Single- and Multi-level Network Sparsification by Algebraic Distance

Emmanuel John[*]   Ilya Safro[†]

January 20, 2016

Abstract

Network sparsification methods play an important role in modern network analysis when fast estimation of computationally expensive properties (such as the diameter, centrality indices, and paths) is required. We propose a method of network sparsification that preserves a wide range of structural properties. Depending on the analysis goals, the method allows to distinguish between local and global range edges that can be filtered out during the sparsification. First we rank edges by their algebraic distances and then we sample them. We also introduce a multilevel framework for sparsification that can be used to control the sparsification process at various coarse-grained resolutions. Based primarily on the matrix-vector multiplications, our method is easily parallelized for different architectures.

Keywords: Networks, Sparsification, Multilevel Methods, Software, Scalable Algorithm

1 Introduction

Networks are an abstract model of the relationships between discrete objects. Examples include networks of genes, consumers and generators in the power grid, and networks of friendships or followers in social communities. In order to study real world networks, they are often represented as graphs, where the vertices represent the objects and edges model the relationship or interaction between them. Modeling networks this way facilitates the analysis and understanding of many different structural properties of the underlying complex system. Several powerful software packages such as SNAP [23], Pajek[11], NetworkIt [39], NetworkX [13], and Gephi [4] have been developed to provide this capability. However, many complex networks are massive in size. For example, Facebook users post about 3.2 billion likes and comments each day [1], Twitter has more than 190 million users and about 65 million tweets are posted each day [43], and the human gene network contain several million edges [29]. Although, modeling and understanding these networks is very important in many application domains, the massive size of the network makes it often impractical to perform network analysis on the entire dataset.

In sparsification methods, we aim to select a representative sample of the corresponding graph such that some properties of the original graph are preserved. In other words, central to sparsification is the idea that if an algorithm depends on or computes the properties that are preserved in the sparsified graph, we can expect that the results will be similar for the original graph [15] while the algorithm will perform much faster on the sparsified graph. Sampling is broadly being carried

[*]School of Computing, Clemson University, Clemson, SC `emmanuj@g.clemson.edu`
[†]School of Computing, Clemson University, Clemson, SC `isafro@clemson.edu`

out in real world networks. Most network analytics consider just a sample in time of the networks under study which is usually the result of data collection limitations [1]. Thus, it is important to understand and develop *scalable* methods for sampling massive networks.

There are several motivating examples for network sparsification. One obvious example is in the domain of visualization. It is often computationally intensive to render huge graphs on a computer screen as well it is hard to visually analyze such graphs. Sparsification helps to visualize a sample of the graph that reveals structural properties that would have been difficult to visualize and visually analyze in the original graph [21, 35]. The computational difficulty of visualization often arises from its objective, which requires solving a computational optimization problem [16, 2]. Another broad application is the reduction in the cost of computational network analysis. In computing the betweenness centrality of every node in a massive network, for example, by prioritizing what edges should be retained and what should be removed, it is possible to improve the running time of the algorithms at a very minimal cost in optimality [3]. Thirdly, graph sparsification can be applied to revealing hidden populations which are hard for researchers to find by just looking at the entire population. For example, Salganik et. al showed that when trying to sample the population of injection drug dealers, it is difficult to sample directly as this population is hidden and so specialized sampling algorithms are needed [34]. Methods applied usually involve starting out with a sample of the desired population and using that as a seed for revealing the other members of the sample population [15]. Existing methods include snowball sampling [14, 12] and respondent driven sampling [34]. In addition, in the case where there is an incomplete data, sampling can be used to estimate properties of the original graph. This is particularly useful in dynamic graphs [40], graph streaming algorithms [1] and collective classification [33].

There are several approaches to sampling a large graph while preserving the desired properties. An example involves formulating a mathematical programming problem to minimize the distance between the sparse graph and the original graph [15]. However, such approaches are often quite complex and running them might be costlier than running the algorithm on the larger graph. Spectral approximation algorithms also exist [38]. However, those algorithms are not very fast as well and often infeasible for large graphs [15] as they often involve hidden constants and require convergence in eigen-problems. The more common approaches are (1) vertex sampling, which involves selecting a number of vertices from the original graph and retaining the vertex-induced subgraph, and (2) edge sampling, which involves the selection of edges and corresponding edge-induced subgraph. Other variations of edge and vertex sampling have been developed (see [15] for a full survey). In our method we focus on the edge sampling and also preserve the nodes from the original graph. In order to achieve this, we ensure that every node has at least one incident edge in the sparsified graph.

1.1 Strength of Connectivity in Sparsification

If the properties to be preserved are known beforehand, then, in many cases, it is possible to determine what kind of edges are important to preserve those properties and which ones are redundant. Thus, the sampling transformation can then be designed with the objective of retaining those edges. A general framework for sparsification involves: (1) ranking the edges and assigning each edge an edge score; and (2) sampling edges based on their scores [15]. Scoring edges provides a motivation for rating the strength of connection between two vertices. In particular, this is extremely important in weighted networks, where the weights can be approximate, noisy or even completely missing. Different types of the connection strength have been proposed for scoring edges. We refer

the reader to [24] for a brief survey on the sparsification-relevant types of connectivity strength. The most relevant to our work is a cohort of spectral methods widely used in theoretical computer science to sparsify dense graphs such that some spectral properties are preserved. These are usually cut-based properties that are formulated using Cheeger inequality. For example, Spielman et al. introduced the edge effective resistance [36]. The effective resistance is computed using the linear system solver [37] which runs in $O(mlog^{15}n)$ time which can be time consuming to be feasible. Another example is the vectorized PageRank [10]. Various interpretations of the diffusion have been proposed and analyzed [19, 41] for graph kernels. However, these methods usually suffer from impractical complexity.

Another relevant class of methods is based on the Jaccard index in which a similarity between two vertices is measured by computing the overlap in their neighborhoods. In [35], Satuluri et. al rated edges according to the local similarity $sim(i, j) = |N_i \cap N_j|/|N_i \cup N_j|$, where N_i is the neighborhood of node i. This method was designed for clustering objectives assuming that nodes with larger shared neighborhoods are likely to belong to the same cluster. A global similarity threshold is then chosen for which edges are filtered. The authors also introduced a method for local sparsification in which they rate and filter edges per node by selecting the top d_i^e edges ranked by their similarity score, where $e \in (0, 1)$. Their method ensures that there is at least one edge per node after sparsification. We explore this property in our method. This sparsification technique can be computationally expensive since it requires counting the number of triangles an edge is a part of. The authors, however, provided an approximation for computing the similarity. Based on the work of Satuluri et al. [35], local degree method favors the retention of high degree nodes - also known as hub nodes [24]. As in the local similarity, for each node, they include edges to the top d_i^e nodes. However, edges are sorted according to the degree of their neighbors in descending order. The main idea of this method is to keep edges in sparsified graph that leads to nodes with high degree. Additionally, vertex connectivity can be measured by the betweenness centrality, the shortest path length, the weight of substructures (such as spanning rooted forests, routes, overlapping paths that connect two vertices [8]) and algebraic distance [9] which we will discuss in Section 3.

1.2 Our Contribution

We introduce two methods for complex network sparsification that distinguish between strong and weak connectivity through neighborhoods of limited distance from the endpoints of edges. In some networks (such as those that include geospatial information), these types of connections can be interpreted as long- and short-range connections while in other (such as social networks) as inner- and outer-community connections. In both methods the sampling is based on the connectivity measured by the algebraic distance between nodes [9]. It generalizes the idea of methods that estimate the Jaccard coefficient for more distant neighborhoods through limited application of lazy random-walks (also known as algebraic distance [9]). In the first method (the single level approach) we demonstrate multiple settings of filtering local and global connections with the sampling similar to [35]. In the second method we propose a multilevel algorithm that combines the single level approach with the multilevel framework [28] to sparsify graphs at different coarse-grained resolutions. We provide a robust method that can be tuned to preserve different network properties that are important in a variety of applications. The multi- and single level methods can both be used to either preserve the global structure or the local structure. We also discuss how our method can be parallelized and show the ruining time in OpenMP implementation. Evaluation of methods is demonstrated through comparison of several network properties with those measured on the

original network. The proposed methods are implemented and available at [18].

2 Preliminaries

We denote the graph underlying a given network by $G = (V, E, w)$, where V is a set of vertices, E is a set of edges, and $w : E \to \mathbb{R}_{\geq 0}$ is a weighting function on E that represents the strength of connectivity between two vertices. The graph is undirected, containing no self-loops and multi-edges. For each node $i \in V$ we define its degree by d_i and its neighbors by N_i. The clustering coefficient is a measure of the probability that neighbors of a node are connected to each other [27]. Consequently, it is a measure of the degree to which nodes in a network tend to cluster [43]. The clustering coefficient of a node i is defined as $c_i = \lambda_i / \tau_i$, where λ_i is the number of triangle subgraphs i participates in, and $\tau_i = d_i(d_i - 1)/2$, i.e., the number of triples. The clustering coefficient of a graph G is defined as

$$C_G = \frac{1}{|V'|} \sum_{i \in V'} c_i, \tag{1}$$

where $V' = \{i \in V \mid d_i > 1\}$. The diameter of a graph is defined as the maximum distance shortest path among all pairs of vertices in G from the same connected component. The resulting sparsified networks are compared with the original network using following properties: degree distribution, clustering coefficient, number of connected components[1], diameter, betweenness centrality, PageRank centrality, and modularity [27]. We will use the Spearman Rank Correlation Coefficient (ρ) that is a measure of the correlation between two distributions. It is defined as $\rho = 1 - (6 \sum p_i^2)/(n(n^2 - 1))$, where $p_i = x_i - y_i$, and x_i and y_i are the ranks computed from the scores X_i and Y_i.

3 Algebraic distance

In order to determine the strength of connection of edges for the purpose of sparsification, we use the algebraic distance introduced in [28, 9]. The algebraic distance of an edge ij (denoted by δ_{ij}) is interpreted as locally converged iterative process that propagates the weighted average of values from N_i and N_j initialized by random numbers [9]. This expresses the strength of connectivity between two nodes through their local neighborhoods. The process is essentially a Jacobi over-relaxation (JOR) or a lazy random walk with limited number of steps (see Algorithm 1). The algebraic distance was successfully used in several algebraic multigrid algorithms [25, 7] and in multilevel algorithms for discrete optimization on graphs (such as the minimum linear arrangement [28], and graph partitioning [32]) to reduce the order of interpolation that results in a sparsified coarse system.

Other stationary iterative relaxations can also be applied in a similar setting but since JOR is implicitly parallelizable using matrix-vector multiplications, we prefer to use it instead of other relaxations (such as Gauss-Seidel) that converge faster. Optionally, the algebraic distance can also be normalized by the square-root of the product of the weighted degrees of the two nodes to reduce extremely high strength of connection between hub nodes.

---

[1]In many existing sparsification methods, the number of connected components is preserved "artificially", i.e., even if the edge is marked for deletion, it is not deleted if it increases the number of connected components. Here we do not restrict our algorithms with such requirement.

Algorithm 1 Algebraic distance implementation: ComputeAlgDist

---

1: **Input:** Parameter α (in our experiments $\alpha = 1/2$)
2: $\forall ij \in E \ R_{ij} = 0$
3: **for** $r = 0, 1, 2, ...$ **do** $\triangleright$ the number of test vectors r is small
4:     $\forall i \in V \ x_i^{(0)} \leftarrow rand(-0.5, 0.5)$
5:     **for** $k = 0, 1, 2, ...$ **do** $\triangleright$ the number of JOR iterations k is small
6:         $\forall i \in V \ x_i^{(k)} \leftarrow \alpha x_i^{(k-1)} + (1-\alpha)\dfrac{\sum_{j \in N_i} w_{ij} x_j^{(k-1)}}{\sum_{j \in N_i} w_{ij}}$
7:     **end for**
8:     Rescale x back to $(-0.5, 0.5)$
9:     $\forall ij \in E \ R_{ij} = R_{ij} + (x_i - x_j)^2$
10: **end for**
11: **return** $\forall ij \in E \ \delta_{ij} \leftarrow \dfrac{1}{\sqrt{R_{ij} + \epsilon}}$ $\triangleright$ ϵ is sufficiently small
12: $\forall ij \in E \ \delta_{ij} \leftarrow \dfrac{\delta_{ij}}{\sqrt{d_i * d_j}}$ $\triangleright$ optional normalization

---

The algebraic distance will serve as the main criterion for choosing edges for sparsification in the algorithms below. Because it helps to distinguish between so called short- and long-range connections [9], we will use it to demonstrate different types of sparsification in which local and global properties are preserved correspondingly to the types of algebraic distances that we choose. The short-range connections (large values of δ_{ij}) will be called δ-strong. The long-range connections (small values of δ_{ij}) will be called δ-weak.

4 Single-level sparsification

In the single-level approach we demonstrate three types of sparsification in which we filter δ-weak, δ-strong edges and their mixture. In all of these cases, first, for each edge in the graph, we compute the algebraic distance. Then, for each node i, we sample the top d_i^e neighbors ranked by their algebraic distances, where $e \in [0, 1]$. In this approach it is possible to sample for local or global structure preservation or a combination of both. To preserve the global structure, we select d_i^e weakest connections and add them to the sparse graph (see Figure 1c). Similarly, d_i^e strongest connections are preserved to emphasize the importance of a local structure in the sparse graphs (see Figure 1b).
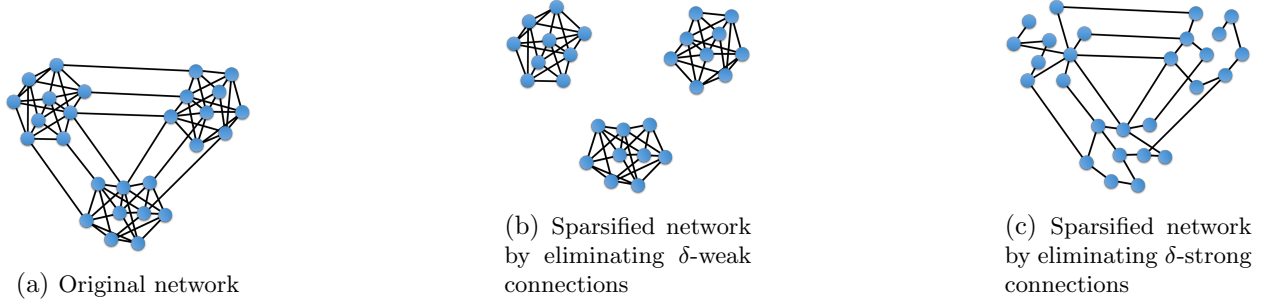
5

(a) Original network

(b) Sparsified network by eliminating δ-weak connections

(c) Sparsified network by eliminating δ-strong connections

Figure 1: An example of a small network with 3 dense clusters and sparse cuts between them (a). Sparsification of δ-weak connections will result in network presented in (b). Sparsification of δ-strong connections is presented in (c).

Algorithm 2 Single-level sparsification: Sparsify(G)

1: **Input:** Sparsification parameter e, Graph G
2: **Output:** Sparsified graph G_{sparse}
3: $G_{sparse} \leftarrow$ empty graph
4: COMPUTEALGEBRAICDISTANCES(G)
5: **for** $i \in V$ **do**
6:     Sort N_i by δ_{ij} in ascending (or descending) order
7:     Add top d_i^e edges to G_{sparse}
8: **end for**
9: **return** G_{sparse}

It is also possible to partially preserve both global and local structures with a slight change in the algorithm, namely, by distributing the algebraic distances into bins, and sampling the edges from all bins. In order, to distribute the algebraic distances into bins, we define the bin width $h = \frac{3.5*\sigma}{\sqrt[3]{d_i}}$, and the number of bins $k = (\max \delta_{ij} - \min \delta_{ij})/h$, where σ is the standard deviation of algebraic distances.

Algorithm 3 Single-level sparsification with binning: SparsifyB(G)

1: **Input:** Sparsification parameter e, Graph G
2: **Output:** Sparsified graph G_{sparse}
3: $G_{sparse} \leftarrow$ empty graph
4: COMPUTEALGEBRAICDISTANCES(G)
5: **for** $i \in V$ **do**
6:     Distribute N_i into bins, each bin corresponds to edges ...
7:     Randomly select bins and edges up to d_i^e to G_{sparse}
8: **end for**
9: **return** G_{sparse}

---

5 Multilevel sparsification

The multilevel approach [6, 42] can be applied as a general framework for many different numerical methods. Most real-world instances are not completely random, i.e., a particular similarity or dependence between variables exists and, thus, can partially be detected to reduce their number in complex computations. Here we introduce and advocate the use of multilevel approach as a general purpose framework for network sparsification. In the heart of the proposed method lies an idea to sparsify the network at multiple scales of coarseness which, in contrast to most existing sparsification methods that sample single edges, will allow to sample clusters of edges of different sizes and δ-weakness.

It is known that the topology of many complex networks is hierarchical (or multiscale) and, thus, often might be self-dissimilar across scales [17, 44, 5, 26]. In such hierarchical representations, groups of nodes are aggregated into communities, which automatically bundles edges into coarse connections. Bundling the edges at different scales of coarseness will introduce different levels of δ-weakness for such coarse connections which may or may not be required to be sparsified for the required analysis. For example, in the analysis of a social network, we may want to visualize only a certain type of edges that connect dense communities of small sizes, while connections between large communities and local inner connections are out of the scope. In the proposed framework, this can be achieved by creating a hierarchy of coarse representations, and sparsifying at those levels that do not correspond to the desired communities. To create a multilevel framework we use the algebraic multigrid (AMG) aggregation strategy that was introduced in [31]. For simplicity, we do not split fine nodes across the aggregates (like in some optimization problems [31, 32]) but instead cover the graph with star-like structures and coarsen them. For the completeness of paper we briefly repeat the main components of the coarsening algorithm.

Given an original graph G, in the multilevel framework we recursively construct a hierarchy of decreasing size coarse graphs $G_0 = G, G_1, ..., G_l$. The original graph is gradually coarsened into the smaller graphs until the small enough graph G_l is reached. The sparsification algorithm is then run on the coarsest level and the results (i.e., edges to eliminate for sparsification) are inherited by the finer graph and the uncoarsening continues until G_0 is reached. In most cases, our discussion is focused on fine-to-coarse and coarse-to-fine transformations of graphs and solutions, respectively. For this purpose, we denote the fine and coarse level graphs by $G_f = (V_f, E_f)$, and $G_c = (V_c, E_c)$, respectively. At each level, after sparsifying edges inherited from G_c, Algorithm 1 is applied to recompute algebraic distance on G_f.

The Coarsening We begin with selecting a dominating set of seed nodes $C \subset V_f$ that will serve as centers of future coarse nodes in V_c. Setting initially $F = V_f$ and $C = \emptyset$, the selection is done by traversal of F and moving to C such nodes that are not strongly coupled to those that are already in C. At each step $F \cup C = V_f$ is preserved, and at the end the size of V_c is known, namely, $|V_c| = |C|$. After C is selected, nodes in $F = V \setminus C$ are distributed to their aggregates according to the restriction operator $P \in \{0, 1\}^{|V_f| \times |C|}$, where

$$
P_{iJ} = \begin{cases} 1 & \text{if } i \in F, \ J = I_c\left(\text{argmax}_{j \in C} \dfrac{\delta_{ij}}{\sum\limits_{k \in C} \delta_{ik}}\right) \\ 1 & \text{if } i \in C, \ J = I_c(i) \\ 0 & \text{otherwise,} \end{cases} \tag{2}
$$

7

Algorithm 4 Multilevel sparsification of graph: MLSparsify

1: **Input:** fine graph $G_f = (V_f, E_f)$, vector of sparsification ratios
2: **Output:** sparse graph $G_f^s = (V_f, E_f^s)$
3: **function** $\mathrm{ML}(G_f)$
4:     $\mathrm{COMPUTEALGDIST}(G_f)$
5:     **if** $|V_f|$ is small enough **then**
6:         $E_f^s \leftarrow \mathrm{SPARSIFYB}(G_f)$ ▷ Sparsify coarse edges
7:     **else**
8:         $\mathrm{CREATESEEDS}(G_f)$ ▷ Coarsening: seeds
9:         Compute P ▷ Coarsening: restriction operator
10:       $G_c \leftarrow (L_c = P^T L_f P)$ ▷ Coarsening: coarse graph
11:       $G_c^s \leftarrow \mathrm{MLSPARSIFY}(G_c)$ ▷ Recursive call to sparsify the next coarser level
12:       $G_f^s \leftarrow \mathrm{UNCOARSEN}(G_c^s)$ ▷ Sparsification of edges inherited from coarse level
13:       $\mathrm{COMPUTEALGDIST}(G_f^s)$ ▷ Algebraic distances are recomputed
14:       $G_f^s \leftarrow \mathrm{SPARSIFYB}(G_f^s)$ ▷ Sparsification of current level edges
15:     **end if**
16: **end function**
17: **return** G_f^s

---

and $I_c(j)$ returns an index of coarse node J that corresponds to $j \in C$. Then, the Galerkin coarsening creates a coarse graph Laplacian $L_c = P^T L_f P$, where L_f is the Laplacian of G_f.

Coarsest Level   At the coarsest level, we sparsify the edges by using the single-level Algorithm (3). These edges correspond to bundles of edge chains at the fine levels that connect the most distant regions in a graph, so if the goal is to preserve the global structure, the user should avoid of sparsification at deep coarse levels.

Uncoarsening   We initialize the solution (sparsification) of G_f by uncoarsening the edges sparsified in G_c. When the order of interpolation in the multilevel algorithm equals 1 (i.e., there is only one non-zero entry per row in P, see Eq. 2), each coarse edge $IJ \in E_c$ can bundle at most two types of edges in E_f, namely, at most one edge that connect two seeds $I_c^{-1}(I)$ and $I_c^{-1}(J)$, and possibly multiple edges $pq \in E_f$ such that $P_{pI_c^{-1}(I)} = 1$, and $P_{qI_c^{-1}(J)} = 1$. If IJ is sparsified at the coarse level, then edges of both types are sparsified at the fine level. After initialization of the fine level, we recompute algebraic distances to update the information about connectivity in the sparsified fine graph, and, then, more edges may or may not be sparsified at the fine level depending on the parameter settings. Full multilevel cycle is presented in Algorithm 4. Example of full multilevel cycle on a Facebook network (see fb-uf in Table 1) is shown in Figure 2.

6   Computational Results

Implementation and Evaluation   We provide C++ implementation for both the single- and multilevel algorithms in [18]. For the comparison of original and sparsified networks, we employed methods implemented in NetworKit [39]. We experimented with varying degrees of sparsification, taking values of e ranging from 0.1 to 0.9 (see Section 4). All numerical properties for the comparison
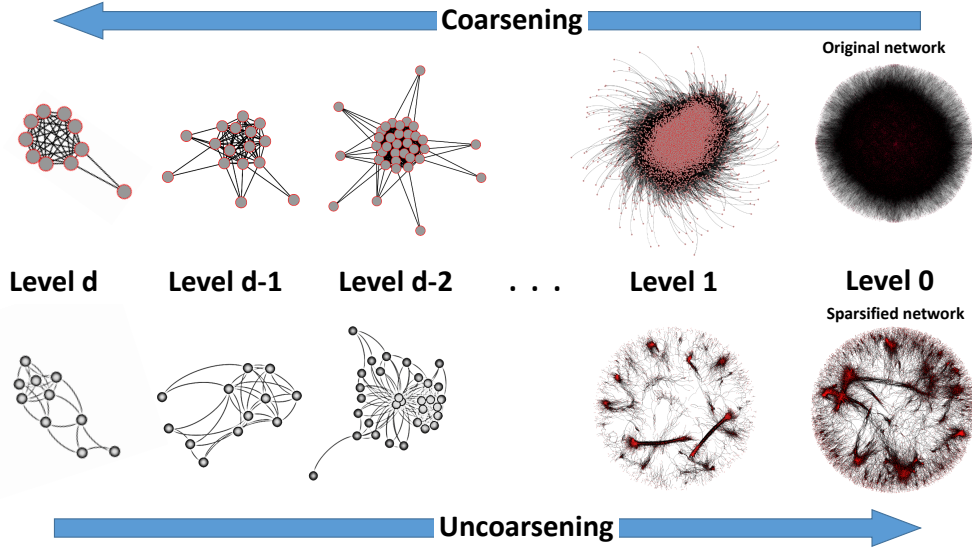
Figure 2: Complete Sparsification V-Cycle

are the averages over 10 runs with different random seeds for each parameter setting. The following parameter were used in computation of algebraic distance: $R = 10$, $k = 40$, and $\alpha = 0.5$. Their robustness is discussed in [9]. In addition, for the single-level algorithm (3), we provide two sets of results for each graph, namely, with and without the normalization (see last step in Algorithm 1) of algebraic distance. In each case we experimented with sparsification of weak edges, strong edges and mixture of both.

In the multilevel algorithm (4), we experimented with sparsifying at the coarsest, middle and the finest levels. In our experiments, we split the number of levels in the multilevel algorithm into 3 equal segments, and choose a parameter, level-span which determine how many levels in each segment gets sparsified. We then sparsify one segment at a time and observe the corresponding network properties. For example, for a graph with 6 levels, with a level-span of 2, to sparsify the coarsest levels only we use the following parameter configuration: $(0.3, 0.3, -1, -1, -1, -1)$, where a setting of -1 indicates no sparsification occurs at this level. Similarly, the middle and finest levels can be sparsified using $(-1, -1, 0.3, 0.3, -1, -1)$, and $(-1, -1, -1, -1, 0.3, 0.3)$ configuration settings respectively. However, in our implementation, users can specify any combination of settings for different levels. The sparisification ratio (ratio of number edges in the sparse graph to the number of edges in the original graph), is kept between 20% to 40% for each stage in order to make the results comparable. For the purpose of our study, we maintain a level span of 3.

Datasets We experiment with 18 real-world networks (see Table 1), which for the purpose of our study we grouped into two groups of social networks, one group of citation networks (CIT) and one group of biological networks (BIO). We split the social networks into 2 groups (SN1, and SN2) - one consisting of Facebook networks, Livejournal and Google+ (general purpose social networks), and the other consisting other consisting of Flickr, Buzznet, Foursquare, Catster, Blogcatalog and Livemocha. The graphs were retrieved from the NetworkRepository [30], the Koblenz [20], and the SNAP [22] collections. The size of the networks range between 1 million to 34 million edges.

9

Table 1: Benchmark graphs

Group	Graph	$|V|$	$|E|$	min deg.	max deg.	avg degree
Social	fb-indiana	29.7K	1.3M	1	1.4K	87
Networks 1	fb-texas84	36.4K	1,6M	1	6.3K	87
(SN1)	fb-uf	35.1K	1.5M	1	8.2K	83
	fb-penn94	41.5K	14M	1	4.4K	65
	livejournal	4M	27.9M	1	2.7K	13
	google-plus	107.6K	12.2M	1	20.1K	227.4
Biological	human-gene1	22K	12M	1	7.9K	1.1K
Networks	human-gene2	14K	9M	1	7.2K	1.3K
(BIO)	Mouse	43K	14.5M	1	8K	670
Social	flickr	105K	2.3M	7	5.4K	43.7
Networks 2	buzznet	101.2K	2.8M	1	64.3K	54
(SN2)	foursquare	639K	3.2M	1	106.2	10
	catster	149.7K	5.4M	1	80.6K	72
	blogcatalog	88.8K	2.1M	1	9.4K	47
	livemocha	104.1K	2.2M	1	3K	42
Citation	ca-cit-Hepth	22.9K	2.6M	1	11.9K	233.38
Networks	cit-patent	3.7M	16.5M	1	793	8.75
(CIT)	codblp	540.5K	15.2M	1	3.3K	56

Methods of Comparison   We studied various levels of sparsification while comparing the following properties of the sparse graph G_s, to those in the original graph G_o. The single value properties are: (a) **Diameter** - We measure the ratio of the diameter in G_o to the new diameter in G_s (in plots "orig diameter/diameter"); (b) **Number of connected components** - we measure the ratio of the number of connected components in G_s to that in G_o (in plots "comp/orig comp"); (c) **Modularity** - we measure the ratio of modularity in G_s to that of G_o (in plots "mod/orig mod", Networkit [39] provides an implementation of the Louvain method). Certain network properties are represented better by their distributions over the nodes. In order to accurately compare the distributions, we use the Spearman rank correlation coefficient. This effectively, reveals how different the sparse graph is from the original in the context of these properties where a correlation value of 1 means they are perfectly correlated and correlation value of 0 means no correlation. The following distributions are compared using the Spearman rank: (a) Node **betweeenness** centrality; (b) **PageRank** centrality; (c) **Degree distribution**; and (d) **Clustering coefficient distribution** (c_i). The method changes slightly in comparing node betweenness centrality. Considering that the cost of computing betweenness for large graphs is very expensive, we make use of an approximate method provided by Networkit. However, to ensure accuracy we compute this 10 times and take average of the positional rankings and then compute the Spearman rank correlation.

6.1   Single-level Algorithm

The single-level algorithm was tested with both unnormalized and normalized algebraic distances. The results for unnormalized algebraic distance are presented in Figures 3, 4, 5, and 6 for groups SN1, SN2, CIT, and BIO, respectively. (The results for the normalized algebraic distance can be found in Appendix A.) In each figure, 3 columns, and 7 rows of plots are presented. In all 4 figures: (a) each column corresponds to the type of filtering, i.e., to the types of edges that retain after sparsification; (b) each row corresponds to the type of comparison. Each plot contains several colored curves that correspond to the respective graphs (see vertical legend). One point

in each curve corresponds to an average of the measured comparison method over 10 runs for the corresponding edge ratio in each. The x- and y-axes correspond to the sparsification ratio and method of comparison, respectively. In the y-axis of betweenness, PageRank, degree, and clustering coefficient distribution centralities, the Spearman rank is denoted by ρ. For example, we examine the behavior of the degrees in social network Google+ in SN1 when δ-strong edges retain after sparsification. In Figure 3, we find a row "Degree centrality" (row 3). The results for retaining δ-strong edges are found in the third column. The black curve corresponds to Google+, where each point is an average of 10 runs.

Note: Most curves do not reach a visible zero of the x-axis. This is because the sparsification is interrupted when the number of edges becomes less than the number of nodes.

δ-weak edges   Plots labelled as δ-weak (column 1) are results obtained by retaining only weak edges, when δ-weak edges are preferred during sparsification (i.e, δ-strong edges are deleted). In this type of sparsification, we expect that sparsification of the local structure will mostly dominate the sparsification of the global structure. Indeed, we observe that properties (such as the betweenness centrality, diameter, and the number of components) that heavily depend on usually limited number of long-range weak connections are well preserved.

δ-strong   Plots labelled as δ-strong (column 3) are results obtained by retaining δ-strong edges and removing δ-weak edges. By preferring δ-strong edges during sparsification, we attempt to preserve properties that depends on the local structure of the graph. Such properties as clustering coefficient, pagerank and degree centrality survive sparsification better when this method is used. In particular, we can observe that the clustering coefficient (which is in many cases the reason for a strong community structure) is preserved at the level of $\approx 75\%$ in SN1 when 70% of edges are removed (instead of $\approx 40\%$ for δ-weak sparsification). A similar phenomena is observed in BIO. It is interesting to note that in SN2, in comparison to the δ-weak sparsification, the changes in the clustering coefficient are not significant.

Mixture sparsification   In plots labelled as mixed, we maintain a balance between the δ-weak and δ-strong types of sparsification by preferring ensuring that both are sparsified. For such properties as the betweenness centrality, PageRank and degree centrality, the results are better for up to 20% sparsification ratio when compared to selecting either weak or strong edges. For such properties as the clustering coefficient, modularity, diameter and connected components, retaining both weak and strong edges provides results that is in between that produced by weak or strong edges sparsification.
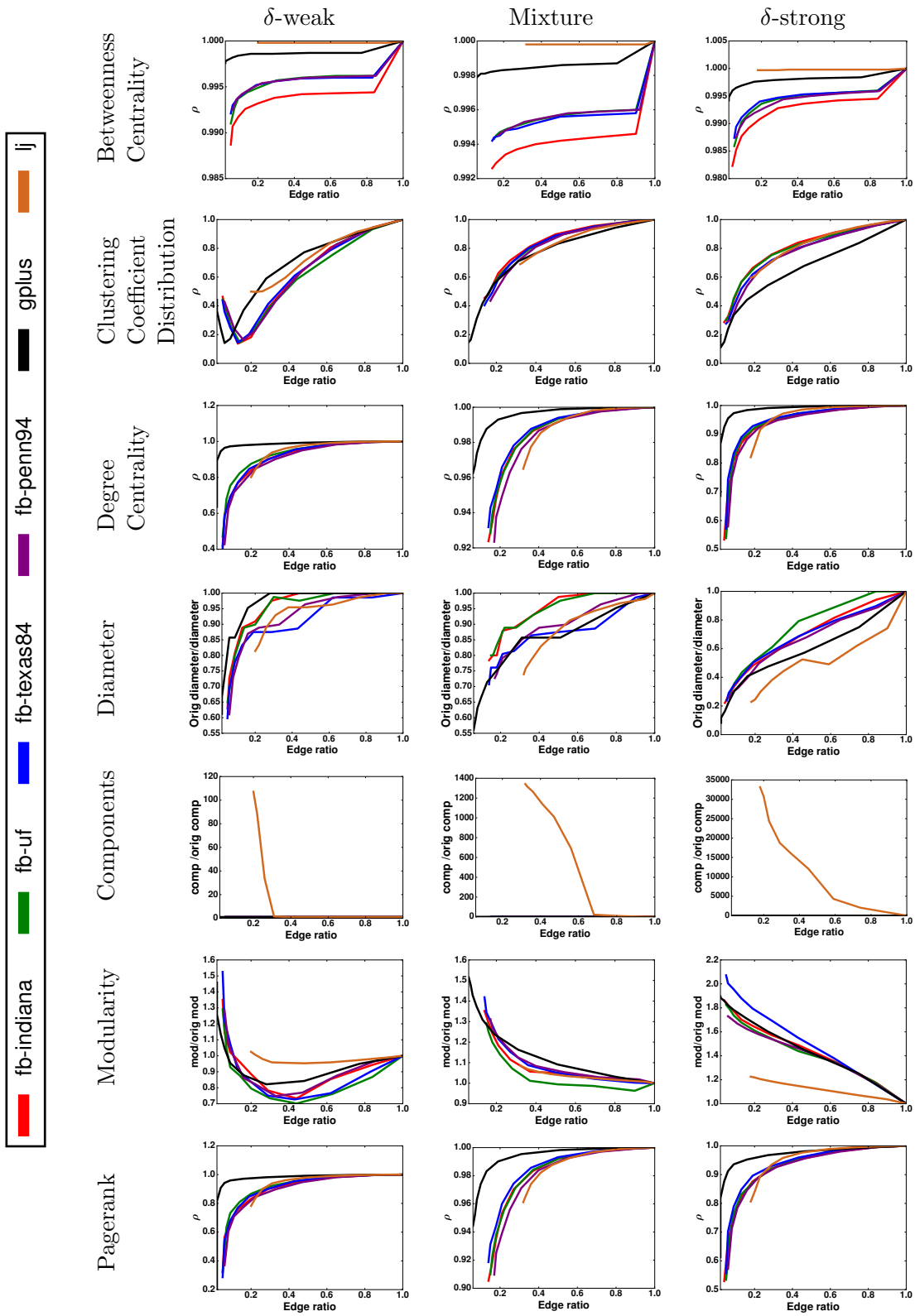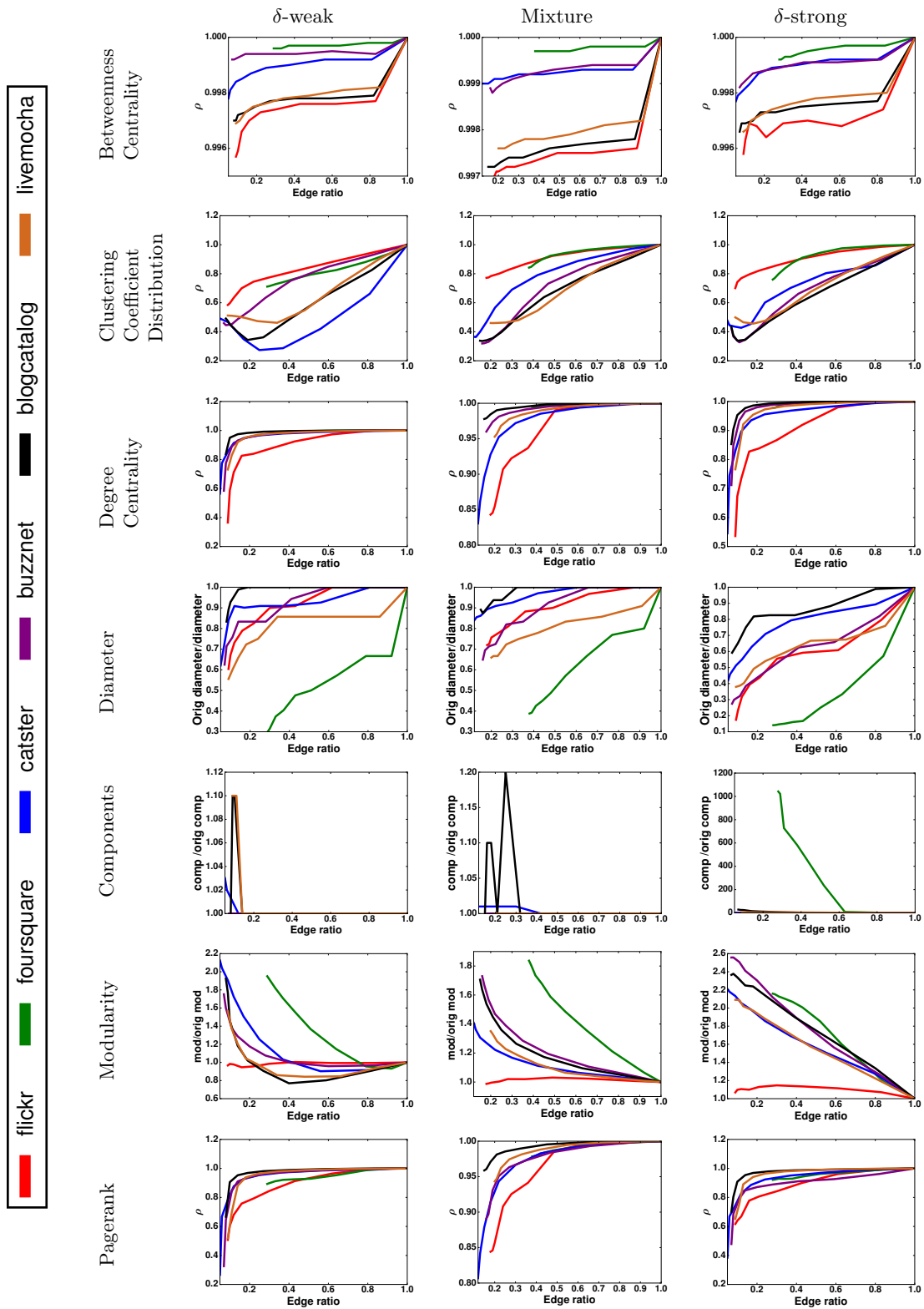
Figure 3: Social Networks 1
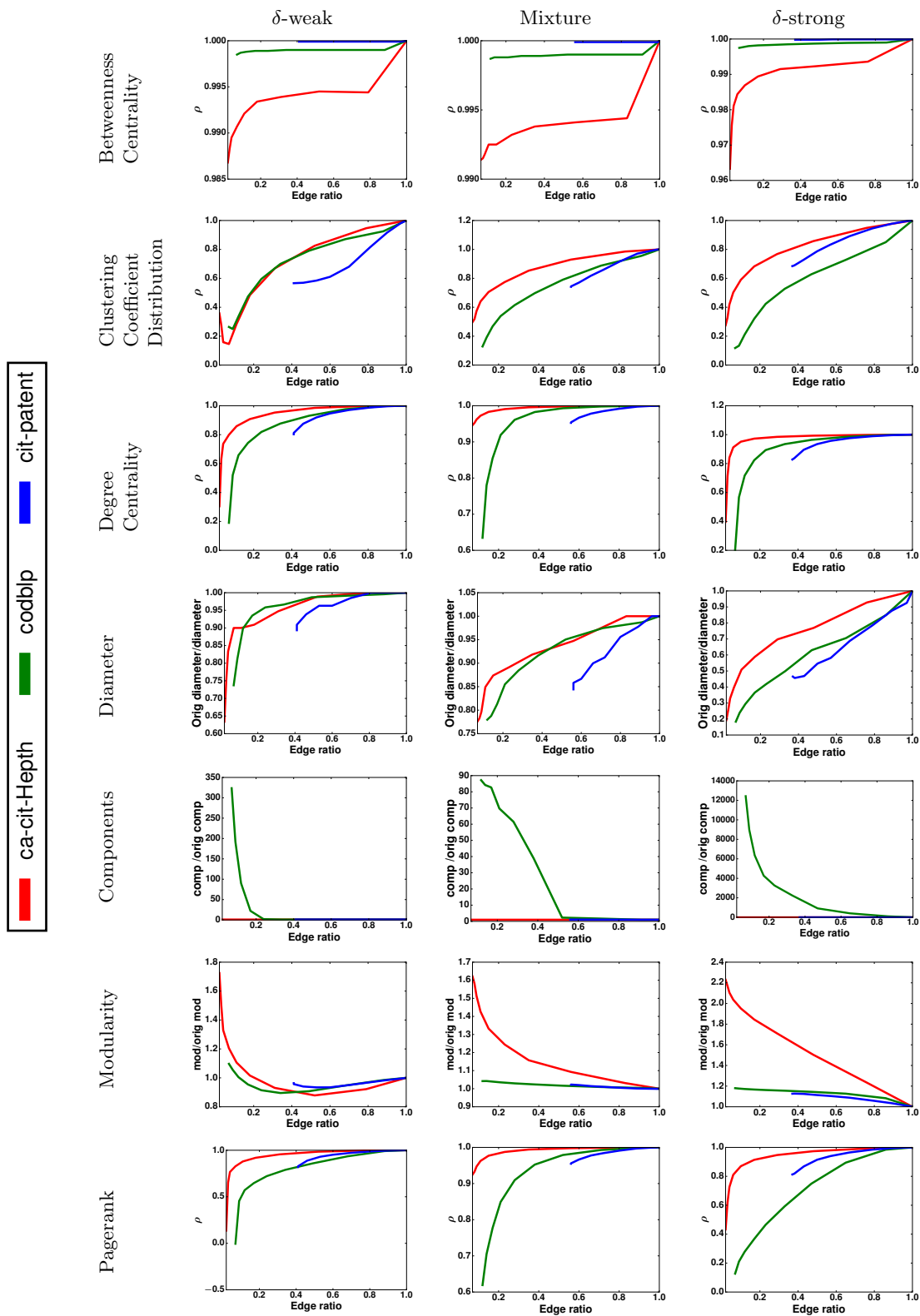
Figure 4: Social Networks 2
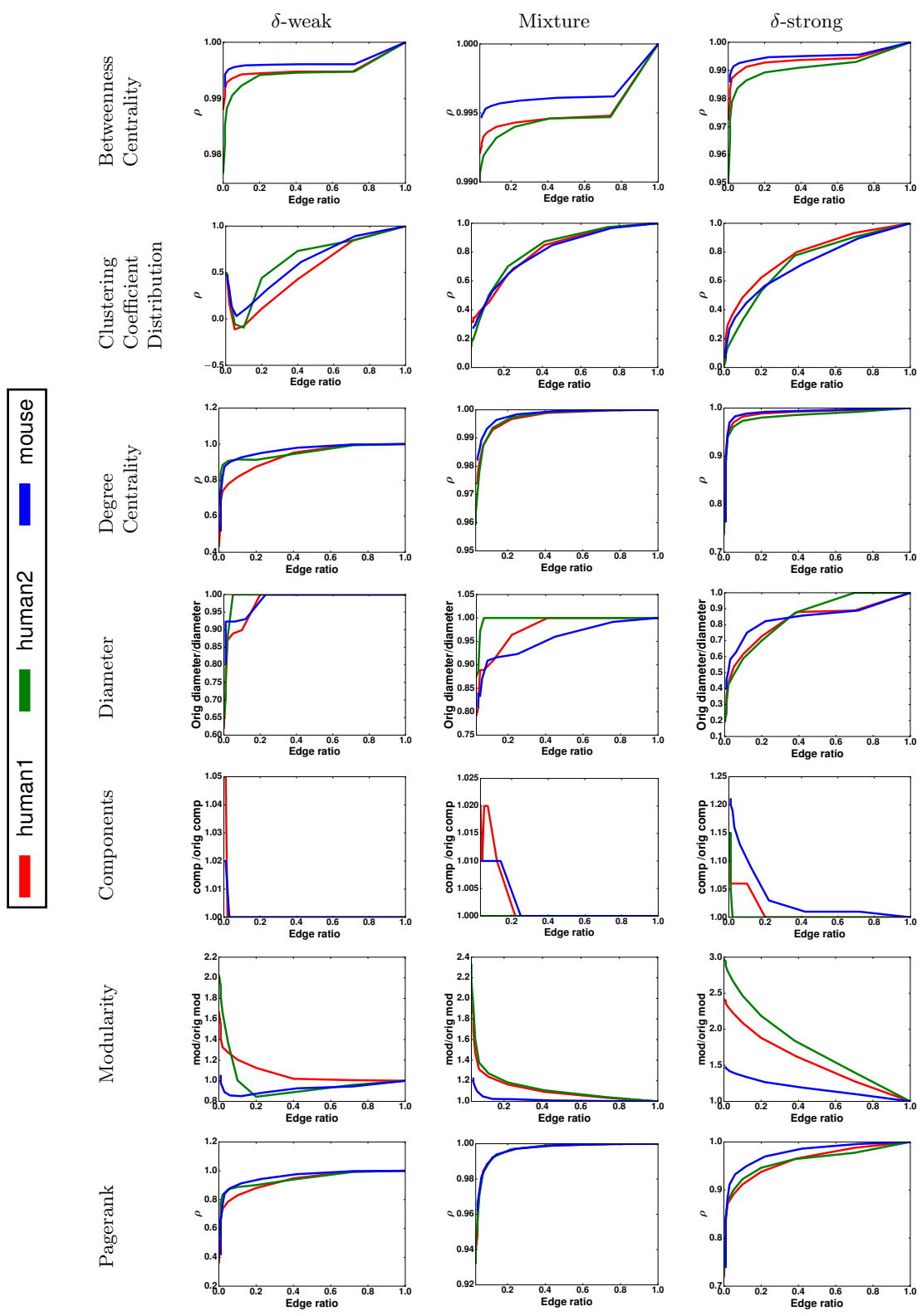
Figure 5: Citation Networks

14

Figure 6: Biological Networks

Comparing with Local Degree   In the introduction, we mentioned the Local Degree method (LD) [24] which favors the retention of edges participating in hubs (nodes with high degree). In order to compare our method to LD, we ran the single level algorithm for retaining weak edges, strong edges and a mixture of both on the Google+ graph (google-plus in Table 1). Same set of network properties discussed earlier in this section were used for comparison. For betweenness centrality, degree centrality, local clustering coefficient and PageRank, we plot the Spearman rank correlation against the edge ratio. Figure 7 shows the plots of δ-weak, δ-strong, mixed and local degree(LD) for each property. The results are similar for betweenness centrality, degree centrality and PageRank. However, for such properties as modularity and clustering coefficient, the algebraic distance performs better than LD especially when sparsification is aggressive. The δ-weak sparsification preserves the diameter slightly bettern than LD while the δ-strong method did not perform well on it and on the number of components. We note that the LD method was comprehensively studied on the Facebook networks only. Four Facebook networks in SN1 demonstrate similar performance with both methods. The Google+ network has exceptionally high clustering coefficient (0.52 vs. 0.23 in Facebook networks) and smaller diameter (6 vs. 8 in Facebook networks) which are more difficult to preserve if the method does not distinguish between local- and global-range connections.
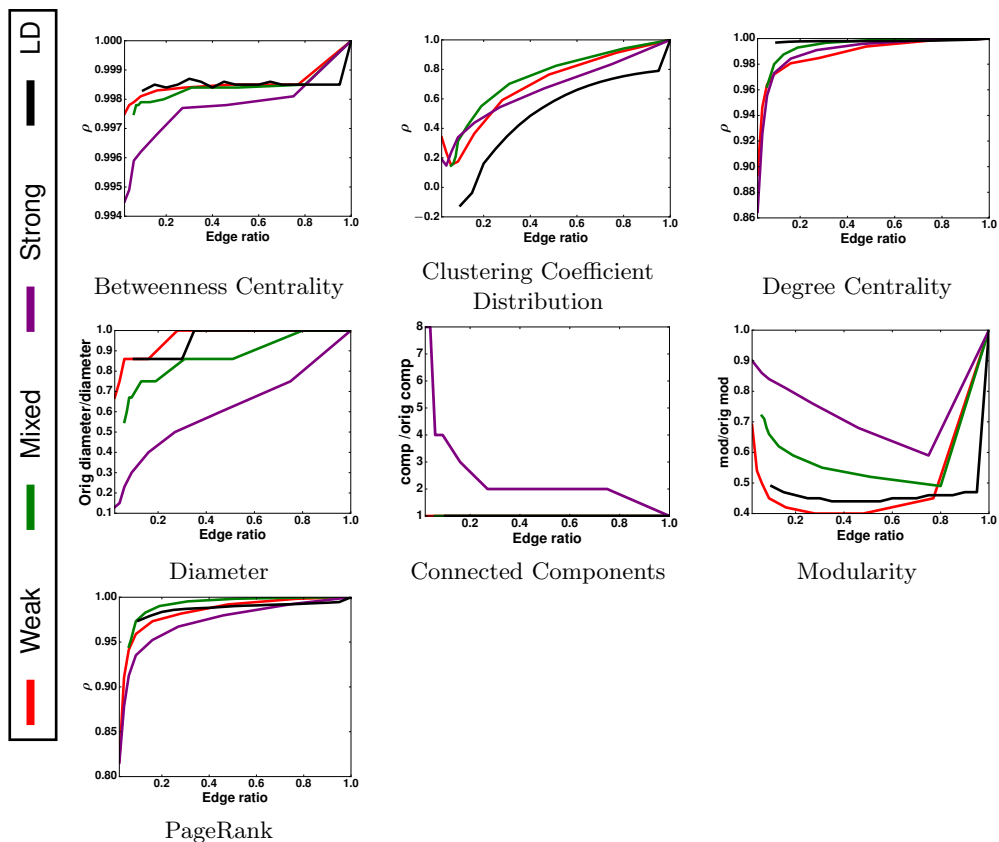

Figure 7: Comparison of LD and single-level algebraic distance methods.

6.2 Multi-Level Results

The purpose of the multilevel approach is to extend the general sparsification framework to enable highly controllable sparsification of bundles of edges at multiple coarse-grained resolutions. Similar to the single-level experiments, we group the networks into 4 groups. Tables 2, 3, 4, and 5 show the results of the multilevel algorithm for sets SN1, SN2, BIO and CIT, respectively. The 4 major column sections in the aforementioned tables consists of the graph name, the levels' configuration for which sparsifcation is tested, the number of edges at each setting, and the network properties that we study. The properties column consist of the following properties: a) CC - clustering coefficient b) D - Diameter of the graph, c) Q - Modularity of the graph, d) Γ - the number of components in the network, e) BC_ρ - Spearman rank correlation for betweenness centrality, f) PR_ρ - Spearman rank correlation of Pagerank, g) DC_ρ - Spearman rank correlation of degree centrality, and h) CC_ρ - Spearman rank correlation of the clustering coefficient. Correlation here represents the correlation between the original graph and the sparse graph. The "Level" column contains the sparsification settings at different levels, where G0 represents the original graph, G1 is the graph with sparsification only at the coarsest levels, G2 is the graph with sparsification only at the middle levels and G3 represents the graph with sparsification at the fine levels. In order to keep the results comparable, we keep the sparsification ratio between 20% to 40%. The sparsification parameter is obtained by a binary search fitting algorithm. Note that we do not compare the sparse graphs (G1, G2, G3) to themselves but only with the fine graph G0. The parameter setting of a coarsening was similar to one described in [28] with interpolation order 1.

Graph Name	Level	$|E|$	Properties							
			CC	D	Q	Γ	BC_ρ	PR_ρ	DC_ρ	CC_ρ
flickr	G0	2.3M	0.09	9.0	0.67	83	1.0	1.0	1.0	1.0
	G1	921.0K	0.15	35.0	0.91	144	1.0	0.6	0.62	0.78
	G2	496.1K	0.12	18.0	0.84	134	1.0	0.71	0.73	0.8
	G3	634.9K	0.04	25.0	0.55	5.8K	1.0	0.64	0.77	0.74
buzznet	G0	2.8M	0.25	5.0	0.31	1	1.0	1.0	1.0	1.0
	G1	713.8K	0.26	11.0	0.5	1	1.0	0.82	0.91	0.6
	G2	919.3K	0.21	18.0	0.3	12	1.0	0.83	0.94	0.6
	G3	666.6K	0.1	16.0	0.29	239	1.0	0.75	0.89	0.28
foursquare	G0	3.2M	0.22	4.0	0.41	1	1.0	1.0	1.0	1.0
	G1	1.1M	0.17	36.0	0.94	18	1.0	0.74	0.88	0.87
	G2	765.9K	0.19	47.0	0.94	366	1.0	0.49	0.78	0.79
	G3	1.1M	0.04	14.0	0.57	10.0K	1.0	0.36	0.74	0.73
catster	G0	5.4M	0.41	10.0	0.39	281	1.0	1.0	1.0	1.0
	G1	1.3M	0.31	15.0	0.69	293	1.0	0.59	0.59	0.39
	G2	1.7M	0.26	11.0	0.37	360	1.0	0.86	0.87	0.63
	G3	1.5M	0.27	14.0	0.29	1.1K	1.0	0.76	0.84	0.4
blog-catalog	G0	2.1M	0.46	9.0	0.32	1	1.0	1.0	1.0	1.0
	G1	423.9K	0.41	14.0	0.67	1	1.0	0.81	0.87	0.47
	G2	570.8K	0.26	11.0	0.27	9	1.0	0.85	0.89	0.44
	G3	566.1K	0.18	12.0	0.23	391	1.0	0.7	0.83	0.29
livemocha	G0	2.2M	0.06	6.0	0.36	1	1.0	1.0	1.0	1.0
	G1	636.9K	0.04	12.0	0.45	3	1.0	0.89	0.91	0.48
	G2	869.1K	0.04	10.0	0.29	8	1.0	0.9	0.91	0.49
	G3	556.7K	0.02	11.0	0.29	1.4K	1.0	0.77	0.88	0.38

Table 3: Multiscale results for social networks 2(SN2) graphs

Graph Name	Level	$\lvert E\rvert$	Properties							
			CC	D	Q	Γ	BC_ρ	PR_ρ	DC_ρ	CC_ρ
fb-indiana	G0	1.3M	0.21	8.0	0.45	1	1.0	1.0	1.0	1.0
	G1	361.4K	0.31	13.0	0.93	18	1.0	0.84	0.83	0.64
	G2	349.8K	0.14	10.0	0.34	8	0.99	0.89	0.89	0.57
	G3	402.7K	0.04	12.0	0.3	37	0.99	0.93	0.95	0.03
fb-texas84	G0	1.6M	0.2	7.0	0.38	1	1.0	1.0	1.0	1.0
	G1	374.7K	0.29	16.0	0.92	16	1.0	0.86	0.82	0.57
	G2	574.8K	0.13	9.0	0.31	5	0.99	0.94	0.94	0.66
	G3	521.9K	0.05	12.0	0.24	29	1.0	0.94	0.96	0.08
fb-uf	G0	1.5M	0.22	8.0	0.44	1	1.0	1.0	1.0	1.0
	G1	423.4K	0.24	12.0	0.61	3	0.99	0.88	0.88	0.58
	G2	425.1K	0.16	11.0	0.37	15	1.0	0.9	0.9	0.6
	G3	418.9K	0.05	11.0	0.27	57	1.0	0.92	0.95	-0.01
fb-penn94	G0	1.4M	0.22	8.0	0.48	1	1.0	1.0	1.0	1.0
	G1	351.7K	0.33	16.0	0.95	16	1.0	0.85	0.8	0.57
	G2	463.4K	0.16	11.0	0.38	23	1.0	0.89	0.9	0.6
	G3	365.6K	0.04	14.0	0.3	122	1.0	0.89	0.92	-0.09
livejournal	G0	34.7M	0.35	21.0	0.75	1	1.0	1.0	1.0	1.0
	G1	13.6M	0.47	72.0	1.0	1.7K	1.0	0.85	0.81	0.75
	G2	13.4M	0.39	42.0	0.8	7.1K	1.0	0.91	0.87	0.76
	G3	8.2M	0.02	30.0	0.72	316.9K	1.0	0.67	0.73	0.37
gplus	G0	12.2M	0.52	6.0	0.47	1	1.0	1.0	1.0	1.0
	G1	3.6M	0.54	14.0	0.89	15	1.0	0.91	0.9	0.59
	G2	3.0M	0.42	12.0	0.38	18	1.0	0.9	0.88	0.57
	G3	3.3M	0.15	19.0	0.26	905	1.0	0.79	0.88	-0.26

Table 2: Multiscale results for social networks 1 (SN1) graphs

Graph Name	Level	$\lvert E\rvert$	Properties							
			CC	D	Q	Γ	BC_ρ	PR_ρ	DC_ρ	CC_ρ
bio-human-gene1	G0	12.3M	0.63	8.0	0.38	17	1.0	1.0	1.0	1.0
	G1	2.8M	0.66	18.0	0.8	19	0.99	0.9	0.95	0.74
	G2	4.0M	0.33	9.0	0.39	22	0.99	0.91	0.91	0.71
	G3	3.9M	0.06	11.0	0.33	78	0.99	0.89	0.93	0.21
bio-human-gene2	G0	9.0M	0.66	7.0	0.31	2	1.0	1.0	1.0	1.0
	G1	2.4M	0.67	7.0	0.74	15	1.0	0.87	0.86	0.74
	G2	3.1M	0.64	26.0	0.52	14	0.98	0.87	0.88	0.74
	G3	2.5M	0.62	39.0	0.42	66	0.93	0.88	0.87	0.64
bio-mouse-gene	G0	14.5M	0.53	12.0	0.62	97	1.0	1.0	1.0	1.0
	G1	4.6M	0.6	21.0	0.89	105	0.99	0.94	0.95	0.72
	G2	4.1M	0.28	13.0	0.56	132	1.0	0.93	0.94	0.65
	G3	4.1M	0.06	14.0	0.52	400	1.0	0.91	0.95	0.08

Table 4: Multiscale results for biological (BIO) networks

Graph Name	Level	$|E|$	Properties							
			CC	D	Q	Γ	BC_ρ	PR_ρ	DC_ρ	CC_ρ
ca-cit-Hepth	G0	2.4M	0.61	9.0	0.41	74	1.0	1.0	1.0	1.0
	G1	487.5K	0.7	18.0	0.93	86	0.99	0.84	0.84	0.75
	G2	875.3K	0.49	13.0	0.37	111	0.99	0.91	0.94	0.81
	G3	722.2K	0.21	16.0	0.25	279	0.99	0.83	0.95	-0.04
cit-patent	G0	16.5M	0.09	26.0	0.81	3.6K	1.0	1.0	1.0	1.0
	G1	5.8M	0.16	61.0	0.93	43.8K	1.0	0.79	0.78	0.73
	G2	3.4M	0.13	57.0	0.97	631.4K	1.0	0.58	0.64	0.59
	G3	5.0M	0.01	39.0	0.84	235.5K	1.0	0.68	0.74	0.54
codblp	G0	15.2M	0.82	23.0	0.84	1	1.0	1.0	1.0	1.0
	G1	5.3M	0.91	30.0	0.98	20.1K	1.0	0.55	0.78	0.68
	G2	3.4M	0.81	32.0	0.97	44.3K	1.0	0.27	0.63	0.62
	G3	4.7M	0.5	29.0	0.84	29.1K	1.0	0.51	0.82	0.38

Table 5: Multiscale results for citation (CIT) networks

6.3 Running time

Figures 8(a-b) show the running time of both single- and multi-level algorithms for varying sparsification ratios. Each point in the plot represents the number of edges in the graph versus the runtime in seconds averaged over three runs. The coefficient of determination, R^2, shows how well the regression line fits the model. An R^2 of 0 indicates the line does not fit the data and an R^2 of 1 indicates the line perfectly fits the data. The results show that both algorithms scales linearly with the number of edges in the graph. As mentioned earlier, this is important as it defeats the purpose of sparsification if the algorithm is slow. The experiments were performed in a Linux environment on a multicore compute server with 64 Intel Xeon cores and 64 GB of memory.
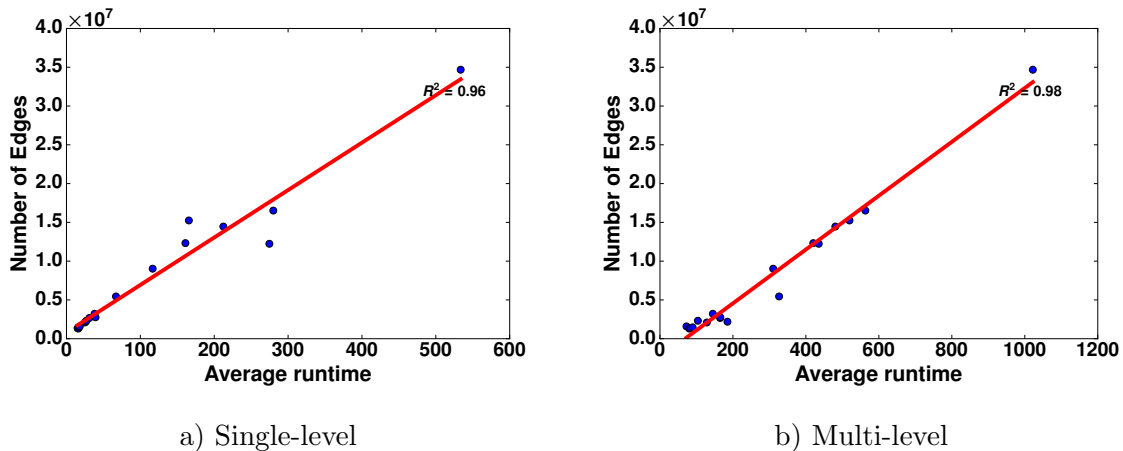

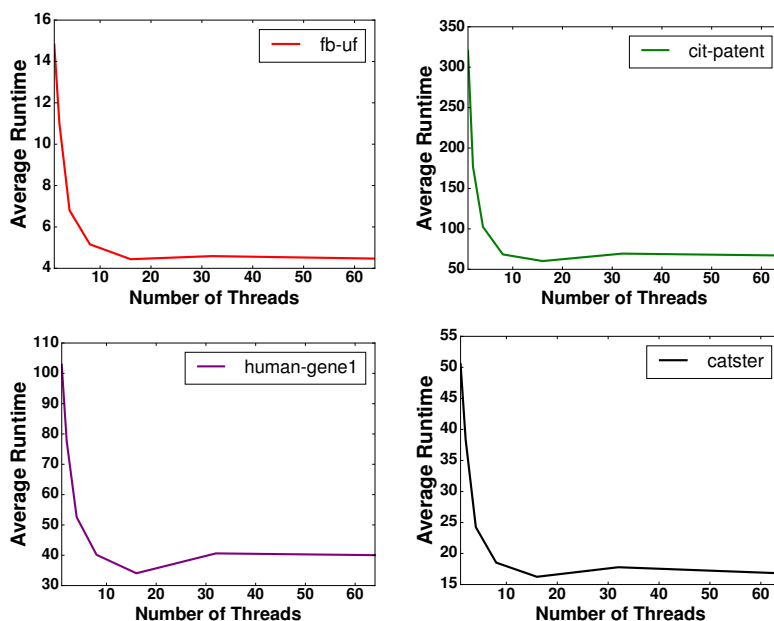a) Single-level   b) Multi-level

Figure 8: Running time of sparsification.

6.4 Parallelization

Parallelization of the single-level algorithm does not require redesigning it. There are two computationally intensive parts of our method that gain from parallelization. One is the computation of the algebraic distance and the other the deletion of edges. Because of the implicitly parallel nature of the Jacobi over-relaxation, we are able to parallelize it by using OpenMP's shared data, multiple thread model. Since vector updates are independent, this method is highly efficient, creating speed gains of more than 50% with only 8 threads as seen in Figure 9. Figure 9 shows the benchmark results of parallelizing the algebraic distance computation where y-axis represents the average run-time averaged over 3 runs and x-axis represents the number of threads. We tested with number of threads ranging from 1 to 64 on 4 networks, namely, fb-uf, human-gene1, cit-patent, and catster. The experiments were performed in a Linux environment on a multicore compute server with 64 Intel Xeon cores and 64 GB of memory.

Figure 9: Running time in shared memory model.

7 Conclusions

In this study we introduced single- and multi-level methods of network sparsification by algebraic distance. While many sparsification methods exist, most of them target certain properties without distinguishing short- and long-range connections that is the main goal of our method. We showed that by enabling different filtering capabilities, sparsification can be tuned to preserve either global or local structure or a combination of both. In addition to preserving a host of graph properties, we believe that the development of the multilevel sparsification framework can serve as a foundation for future work in that direction in which a variety of sparsification criteria (such as the algebraic distance) can be incorporated into it.

A Normalized Sparsification

We experimented with the single-level algorithm that employ normalized algebraic distances (see line 15 of Algorithm 1). The purpose of this normalization is to decrease the strength of connection expressed in the algebraic distance between hubs. The normalization results show that normalizing the algebraic distance further improves properties that are sensitive to the existence of weak edges. Example are diameter and connected components. As seen in the plots for diameter (see δ-weak column in Figures 10, 11, 12, and 13), the minimum edge ratio before the diameter deteriorates is further improved. Similarly for the number of components the number of components for the smallest sparse graph is reduced and some case kept constant as seen in δ-weak column in Figures 10, 11, 12, and 13). Such properties as local clustering coefficient, degree centrality, and PageRank that do not depend on global edges are relatively unaffected.
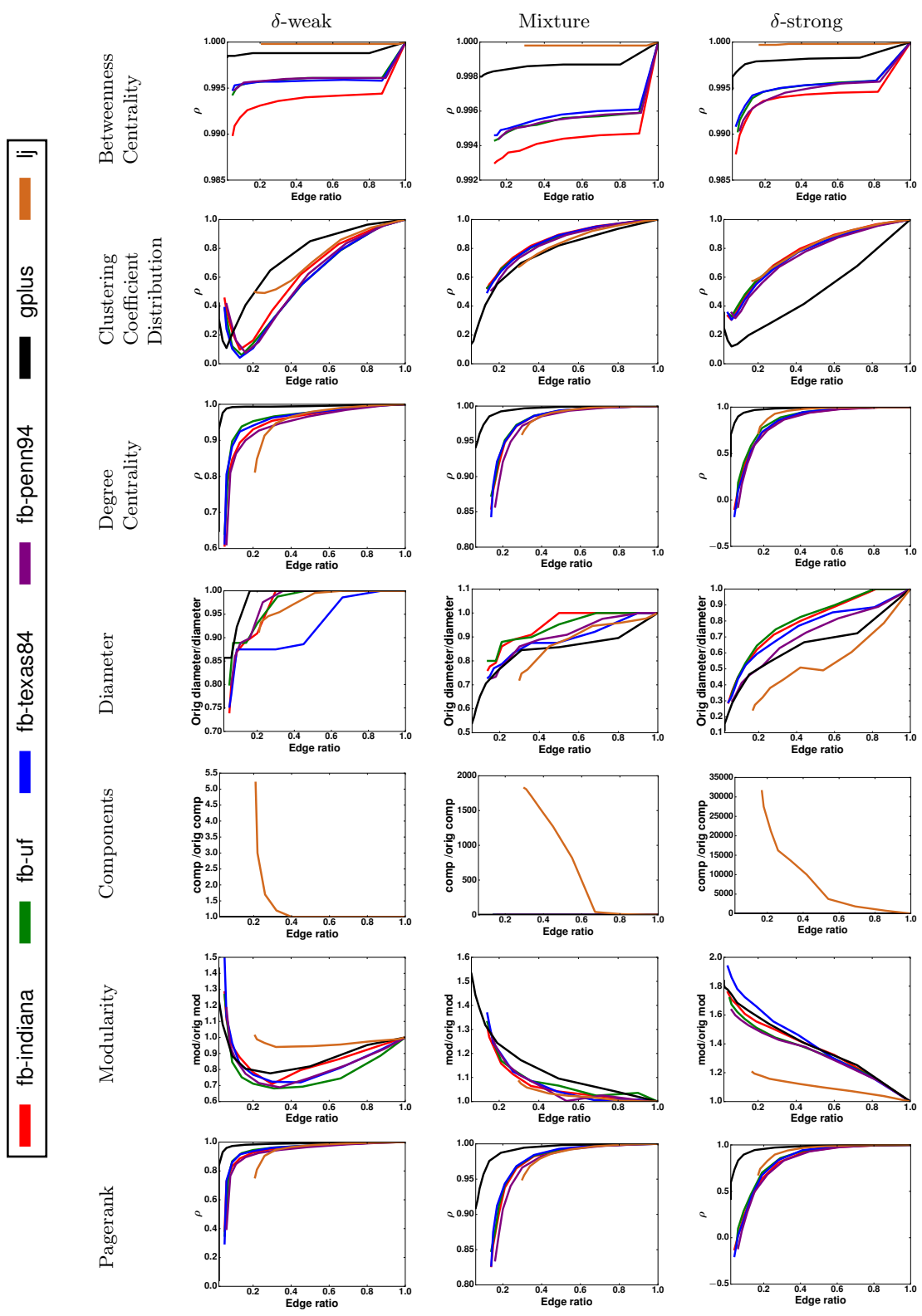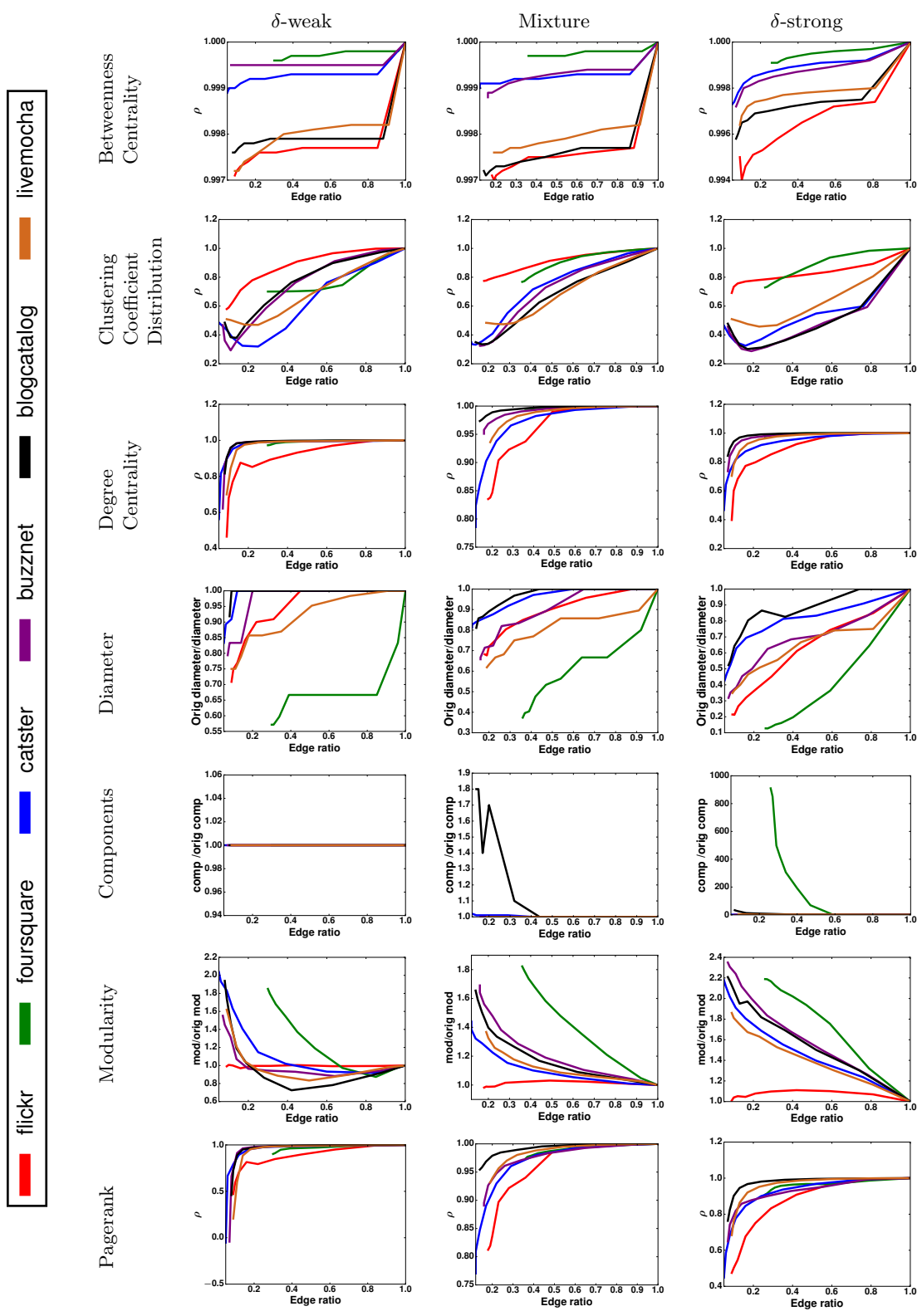
Figure 10: Social Networks 1
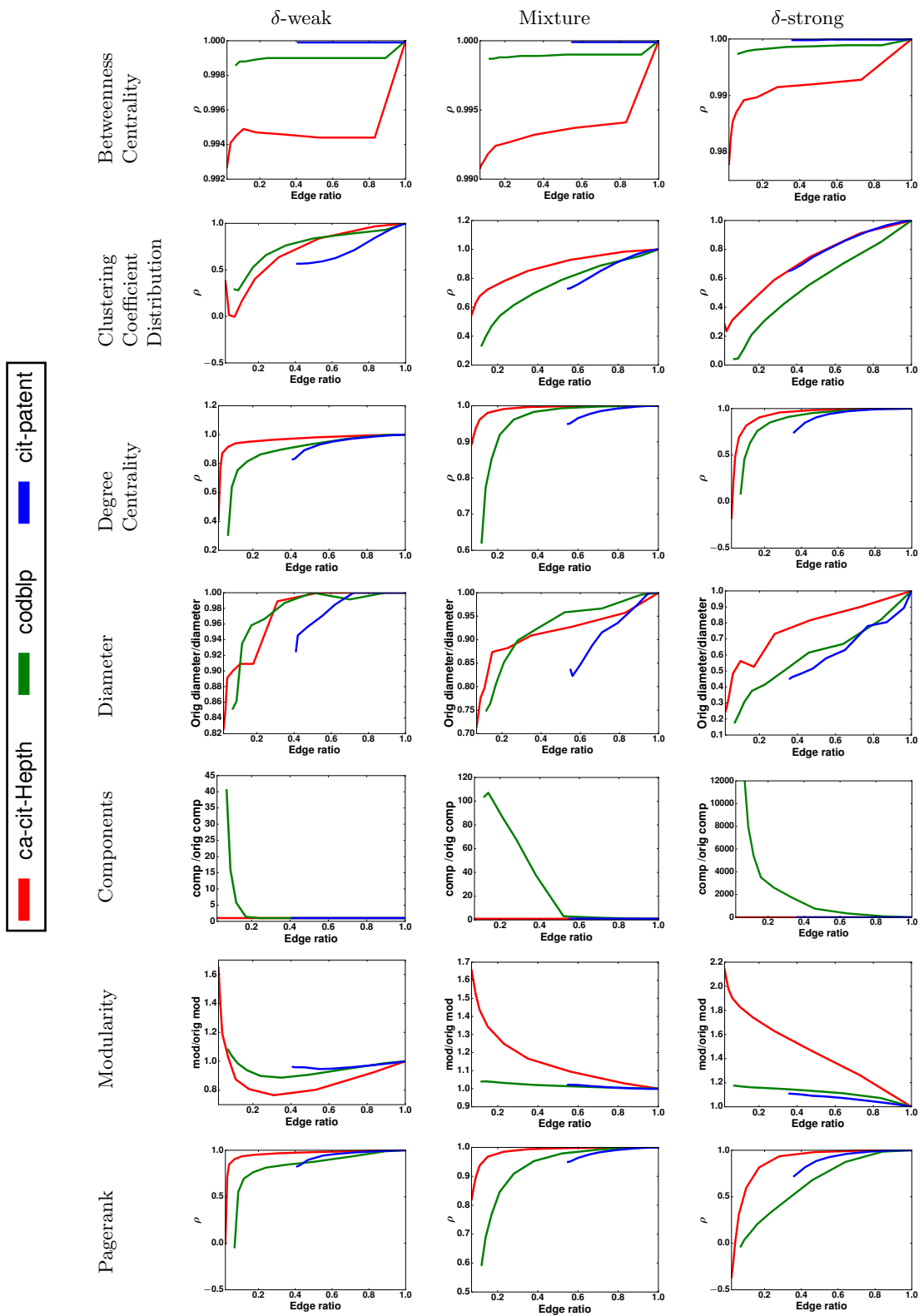
Figure 11: Social Networks 2
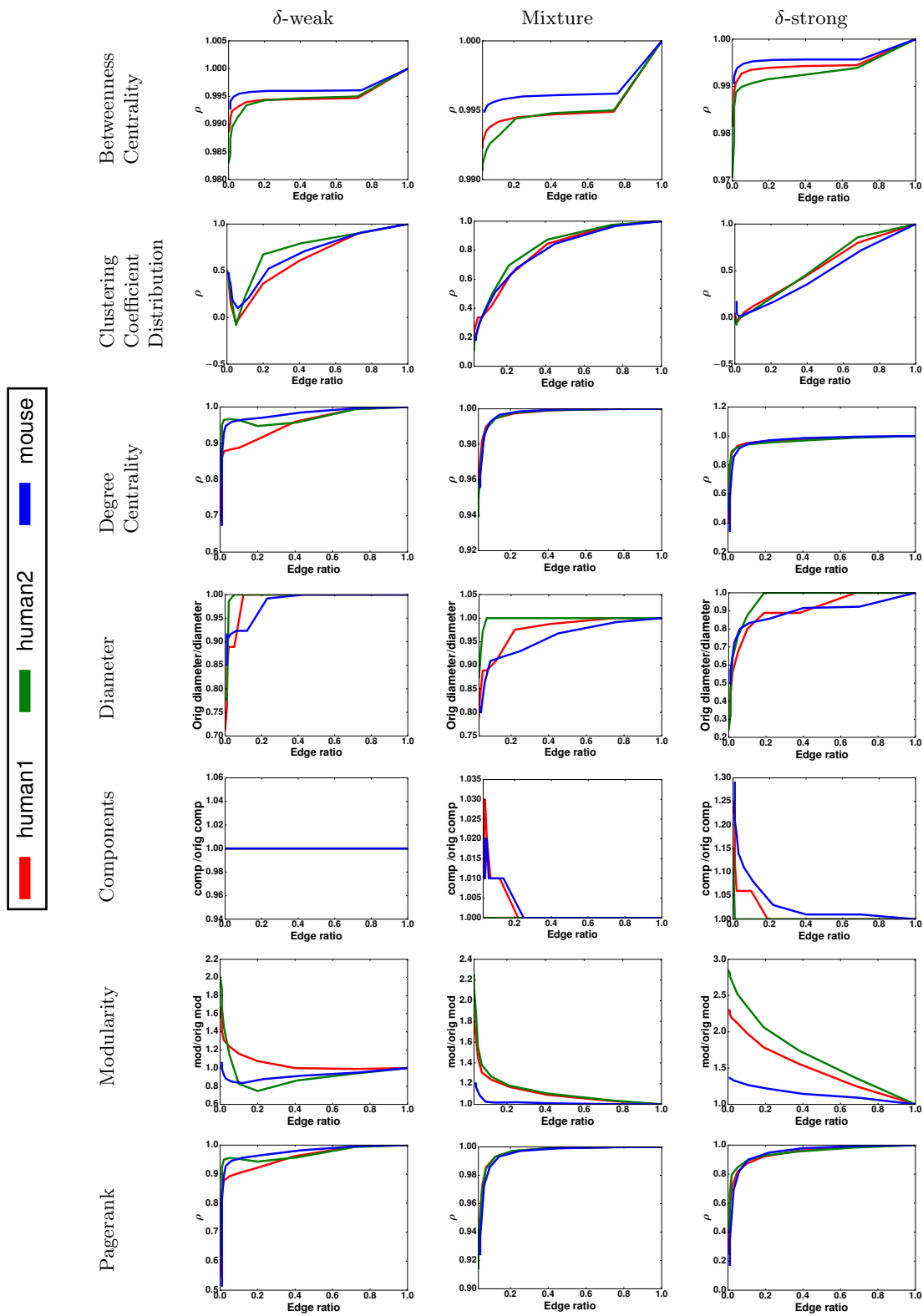
Figure 12: Citation Networks

Figure 13: Biological Networks

25

References

[1] Nesreen K. Ahmed, Jennifer Neville, and Ramana Kompella. Network sampling: From static to streaming graphs. *ACM Trans. Knowl. Discov. Data*, 8(2):7:1–7:56, jun 2013.

[2] David Auber. Tulipa huge graph visualization framework. In *Graph Drawing Software*, pages 105–126. Springer, 2004.

[3] David A Bader, Shiva Kintali, Kamesh Madduri, and Milena Mihail. Approximating betweenness centrality. In *Algorithms and Models for the Web-Graph*, pages 124–137. Springer, 2007.

[4] Mathieu Bastian, Sebastien Heymann, Mathieu Jacomy, et al. Gephi: an open source software for exploring and manipulating networks. *ICWSM*, 8:361–362, 2009.

[5] P. M. Binder. Frustration in complexity. *Science*, 322:323, 2008.

[6] A. Brandt and D. Ron. Chapter 1 : Multigrid solvers and multilevel optimization strategies. In J. Cong and J. R. Shinnerl, editors, *Multilevel Optimization and VLSICAD*. Kluwer, 2003.

[7] Achi Brandt, James J. Brannick, Karsten Kahl, and Irene Livshits. Bootstrap AMG. *SIAM J. Scientific Computing*, 33(2):612–632, 2011.

[8] Pavel Chebotarev and Elena Shamis. On proximity measures for graph vertices. *arXiv preprint math/0602073*, 2006.

[9] Jie Chen and Ilya Safro. Algebraic distance on graphs. *SIAM Journal on Scientific Computing*, 33(6):3468–3490, 2011.

[10] Fan Chung, Wenbo Zhao, and Mark Kempton. Ranking and sparsifying a connection graph. *Internet Mathematics*, 10(1-2):87–115, 2014.

[11] Wouter De Nooy, Andrej Mrvar, and Vladimir Batagelj. *Exploratory social network analysis with Pajek*, volume 27. Cambridge University Press, 2011.

[12] Leo A Goodman. Snowball sampling. *The annals of mathematical statistics*, pages 148–170, 1961.

[13] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pages 11–15, Pasadena, CA USA, August 2008.

[14] M. S. Handcock and K. J. Gile. On the Concept of Snowball Sampling. *ArXiv e-prints*, 2011.

[15] Pili Hu and Wing Cheong Lau. A survey and taxonomy of graph sampling. *CoRR*, abs/1308.5865, 2013.

[16] Yifan Hu. Efficient, high-quality force-directed graph drawing. *Mathematica Journal*, 10(1):37–71, 2005.

[17] S. Itzkovitz, R. Levitt, N. Kashtan, R. Milo, M. Itzkovitz, and U. Alon. Coarse-graining and self-dissimilarity of complex networks. *Physical Review E*, 71(1):016127, 2005.

[18] Emmanuel John and Ilya Safro. Network sparsification by algebraic distance. `https://github.com/emmanuj/ml-sparsifier`, 2015 – 2016.

[19] Risi Imre Kondor and John Lafferty. Diffusion kernels on graphs and other discrete structures. In *Proceedings of the 19th international conference on machine learning*, pages 315–322, 2002.

[20] Jérôme Kunegis. Konect: The koblenz network collection. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13 Companion, pages 1343–1350, Republic and Canton of Geneva, Switzerland, 2013. International World Wide Web Conferences Steering Committee.

[21] Maciej Kurant, Minas Gjoka, Yan Wang, Zack W. Almquist, Carter T. Butts, and Athina Markopoulou. Coarse-grained topology estimation via graph sampling. *CoRR*, abs/1105.5488, 2011.

[22] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[23] Jure Leskovec and Rok Sosič. SNAP: A general purpose network analysis and graph mining library in C++. `http://snap.stanford.edu/snap`, June 2014.

[24] Gerd Lindner, Christian L. Staudt, Michael Hamann, Henning Meyerhenke, and Dorothea Wagner. Structure-preserving sparsification of social networks. In *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2015*, ASONAM '15, pages 448–454, New York, NY, USA, 2015. ACM.

[25] Oren E Livne and Achi Brandt. Lean algebraic multigrid (lamg): Fast graph laplacian linear solver. *SIAM Journal on Scientific Computing*, 34(4):B499–B522, 2012.

[26] Enys Mones, Lilla Vicsek, and Tamas Vicsek. Hierarchy measure for complex networks. *PLoS ONE*, 7(3):e33799, 03 2012.

[27] M. E. J. Newman. *Networks, An Introduction*. Oxford University Press, 2010.

[28] Dorit Ron, Ilya Safro, and Achi Brandt. Relaxation-based coarsening and multiscale graph organization. *Multiscale Modeling & Simulation*, 9(1):407–423, 2011.

[29] Ryan A. Rossi and Nesreen K. Ahmed. bio-human-gene1 - biological networks, 2013.

[30] Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[31] I. Safro, D. Ron, and A. Brandt. Graph minimum linear arrangement by multilevel weighted edge contractions. *Journal of Algorithms*, 60(1):24–41, 2006.

[32] Ilya Safro, Peter Sanders, and Christian Schulz. Advanced coarsening schemes for graph partitioning. *Journal of Experimental Algorithmics (JEA)*, 19:2–2, 2015.

[33] Tanwistha Saha, Huzefa Rangwala, and Carlotta Domeniconi. Sparsification and sampling of networks for collective classification. In *Social Computing, Behavioral-Cultural Modeling and Prediction*, pages 293–302. Springer, 2013.

[34] Matthew J Salganik and Douglas D Heckathorn. Sampling and estimation in hidden populations using respondent-driven sampling. *Sociological methodology*, 34(1):193–240, 2004.

[35] Venu Satuluri, Srinivasan Parthasarathy, and Yiye Ruan. Local graph sparsification for scalable clustering. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 721–732. ACM, 2011.

[36] Daniel A Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. *SIAM Journal on Computing*, 40(6):1913–1926, 2011.

[37] Daniel A Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 81–90. ACM, 2004.

[38] Daniel A Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *SIAM Journal on Computing*, 40(4):981–1025, 2011.

[39] Christian Staudt, Aleksejs Sazonovs, and Henning Meyerhenke. Networkit: An interactive tool suite for high-performance network analysis. *CoRR*, abs/1403.3005, 2014.

[40] D. Stutzbach, R. Rejaie, N. Duffield, S. Sen, and W. Willinger. Sampling techniques for large, dynamic graphs. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pages 1–6, April 2006.

[41] Arthur D Szlam, Mauro Maggioni, and Ronald R Coifman. Regularization on graphs with function-adapted diffusion processes. *The Journal of Machine Learning Research*, 9:1711–1739, 2008.

[42] C. Walshaw. Multilevel refinement for combinatorial optimisation problems. *Annals Oper. Res.*, 131:325–372, 2004.

[43] Tianyi Wang, Yang Chen, Zengbin Zhang, Tianyin Xu, Long Jin, Pan Hui, Beixing Deng, and Xing Li. Understanding graph sampling algorithms for social network analysis. In *Distributed Computing Systems Workshops (ICDCSW), 2011 31st International Conference on*, pages 123–128. IEEE, 2011.

[44] D.H. Wolpert and W. Macready. Using self-dissimilarity to quantify complexity. *Complexity*, 12(3):77–85, 2007.