Weighted Graph Compression for Parameter-free Clustering With PaCCo

Nikola S. Mueller[*†] Katrin Haegler[*‡] Junming Shao[§] Claudia Plant[¶]
Christian Böhm[‖]

Abstract

Object similarities are now more and more characterized by connectivity information available in form of network or graph data. Complex graph data arises in various fields like e-commerce, social networks, high throughput biological analysis etc. The generated interaction information for objects is often not simply binary but rather associated with interaction strength which are in turn represented as edge weights in graphs. The identification of groups of highly connected nodes is an important task and results in valuable knowledge of the data set as a whole. Many popular clustering techniques are designed for vector or unweighted graph data, and can thus not be directly applied for weighted graphs. In this paper, we propose a novel clustering algorithm for weighted graphs, called *PaCCo* (*Pa*rameter-free *C*lustering by *Co*ding costs), which is based on the Minimum Description Length (MDL) principle in combination with a bisecting k-Means strategy. MDL relates the clustering problem to the problem of data compression: A good cluster structure on graphs enables strong graph compression. The compression efficiency depends on the underlying edges which constitute the graph connectivity. The compression rate serves as similarity or distance metric for nodes. The MDL principle ensures that our algorithm is parameter free (automatically finds the number of clusters) and avoids restrictive assumptions that no information on the data is required. We systematically evaluate our clustering approach PaCCo on synthetic as well as on real data to demonstrate the superiority of our developed algorithm over existing approaches.

Keywords: Clustering, Weighted Graph, Data Compression, Biological Network

[*]These authors contributed equally to this work.

[†]Max Planck Institute of Biochemistry, Germany. nimuell@biochem.mpg.de

[‡]University of Munich, Germany. katrin.haegler@med.uni-muenchen.de

[§]University of Munich, Germany. shao@dbs.ifi.lmu.de

[¶]Florida State University, USA. cplant@fsu.edu

[‖]University of Munich, Germany. boehm@dbs.ifi.lmu.de

1 Introduction

Large-scale technologies generate huge amounts of data on an every day basis. The first steps towards the understanding of underlying patterns is the identification of meaningful subgroups. In order to extract this meaningful information data can usually be subdivided into two or more partitions based on an object similarity measure without any information on how the data should be divided. This process is called clustering. Clustering is very complex when data is available in a network format: Objects to be clustered are nodes in the graphs which have node-to-node edges determining the similarity or linkage between one another.

Organism-wide protein-protein interaction networks or social connectivity in social networks are easily and often quickly obtained, but their interpretation is rather difficult. Beyond single node statistics on graphs, the quantification of object relationships represents a challenging task but is eventually able to reveal clusters of highly interlinked nodes that are groups of similar nodes. Graph partitioning splits the data set into non-overlapping meaningful subsets of nodes. The edges in the graph serve as node-to-node similarity information used by the clustering algorithm. Considering not only whether two nodes are connected or not, the strength of the connection (represented as edge weights) adds additional information to node similarities. The edge weight information does contribute to graph clusters and has to be separately handled by the clustering algorithm. If edge weights are neglected through binarization or thresholding, the true graph clusters cannot be revealed but at most roughly approximated.

To illustrate the desired result of a graph clustering, we introduce a simplified example in Figure 1. Weighted graph clustering starts with a rather confusing network (Fig. 1a) and reveals an underlying simpler graph structure (Fig. 1b). The nodes assigned to the same cluster are drawn in close proximity to better illustrate the clustering process. Nodes are colored with respect to synthetic cluster membership and edges with respect to their weights. Note, that the each final cluster contains edges with similar weights (low, medium and high edge weights for the three clusters). This emphasizes that the

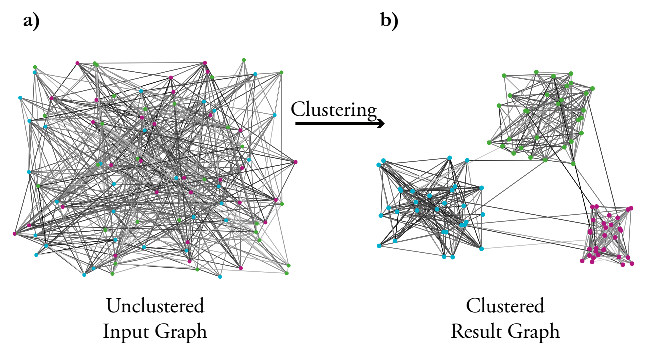

Figure 1: Weighted Graph Clustering Example. Clustering a weighted graph maximizes the number of edges inside each cluster while minimizing edges between clusters. Edge weights are colored in gray with respect to their weights (dark gray for high weights). The nodes are colored with respect to their synthetic cluster label. Graph layout visually grouped nodes with similar cluster membership.

clustering process does not only minimize the number of edges between clusters but it also maximizes weight similarity inside clusters.

Nodes belonging to one cluster are assumed to either share similar interest, e.g. mobile users in a social network, or similar function, e.g. proteins in a protein-protein network. With the help of clustering techniques the prediction of e.g. protein function of an unknown protein is feasible by identifying functionally homogeneous cluster based on known proteins. Here the continuous similarity information between proteins provides more biologically relevant information than simple "zero-or-one" interactions. Thus, the weighted clustering reveals functional classes which are e.g. biologically more reliable. With respect to social networks, based on calling behaviors from mobile phone companies, groups of common interest can be identified. For example people strongly interacting with a group of iPhone-users are more likely to buy an iPhone as well.

Current problems in graph mining algorithms [15] are the selection of parameters, scalability, and runtime. In addition the evaluation of graph clustering results is an important and difficult issue that also has to be addressed when introducing novel graph clustering algorithms. We address the problem of parameter settings by consistently using the principle of data compression for clustering without any *a priori* knowledge of the data. The idea of data compression is based on the identification of regularities in the data these regularities can then be used for efficiently compressing the data. The Minimum Description Length (MDL) principle is a method from information theory to measure regularities in data [14], consequently, more regular data can better compressed than irregular data. We employ an objective function for clustering solely based on graph compression.

We present a novel parameter-free and fully automatic weighted graph clustering approach, called PaCCo. It uses the data compression principle with MDL to efficiently cluster weighted graphs into meaningful subgraphs. By iteratively splitting clusters into subclusters until the splitting does not result in stronger graph compression, we can not only produce accurate results but also save computation time compared to existing graph clustering methods. Our major contributions are:

- *Parameter-free*: PaCCo does not require any user specific parameter settings (like e.g. number of clusters in the data or number of communities in a mobile phone network) in order to obtain meaningful clustering results.
- *Fully automatic*: Entirely automatic, PaCCo does not require any intervention of the user since we use MDL principle as abortion criterion.
- *Reduced runtime*: The top-down splitting approach of PaCCo saves computational time while keeping high clustering accuracy. PaCCo runtime is comparable to parameter-dependent methods.

The rest of the paper is structured as follows: Section 2 reviews some well-known graph clustering approaches. In Section 3 we present our proposed method PaCCo which searches for optimal clustering results regarding computation time and accuracy. In Section 4 we introduce known cluster evaluation measures that were used for evaluation. An extensive evaluation of PaCCo including method comparison and benchmarking of synthetic as well as real data is provided in Section 5 followed by the conclusion of the paper in Section 6.

2 Related Work

Although many graph clustering approaches are available today, only few are applicable to weighted graphs. In general, we focused on approaches which are not only well-known in the data mining field, but also in other communities, e.g. life science research. For detailed reviews on graph mining refer to [15, 7].

2.1 Spectral Clustering. Spectral clustering refers to a class of techniques which rely on the Eigenstructure of a similarity matrix in order to partition objects into disjoint clusters. It is a well-known partitioning technique for similarity matrices. The objective function minimizes the normalized cut commonly achieved by Eigenvector decompositions. The algorithm proposed by [13] detects arbitrarily shaped clusters by considering the clustering problem from a graph-theoretic perspective. A cluster is obtained by removing the weakest edges between highly connected subgraphs. Another algorithm is a learning method [9] to derive a new cost function based on a measure of error between a given partition and a solution of the spectral relaxation of a minimum normalized cut problem.

Similar to k-Means [6], the problem of most spectral clustering approaches is the choice of a suitable number of k clusters. In addition, they are sensitive to outliers, i.e. noise in the similarity matrix. To overcome the difficulty of selecting a suitable number of clusters, Zelnik-Manor and Perona [19] recently proposed a spectral clustering method which investigates the structure of the Eigenvectors to infer the number of clusters. For a detailed tutorial on spectral clustering refer to [18].

2.2 Markov Clustering. The Markov Cluster algorithm (MCL) is a popular algorithm used in life sciences for fast clustering of weighted graphs. MCL basically identifies high-flowing regions, which are the clusters, in a weighted graph [5]. The inflation parameter alters the steps used to separate weak and strong flow regions. Consequently, the inflation parameter determines the granularity of the resulting clusters, thus k.

2.3 Metis. Metis is a class of well-known multi-level partitioning techniques proposed by Karypis and Kumar [10, 12, 11]. For graph partitioning in general a sequence of successively smaller (coarser) graphs is constructed first and a bisection of the coarsest graph is computed. Then the bisection is successively brought to the next level of a finer graph, and at each level an iterative refinement algorithm such as Kernighan-Lin (KL) or Fiduccia-Mattheyses (FM) is used to further improve the bisection. A more robust overall multilevel paradigm was introduced [10] which presented a powerful graph coarsening scheme where even a good bisection of the coarsest graph is a good bisection of the original graph. It also allows the use of simplified variants of KL and FM to speed up the refinement without compromising the overall quality. Likewise to spectral clustering, a suitable number of k clusters has to be set for the algorithm.

2.4 MDL-based Clustering of Unweighted Graphs. Being not directly related to our problem definition of partitioning weighted graphs, the Cross-Association clustering algorithm has to be mentioned as well. Similar to our approach, Cross-Association finds groups in (bipartite) unweighted graphs by lossless compression with MDL in a parameter-free algorithm [3]. Although the algorithm is not explicitly designed for only bipartite graphs, the meaningful interpretation of off-diagonal clusters (where one object might be at the same time inside and outside one cluster, depending if rows or column groups are analyzed) is rather intriguing. However, the idea of using MDL for clustering served as inspiring example for our algorithm.

3 PaCCo

To cope with the short comings of current weighted graph clustering algorithms, we have developed a *Pa*rameter-free *C*lustering algorithm based on *Co*ding costs, short *PaCCo*. PaCCo clusters nodes in a weighted graph by combining a bisecting k-Means [6] strategy with the principle of data compression. In a top-down splitting approach PaCCo uses data compression not only to determine the number of clusters in a graph but also to assign each node in the graph to one cluster. As data compression principle we apply the Minimum Description Length (MDL) principle to evaluate the goodness of a clustering.

The data compression principle allows us to infer costs of a graph clustering. Thus, a good clustering of a graph G leads to a strong graph compression which is in turn equivalent to low coding costs. Consequently, the graph clustering of PaCCo has to minimize the costs of a graph partitioning $C = \{C_1, ..., C_k\}$ while the costs caused by the k clusters and the parameters required for storing the clustering model are defined as:

$$Model - Cost(G|C) = \sum_{l=1}^{k} c(C_l) + c(p).$$

The total model costs take not only the costs of each cluster $c(C_l)$ into account, but compression costs of the cluster model p. With p we correct for cluster model complexity depending on of the number of clusters.

PaCCo is able to find a graph clustering using solely an undirected weighted graph as input. Clusters to be identified are highly interlinked subgraphs with similar weights with minimal links between clusters. In general, PaCCo algorithm has two major steps which both minimize the model costs:

- Graph splitting and
- Graph clustering using k-PaCCo.

The graph splitting bisects any (sub)graph while the clustering step with k-PaCCo evolves the subgraph clusterings. In detail, k-PaCCo uses a k-Means strategy as a scaffold to assign the nodes to k clusters by

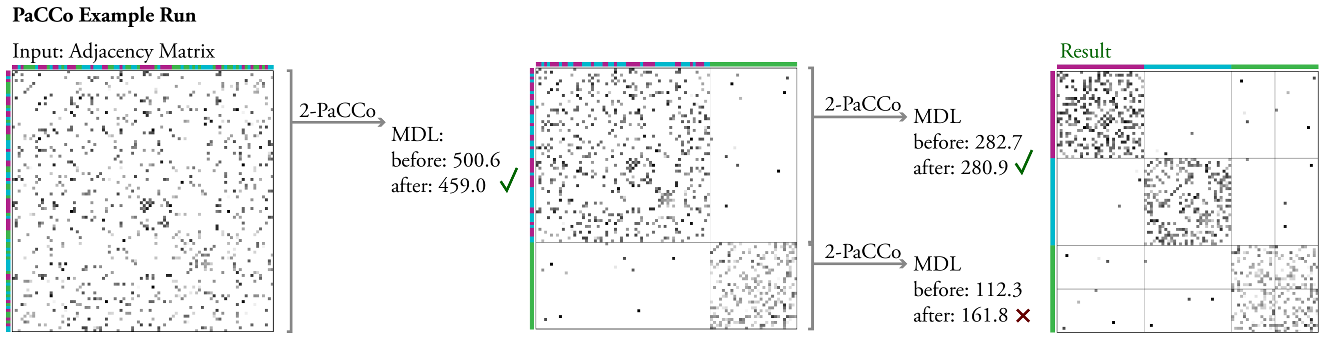

Figure 2: **PaCCo weighted graph clustering.** Only the adjacency matrix of a graph is used as input. The matrix contains continuous weight information as node-to-node similarities. Iteratively the subroutine, k-PaCCo, is called with k=2 to split the (sub-) graph. The MDL is calculated to determine the graph compression (coding costs). If the costs are lower after the k-PaCCo splitting, the split is accepted and subsequently split, otherwise rejected and kept as final cluster. Nodes are colored in pink, green and blue according to their synthetic cluster membership. Clusters are blocks along the diagonal and split lines are drawn where cluster membership changes.

minimizing the model costs. The model costs are furthermore used as the convergence criterion to stop the splitting process.

As a visual outline, Fig. 2 depicts a sample run of PaCCo. Within a top-down bisecting strategy, the graph (given as adjacency matrix) is iteratively split into two subgraphs if the split "pays off". Starting with the entire graph PaCCo tries to split the input graph using k-PaCCo with $k = 2$ until convergence. In the following we will refer to the k-PaCCo routine with $k = 2$ as 2-PaCCo. The lower the MDL coding costs, the better the data can be compressed, consequently the resulting clustering is a more accurate clustering of the graph. Thus, if the model costs are lower after the 2-PaCCo run, the (sub-)graph split is accepted. Each subgraph is subsequently handled separately, whereby, 2-PaCCo is once again applied to each of them. Finally, PaCCo converges if every already accepted cluster is cheaper than its split version. In the end, the adjacency matrix is restructured by equally reordering the rows and columns with respect to the clustering result. The algorithm will be formally defined in the following sections.

3.1 Graph and Cluster Notion

Undirected Graph. Let $G = (V, E)$ be an undirected weighted graph with a set of $n = |V|$ nodes and a set of $|E|$ edges, whereby, the undirected edge $e_{i,j} = e_{j,i}$ indicates a connection between the nodes v_i and v_j. Furthermore, let G be stored in the adjacency matrix $A = (a_{i,j})$ containing $n \times n$ entries of the form

$$a_{i,j} = \begin{cases} w_{i,j}, & \text{if } e_{i,j} \in E \\ 0, & \text{otherwise} \end{cases}$$

with $w_{i,j}$ being the weight of the edge $e_{i,j}$. The matrix A of the undirected graph G is, therefore, square and symmetric. On-diagonal entries are defined by 0, as we require no self-interaction information. Self-interactions of nodes will not alter the clustering result of PaCCo since they can, by definition, never be between two clusters.

Graph Clustering. Graph clustering is a partitioning of the graph into k disjoint clusters $C = \{C_1, \ldots, C_k\}$. A cluster C_l is a set of nodes $V_l = \{v_1, ..., v_m\}$ which subsequently describes a corresponding subgraph $G_l = (V_l, E_l)$, with $m = |C_l|$ being the number of nodes contained in the subgraph and $|E_l|$ being the number of edges between the nodes $\{v_i, v_j\} \in V_l$. The sub-adjacency matrix A_l has the dimensionality $m \times m$.

As a result of the cluster definition, clusters are located around the diagonal since a node cannot be part of two clusters at the same time. If the adjacency matrix is restructured, rows and columns are sorted in the same order.

3.2 PaCCo Algorithm. PaCCo begins the graph clustering with an undirected weighted graph and identifies the number of clusters without knowing the actual value of k. The algorithm follows a basic recursive concept as shown in Algorithm 1. The initial adjacency matrix A is used as single input to the algorithm. In order to get a first information on the entire input graph, we initialized the clustering with a k-PaCCo step using $k = 1$. Especially, information on the weight distribution of the entire graph offers a better-than-random cluster initialization. This step requires only one iteration until convergence (since the cluster membership never changes). With this first calculation we obtain an initialization of the top cluster which is better than random. The graph is subsequently bisected in a top-down manner whereby the final clustering result V is tracked. The final result inherently holds the number of k clusters in the graph.

Algorithm 1 PaCCo

```
Input: Adjacency Matrix $A$

$V$ = []; // Final Clustering

// initialize graph as one cluster
$C_{init} = k$-PaCCo ($k = 1$, $A$);

// run PaCCo
$V$ = splitCluster ($C_{init}$, $A$, $V$);

$k = |V|$ // number of clusters
return $V$;
```

3.3 k-PaCCo Clustering by Similarity. The core of PaCCo is the k-PaCCo routine which performs the clustering of an input graph into k non-empty clusters. During runtime of PaCCo, the k-PaCCo routine partitions (super-)clusters into $k = 2$ new clusters. For graph compression based on nodes we calculate the coding costs of each cluster $c(C_l)$ as a sum of the costs of each node v_i in the cluster

$$c(C_l) = \sum_{v_i \in C_l} c(v_i|C_l).$$

A node in a graph belongs to one cluster if it shares similarities to the other nodes in the cluster. With the k-PaCCo clustering we simultaneously maximize the number of edges as well as the weight similarities inside a cluster. In other words, we cluster highly interconnected nodes with similar edge weights. Thus, the costs of a node in a cluster C_l are determined by two factors: 1) the cluster weights of C_l and 2) the number of links (edges) inside a cluster while correcting for links to other clusters. We can formalize the weight costs and the linkage costs of each cluster separately:

$$c(v_i|C_l) = c_{weights}(v_i|C_l) + c_{linkage}(v_i|C_l).$$

A key feature of PaCCo is, that we build clusters sharing similar weight information. To actually code the weights in a cluster we introduce a novel concept of describing a cluster weight representative. We evaluate edge weight similarities to originate from a common probability density function (PDF). Without prior knowledge, PaCCo identifies the underlying cluster PDF which is adjusted to the data during runtime. A technique to compress a PDF is Huffman coding. The coding is defined as the inverse logarithm of an object's probability. Hence, this negative log-likelihood can be considered as the costs $c_{weights}$ for coding the weights of a node v_i in the cluster C_l given a PDF:

$$c_{weights}(v_i|C_l) = -log_2(f_{PDF}(v_i)).$$

We specify the approximation of the weights inside the subgraph of a cluster with a Gaussian PDF (GD). We chose to use a GD to get a rough approximation of the edge weights, since many natural processes already produce Gaussian data. Here, the assumption of a Gaussian model is not a severe restriction: The GD is only part of the codebook which is mainly used to compare several candidate clustering. Thus, best model selection for data compression does not restrict the data to follow exactly a GD. Although optimal compression cannot be achieved for non-Gaussian data with a Gaussian codebook, the model selection with the Gaussian codebook is nonetheless applicable for approximately symmetric data distributions. Note, that the PDF can easily be exchanged for another PDF if the weight distribution on the edge weights is known. Given the weighted edges in a graph interlinking the cluster nodes, a suitable cluster representative in the subgraph is introduced as a PDF $f_{GD}(w_{C_l})$ on the weights w_{C_l} of inter-cluster edges. We define our coding costs with respect to the GD where a cluster C_l has a characteristic cluster mean μ_{C_l} and a standard deviation σ_{C_l} of all weights in one cluster. Each node in a cluster can be compressed as

$$c_{weights}(v_i|C_l) = -log_2(f_{GD}(v_i; \mu_{C_l}, \sigma_{C_l}))$$

where f_{GD} for an existing edge weight $w_{i,j}$ is defined as

$$f_{GD}(w_{i,j}; \mu_{C_l}, \sigma_{C_l}) = \frac{1}{\sqrt{2\pi\sigma_{C_l}^2}} e^{-\frac{(w_{i,j} - \mu_{C_l})^2}{2\sigma_{C_l}^2}}.$$

The idea is to fit a node with its set of edges into the GD by determining the edge weights with respect to all nodes v_j in the cluster:

$$f_{GD}(v_i; \mu_{C_l}, \sigma_{C_l}) = \frac{1}{|C_l|} \sum_{\forall v_j \in C_l} \frac{1}{\sqrt{2\pi\sigma_{C_l}^2}} e^{-\frac{(w_{i,j} - \mu_{C_l})^2}{2\sigma_{C_l}^2}}.$$

In addition to the weight coding costs, the inner cluster connectivity $c_{linkage}$ has to be maximized in a cluster while connections to other clusters have to be punished. If the node v_i is assigned to a cluster C_l, it causes the following linkage costs which are determined by the edges to nodes of the cluster $v_{j'} \in C_l$, $\forall e_{i,j'} \in E$, and the node degree ($v_{j''} \in C$, $\forall e_{i,j''} \in E$) as

$$c_{linkage}(v_i|C_l) = -.5log_2(|e_{i,j'}|) + .5log_2(|e_{i,j''}|).$$

We directly compressed the number of edges a node has to one cluster as well as to the other clusters, in order to obtain a clustering which balances the intra- and inter-cluster edges. Thereby, the existing number of links of a node to a cluster is maximized, since we correct for the number of links to the cluster C_l with the number of total edges the node v_i has.

Taking a step back, we can integrate the objective function of model costs into the k-Means strategy to cluster a graph (Alg. 2). Thereby two steps have to be formalized:

Reassignment step. The reassignment step requires the minimization of the objective function for model costs. The idea is to explicitly maximize the connectivity and similarity of each cluster, while implicitly taking care of minimizing the connectivity and similarity between the clusters. In other words, the reassignment of a node v_i to the best fitting cluster C_{new} is determined by

$$C_{new}(v_i) = \min_{C_l \in C} c(v_i|C_l)$$

Note, that we minimize the costs of a node which is equivalent to a better graph compression.

Update step. The update step has to explicitly adjust to weight distribution per cluster while the number of links are already precomputed during the reassignment step. The update of the weight distribution in each cluster is achieved by updating the mean μ_{C_l} and standard deviation σ_{C_l} of the node costs with respect to all weights $w_{i,j}$ entirely enclosed in a cluster:

$$\mu_{C_l} = \frac{\sum w_{i,j}}{n}, \ \forall v_i, v_j \in C_l, \ v_i \neq v_j$$

Accordingly for σ_{C_l}.

The objective function is always minimized or kept equal during the reassignment as well as the update step.

3.4 PaCCo Splitting Strategy. PaCCo performs recursive splitting of a graph in a top-down approach (Alg. 3). To perform the split of a (sub)graph we have the option to either randomly assign the nodes to a new cluster or direct the splitting to be better than random and save runtime. For non-random splitting the edges and their edge weights can be used. We use a heuristic to drive the separation of the cluster weights. Since we already have information on the GD spanning of all weights of the cluster, the two subclusters are initialized by shifting the GD one standard deviation up and one down on the weight distribution spectrum. Figure 3 depicts the subcluster initialization. In other words, the current Cluster C_l to be evaluated, might still subsume at least two real clusters. Thus, the current GD of the cluster weights is torn apart on the weight spectrum. Assuming the initial GD subsumes two clusters with separate weights, we initialize the subclustering with

Algorithm 2 k-PaCCo

```
Input: $k$, Adjacency Matrix $A$, $\mu$, $\sigma$

Clusters $C = []$;
randomly assign $o_i$ to a Cluster PDF

iter = iteration counter
while cluster assignment changes & iter < maxIteration do
    for all Objects $o_i \in A$ do
        assign $o_i$ to cluster by $c(v_i|C_l)$
    end for
    for all $C_l \in C$ do
        update cluster representative $\mu_{C_l}$, $\sigma_{C_l}$
    end for
end while
```

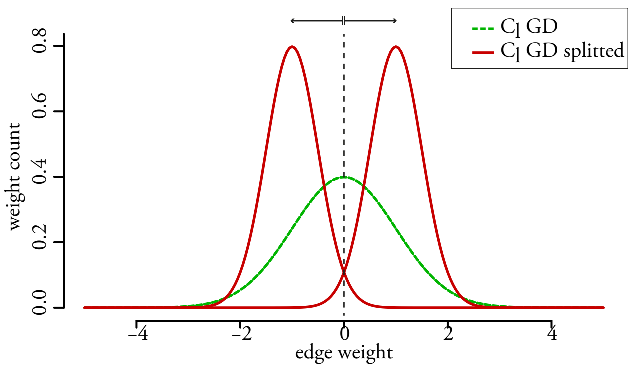

Figure 3: Initialization of a cluster split for k-PaCCo given a current cluster C_l GD. Thus, the approximation of the weights inside C_l are torn apart to obtain a guess of the possible underlying data which is better than a random guess.

the prior knowledge of the current cluster instead of a random initialization of the two new GD during a cluster bisection step.

The top-down iterative splitting is performed until the cost of a cluster is cheaper than itself being split. As a lower convergence criterion the algorithm splits the graph into singletons. To evaluate if (sub)graph splitting ($Model - Cost(G_{l_{split}}|C_{l_{split}})$) results in a stronger compression than the (sub)graph ($Model - Cost(G_l|C_l)$) we split the graph only if the model costs are minimized as

$$Model - Cost(G_{l_{split}}|C_{l_{split}}) < Model - Cost(G_l|C_l).$$

Note, that the parameter costs $c(p)$ of a clustering have to be considered. In PaCCo, we have to carry along the costs for saving all μ and σ of each cluster (thus the parameter costs p directly depend on the number of clusters k). We code both parameters with floating point precision. Since we only bisect each cluster separately, we only have to account to an additional GD to be compressed, thus, raising the costs.

Algorithm 3 splitClusters

```
Input: Graph Partitioning $C$, Adjacency Matrix $A$,
Final Clustering $V$

for all $C_l \in C$ do
  if $k < size(A_l)$ OR $1 < |C_l|$ then

    // Prepare two PDFs for cluster bisection
    $\mu' = [\mu_{C_l} + \sigma_{C_l}; \mu_{C_l} - \sigma_{C_l}]$;
    $\sigma' = [\sigma_{C_l}/2; \sigma_{C_l}/2]$;

    $C_{l_{split}} = k$-PaCCo ($k = 2$, $A_l$; $\mu', \sigma'$);

    if Model-Cost($G_{l_{split}}|C_{l_{split}}$) $\geq$ Model-Cost($G_l|C_l$) then
      // Present Cluster $C_l$ is already good
      $V = V \cup C_l$;
    else
      // Bisection of Cluster "pays off"
      splitClusters($C_{l_{split}}$, $A_l$, $V$);
    end if
  else
    $V = V \cup C_l$;
  end if
end for

return $V$
```

4 Graph Cluster Evaluation Measures

The evaluation of graph clustering algorithms can basically be performed in two ways: If class label information is available for each node, the clustering performance can be directly measured. However, for real data this information is often not available. For those graphs, a measure for cluster connectivity is required.

4.1 With Node Class Labels. In the case of the existence of node class labels a simple calculation of precision or accuracy is not possible for graph clustering, since the cluster IDs are interchangeable. Therefore, we compute equivalent measures based on information theoretic measures. Hence, we decided to choose the best of the four measures presented in [17] for clustering comparison, namely the adjusted mutual information (AMI). AMI has fixed range allowing a direct comparison of different approaches. In contrast to the normalized mutual information (NMI) [16], AMI is corrected for chance. It scales between 0 and 1 for a random or a perfect clustering result, respectively.

4.2 Without Node Class Labels. Many graph clustering algorithms are based on the principle of modularity [8]. We use modularity as a cost function if no class label information is available. The modularity value is higher if links inside a cluster are maximized while the links between clusters are minimized. As a common definition for this principle, modularity optimizes for the inner-cluster connectivity above a random. Formally, modularity on a graph partitioning of an unweighted graph into k clusters is defined as

$$M = \sum_{c=1}^{k} \left(\frac{l_c}{L} - \left(\frac{d_c}{2L} \right)^2 \right)$$

with L being the number of all edges inside the graph, l_c being the number of links enclosed by the cluster c, and d_c being the sum of the degree of the nodes of cluster c.

For weighted graphs, however, the modularity formula itself will not compute the real graph modularity, but rather a rough approximation. Thus, we can reformulate modularity for a weighted graph, by considering the edge weights instead of their counts. We define weighted modularity as

$$M = \sum_{c=1}^{k} \left(\frac{w_c}{W} - \left(\frac{d_{w_c}}{2W} \right)^2 \right)$$

with the sum of all weights W, the weighted edges w_c inside the cluster c, and d_{w_c} being the weighted degree of each node inside a cluster.

5 Experiments

We conducted multiple experiments to evaluate the performance and accuracy of our novel algorithm PaCCo.

In our extensive evaluation we compare the results of PaCCo to three other existing weighted graph clustering approaches. Two of the three comparative methods are parameter dependent; both algorithms require a parameter which influences the number of clusters to be obtained. (1) We use the multilevel partitioning technique Metis by Karypis and Kumar [11] which requires the number of clusters k as parameter. (2) The Markov Cluster algorithm (MCL) by Dongen [5] requires an inflation parameter which directly influences the granularity of the clustering, thus, the number of clusters. In addition one of the three comparative methods is parameter-free. (3) The spectral clustering approach by Zelnik-Manor and Perona [19] is parameter-free (in the following named SpectralZM). With regard to the parameter dependent methods, we sampled the free parameter for each experiment separately and always used the best performing result. All experiments were performed on a 2.9 GHz Windows computer with 3 GB RAM. PaCCo was implemented in Java.

The experiments will demonstrate that PaCCo outperforms the three methods in most settings while being faster than parameter-free SpectralZM and comparable to the parameter dependent methods Metis and MCL. We generated several synthetic weighted graphs varying the number of noise edges added to the weighted graphs, the spacing between the means of the cluster distributions, and the number of clusters k. Since we generated the graph cluster by cluster, we have information on the class label of each node which we used for benchmarking of the algorithms. For evaluating the cluster outputs we used the adjusted mutual information (AMI) measure, due to the fact that class labels were present for the synthetic data. AMI measures the agreement between two clustering results based on entropy while adjusting for chance. We calculate the AMI for each clustering result with respect to the real clustering formed by the class labels. AMI value of 1 and 0 correspond to a perfect cluster agreement and a clustering agreement expected by chance, respectively.

As real world example we used a weighted undirected protein network of a protein interaction screen by Costanzo et al [4] which was evaluated using the modularity measure as no class labels were present. In addition, we evaluated the clustering result with respect to biological enrichment.

5.1 Synthetic Data Description.

PaCCo was designed to compress the graph weights with a Gaussian distribution. To demonstrate that other weight distri-

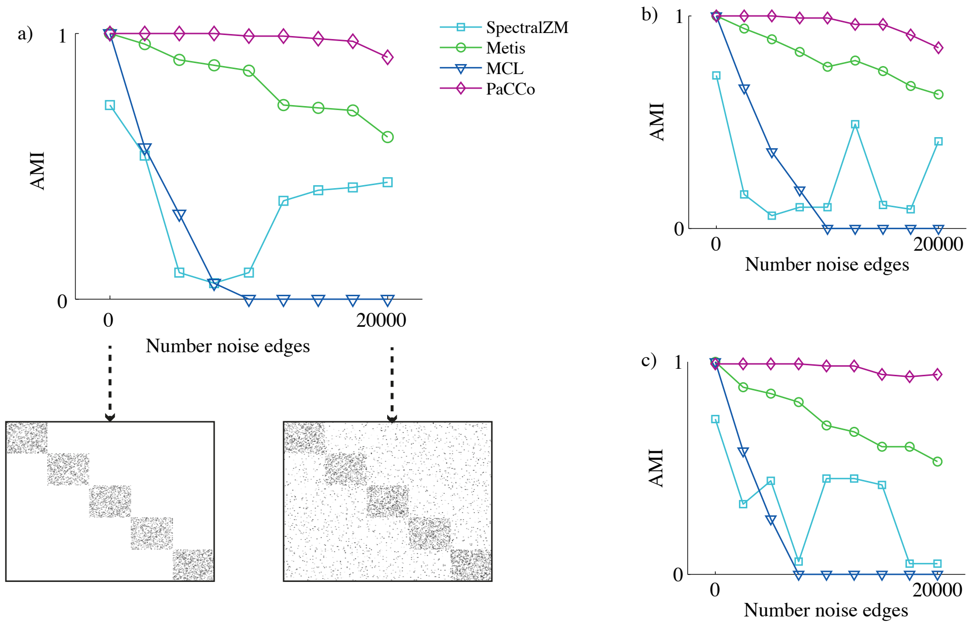

Figure 4: Varying the number of inter cluster edges (noise edges) in the data. The number of noise edges are added to the graph in addition to the existing edges. The weighted graph has 1000 nodes (k=20 clusters with each 50 nodes) and has (without noise) 70 % of intra cluster edges (17 000 intra cluster edges). Means and standard deviation of the cluster distributions were set to 1. Adjacency matrices in a) exemplify the range of graph used for experiments. Underlying cluster distributions were a) Gaussian, b) Uniform, and c) Laplacian.

butions can be compressed equally well, we used three underlying distributions. These underlying distributions for the edge weights in the synthetic graphs were either Gaussian, Uniform, or Laplacian. As a result each experiment was executed three times, once using Gaussian distributions, once with uniformly distributed cluster distributions, and once using Laplacian distributions. The default number of nodes per cluster for all synthetic experiments was set to 50. Each cluster of 50 nodes was randomly interlinked with 70 % intra cluster edges. After a synthetic graph was generated the nodes were randomly shuffled to enhance complexity in the clustering process. If not specified elsewhere, we generated $k = 20$ clusters, which corresponds to a total amount of 1,000 nodes and around 17,000 edges in the weighted graph. We performed three experiments on synthetic data to benchmark various graph characteristics:

First, we evaluated how well the graph clustering algorithms can handle additional edges in the graph, which we call noise edges. In addition to the approximately 17,000 intra-cluster edges, the number of noise edges, which are additional edges randomly added to random nodes, present in the data was varied from 0 to 20,000. The noise in the data was represented by inter cluster edges being added to the data, thus, introducing inter-cluster connectivity to hamper cluster separation. The number of clusters was kept constant at $k = 20$, the means of all cluster distributions were separated by 1, and the standard deviation of all cluster distributions was chosen to be 1.

Second, cluster weights' intervals were varied, having a constant cluster value of $k = 20$ with additional 5,000 inter-cluster edges. Starting with all 20 means of the cluster distributions around a mean of 1, they were gradually spread out until the means of the cluster distributions were separated by 1; As a result the lowest cluster distribution mean was 1 and the highest cluster distribution mean was 20. Thus, we altered the numerical spaces between the cluster weights.

Finally, the number of clusters k was varied from 10 to 100, each containing 50 nodes, leading to a maximum of 5,000 nodes, while approximately 70 % of the intra cluster edges (i.e. ca. 9,000 to 90,000 intra cluster

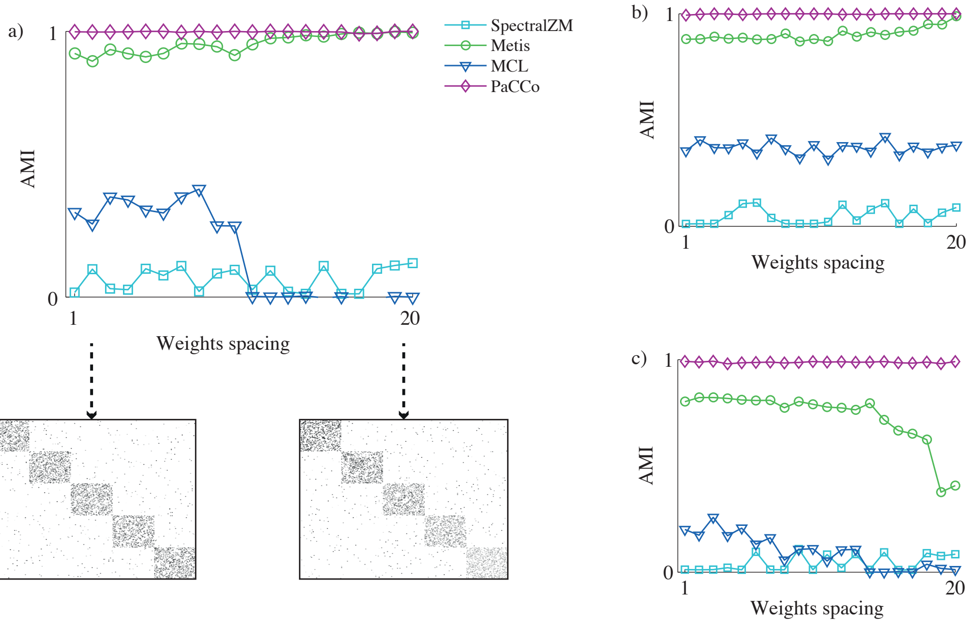

Figure 5: Varying the cluster weights' spacing intervals. The underlying cluster distributions were a) Gaussian, b) Uniform, and c) Laplacian. The number of clusters was set to $k = 20$ with 70 % of intra cluster edges (approximately 17,000 intra cluster edges) being connected, and ca. 5,000 inter cluster edges considered as noise being present in the data. One on the x axis indicated that all means of the cluster distributions had a value of one, while 20 on the x axis indicates that cluster distribution means were all different being separated by one unit each. Therefore, the lowest cluster distribution mean was 1 and the largest cluster distribution mean had a value of 20. Adjacency matrices in a) exemplify the range of graph used for experiments. Underlying cluster distributions were a) Gaussian, b) Uniform, and c) Laplacian.

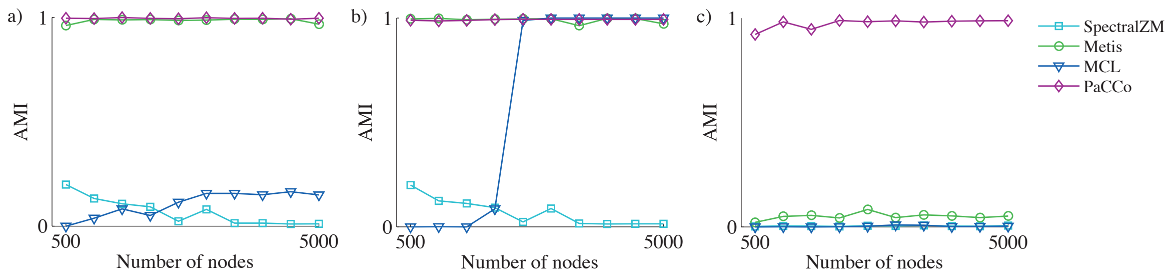

Figure 6: Varying the number of clusters k from 10 to 100, each containing 50 nodes. The underlying cluster distributions were a) Gaussian, b) Uniform, and c) Laplacian. Approximately 70 % of the intra cluster edges were connected (i.e. ca. 9,000 to 90,000 intra cluster edges) and additionally 30 % of these intra cluster edges were added as inter cluster edges being considered as noise (i.e. ca. 3,000 to 30,000 inter cluster edges). The mean and standard deviation of all cluster distributions was set to a value of 1.

edges) were connected and 30 % of these intra cluster edges were added as inter cluster edges (i.e. ca. 3,000 to 30,000 inter cluster edges). The mean and standard deviation of all cluster distributions were set to 1.

5.2 Performance on Synthetic Data. For Metis and MCL we selected the parameters as follows: The number of clusters required for Metis was given in all synthetic data sets, thus was directly set k to the given value; the inflation parameter required for MCL was set to the default value of 1.4 for all synthetic experiments as this parameter also achieved the best results.

We began by adding noise to the synthetic weighted graph (Fig. 4). The more edge noise we added to the graph, all four approaches resulted in a decrease of their clustering performance, as measured by the information theoretic measure AMI. MCL was only able to handle data with up to 10 000 inter cluster edges independent of the underlying distributions. As soon as noise was added the performance started to decrease. SpectralZM had large performance fluctuations while processing the noise data for all three data distributions. Even in the data set containing no noise it was not able to achieve an optimal clustering. Metis performed slightly worse on the Laplacian data set than on the Gaussian and Uniform data, having a constant decrease with increased noise. PaCCo was the only algorithm which was able to achieve better results than the other three methods for increased noise; even for the largest number of noise edges PaCCo outperformed the other three graph clustering methods.

Next, we evaluated how the algorithms respond to a change of the cluster weights (Fig. 5). MCL and SpectralZM performed poorly with AMI indices between 0 and 0.5. Metis increased performance when the cluster weights were clearly separated than with all weights being equal for each cluster for the Gaussian and the Uniform distributed data. On the Laplacian data set Metis also achieved overall poor results like MCL and SpectralZM. Gradually changing the spacing of the cluster weights means, PaCCo achieved the best overall results for all three cluster distributions showing the highest benefit for the Laplacian data set. This result demonstrates that PaCCo is able to perform better on weighted graphs than all other approaches independent on the underlying data distribution.

Finally, we varied the number of clusters k (Fig. 6) what should be a trivial task for each algorithm. Metis achieved convincing results for the Gaussian and the uniform distributed cluster distributions but showed no satisfactory results for the Laplacian data set. MCL was only able to perform well for larger data sets with uniformly distributed data. In all other cases it obtained poor results. SpectralZM was not able to achieve convincing results in any of the given data sets. In contrast to all other methods our parameter-free approach PaCCo achieved equally good results, independent of the number of clusters for the Gaussian, the uniform, and the Laplacian distributed data. Note, that this version of PaCCo can only handle data sets which can be fully loaded as matrix into the virtual memory similar to SpectralZM.

To conclude, we demonstrated that the parameter-free algorithm PaCCo outperforms the other methods. MCL was not able to handle noise and, in addition, requires the setting of an inflation parameter. Metis is not designed to handle increasing noise and additionally requires the number of clusters in the data which for real data is rarely known. SpectralZM showed difficulties with respect to noise present in the data as well as with respect to data size.

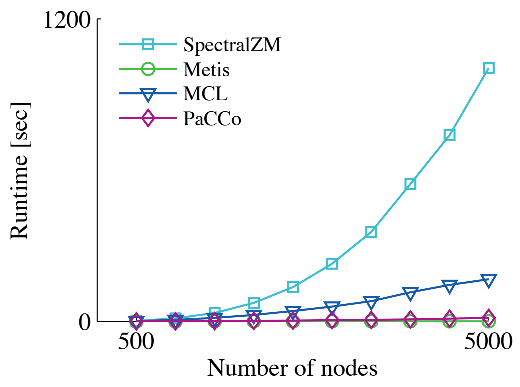

Figure 7: The runtime of the parameter-free methods PaCCo and Spectral, as well as the runtime of Metis and MCL.

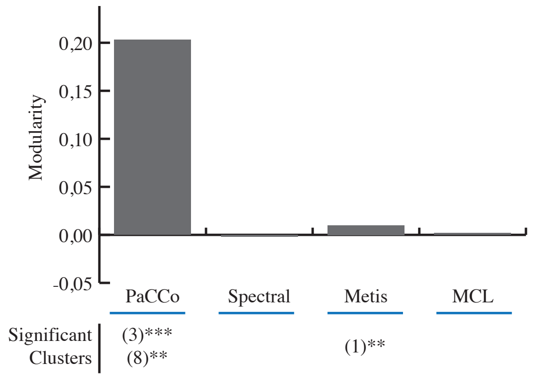

Figure 8: Performance on protein data set. The best clustering result of each algorithm shown as a bar graph. We measured performance by modularity. In addition, if the best clustering also enriched a molecular function, we denoted the number of the cluster e.g. (3) with its significance level (**$p < .05$, ***$p < .01$).

5.3 Runtime. For runtime comparisons we varied again the number of clusters k from 10 to 100, while each cluster contained 50 nodes. Approximately 70 % of the intra cluster edges were connected and no inter cluster edges were present. The mean and standard deviation of all cluster distributions were set to 1. In order to obtain accurate runtime results each data set was processed 10 times by each method, subsequently building a mean of the 10 rounds.

The runtime of one execution of each algorithm is tracked. Importantly, for the parameter dependent methods Metis and MCL we first had to sample for optimal parameter setting before actually tracking the execution time of one run. We did not account for this time effort here. Due to the fact that, to our knowledge, the approach by Zelnik-Manor and Perona is the only existing weighted graph clustering algorithm without requiring parameters like our approach, thus the runtimes of SpectralZM and PaCCo were directly comparable. The runtime of PaCCo and SpectralZM (Fig. 7) clearly shows that PaCCo is faster than SpectralZM. While SpectralZM has a time complexity of O(n^3) due to the eigenvalue decomposition, PaCCo's time complexity is only super-linear. For example, having 5 000 nodes in a graph SpectralZM requires 16.8 minutes to obtain a clustering result while PaCCo only needs 14.1 seconds. Thus, PaCCo is approximately 70 times faster than SpectralZM. PaCCo was even faster than the parameter dependent approach MCL, while being slightly slower than Metis.

5.4 Real World Data. We evaluated the clustering result of PaCCo on a real data set generated by high-throughput biology. Biologists are able to determine whether two genes of an organism are genetically interacting. In that sense, the deletion of one gene from the organism has no effect on the fitness of an organism, but the deletion of an additional gene results in a significant fitness defect. The so called double knockout may be either lethal to the organism (called "synthetic lethal"), or in contrast result in increased fitness, thus, stronger growth. Note, that two proteins which are synthetic lethal are supposed to function in two parallel pathways, where one can compensate for the loss of the other. In such a synthetic lethal screen [4], two yeast genes were simultaneously deleted while the increased (positive) or decreased (negative) colony growth is read out. If two proteins show an altered growth rate of 0.15 a weighted edge in the graph containing 1139 nodes (the proteins) is inserted in the network. We applied PaCCo and SpectralZM without parameter setting while Metis and MCL were sampled for k and inflation. Figure 8 depicts the graph clustering details for the best run of each algorithm. Metis performed best for $k = 3$ and MCL for the default inflation parameter of 1.4 (resulting in k=1121). PaCCo and SpectralZM automatically identified 11 and 3 clusters. The number of clusters found by SpectralZM and MCL were extreme: SpectralZM generated 3 clusters whereas MCL generated only singleton clusters except for one.

Protein interaction network clusters can be biologically evaluated with the help of the gene ontology (GO) database [1]. GO contains functional annotations of proteins, thus we can calculate statistical enrichment of GO molecular functions. With the hypergeometric probability, statistically significant functions for each non-singleton cluster can be identified similarly to [2]. Only with PaCCo the graph clustering result was enriched for two molecular functions, whereby, Metis was enriched for one of the two enrichments found by PaCCo. SpectralZM and MCL clustering did not generate any significant results. PaCCo and Metis were both enriched for *hydrolase activity*, suggesting that synthetic lethality is more likely to be part of two parallel functional pathways. Proteins of the hydrolase activity class catalyze an essential chemical reaction called hydrolysis during which water molecules are split. The PaCCo clustering is able to enrich for hydrolase activity even better than Metis (Fig. 8, PaCCo cluster 3 and Metis cluster 1). Interestingly, PaCCo was able to reveal another cluster enriched for *isomerase activity*, not identified by any other algorithm. Isomerase proteins take care of structural arrangements of isomers. Isomers are proteins which are structurally different while their molecular formula stays constant. This essential process may even inhibit or enable proper protein function.

For the experimental data PaCCo was able to find a strong clustering result regarding the modularity measure, in contrast to all other approaches which did not succeed in providing comparable results. Moreover, PaCCo outperformed the other algorithms with regard to biologically meaningful clusters.

6 Conclusion

We propose PaCCo – a parameter-free clustering approach for weighted graphs. PaCCo couples a bisecting k-Means strategy with a graph compression principle being an efficient and accurate graph clustering technique. Since PaCCo is parameter-free and fully automatic, it can be easily applied to real weighted graphs without requiring any parameters like the number of subgroups present in the data or available evaluation criteria. Moreover, our clustering results do not suffer from fast runtime. PaCCo can support the analysis of weighted graphs, such as protein-protein interaction networks, by revealing interesting and relevant clusters.

References

[1] M. Ashburner, C. A. Ball, J. A. Blake, D. Botstein, H. Butler, J. M. Cherry, A. P. Davis, K. Dolinski, S. S. Dwight, J. T. Eppig, and et al., *Gene ontology: tool for the unification of biology. the gene ontology consortium*, Nat. Genet. 25 (2000), 25–29.

[2] S. Brohee, K. Faust, G. Lima-Mendez, G. Vanderstocken, and J. van Helden, *Network analysis tools: from biological networks to clusters and pathways*, Nat Protoc 3 (2008), 1616–1629.

[3] D. Chakrabarti, S. Papadimitriou, D. S. Modha, and C. Faloutsos, *Fully automatic cross-associations*, KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining (New York, NY, USA), 2004, pp. 79–88.

[4] M. Costanzo, A. Baryshnikova, J. Bellay, Y. Kim, E. D. Spear, C. S. Sevier, H. Ding, J. L. Y. Koh, K. Toufighi, S. Mostafavi, and et al., *The genetic landscape of a cell*, Science 327 (2010), no. 5964, 425–431.

[5] S. Dongen, *A cluster algorithm for graphs*, Tech. report, CWI (Centre for Mathematics and Computer Science), Amsterdam, The Netherlands, 2000.

[6] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern classification (2nd edition)*, Wiley-Interscience, 2000.

[7] S. Fortunato, *Community detection in graphs*, Physics Reports (2010).

[8] M. Girvan and M. E. J. Newman, *Community structure in social and biological networks*, Proceedings of the National Academy of Sciences of the United States of America 99 (2002), no. 12, 7821–7826.

[9] M. I. Jordan and F. R. Bach, *Learning spectral clustering*, Advances in Neural Information Processing Systems 16, 2003.

[10] G. Karypis and V. Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM Journal on Scientific Computing 20 (1998), 359–392.

[11] G. Karypis and V. Kumar, *Metis: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. version 4.0*, Tech. report, Dept. of Computer Science, University of Minnesota, 1998.

[12] G. Karypis and V. Kumar, *Multilevel k-way partitioning scheme for irregular graphs*, Journal of Parallel and Distributed Computing 48 (1998), 96 – 129.

[13] A. Y. Ng, M. I. Jordan, and Y. Weiss, *On spectral clustering: Analysis and an algorithm*, Advances in Neural Information Processing Systems 14, 2001, pp. 849 – 856.

[14] J. Rissanen, *A universal prior for integers and estimation by minimum description length*, The Annals of Statistics 11 (1983), no. 2, 416–431.

[15] S. E. Schaeffer, *Graph clustering*, Computer Science Review 1 (2007), no. 1, 27 – 64.

[16] A. Strehl and J. Ghosh, *Cluster ensembles — a knowledge reuse framework for combining multiple partitions*, J. Mach. Learn. Res. 3 (2003), 583–617.

[17] N. X. Vinh, J. Epps, and J. Bailey, *Information theoretic measures for clusterings comparison: is a correction for chance necessary?*, ICML '09: Proceedings of the 26th Annual International Conference on Machine Learning (New York, NY, USA), 2009, pp. 1073–1080.

[18] U. von Luxburg, *A tutorial on spectral clustering*, Statistics and Computing 17 (2007), no. 4, 395–416.

[19] L. Zelnik-Manor and P. Perona, *Self-tuning spectral clustering*, Advances in Neural Information Processing Systems, vol. 17, 2004, pp. 1601–1608.