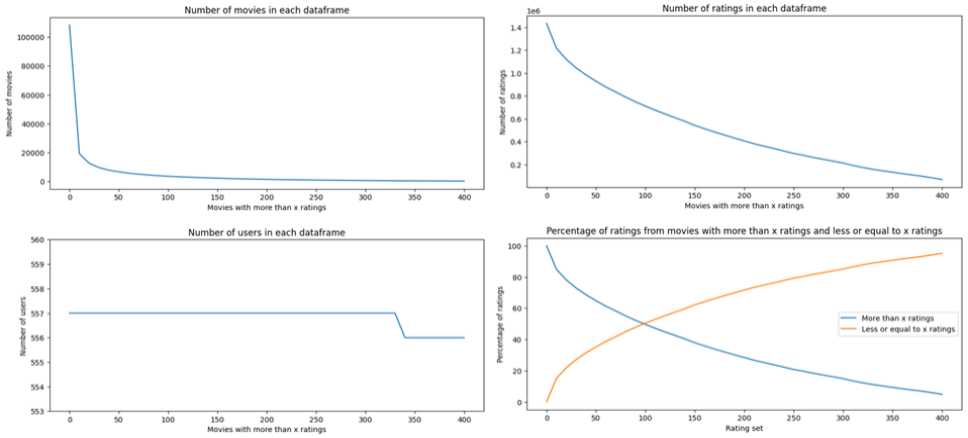

Figure 1. *Evolution of movies, ratings, users and rating percentage across samples.*

After observing the properties of different sample sets, we found that even though this sampling method allows for the keeping of structure, the samples are still quite big. So, in addition to this sampling method, we also used:

- Sampling based on random sets of movies (e.g., random 500, 1000, … movies)
- Random sampling of rows (the simplest but also the most random and keeps no structure)

Data Enrichment

Augmenting the dataset is an important step of this project, as we started with barebones data. We only had anonymous users, movie titles, and ratings. To interpret the communities we detect in the data, we chose to get the following information:

- Release date
- Duration of the movie in minutes
- Mean rating
- Genres
- Production country

These were obtained mainly from the following datasets:

- Letterboxd from Simon Garanin, specifically the movies.csv, genres.csv, and countries.csv. [2]
- IMDB Movies Dataset from Ashish Jangra [3]

After augmentation of the dataset, we ended up with multiple missing values across the different augmented columns. It is explained by: 1) Movie titles with different names across the databases. This would be easier to solve if our original dataset had movie IDs in addition to the movie titles. However, even movie IDs can vary across datasets, which would not solve the problem entirely. 2) Our dataset contains movies that are not well known (concerts, experimental movies). These are not typically present in most databases containing movies from letterboxd or IMDB as they cannot scrape the entirety of the websites and focus on more popular movies.

From these suppositions, we would expect that the number of missing values should decrease very quickly at first (where we remove the movies that are rated very few times) and then steadily for each sample (since we are removing rows and so missing values are removed either way). We plotted the evolution of missing augmented values across sampling to confirm this.

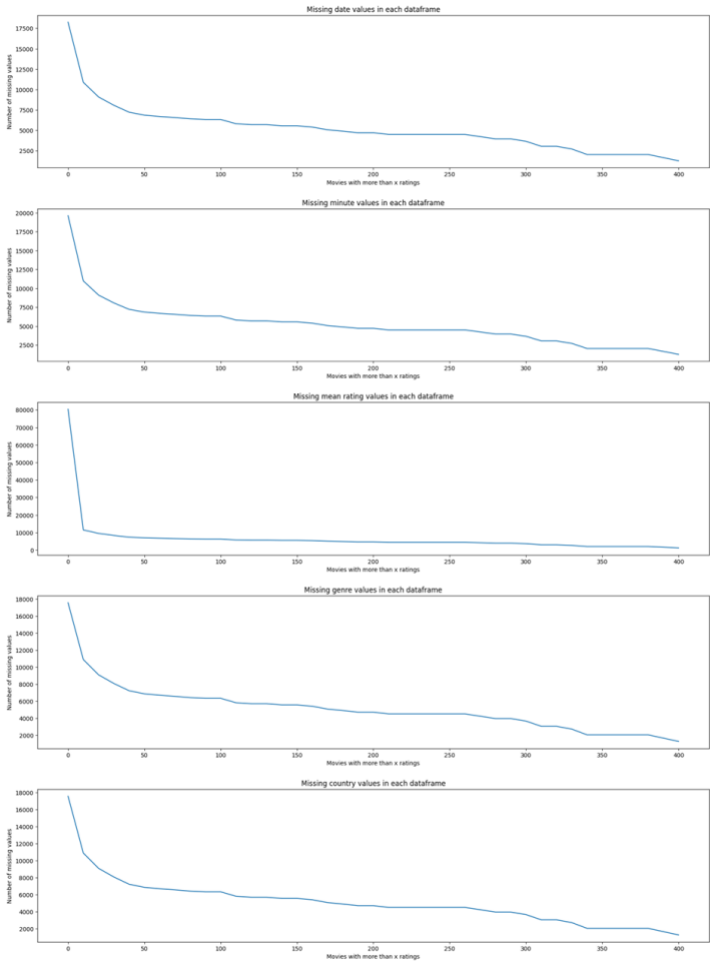

Figure 2. *Evolution of missing date, minute (duration), mean rating, genre, country augmented values across the sampling of the dataset.*

In Fig. 2, as expected, the number of missing values decreases quickly at first and then decreases steadily over sampling. This confirms the suspicions of missing values being tied to less known movies since the very first steps don't actually remove that many ratings (they remove many movies but few ratings because those movies were not rated many times) and since the augmented values are counted across ratings and not across movies, the high decline firstly observed indicates that most of the removed movies with few ratings were missing augmentation.

Network Exploration

Understanding the network we work with is a crucial step in graph analytics. It can be done through centrality measures (ranking nodes according to their importance and influence "centrality" in a network) and similarity measures (computing how similar pairs of nodes are).

During exploration, we focused on centrality measures as they are crucial to understanding the importance and influence of nodes within a network. Unlike similarity measures, which primarily highlight how similar pairs of nodes are based on different metrics, centrality provides broader insight into the contribution of nodes to the network's connectivity and flow of information. On our network of movie ratings, centrality can reveal influential users and movies with high engagement.

Degree Centrality (DC)

DC measures importance depending on the direct connections of each node.

Users have a higher degree centrality (about 200x) on average compared to the movie nodes. This disparity was expected, given the small number of users compared to movies rated.

- Top users have 10x higher DC than average users. This suggests a skewed distribution with few very active users and most users being less active.

- Users having a high degree centrality are more engaged and potentially more informative about the network's preferences. Identifying these users can help understand user behavior and target highly active users for further engagement and recommendations.

The average DC for movies is about 40x lower than for the top movies. This suggests that few movies are rated by many users, while most movies receive fewer ratings. This was promptly confirmed during sampling, where movies were removed based on their number of ratings.

- Highly rated movies are key indicators of popularity. These can be leveraged and studied to understand broader trends and preferences in the user base.
- Popular movies with high DC can be recommended to new users or those with less activity. This allows for increased engagement without necessarily identifying whether the user would like the movie or not.

In Fig. 3, we can observe that some genres and countries are clearly more represented in the top DC movies. This type of information can be beneficial for movie producers and/or investors when deciding which kind of movie would produce more engagement with users.

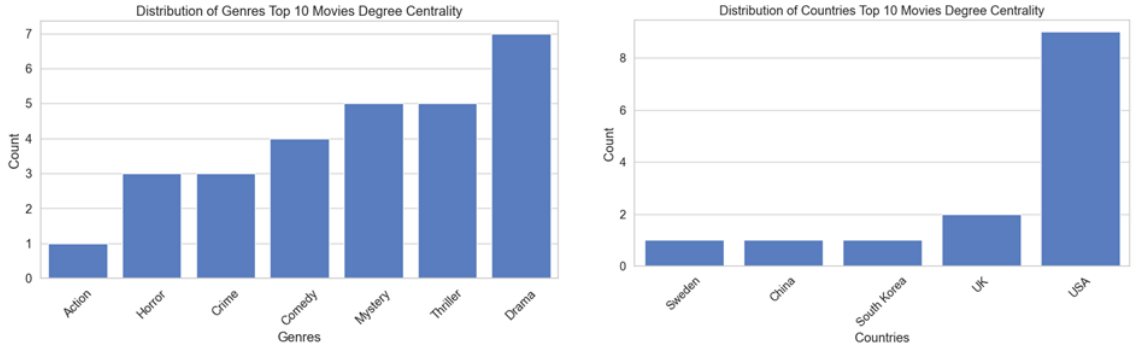

Figure 3. *Distribution of the genres and countries of the movies with the top 10 highest degree centralities.*

Betweenness Centrality (BC)

BC measures to what extent a node lies on the shortest paths between other nodes. It indicates its role as a bridge in the network.

Users have a higher average betweenness centrality than movies, highlighting their critical role in connecting different nodes in the network.

- The top BC users are essential for the network's cohesion. They act as bridges that connect isolated clusters of users and movies. Their BC is approximately 40x higher than that of the average user.
- Influential users are pivotal for spreading information and recommendations across the network. Targeting them can enhance the diffusion of new movies.
- Although lower on average, movies with high betweenness centrality are crucial in connecting diverse user groups. These are likely rated by a wide variety of users with different tastes, which indicates their broad appeal.
- The average BC of top movies is about 90x higher than that of the average movie. Their high centrality suggests that they could help link different user groups. We can again observe the network's skewness, where few movies act as bridges to diverse user groups.
- Promoting movies with high betweenness can increase overall user engagement due to their large appeal. Furthermore, studying the properties of such movies could give insights into how to get appeal from a broad range of users with diverse backgrounds and tastes.

In Fig. 4, we can observe the distribution of genres and countries across the top BC movies. Drama and mystery are less represented, horror and comedy are more represented, and there are a greater number of genres. This can indicate the effect of genres on appeal and popularity.

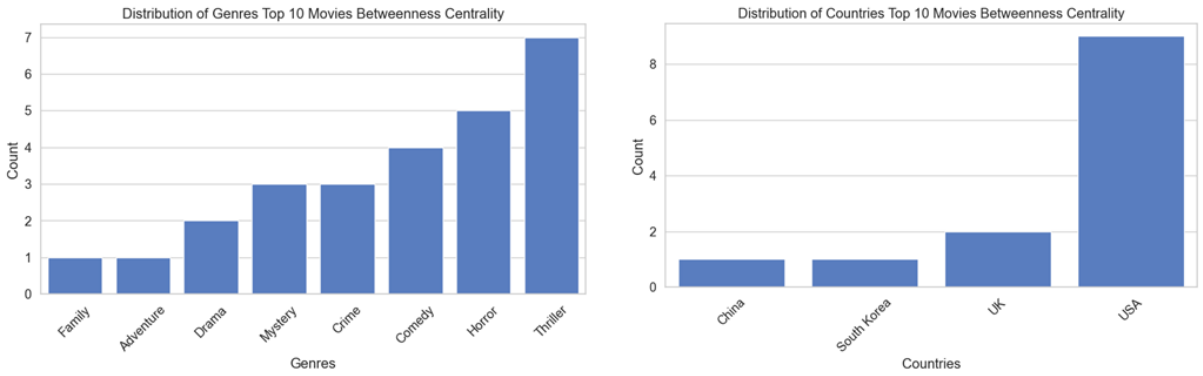

Figure 4. *Distribution of the genres and countries of the movies with the top 10 highest betweenness centralities*

Network Analytics: Bilouvain Algorithm

Our graph has two types of nodes, movies, and users, and edges occur only between nodes of different types, so our graph is bipartite. Therefore, we wanted to implement a community detection algorithm specifically for bipartite graphs. We have decided to implement the algorithm described in the paper "A novel community detection method in bipartite networks" by Zhou, Feng and Zhao [4]. The paper proposes a two-stage method for detecting communities in bipartite networks, which involves an extension of the Louvain algorithm and an agglomerative clustering approach. We have chosen this algorithm as we are already familiar with the normal Louvain algorithm, and it was, therefore, not too hard to understand. Furthermore, as the unipartite Louvain algorithm is quite efficient, we hoped this would also be the case for the bipartite implementation.

The main feature of a bipartite graph is that the nodes are divided into two disjoint sets, and connections exist only between nodes of different sets. Therefore, the proposed algorithm derived a formula for the gain of bipartite modularity, an extension of unipartite modularity tailored for bipartite networks.

The algorithm's first part is similar to the original unipartite Louvain algorithm, which consists of an assignment and an aggregation step (Fig. 5). These two steps are iteratively performed on each part of the network. During the community assignment step, nodes of one part are assigned to communities to maximize modularity. Once a satisfactory community assignment is achieved, the aggregation step starts, and nodes of the same type within the same community are aggregated into super-nodes. This process reduces the size of the network. These steps are performed alternatingly on the two parts of the bipartite network, progressively reducing the network size and balancing the two parts. The iterative process continues until the network becomes balanced, meaning each community contains only one super-node from each part. At this point, modularity gains stabilize, indicating an optimal modularity partition for the bipartite network.

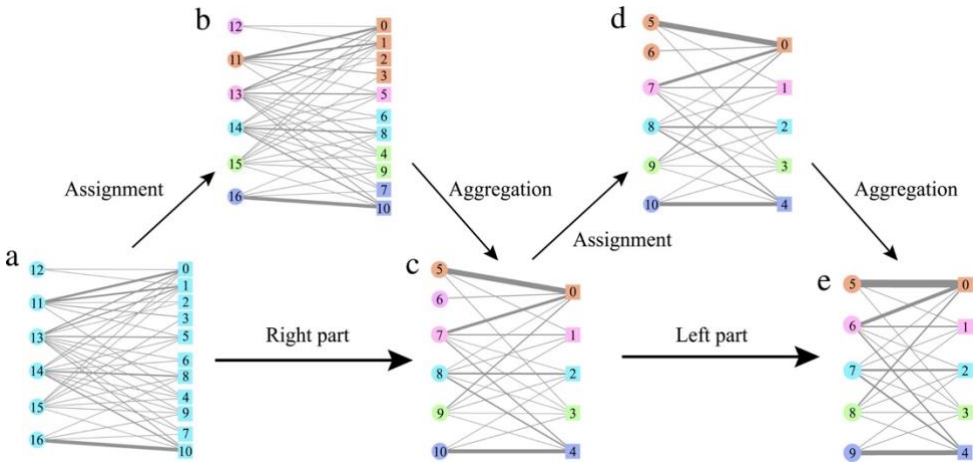

Figure 5. *Assignment and aggregation steps of bipartite Louvain are first performed on one part of the network and then on the other.*

To enhance the modularity and refine the community structure, an agglomerative clustering method is applied to the balanced bipartite network (Fig. 6). Pairs of balanced communities are iteratively joined in a manner that maximizes modularity. This is done using matrix operations to compute the modularity gain for all possible community pairs simultaneously. The authors propose different stop conditions on when to terminate the algorithm based on modularity or other criteria, such as the desired number of communities. In our implementation, we have terminated the algorithm when no combination of two communities would increase modularity anymore.

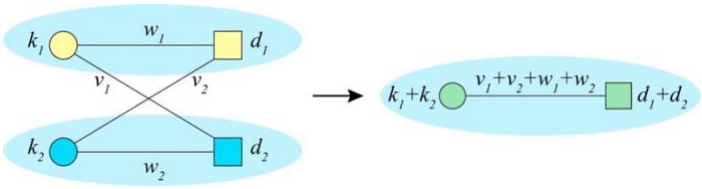

Figure 6. *Agglomerative clustering step of bipartite Louvain.*

For more detailed information on this algorithm, the original paper by Zhou et al. should be consulted.

Our implementation of the algorithm was mainly created by implementing the steps as they are described in the paper. Instead of checking modularity at each step, we continue with the next step if the communities remain stable from the beginning of an iteration to the end of the iteration of that step. To develop and test the algorithm and individual functions, we worked mainly with the Southern Women dataset, a small bipartite graph also used in the paper by Zhou et al. [4][5]. The algorithm was tested with that dataset at each step to ensure the communities were constructed correctly according to modularity gain. As we did not have the same node labels for this dataset as they did in the paper, we could not directly compare the obtained communities. However, we always obtained three to five communities, whereas four communities were found in the paper. In the case where we found four communities, the sizes of those communities and the number of nodes from each type were the same as shown in the paper; therefore, we assumed them to be the same and our algorithm implementation to work. However, we did not achieve the same modularity as in the paper. With a self-implemented bipartite modularity function, based on the formula in the paper, the calculated modularity was much higher than in the paper (0.878 vs. 0324), which seems unrealistic. With the bipartite modularity function from sknetwork, the modularity was very low (-0.002), which also seemed improbable.

Only when the algorithm's implementation was working and finalized did we start applying it to the dataset we had chosen to work with.

The paper states that the algorithm's first part, the adapted Louvain method, has the same complexity as unipartite Louvain, which would be linear in the number of edges. For the agglomerative clustering step, they state that it runs at worst-case in time $O(N^3)$, where N is the number of balanced communities the clustering method starts with. As it is hard to estimate the runtime with only the information of number of edges and nodes when we don't know the number of iterations and number of final communities, we cannot approximate the complexity of the whole algorithm. However, our implementation's runtime seems to be worse, respectively longer, than what they found in the paper.

Evaluation of our implementation

Choosing the sample with the best trade-off

Since our dataset is too large for our main algorithm, obtaining a result for the whole dataset in acceptable runtime was impossible, even after running the code for more than 10 hours. Hence, we needed to sample the dataset.

At first, we tried to run the algorithm with the dataset '*augm_movies400.csv*', containing only the movies with more than 400 ratings and all users. The set contained approximately 500 movies and 70.000 edges. However, due to the large number of edges, it took around 20 minutes to terminate, and it was not a suitable runtime for our purposes.

Second, we tried to sample the dataset by keeping all 557 users and getting 500 random movies from it. However, it took 123.36 seconds to run, making it unsatisfactory. Even with a very small number of movies out of 108.275 movies in the dataset, it took too long to complete.

At last, we decided to sample the nodes randomly based on edge number. We used sample sizes between 1000 and 10000. Initially, the sample with 5000 entries (edges) was chosen as the sample set with optimal trade-off since, from Fig. 7, it can be seen that the number and sizes of the communities don't change significantly. At the same time, runtime increases too much above the 1-minute threshold for larger sample sizes. We took one minute

as a threshold since other community detection algorithms take less than a minute for the whole dataset, and we wanted to avoid having a margin too large to make a reasonable comparison.

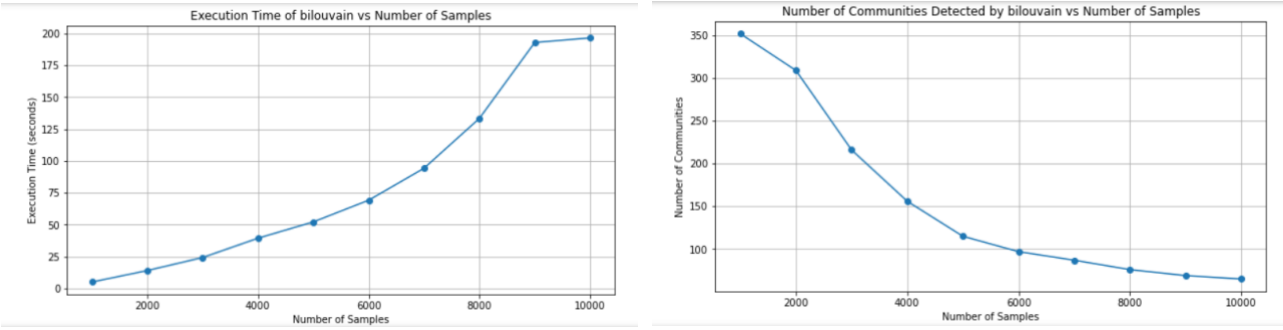

Figure 7. *Execution time and number of communities detected by our algorithm plotted over number of samples.*

However, the modularity value (modularity: -0.002344) was too low and the number of communities was too large (community: 115, size: 2). These values can be considered as out of the margin for comparison with other algorithms since they have modularity values around 0.20, and a negative modularity value means the network performs worse than a random one, so we changed our decision to use the sample with 10000 edges, as it can give a better result. However, the modularity value for 10000 edges was -0.001138, and the community number did not decrease too much (community: 65, size: 2).

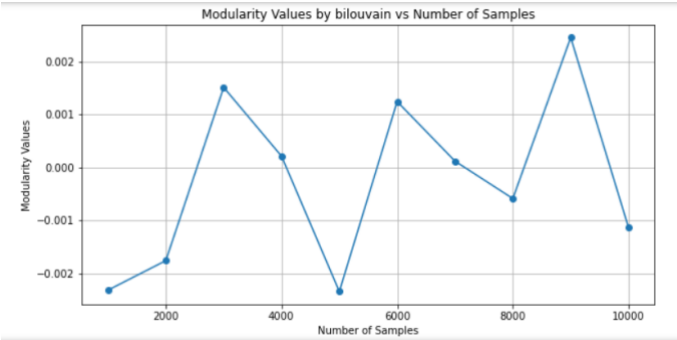

Figure 8. *Modularity of partitions detected by our algorithm plotted over number of samples.*

Then, we plotted the modularity for each sample to see where the turning point is. As seen in Fig. 8, the highest modularity is obtained from the sample with 9000 entries. Since it took around 2-3 mins, we think it is the best trade-off we can obtain and continue our evaluations with this sample set.

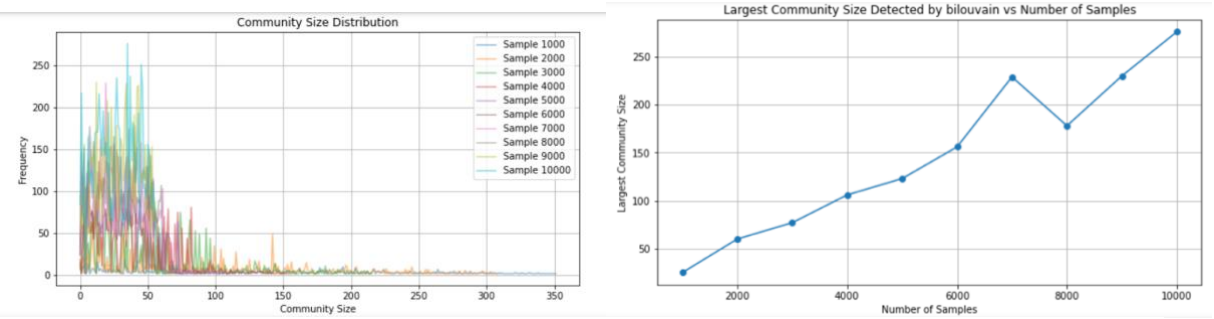

Figure 7. *Distribution of community sizes and largest communities detected per number of samples.*

We also changed the random state parameter when sampling with values like 1,10,15,42,99. We use the random state parameter to ensure reproducibility. Nonetheless, the results significantly vary between each run. While the execution time, number of communities detected, and largest community size detected follow the same increasing trend in each run, the values differ for each run. In addition, community size distribution is mostly under 100 nodes for all of the samples (Fig. 9). Furthermore, modularity values don't show a tendency for a direction, neither for running the algorithm one time for different samples nor for comparing different runs of the algorithm. This is because the algorithm detects different communities based on the order of nodes. The nodes with equal

modularity gain are added to the first community detected with the highest modularity gain, and each run changes the order. Also, the community size distribution may explain the low modularity results obtained for samples.

Results for the sample dataset

We decided to evaluate our algorithm based mainly on runtime and modularity. Runtime is an important metric considering the long execution time of our algorithm, and modularity is an accepted measure to assess the quality of community structures, both in the general literature and specifically for our algorithm, as it is used in the paper by Zhou et al. for comparison with other methods. Modularity quantifies how much more edges occur within communities than between them. A modularity value over 0.3 suggests strong organization into distinct communities [4].

For comparison algorithms, we used unipartite Louvain, as our implementation derives from it and is used in the paper by Zhou et al. [4]. Additionally, to assess the performance of our bipartite method, we have used the bipartite Louvain implementation from scikit-network [6]. Moreover, we considered using other algorithms from the paper and scikit-network like Label Propagation Algorithm (LPA) and K-centers clustering. However, LPA is a semi-supervised learning algorithm, and K-centers require a predefined clustering number, and we don't have the ground truth for our network. Therefore, we decided to use the Leiden algorithm from scikit-network as our third comparison method. The results from the comparison are summarized in Table 1.

Table 1. *Comparison of our implementation with other methods.*

Method	**Runtime (seconds)**	**Modularity**	**No. of communities**
Our implementation	139.5	0.002450	69
sknet- bilouvain	0.01924	0.7747	67
sknet-leiden	0.02515	0.7758	72
Unipartite louvain	0.6231	0.7755	68

The number of users in the sample set with 9000 edges was 545, and the number of movies was 6477. The bipartite modularity value from our implementation was 0.002450, while the sknet biLouvain algorithm resulted in 0.7747, and the Leiden algorithm resulted in a value of 0.7758. For unipartite Louvain, we calculated two different modularities since the assumption for the maximized modularity is the unipartite one for the algorithm. At the same time, we have bipartite modularity values from other methods for comparison. The unipartite modularity was 0.7755, indicating a good community structure. On the other hand, the bipartite modularity was -0.002049, giving contradictory results. Hence, we chose the unipartite modularity value obtained as the more accurate result, matching the assumptions of the algorithm.

The low modularity value may be due to an error in the calculation, as we obtained higher values that are outside of the expected [-1,1] margin of modularity definition, with the formula derived on the paper by Zhou et al. [4]. We compared the modularity value they obtained for the Southern women dataset, which is 0.3243, and our calculation, which is 0.8780, indicating our algorithm was better than all of the other methods discussed in the paper, including the authors' implementation. On the other hand, our implementation of the modularity formula gave 1.67 for our sample dataset, which is outside of the expected value range for modularity. We used the sknetwork's modularity function, which we tried with the sample dataset on their tutorial, *'movie_actor'*, and gave accurate results. The modularity function calculates Barber's modularity for bipartite graphs by default when the dataset is defined as bipartite. All of the results on the table were calculated with the sknetwork function for modularity calculation, except aforementioned unipartite Louvain value. They are reasonable values and match the value obtained from the '*community'* module for unipartite Louvain. Hence, although the number of communities detected is the same as that of other methods, we suspect that the quality of the found communities or the lack of a balanced structure after the aggregation step can cause the low modularity value for our implementation.

As mentioned before, our algorithm's runtime is excessive, at 139.5 seconds, whereas other methods take less than a second to terminate. This difference may be due to the better partitioning and usage of biadjacency matrices in initial steps, which other methods seem to leverage. Likewise, the aggregation step in our algorithm can cause a long runtime.

The number of communities detected is very similar for every method, and we think that our algorithm identifies the correct number of communities.

Using an augmented dataset to assess the results

To further analyze the quality and structure of the communities that our implementation has found, we took inspiration from the paper by Zhou et al. [4] and plotted histograms for the largest eight communities, counting the genres, release dates, and country information about the movie nodes. If we can observe some accumulation towards a specific genre, release date, or country of production, we could assume that the found community captures the users' preference towards that particular feature. Furthermore, these results can provide tools for recommendation systems for future applications. We include a few of the plots obtained here in the report for the sake of readability. The reader can refer to the notebook '7.3_Evaluation_augmented_data.ipynb' for all the results.

As can be seen from Fig. 10 for the first community, the users have watched mostly drama and thriller movies, while for the release date, the movies are either from the 80s, 90s, or 2010s. Additionally, movies are produced mainly in the USA, followed by Germany and Hong Kong.

From Fig. 10, one can observe that for the second community, the genres are equally distributed between Comedy, Action, Adventure, and Sci-fi. At the same time, they are made only in the USA, and only in 2004 and 2014.

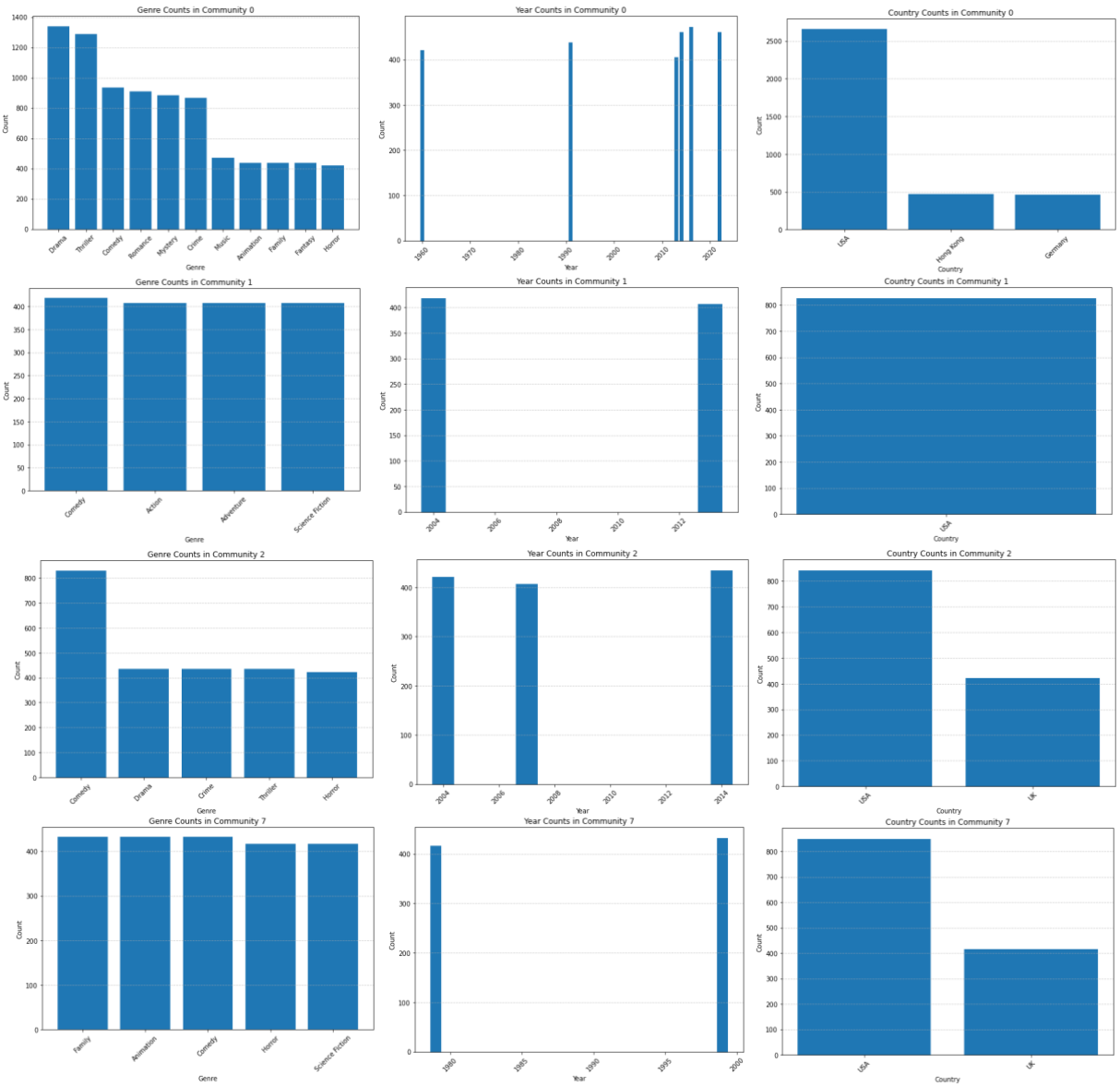

***Figure 8**. Genre, year, and country counts of four different communities.*

Fig. 10 shows that the users in the third community are watching comedy movies most, from 2004, 2007 and 2014. The country plot shows that the movies are either from the USA or the UK.

While community 7 has a significantly similar distribution of countries to community 2 and similar genres to community 1, it differs from them in the release date distribution. This confirms that even with data with similar features, our algorithm was able to capture the differences in other features and form communities based on more specific user tastes.

Our results shed light on the relationship between the dataset's features and the communities detected by our algorithm. There is evidence for some preference among users in the same community. Moreover, our findings for the communities further confirm the USA's centrality.

Results for the whole dataset

Unfortunately, we could not obtain results with our algorithm for the whole dataset. Nevertheless, we obtained satisfactory and matching results for other methods.

Table 2. *Results obtained from other methods on the whole dataset.*

Method	**Runtime (seconds)**	**Modularity**	**No. of communities**
sknet- bilouvain	11.88	0.2384	8
sknet-leiden	13.46	0.2385	9
Unipartite louvain	46.26	0.2394	9

Table 2 shows that all the algorithms approximately give the same modularity and number of communities while the execution times increase, especially for unipartite Louvain. Further, from Fig. 11, it can be seen that the final structure obtained is a balanced bipartite graph.

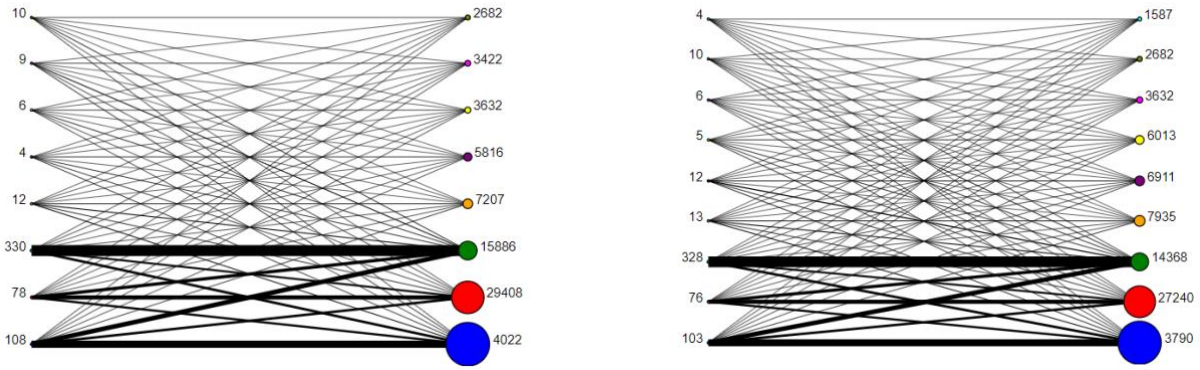

Figure 9. *Obtained from bilouvain implementation (left) and Leiden implementation (right) from sknetwork library, it shows the aggregated graph with user nodes on the left and movie nodes on the right. Colors depict the communities, and numbers denote the number of nodes in the hypernode.*

Table 3. *Unipartite Louvain Communities.*

Community sizes obtained from Unipartite Louvain implementation for the whole dataset		
community: 0, size: 15542	community: 3, size: 2706	community: 6, size: 9662
community: 1, size: 26439	community: 4, size: 3482	community: 7, size: 6483
community: 2, size: 39173	community: 5, size: 3484	community: 8, size: 1861

As can be observed from Fig. 11 and Table 3., community sizes for each method are similar, and consistent results have been attained.

Network Visualization

We experimented with various visualization tools, including Gephi, Cosmograph, and the networkx library in Python, to visualize either the entire dataset or a sample. Visualizing the whole dataset proved ineffective due to the overwhelming number of nodes and edges. Therefore, we chose to do sample visualizations, which, however, also provided limited informative value. Ultimately, we selected networkx for visualization because it allowed for easy manipulation of parameters such as layout and colors.

Fig. 12 illustrates a sample visualization featuring the 400 most-rated movies and all users who rated them. We used the "spring layout," a force-directed representation of the network. In this visualization, movies cluster at the center while users are distributed around them. It is important to note that none of the movies are directly connected. Connections only exist between nodes of different types. The clustering of movies results from the distribution of users around them, as a single user can be linked to many movies. This leads to a visualized structure that does not resemble a typical unipartite graph with distinct communities.

When we attempted to color the nodes according to their communities identified by our algorithm, the expected community structure was not visible. The same layout caused movies to cluster in the middle, obscuring the community distinctions. This issue arises from the bipartite nature of our network, with connections only between movies and users, not within the same node type, and especially the vast number of edges that connect each movie to more than 400 users.

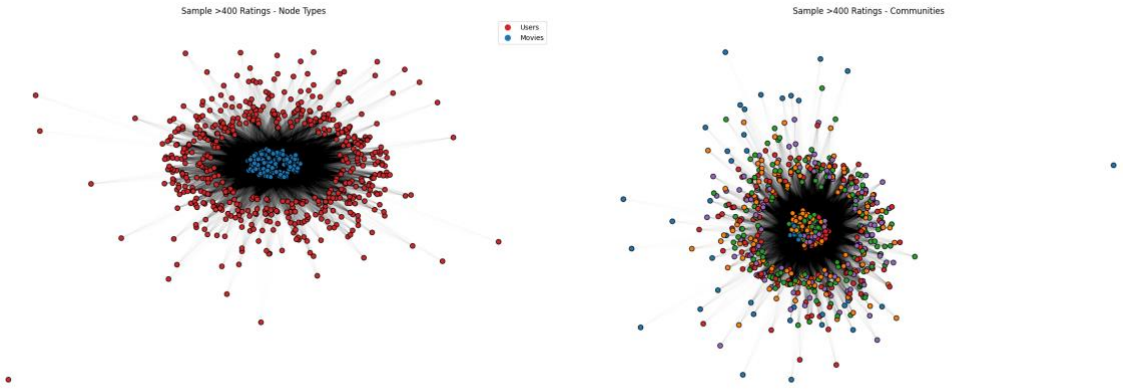

Figure 12. *Visualization of a sample containing only movies with more than 400 ratings, nodes colored according to node type (left) and communities (right).*

Using a sample of 500 random movies and all users who rated them, we observed that nodes in the same community cluster more closely together. This improvement is likely due to fewer ratings per movie on average, preventing the central clustering seen in the previous sample.

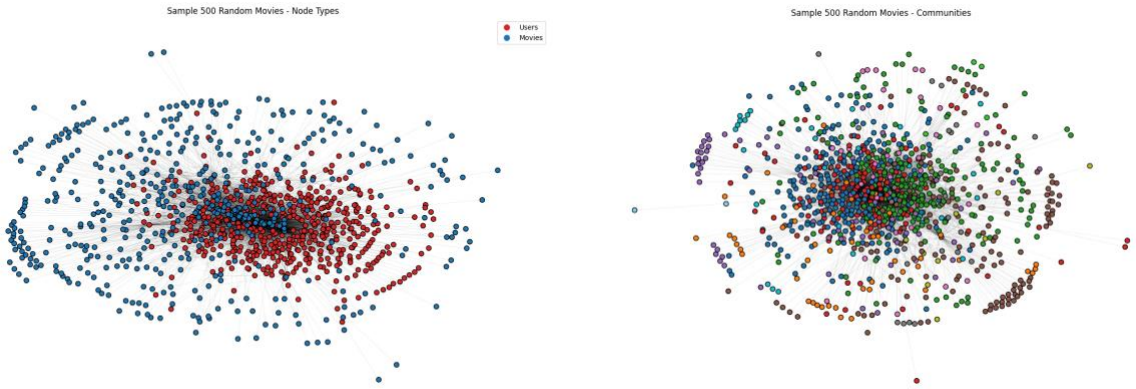

Figure 13. *Visualization of a sample containing 500 movies and all users who have rated them, nodes colored according to node type (left) and communities (right).*

Limitations and Future Directions

Our implementation of the algorithm still has some limitations and could be further improved. First of all, the order of the nodes when we iterate over multiple nodes is not randomized. This is the case because we mostly use sets to store groups of nodes. We chose to do this to ensure that nodes are not present more than once. However, as sets do not have an order, we cannot randomize the order we iterate over them. Still, the order is not the same every time due to this inherent property of sets. Therefore, we can still observe that if the algorithm is run multiple times on a dataset, different community partitions are detected due to a different order of nodes.

To truly randomize the node order at every step, though, the sets would have to be converted to lists, and the lists would have to be shuffled. To do this, multiple operations would also have to be changed to work on lists. This would for sure be possible. However, we were not able to do it and make it work correctly in the provided time frame.

Furthermore, our implementation includes many loops, which make the algorithm's runtime on a large dataset very long. The implementation of the algorithm could be improved with more time and coding experience.

To make the algorithm publicly available, it would have to be revised extensively. It currently includes many comments and descriptions, which made the development process easier and helped us understand what we did and contemplated at each step. The format would also have to be adapted to follow general guidelines like PEP 8, which it does not entirely follow so far.

As there is no public implementation of the algorithm proposed in the paper, we haven't been able to compare our algorithm with a different implementation of the same algorithm. This would be interesting to do, if the implementation of the authors could be used for comparison. Unfortunately, we have not been able to reach them.

It would also be interesting to apply our algorithm to different datasets with similar structures. One example could be to apply it to data from Goodreads, a similar social network to Letterbox, but for books instead of movies.

Graph Persistence

We have decided not to use a graph database for our project. We have mainly looked into Neo4j, but we did not see any additional benefits we would get from using it. Furthermore, as this project was already very time-consuming, we wanted to use interfaces and languages we were already familiar with to spend more time doing analyses than first learning to work with a new application and interface.

References

[1] R. Vámosi, "Letterboxd ratings (1.4M)." Kaggle. Accessed: Mar. 05, 2024. [Online]. Available: https://www.kaggle.com/datasets/rbertvmosi/letterboxd-ratings-14m/data
[2] S. Garanin, "Letterboxd." Kaggle. Accessed: Mar. 27, 2024. [Online]. Available: https://www.kaggle.com/datasets/gsimonx37/letterboxd
[3] A. Jangra, "IMBD Movies Dataset." Kaggle. Accessed: Mar. 27, 2024. [Online]. Available: https://www.kaggle.com/datasets/ashishjangra27/imdb-movies-dataset
[4] C. Zhou, L. Feng, and Q. Zhao, "A novel community detection method in bipartite networks," *Physica A: Statistical Mechanics and its Applications*, vol. 492, pp. 1679–1693, Feb. 2018, doi: 10.1016/j.physa.2017.11.089.
[5] A. Davis, B. B. Gardener, and M. R. Gardener, "Southern women (large)." KONECT. Accessed: Apr. 15, 2024. [Online]. Available: http://konect.cc/networks/opsahl-southernwomen/
[6] scikit-network, "scikit-network Clustering." Accessed: May 10, 2024. [Online]. Available: https://scikit-network.readthedocs.io/en/latest/reference/clustering.html#module-sknetwork.clustering