CS-E4600 - Algorithmic Methods of Data Mining
Final Project

Aleksi Hämäläinen (425287)   Aarno Vuola (355085)

December 21, 2018

1 Introduction

In this final project, the goal was to explore various graphs and to find community structures within them. One algorithm, spectral clustering, was required to be implemented, as well as some own algorithm. We implemented and tested five different algorithms for graph community clustering, which all had their strengths and drawbacks. Rigorous exploration of results can be found in Sec. 4.

Graph communities are defined as sub-graphs that have many interconnections within and few connections to outside of the clusters. With this in mind, the objective score defined in the project description was

$$\phi(V_1, \ldots, V_k) = \frac{|E(V_1, \ldots, V_k)|}{\min_{1 \leq i \leq k} |V_i|} \tag{1}$$

where $E(V_1, \ldots, V_k) = \{(u, v) \in E | u \in V_i \text{ and } v \in V_j \text{ with } i \neq j\}$, i.e. the set of edges cut by the communities. The denominator is the size of the smallest cluster.

A heavy emphasis on the associated competition was placed. We mainly focused on improving the scores on the leaderboard, which is reflected in the results compared to the basic graphs not part of the competition. Significantly more running time was given to the competition graphs, and hyperparameter tuning was conducted in more detail with them. However, many different experiments were done for the non-competition graphs as well.

This report begins with a data exploration section, after which the various methods used are detailed. These are followed by the results section, where experiments are detailed and reported. Finally, the results are discussed and some conclusions are drawn.

1.1 Contributions by team member

Aarno Vuola implemented the initial spectral clustering algorithm, fast modularity algorithm, label propagation algorithm and DeepWalk algorithm.

Aleksi meanwhile made the balanced k-means algorithm, and afterwards targetet most effort to the "algorithm gym", where different algorithms could be easily run with different graphs and options. Both contributed equal amount of work to this project and this report. Literature review was conducted, and the papers are referred to in Sec. 3.

2 Data

For an algorithm comparison, the following datasets were utilized:

1. ca-AstroPh: The collaboration network of authors that have submitted to the Astro Physics category in Arxiv [6].
2. ca-CondMat: The collaboration network of authors that have submitted to the Condense Matter Physics category in Arxiv [6].
3. ca-HepTh: The collaboration network of authors that have submitted to the High Energy Physics - Theory category category in Arxiv [6].
4. ca-HepPh: The collaboration network of authors that have submitted to the High Energy Physics - Phenomenology category in Arxiv [6].
5. ca-GrQc: The collaboration network of authors that have submitted to the General Relativity and Quantum Cosmology category in Arxiv [6].
6. Oregon-1: Autonomous Systems peering information inferred from Oregon route-views on 31th of March in 2001 [7].
7. soc-Epinions1: A trust relationship between online users in the Epinionson service [8].
8. web-NotreDame: Hyperlink relationships between pages within University of Notre Dame service. [9].
9. roadNet-CA: A road network of California. [13].

In our project, each of the graphs are considered as an undirected graph. In addition, only the largest connected component of each graph is considered. Properties of the graphs can be observed from the table 1. The average clustering coefficients and 90-percentile effective diameter are calculated based on the original graphs. These values were given by SNAP [15]. Hence, these values are only approximations of the processed graphs that were utilized in our project. Nevertheless, some insight how the graphs differ from each other can be observed based on the approximations.

No major difference between the collaboration networks can be observed. The average clustering coefficient of ca-HepTh is slightly smaller than the rest of collaboration networks. soc-Epinions1, soc-Epinions1, web-NotreDame and roadNet-CA are larger than the collaborative networks, and their average clustering coefficients are smaller than the collaborative networks have. Nevertheless, the 90-percentile effective diameters of soc-Epinions1, soc-Epinions1 and web-NotreDame are close to the diameters of the collaborative networks. roadNet-CA differs mostly from the rest of graphs, as its 90-percentile diameter 500 and average clustering coefficients is 0.0464. This may indicate that properties of a road network differs from a social network.

Dataset	Nodes	Edges	Average clustering coefficient	90-percentile effective diameter
ca-AstroPh	17903	197031	0.6306	6.5
ca-CondMat	21363	91342	0.6334	6.5
ca-HepTh	8638	24827	0.4714	7.4
ca-HepPh	11204	117649	0.6115	5.8
ca-GrQc	4158	13428	0.5296	7.6
Oregon-1	10670	22002	0.2970	4.4
soc-Epinions1	75879	405740	0.1378	5
web-NotreDame	325729	1117563	0.2346	9.4
roadNet-CA	1957027	2760388	0.0464	500

Table 1: Properties of the graphs

Figure 1 shows degree distrubtions of the graphs in logarithmic scale. It has been shown, that many graphs follow a power-law degree distribution [14]. From our graphs, Oregon-1 and soc-Epinions1 follow most closely the power-law degree distribution, as they are nearly linear in logarithm scale. ca-HePh, ca-GrQc, ca-AstroPh and web-NotreDame have more small and middle degrees to achieve equal linearity as Oregon-1 and soc-Epinions1. On the other hand, RoadNet-Cat does not follow linearity. Again, nature of RoadNet-Cat may be the reason for different behavior. More than one connections occur in road networks. The zero space in the axes show, that some degrees do not occur graphs at all. Especially, at the end of the tails, the zero occurrences seems to be more likely. However, no assumptions can be made from the zero degree occurrences.

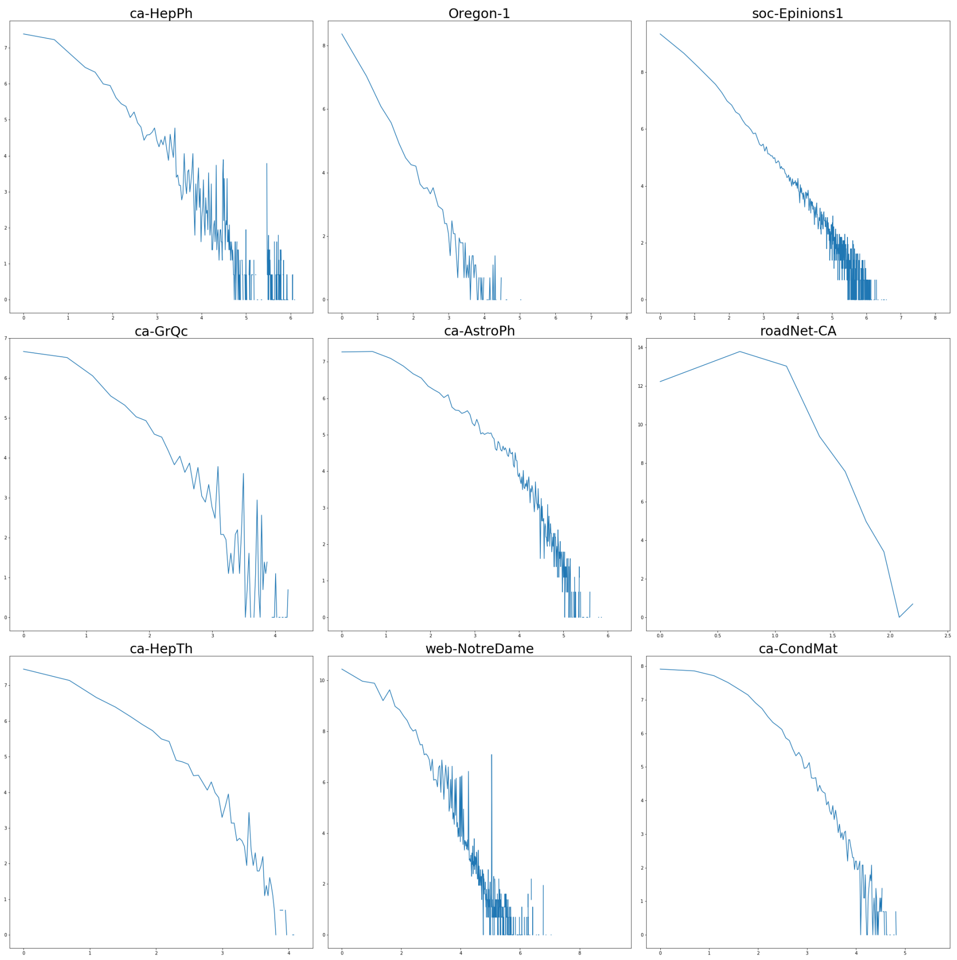

Figure 1: Degree distributions of the graphs in logarithmic scale. Oregon-1 and soc-Epinions1 follow most closely the power-law degree distribution. Due to nature of RoadNet-Cat, it produces a degree distribution, that does not follow the power-law degree distribution.

3 Method

Multiple different algorithms were developed and tested in this project. The implemented algorithms were

1. Spectral clustering
2. Modularity optimization
3. Label propagation
4. DeepWalk

Many different approaches to the community clustering problem exist. They can be split broadly into three categories: spectral, hierarchical and agglomerative. We focused on spectral and agglomerative approaches in this project, as the hierarchical approach was thought to be computationally too intensive. The spectral clustering approach naturally belongs to the spectral category, and for this approach two algorithms were implemented. Modularity optimization and label propagation are agglomerative algorithms, where each vertex starts as its own community and is afterwards merged to another based on some rules and optimization methods. Finally, the DeepWalk algorithm does not clearly fall into any of these categories, as it calculates feature vectors for each vertex in the graph which can be used for clustering. Literature of these different approaches are referred to in their respective sections.

The next sections describe these algorithms, and their strengths and weaknesses.

3.1 Spectral clustering

This was the basic algorithm that was required to be implemented in the project. It works by first computing the Laplacian of the graph adjacency matrix, calculating the eigendecomposition of the Laplacian and using the first k eigenvectors as node feature vectors for clustering. This clustering on the eigenvectors relies on the following property of the Laplacian: in the ideal case of vertices in different clusters being infinitely far apart, i.e. not connected, the Laplacian will be block diagonal. Computing the eigenvectors of such matrix with $k = 3$ clusters results in [1]

$$\hat{X} = \begin{bmatrix} x_1^{(1)} & \vec{0} & \vec{0} \\ \vec{0} & x_1^{(2)} & \vec{0} \\ \vec{0} & \vec{0} & x_1^{(3)} \end{bmatrix} \in R^{n \times 3} \qquad (2)$$

which is trivial to cluster. In the more general case, where the points between different clusters are sparsely connected, such behaviour from the eigenvectors is still observed.

Clustering the eigenvector-representation can be done with e.g. the k-means algorithm, resulting in cluster labels for all nodes in the graph. In this project, this approach was implemented and it produced very good results on most graphs.

In addition to k-means, a *balanced* k-means algorithm powered by with the kmean++ was developed after considering the objective function defined by the project description, shown in Eq. 1. The denominator is the size of the smallest cluster found, and simple k-means initialization was with some graphs observed to produce very small size clusters. The balanced k-means clusters the eigenvectors by keeping the sizes of different clusters approximately constant. In some graphs, this approach achieved increased performance.

It was observed that both computing the Laplacian and especially its eigendecomposition was very unstable. Initially, these were implemented by hand, but it became clear that more advanced methods were required, especially after moving to sparse adjacency matrices for the larger graphs which were unstable to decompose. Our second attempt was to use scipy-library [5] to calculate the laplacian and eigendecomposition, which did increase the stability. However, results still differed based on the random state with which the eigendecomposition function was initialized with. This led us to delve into sklearn's source code to find a stable implementation of eigendecomposition for sparse matrices: this finally removed the unwanted instability.

In the results, spectral clustering with the basic k-means is marked with **kmeans**, and spectral clustering with balanced k-means with **balanced**.

3.2 Modularity optimization

As calculation of the eigendecomposition and clustering with the spectral algorithm was computationally intensive, a small literature review was conducted to find alternatives. An algorithm based on modularity optimization of the communities was developed in [3], which was one of the algorithms chosen for implementation. This approach initially considers each node in a network as its own community, and then iteratively merges communities that would result in increase in the network modularity. The *modularity* of a network with adjacency matrix A, number of edges $m = \frac{1}{2}\sum_{vw} A_{vw}$ and node degrees $k_v = \sum_w A_{vw}$ is defined as [2]

$$Q = \frac{1}{2m}\sum_{vw}\left[A_{vw} - \frac{k_v k_w}{2m}\right]\delta\left(c_v, c_w\right) \tag{3}$$

where $\delta(i, j)$ is 1 if $i = j$ and 0 otherwise. Large values of Q indicate a good community structure, and consequently maximizing this value should result in good communities. This is somewhat analogous to the objective function defined in the project, leading us to believe that this approach could produce good results in the competition.

In [3], this is done by using a greedy algorithm that merges communities which would result in the largest increase in the Q value. Instead of calculating these changes for each possible community in each iteration, a $\Delta Q \in \Re^{d \times d}$ matrix that stores the changes in modularity that would result in merging communities i and j, where d is the number of nodes in the network. To construct this matrix, the modularity equation for Q can be decomposed into (see [2] for details)

$$Q = \sum_i \left(e_{ii} - a_i^2\right) \tag{4}$$

where

$$e_{ij} = \frac{1}{2m} \sum_{vw} A_{vw} \delta\left(c_v, i\right) \delta\left(c_w, j\right) \tag{5}$$

is the fraction of edges joining nodes in community i to community j, and

$$a_i = \frac{1}{2m} \sum_v k_v \delta\left(c_v, i\right) \tag{6}$$

that is the fraction of edges that end in community i. Initially, each node is its own community, and $e_{ij} = 1/2m$ if communities i and j are connected and zero otherwise, and $a_i = k_i/2m$, and the ΔQ matrix is at time step $t = 0$ set to be

$$\Delta Q_{ij} = \begin{cases} 1/2m - k_i k_j/(2m)^2 & \text{if } i, j \text{ are connected} \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

Each iteration finds the largest element in ΔQ, marked with $\Delta Q_{i,j}$, and merges communities i and j. Afterwards, ΔQ is updated with the following rules: for each community $k \neq i, j$, if k is connected to both i and j:

$$\Delta Q'_{jk} = \Delta Q_{ik} + \Delta Q_{jk}, \tag{8}$$

if k is connected only to i:

$$\Delta Q'_{jk} = \Delta Q_{ik} - 2a_j a_k \tag{9}$$

and if k is connected only to j:

$$\Delta Q'_{jk} = \Delta Q_{jk} - 2a_i a_k \tag{10}$$

Three data structures are required: The $\Delta Q \in \Re^{d \times d}$ matrix storing the changes in Q, adjacency matrix A that stores the community-wise connections, and an array that stores which nodes belong to which communities.

When communities i and j are merged, the adjacency matrix row j is updated with connections from row i, and the community array updated. Finally, row and column i are removed from Q and A. The main performance decrease comes from these removals of rows and columns from these sparse matrices. Initially, conversions between different sparse matrix formats were made, but this was very slow and led us to search and implement more efficient methods for these operations and to consider which sparse matrix type to use in which situation.

At some point, the ΔQ matrix is zero and the optimal community structure is found. However, the number of communities at this point was often still too large compared to the desired number of clusters k inputted, and the solution was to stop the algorithm here and combine communities in the order of their size, starting from the smallest communities, until the desired number of clusters was reached. This is most likely not an optimal strategy, but suffices here as the objective function penalizes for finding small communities. However, for the non-competition graphs, the

A faster implementation using heaps was developed in [2], but this was deemed to be out of scope for this project. This is not an ideal result, as we were unable to run the slower algorithm with the largest graphs. An even faster method that merges multiple communities exists as well, detailed in [4], which would have been nice to implement as well but was impossible due to time constraints.

In the results, the scores from this algorithm are marked with **Fast**, somewhat ironically as our implementation was rather slow with the larger graphs, and indeed too slow for two of the largest.

3.3 Label propagation

Another interesting algorithm that we came across during the literature review was *label propagation* [10]. This is a simple algorithm, that similarly to the modularity optimization initially considers each node in the graph as its own community. It then does several passes through the data, where in each step each node's label is updated with the maximum of its neighbor's labels, breaking ties randomly. This makes the labels propagate through the network, as densely connected nearby nodes will share the same label after only a few iterations. This is fast and efficient to compute, as the labels reach a consensus quickly. However, the computers this was tested on ran out of memory with the largest graphs, which was disheartening as this was especially developed to get fast results with them.

The algorithm works as follows:

1. Set each node's label $x_i = i$

2. Permute the order in which nodes are visited

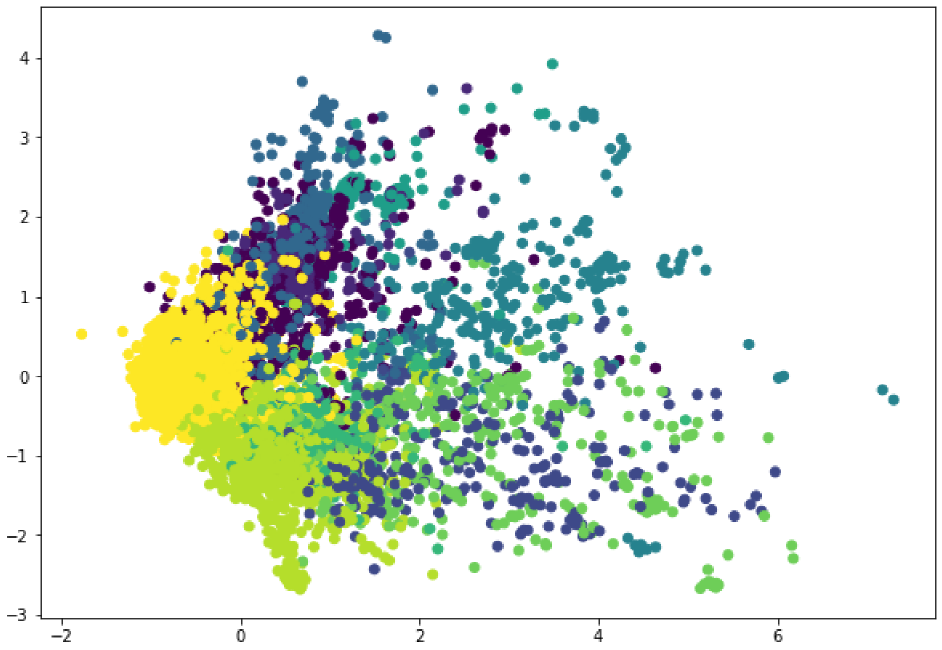

Figure 2: Oregon-1 dataset embedded with DeepWalk and reduced to 2D using PCA. Colors correspond to clusters, calculated using k-means with $k = 5$.

3. In the permuted order, visit each node and assign its label to be the maximum of its neighbors, breaking ties randomly

4. Repeat from step 2 until a consensus is reached

The number of unique communities resulting form this process was rarely the number of desired clusters, and similarly to the modularity algorithm, this was solved by merging smallest communities until the number of communities equalled the desired number. Again, for non-competition graphs, the algorithm was also tested when skipping the post-processing step of community merging, and the corresponding k reported.

3.4 DeepWalk

An interesting clustering method for community detection, named DeepWalk, was developed in [11], where a *Word2Vec* [12] model was used to calculate d-dimensional feature representations for nodes. The Word2Vec model calculated features for *words* by predicting the context where they appear using a neural network architecture, and using a gradient descent to iteratively construct feature vectors for words which make this prediction possible. The training data for the model is usually large documents, out of which it learns contexts and feature vector representations for words. In the DeepWalk pipeline, these documents are *random walks*, with words being the index of the node these walks pass through and documents all nodes in some walk. For example, a random walk starting from node i and ending in node j, the document resulting from a random walk could be e.g.

$$Walk(i, j) = [''i'', ''1'', ''2'', ''3'', ..., ''j''] \tag{11}$$

if the subsequent nodes were connected. A random walk is a path through the adjacency matrix starting from some node i, choosing the next node to visit randomly (weighted or unweighted, based on if the edges have weights) from its neighbors, adding this new node to the path and repeating the process in this new node. The random walk terminates when the desired number of steps is reached.

In the DeepWalk model, a random walk of length t is started γ times from each node, where t, γ are hyperparameters of the model. This results in a large number of walks, which can then be used to train the Word2Vec model to get feature representations for each node in the graph. These nodes are then clustered based on their features using k-means algorithm.

This algorithm was simple to implement and produced overall good results. For the competition graphs, various hyperparameters (such as the size of the feature representation d) were tuned for each different graph and the Word2Vec model trained longer, resulting in good performance. For non-competition graphs, the model was ran with the default settings, and the results are consequently somewhat worse. The algorithm was also the fastest one after labelpropagation for larger networks, as creating the random walks and training the network could be parallelized.

DeepWalk allows for nice visualization of the network. Fig. 2 shows a PCA-transformed embedding for Oregon-1 graph, which has been clustered using the k-means algorithm with $k = 5$ clusters. Plotting the graph based on its nodes and edges produces plots that do not help with drawing any conclusions from the network, but in this visualization it is easy to spot outliers and the basic graph structure. While DeepWalk did not produce the best results in this project, it helps with visualization tremendously.

3.5 Code

This project contains two main files: helpers.py and algo_gym.py. Helpers contain helper functions, as well as the implementations for the different models. The algorithm gym is a script that makes it possible to run different combinations of graphs, algorithms and cluster sizes easily, a crucial tool in a research project like this.

4 Results

These results are divided into two parts: competitive and non-competitive graph results. For the competetive graphs, the emphasis was on reaching the highest score on the leaderboards, and little effort was made to test different cluster sizes. However, with the non-competitive graphs, this exploration of the differences of the algorithms with different cluster sizes was a focal point when designing experiments.

4.1 Competitive

Table 2 shows the final results for all competition graphs. It is nice to see that two of the five graphs had the best results with our own algorithm implementations. However, it seems like the spectral clustering approach with k-means is the best all-around method for community detection, as it had the best score with three out of five graphs. On the other hand, in Oregon-1 graph, it had significantly worse performance than any other algorithm, if label propagation is not considered as it had poor performance in all situations. This makes it clear that spectral clustering with k-means it no panacea to community detection, and that other algorithms should be tested before committing to it in production.

The cells marked with "X" in the table were algorithms that were too intensive to run within the timeframe of the project, and most likely too intensive to run in *any* project. Label propagation crashed with a segmentation fault when tested with the larger graphs, and as its results were poor with the smaller graphs, this issue was not fixed. The balanced k-means performance could probably be improved somehow, although it is not clear how. However, the biggest failure of this project was the failure to implement a faster version of the modularity optimization algorithm, as it would have been very interesting to see its performance with the larger graphs. The bottleneck was the sparse matrix row and column deletions, which scaled nearly linearly in time. This was corrected with the implementation in the article [2] and even further in [4], but we did not have enough time to implement these approaches.

4.2 Non-competitive

With the non-competitive graphs, a more thorough analysis was conducted. The basic evaluation based on the k-values given in the graph files for all algorithms is shown in Table 3. One again, k-means had good performance, but the modularity optimization algorithm had the edge in with many graphs, and was very near the k-means solution when clustering the ca-HepTh graph. DeepWalk has poor performance the larger the k-value is, and as such with these large cluster sizes it had poor performance. Label

Table 2: Objective function score for competition graphs with all algorithms. X means that the algorithm was too slow to calculate the score.

Graph	kmeans	balanced	fast	labelprop	deepwalk	k
ca-GrQc	**0.075**	0.410	0.518	3.416	0.331	2
Oregon-1	14.30	**2.968**	4.28	108.445	4.55	5
soc-Epinions1	4.0	31.523	**3.552**	512.828	61.999	10
web-NotreDame	**0.594**	X	X	X	8.373	20
roadNet-CA	**0.357**	X	X	X	27.099	50

Table 3: Objective function score for non-competition graphs with all algorithms, with k defined in the graph file.

Graph	kmeans	balanced	fast	labelprop	deepwalk	k
ca-HepTh	**11.0**	23.1	13.9	689.354	97.418	20
ca-HepPh	105	117	**88.0**	963.820	638.6785	25
ca-CondMat	258	543	**103**	81609.00	3204.2222	100
ca-AstroPh	227	491.205	**54.05**	3386.851	10726.166	50

propagation was once again the worst algorithm, but on the other hand also the fastest, which makes it suitable for situations where time is of essence.

Table 4 shows results when varying the number of clusters. Here it is easy to see that DeepWalk has very poor performance when the number of clusters is increased when compared to any other algorithm. A conclusion drawn from this is that it is mainly suitable for small-size community detection and for visualization purposes. Other algorithms' results do not degrade in such a harsh manner, but some decrease in performance is observable when the number of clusters is increased. It seems that the AstroPh-graph is the most challenging for most of the algorithms, especially k-means, but it is notable that the modularity optimization approach does very well here, outperforming others by a significant margin with every graph.

One other interesting result is the number of *optimal* clusters modularity optimization and label propagation find. Table 5 shows results for these algorithms when they were stopped after the optimal number of clusters were achieved. The discussion here focuses on the modularity optimization algorithm, as label propagation had poor performance across the board. For AstroPh graph, modularity optimization found the optimal number of clusters to be 52, with the same score as in 4 for $k = 50$. However, as can be seen from the latter table, merging the clusters decreased the objective score significantly. This results indicates that a smaller number of communities decreases the objective score, even if the optimal number of clusters would be larger. Perhaps the objective function specified in the project could be

Table 4: Objective function score for non-competition graphs with all algorithms, with varying k-values.

Graph	kmeans	balanced	fast	deepwalk	k
ca-HepTh	0.700	4.059	2.8281	3.456	5
ca-HepPh	1.187	8.587	5.469	8.207	5
ca-CondMat	1.333	5.903	3.187	6.332	5
ca-AstroPh	11.300	21.211	2.945	13.069	5
ca-HepTh	6.0	9.456	6.464	14.085	10
ca-HepPh	7.2	25.215	11.0	19.118	10
ca-CondMat	4.0	14.657	6.833	26.908	10
ca-AstroPh	12.9	53.368	6.812	355.971	10
ca-HepTh	17.6	22.944	13.923	293.866	20
ca-HepPh	124.6	77.859	22.0	718.529	20
ca-CondMat	11.714	35.687	15.222	75.591	20
ca-AstroPh	124.0	124.797	13.625	547.881	20
ca-HepTh	78.4	79.638	45.375	581.0	50
ca-HepPh	244.25	433.907	88.0	2050.364	50
ca-CondMat	52.0	126.719	51.375	3771.285	50
ca-AstroPh	227.333	491.205	54.5	8610.25	50

updated to reflect this property.

For other graphs, the optimal k found with modularity optimization was rather large compared to the specified ones in the graph files. However, the approach of merging communities based on their size seems to work well, which is not surprising considering that it was developed with the objective function in mind.

Table 5: Objective function score for non-competition graphs for Modularity optimization and Labelprop, with stopping at their optimal community structure. The optimal number of clusters shown in parantheses next to the score.

Algorithm	fast	labelprop
ca-HepTh	181.5 (165)	21560.0 (41)
ca-HepPh	88.0 (80)	107348.0 (124)
ca-CondMat	205.5 (171)	81525.0 (80)
ca-AstroPh	54.5 (52)	184390.0 (169)

5 Discussion & Conclusion

The spectral clusering algorithm showed its power in this project. In the competition, most of the best results per graph were achieved with it and it was computationally more efficient than the modularity optimization. Label propagation proved to be a fast algorithm, but very unstable: results varied from run to run significantly. DeepWalk had a solid performance with all graphs, but failed to achieve best possible performance with any graph. This approach could be further improved, for example by using Bayesian optimiziation for optimizing the hyperparameters of the architecture. Its performance suffered especially when the number of clusters was increased, which indicates that it can create feature vectors that approximate the node properties, these vectors are not good enough to split them into more than a few categories.

Our modularity optimization implementation was very slow to calculate, taking hours on powerful machines. Better performance may be achieved, when the number of desired cluster is large, than k-mean. It would have been nice to test the modularity optimization algorithm on the larger graphs as well, but due to time constraints it was not possible to implement the faster version. Also, in the case of short-tailed distributions, the modularity optimization approach is recommended.

Label propagation had poor performance with all graphs when compared to other approaches. However, it was significantly faster to calculate, and as such could be used in situations where just a broad overview of the community structure within a graph is desired.

References

[1] Ng, A.Y., Jordan, M.I. and Weiss, Y., 2002. On spectral clustering: Analysis and an algorithm. In Advances in neural information processing systems (pp. 849-856).

[2] Clauset, A., Newman, M.E. and Moore, C., 2004. Finding community structure in very large networks. Physical review E, 70(6), p.066111.

[3] Newman, M.E., 2004. Fast algorithm for detecting community structure in networks. Physical review E, 69(6), p.066133.

[4] Schuetz, P. and Caflisch, A., 2008. Efficient modularity optimization by multistep greedy algorithm and vertex mover refinement. Physical Review E, 77(4), p.046112.

[5] Jones E, Oliphant E, Peterson P, et al. SciPy: Open Source Scientific Tools for Python, 2001-, http://www.scipy.org/ [Online; accessed 2018-12-21].

[6] J. Leskovec, J. Kleinberg and C. Faloutsos. Graph Evolution: Densification and Shrinking Diameters. ACM Transactions on Knowledge Discovery from Data (ACM TKDD), 1(1), 2007.

[7] J. Leskovec, J. Kleinberg and C. Faloutsos. Graphs over Time: Densification Laws, Shrinking Diameters and Possible Explanations. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), 2005.

[8] M. Richardson and R. Agrawal and P. Domingos. Trust Management for the Semantic Web. ISWC, 2003.

[9] R. Albert, H. Jeong, A.-L. Barabasi. Diameter of the World-Wide Web. Nature, 1999.

[10] Raghavan, U.N., Albert, R. and Kumara, S., 2007. Near linear time algorithm to detect community structures in large-scale networks. Physical review E, 76(3), p.036106.

[11] Perozzi, B., Al-Rfou, R. and Skiena, S., 2014, August. Deepwalk: Online learning of social representations. In Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 701-710). ACM.

[12] Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S. and Dean, J., 2013. Distributed representations of words and phrases and their compositionality. In Advances in neural information processing systems (pp. 3111-3119).

[13] J. Leskovec, K. Lang, A. Dasgupta, M. Mahoney. Community Structure in Large Networks: Natural Cluster Sizes and the Absence of Large Well-Defined Clusters. Internet Mathematics 6(1) 29–123, 2009.

[14] Newman, M.E., 2003. The structure and function of complex networks. SIAM review, 45(2), Newman, Mark EJ. "The structure and function of complex networks." SIAM review 45.2 (2003): 167-256.

[15] Leskovec, Jure, and Andrej Krevl. "SNAP Datasets:Stanford Large Network Dataset Collection." (2015).