Local Graph Sparsification for Scalable Clustering

Venu Satuluri, Srinivasan Parthasarathy and Yiye Ruan
Dept. of Computer Science and Engineering
The Ohio State University
{satuluri,srini,ruan}@cse.ohio-state.edu

ABSTRACT

In this paper we look at how to *sparsify* a graph i.e. how to reduce the edgeset while keeping the nodes intact, so as to enable faster graph clustering without sacrificing quality. The main idea behind our approach is to preferentially retain the edges that are likely to be part of the same cluster. We propose to rank edges using a simple similarity-based heuristic that we efficiently compute by comparing the minhash signatures of the nodes incident to the edge. For each node, we select the top few edges to be retained in the sparsified graph. Extensive empirical results on several real networks and using four state-of-the-art graph clustering and community discovery algorithms reveal that our proposed approach realizes excellent speedups (often in the range 10-50), with little or no deterioration in the quality of the resulting clusters. In fact, for at least two of the four clustering algorithms, our sparsification consistently enables higher clustering accuracies.

Categories and Subject Descriptors

G.2.2 [**Graph Theory**]: Graph Algorithms; I.5.3 [**Pattern Recognition**]: Clustering

General Terms

Algorithms,Performance

Keywords

Graph Clustering, Graph Sparsification, Minwise Hashing

1. INTRODUCTION

Many real world problems can be modeled as complex interaction networks or graphs. Graph clustering is one of the fundamental primitives for analyzing graphs, with applications as diverse as protein complex detection, community discovery in social networks, optimizations for keyword search [10] and many others. Intuitively, graph clus-

tering seeks to organize the nodes of the graph into clusters such that the nodes belonging to the same cluster are more strongly connected to each other, compared to the nodes in the rest of the graph. A number of algorithms for graph clustering have been previously proposed, including spectral methods [34], weighted kernel k-means [11], local spectral methods [36], stochastic flow methods [38, 31], max-flow based methods [21], and Kernighan-Lin style approaches [17]. However, extracting such structure in a scalable fashion remains a major challenge [12]. Large-scale graphs are especially common in the WWW domain, with the rapid growth of many services such as Facebook, Twitter, Orkut and so on, which can be very naturally analyzed using a graph representation [19, 26].

A general purpose solution to scale various data mining algorithms that is often considered in the literature is that of sampling. The basic premise is that operating on a smaller scale version of the problem will realize approximately similar results to processing the entire dataset, at a fraction of the execution time. This premise has been exploited for various machine learning problems and data mining problems in the past [28]. In a similar spirit, in this paper we attempt to reduce the size of a graph in order to scale up community discovery/graph clustering algorithms. However, we aim to retain all the nodes in the graph and only filter out the edges in the graph - in other words, *sparsify* the graph. Our goal is to sparsify the graph in such a way that the cluster structure is retained or even enhanced in the sparsified graph. We are not proposing a new graph clustering algorithm; rather we aim to transform the graph in such a way that will help almost any existing graph clustering algorithm, in terms of speed and at little to no cost in accuracy. Our sparsified graphs can also help in visualizing the cluster structure of the original graph.

An example illustrating the results that can be obtained using our proposed sparsification algorithm is shown in Figure 1. The original graph (Figure 1(a)) consists of 30 nodes and 214 edges, with the vertices belonging to 3 clusters (indicated by the color coding). The result of sparsifying this graph using our proposed sparsification algorithm is shown in Figure 1(b). Note that the sparsified graph only contains 64 of the original 214 edges, and the cluster structure of the graph is also much clearer in this new sparsified graph.

The main idea behind our sparsification algorithm is to preferentially retain the edges that are likely to be part of the same cluster. We use a similarity-based sparsification heuristic, according to which edges that connect nodes which have many common neighbors are retained. Using a global

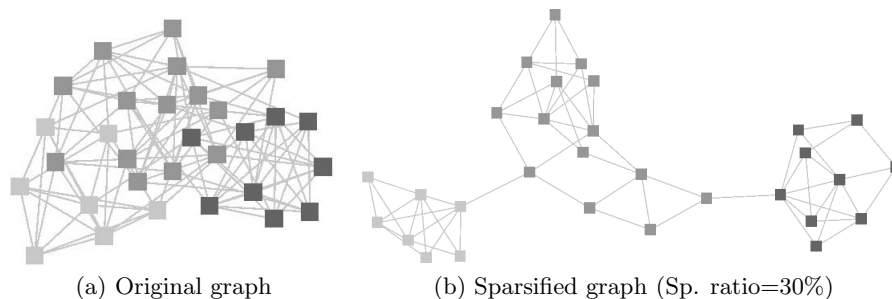(a) Original graph   (b) Sparsified graph (Sp. ratio=30%)

Figure 1: Proposed local sparsification strategy applied on an example graph with 30 vertices and 3 clusters.

similarity threshold for selecting which edges to retain in a graph, however, is problematic since different clusters may have different densities, and using a global threshold may choose far many more edges in the denser clusters, disconnecting the less dense clusters. To overcome this problem, we propose a *local* sparsification algorithm which chooses a few top edges per node, and thereby avoids this problem by using a locally appropriate threshold for including the edges in the sparsified graph. Local sparsification also ensures that all nodes in the graph are covered, i.e. there is at least one edge incident on each node in the sparsified graph. Specifically, we retain d^e edges for a node of degree d, where e is a parameter that helps control the overall sparsification ratio of the result. We analyze the characteristics of the resulting sparsified graph in terms of the degree distribution and the final sparsification ratio in terms of the parameter e.

Since exact similarity computation between each pair of connected nodes is very expensive, we efficiently approximate the similarity by hashing via *minwise independent permutations* [6] (minwise hashing). Estimating similarity for all pairs of connected nodes in the graph using minhashing is linear in the number of edges in the graph, allowing us to sparsify the graph very quickly.

We evaluate the performance of our proposed local sparsification algorithm (named **L-Spar**) on several real networks such as Wikipedia, Twitter, Orkut, Flickr and also biological networks with ground truth. Our main evaluation method is to compare the quality of clusters obtained from the original graph with the quality of those obtained from the sparsified graph, as well as compare the respective execution times and balance of the resulting cluster sizes. In terms of baselines, we compare against random sampling [16, 2, 1], sampling based on the ForestFire model [22] and the global similarity sparsification (named **G-Spar**) that we initially propose. We examine the performances of the different sparsifications when coupled with different state-of-the-art clustering algorithms (we use Metis [17], Metis+MQI [21], MLR-MCL [31] and Graclus [11]). We summarize our key findings:

- For different networks and different clustering algorithms, L-Spar sparsification enables clustering that is, in terms of quality, comparable to the clustering result obtained from the original graph, despite containing far fewer edges (typically 10-20%), and significantly superior to the result obtained by clustering the baseline sparsifications. Indeed, clustering our sparsified graph often improves upon the clustering obtained from the original graph - for example, the F-score of

Metis improves by 50% when clustering the sparsified Wiki graph. Similarly, Metis+MQI outputs a giant core cluster containing 85% of the nodes when clustering the original Flickr graph, while the clustering result obtained using the sparsified graph is much more meaningful.

- Clustering the L-Spar sparsified graph as well as the sparsification itself are extremely fast. Metis for instance obtains a 52x speedup (including clustering and sparsification times) on the Wiki graph.MLR-MCL similarly obtains speedups in the range of 20x.

- Examples of the kinds of edges that are retained and those that are discarded in the sparsification process confirm that the algorithm is able to discern noisier/weaker connections from the stronger/semantically closer connections.

- We also systematically vary the average degree and "mixing"-ness of the clusters of a synthetic graph generator [20] and find that our method becomes increasingly effective with higher average degrees as well as with higher mixing parameters.

2. RELATED WORK

Previous researchers have developed approaches for sparsifying graphs, but not always with the same goals as ours.

In theoretical computer science, graph sparsification approaches that approximately preserve (unnormalized) edge cuts have been developed, such as random sampling [16], sampling in proportion to edge connectivities [4] and sampling in proportion to the effective resistance of an edge [35]. The last scheme - which has the best theoretical properties - favors the retention of edges connecting nodes with few short paths, as such edges have high effective resistance. However, this is quite different from our own approach, where instead we discard such edges, since they distract from the overall cluster structure of the graph. Furthermore, calculating effective resistances requires invoking a specific linear system solver [36], which runs in time at least $O(m \log^{15} n)$ - meaning that the overall approach is too time-consuming to be practical for large-scale graphs.[1]

The statistical physics community has studied the problems of *edge filtering* [37] and *complex network backbone detection* [14]. This line of work also does not aim to preserve

---

[1]n and m denote the number of nodes and number of edges in the graph respectively.

the cluster structure, instead looking to identify the most interesting edges, or construct spanning-tree-like network backbones. Furthermore, these approaches are inapplicable in the case of unweighted networks.

There has also been work on *graph sampling* [18, 22] where the goal is to produce a graph with both fewer edges as well as fewer nodes than the original, and which preserves some properties of interest of the original graph such as the degree distribution, eigen-value distribution etc. The fact that these approaches select only a small subset of the nodes in the original graph makes them unsuitable to our task, since we aim to cluster the (entire) nodeset of the original graph. Recently, Maiya and Berger-Wolf [25] propose to find representative subgraphs that preserve community structure based on algorithms for building expander-like subgraphs of the original graph. The idea here is that sampling nodes with good expansion will tend to include representatives from the different communities in the graph. Since partitioning the sampled subgraph will only give us cluster labels for the nodes included in the subgraph, they propose to assign cluster labels to the unsampled nodes of the original graph by using collective inference algorithms [24]. However, collective inference itself utilizes the original graph in its entirety, and becomes a scalability bottleneck as one operates on bigger graphs.

Sparsifying matrices (which are equivalent representations for graphs) has also been studied [1, 2]. These approaches, in the absence of edge weights, boil down to random sampling of edges - an approach we evaluate in our experiments.

The literature on clustering algorithms for graphs is too vast for us to review here (see [12] for a comprehensive survey), but we point out that the contributions of our paper should be seen as *complementing* existing graph clustering algorithms.

3. SIMILARITY SPARSIFICATION

We aim to design a method for sparsifying a graph such that:

1. Clustering the sparsified graph should be much faster than clustering the original graph.

2. The accuracy of the clusters obtained from the sparsified graph is close to the accuracy of the clusters obtained from the original graph.

3. The sparsification method itself should be fast, so that it is applicable to the massive networks where it is most beneficial.

Our main approach to sparsification is to preferentially retain intra-cluster edges in the graph compared to inter-cluster edges, so that the cluster structure of the original graph is preserved in the sparsified graph. Of course, if a graph possesses cluster structure at all, then the majority of the edges in the graph will be contained inside clusters, and so to achieve any significant sparsification, one needs to inevitably discard some intra-cluster edges. Nevertheless, as long as a greater fraction of the inter-cluster edges are discarded, we should expect to still be able to recover the clusters from the original graph. The critical problem here, then is to be able to *efficiently* identify edges that are more likely to be within a cluster rather than between clusters.

Prior work has most commonly used various edge centrality measures in order to identify edges in the sparse parts

of the graph. The edge betweenness centrality [27] of an edge (i, j), for example, is proportional to the number of shortest paths between any two vertices in the graph that pass through the edge (i, j). Edges with high betweenness centrality are "bottlenecks" in the graph, and are therefore highly likely to be the inter-cluster edges. However the key drawback of edge betweenness centrality is that it is prohibitively expensive, requiring $O(mn)$ time [27] (m is the number of edges and n is the number of nodes in the graph).

For this reason, we propose a simpler heuristic for identifying edges in sparse parts of the graph.

Similarity-based Sparsification heuristic

An edge (i, j) is likely to (not) lie within a cluster if the vertices i and j have adjacency lists with high (low) overlap.

The main intuition here is that the greater the number of common neighbors between two vertices i and j connected by an edge, the more likely it is that i and j belong to the same cluster. Another way of looking at it is that an edge that is a part of many triangles is probably in a dense region i.e. a cluster. We use the Jaccard measure to quantify the overlap between adjacency lists. Let $Adj(i)$ be the adjacency list of i, and $Adj(j)$ be the adjacency list of j. For simplicity, we will refer to the similarity between $Adj(i)$ and $Adj(j)$ as the similarity between i and j itself.

$$Sim(i, j) = \frac{|Adj(i) \cap Adj(j)|}{|Adj(i) \cup Adj(j)|} \qquad (1)$$

Global Sparsification

Based on the above heuristic, a simple recipe for sparsifying a graph is given in Algorithm 1. For each edge in the input graph, we calculate the similarity of its end points. We then sort all the edges by their similarities, and return the graph with the top $s\%$ of all the edges in the graph (s is the sparsification parameter that can be specified by the user). Note that selecting the top $s\%$ of all edges is the same as setting a similarity threshold (that applies to all edges) for inclusion in the sparsified graph.

Algorithm 1 Global Sparsification Algorithm

Input: Graph $G = (V, E)$, Sparsification ratio s

$G_{sparse} \leftarrow \emptyset$
for each edge $e=(i,j)$ in E **do**
  $e.sim = Sim(i, j)$ according to Eqn 1
end for

Sort all edges in E by $e.sim$
Add the top $s\%$ edges to G_{sparse}

return G_{sparse}

However, the approach given in Algorithm 1 has a critical flaw. It treats all edges in the graph equally, i.e. it uses a global threshold (since it sorts all the edges in the graph), which is not appropriate when different clusters have different densities. For example, consider the graph in Figure 2(a). Here, the vertices $\{1, 2, 3, 4, 5, 6\}$ form a dense cluster, while the vertices $\{7, 8, 9, 10\}$ form a less dense cluster. The sparsified graph using Algorithm 1 and selecting the top 15 (out of 22) edges in the graph is shown in Figure 2(b). As can be seen, all the edges of the first cluster have been retained, while the second cluster consisting of $\{7, 8, 9, 10\}$ is completely empty, since all the edges in the first cluster have higher similarity than any edge in the sec-
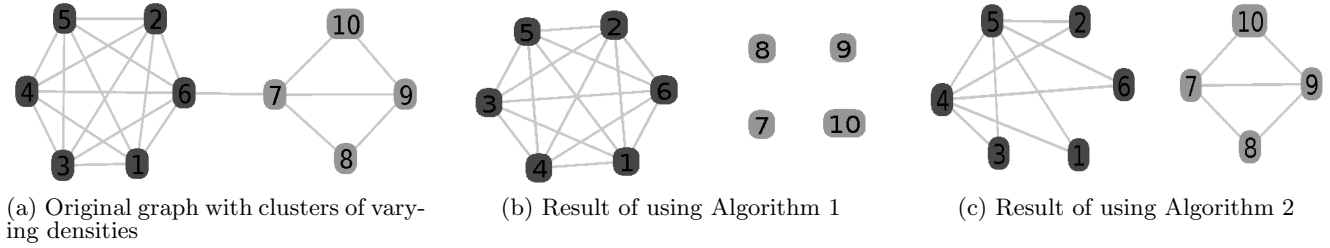
(a) Original graph with clusters of varying densities

(b) Result of using Algorithm 1

(c) Result of using Algorithm 2

Figure 2: Global vs. local sparsifications

ond cluster. Therefore, clustering the resulting graph will be able to recover cluster 1, but not cluster 2.

Therefore, we look to devise an alternative sparsification strategy that still relies on the similarity-based sparsification heuristic, but which can also handle situations when different clusters have different densities.

Local Sparsification

We solve the problem with Algorithm 1 described above, by avoiding the need to set a global threshold. Instead, for each node i, with degree d_i, we pick the top $f(d_i) = d_i^e$ edges incident to i, ranked according to similarity (Eqn. 1). Sorting and thresholding the edges of each node separately allows the sparsification to adapt to the densities in that specific part of the graph; furthermore, this procedure ensures that we pick at least one edge incident to each node. Here e ($e < 1$) is the local sparsification exponent that affects the global sparsification ratio, with smaller values of e resulting in a sparser final graph. The full algorithm is given in Algorithm 2.

Algorithm 2 Local Sparsification Algorithm

Input: Graph $G = (V, E)$, Local Sparsification exponent e
Output: Sparsified graph G_{sparse}

$G_{sparse} \leftarrow \emptyset$
for each node i in V **do**
    Let d_i be the degree of i
    Let E_i be the set of edges incident to i
    for each edge $e=(i,j)$ in E_i **do**
        $e.sim = Sim(i, j)$ according to Eqn. 1
    end for
    Sort all edges in E_i by $e.sim$
    Add top d_i^e edges to G_{sparse}
end for

return G_{sparse}

The result of sparsifying the example graph (14 out of 22 edges) in Figure 2(a) using the local sparsification algorithm is shown in Figure 2(c). This graph reflects the cluster structure of the original graph much better than Figure 2(b), since the cluster {7,8,9,10} is still recoverable from the sparsified graph, unlike the one that was obtained by global sparsification.

Discussion

For a node of degree d, what is a good choice of the function $f(d)$ that tells us the right number of edges to retain? We want to retain at least one edge per node, and so we must have $f > 0$ and also $f(d) \leq d, \forall d$, since one cannot retain more edges than are already present.

We first observe that for hub nodes (i.e. nodes with high degree), a higher fraction of the incident edges tend to be inter-cluster edges, since such nodes typically tend to straddle multiple clusters. For this reason, we would prefer to sparsify nodes with higher degree more aggressively than nodes with lower degree. This implies that we want a strictly concave function f, which rules out linear functions of the sort $f(d) = c * d, c < 1$. Two prominent choices for strictly concave functions are $f(d) = \log d$ and $f(d) = d^e, e < 1$.

The advantage with $f(d) = d^e, e < 1$ is that we can control the extent of the sparsification easily using the exponent e, while retaining the concave property of the function. Furthermore, we have the following in the case of $f(d) = d^e$.

PROPOSITION 1. *For input graphs with a power-law degree distribution with exponent α, the locally sparsified graphs obtained using Algorithm 2 also have a power-law degree distribution with exponent $\frac{\alpha+e-1}{e}$.*

PROOF. Let D_{orig} and D_{sparse} be random variables for the degree of the original and the sparsified graphs respectively.

Since D_{orig} follows a power-law with exponent α, we have $p(D_{orig} = d) = Cd^{-\alpha}$ and the complementary CDF is given by $P(D_{orig} > d) = Cd^{1-\alpha}$ (approximating the discrete power-law distribution with a continuous one, as is common [8]).

From Algorithm 2, $D_{sparse} = D_{orig}^e$. Then we have

$$
\begin{aligned}
P(D_{sparse} > d) &= P(D_{orig}^e > d) = P(D_{orig} > d^{1/e}) \\
&= C\left(d^{1/e}\right)^{1-\alpha} = Cd^{1-\frac{\alpha+e-1}{e}}
\end{aligned}
$$

Hence, D_{sparse} follows a power-law with exponent $\frac{\alpha+e-1}{e}$.

Let the cut-off parameter for D_{orig} be d_{cut} (i.e. the power law distribution does not hold below d_{cut} [8]), then the corresponding cut-off for D_{sparse} will be d_{cut}^e. □

We can use Proposition 1 to prove the next one.

PROPOSITION 2. *The sparsification ratio (i.e. the number of edges in the sparsified graph $|E_{sparse}|$, versus the original graph $|E|$), for the power law part of the degree distribution is at most $\frac{\alpha-2}{\alpha-e-1}$.*

PROOF. Notice that the sparsification ratio is the same as the ratio of the expected degree on the sparse graph versus the expected degree on the original graph. We have, from the expressions for the means of power-laws [8]

$$
E[D_{orig}] = \frac{\alpha-1}{\alpha-2}d_{cut}
$$

$$E[D_{sparse}] = \frac{\frac{\alpha+e-1}{e} - 1}{\frac{\alpha+e-1}{e} - 2} \cdot d_{cut}^e$$

Then the sparsification ratio is

$$\frac{\frac{\frac{\alpha+e-1}{e}-1}{\frac{\alpha+e-1}{e}-2}}{\frac{\alpha-1}{\alpha-2}} \cdot d_{cut}^{e-1} \leq \frac{\frac{\frac{\alpha+e-1}{e}-1}{\frac{\alpha+e-1}{e}-2}}{\frac{\alpha-1}{\alpha-2}} = \frac{\alpha-2}{\alpha-e-1}$$

For a fixed e and a graph with known α, this can be calculated in advance of the sparsification. $\square$

Many real graphs are known to follow power-laws in the range $2 < \alpha < 3$, and assuming an $\alpha = 2.1$ and $e = 0.5$, the sparsification ratio will be less than 17%, according to the calculations above. Higher values of α (i.e. steeper power-law graphs) yield higher sparsification ratios, as do higher values of the exponent e (as expected).

Time complexity

The main component in the running time for both the Global and Local Sparsification is the computation of the similarities on each edge according to Eqn 1. Assuming the adjacency lists for each node are pre-sorted, intersecting the adjacency lists of two nodes i and j with degrees d_i and d_j takes number of operations proportional to $d_i + d_j$. Since a node of degree d_i requires d_i intersections, the total number of operations is proportional to $\sum_i d_i^2$. This is at least $n \cdot d_{avg}^2$ (by Jensen's inequality), where d_{avg} is the average degree of the graph, which is prohibitively large for most graphs.

We suggest a faster, approximate method next.

3.1 Minwise Hashing for Fast Similarity

Hashing by minwise independent permutations, or minwise hashing, is a popular technique for efficiently approximating the Jaccard similarity between two sets, first introduced by Broder et al. [6]. Minwise hashing has been used previously in problems such as compressing web graphs [7], discovery of dense subgraphs [13] and local triangle counting [3]. Our use of minwise hashing is largely orthogonal to these existing efforts as it has a completely different goal – as a simple mechanism to speed up graph sparsification, and eventually to scale up complete graph clustering.

Given two sets A and B, and a permutation π on the space of the universal set, randomly chosen from a family of minwise independent permutations [6], we have that the first element of set A under the permutation π is equal to the first element of set B under π with probability equal to their (Jaccard) similarity :

$$Pr(min(\pi(A)) = min(\pi(B))) = \frac{|A \cap B|}{|A \cup B|} \qquad (2)$$

Based on Equation 2, a simple estimator for $sim(A,B)$ is $I[min(\pi(A)) = min(\pi(B))]$, where $I[x]$ is the indicator variable i.e. $I[x] = 1$ if x is true and 0 otherwise.

PROPOSITION 3. $I[min(\pi(A)) = min(\pi(B))]$ is an unbiased estimator for $sim(A,B)$, with variance $sim(A,B) * (1 - sim(A,B))$

PROOF. The estimator in question is a Bernoulli random variable, with probability of success $sim(A,B)$. The proposition follows. $\square$

The variance of the above estimator can be reduced by taking multiple independent minwise permutations. Let $\pi_i, i =$

$1 : k$ be k independent minwise permutations on the universal set. Let $mh_i(A) = min(\pi_i(A))$ for any set A, i.e. $mh_i(A)$ represents the minimum element of A under the permutation π_i (i.e. it is the ith "minhash" of A). We can construct a length-k signature for a set A consisting of the k minhashes of A in order - note that the signature is itself not a set, and the order is necessary for the similarity estimation.

A better estimator for $sim(A,B)$ then is

$$\hat{sim}(A,B)_k = \frac{1}{k}(\sum_{i=1}^{k} I[mh_i(A) = mh_i(B)]) \qquad (3)$$

PROPOSITION 4. $\hat{sim}(A,B)_k$ is an unbiased estimator for $sim(A,B)$, with the variance inversely proportional to k.

PROOF. $\hat{sim}(A,B)_k$ is the average of k unbiased estimators, and therefore by the linearity of expectation is itself unbiased. It is easy to show the variance is $\frac{sim(A,B)*(1-sim(A,B))}{k}$, and thus inversely proportional to k. $\square$

For the approximation of $sim(A,B)$ using the estimator in Eqn 3 to be fast, we still need an efficient way to generate random permutations. Since the space of all permutations is exponentially large, drawing a permutation uniformaly at random from this space can be expensive. For this reason, it is typical to use *approximate* minwise independent permutations such as *linear permutations* [5]. A linear permutation is specified by a triplet (a, b, P) where P is a large prime number and a is a integer drawn uniformly at random from [1,P-1] and b is an integer drawn uniformly at random from [0,P-1]; the permutation of an input $i \in [0, P-1]$ is given by $\pi(i) = (a*i + b)\%P$. Multiple linear permutations can be generated by generating random pairs of (a,b). Since all we care about is only the minimum element under the permutation π, minhashing a set using linear permutations simply involves one pass over the elements of the set, where we maintain a variable that keeps track of the minimum element seen thus far in the scan.

Local Sparsification using Minwise Hashing

We first generate k linear permutations, by generating k triplets (a, b, P). Next, we compute a length k signature for each node by minhashing each node k times - more precisely, we minhash the adjacency list of each node in the graph k times - and fill up a hashtable of size $n*k$. For an edge (i, j), we compare the signatures of i and j, minhash-by-minhash, and count the number of matching minhashes. Since the similarity of an edge is directly proportional to the number of matches, we can sort the edges incident to a node i by the number of matches of each edge. Furthermore, since the number of matches is an integer between 0 and k, one can use counting sort to sort the edges incident on a node in linear time. After the sort, we pick the top d_i^e edges (d_i is the degree of node i) for inclusion in the sparsified graph.

The implementation of global sparsification using minwise hashing is similar and is omitted.

Time complexity

Local sparsification using minwise hashing is extremely efficient. Generating one minwise hash of a set using linear permutation requires one scan of the set, therefore generating a minwise hash of a single node i takes $O(d_i)$ time. The time to generate one minwise hash for all nodes takes time $O(m)$, where m is the number of edges in the graph, and generating k minwise hashes for all nodes in the graph takes

Dataset	Vertices	Edges	d_{avg}	$|C|(size_{avg})$
BioGrid	5,964	192,851	64.7	700(9)
DIP	4,741	15,147	6.4	700(7)
Human	11,750	63,419	10.8	900(13)
Wiki	1,129,060	53,021,551	93.9	10000(113)
Orkut	3,072,626	117,185,083	76.3	15000(205)
Twitter	146,170	83,271,147	1139.4	1500(97)
Flickr	31,019	520,040	33.5	300(103)

Table 1: Dataset details. $|C|$ is the number of clusters and $size_{avg}$ the average cluster size.

$O(km)$ time. Estimation of similarity for an edge (i, j) only takes time $O(k)$, since we only need to compare the length-k signatures of i and j. The estimation of similarity for all edges takes time $O(km)$ time. Sorting the edges incident on each node by similarity can be done in linear time using counting sort, since we only sort the number of matches which lie in $\{0, \ldots, k\}$. Hence, sorting the edges for all the nodes requires time $O(m)$. In sum, the time complexity for local (or global) sparsification using minwise hashing is $O(km)$ i.e. linear in the number of edges.

4. EMPIRICAL EVALUATION

4.1 Datasets

We perform experiments on seven real-world networks, including information networks, social networks and protein-protein interaction networks (also see Table 1):

1. **Yeast-BioGrid (BioGrid):** This is a protein-protein interaction (PPI) network of Yeast, obtained from the BioGrid databaseThis network has a relatively high average degree of 65, with many noisy interactions.

2. **Yeast-DIP (DIP):** This is also a PPI network of Yeast, obtained from the Database of Interacting Proteins (DIP)The interactions in this network are more carefully derived (compared to BioGrid), resulting in a much lower average degree of 6.

3. **Human-PPI (Human):** The third network is a Human PPI network, obtained from iRefIndex.

4. **Wikipedia (Wiki):** This is an undirected version of the graph of hyperlinks between Wikipedia articles (from Jan–2008). The original downloaded corpus had nearly 12 million articles, but a lot of these were insignificant or noisy articles which we removed to obtain the final graph with 1.12M nodes and 53M edges.

5. **Orkut:** This is an anonymized friendship network crawled from Orkut [26]. This is our largest dataset consisting of more than 3M nodes and 117M edges.

6. **Twitter:** This is the Twitter follower-followee network collected by Kwak et al. [19]. We retain only users with at least 1000 followers, obtaining a network of 146,170 users, and we retain only edges (i, j) such that both i follows j and j follows i. The resulting number of edges in this graph is still very high - around 83M, with an average degree of 1140 per user.

7. **Flickr tags (Flickr):** We downloaded the tag information associated with ˜6M photos from Flickr. We

then built a tag network using the tags that were used by at least 5 distinct users, placing an edge between two tags if they were used together in at least 50 photos. The resulting network has 31019 tags with ˜520K edges, and an average degree of 33.5 per tag.

Ground Truth

For evaluating the clustering results of the two Yeast PPI networks, we used the CYC2008 [29] protein complex data as the gold standard for ground truth validation. Note that only around 30% of the proteins in the PPI datasets have associated protein complex information, which means that the precision of any clustering result is at most 30%. For evaluating the results of the Human PPI dataset we used the CORUM protein complex data [30], which again has annotations for only 15% of the total proteins in the graph. For Wiki we prepared ground truth from the categories assigned at the bottom of the page by the editors. We removed many noisy categories; the final number of categories was 17950, covering around 65% of the articles in the dataset.

4.2 Baselines

We compare the following sparsification algorithms:

1. Our local similarity-based sparsification algorithm using minwise hashing (which we refer to as **L-Spar**, short for Local-Sparsifier, henceforth). It takes two parameters, k referring to the number of minhashes, and e referring to the sparsification exponent.

2. The global similarity-based sparsification algorithm using minwise hashing, referred to as **G-Spar** henceforth.

3. Random Edge Sampling [22, 16, 1, 2], that selects $r\%$ of the edges from the graph uniformly at random.

4. ForestFire [22] that selects a seed node at random and recursively burns edges and nodes in its vicinity, with burn probability $r\%$. We repeatedly initiate fires choosing an as-yet unburned seed node so as to cover most of the nodes in the graph. [2]

We set the parameters for the baselines 2-4 above so as to achieve the same sparsification ratio as that achieved using L-Spar. We use a default sparsification exponent of $e = 0.5$ (unless otherwise mentioned), and default number of minhashes $k=30$.

Additional experiments comparing with the sampling approach of Maiya and Berger-Wolf [25] are presented elsewhere [33].

4.3 Evaluation method

Our primary quantitative evaluation method is to cluster the original and the sparsified graphs, and to assess the quality of the clusterings either w.r.t. the ground truth or w.r.t. the structure of the (original) graph. We use four state-of-the-art algorithms for clustering the original and sparsified graphs - Metis [17], Metis+MQI [21], MLR-MCL [31] and

---

[2]Note that a burned node may still remain a singleton if none of its edges get burned in the probabilistic burning process.

Graclus [11].[3] All the experiments were performed on a PC with a Quad Core Intel i5 CPU, each with 3.2 GHz, 16 GB RAM and running Linux. The output number of clusters for each network, along with the average cluster size, is provided in Table 1. [4] We used a small average cluster size for the protein networks since most protein complexes are in the size range of 5-15. The average cluster sizes for the other networks is around 100, which is on average the right size for communities in such networks [23] (in the case of Orkut, we had to reduce the number of clusters as Metis and Metis+MQI would run out of memory otherwise).

When we have ground truth for a network, we measure cluster quality using the average F-score of the predicted clusters, which we compute as follows. The F-score of a predicted cluster w.r.t. a ground truth cluster is the harmonic mean of its precision and recall. The F-score of a predicted cluster by itself is its best possible F-score with a ground truth cluster (i.e. with the cluster that it approximates the best). The average F-score of all the predicted clusters is the weighted average of all F-scores, where the cluster sizes are the weights. We denote it as a percentage between 0 and 100, with higher being better.

In the absence of ground truth, we use average conductance. The conductance of a cluster ϕ is the number of edges leaving the cluster, divided by either the total number of edges originating in the cluster or of the rest of the graph, whichever is smaller (see [21, 11]). Conductance always lies between 0 and 1, with lower conductances being better. The average conductance ϕ_{avg} then is simply the average of the conductances of all the clusters. There are two important points to note about the way we measure conductances: (i) For the clusters obtained from the sparsified graph, we cannot simply measure the conductance using the very same sparsified graph, since that would tell us nothing about how well the sparsified graph retained the cluster structure in the original graph. Therefore, we report the conductances of the clusters obtained from the sparsified graphs also using the *structure of the* **original** *graph.* (ii) The G-Spar, RandomEdge and ForestFire (but not L-Spar) often isolate a percentage of nodes in the graph in their sparsification. We *do not* include the contribution to the final average conductance arising from such singleton nodes. Note that this biases the comparison based on conductance against the results from the L-Spar and the original graph and in favor of the baselines.

Since low conductances can often be achieved by very imbalanced clusterings [23], we also report on the balance of a clustering arrangement using the *coefficient of variation* c_v of the cluster sizes, which is the ratio of the standard deviation of the cluster sizes to the average cluster size (i.e. $c_v = \sigma/\mu$). Therefore, a lower c_v represents a more balanced clustering.

---

[3]We downloaded the softwares from the respective authors' webpages, except for Metis+MQI which we re-implemented using Metis and hipr as black-box modules. The versions are: Metis 4.0, Graclus 1.2, MLR-MCL 1.1 and hipr 3.4. We thank Kevin Lang for help with Metis+MQI.

[4]The number of clusters is only indirectly specifiable in MLR-MCL, and hence the final number of clusters in the case of MLR-MCL varied slightly from that reported in Table 1.

4.4 Results

The results obtained using the various sparsification approaches as well as the original graph and using the different clustering algorithms are presented in Table 2. The sparsification ratios obtained using L-Spar by setting $e = 0.5$ vary for different networks depending on their specific degree distributions, but they never exceed 0.2 for the five biggest graphs - this means that for those graphs, the sparsified graphs contain at most a fifth of the edges in the original graph. The speedups are w.r.t. clustering the original graph, and take into account both sparsification as well as clustering times.

Let us first consider the results using Metis. For all four datasets with ground truth, clustering the L-Spar graph actually gives better results than clustering the original graph. This suggests that the removal of noisy edges by L-Spar helps Metis discover clusters that it was previously unable to. The Wiki graph provides a particularly stark example, where the F-score improves from 12.34 to 18.47, while also executing 52x times faster. Similarly on Orkut, the average conductance for the clusters discovered from the L-Spar graph is 0.76 as compared to 0.85 for the clustering from the original graph. On Flickr, the G-Spar clustering has a much lower conductance (0.71) than either the original or L-Spar, but on the flip side, we found that G-Spar introduced around 30% singletons in the sparsified graph. The speedups are 36x, 6x and 52x for our three biggest graphs (Orkut, Twitter and Wiki).

Looking at the results using MLR-MCL, we can see that L-Spar enables clusterings that are of comparable quality to clustering the original graph, and are much better compared to the other sparsifications. On Wiki, for instance, L-Spar enables an F-score of 19.3, far superior to that obtained with the other baselines, and quite close to the original F-score of 20.2. On Orkut, Flickr and Twitter - datasets where we have no ground truth - clustering the L-Spar graph results in clusters with better ϕ_{avg} (on the original graph), and also better balance (i.e. lower c_v). Furthermore, L-Spar followed by clustering is at least 22x faster on our three biggest graphs (Orkut, Twitter, Wiki), bringing the total times down from the order of hours down to the order of minutes.

Coming to the results obtained using Metis+MQI, we see results that are of a similar flavor to those obtained using MLR-MCL. On the four graphs with ground truth, the accuracy of clustering either the original graph or the L-Spar graph are very much comparable, and the other three sparsifications fare worse in comparison. On Orkut and Twitter, we similarly find the results from clustering either the original graph or the L-Spar graph comparable. On the Flickr dataset, clustering the original graph results in a low ϕ_{avg} of 0.55, but only at the cost of high imbalance (c_v=14.1), with 85% of the nodes being placed in one giant cluster. The clusters from the L-Spar graph are much more balanced (c_v=1.0). Some examples of clusters obtained from the L-Spar graph that were previously merged together in the giant cluster are :{astronomy, telescope, astrophotography, griffithobservatory, solarsystem}, {arcade, atlanticcity, ac, poker, slots}, {historicdistrict, staugustine, staugustineflorida, castillodesanmarcos}. Looking at the speedups, we find a counter-intuitive result - Metis+MQI executes slower on the L-Spar graph for Wiki and Orkut. The explanation is as follows. Metis+MQI recursively bipartitions the graph, operating on the currently biggest partition at each step.

The running time for Metis+MQI therefore depends a lot on the step at which Metis+MQI discovers a nearly balanced bipartition (since that will halve the size of the partition on which Metis+MQI subsequently needs to operate on). On Wiki, we found that a balanced bipartition is found at ˜150th step on the original graph, versus at ˜1300th step on the L-Spar graph, causing almost all the difference in speed we see. However, the balance of the clustering arrangement (given by c_v) from L-Spar at the end of all the recursive bipartitions is at least as good as the balance of the clustering of the original graph (much better for some graphs, as the example of Flickr above showed). Note that we still obtain a 14x speedup on the ˜80M edge Twitter graph.

In the results for Graclus, we see that L-Spar enables more accurate clusterings than the original graph on the three biological datasets [5]. For Flickr, we note that ϕ_{avg} for the three baseline sparsifications are close to 0.99 if we include the contribution of singletons, and the shown ϕ_{avg}s are comparable to those obtained using L-Spar only because their contributions were not included. On Twitter, we find that L-Spar enables ϕ_{avg} that is comparable to clustering the original graph, with better balance and 5x faster.

We would like mention to an interesting cross-cutting result beyond the results showed in Table 2. The time taken to cluster the L-Spar and G-Spar sparsified graphs is typically smaller than the time taken to cluster the other baseline sparsifications (RandomEdge and ForestFire), although the reverse trend holds true for the sparsification itself. For a representative example, on the Wiki dataset, Metis requires 80 seconds to cluster the L-Spar sparsified graph and 8 seconds to cluster the G-Spar one, compared to 940 seconds for RandomEdge and 1040 seconds for ForestFire. This difference in clustering times, despite the fact that all the sparsified graphs contain almost the *same number of edges*, is because clustering algorithms generally tend to execute faster on graphs with clearer cluster structures.

4.5 Examining L-Spar sparsification in depth

Let us examine L-Spar sparsification in depth. The BioGrid protein interaction network consists mainly of edges derived from high-throughput interaction detection methods such as Yeast Two Hybrid analysis [9], which are well known to detect many false positive interactions. It is especially interesting that for this network, *all* four clustering algorithms enjoy better clustering accuracies on the L-Spar sparsified graph compared to the original graph, suggesting that the sparsification does remove many spurious interactions.

We provide some examples from the Wiki, Twitter and Flickr graphs in Table 3 that shine some light on the kinds of edges that are removed vis-a-vis the kinds that are retained. For the three Wikipedia examples of *machine learning, graph* and *Moby Dick*, the retained edges are clearly semantically closer to the example node, while the discarded edges are noise, arguably. For Twitter, we highlight the neighborhoods of Twitter founder Jack Dorsey, champion cyclist Lance Armstrong and conservative blogger Michelle Malkin. The retained neighbors for Dorsey are either high-ranking employees at Twitter (Stone, Williams, Goldman) or people with strong Silicon Valley connections (Dash, Lacy), while the discarded neighbors are more peripherally con-

nected (Jet Blue Airways, Parul Sharma). The retained neighbors for Lance Armstrong are all professional bicyclists, while the discarded neighbors come from other backgrounds. For Michelle Malkin, the retained neighbors are Republican governors Bobby Jindal, Rick Perry and other conservatives or media personalities; clearly the discarded neighbors do not share as close a connection. The examples from Flickr similarly show tags that are semantically closer to the highlighted tag being retained, with the noisier tags being discarded.

Trend of speedup and f-score as e changes

Figures 3(a) and 3(b) report the trends of F-score and speedup on the Wiki dataset as e changes, using Metis. It is interesting that even with e=0.3, and retaining only 7% of the edges in the graph, Metis achieves an F-score of 17.73 on the sparsified graph, which is significantly better than the 12.34 on the original graph. The F-scores gradually increase with increasing e. Coming to the speedups, the biggest speedups are observed for smaller values for e - with an 81x speedup for e=0.3 (242x considering only the clustering times) - and the speedups gradually reduce to 13x for e=0.7.

Figures 3(c) and 3(d) report the corresponding trends for MLR-MCL and are quite similar. With e=0.7 (using 34% of the edges), clustering the sparsified graph delivers a better F-score than clustering the original graph, with a 35x speedup to boot.

Speedup and F-score as k changes

Figures 3(e) and 3(f) show the effect of k, the number of min-wise hashes used for approximating similarities. We also include the result of sparsification using exact similarity calculations (denoted as 'exact'). (We hold e constant at 0.5.) We see a gradual improvement in F-scores with increasing k, as the similarity approximation gets better with increasing k (refer Proposition 4). The F-scores obtained using the exact similarity calculation is 18.89, compared to the 18.47 obtained using k=30, which indicates that the similarity approximation works quite well. Also, exact similarity sparsification requires about 240x more time to compute compared to sparsification using minwise hashing with k=30 (4 hours vs 1 minute). Therefore we are gaining a lot in speedup by using minwise hashing while suffering little loss in quality. If we consider the clustering times alone, then clustering the exact sparsified graph is faster than clustering the minwise hashing sparsified graph with k=30 (50s vs 80s).

Figures 3(g) and 3(h) show the corresponding trends using MLR-MCL, which are quite similar.

**Trends with varying degree and mixing parameter**

We examine the quality of the sparsification while varying the average degree of the graph and the "mixing"-ness of different clusters. We generated synthetic graphs using the LFR generator [20], with 10000 nodes and power-law exponent 2.1 for both degree distribution as well as community size distribution. The LFR generator allows the user to vary the average degree as well as the *mixing parameter* - a number between 0 and 1 which is effectively like the average conductance of the clusters; graphs with higher mixing parameters are more challenging to cluster accurately as the clusters are more mixed up in such graphs. Figures 3(i) and 3(j) show the change in F-scores for both original and sparsified graph using Metis+MQI and MLR-MCL, while vary-

---

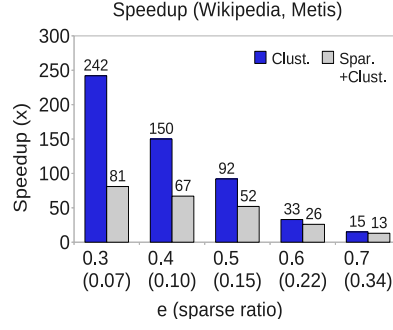[5]Graclus ran out of memory for both Wiki and Orkut and could not be executed.

Clustering Algorithm: Metis

Dataset	Original F-score	Time (s)	Sp. Ratio	RandomEdge F-score	Spdup	G-Spar F-score	Spdup	ForestFire F-score	Spdup	L-Spar F-score	Spdup
BioGrid	**17.78**	3.02	0.17	15.98	11x	15.15	30x	16.18	9x	**<u>19.71</u>**	25x
DIP	**20.04**	0.11	0.53	17.58	2x	19.38	2x	15.41	2x	**<u>21.58</u>**	2x
Human	**8.96**	0.59	0.39	7.75	4x	8.64	4x	7.47	4x	**<u>10.05</u>**	5x
Wiki	**12.34**	7485	0.15	9.11	8x	9.38	104x	9.96	7x	**<u>18.47</u>**	52x
	ϕ_{avg} (c_v)			ϕ_{avg} (c_v)		ϕ_{avg} (c_v)		ϕ_{avg} (c_v)		ϕ_{avg} (c_v)	
Orkut	**0.85** (0.1)	14373	0.17	0.82 (0.4)	13x	0.76 (0.4)	30x	0.82 (0.4)	12x	**<u>0.76</u>** (0.0)	36x
Flickr	**0.87** (2.5)	4.7	0.2	0.91 (0.1)	8x	*0.71* (0.1)	1x	0.91 (0.1)	9x	0.84 (0.3)	3x
Twitter	**<u>0.95</u>** (0.1)	2307	0.04	1.0 (0.4)	35x	0.97 (0.0)	85x	0.99 (0.4)	14x	**0.96** (1.7)	6x

Clustering Algorithm: MLR-MCL

Dataset	Original F-score	Time (s)	Sp. Ratio	RandomEdge F-score	Spdup	G-Spar F-score	Spdup	ForestFire F-score	Spdup	L-Spar F-score	Spdup
BioGrid	**23.95**	8.44	0.17	20.28	6x	18.29	38x	20.55	7x	**<u>24.90</u>**	17x
DIP	**<u>24.85</u>**	0.28	0.53	20.57	3x	22.45	3x	18.51	3x	**24.38**	3x
Human	**<u>10.55</u>**	1.68	0.39	8.81	4x	9.21	6x	8.37	4x	**10.43**	5x
Wiki	**<u>20.22</u>**	7898	0.15	8.74	19x	9.3	92x	11.59	14x	**19.3**	23x
	ϕ_{avg} (c_v)			ϕ_{avg} (c_v)		ϕ_{avg} (c_v)		ϕ_{avg} (c_v)		ϕ_{avg} (c_v)	
Orkut	**0.78** (6.4)	21079	0.17	0.85 (1.2)	6x	0.91 (10.1)	39x	0.86 (1.1)	6x	**<u>0.78</u>** (0.5)	22x
Flickr	**0.71** (0.6)	16.56	0.2	0.83 (2.2)	3x	0.72 (3.6)	2x	0.88 (1.9)	3x	**<u>0.70</u>** (0.7)	4x
Twitter	**0.90** (5.6)	14569	0.04	0.99 (0.6)	63x	0.89 (1.0)	188x	0.99 (11.0)	16x	**<u>0.86</u>** (4.3)	22x

Clustering Algorithm: Metis+MQI

Dataset	Original F-score	Time (s)	Sp. Ratio	RandomEdge F-score	Spdup	G-Spar F-score	Spdup	ForestFire F-score	Spdup	L-Spar F-score	Spdup
BioGrid	**23.16**	4.0	0.17	19.76	11x	17.74	4x	19.13	11x	**<u>23.23</u>**	5x
DIP	**<u>23.09</u>**	0.32	0.53	19.55	1x	21.18	1x	16.09	2x	**22.93**	1x
Human	**10.17**	1.16	0.39	8.42	1x	9.1	1x	8.08	2x	**<u>10.28</u>**	1x
Wiki	**<u>19.21</u>**	35511	0.15	14.97	5x	9.98	360x	14.18	5x	**18.32**	0.46x
	ϕ_{avg} (c_v)			ϕ_{avg} (c_v)		ϕ_{avg} (c_v)		ϕ_{avg} (c_v)		ϕ_{avg} (c_v)	
Orkut	**0.756** (1.2)	19799	0.17	0.86 (0.5)	2x	0.77 (0.2)	1x	0.86 (0.3)	3x	**<u>0.755</u>** (1.2)	0.7x
Flickr	**<u>0.55</u>** (14.1)	72.35	0.2	0.68 (0.4)	3x	*0.67* (0.3)	1x	0.70 (0.2)	5x	0.69 (1.0)	4x
Twitter	**<u>0.86</u>** (0.6)	11708	0.04	0.99 (0.6)	35x	0.97 (0.0)	334x	0.99(0.5)	19x	**0.89** (0.6)	14x

Clustering Algorithm: Graclus

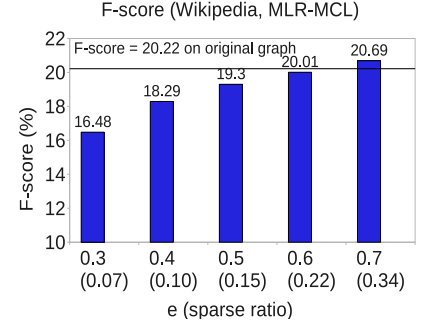| Dataset	Original F-score	Time (s)	Sp. Ratio	RandomEdge F-score	Spdup	G-Spar F-score	Spdup	ForestFire F-score	Spdup	L-Spar F-score	Spdup
BioGrid	**19.15**	0.32	0.17	17.59	4x	16.56	2x	16.67	2x	**<u>21.42</u>**	2x
DIP	**21.77**	0.19	0.53	18.27	2x	21.27	3x	15.59	5x	**<u>22.45</u>**	1x
Human	**9.53**	0.81	0.39	8.03	2x	8.75	5x	7.47	6x	**<u>9.90</u>**	1x
	ϕ_{avg} (c_v)			ϕ_{avg} (c_v)		ϕ_{avg} (c_v)		ϕ_{avg} (c_v)		ϕ_{avg} (c_v)	
Flickr	**<u>0.66</u>** (1.3)	1.35	0.2	0.72 (0.1)	2x	*0.66* (0.1)	1x	0.71 (0.1)	2x	0.72 (1.7)	2x
Twitter	**<u>0.90</u>** (2.4)	1518	0.04	1.0 (0.7)	138x	0.97 (0.0)	66x	0.99(0.6)	138x	**0.91** (0.9)	5x

Table 2: Avg. F-score (higher is better) – when ground truth is available; or Conductance (ϕ_{avg}, lower is better) – when ground truth is not available – using different clustering algorithms. $c_v = \sigma/\mu$ is coefficient of variation of cluster sizes. The best result for each dataset is underlined; the results in the original column and the best results from among the four sparsifications for each dataset are also bolded. ϕ_{avg} is always calculated w.r.t. the original graph. Contributions of singletons to ϕ_{avg} are not included for RandomEdge, G-Spar and ForestFire (L-Spar does not introduce singletons) – including the effect of singletons will increase the conductance (i.e. make it worse). Speedups are w.r.t. time for clustering the original graph and take into account *both the sparsification as well as clustering times on the sparsified graph.* 'Sp. Ratio' is the sparsification ratio (same for all sparsifications). Although G-Spar enabled low conductances on the Flickr dataset, those results are italicized since G-Spar also generated 30% singletons for this dataset, whose contributions to the conductance are not included.
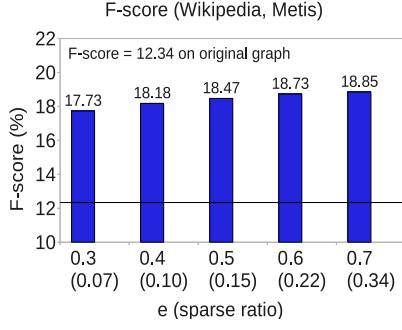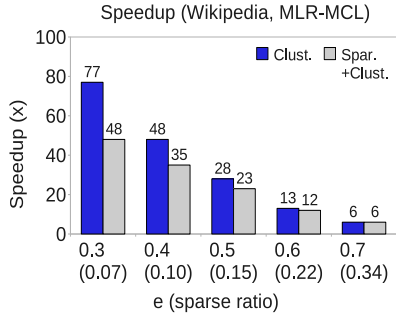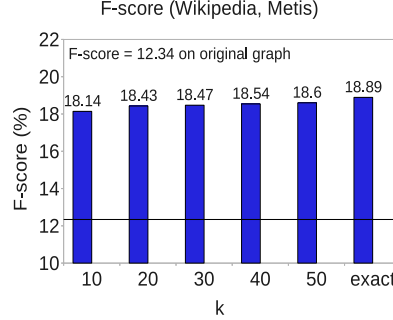
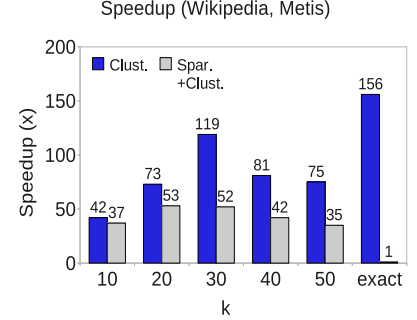(a) F-score as e changes; Metis
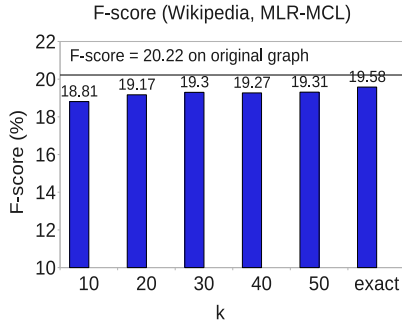(b) Speedup as e changes; Metis
(c) F-score as e changes; MLR-MCL
(d) Speedup as e changes; MLR-MCL
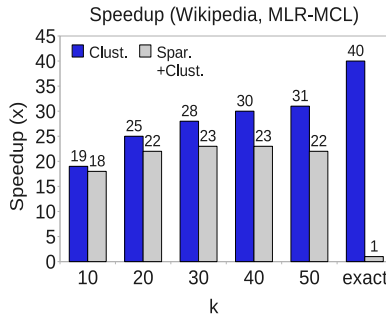(e) F-score as k changes; Metis
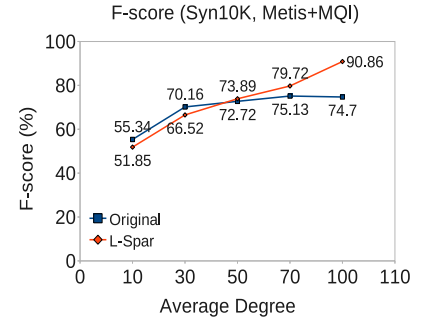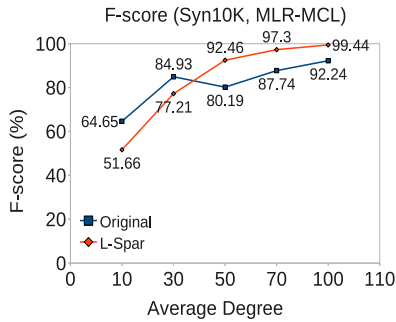(f) Speedup as k changes; Metis
(g) F-score as k changes; MLR-MCL
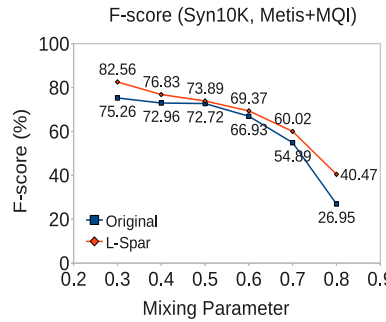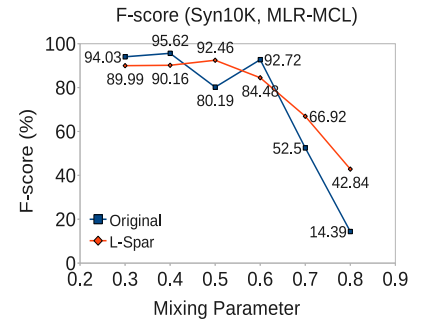(h) Speedup as k changes; MLR-MCL
(i) F-score as average degree changes; Metis+MQI
(j) F-score as average degree changes; MLR-MCL
(k) F-score as mixing parameter changes; Metis+MQI
(l) F-score as mixing parameter changes; MLR-MCL

Figure 3: The performance of L-Spar under varying conditions

Node	Examples of retained edges (neighbors)	Examples of discarded edges (neighbors)
Wiki		
Machine Learning	Decision tree learning, Support Vector Machine, Artificial Neural Network, Predictive Analytics, Document classification	Co-evolution, Case-based reasoning, Computational neuroscience, Immune system
Graph (mathematics)	Graph theory, Adjacency list, Adjacency matrix, Model theory	Tessellation, Roman letters used in Mathematics, Morphism, And-inverter graph
Moby-Dick	Billy Budd, Clarel,Moby Dick (1956 film), Jaws (film), Western canon, Nathaniel Hawthorne	Bruce Sterling, 668, Dana Scully, Canadian English, Giant squid, Redburn
Twitter		
Jack Dorsey	Biz Stone, Evan Williams, Jason Goldman, Sarah Lacy, Anil Dash	Alyssa Milano, Parul Sharma, Nick Douglas, JetBlue Airways, Whole Foods Market
Lance Armstrong	Dave Zabriskie, Michael Rogers, Vande Velde, Levi Leipheimer, George Hincapie	Chris Sacca, Robin Williams, George Stephanopoulos, Alyssa Milano, Dr. Sanjay Gupta
Michelle Malkin	Bobby Jindal, Rick Perry, Howard Kurtz, Neal Boortz, George Stephanopoulos	Barack Obama, The Today Show, Jim Long, LA Times
Flickr		
gladiator	colosseum, worldheritage, site, colosseo, italy	europe, travel, canon, sky, summer
telescope	observatory, astronomy, telescopes, sutherland	2007, geotagged, travel, california, mountain
lincolnmemorial	memorials, reflectingpool, lincoln, nationalmall	travel, usa, night, sunset, america

Table 3: Examples of retained and discarded edges using L-Spar sparsification

ing the average degree and keeping the mixing parameter constant at 0.5. The sparsification is more beneficial with increasing degree, and it actually outperforms the original clustering starting from degree 50. Figures 3(k) and 3(l) report the trend of F scores as we vary the mixing parameter keeping average degree to 50, using Metis+MQI and MLR-MCL. The performance on both original and sparsified graphs drops with increasing mixing parameter, which is to be expected. However, the sparsification does enable better performance with higher mixing parameters compared to clustering the original graph, suggesting that its ability to weed out irrelevant edges helps clustering algorithms. For instance, with the mixing parameter at 0.8, Metis+MQI achieves an F-score of 40.47 on the sparsified graph, improving upon the 26.95 achieved on the original graph.

5. DISCUSSION

We would like to begin by emphasizing that the local sparsification approach we present is extremely effective across the board for Metis, MLR-MCL and Graclus and often helpful for Metis+MQI. Accounting for local densities in the sparsification process provide significant gains across the board on all four clustering algorithms (seen by comparing results between global and local sparsification). In the case of Metis and Graclus, clearly observed for results with ground truth data, the local sparsification procedure has the added benefit of often improving the quality of the resulting partitions while also significantly speeding up the time to compute partitions. Another way of viewing this result is that these methods, particularly Metis, are less tolerant of noise in the data (presence of spurious edges often undermine their effectiveness in identifying good communities) and so post-sparsification where spurious edges are eliminated (see Table 3) the results are significantly better (e.g. Wiki). For MLR-MCL the general observation is that local sparsification results in a significant speedup at little or no cost to accuracy across the board. For Metis+MQI, as explained earlier the results are a bit mixed in the sense that we do find instances where the sparsification results in an overall slowdown to counter other instances where there is a signficant speedup. We also note that the local sparsi-
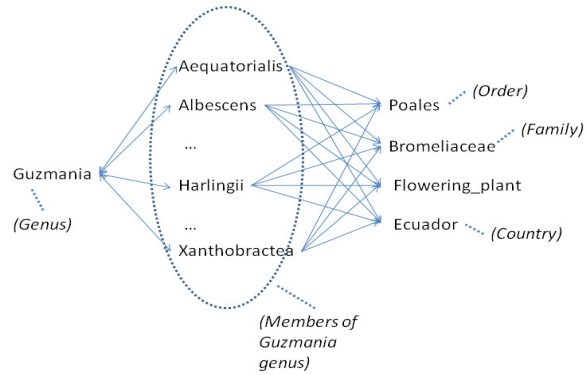
Figure 4: Parallel Path/Bipartite Community in Wiki. Circled nodes all belong to the same cluster.

fication mechanism, for all algorithms, typically results in clusters with significantly improved balance – an important criteria for many applications. Finally, the G-Spar sparsification mechanism seems well-suited for applications where the requirement is only to discover the densest clusters in the graph, although we have not directly tested this. We next discuss three ongoing research directions we are currently examining.

Bipartite/Parallel Path Communities: The reader may wonder how our sparsification will work in the case of communities without triangles, such as bipartite or parallel-path communities. As currently described the procedure we have outlined thus far will be unable to uncover such structure. However, recently suggested transformations based on bibliographic coupling and co-citation offer an opportunity in this regard [32]. By first applying such transformations and then subsequently sparsifying the transformed graph on the Wiki dataset using L-Spar allowed us to find extremely meaningful parallel-path clusters of the sort exemplified in Figure 4.

Out of Core Processing: L-Spar sparsification based on minwise hashing is also well suited for processing large, disk-resident graphs and reducing their size so that they can

be subsequently clustered in main memory. If the graph is stored as a sequence of adjacency lists (as is common practice with large graphs), minwise hashing to build length-k signatures for each node in the graph requires one sequential scan of the graph. The subsequent sparsification itself also requires only one sequential scan of the graph. Assuming the $n * k$ hashtable containing the signatures can fit in the main memory, the out-of-core version of L-Spar sparsification requires no disk I/O other than the two sequential scans. A blocked single pass algorithm may also be feasible under certain assumptions (e.g. dependent on degree distribution) and is currently under investigation.

Extensions: There are many interesting avenues for future work extending the methods detailed here. One example is the question of how to incorporate additional information if present- such as weights, textual attributes of nodes and edges, directionality of edges etc.- into the sparsification, apart from the connectivity information that we are primarily concerned with here. Additionally, we would like to investigate the role of such methods in the context of scaling up keyword search algorithms. As recently demonstrated, graph clustering has a central role to play in scaling up such algorithms to out of core datasets[10, 15].

6. CONCLUSION

In this paper, we introduced an efficient and effective localized method to sparsify a graph i.e. retain only a fraction of the original number of edges. Our sparsification strategy outperforms baselines from previous literature, as well as enables clustering that is many times faster than clustering the original graph while maintaining at least as good quality as clustering the original graph. The proposed sparsification is thus an effective way to speed up graph clustering algorithms without sacrificing quality.

7. REFERENCES

[1] D. Achlioptas and F. McSherry. Fast computation of low rank matrix approximations. In *STOC '01*, pages 611–618, 2001.

[2] S. Arora, E. Hazan, and S. Kale. A fast random sampling algorithm for sparsifying matrices. *APPROX-RANDOM '06*, pages 272–279, 2006.

[3] L. Becchetti, P. Boldi, C. Castillo, and A. Gionis. Efficient semi-streaming algorithms for local triangle counting in massive graphs. In *KDD '08*, pages 16–24, 2008.

[4] A. A. Benczúr and D. R. Karger. Approximating s-t minimum cuts in O(n2) time. In *STOC '96*, pages 47–55, 1996.

[5] T. Bohman, C. Coopery, and A. Friezez. Min-Wise independent linear permutations. *the electronic journal of combinatorics*, 7(R26):2, 2000.

[6] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations (extended abstract). In *STOC '98*, pages 327–336, 1998.

[7] G. Buehrer and K. Chellapilla. A scalable pattern mining approach to web graph compression with communities. In *WSDM '08*, pages 95–106, 2008.

[8] A. Clauset, C. Shalizi, and M. Newman. Power-law distributions in empirical data. *SIAM review*, 51(4):661–703, 2009.

[9] M. Costanzo, A. Baryshnikova, J. Bellay, Y. Kim, E. Spear, C. Sevier, H. Ding, J. Koh, K. Toufighi, S. Mostafavi, et al. The genetic landscape of a cell. *Science's STKE*, 327(5964):425, 2010.

[10] B. B. Dalvi, M. Kshirsagar, and S. Sudarshan. Keyword search on external memory data graphs. *PVLDB*, 1(1):1189–1204, 2008.

[11] I. S. Dhillon, Y. Guan, and B. Kulis. Weighted Graph Cuts without Eigenvectors: A Multilevel Approach. *IEEE Trans. PAMI*, 29(11):1944–1957, 2007.

[12] S. Fortunato. Community detection in graphs. *Phys. Reports*, 486:75–174, 2010.

[13] D. Gibson, R. Kumar, and A. Tomkins. Discovering large dense subgraphs in massive graphs. In *VLDB '05*, pages 721–732, 2005.

[14] J. Glattfelder and S. Battiston. Backbone of complex networks of corporations: The flow of control. *Phys. Rev. E*, 80(3):36104, 2009.

[15] R. C. K. and S. Sudarshan. Graph clustering for keyword search. In *COMAD '09*, 2009.

[16] D. R. Karger. Random sampling in cut, flow, and network design problems. In *STOC '94*, pages 648–657, 1994.

[17] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359, 1999.

[18] V. Krishnamurthy, M. Faloutsos, M. Chrobak, L. Lao, J. Cui, and A. Percus. Reducing large internet topologies for faster simulations. *NETWORKING '05*, pages 328–341, 2005.

[19] H. Kwak, C. Lee, H. Park, and S. Moon. What is Twitter, a social network or a news media? In *WWW '10*, pages 591–600, 2010.

[20] A. Lancichinetti, S. Fortunato, and F. Radicchi. Benchmark graphs for testing community detection algorithms. *Phys. Rev. E*, 78(4):46110, 2008.

[21] K. Lang and S. Rao. A flow-based method for improving the expansion or conductance of graph cuts. *LNCS '04*, pages 325–337, 2004.

[22] J. Leskovec and C. Faloutsos. Sampling from large graphs. In *KDD '06*, pages 631–636, 2006.

[23] J. Leskovec, K. Lang, A. Dasgupta, and M. Mahoney. Statistical properties of community structure in large social and information networks. In *WWW '08*, pages 695–704, 2008.

[24] S. Macskassy and F. Provost. Classification in networked data: A toolkit and a univariate case study. *The Journal of Machine Learning Research*, 8:935–983, 2007.

[25] A. S. Maiya and T. Y. Berger-Wolf. Sampling community structure. In *WWW '10*, pages 701–710, 2010.

[26] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and Analysis of Online Social Networks. In *IMC '07*, pages 29–42, October 2007.

[27] M. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phys. Rev. E*, 69(2):026113, Feb 2004.

[28] F. Provost and V. Kolluri. A survey of methods for scaling up inductive algorithms. *DMKD*, 3(2):131–169, 1999.

[29] S. Pu, J. Wong, B. Turner, E. Cho, and S. Wodak. Up-to-date catalogues of yeast protein complexes. *Nucleic acids research*, 37(3):825–831, 2009.

[30] A. Ruepp et al. CORUM: the comprehensive resource of mammalian protein complexes. *Nucleic Acids Research*, 36(suppl 1):D646, 2008.

[31] V. Satuluri and S. Parthasarathy. Scalable graph clustering using stochastic flows: applications to community discovery. In *KDD '09*, pages 737–746, 2009.

[32] V. Satuluri and S. Parthasarathy. Symmetrizations for clustering directed graphs. In *EDBT '11*, 2011.

[33] V. Satuluri, S. Parthasarathy, and Y. Ruan. Local graph sparsification for scalable clustering. Technical Report OSU-CISRC-11/10-TR25, The Ohio State University.

[34] J. Shi and J. Malik. Normalized Cuts and Image Segmentation. *IEEE Trans. PAMI*, 22(8), 2000.

[35] D. A. Spielman and N. Srivastava. Graph sparsification by effective resistances. In *STOC '08*, pages 563–568, 2008.

[36] D. A. Spielman and S. Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *STOC '04*, pages 81–90, 2004.

[37] M. Tumminello, T. Aste, T. Di Matteo, and R. Mantegna. A tool for filtering information in complex systems. *PNAS*, 102(30):10421, 2005.

[38] S. Van Dongen. *Graph Clustering by Flow Simulation*. PhD thesis, University of Utrecht, 2000.