Karlsruhe Institute of Technology

Parameter-free Outlier-aware Clustering on Attributed Graphs

Bachelor Thesis of

Uwe L. Korn

At the Department of Computer Science
Institute for Program Structures
and Data Organization (IPD)

Reviewer:	Prof. Dr.-Ing. Klemens Böhm
Second reviewer:	Prof. Dr. rer. nat. Dorothea Wagner
Advisors:	Dr. rer. nat. Emmanuel Müller
	Dipl.-Inform. Patricia I. Sánchez
	Dipl.-Inform./Dipl.-Math. Tanja Hartmann
	Dipl.-Inform. Andrea Kappes

Duration: 16th December 2012 – 3rd April 2013

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

Karlsruhe, 3rd April 2013

. . . .

(**Uwe L. Korn**)

Contents

List of Figures

1. Introduction

Outliers represent observations that differ fundamentally from all other observations and are an indicator for human errors, mechanical defects, suspicious behaviour or just natural deviation. There is no general definition what is considered an outlier but the most commonly accepted definition by Hawkins [Haw80] states that "an outlier is an observation that deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism". Outliers can occur in a global perspective, i.e. they are a an observation that deviates significantly from all other objects, or in a local setting. Outliers in a local scope, called community outliers, would be considered regular in the entire graph but deviate from their neighbouring objects in the graph. Viewed from a local perspective, i.e. only considering the vector and its larger neighbourhood, not all attributes in the feature vector contain relevant information in this context. Some of the attributes just follow a random behaviour.

Objects that are not considered as an outlier can be categorised into clusters, groups that share high similarity and are dissimilar to other objects around them. Cluster should be densely connected inside the graph. As well as outliers, cluster members only are similar to each other in a subset of all attributes contained in the feature vector. The clusters of a graph give an insight which patterns are common among the stored observations. Members of a cluster can often be treated very similar on using the extracted information.

For the detection of outliers and the grouping of all objects into clusters there are various applications. Outliers could be removed in a preprocessing as they might have negative effects on the output of other algorithms. Though outliers are often removed from data sets, they can have valuable information that should be explicitly analysed. Objects of a cluster could be treated as nearly equal objects in some processes as they share similar information. In some other algorithms members of a clusters cannot be treated equal but clusters can be processed in parallel to exploit current multiprocessing technology. Some of the real world applications of these two information categories include the fraud detection of stolen credit cards [EOZPP11] and the protein - protein interaction as a network with folding properties as the attributes in pharmaceutical research. In online shops the outlier information could be used to find suspicious products which affect the reputation of the seller. Clusters in an online shop may used to create bundles or coupons that attract customers to buy a related product or could be used as the basis of a recommender system.

A famous example for data that is represented as an attributed graph are online social networks. In the context of a social network, an example cluster is made up of the members of a youth team of a small football club. These members are connected through the friendship relation as they all know each other. They have common attribute values in their feature vector as they are from nearly the same age and live in the same county. As part of the friendship relation their coach knows everybody of the team too and is likely to have connected to everyone on the social network. Though his good integration in this community environment, he does not fit into this group as he is significantly older than everybody else. He can be considered as a community outlier. In contrast from a global perspective the coach would be a normal person at a common age and likes football as a lot other people do.

Existing traditional algorithms could identify some of these patterns in the data but would find imprecise results as they do not take the correlation of graph structure and feature vectors into account. Information like local outliers cannot be found by them as they only occur in the combination of both data sources. If the algorithm only looks at the attributes the objects in the found clusters would share high similarity but sparse connectivity as the structure was not respected. In contrast, in a clustering generated by an algorithm that only takes the structure into account, the objects are densely connected but do not share similar attribute values in their feature vector. Another drawback of several existing clustering algorithms is that are not outlier-aware, i.e. the clustering process suffers from outlier objects that distract the found clusters.

In the recent years, algorithms that consider the combination of graph structure and feature vectors have been developed. Though some of these algorithms can find some of the local outliers, they are likely to suffer from the Curse of Dimensionality. As another disadvantage, most of these algorithms require user-given parameters that are hard to choose by a non-expert user and cannot be easily guessed. A further drawback of some of these algorithms is that they only look at the full set of attributes, i.e. if some patterns could only be recognised using a subset of these attributes, these algorithms fail to detect them. Overall, these algorithms do not scale efficiently with the number of attributes or the number of vertices. Some of the algorithms have an exponential runtime making them unusable for large input data.

In this work, we present a quality measure for rating clusterings on attributed graphs. In addition a scoring function that ranks objects by their outlierness w.r.t. the clustering is introduced. For the calculation and optimisation of these functions, an parameter-free algorithm scaffold with efficient instantiations is developed.

At the beginning of this work in Chapter 2 an overview of existing technologies and related work that has influenced the proposed solution will be given. To measure the quality of clustering in Chapter 3.2 a new measure for clusterings on attributed graphs is introduced. This measure respects the fact that only a subset of all attributes have characteristic values for a cluster. The quality is rated without any user-given parameters with the aspect that attributes and structure should have a balanced influence on the found clusters. To rank the clustered object by the possibility that they are an outlier, in Chapter 3.3 a scoring function is proposed which uses the information from the clustering process to detect community outliers. To overcome the three mentioned drawbacks of the existing algorithms, in Chapter 4 an algorithm scaffold which could be instantiated with the three methods described in the Chapters 4.1.1, 4.1.2 and 4.1.3, is introduced. All these instantiations run without any input than the attributed graph. Through the evaluation in Chapter 5 the quality and efficiency of these algorithms will be shown.

2. Related Work

The amount of data we collect and analyse today by far exceeds what can be manually reviewed by humans as it was done in earlier times. As an example, the 1880 U.S. Census, which was entirely manually processed, took eight years to be analysed and summed up [Hol94]. In contrast, the 1890 U.S. Census took only about one year to sum up, despite polling more information from an increased population. This was made possible by the introduction of electrical tabulating machines [Hol94]. The population of the United States of America has enormously increased since, and the amount of information captured per person has risen with it. Most of the information is available within a single year [Cen13] through the advance of algorithms and technology.

As the quantity of data and its structure constantly increases and changes, new techniques and algorithms need to be developed to cope with the upcoming problems. The development of these new techniques is part of Data Mining. There are various ways to structure data but as graph structured and relational data are the dominant formats, a lot of research is done in this area. Recently the combination of both types has come up as a new research topic. Data represented as a graph, consists of vertices (named data point in the context of attribute-related mining) which are connected through edges. Each edge has been given a numerical weight to quantify the importance of its connectivity. Relational data of the kind considered here consists of a data point to which a fixed number of attributes, either numerical or categorical, is assigned. This work addresses the combination where the data points (vertices) are connected through edges and annotated with feature vectors.

Clustering is the process of grouping the input data so that for each group, the data points within are similar to each other and dissimilar to all other points. The definition of similarity is w.r.t. a given measure. Changing the similarity measure will change the clusters even if the same input data is used. Some data points may have a rather large effect on the objective function depending on how they are clustered. These objects are part of the set of data points considered as outliers. Though most of the outliers will have significant impact on the objective function, not all do. There is no general definition of what is considered an outlier but the most commonly accepted definition by Hawkins [Haw80] states that “an outlier is an observation that deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism”.

In the field of clustering and outlier mining of either graph structured or relational data, years of research has already been performed. A short overview of important approaches which include work that has lead to the proposed solution in this work is outlined in the next subchapter 2.1. The following subchapter 2.2 considers methods that combine the graph structure and the relational data to get better results. For each algorithm it is explained why it does not solve the given problem satisfactorily.

2.1. Traditional Mining Methods

Clustering on relational data

As mentioned above, the focus of this work lies on graphs and relational data. Traditionally most algorithms look at either of them. The first simple well-known approach to relational data clustering is the K-Means [M$^+$67] algorithm. Though this approach already works well for several problems its suffers from the Curse of Dimensionality [Bel57]. The increasing number of dimensions does not only involve a growth in runtime, Beyer et al. [BGRS99] has shown that the minimal and maximal distance between two objects get similar the more dimensions a dataset has. To overcome this problem there are already some solutions like reducing the overall number of dimensions by removing redundancy (e.g. PCA [Pea01]) or by selecting only subspaces that seem to have relevant information (e.g. ENCLUS [CFZ99] or HiCS [KMB12]). Though this reduces the problem of high dimensional input, there are still some drawbacks like an exponential increase in runtime and disrespect of graph structure.

One of the first algorithms that clustered data on only a subset of all dimension was CLIQUE [AGGR98]. This grid-based approach tries to find cluster in higher dimensions using Apriori techniques based on dense regions in lower dimensional subspaces. One of the drawbacks of this algorithm is that results depend on the positioning of the grid and how fine-grained the grid and the threshold is adjusted. A more efficient algorithm is ProClus [AWY$^+$99] which uses the K-Medoid method to cluster objects and then iteratively refines the relevant dimensions for each cluster. Though leading to good results with short runtimes, ProClus requires two parameters k, l set by the user. The average cluster size k which could be guessed by the user as in most cases one has a basic assumption and the average number of dimensions l. As the number of dimensions may vary between clusters and in total exceed 100 in current data sets, it is not easily guessed by a user without expertise. ProClus monitors the quality of its clustering during the cluster process but the objective functions uses the above two parameters and cannot be applied to graphs.

One algorithm that does not need user-given parameters as input is HARP [YCN04]. It still provides one parameter that could be used to limit the runtime but it runs without any parametrisation by the user. HARP uses a bottom-up approach by combining clusters starting from size-one communities. On deciding which clusters are combined, HARP computes the relevance of each cluster and the combined relevance of the two to-be-merged clusters which is

$$R_{k,l} = 1 - \frac{\sigma_{k,l}^2}{\sigma_k^2} \tag{2.1}$$

where $R_{k,l}$ is the relevance of cluster l w.r.t dimension k, σ_k^2 the global variance of dimension k and $\sigma_{k,l}^2$ the variance along dimension k w.r.t. the cluster l. Some features of HARP with focus on the relevance are used in the later work as they introduce a quality measurement on clustering w.r.t. to different subspaces for each cluster without user-given parameters.

Outlier detection in relational data

As not all data points fit into the clusters found by the above algorithms we need to detect which objects should be considered as outliers. There are various ways to rank objects by their outlierness. A basic approach described by Leroy and Rousseeuw [LR87] assumes that the data was generated using a normal distribution and labels all objects outside of 1% significance as outliers. This technique has the drawback that it needs a basic assumption about the underlying distribution. Without using an underlying distribution Knox and Ng [KN98] propose an approach that defines a data point as an outlier if a certain percentage of all objects have a distance larger than a given value to the data point. Though there are some distance-based methods that do not need the distance as an input, all distance-based methods suffer from the usage of absolute distances. When using absolute distances, outliers that are relatively far away from a dense community but with a distance less than objects in more scattered regions are not detected. The group of density-based outlier detection methods with its most known representative LOF [BKNS00] respect the locality of each data point and measure its outlierness on the distances to its neighbours. Though LOF respects the local density, it is not used in this work as it only considers attributes and ignores the graph structure. In contrast to this the neighbourhood of an object in this work is based on its neighbouring communities and the size of its own community.

Graph Clustering

In addition to relational data a large number of clustering and outlier mining algorithms exist for graph structured data. In the field of graph structured data *community* is often used as an interchangeable term for *cluster*. This is used as a synonym throughout this work. Graph clustering started out of the problem of graph partitioning where the problem is to split a graph into a fixed number of subgraphs so that the sum of cut edges is minimal and all subgraphs are nearly equal in size. An early but still used algorithm is [KL70] which splits the graph into even sized subgraphs and then tries to optimise the partitioning by using local optimisation. One remarkable property of the algorithm is that it makes steps that decrease the overall score but lead out of local optima for a better final result. Despite the benefit of depth research in this field, these algorithms are fixed to the user-given amount of clusters and do not meet the requirements to solve the given problem. They do not work well if one needs to detect a unknown number of clusters of hardly balanced size. Although graph-based data varies significantly from relational data there are some approaches like Spectral Clustering that transforms graph-based data so that it can be used with attribute oriented clustering methods. Spectral clustering uses the Laplacian matrix of a graph which has several mathematical characteristics such as "from the eigenvector of the second smallest eigenvalue of the Laplacian matrix it was possible to obtain a bipartition of the graph with very low cut size" [Fie73] which enable the use of algorithms like K-Means on graph-based data.

In recent years the measure Modularity, which was introduced by Newman and Girvan [New04], was used as the basic quality measure in several algorithms. Blondel et al. [BGLL08] proposed a algorithm that optimises Modularity by local movements of vertices and iterative coarsening of the graph. This method and some other techniques mentioned, like merging *Modularity* clusters from a global perspective, in [NR09] are used in this work as a scaffold for the proposed algorithm elements.

Another approach to Graph Clustering is the SCAN [XYFS07] algorithm which clusters vertices based on their structural similarity. It introduces the notion of hubs, vertices that are important connections between two or more clusters, and respects outliers. Though SCAN introduces outlier mining to graph clustering, general outlier detection on graph was already part of other work mainly in the area of intrusion detection. Noble and Cook

[NC03] present an approach to identify anomalous subgraphs by comparing subgraphs with the most used patterns in the whole graph. This approach was optimised by Eberle and Holder [EH07] and Davis et al. [DLMR11]. An algorithm based on their ideas is introduced which supports numerical labels in addition to the structural properties. The drawback of this approach is that it is assumed that the graph is created following some globally famous patterns which makes these algorithms unsuitable to detect outliers that only vary from their local neighbourhood.

A simple approach for clustering the combination of graph structure and attribute data could done using existing techniques. It would be possible to use a technique from either category e.g. mining only the graph structure to get information from the given data. Using only one technique would only lead to good results if attributes and graph structure are highly correlated, e.g. the connectivity of the edges could be computed out if the attributes of a vertex. Another approach would be to run two existing concepts iteratively using the input data of the previous one. These methods have the drawback that only respecting one paradigm of data will lead to a loss of quality in the other. While attribute-only clustering generates clusters that distinguish themselves very good in the attribute space, the cluster may have sparse connectivity in the graph. In contrast graph clustering algorithms may create clusters with dense intraconnections and sparse interconnections but the attribute values of objects in these clusters could be very dissimilar.

2.2. Attribute-aware Graph Mining

Today, graphs such as social networks are made up of the combination of graph structure and relational data. A data point has various numerical or categorical attributes and is connected to other data points through (weighted) edges. In a social network a person may have several attributes like age and education and is connected by the friendship relation to other people. In the following this kind of structured data is called *an attributed graph.*

A technique to use existing algorithms is to transform the input data to either only graph-structure or relational data. One such approach is described by Zhou, Cheng and Yu [ZCY09] that transforms the categorizational attributes of a vertex into additional edges that augment the graph. After the transformation, the graph is clustered using random walks. Though this would be a possibility to reuse the pool of existing algorithms the mentioned transformation does not respect the locality of vertices cannot work in local subspaces of the attribute set.

As a first approach CoPaM [MCRE09] is looking for dense patterns occurring in the combination of graph and relational data. Although it takes care of subspaces in the attributes, the algorithm needs four user-supplied parameters which are hard to choose by a non-expert user. In addition to the choice of the parameters, as another disadvantage CoPaM does not always cluster all objects but leaves some objects uncategorised, i.e. not marking them as a cluster-member or an outlier. GAMER searches the graph for quasi-cliques in multiple subspaces. Using this technique the algorithm reveals less redundant and very dense clusters but leaving a lot of objects undescribed if a graph contains sparse areas. In DB-CSC [GBS11] some extension were proposed so that not only quasi-cliques were clustered but less dense patterns too. To get a less redundant number of clusters, it proposes the usage of density-based clustering with graph structure. Though less dense areas are clustered now, objects may still not be categorised into any cluster. Objects without a cluster may not always be an outlier and some objects that are clustered but occur in a small number of clusters could be an outlier. Though these algorithms provide a solution to the clustering problem, they need user-specified parameters and are not efficient.

For outlier search in the combination of relational data and graph structure, Gao et al. [GLF$^+$10] proposed CODA. This approach searches for outliers in graphs given a fixed number of communities. CODA itself was only designed for one dimension but can be easily extended to a larger number of attributes. Due to its design CODA does not take care of subspaces which occur in the context of a community. As a result always the full attribute is considered so that outliers hidden in subspaces may not be found and the general analysis suffers from the curse of dimensionality and does not scale with the number of dimensions.

3. Quality Measures

At the beginning of this chapter basic notions which are used for the proposed measures are introduced. Afterwards the proposed measures itself are defined. The suggested algorithm introduces the concept of projected clustering to the field of attributed graphs. Each vertex in the graph should be assigned to a cluster and depending on this cluster the attributes that define this vertex, i.e. are characteristic for the cluster, are determined. As some objects vary significantly from all other data points an outlier ranking of all points is generated, so that those objects can be evaluated specifically.

It is assumed that a graph consists of clusters that are highlighted through the structure and the distribution of a subset of the attributes in its local environment. In these subsets of attributes that are characteristic for a cluster, all vertices show similar values, i.e. belong to a certain, small range. All other attributes may show a random distribution without any indication of structure. The vertices of a cluster are highly connected through edges so that the cluster have a more dense connectivity between its vertices than to vertices lying outside of the cluster. For the selection of the right clusters, a quality measure that takes the combination of graph structure and relational data into account (so that neither one should outweigh the other) is needed. As an example this measure should respect that cluster that consider both types of data are neither the direct projection of a graph or relational cluster. They may overlap or are a subset of these traditional clusters. In addition to the cluster quality measure, a ranking function which is able to find the outliers in the attributed graph needs to be developed. As already specified in the introduction, these measures should not require any user-given parameters.

Subchapter 3.1 introduces basic notions which are used in the second part to propose a quality measure for rating the clustering of a whole graph. The last subchapter introduces a scoring function to detect objects that do not fit to their local context, i.e. can be considered as outliers.

3.1. Basic Notions

The proposed approach only takes a graph combined with attributes as input without any parametrisation by the user. In formal terms the input consists of a graph $G = (V, E, w)$ where V is the set of vertices and $E \subseteq \{\{u, v\} \mid u \in V, v \in V\}$ the set of undirected edges connecting these vertices. For each edge a positive weight $w(e) = w(\{u, v\}) = w(u, v) = w(v, u)$ is defined. $n = |V|$ and $m = \sum_{e \in E} w(e)$ respectively are the sums of these two sets.

As an alternative notation the symmetric adjacency matrix A with $A_{ij} = w(i, j)$ is used in the context of *Modularity*. If there is no edge between two vertices they are unconnected but one could assume an imaginary connection via an edge of weight 0. To handle graphs without edge weight, a general default of $w(e) = 1$ is used. The total weighted degree of a vertex i is a property which is often used and abbreviated by $k_i = \sum_{u \in V} w(u, i)$.

In addition to the graph properties each vertex has d numerical attributes. As an alternative to the common names u, v for vertices we use x_i as a name for one vertex as it is a data point in the world of attribute-only clustering. $x_{i,j}$ is the short form for using the j-th attribute of the vertex x_i. The values of these attributes may be from an arbitrary numerical range.

The algorithm will output for each vertex x_i its cluster c_i. For each cluster the importance of each attribute will ranked from 0 (no importance) to 1 (all members of this cluster have the same value). In addition to the clustering, a score for each vertex is calculated which reflects how well the point fits to the patterns of its neighbourhood. The higher this score, the more an object should be considered as an outlier. The outlier score ranges from 0 (fits perfectly to its neighbourhood) to infinity, i.e. there is no predefined global limit.

Attribute based notions

The proposed quality measure and the algorithmic approaches make use of several already existing measures and properties that provide the methods to incrementally compute the proposed scoring functions. To indicate how scattered a set of data points is, the most well-known measure, the variance $\sigma^2 = S/N$ is used. It is easily computed on a set of data points using the so-called *two-pass formula* for a dataset of N objects with a mean of $\overline{x}$:

$$\frac{S}{N} = \frac{1}{N} \sum_{i=1}^{N} (x_i - \overline{x})^2$$

This version has the drawback that if you add a data point, a scan over all data points is needed as the mean changed. Chan, Golub and LeVeque [CGL83] outline various methods that are able to overcome this drawback. Some of those suggestions are used later in chapter 4 as different algorithmic approaches are discussed which incrementally compute the variance. Another property of variance is that it is sensible to outliers, i.e. if a object is added to a dataset that varies significantly from all others, the variance increases dramatically. Although this is often considered as a disadvantage, this feature can be used to detect if an outlier was added to a set of points. Alternatively if the variance is kept small, this will prohibit the addition of outliers to data sets.

Yip, Cheung and Ng [YCN04] introduce Relevance as a measure for the importance of an attribute for a cluster. Given the global variance σ_k^2 of a dimension k and the local variance of dimension k in cluster l, the relevance R_{kl} of this dimension w.r.t. to the cluster is:

$$R_{kl} = 1 - \frac{\sigma_{kl}^2}{\sigma_k^2} \tag{3.1}$$

A relevance of 1 indicates that all data points in this cluster have the same value. If the data points of the local cluster have the same scattered distribution as the whole data set in a specific dimension, the relevance for this dimension is 0 w.r.t. the cluster. Attributes having a relevance below 0 in a cluster are more scattered than the attribute is from a global view, i.e. the attribute is not characteristic for this cluster. Attributes which have a global variance of 0 are not taken into account, as they do not provide any information on how the data should be grouped into clusters. Having all the same attribute value, using attribute-only clustering there would be only one cluster w.r.t. this dimension. Using the relevance as a measure overcomes the problem of different ranges in different dimensions as the relevance is only dependent on the ratio between the local and the global variance of a dimension. For the computation of the relevance of an attribute w.r.t. a cluster only the local variance needs to be calculated. This property enables the incremental and numerically stable computation of the relevance using the already developed methods for the variance.

Structural based notions

As a basis to rank the quality of the structural part of a clustering of a graph, the measure *Modularity* [New04] is used. This well-known measure is used because it provides a parameter-free rating of cluster quality w.r.t. the structure. The *Modularity* score per vertex is the fraction of edges that is connected to vertices of the same cluster minus the expected fraction of edges if they were distributed according to a certain random process. Overall the Modularity score using the adjacency matrix A, the weighted degree k_i of vertex i is given by:

$$\text{Mod} = \sum_{i,j} \left(\frac{A_{ij}}{2m} - \frac{k_i * k_j}{4m^2} \right) \delta(c_i, c_j) \tag{3.2}$$

Given a clustering and its modularity score, after simple transformations like merging two clusters or moving a vertex from one cluster to another, the *Modularity* score can be updated in constant time. These transformation require some preprocessing like that the weight of the edges between two clusters has to be maintained and the total weighted degree of each vertex needs to be computed upfront.

Though the Modularity score can be updated in a fast way after transformations of the graph clustering, it was proven by Brandes et. al [BDG+08] that optimising Modularity is NP-hard. Due to this fact the above mentioned properties are utilised to in various algorithms which cluster using Modularity.

3.2. Cluster Quality

In this chapter we want to propose *Attributed Modularity* as a measure for determining the quality of a graph clustering w.r.t. the vertex attributes. The structure of this measure is motivated and an outline is given how this measure overcomes problems the problems of existing, traditional measures. Basic properties of the proposed measure are explained using examples that highlight these features.

One possible solution to the given problem is to either use methods that take care of attributes or of the graph structure. Using only one information may lead to the problem that the clusters only fit to this information, e.g. attribute-only clusters would not be densely connected and structure only clusters may cover a subset of objects that are not distinguishable from a random selection.

We propose to combine these measure to a single function called *Attributed Modularity.* To select the dimensions that are characteristic a cluster, the Relevance score of HARP [YCN04] is used. This is integrated with Modularity [New04] as the measurement for the structural quality. After evaluating different measures that either respect graph structure or relational data, the combination of both works well for attributed graph clustering. Other measures had the drawback, that they needed user-given parameters as an input or that they could not be split up to be combined with another measure that evaluates other types of data. In the proposed measured, for each vertex the fraction of edge weight connected to its own cluster is compared to the edge weight in a random clustering. This score is multiplied by the sum of all positive relevance scores w.r.t. to its community.

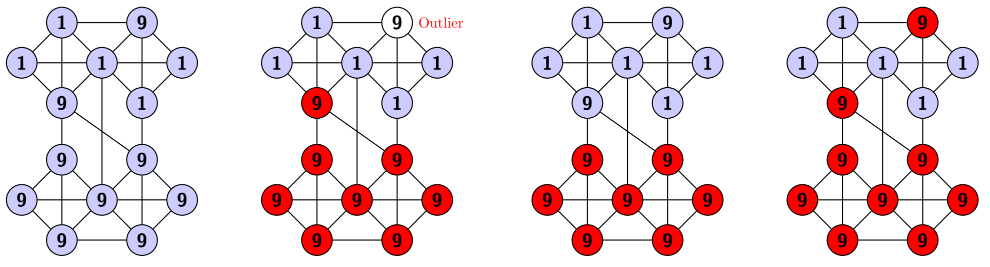
Definition 1 *Given the community $c(i)$ of a vertex i, the local variance $\sigma^2_{k,c(i)}$ w.r.t. the cluster $c(i)$, the set of vertices V, dimensions D, clusters C, the total weighted degree k_i of a vertex i, the global variance σ^2_k of dimension k and the adjacency matrix A,* Attributed Modularity *is calculated via:*

$$\begin{aligned} \mathrm{AM} &= \sum_{i\in V}\left(\frac{1}{d}\sum_{k\in D}\max\left(1-\frac{\sigma^2_{k_{c(i)}}}{\sigma^2_k},0\right)\sum_{j\in V}\left(\left(\frac{A_{ij}}{2m}-\frac{k_i*k_j}{4m^2}\right)\delta(c(i),c(j))\right)\right) \\ &= \sum_{c\in C}\left(\left(\sum_{k\in D}\max\left(1-\frac{\sigma^2_{k_c}}{\sigma^2_k},0\right)\right)\left(\sum_{u\in c,v\in c}\left(\frac{A_{ij}}{2m}-\frac{k_i*k_j}{4m^2}\right)\right)\right) \end{aligned} \tag{3.3}$$

The range of *Attributed Modularity* scores is the same as the range of *Modularity* [-1/2, 1). The part of equation 3.4 regarding the relevance of the attributes in a cluster has [0,1] as boundaries so that it would only lower the absolute *Modularity* per cluster but does not extend its range.

$$0 \leq \frac{1}{d}\sum_{k\in D}\max\left(1-\frac{\sigma^2_{k_{c(i)}}}{\sigma^2_k},0\right) \leq 1 \tag{3.4}$$

Though *Attributed Modularity* scores have a fixed range, the values of one graph are not comparable to the values of another graph as they have other global variances and total edge weights. It can only be used to compare clusterings of one graph where higher scores indicate a better clustering.



(a) Unclustered (b) Wanted (c) Modularity (d) Attribute-only

Figure 3.1.: Different clusterings of a graph depending on the paradigm used

To show some properties of *Attributed Modularity* in Figure 3.1 a graph and some possible clusterings are given. In the next section, it is shown that depending on how good the clustering reflects the model we expect, *Attributed Modularity* scores increase. One of the clusterings is a combination of attribute and structure properties whereas the other two only utilise one type of information.

The graph 3.1(a) is a simple example of a one-dimensional attributed graph clustering problem. All edges have weight one and each vertex has one attribute having either the value 1 or 9. The vertex attribute has a global variance of $\sigma^2_1 = 14.694$. The expected clustering is shown in 3.1(b) having two clusters which have high connection in the cluster itself and sparse connections between the clusters. From the attribute side each cluster is distinguished in having exactly one attribute value which differs from that of the other clusters. One vertex having the attribute value 9 surrounded by only vertices having attribute value 1 is considered as an outlier as it would fit from the perspective of the structure into the upper cluster but its attribute value separates it significantly from its neighbours. Using the existing measures this clustering has a modularity of 0.176 and each cluster has a relevance of 1 as all local variances are 0. Due to the relevance of 1 in all clusters (which have more than one object) the *Attributed Modularity* of this clustering is 0.176, too.

If a modularity based algorithm is applied to the graph, the result would be as shown in clustering 3.1(c). In contrast to the first clustering, two objects having the attribute value 9 are clustered in a community which mainly consists of vertices having attribute value 1. This is the effect that the taken algorithm only clusters by using the graph structure and ignores all attribute values. Resulting of the fact that a modularity-based clustering algorithm is used, this cluster has a score of 0.2728 in *Modularity* which is higher than the previous graph. As a direct consequence of the structure-only clustering only the lower cluster has a relevance of 1 but the upper cluster now has a local variance of $\sigma^2_{1,u} = 13.061$, so that its relevance score decreases to $1 - 13.061/14.694 = 0.11113$. The lower relevance of the upper cluster acts as a penalty on the overall clustering score so that the *Attributed Modularity* decreases to 0.1516.

The last graph 3.1(d) is used as an example to show how the effects of attribute-only clustering are penalised by *Attributed Modularity*. Due to the fact that the clustering only recognises the attributes values both clusters have a relevance of 1 on the attribute. A major effect of ignoring the graph structure is visible in the fact that the cluster containing all objects having the attribute value 9 is made up of two unconnected components. This decreases the *Modularity* score to 0.1275 and following out of that *Attributed Modularity* is at 0.1275, too.

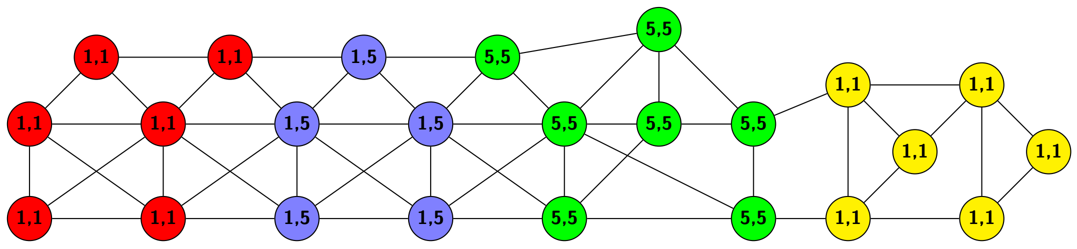


Figure 3.2.: Clustering where a *Modularity* based cluster would be split

As more information than in traditional graph clustering algorithms is used, it is likely that the resulting clusters will be smaller. This will occur if a *Modularity*-based cluster consists of several attribute-based which have still dense connections. The graph in Figure 3.2 would be an example where *Modularity* would suggest two clusters, the yellow and a cluster containing the other colours, with an overall score of 0.206. The graph has two dimensions and two attributes values 1 and 5 per dimension. The first dimension (the values before the comma) has a variance of $\sigma_1^2 = 3.306$ whereas the second has a variance of $\sigma_2^2 = 4$. The right *Modularity*-cluster has a relevance of 2, i.e. in both dimensions the vertices have all the same values. The left *Modularity*-cluster has a total relevance of 0.111 as in the first dimension the variance is higher than the global variance and in the second dimension it is only slightly smaller. Due to the relevances the overall *Attributed Modularity* is 0.1204 for the *Modularity*-based clustering.

If the clusters as indicated by the colours in Figure 3.2 were used, the *Modularity* score would decrease to 0.181. Regarding the attributes, the values in each cluster are the same among the vertices which results in a relevance of 2 for each cluster. As all cluster have full relevance the *Attributed Modularity* score is the same as the *Modularity* score 0.181 which is significantly higher than the *Attributed Modularity* score of the *Modularity*-based clustering.

A computational disadvantage of *Modularity* is that its optimisation is an NP-hard problem [BDG$^+$08]. This computational complexity can be transitively applied to *Attributed Modularity* with the following scheme. Given a graph which would be used for *Modularity* clustering. Add an arbitrary number of attributes to it such that all vertices should have the same values. Afterwards an isolated node that has differing values in all attributes is added so that the global variance of each attribute is greater than 0. The relevance of all dimensions in the original graph is 1 independent of chosen clusters so that in that part of the graph *Attributed Modularity* is equivalent to *Modularity*. If the isolated node is added to a cluster the relevance of all attributes of this cluster less than 0. Due to this fact and that it is not connected to another vertex, the isolated node will never be part of another cluster. The resulting optimal clustering for *Attributed Modularity* will be an optimal clustering for *Modularity* on the original graph without the isolated node. Hence, optimising *Attributed Modularity* is an NP-hard problem, too.

3.3. Outlier Scores

Score w.r.t. a community

In the local environment of a community the relevance and variance of each dimension are already determined by the clustering process. The knowledge of these parameters enables the normalisation of the data along each axis which eliminates the need of techniques that cope with different local densities, cp. LOF [BKNS00]. Let R be the set of relevances of all relevant dimensions with positive variance σ^2, so that $1/\sigma^2$ is defined. Using the mean $\overline{x_d}$ of dimension d $\overline{x_d}$ from the clustering process leads to the normalised deviation of x_i in community c with size n_c along dimension d given by:

$$\overline{x}_{i,c,d} = \left| \frac{x_{i,d} - \overline{x_d}}{\sigma_{c,d}} \right| \tag{3.5}$$

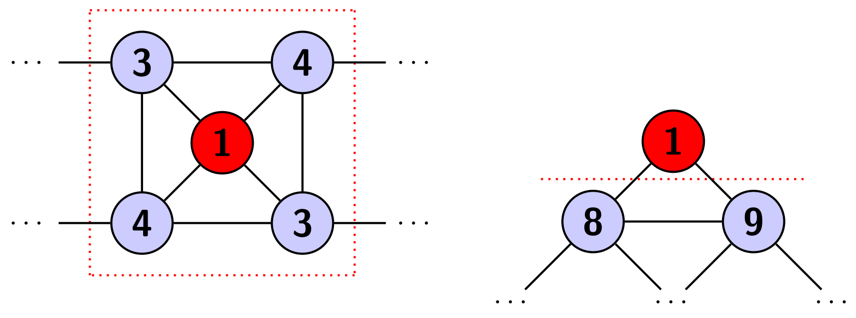
As the above function does not distinguish how relevant w.r.t. the community a dimension is, this should be included on calculating the overall score w.r.t. the community. As outlier scores should be comparable between different communities the combined score is normalised by the total number of relevant dimensions D with each having the relevance r_d.

$$\overline{x}_{i,c} = \frac{1}{|D|} \sum_{d \in D} \overline{x}_{i,c,d} * r_d$$

Overall Combined Score

Regarding the output of the clustering process, outliers occur in two different settings. An outlier may vary significantly in the attributes from its neighbours though it will integrate with its structure in the community as shown in Figure 3.3(a), so that the increase in modularity outweighs the decrease in relevance. The larger a community is the likelier this will occur. Whereas in the other situation shown in Figure 3.3(b) it will form a community with a very small size (including size 1) as the decrease in relevance is far higher than the possible structural increase in modularity.

To overcome the problem that outliers occur in different forms the outlierness of a vertex should not only be computed w.r.t. its own community. If the community of a vertex is small, the structure may be too weak to form a larger cluster.



(a) Outlier embedded in a cluster (b) Outlier in a size-one cluster

Figure 3.3.: Different settings the outlier scores has to take care of (red lines indicate community borders)

Definition 2 *Given the set of edges $E_{x_i,c}$ which connect x_i to vertices in community c, $c(x_i)$ the community of x_i and C the set of communities incident to x_i excluding $c(x_i)$, the proposed outlier score is computed via:*

$$\begin{aligned} m_{x_i,c} &= \sum_{e\in E_{x_i,c}} w(e), m_{c(x_i)} = \sum_{e\in c(x_i)} w(e) \\ score(x_i) &= \sqrt[\max(m_{c(x_i)},\max_{c\in C}(m_{x_i,c}))]{\frac{(\overline{x}_{i,\overline{c}})^{m_{c(x_i)}} + \sum_{c\in C}(\overline{x}_{i,c})^{m_{x_i,c}}}{1+|C|}} \end{aligned} \tag{3.6}$$

Though this formula seems complex, it has some simple asymptotic properties that match the settings in Figure 3.3. Taken the assumption (see [LLDM08] for a list of average node degrees in large social networks) that the degree of a vertex in a rather large community is significantly smaller than the community size, the score could be approximated through

$$score(x_i) \approx \overline{x}_{i,c(x_i)} \tag{3.7}$$

This approximation is inferred through the following asymptotic behaviour under the condition $\lim_{m_{c(x_i)}\to\infty} m_{c(x_i)}/d(x_i) = \infty$ that the score will only depend on its own community:

$$\begin{aligned} \lim_{m_{c(x_i)}\to\infty} score(x_i) &= \frac{\lim_{m_{c(x_i)}\to\infty} \sqrt[m_{c(x_i)}]{(\overline{x}_{i,c(x_i)})^{m_{c(x_i)}} + \sum_{c\in C}(\overline{x}_{i,c})^{m_{x_i,c}}}}{\lim_{m_{c(x_i)}\to\infty} \sqrt[m_{c(x_i)}]{1+|C|}} \\ &= \lim_{m_{c(x_i)}\to\infty} \sqrt[m_{c(x_i)}]{(\overline{x}_{i,c(x_i)})^{m_{c(x_i)}} + \sum_{c\in C}(\overline{x}_{i,c})^{m_{x_i,c}}} \end{aligned} \tag{3.8}$$

$$\left\{ \begin{array}{l} \geq \lim_{m_{c(x_i)}\to\infty} \sqrt[m_{c(x_i)}]{(\overline{x}_{i,c(x_i)})^{m_{\overline{c}}}} = |\overline{x}_{i,c(x_i)}|_\infty \\ \leq |\overline{x}_{i,c}|_\infty + \lim_{m_{c(x_i)}\to\infty} \underbrace{\sqrt[m_{c(x_i)}]{\sum_{c\in C}(\overline{x}_{i,c})^{m_{x_i,c}}}}_{\to 0} \end{array} \right\} = \overline{x}_{i,\overline{c}}$$

Looking at the other extreme that the viewed vertex is forming a community of size one and it has at most one edge (with weight 1) to its neighbouring communities the score is simplified to a rather uncomplicated average.

$$score(x_i) = \frac{\sum_{c \in C} \overline{x}_{i,c}^{m_{x_i,c}}}{1 + |C|} \tag{3.9}$$

As an example the outlier in Figure 3.3(b) is forming a size-one community so that it has an outlier score of 0 w.r.t to its community. Under the assumption that the neighbouring community has a mean of 8.5, a variance of 0.25 and a relevance of 0.8, the outlier has a score of 9.5244. In contrast Figure 3.3(a) shows an outlier which is part of a community. Taken that this community has a mean of 3, a variance of 1.2 and a relevance of 0.8 the score for the outlier is directly its community outlier score which is 1.8257. Compared to the outlier, the vertices with attribute 3 in this community have an outlier score of 0. Those with attribute value 4 have a score of 0.91287 which makes the outlier clearly distinguishable from the other community members.

4. Algorithmic Approaches

To optimise the scores presented in the previous chapter, some algorithmic approaches are presented in the following. The proposed algorithms all optimise the *Attributed Modularity* score on a heuristic basis to achieve polynomial running time. As optimising *Attributed Modularity* is an NP-hard problem, in the current situation, a algorithm with exponential runtime would be needed. All heuristics use a general scaffold which is presented as the first part of this chapter. As the main part of this chapter, the functionality of all proposed algorithms is outlined. For each approach the runtime in the O-calculus is provided.

Due to different available techniques the proposed algorithm is given as a scaffold in which at one point different techniques are exchangeably used as described in Algorithm 1. The proposed techniques either look at the local context of a vertex or at the global scope of all vertices when deciding which change to the clustering should be made. After loading the graph, the global variance σ_k^2 for each dimension k is computed using the *corrected two-pass algorithm* suggested by Chan, Golub and LeVeque [CGL83]:

$$\sigma_k^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,k} - \overline{x_k})^2 - \frac{1}{N^2}\left(\sum_{i=1}^{N}(x_{i,k} - \overline{x_k})\right)^2$$

At this point the techniques that will be explained in the following subchapters can be interchangeably integrated. To concatenate multiple approaches a common interface is used. As shown later in Chapter 5 the combination of two techniques might lead to better or faster results than only using one of them. Each algorithm is given the graph on which clustering is done, the communities determined by the previous step and the precomputed global graph properties. If there is not a previous clustering, an initial singleton clustering where each vertex forms a cluster is set. One technique is iteratively used until it converges, i.e. no more increase in *Attributed Modularity* could be obtained by applying it to the graph.

After a combination of the techniques has run, the outlier score for each vertex is computed. Its own community, neighbouring communities and the total weight of all edges linking the current vertex to its neighbouring communities is collected. Using the list of communities the outlier score of the current vertex is calculated w.r.t. each of these communities. Having computed the outlier scores for each community using Equation 3.6 the combined score is computed.

Algorithm 1 Algorithm Scaffold

```
compute all global variances
compute weighted degree $k(x_i)$ for $x_i \in V$, m := $\sum_{x_i \in V} k(x_i)$
c := $\{\{v\} \mid v \in V\}$
for technique $\in$ usedTechniques do
    c := runTechnique(c)
emit c
for $x_i \in V$ do
    score := outlierScore$(x_i, c(x_i))^{m_{c(x_i)}}$
    for nC $\in \{c(v) \mid \{x_i, v\} \in E\}$ do
        weight := $\sum_{\{x_i,v\} \in E} w(\{x_i, v\})$
        score += outlierScore$(x_i, \text{nC})^{\text{weight}}$
    score /= $1 + |\{c(v) \mid \{x_i, v\} \in E\}|$
    emit outlierRoot$(x_i, score)$
```

Regarding the optimisation of *Modularity* scores, Noack and Rotta [NR09] give an overview of the currently available and used techniques. These approaches provide a template for the fast optimisation of graph clustering and have shown their practical usability for optimising *Modularity* scores. The following techniques are based on them but are modified to use *Attributed Modularity* as their objective function. Though the main task was to convert the objective function to support numerically stable incremental computation there are some small changes to the algorithm techniques used.

For the underlying graph structure some runtime properties are assumed. The list of edges incident to a vertex could be inserted in $O(degree)$ into the graph and adding a edge takes $O(1)$. A vertex can be retrieved using its vertex *id* in expected constant time when using arbitrary vertex ids. This can be a achieved using a *HashMap*-like structure. This runtime could lowered to constant deterministic time if all vertex ids were moved into a continuous integer range. Adding a vertex should take (probabilistic) constant time, too.

4.1. Clustering Approaches

To maximise the objective function in the following different techniques are presented. The techniques vary in runtime and their approach on how to increase the *Attributed Modularity*. Though most techniques already produce good results on their own, combining them will lead to better results and sometimes a decrease in overall runtime. In Chapter 5 an evaluation how these steps perform will be given.

4.1.1. Local Move *(LM)*

The first proposed approach only considers the local neighbourhood of each vertex on deciding which change to do. In one iteration each vertex of the graph is scanned and moved temporally into a size-one community. Then, for all neighbouring communities, the increase in *Attributed Modularity* when moving the vertex into this community is computed. The vertex is moved to the community with the highest increase in its neighbourhood. Each inner iteration of the *while*-loop of Algorithm 2 takes $O(n + m * d)$ time. One of the major properties to achieve this is that the calculation of the change in the *Attributed Modularity* on removing or adding a vertex v to a community c can be calculated in $O(d + degree(v))$ (exactly in $O(d + |E_{v,c}|)$)time. In the following paragraphs the mathematical details are depicted how existing equations need to be transformed for fast and numerically stable computation.

Algorithm 2 Local Move Iteration

```
while increase > 0 do
    increase := 0
    for $x_i \in V$ do
        curCommunity := $c(x_i)$
        decrease := $\mathrm{AM}_{Dec}(\mathrm{curCommunity}, x_i)$
        curCommunity $-=$ $x_i$
        neighCommunities := $\{c(v) \mid \{x_i, v\} \in E\} \cup \mathrm{curCommunity} \cup \textit{new}\ \mathrm{Community}$
        for nC $\in$ neighCommunities do
            score(nC) := $\mathrm{AM}_{Inc}(\mathrm{nC}, x_i)$
        destination := $\mathrm{maxarg}_{\mathrm{nC} \in \mathrm{neighCommunities}}(\mathrm{score(nC)})$
        assert score(destination) + decrease $\geq 0$
        destination $+=$ $x_i$
    assert increase $\geq 0$
```

Incremental *Modularity* computation

For *Modularity* increase Blondel et al. [BGLL08] already uses the ability to compute the increase of modularity when adding a vertex to a community directly instead of recalculating it on the whole graph. The weighted degree k_i of each vertex i and the total sum of all weighted degrees $dSum$ are precomputed and constantly updated per community. The increase of modularity $\mathrm{MI}_{i,l}$ by adding a vertex x_i (without a community) to a community l is given by (where $c(j)$ is the community of j):

$$\mathrm{MI}_{i,l} = \frac{\sum_j A_{ij}\delta(c(j), l)}{m} - \frac{k_i * \mathrm{dSum}_l}{2m^2} \tag{4.1}$$

As we only sum over all edges connected to the suggested community and all other values are already precomputed, the increase for all neighbouring communities is calculated in $O(degree(x_i))$ time.

Incremental *Relevance* computation

For the fast computation of the relevance part of *Attributed Modularity*, the essential part is to compute the variance of each dimension k numerically stable in $O(1)$. In the *Local Move* algorithm the relevant operations are the addition and the removal of a data point which change the variance. In statistical textbooks the following *one-pass algorithm* is suggested for incremental computation of the variance for N data points:

$$S_k = \sum_{i=1}^{N} x_{i,k}^2 - \frac{1}{N}\left(\sum_{i=1}^{N} x_{i,k}\right)^2 \tag{4.2}$$

$$\sigma_k^2 = \frac{1}{N} S_k \tag{4.3}$$

Though this formula could be easily used for incremental computation, it is numerically unstable if used with double values. Chan, Golub and LeVeque [CGL83] suggest West's algorithm [Wes79] and the updating formulae of Youngs and Cramer [YC71] for numerically stable computation of the variance. As Youngs and Cramer [YC71] support the merging of two clusters in $O(1)$ under stable conditions it is always used in all upcoming situations where not a total recomputation is done. Starting with $T_{1,1} = x_1$ and $S_{1,1} = 0$

using Equations 4.4 and 4.5 the variance of a set of j data points with the knowledge of the variance for the first $j-1$ data points can be computed using:

$$T_{1,j} = T_{1,j-1} + x_j \tag{4.4}$$

$$\begin{aligned} S_{1,j} &= S_{1,j-1} + \frac{1}{j(j-1)}\left(jx_j - T_{1,j}\right)^2 \\ \sigma_j^2 &= \frac{S_{1,j}}{j} \end{aligned} \tag{4.5}$$

The increase in variance on adding a data point x_j to a set of $j-1$ data points can be easily obtained:

$$\begin{aligned} \sigma_j^2 - \sigma_{j-1}^2 &= \frac{1}{j}S_{1,j} - \frac{1}{j-1}S_{1,j-1} \\ &= \frac{1}{j(j-1)}\left(\frac{1}{j}\left((j-1)x_j - T_{1,j-1}\right)^2 - S_{1,j-1}\right) \end{aligned} \tag{4.6}$$

The above Equations 4.4, 4.5 and 4.6 can be transformed to calculate the decrease of variance on removing a data point of a set of j data points.

$$T_{1,j-1} = T_{1,j} - x_j \tag{4.7}$$

$$S_{1,j-1} = S_{1,j} - \frac{1}{j(j-1)}\left(jx_j - T_{1,j}\right)^2 \tag{4.8}$$

$$\sigma_{j-1}^2 - \sigma_j^2 = \frac{1}{j(j-1)}\left(S_{1,j} - \frac{1}{j-1}\left(jx_j - T_{1,j}\right)^2\right) \tag{4.9}$$

Looking at a cluster l we need to calculate the relevance $R_{k,l}|_{n+1}$ for a dimension k if we add a vertex to this cluster. Given the current relevance $R_{k,l}|_n$ and the respective variance $\sigma_{k,l}^2|_n$ and $\sigma_{k,l}^2|_{n+1}$, the relevance increase is obtained using the variance increase.

$$\begin{aligned} R_{k,l}|_{n+1} - R_{k,l}|_n &= \left(1 - \sigma_{k,l}^2|_{n+1}\right) - \left(1 - \sigma_{k,l}^2|_n\right) \\ &= -\frac{1}{\sigma_k^2}\left(\sigma_{k,l}^2|_{n+1} - \sigma_{k,l}^2|_n\right) \end{aligned} \tag{4.10}$$

In *Attributed Modularity* the relevance is capped at 0, so that only dimensions with a positive relevance count towards the score. To respect this property in the incremental computation, the increase of relevance $RI_{k,l,n+1}$ that counts towards the global score is:

$$RI_{k,l,n+1} = R_{k,l}|_{n+1} - R_{k,l}|_n - \min(R_{k,l}|_{n+1}, 0) + \min(R_{k,l}|_n, 0) \tag{4.11}$$

As for each dimension the relevance increase can be computed in $O(1)$, the overall relevance increase is computed in $O(d)$. Having the old *Modularity* Mod_l and *Relevance* $R_{k,l}$ scores the *Attributed Modularity* of a cluster l without and with an additional vertex is given by

$$\mathrm{AM}_l = \mathrm{Mod}_l * \sum_{k=1}^{d} \max(0, R_{k,l}) \tag{4.12}$$

$$\mathrm{AM}_l \mid_{n+1} = (\mathrm{Mod}_l + MI_{i,l}) * \left(\sum_{k=1}^{d} \max(0, R_{k,l}) + \sum_{k=1}^{d} RI_{k,l,n+1} \right) \tag{4.13}$$

Using the difference of the above two formulae the increase $\mathrm{AMI}_{l,i}$ in the objective function if moving vertex x_i into community l could be computed in $O(d + deg(x_i))$ time using:

$$\mathrm{AMI}_{l,i} = \mathrm{Mod}_l \left(\sum_{k=1}^{d} RI_{k,l,n+1} \right) + MI_{i,l} * \left(\sum_{k=1}^{d} \max(0, R_{k,l}) \right) + MI_{i,l} \left(\sum_{k=1}^{d} RI_{k,l,n+1} \right) \tag{4.14}$$

Using Algorithm 2 and the above formulae, one *Local Move* iteration can be run. To show the correctness of the two mentioned asserts in the pseudocode, a simple consideration how the algorithm works should suffice. As a vertex is moved out of a community to which it could be moved back, the highest score is at least as high as the decrease. Given this, in all situations at least moving the vertex back into its original community will not decrease the overall score. Because each move only adds positive scores to the total increase, the second assert is correct, too.

A series of *Local Move* iterations can be considered as converged if a round is reached where there are no vertex moves possible. As only positive increases to the score are made and *Attributed Modularity* has the upper bound of 1, the algorithm always converges.

One iteration of *Local Move* has exactly n inner iterations of its for-loop as each vertex is considered once for a move. Using a HashMap, in expected $O(1)$ time the current community of a vertex is retrieved. To calculate the decrease when moving a vertex out a community using the above formulae, we need to look at most on all outgoing edges. Additionally for each dimension we could determine the relevance decrease in $O(1)$ so that the overall decrease in *Attributed Modularity* is calculated in $O(d + degree)$ time. For selecting the new community a scan over all edges of the current vertex is needed to determine the neighbouring communities and the total edge weight linking to them. For the maximum of *degree* communities neighbouring to the vertex the increase in relevance is computed which sums up to $O(degree * (d + 1))$ time for the calculation of the *Attributed Modularity* scores for the move possibilities. In the end the vertex is assigned to the community with the highest increase in *Attributed Modularity*. One inner iteration of the for-loop takes $O(d * degree)$ time and given the fact that each vertex is looked at once, the total runtime of an outer for-loop iteration of *Local Move* can be amortised to $O(n + m * d)$.

4.1.2. Coarsening Local Move *(Coarse)*

As *Local Move* only considers movements of the size of one vertex, it will likely end up in a local maximum where there are no increases which could be achieved by moving one vertex. To utilize the concept and performance of *Local Move* the graph is transformed so that we could still run *Local Move* on it but considering “higher level” movements.

The main concept of this approach it to iteratively merge each community into a single vertex and and run *Local Move* on the resulting graph. Contracting the graph structure is simple as the sum of the edge weight of a community is added as a loop and to each neighbouring community an edge of the total weight linking both is created. In contrast to the structure the contraction of the attribute values is less straight-forward. As mentioned in the previous subchapter, the variance of a community can be stored using two variables S and T but in the contracted graph groups of vertices are moved (in contrast to a single vertex in *Local Move*). To keep the runtime of $O(1)$ in computing the increase/decrease in variance if one vertex is moved into another community other formulae need to be used.

Using the algorithm of Youngs and Cramer [YC71], the combined variance of two sets of data points of size n, m can be computed using:

$$T_{1,m+n} = T_{1,m} + T_{m+1,m+n} \tag{4.15}$$

$$S_{1,m+n} = S_{1,m} + S_{m+1,m+n} + \frac{m}{n(m+n)}\left(\frac{n}{m}T_{1,m} - T_{m+1,m+n}\right)^2 \tag{4.16}$$

Using a simple transformation the above formula could be used to remove a set of n data points given their respective S, T values from a set of $m+n$ data points in $O(1)$ time. In addition to updating the value of S and T the increase/decrease in the biased variance is calculated through:

$$\sigma^2_{m+n} - \sigma^2_m = \frac{S_{m+1,m+n}}{m+n} + \frac{m}{n(m+n)^2}\left(\frac{n}{m}T_{1,m} - T_{m+1,m+n}\right)^2 - \frac{n}{m(m+n)}S_{1,m} \tag{4.17}$$

$$\sigma^2_m - \sigma^2_{m+n} = \frac{n}{m(n+m)}S_{1,m+n} - \frac{S_{m+1,m+n}}{m} - \frac{1}{n(m+n)}\left(\frac{n}{m}T_{1,m} - T_{m+1,m+n}\right)^2 \tag{4.18}$$

Algorithm 3 Coarse Iteration

```
$g_{new}$ := $new$ Graph
Communities$_{new}$ = $new$ Set
for $c \in$ Communities$_{old}$ do
    $g_{new}$. addVertex($v(c)$)
    Communities$_{new}$ . add($new$ Community($\{c\}$))
    $g_{new}$. addEdge($v(c), v(c), \text{tot}_c$)
for $c \in$ Communities$_{old}$ do
    neighCommunitites := $\{c_{old}(v) | x_i \in c, \{x_i, v\}, c_{old}(v) \neq c\}$
    for $nC \in$ neighCommunitites do
        w(c,nC) := 0
for $\{u, v\} \in E_{old}$ do
    w(c(u), c(v)) += w({u,v})
localMove($g_{new}$, Communities$_{new}$)
```

Algorithm shows an iteration of *Coarsed Local Move* where the graph is contracted. *Coarsed Local Move* itself converges if a *Local Move* directly after contraction does not show any more increase, i.e. there are no more movements possible even if considering moving vertices that were contracted out of communities.

Using the mentioned formulae for handling the move of a vertex which represents a set of data points, *Local Move* is used with its $O(n + m * d)$ runtime. In addition the contraction of the graph adds some overhead to the algorithm. As a worst case upper bound, the original graph may be clustered in to a maximum of n communities, i.e. each vertex still forms one community. The size of the newly generated graph is always at most the size of the original graph. The creation of the new graph including the vertices could be done in $O(n)$ though each community needs to be copied as we instantiate new communities with the original community as the only vertex. As the data structure of a community stores the local variances of all attributes, it needs $O(d)$ space so that the copy process will consume overall $O(n * d)$ time. To get all neighbours of a community a scan over all edges of a community is needed, taking $O(\sum_{v \in c} degree(v))$ in time. Overall an edge is only looked at twice while determining the neighbourhoods so that the runtime of this part can be amortised to $O(m)$. For calculating the weights and the creation of the connecting edges and the same amount of time is needed as we will not look at any edge more than twice. To ensure that an edge is not added twice during the creation of the new communities, each community is assigned an index. If a new edge should be added it is checked that the index of the current community is smaller than that of the neighbouring community. This check only needs $O(n)$ extra preprocessing time which does affect the total runtime in the O-calculus. In sum the contraction of a graph to a graph where each community is represented by a vertex takes $O(n * d + m)$ time.

After using the *Coarsed Local Move* step some postprocessing is required as we need to determine to which community one of the original data points belong. Using the map of which vertex belongs to one of the communities of a previous run, i.e. the non-contracted graph, based on the clustering of the contracted graph each vertex is reassigned to one of the contracted communities. This back-transformation needs to be made after all contraction rounds but each round only adds $O(n)$ time as it contributes at maximum one layer.

4.1.3. Global Merge *(Merge)*

In contrast to the two already mentioned steps, this approach varies in its concept as it joins clusters instead of moving vertices between them. Though we can determine the overall runtime of *Global Merge* instead of only per inner iteration, it is significantly slower than the other steps. The concept of this step is influenced by [New04].

During this approach, a priority queue of all possible merges of **two** clusters is maintained. The head of this queue is the merge which would end up in the highest increase in *Attributed Modularity*. At the beginning all possible merge scores are computed and inserted into the queue. Merge scores are only computed for neighbouring clusters as non-connected cluster would have worse modularity scores as well as that they disagree with the definition of the searched clusters (high interconnection).

For the calculation of the variance Equations 4.15 and 4.16 can be used to get the combined variance of a dimension in constant time. To compute the increase in Modularity on merging two clusters k_1 and k_2 only the increase in the total inner-community degree weight tot_{Inc} needs to determined, the new modularity itself can then be calculated in constant time.

$$\mathrm{tot}_{Inc} = 2 * \sum_{v \in k_1} \sum_{u \in k_2} A_{vu} \tag{4.19}$$

$$\mathrm{Mod}_{k_1 \cup k_2} = \frac{\mathrm{tot}_{Inc} + \mathrm{tot}_{k_1} + \mathrm{tot}_{k_2}}{2m} - \left(\frac{dSum_{k_1} + dSum_{k_2}}{2m} \right)^2 \tag{4.20}$$

As already mentioned in Chapter 4.1 the sum of all node degrees dSum_k in a community k is continuously maintained throughout all operations so that it is available in constant time.

After a successful merge, the neighbours of a the newly created community are obtained. All scores which consider one of the merged clusters are removed from the priority queue. For each neighbouring community the score is calculated and added to the priority queue. This procedure is repeated until no more merges with a positive increase are in the priority queue.

Algorithm 4 Global Merge

```
increase := 0
pq := *new* PriorityQueue
**for** $c_1 \in$ Communities **do**
    **for** $c_2 \in$ Communities **do**
        **if** $idx(c_1) \leq idx(c_2)$ **then** continue
        score := mergeScore($c_1$, $c_2$)
        **if** score >0 **then** pq.add(score, $c_1$, $c_2$)
**while** pq.hasElement **do**
    (score, $c_1$, $c_2$) = pq.pop
    increase += score
    $c_3$ := merge($c_1$, $c_2$)
    **for** $c_4 \in R$ **do**
        pq.remove(_, $c_4$, $c_1$)
        pq.remove(_, $c_4$, $c_2$)
        score := mergeScore($c_3$, $c_4$)
        pq.add(score, $c_3$, $c_4$)
```

In addition to the above pseudocode, during this step a list of the edge weight between each pair of communities is maintained. This list is pregenerated at the beginning in $O(m)$ time and updated after the merge for each combination of the new community and its neighbours. Using this list, it is possible to calculate $\text{tot}_{Increase}$ and so *Modularity* in constant time while generating the merge scores. Overall the computation of the merge score of two communities takes $O(d)$ time.

For the updating of the relevance during merges, the Equations 4.15 and 4.16 can be used. They allow updating the relevance of the combination of two communities in $O(d)$ time. A merge of two communities is just treated like if one community would have been moved into another community.

At the beginning a maximum of $O(m)$ initial merge scores need to be calculated, giving the initialisation phase $O(m * d)$ time. After each new merge, $O(n)$ scores need to be updated and added to the priority queue thus this will take $O(n*(d+log(m)))$ time as the priority queue may store up to $O(m)$ merge candidates. The number of merges is capped at $n - 1$ which would end in a single community. The priority queue could be implemented as a binary so that the initial creation would take $O(m)$ and the update of a score $O(log(m))$ time. Summed up, this step will run in $O(n^2 * (log(m) + d))$ time until convergence.

4.2. Handling Communities

Throughout the whole algorithm a list of all communities is used as stateful information and passed on by all the steps. Besides the main information, which vertices belong to a community, several values are stored to enable fast computation of the used measures. For computing modularity, the modularity itself is kept as a variable as well as the total weight *tot* of all edges that connect two vertices in the community. As already described the sum of the degree of all vertices of the community is stored to support the computation of the modularity during cluster merges. In contrast to *Modularity* the relevance of a dimension is not stored directly but computed on demand in constant time out of the variance of each dimension. To get the current variance of a dimension, the sum of all data points T per attributes in a cluster and the squared distance of each data point to the mean of this dimension S are held in memory to utilize the algorithm described by Youngs & Cramer [YC71].

On communities only three basic operations are performed: the addition and removal of a single data point and the merge of two communities. The first two are used (and described) during the *Local Move* and *Coarsening* steps whereas the merging is only used by the *Merge* approach.

To get the community of a vertex or handle the list of all vertices of a community hash function based structures (e.g. *HashMap* and *HashSet*) could be used for arbitrary vertex ids. This allows the addition and removal to the list of all vertices of a community as well as retrieving the community of a vertex in probabilistic constant time. If one converts all vertex ids to a continuous integer range, a simple array could be used so that some of these operations could be run in deterministic constant time.

5. Evaluation

In this chapter the previously introduced algorithms are tested on their practical efficiency and quality. The algorithms will be applied to synthetic data and to real world data. The influence of the number of vertices, the number of communities or dimensionality is measured. In contrast to other algorithms we do not need to test the variation of the algorithm parameters as all algorithm instantiations are parameter-free. As part of the evaluation it will be highlighted that *LM_Merge* is the best instantiation for optimising *Attributed Modularity* as it is one of the best in quality and achieves this with good runtime results.

As the proposed algorithm is a scaffold with three possible steps that could be run as a combination, we need to test various instantiations. In the following *LM* is the instantiation where only *Local Move* is used and *Merge* where only *Global Merge* is used. There are the instantiations *LM_Merge* and *LM_Merge_LM* where first *Local Move* is run and then *Global* and in the second one another *Local Move* run is done afterwards. The instantiation *Coarse* combines an initial *Local Move* step with flowing contraction using *Coarsed Move*.

To check the quality of the given approaches the synthetic data has known outliers and clusters. The found clusters should be part of a apriori generated cluster to be an acceptable clustering. Apriori known outlier objects should be among the highest ranked object which the algorithm indicates that they show abnormal behaviour.

In the second part of this chapter the quality of the proposed algorithm is measured on real world data. Some of the used data set have labelled outliers and an AUC score can be computed. For the data sets that do not have pre-labelled information the result are discussed by manual exploration of detected outliers.

As a comparison to existing solutions the attribute-only outlier detection algorithm LOF [BKNS00] and the clustering approach SCAN [XYFS07] which clusters only w.r.t. the graph structure are used. In addition CODA [GLF$^+$10] serves as a third competitor that takes graph structure and attribute space into account. An already mentioned problem is that all the competing algorithms require user-given parameters. For each algorithm different parameter instantiations are used and the best result is selected. For compensating the effects of randomisation each algorithm instantiation is run five times and the average of the runtime is used. If there are any differences in the score due to randomisation the best score will be listed.

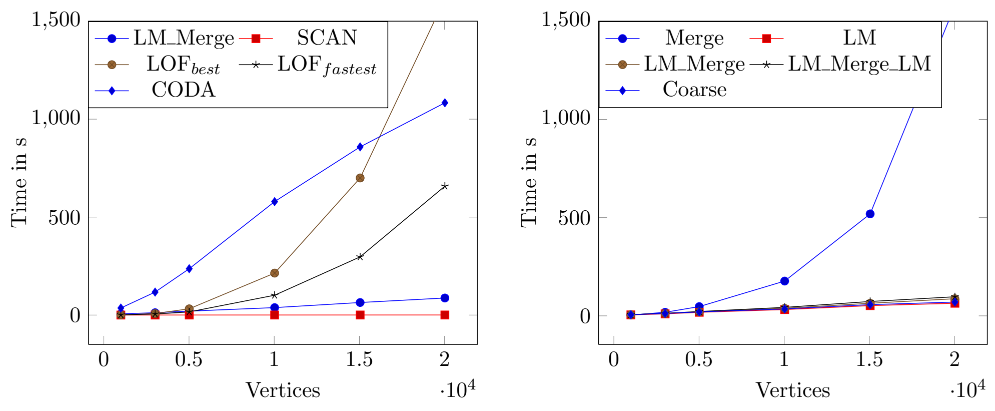
5.1. Synthetic Data

To evaluate the behaviour of the above mentioned instantiations a synthetically generated data set is used. The data set varies in the number of vertices, the number of communities and the number of dimensions. For the evaluation three sets of synthetic graphs were generated using the proposals from [LFR08]. The first set contains graphs with a fixed number of 40 communities and a varying quantity from 1000 to 20000 vertices and ten attribute dimensions. The second set of graphs has a vary number of communities starting from 5 to 30 with an average of 250 vertices per community and a fixed number of 10 dimensions. To measure the behaviour of the algorithms with a higher dimensionality the third set has a fixed number of about 1000 vertices and eight communities but varies in the number of dimensions from 10 to 100. Each graph contains 5% of structural outliers which have sparse connections to other vertices and 5% of local subspace outliers which are densely embedded in a community but deviate significantly in the relevant attributes for this cluster. In general the synthetic graph consist of one connected component which has more edges in communities than between them. The size of the communities as well as the degree of the vertices is power-law distributed.

The tests were run on a Intel(R) Core(TM) i7 CPU L 640 dual-core processor with a frequency of 2.13GHz per core. There was 8GB of RAM available to each algorithm. All algorithms were implemented in Java or Scala and run using Java 1.7.

5.1.1. Runtime

To show the scalability we evaluate the runtimes of the proposed algorithm instantiations and the competitors. As LOF varies in its runtime significantly depending on the parameters, the runtime of the fastest and of the best instantiation is listed. In this context the best instantiation is referred to w.r.t. the AUC score.

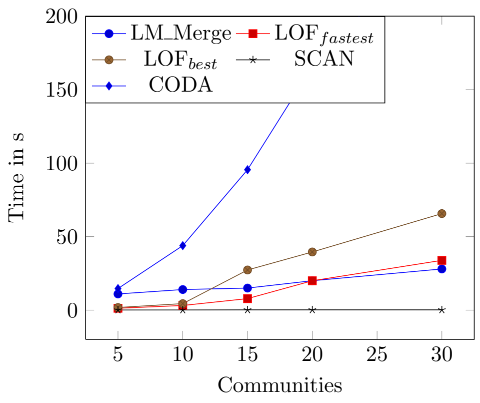


(a) LM_Merge and competitors (b) AM variants

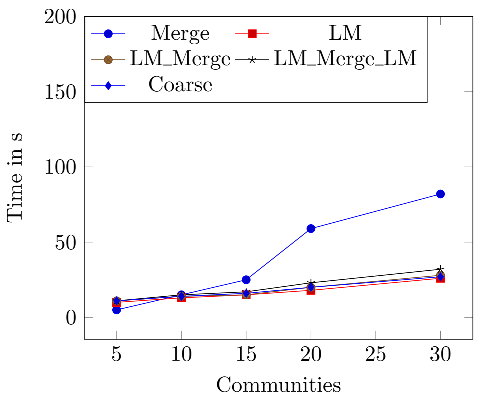
Figure 5.1.: Runtime w.r.t. number of vertices

For measuring the scalability w.r.t. to the number of vertices, a set of graphs with the fixed number of 40 communities but a varying number of vertices is used. The sizes range from 1000 to 20000 vertices with an outlier ratio of 10 percent. In Figure 5.1(b) the variants of the *Attributed Modularity* instantiations are compared to each other. Using the *Merge* step without any preprocessing leads to the behaviour that is expected from the O-calculus whereas if *LM* is used upfront, the overall runtime shows a linear behaviour as the other variants of the proposed algorithms do.

As LOF varies significantly in its runtime depending on the parameter values, the instantiation with the best AUC score and the fastest are listed. In comparison with the competitors, *LM_Merge* is slower than SCAN which only clusters using the graph structure but faster than all others. LOF needs more time as it always looks at the attribute data from a global perspective while the *Attributed Modularity* variants profit from the locality obtained by the graph structure. Though CODA respects both types of information due to its algorithm design which scales quadratically w.r.t. the number of vertices it does not scale as good as *LM_Merge*.



(a) LM_Merge and competitors



(b) AM variants

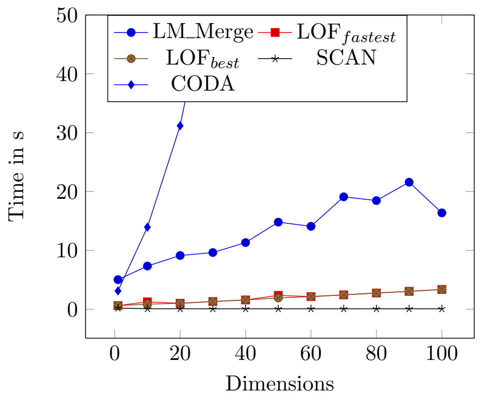
Figure 5.2.: Runtime w.r.t. the number of communities

In Figure 5.1 only the number of vertices is increased but the number of communities stayed at a constant level. To analyse the effects of the number of communities, Figure 5.2 lists the runtime with increasing number of communities. Except *Merge* all *Attributed Modularity* instantiations show the same behaviour whereas *Merge* scaled better in comparisons with the constant number of communities. This highlights that a lot of the work of *Merge* is done in merging inside a community. As graph size increases but the size of each community stays at a constant level, the number of merges still increases but due to better locality constraints *Merge* has to analyse less neighbouring communities after each successful merge of two communities.

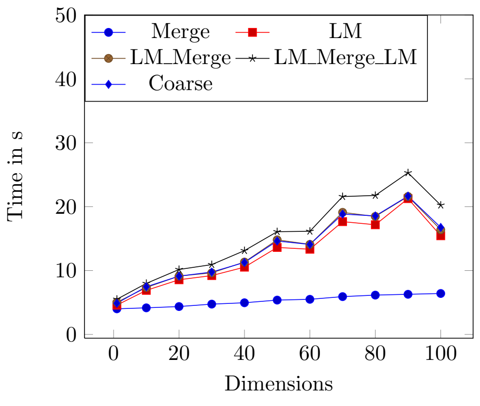
Regarding the competitors, LOF and SCAN show the same scalability properties. In contrast to them, the runtime of CODA dramatically increases with the increasing number of communities. The increase in runtime of CODA is significantly larger compared to the increase of it with a fixed number of communities.

If the number of vertices and communities are fixed and only the number of dimensions varies, as listed in Figure 5.3, all *Attributed Modularity* variants increase linearly in runtime with increasing dimensions. Though all instantiations scale linear, the variants that include *LM* as first step scale with a larger factor. This effect could be explained that *LM* computes a very high number of move scores which scale directly with the number of dimensions. In contrast *Merge* spends more time comparing scores than computing them which is independent of the number of dimensions.

Compared to the already mentioned competitors, *LM_Merge* shows the same linear asymptotic behaviour as SCAN and LOF do. Its runtime is just a constant factor higher as it requires more preprocessing on the graph. In contrast CODA suffers from the curse of dimensionality and its runtime dramatically increases with the number of dimensions.



(a) LM_Merge and competitors



(b) AM variants

Figure 5.3.: Runtime w.r.t. the number of dimensions

5.1.2. Quality of Outlier Detection

To analyse the quality of the outlier detection which is part of the proposed algorithm, the set of graphs with increasing dimensionality is analysed. As a measure to compare the different instantiations the "Area under curve" (AUC) is utilised. The AUC score is a comparison of the outlier ranking given by an algorithm with a random ranking. Given a score of 50, the outlier ranking is considered random and higher scores (up to 100) indicate better quality in finding outliers. The scores of the instantiations of the proposed algorithm are compared to the competitors which already have been used in the runtime analysis. As a basis for the evaluation a set of graphs with dimensionality from 1 to 100 using a fixed number of vertices is used. For each dimensionality the scores were computed on five different graphs and the average of all these score is listed in the diagrams. As an additional info the standard deviation of the result per dimensionality is listed too.

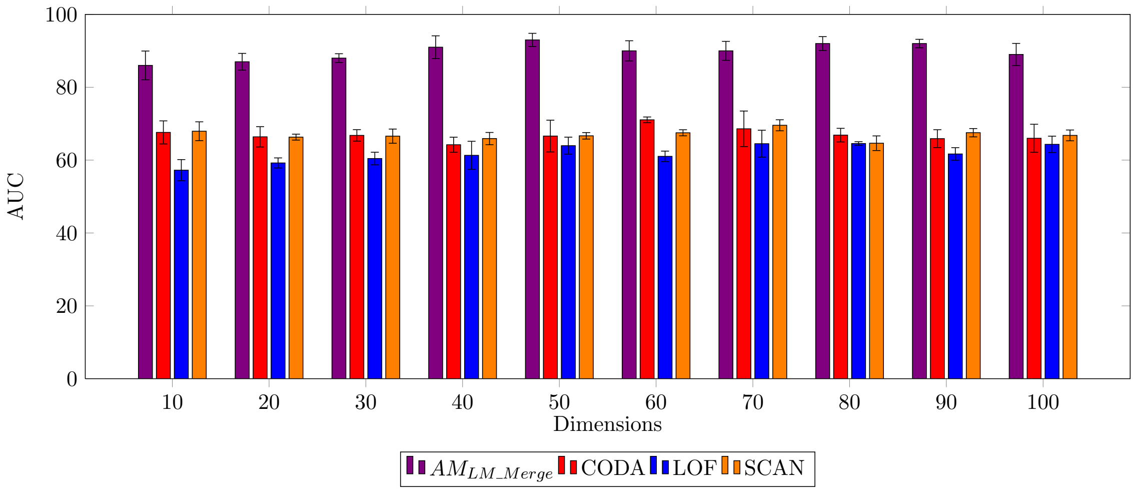


Figure 5.4.: Quality (AUC) w.r.t. increasing dimensionality (AM vs. Competitors)

As shown in Figure 5.4, *LM_Merge* has a far better quality in outlier detection than the listed competitors. SCAN only considers graph data so that it does not detect outliers which fit from the structural pattern to their neighbourhood but deviate in the attributes from the local environment. In contrast LOF only detects outliers from the attribute perspective leaving outliers which occur only in their neighbourhood w.r.t. graph structure undetected. Though CODA should detect the outliers better than the two other competitors it suffers from the curse of dimensionality. As CODA does not distinguish which attributes are characteristic for a cluster/an object it cannot distinguish local anomalies and objects that fit to the local pattern.

In Figure 5.5 the different instantiations of the *Attributed Modularity* optimising algorithm are shown. The main feature visible here is that all instantiations have constantly good results independent of the dimensionality. The three instantiations which include a *Merge* step are always a bit better than the other two. None of the three is significantly better than the other two so that cannot be given specific instantiation that performs best. Though *LM_Merge* has the best runtime properties is it used to compare the *Attributed Modularity* optimisations methods with the competitors.

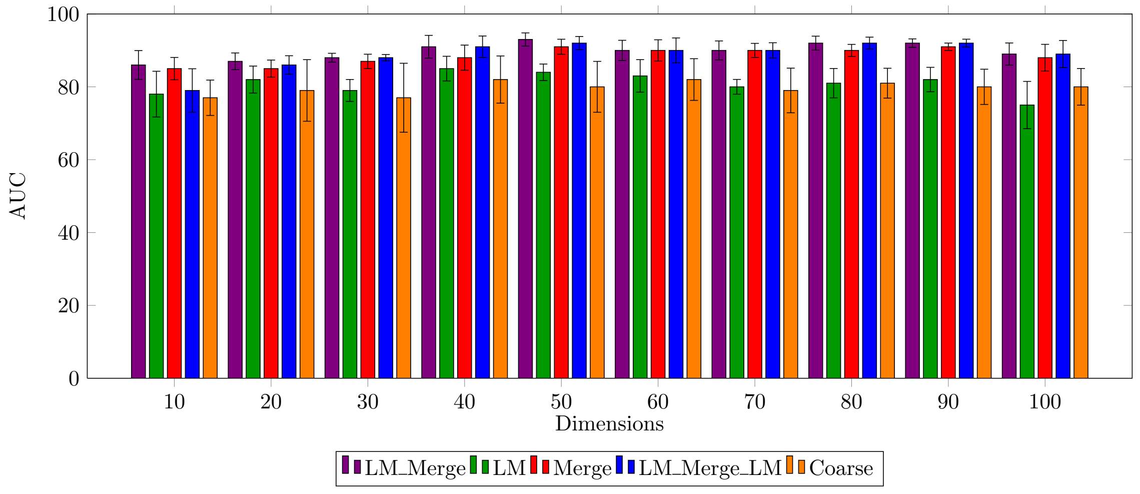


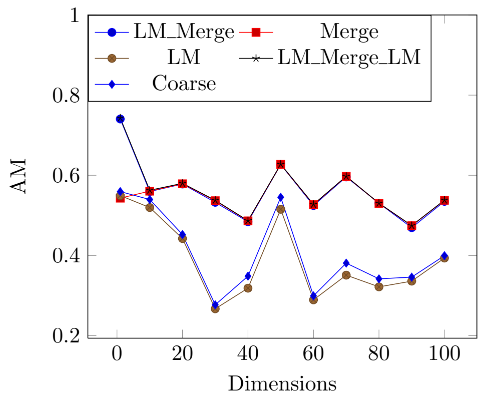
Figure 5.5.: Quality (AUC) w.r.t. increasing dimensionality (AM variants)

5.1.3. Clustering Quality

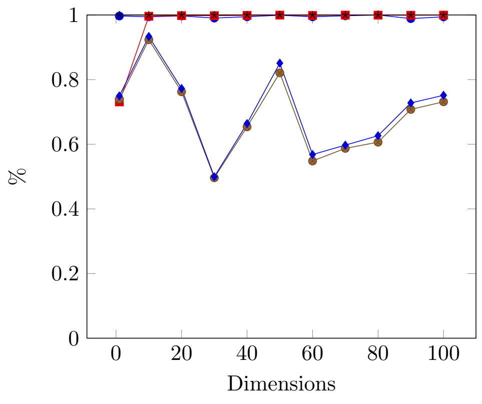
For the evaluation of the quality of the generated clusters only the *Attributed Modularity* variants are compared. To highlight which of the instantiations optimises the proposed measures best, the *Attributed Modularity* scores are computed in the three scenarios that were already used for the runtime evaluation. As *Attributed Modularity* scores are not comparable between two graphs, in each scenario the absolute scores and the scores relative to the best instantiation are listed.

In the first scenario a group of graphs with an increasing amount of dimensions is evaluated. Overall *LM_Merge*, *LM_Merge_LM* and *Merge* produce the same best result as shown in Figure 5.6. As *LM_Merge_LM* has an additional *LM* step its scores are slightly but insignificantly better then those of *LM_Merge*. The other two variants *LM* and *Coarse* have on average about 70% of the *Attributed Modularity* score of better instantiations. Depending on the graph, *LM* which is used as the first step for all variants except *Merge* is nearly as good as the instantiations based on it. In this case the local optima which was found by *LM* includes larger cluster which in similar but varied graphs may only have been formed by a following *Merge* step.

As detailed in Figure 5.7, *LM* and *Coarse* produce results about a quarter worse than those of the other instantiations independent of the number of vertices a graph contains. With an increasing size of the graph, the scores of *Coarse* slightly converges against those of *LM* though staying always above them. In conclusion *LM* suffers from its local perspective

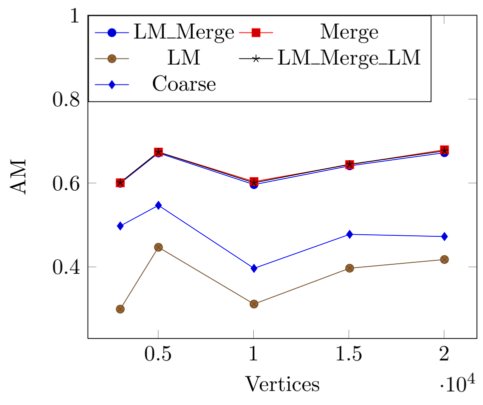


(a) Absolute *Attributed Modularity* scores

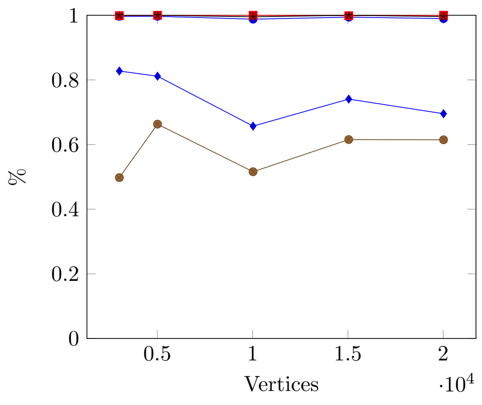


(b) Relative *Attributed Modularity* scores

Figure 5.6.: *Attributed Modularity* scores w.r.t. the number of dimensions



(a) Absolute *Attributed Modularity* scores



(b) Relative *Attributed Modularity* scores

Figure 5.7.: *Attributed Modularity* scores w.r.t. the number of vertices

of the neighbourhood of one vertex to build some of the larger communities as it may have to move more than one vertex into another community to increase the score. Though *Coarse* is able to make movements on a higher level, it fails to pick the right movements and end very early in a local optima. The same behaviour could be seen when the number of communities is varied as shown in figure A.1.

As this part of the evaluation uses synthetic data, the clusters in the graph are known. Due to the additional information considered by the algorithm, it is very likely that these clusters are split up. To measure how good the hidden clusters were detected by the instantiations, the cluster purity, as defined by Aggarwal [Agg04], is used. To calculate purity, a cluster is assigned the label of a hidden cluster which is most frequent among its objects. The score is the fraction of correctly assigned objects in all clusters. Figure 5.8 shows how pure the clusters of the *Attributed Modularity* instantiations are. The results were generated using a set of graphs with an increasing number of vertices but a constant number of 40 clusters. As the number of clusters is constant the community size rises with the size of the graph and shall evaluate the behaviour of clustering objects w.r.t. the community size. Overall *LM* does nearly always respect the bounds of the synthetic clusters as its cluster are small and have a dense connectivity. All instantiations that include a *Merge* step do add some objects of a neighbouring cluster to larger communities.

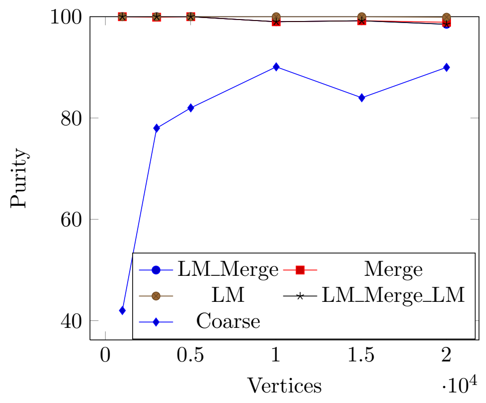


Figure 5.8.: Purity of the clusterings w.r.t. the number of vertices

Still the bounds of the synthetic clusters are respected as only about 1% of all objects are part of clusters that are mainly placed in another synthetic cluster. In contrast the *Coarse* instantiation clusters about 15% of all objects into a neighbouring cluster as it often falls into a local maxima that includes clusters that span between the “borders” of synthetic clusters.

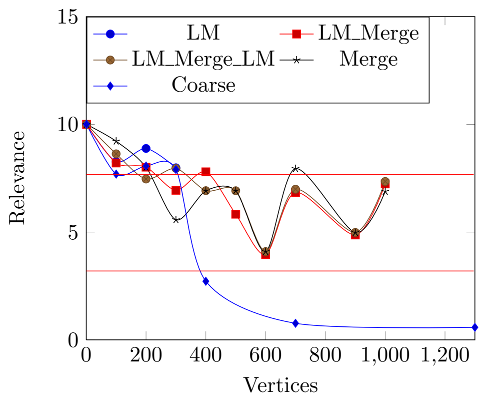


Figure 5.9.: average relevance by community size ($n = 10025$)

To measure if specific instantiations stick more to the attributes or to the graph structure, in figure 5.9 the relevance per community of a graph of size $n = 10025$ is listed. The graph has ten dimension so that the maximal relevance is 10, too. As the pre-generated clusters have on average about 5 relevant dimensions, in an optimal case, an algorithm finds large cluster in the area between the two red lines. A large cluster with a relevance of about 5 indicates that the algorithm has found the correct amount of dimensions that are characteristic for this cluster. The size is an important aspect as the formation of small clusters with high relevance is less complex than the formation of large clusters with the same relevance and a higher *Modularity* score. The three instantiations which include a *Merge* step fulfil this wanted property. In contrast *Coarse* trades relevance for cluster size and builds larger *Modularity* based cluster ignoring most attributes. The clusters generated by *LM* stay small but have a high relevance, i.e. *LM* fails to decrease in relevance so that it could create larger clusters with a higher *Modularity* score. The overall trend that the larger a cluster the lower its relevance is reasonable as small clusters only have few attribute values. Especially clusters of size one have a total relevance of 1 as only one vertex contributes to the attribute value, i.e. there is no scattering at all in the data.

5.2. Real World Data

In the previous section only data that was generated to fit a given model was considered. The utilised concepts of *Attributed Modularity* were modelled using properties discovered in real world data. As the real world cannot be described perfectly by a formula, the proposed algorithms are evaluated on different data sets to analyse their quality and discovered results.

5.2.1. Outlier Experiments

The data sets covered in this section are marked with labels that either have been generated through empirical study or by using a specific information already contained in the data as a label for an object being an outlier. These labels are only used for the calculation of the quality and are not considered by any of the algorithms. All the algorithm results listed have been run with less than five minutes of runtime for each. Only those instantiations that would have taken significantly longer are not listed.

The first experiment is done on a subgraph of the Amazon co-purchase Network [LAH07] covering the *Disney* category. This subgraph consists of 124 vertices and 334 edges. The outliers in this graph were labelled using high school students as domain experts. The students were given each subcategory and labelled suspect objects as outliers. Objects that were labelled by at least 50% of all students as an anamolie as considered as an outlier in the data. Table 5.10(a) shows the AUC score of the *Attributed Modularity* instantiations and the competitors which where used in the previous subchapter. As an example an outlier ranked in the top three of each of the *Attributed Modularity* instantiations is “Rudyard Kipling's The Jungle Book (1994)”. This film has a significant higher used price and a lower sales rank than all its neighbours. Its deviation in the attributes from its neighbours is higher than its structural integration so that is clustered in most cases as a size-one cluster.

Algorithm	**AUC**
LM	65.33
LM_Merge	75.46
LM_Merge_LM	74.01
Merge	**76.59**
Coarse	63.83
CODA	50.56
LOF	56.85
SCAN	52.68

(a) Disney scores

Algorithm	**AUC**
LM	**59.3**
LM_Merge	47.9
LM_Merge_LM	51.9
Merge	42.8
Coarse	43.3
CODA	53.35
LOF	**29.3**
SCAN	41.8

(b) Books scores

Algorithm	**AUC**
LM	56.1
LM_Merge	50
LM_Merge_LM	**77.8**
Merge	-
Coarse	-
CODA	45.71
LOF	61.1
SCAN	62.1

(c) Enron

As another category a subgraph out of the books section of Amazon is used. For this graph tags provided by the users are used for the labelling. A popular tag to show the disagreement with the manipulation of Amazon sales ranks utilised by users is *amazonfail*. If a product is tagged at least of 20 users with *amazonfail*, it is considered as an outlier in this data. On this dataset using *Attributed Modularity* outliers are not found as good as in the previous one but with *LM* an instantiation exists that has a remarkable better score than CODA. The two competitors SCAN and LOF rank the outliers more as inlier (as shown in table 5.10(b) than as a common inlier which makes the scores of them better than a random ranking. Still this behaviour is not wanted as a non-expert user may only look at the objects that are ranked high.

The last real world set with labelled outliers is data taken from the email communication inside of Enron whose scores are found in table 5.10(c). The objects marked as outliers are people which are considered as spammers in the communication. Due to the size of the graph having about 13500 vertices, the instantiations *Merge* and *Coarse* did not finish in a reasonable time. During *LM_Merge* the algorithm found a local optima describing 90% as one cluster. This cluster has a low but still positive score in *Modularity* and a high relevance in some attributes giving it an overall high *Attributed Modularity* score. The attributes that have high relevance include the average time between two mail and the average number of people listed in the CC. Because all objects in these attributes have the same attribute values the outlier scores are low so that there is no real indication if an object should be considered as an inlier or an outlier. Adding an extra *LM* afterwards detects the outliers hidden in this large cluster as it reviews each vertex on its local context.

5.2.2. DBLP

As the last data set, a subset of the DBLP graph is considered where vertices represent authors and edges the co-authorship relation between them. Each author is described by 46 numerical attributes which describe the publication ratio at major database, data mining, artificial intelligence, and statistics conferences. The largest connected component of this graph contains about 28000 vertices and was analysed using *Attributed Modularity*.

In the following, we describe some of the top 100 ranked authors from the results of the *LM_Merge* instantiation. A preliminary analysis shows that most of the top-ranked authors are professors. The common characteristic of these professors within the top-100 ranking is that they have highly number of publications in several research areas. In particular, the communities they belong to are mainly focused on a specific topic (e.g., databases or data mining). These top ranked authors have usually publications in other research areas as the result of a collaboration project or an intention to expand the research area of their department. Thus, they have publications in conferences where their communities do not publish, and show deviating values w.r.t. their community.

Jiawei Han (rank:1) and Jian Pei (rank: 37): They belong to the same community with 12 members. This community does not publish in specialised artificial intelligence and machine learning (e.g., ICML and AAAI/IAAI) conferences or workshops as their main topic is data mining. However, Jiawei Han has several publications in machine learning and artificial intelligence.

Hans Peter Kriegel(rank:95): The community of Hans Peter Kriegel, represented by 42 authors (e.g., Peer Krüger, Alvin Cheung, Markus M. Breunig), has a more specific profile for publishing than the previous community. It is characterised by not publishing neither in specialised conferences about web, information retrieval and artificial intelligence (e.g., AAAI/IAAAI, WWW, ECIR) nor in workshops. Thus, all authors in this community do not have any publications in these conferences and workshops. However, Hans Peter Kriegel has recently published on WWW and has few publications in workshops.

6. Conclusion

Summary

As part of this work a measure for the quality of clusterings for graphs with attributed vertices was developed that did not depend on any user-given parameters. The proposed measure scores a clustering based on the structural density and the selection of the characteristic attributes for a cluster. A cluster gets a higher score for its structural formation the higher the fraction of edges per vertex that are inside the cluster is than the fraction of edges that would be inside the cluster if the edges were distributed at random. On the attribute side a cluster will get a better score for each attribute that has a lower local than global variance. To get an overall high score both of these properties need to be balanced.

As not all objects in the graph fit into the patterns that are described by the clusters, these outliers should be ranked for e.g. inspection. Because outliers occur in different settings which are to the extreme either as a size-one community or embedded in a large community, the scoring function has to take care of this. The developed methods combine these settings to give a uniform outlier score for all objects. For the computation of the outlier scores the selected dimension and chosen clusters by the clustering process are used to respect the local attribute contexts.

To optimise the proposed score, an efficient algorithm scaffold that can be instantiated with three different optimising steps has been develop. One of the steps optimises the score by looking at the local context of each vertex and moving it to the best community within this context w.r.t. to the attributes and the structure. As local optima in the area of a single vertex are quickly found, a further approach contracts the graph so that each community forms a vertex. On this graph the previous step could be applied recursively. A third technique computes for each possible merge of two cluster the increase in *Attributed Modularity* and merges the best ones until no further increase is possible. All these instantiations of the algorithm run without any user-given parameter and their performance and quality have been shown in experiments on synthetic and real world data.

Future work

One of the theoretical problems of this work is to determine the maximal runtime of the outer iteration in the *LM* and *Coarse* steps. In the evaluation it was shown that these algorithms run in near linear time on the given data sets but no upper bound could be specified.

In the real world experiments, the *LM_Merge* instantiation produced a cluster which contained a large part of the vertices. This was caused by a number of attributes that only differed in a small set of vertices and did not distinguish the hidden clusters contained in the large cluster. A future approach could look at these clusters and try to build up a hierarchy of clusters using the large cluster as a first recursion step, Another concept could limit the merges done by a *Merge* step to specific properties so that not only merges with the highest increase are done at first. Some of the possible limits include that small clusters or cluster with high relevance should be merged first to prevent the creation of large cluster with low relevances.

In this work the clustering is for efficiency reasons a partition and overlapping clusters will be split up. As an addition to the existing algorithm, a scheme could be developed that enables the detection of overlapping cluster. An algorithm capable of detecting and handling hubs, i.e. objects that have high connectivity to two or more clusters and fit well into the attribute spaces of their neighbours, would cluster these objects in all its neighbouring communities which which share similar attribute values instead of assigning it to only one of these clusters.

Bibliography

[Agg04] C. C. Aggarwal, “A human-computer interactive method for projected clustering,” *Knowledge and Data Engineering, IEEE Transactions on*, vol. 16, no. 4, pp. 448–460, 2004.

[AGGR98] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan, *Automatic subspace clustering of high dimensional data for data mining applications.* ACM, 1998, vol. 27, no. 2.

[AWY$^+$99] C. Aggarwal, J. Wolf, P. Yu, C. Procopiuc, and J. Park, “Fast algorithms for projected clustering,” *ACM SIGMOD Record*, vol. 28, no. 2, pp. 61–72, 1999.

[BDG$^+$08] U. Brandes, D. Delling, M. Gaertler, R. Gorke, M. Hoefer, Z. Nikoloski, and D. Wagner, “On modularity clustering,” *Knowledge and Data Engineering, IEEE Transactions on*, vol. 20, no. 2, pp. 172–188, 2008.

[Bel57] R. Bellman, “Dynamic programming, princeton,” *NJ: Princeton UP*, 1957.

[BGLL08] V. Blondel, J. 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.

[BGRS99] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, “When is “nearest neighbor” meaningful?” *Database Theory—ICDT’99*, pp. 217–235, 1999.

[BKNS00] M. Breunig, H. Kriegel, R. Ng, and J. Sander, “Lof: identifying density-based local outliers,” in *ACM Sigmod Record*, vol. 29, no. 2. ACM, 2000, pp. 93–104.

[Cen13] U. S. Census Bureau, “Interactive timeline of u.s. census 2010,” Feburary 2013. [Online]. Available: http://www.census.gov/2010census/about/timeline-text.php

[CFZ99] C.-H. Cheng, A. W. Fu, and Y. Zhang, “Entropy-based subspace clustering for mining numerical data,” in *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, 1999, pp. 84–93.

[CGL83] T. Chan, G. Golub, and R. LeVeque, “Algorithms for computing the sample variance: analysis and recommendations,” *The American Statistician*, vol. 37, no. 3, pp. 242–247, 1983.

[DLMR11] M. Davis, W. Liu, P. Miller, and G. Redpath, “Detecting anomalies in graphs with numeric labels,” in *Proceedings of the 20th ACM international conference on Information and knowledge management.* ACM, 2011, pp. 1197–1202.

[EH07] W. Eberle and L. Holder, “Discovering structural anomalies in graph-based data,” in *Data Mining Workshops, 2007. ICDM Workshops 2007. Seventh IEEE International Conference on.* IEEE, 2007, pp. 393–398.

[EOZPP11] A. Elías, A. Ochoa-Zezzatti, A. Padilla, and J. Ponce, "Outlier analysis for plastic card fraud detection a hybridized and multi-objective approach," *Hybrid Artificial Intelligent Systems*, pp. 1–9, 2011.

[Fie73] M. Fiedler, "Algebraic connectivity of graphs," *Czechoslovak Mathematical Journal*, vol. 23, no. 2, pp. 298–305, 1973.

[GBS11] S. Günnemann, B. Boden, and T. Seidl, "Db-csc: A density-based approach for subspace clustering in graphs with feature vectors," *Machine Learning and Knowledge Discovery in Databases*, pp. 565–580, 2011.

[GLF$^+$10] J. Gao, F. Liang, W. Fan, C. Wang, Y. Sun, and J. Han, "On community outliers and their efficient detection in information networks," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, 2010, pp. 813–822.

[Haw80] D. Hawkins, *Identification of outliers.* Chapman and Hall London, 1980, vol. 11.

[Hol94] H. Hollerith, "The electrical tabulating machine," *Journal of the Royal Statistical Society*, pp. 678–689, 1894.

[KL70] B. Kernighan and S. Lin, "An eficient heuristic procedure for partitioning graphs," *Bell system technical journal*, 1970.

[KMB12] F. Keller, E. Müller, and K. Böhm, "Hics: high contrast subspaces for density-based outlier ranking," in *Data Engineering (ICDE), 2012 IEEE 28th International Conference on.* IEEE, 2012, pp. 1037–1048.

[KN98] E. M. Knox and R. T. Ng, "Algorithms for mining distance-based outliers in large datasets," in *Proceedings of the International Conference on Very Large Data Bases.* Citeseer, 1998.

[LAH07] J. Leskovec, L. A. Adamic, and B. A. Huberman, "The dynamics of viral marketing," *ACM Transactions on the Web (TWEB)*, vol. 1, no. 1, p. 5, 2007.

[LFR08] A. Lancichinetti, S. Fortunato, and F. Radicchi, "Benchmark graphs for testing community detection algorithms," *Physical Review E*, vol. 78, no. 4, p. 046110, 2008.

[LLDM08] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney, "Statistical properties of community structure in large social and information networks," in *Proceeding of the 17th international conference on World Wide Web.* ACM, 2008, pp. 695–704.

[LR87] A. M. Leroy and P. J. Rousseeuw, "Robust regression and outlier detection," *Wiley Series in Probability and Mathematical Statistics, New York: Wiley, 1987*, vol. 1, 1987.

[M$^+$67] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, no. 281-297. California, USA, 1967, p. 14.

[MCRE09] F. Moser, R. Colak, A. Rafiey, and M. Ester, "Mining cohesive patterns from graphs with feature vectors," in *SIAM Data Mining Conf.(SDM)*, 2009, pp. 593–604.

[NC03] C. C. Noble and D. J. Cook, "Graph-based anomaly detection," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, 2003, pp. 631–636.

[New04] M. E. Newman, "Fast algorithm for detecting community structure in networks," *Physical Review E*, vol. 69, no. 6, p. 066133, 2004.

[NR09] A. Noack and R. Rotta, "Multi-level algorithms for modularity clustering," *Experimental Algorithms*, pp. 257–268, 2009.

[Pea01] K. Pearson, "Liii. on lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901.

[Wes79] D. West, "Updating mean and variance estimates: An improved method," *Communications of the ACM*, vol. 22, no. 9, pp. 532–535, 1979.

[XYFS07] X. Xu, N. Yuruk, Z. Feng, and T. Schweiger, "Scan: a structural clustering algorithm for networks," in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, 2007, pp. 824–833.

[YC71] E. Youngs and E. Cramer, "Some results relevant to choice of sum and sum-of-product algorithms," *Technometrics*, vol. 13, no. 3, pp. 657–665, 1971.

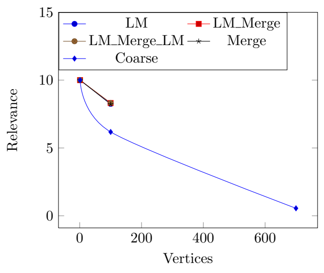
[YCN04] K. Yip, D. Cheung, and M. Ng, "Harp: A practical projected clustering algorithm," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 16, no. 11, pp. 1387–1397, 2004.

[ZCY09] Y. Zhou, H. Cheng, and J. X. Yu, "Graph clustering based on structural/attribute similarities," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 718–729, 2009.

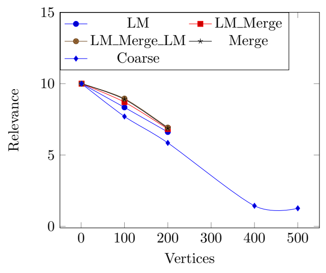
Appendix

A. Cluster Quality

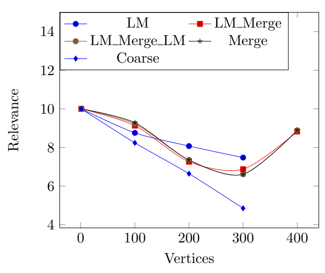
The diagrams listed here are part of the *Cluster Quality* evaluation and show how the algorithms perform in some scenarios. They are not used directly in the discussion in the related chapter but included to highlight some properties with different input data.



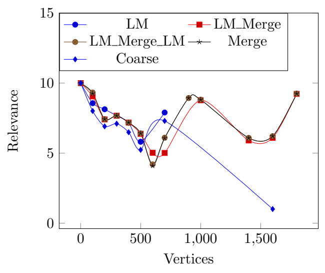
(d) average relevance by community size ($n = 999$)



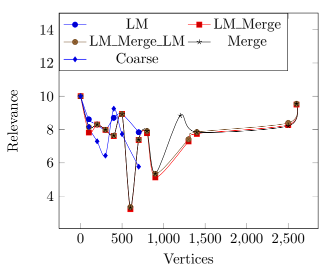
(e) average relevance by community size ($n = 3009$)



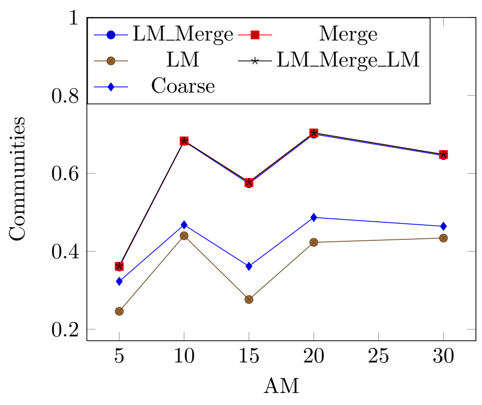
(f) average relevance by community size ($n = 5012$)



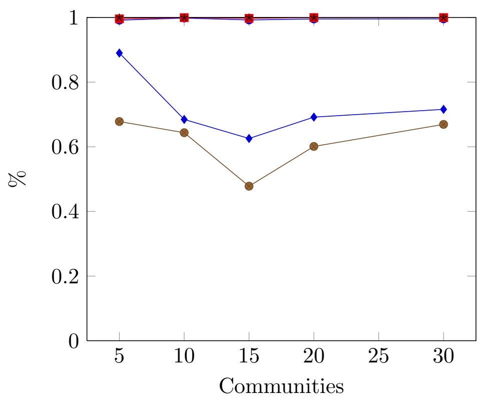
(g) average relevance by community size ($n = 15044$)



(h) average relevance by community size ($n = 20018$)



(i) Absoulte *Attributed Modularity* scores



(j) Relative *Attributed Modularity* scores

Figure A.1.: *Attributed Modularity* scores w.r.t. the number of communities