On Comparing and Enhancing Two Common Approaches to Network Community Detection

Niko Motschnig, Alexander Ramharter, Oliver Schweiger, Philipp Zabka, Klaus-Tycho Foerster
Faculty of Computer Science, *University of Vienna*, Vienna, Austria
{nmotschnig, aramharter, oschweiger, pzabka, ktfoerster}@cs.univie.ac.at

***Abstract*—In this work, we explore two common algorithms for community detection in networks, namely Agglomerative Hierarchical Clustering and the Louvain Method. We investigate their mechanics and compare their differences in terms of implementation and results of the clustering behavior on a standard dataset. We further propose some enhancements to these algorithms that show promising results in our evaluations, such as self-neighboring for Neighbor Matrix constructions and a deterministic and slightly faster version of the Louvain Method that favors fewer bigger clusters.**

***Index Terms*—Community Detection, Clustering, Social Networks, Network Algorithms, Partitioning, Network Analysis**

I. Introduction

A. Motivation

Community detection is a hot topic in network research, with a wide field of possible applications. As networks and graphs are able to offer abstract representations of many different domains, such as social dynamics, disease spread, customer behaviour, and many more, community detection can be employed to analyze and solve a whole range of different problems especially in the field of sociology. It is also applicable in other areas, such as, e.g., biology (protein-protein interaction networks), computer science (for instance finding groups of sites dealing with similar topics), and disease control, e.g., in cattle-trade networks [1].

Whereas there are a multitude of *new* proposals for community detection algorithms [2], we in this work follow a different research direction. Instead of designing the next iteration of some algorithmic variant, we take a step back and investigate the *technical implementation* of common algorithms, based on two selected examples: Agglomerative [2] and the Louvain Method [3], which will be presented in the following sections.

To this end, we illustrate and study the general workings of each of these algorithms, give details on their implementation, and discuss on how they as thus differ in terms of results and clustering behavior. In this investigation, our aim is on the one hand to provide a different perspective on these algorithms, namely, from a technical point of view, but on the other hand, our approach also allows us to provide ideas for new enhancements and optimizations of these seminal works, also outlining possible future research directions.

To this end, we discuss related work in §II, present the selected algorithms in §III, discuss results and adaptions to them in §IV, concluding and proposing next steps in §V.

B. Contributions

We propose, implement from scratch, and evaluate a range of adaptations to these well-known algorithms, namely:

1) **Self-neighboring:** for *agglomerative hierarchical clustering* (using the euclidean distance as the distance function to minimize), **self-neighboring** changes how the neighbor matrix is being constructed and manages to close "gaps" in the clustering results that would otherwise happen without **self-neighboring**.
2) A **deterministic** *Louvain method* that is slightly faster than its counterpart and creates fewer bigger clusters.
3) Lastly, in order to guarantee reproducibility and facilitate other researchers to build upon our work, we make our **source code publicly available** [4].
4) An extended version of this paper that examines two additional clustering algorithms can be found at [5].

II. Background

There are various methods to perform community detection or clustering in (social) networks. One of the most well-known overviews of this area is by Fortunato [2][1], who performed an extensive survey on the topic and compared the different algorithms in terms of quality (using normalized mutual information) and runtime complexity. We follow his work to select two common approaches to community detection, to investigate and compare their implementation details, and to propose and evaluate different enhancements.

A popular method is by means of *hierarchical clustering*. As there are various approaches in this area, we give a brief overview here: Abbas compares them to k-means, self-organization maps, and expectation maximization based techniques and categorizes the kind of data that fits best for each approach [6]. Others compare the performance of different linkage function for agglomerative techniques, such as Yu et al. [7] evaluating their technique based on single and complete link clustering. Charpentier [8] introduces a new way of doing hierarchical clustering using a modularity score called 'Paris' and Bonald et al. [9] present another new technique utilizing Node Pair Sampling. Bateni et al. [10] developed Affinity for Google Research, a hierarchical clustering approach that is able to handle trillions of nodes through MapReduce. Our work focuses on understanding and visualizing the differences of agglomerative clustering based on fundamental decisions

[1]Nearly 10 000 citations on Google Scholar.

such as distance and linkage functions, and moreover we will provide some intuition on how the clustering algorithm behaves based on the underlying choices and how these can affect the structure of the resulting clusters.

Another approach of interest is the *Louvain Method* by Blondel et al. [3], a greedy optimization algorithm for community detection. The Louvain Method uses modularity as its quality metric, showing positive results in comparison with other community detection algorithms on multiple data sets [3]. Lastly, regarding *modularity based community detection* algorithms, the generalized Louvain Method from De Meo et al. [11] is a more modern development in this area.

III. On Investigating Community Detection

In this section, we investigate the selected standard approaches to community detection, namely, Agglomerative Hierarchical Clustering and the Louvain Method (§III-B).

We also performed experiments and adaptations on other algorithms, the results of which can be found in our public repository [4]. However, due to space limitations, in this paper we restrict ourselves to present selected findings of our experiments with the Agglomerative and the Louvain Method.

To this end, for each approach, we first provide an overview, and then present our implementation details and test configurations, which motivate possible enhancements.

A. Agglomerative Hierarchical Clustering

1) Overview: Agglomerative Clustering describes a bottom-up approach: given a network, each node starts off as its own cluster. Step by step, the closest nodes/clusters are joined together. This step is repeated until there is only one cluster left, containing all nodes in the network. Closeness is defined by the used distance function between two nodes i and j. Additionally, there needs to be a method to measure the distance between clusters, for which we employ *linkage functions*. We will discuss how this affects the algorithm later in this section.

Other elements that need to be known prior to running the algorithm are the degree of each node *i*, denoted as k_i, and the Neighbor Matrix n, where each element $n_{i,j}$ describes the number of adjacent nodes which *i* and *j* share in common. Some distance functions also require to know the overall number of nodes in the network *N*.

Lastly, we use a horizontal separation line (HSL) that describes the strictness of the grouping. A dendrogram encodes the order in which the nodes and clusters are joined together in a tree-like structure, so that we can define the HSL to divide the final dendrogram into separate groups again. The decision on where to place the HSL has a major influence on the final results, as every vertical dendrogram line the HSL crosses will define its own cluster. The higher up the HSL is placed, the fewer the amount of final clusters.

2) Implementation and Test Configurations: Our implementations [4] are designed to be modular, such that core components can be swapped and replaced according to the use case. To this end, one can utilize different distance functions, linkage functions, and height of the HSL (among others), and plug in further functions.

We compare three linkage functions and analyzed their behavior, also in comparison to the other algorithms. The distance function used was the network-centric version of the Euclidean distance, defined as $d_{ij} = k_i + k_j - 2n_{ij}$, where n_{ij} describes the common neighbors to both node i and j. In the following, we will limit ourselves to the euclidean distance function only, since other distance function explored, such as cosine similarity or Pearson correlation coefficient did not result in a spatial grouping desired in community detection. However, these experiment and their results can be still be found in [4].

Concerning the linkage function, there are multiple ways of applying the distance function to determine the distance $d(A, B)$ between two clusters *A*, *B*. Most commonly, one would employ a simple aggregation function such as the minimum, maximum or average, as shown below:

1) *minimum* or *single-linkage*:
$$d(A, B) = min\{d(a, b) | a \in A, b \in B\}$$
2) *maximum* or *complete-linkage*:
$$d(A, B) = max\{d(a, b) | a \in A, b \in B\}$$
3) *average linkage*:
$$d(A, B) = \frac{1}{|A||B|} \sum_{a \in A} \sum_{b \in B} d(a, b)$$

Finally, the last important parameter to pass in is the HSL height. The lower the value, the higher up the HSL will be placed, thus reducing the number of final clusters. Our implementation allows to either pass in this value in an absolute or relative level.

This set of parameters and functions was the baseline for experimenting with our own implementation of the algorithm and inspecting their differences in terms of resulting clusters. We will discuss these results and further adaptations in §IV-A.

B. Louvain Method

1) Overview: The Louvain Method was published in 2008 by Blondel et al. [3]. It is a heuristic-based (actual modularity optimization is NP-complete [12]) greedy approach that aims to partition the graph into communities that optimize the modularity score.

Modularity measures the density of links inside clusters/communities in comparison to links between them, the formula for the computation of the modularity in weighted undirected graphs is [2, 3]: $Q = \frac{1}{2m} \sum_{ij} [A_{i,j} - \frac{k_i k_j}{2m}] * \delta(c_i, c_j)$, where:

- m is the sum of all edge weights in the network (if the weight is always 1 it is just the total number of edges).
- i and j are nodes, c_i and c_j denote the community assignments of the respective node.
- $A_{i,j}$ denotes the edge weight between nodes i and j.
- k_i and k_j stands for the sum of weights of incoming nodes to i and j respectively.

- $\delta(c_i, c_j)$ is the Kronecker-delta-function.

The Louvain method uses this formula indirectly as an objective function that gets optimized in a heuristic based greedy way. The algorithm consists of 2 distinct steps [3]:

1) For each node i in the graph calculate the change in modularity that happens when removing it from its current community c_i and adding it to every of its neighbouring communities. Then perform the swap that results in the biggest increase in modularity. If no positive change is observed the algorithm terminates.
2) Merge all nodes belonging to the same community together into one node where all internal connections (from nodes of community c_i to nodes of c_i) become a self-loop to the same node and all external/outgoing connections (from nodes of a community c_i to nodes of a different community c_j) are also merged into one edge. The weight of the merged edge is the sum of edge-weights from all edges merged together.

In order to calculate the change in modularity when node i gets assigned to the community c_j of one of its neighbours in Step 1), a more efficient formula is used that does not require to compute a sum over all nodes [3], by setting $\Delta Q = \left[\frac{\Sigma_{in}+2k_{i,in}}{2m} - \left(\frac{\Sigma_{tot}+2k_i}{2m}\right)^2\right] - \left[\frac{\Sigma_{in}}{2m} - \left(\frac{\Sigma_{tot}}{2m}\right)^2 - \left(\frac{k_i}{2m}\right)^2\right]$, where

- $k_{i,in}$ is the sum of weights of all edges from node "i" to target Community c_j.
- k_i stands for the sum of weights of all edges incoming to node i.
- Σ_{in} denotes sum of weights of internal nodes of target Community c_j (sum of weights of edges between nodes of Community c_j).
- Σ_{tot} equals the sum of weights of all edges that go to the target Community c_j (also includes Σ_{in}).

However we believe this *not to be the "complete" formula* to calculate ΔQ, since the original paper [3] states:

> "*A similar expression is used in order to evaluate the change of modularity when i is removed from its community. In practice, one therefore evaluates the change of modularity by removing i from its community and then by moving it into a neighbouring community.*"

Surprisingly, this similar expression to compute the change in modularity ΔQ for the initial removal is never mentioned in the original paper itself [3], as well as any other literature we came across during our investigations. We discuss whether or not the formula is incomplete and the implications in §IV-B2.

Moving on, the algorithm starts by assigning each node of the network to its own community and then iterates through all in Step 1. The way how it reduces the network in the merging step 2 and the efficient calculation of modularity change are the reasons why the runtime is more efficient than other similar approaches that aim to maximize modularity as well as other community detection methods overall. Its biggest advantage is that it delivers very good results in terms of modularity in a runtime-complexity of $O(n \cdot \log n)$ where n is the number of nodes in the network [13], which outperforms many other community-detection algorithms [3].

2) Planned Adaptations/Experiments: We investigated the following modifications of the Louvain algorithm:

- First, we used the formula that computes the modularity of the whole current partitioning for the network. This would be computationally inefficient, but it would further help to clarify whether the formula for the change in modularity, as discussed in §III-B1 is complete or not and it would be interesting to see the quality improvements (in terms of modularity) by using this approach.
- Then, we checked how to improve the community assignment of nodes and save iterations by chaining community assignments together. In its normal form, the algorithm assigns each node to the community of the best neighbours or the assignment stays the same. This step is done for every node. Hence, in this way, there might be some inefficient assignment where, e.g., node 1 is assigned to c_2, node 2 is assigned to c_3, and node 3 is assigned to c_4, and we propose to instantly merge nodes 1,2,3 together into the same community.
- Additionally, we also studied the effect of the node order (the order in which the nodes get iterated through in step 1 of the algorithm). The original paper [3] states that preliminary results on several test cases seem to indicate that the order of the nodes does not have a significant influence on the result quality in terms of overall modularity score. We therefore randomize the order of nodes in our implementation and want to see how the results differ by running the algorithm repeatedly on the same datasets.
- Lastly, we attempted to remove some of the greedy aspects of the algorithm by removing the merging step. This of course greatly increases the runtime complexity, but it would be interesting to see if one could make such adaptations to get a better result (in terms of modularity) and in essence trade runtime complexity for result-quality. This would make sense if one would e.g. only use the algorithm on a smaller network/graph where quality would be more important than execution speed.

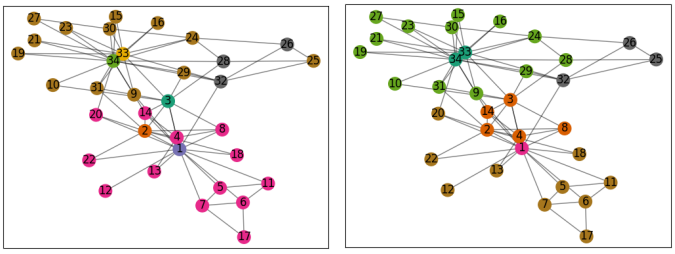
These adaptations and experiments are discussed in §IV-B.

IV. Results & Discussion

To evaluate, test, and compare the implemented algorithms we mainly used the popular *Karate Club dataset* from Zachary [14] with 77 edges and 34 nodes. It is considered "*a standard benchmark in community detection*" [2]: each node corresponds to a member of the club, where the edges represent mutual friendship. Additionally we used small random graphs for testing purposes.

We implemented both of these algorithms and their enhancements from scratch in Python 3.7. NetworkX 2.5 was used to generate the visualizations from the resulting clusterings.

Next we present our evaluations and discussion on both selected algorithms in Sections §IV-A and §IV-B.



((a)) No self-neighboring

((b)) Improved Cluster cohesion through the use of self-neighboring

Fig. 1: Agglomerative Clustering with Euclidean Distance, maximum-linkage and the relative *HSL-level=0.3* on the Karate Club dataset

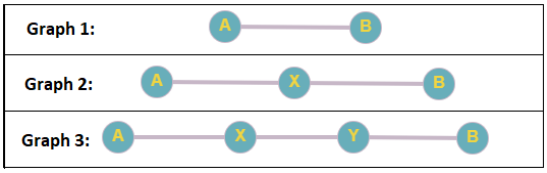


Fig. 2: Illustration of the intuition for self-neighboring

A. Agglomerative Hierarchical Clustering

1) Experiments & Results: Through a multitude of test runs, we identified interesting and unexpected behavior, which we discuss in this section.

Using the *Euclidean distance function*, together with *maximum linkage* and *HSL-level* $= 0.3$ results in a clustering of the Karate Club dataset as shown in Fig. 1 (a).

This figure also illustrates what agglomerative clustering, using Euclidean distance, identifies first and foremost: well-connected nodes. The same behaviour can still be observed for larger HSL levels. This result seems unexpected, as one might anticipate the use of Euclidean distance to result in more tightly grouped node clusters, as it is, for instance, outlined in [15]. To reveal some of these shortcomings, consider applying the Euclidean distance formula from §III-A2 to the nodes A and B from the three example graphs depicted in Fig. 2: for Graph 1 $d_{AB} = 2$, for Graph 2 $d_{AB} = 0$, and for Graph 3 $d_{AB} = 2$ again. Here, a node does not count itself as its own neighbor, effectively making A and B in Graph 1 further apart than in Graph 2.

2) Adaptation: Self-neighboring: As such we experimented on how the neighbor matrix n is being constructed, applying an idea we denote as *self-neighboring*. By having each node count itself as a neighbor, we can avoid the above situation and change the distances between A and B in Fig. 2 to Graph 1: $d_{AB} = 0$; Graph 2: $d_{AB} = 2$; Graph 3: $d_{AB} = 4$.

After testing this new approach, we observed that, for minimum-linkage, self-neighboring nearly always managed to produce a higher amount of total clusters than the conventional way: even with the same HSL levels, self-neighboring results in more diverse groups, though the overall effect of clustering central and well-connected node first still remained. For maximum-linkage, the exact opposite was observed: self-neighboring combined with maximum-linkage tends to produce less outliers, often forming more cohesive groups. When comparing Fig. 1 (a) and Fig. 1 (b), we see how this manifests: with node 2 and 3 being joined with their surrounding cluster, as well as node 25 joining its neighbors (which is especially interesting because, in Fig. 1 (a), node 25 shared a cluster with *none* of its neighbors) and node 33 and 34 now belonging to the same group as well, it can be observed how self-neighboring paired with maximum linkage increases cohesiveness and shrinks the overall amount of clusters, without needing to raise the HSL level. A similar effect also occurs for average linkage.

Based on these results, we propose *self-neighboring*, paired with maximum linkage, as a possible approach to computing neighbor matrices when trying to group clusters based on their Euclidean distance, especially if it is desired to reduce outliers.

B. Louvain Method

1) Reducing Greediness for Quality: In order to adapt the Louvain algorithm, we decrease its greedyness by removing the merging step (2) of the algorithm, which reduces the graph by creating one big node out of all nodes belonging to the same community. Additionally, we also used the total modularity formula for the whole graph (for reasons see Section §IV-B2), which on the other hand increased the runtime complexity. Herein, the general idea was to be able to trade higher runtime complexity for potentially higher quality results.

On our small test graphs, this led to the same "optimal" result that we achieved using the standard greedy version, as well as a standard (merging) version that used the total modularity formula. However, this result was achieved independently of the node order, and hence there is a qualitative benefit, as the greedy method had some node orderings that led to a slightly worse results.

In the bigger graphs, maybe surprisingly, the optimal result also remained the same. However, in general, the results tend to vary a lot more and there is a trend for more smaller communities. Moreover, there are also potential results that are worse than any of the results of the standard/*normal* version. We visualize these findings in the violin plot in Fig. 3. If one considers the violins of *normal* and *total*, the algorithm that uses the total modularity formula instead of the one calculating local change, and compares them to the violins of *noMerge* and *totalNoMerge* its clearly visible that the first 2 versions in general yield better results. A summary of the used methods is also depicted in Table I. Therefore, we can state that the merging step does not only contribute to a drastic reduction of runtime complexity, but also creates better partitionings.

2) Completeness of modularity change formula: At first our results with the standard version of the Louvain-algorithm were appropriate matching the results of the NetworkX Python library, as well as manual calculations using the total modularity formula on a small test graph.

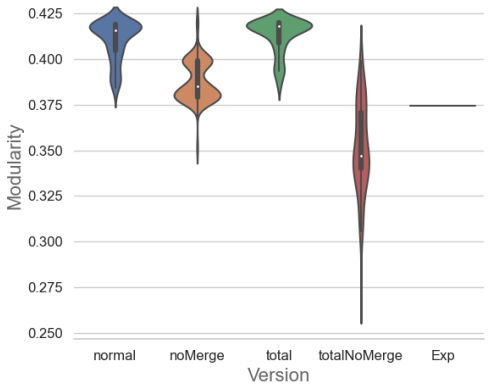
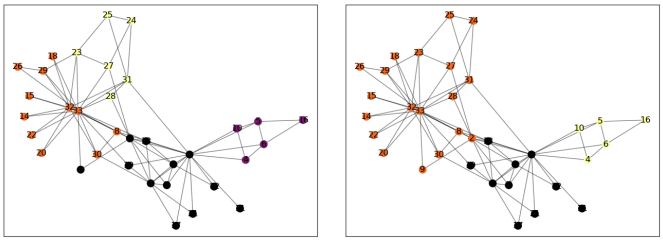


Fig. 3: Modularity scores for various Louvain-Algorithm versions (100 runs per version) on the Karate Club dataset [14]



((a)) Best Partitioning using our standard Version of the Louvain-Algorithm on the Karate Club dataset

((b)) Partitioning using our experimental Version of the Louvain-Algorithm on the Karate Club dataset

Fig. 4: Partitionings using (a) the standard Louvain-Version and (b) our experimental Louvain-Version

However, while implementing the less greedy version (see §IV-B1), we encountered the problem that the algorithm would not terminate and loop indefinitely swapping node community assignments.

The reason for that behaviour was that the formula, which as stated by the original authors [3] and briefly touched upon in §III-B1, is only partial and only captures the increase/decrease in modularity one gets when assigning the node to another community, but not how the removal of the node from its current community affects it. We obtained further validation by calculations using the "incomplete" formula on a small graph where:

- the Assignment of a node to the same community would still result in a modularity increase $\rightarrow$ this should not change modularity at all,
- the removal of a node from a bigger community and assignment to a new community consisting only of that one node resulted in a modularity change of "0" $\rightarrow$ this should decrease modularity.

Another evidence for the incompleteness of the formula is the violin plot in Fig. 3, where one can see that the version using the total modularity formula (*total*) led to better results more often than the *normal* one—if the formula would be complete, then the behaviour of the 2 versions should be exactly the same. We therefore believe that the formula from the original paper [3] and other sources is not complete as it is missing the effects on modularity that the removal of a node from its current community has.

Notwithstanding, the algorithm works well without this part of the formula (in the best case we reach the same modularity score on the Karate Club dataset [14] as the original authors [3]), since the cost of removal seems to be often insignificant when working with the standard version of the algorithm (especially after nodes get merged in Step 2).

Interestingly enough, as also depicted by the violin plot (Fig. 3, label *totalNoMerge*), the total and complete formula performs worse than the "incomplete" one if we omit the merging step as described in §IV-B1. The results vary the most and in 100 runs, the best possible result was not achieved. We cannot pinpoint a definitive explanation for this behavior—our assumption is that even though the "incomplete" formula for just the change calculation sometimes assigns nodes to initially non-optimal communities, this might lead to more favorable assignments later on.[2]

3) Faster merging experiment: We also adapted the algorithm in by how nodes are assigned to clusters in a way that assignments get merged/combined. If, e.g., node 1 gets assigned to c_2, node 2 gets assigned to c_3, and node 3 gets assigned to c_4, they are put all in the same cluster/community as their assignments are merged.

This led to a deterministic algorithm version as visible in Fig. 3 with the label *Exp*. Since the assignments are only done at the end of each iteration, they no longer depend on the order in which one iterates through the nodes. The result had a worse modularity score than all or most of the partitionings that were determined by the standard as well as the other version. In general, the results also had fewer clusters than the near optimal partitioning in terms of modularity. However, when considering the resulting community assignments, as in Fig. 4 (b), the results are still good enough.

This, combined with a slightly faster run-time (see Table I), and the advantage of deterministic results, might make it attractive in certain situations.

Our implementation is available at [4].

V. Conclusion and Outlook

In this paper, we explored and implemented common community detection algorithms from the ground up, in order to perform a technical investigation w.r.t. their algorithmic behaviour *on paper* versus *in code*, and additionally proposed and evaluated a range of enhancements based on our findings.

We investigated the partitioning behavior of Agglomerative Clustering and how it is affected by the choice of distance function, linkage function, and HSL level. Based on our

[2]Similar to evolutionary algorithms, where sometimes one needs to go into a seemingly unfavorable direction to escape local maxima and move towards the global maximum.

TABLE I: Comparison of Louvain versions (100 runs, Karate Club dataset, Intel i7-3770k CPU@3.5GHz, 16GB RAM)

Label	Description	Max. Score	Min Score	Avg. runtime (ms)
normal	Algorithm as described in the original paper [3].	0.41979	0.38108	3.899
total	Algorithm as described in the original paper [3] that uses the formula for the total modularity of the whole graph to calculate change instead of the formula that directly calculates the change.	0.41979	0.3792	440.799
noMerge	Like *"normal"* but the merging step (2.) of the algorithm (see §III-B1 is omitted and the algorithm always works with the full set of initial nodes.	0.41979	0.35396	57.314
total NoMerge	Like *"total"* but the merging step (2.) of the algorithm (see §III-B1 is omitted and the algorithm always works with the full set of initial nodes.	0.39981	0.24499	1966.91
Exp	Experimental version described in §IV-B3, creates less bigger clusters and is deterministic.	0.37443	0.37443	3.349

results, we proposed *self-neighboring* as an alternative way to construct the neighbor matrix, which manages to improve cohesion by *closing gaps* between clusters.

Regarding the Louvain Algorithm, we discovered that the full formula for modularity change was not disclosed in the original paper. We as thus investigated and analyzed the impact of using the partial formula by comparing the results generated by its use with the ones created by the total modularity formula. Additionally, we also experimented with the merging step of the algorithm as we tried to omit it and see how it affects the results. Our main results in this context was that the merging step of the algorithm not only *reduces the runtime* of the algorithm but also helps to create *better* results in terms of *modularity*, but that at the same time, the incomplete formula still works well – except when omitting the merging step, but can nonetheless cause an indefinite loop in certain situations. We moreover proposed an adaptation, where we changed the method how communities are assigned, to the Louvain Method, that is faster, deterministic, and creates bigger clusters compared to the normal version.

A. Future Work

One major area to focus on next would be the exploration of additional data sets. The techniques presented were examined to work on smaller examples and well-known data sets such as the Karate Club one, but it still remains open how these adaptations would fare in the context of thousands to millions of nodes. Additionally, with the agglomerative clustering approach, where one can arbitrarily exchange the distance function, there is good potential for further exploration and investigation by leveraging more intricate distance functions.

Another possible direction for future work would be to investigate and implement more clustering algorithms with other underlying paradigms (such as, e.g., using random walks as presented by Pons et al. [16]).

Regarding the Louvain method, a further step could be to derive the complete formula to calculate the change (as discussed in §IV-B2) and repeat the experiments. Furthermore, there have also been orthogonal adaptations to the Louvain method, such as, e.g., by De Meo et al. [11], who were able to slightly improve the results of the original method using a novel measure of edge centrality based on *k-paths*.

B. Reproducibility

In order to guarantee reproducibility and facilitate other researchers to build upon our work, we make our source code and artifacts publicly available [4].

References

[1] L. Brzoska, M. Fischer, and H. H. Lentz, "Hierarchical structures in livestock trade networks—a stochastic block model of the german cattle trade network," *Frontiers in veterinary science*, vol. 7, p. 281, 2020.

[2] S. Fortunato, "Community detection in graphs," *Physics reports*, vol. 486, no. 3-5, pp. 75–174, 2010.

[3] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of statistical mechanics: theory and experiment*, vol. 2008, no. 10, p. P10008, 2008.

[4] N. Motschnig, A. Ramharter, O. Schweiger, and P. Zabka, "Network community detection," https://gitlab.cs.univie.ac.at/ct-papers/network-community-dection/, 2021.

[5] N. Motschnig, A. Ramharter, O. Schweiger, P. Zabka, and K.-T. Foerster, "On comparing and enhancing common approaches to network community detection," https://arxiv.org/abs/2108.13482, 2021.

[6] O. Abu Abbas, "Comparisons between data clustering algorithms," *Int. Arab J. Inf. Technol.*, vol. 5, pp. 320–325, 07 2008.

[7] S. Yu, K. Yu, and V. Tresp, "Soft clustering on graphs," 01 2005.

[8] B. Charpentier, "Multi-scale clustering in graphs using modularity," Master's thesis, KTH Royal Institute of Technology, Brinellvägen 8, 114 28 Stockholm, Sweden, 2019.

[9] T. Bonald, B. Charpentier, A. Galland, and A. Hollocou, "Hierarchical graph clustering using node pair sampling," *CoRR*, vol. abs/1806.01664, 2018.

[10] M. Bateni, S. Behnezhad, M. Derakhshan, M. Hajiaghayi, R. Kiveris, S. Lattanzi, and V. Mirrokni, "Affinity clustering: Hierarchical clustering at scale," in *NIPS 2017*, 2017, pp. 6867–6877.

[11] P. De Meo, E. Ferrara, G. Fiumara, and A. Provetti, "Generalized louvain method for community detection in large networks," in *2011 11th international conference on intelligent systems design and applications*. IEEE, 2011, pp. 88–93.

[12] U. Brandes, D. Delling, M. Gaertler, R. Görke, M. Hoefer, Z. Nikoloski, and D. Wagner, "Maximizing modularity is hard," *arXiv preprint physics/0608255*, 2006.

[13] A. Lancichinetti and S. Fortunato, "Community detection algorithms: a comparative analysis," *Physical review E*, vol. 80, no. 5, p. 056117, 2009.

[14] W. W. Zachary, "An information flow model for conflict and fission in small groups," *Journal of anthropological research*, vol. 33, no. 4, pp. 452–473, 1977.

[15] D. Beers and R. Campbell, "Community detection with hierarchical clustering algorithms."

[16] P. Pons and M. Latapy, "Computing communities in large networks using random walks," in *International symposium on computer and information sciences*. Springer, 2005, pp. 284–293.