gSparsify: Graph Motif Based Sparsification for Graph Clustering

Peixiang Zhao
Department of Computer Science
Florida State University, Tallahassee, FL 32306
zhao@cs.fsu.edu

ABSTRACT

Graph clustering is a fundamental problem that partitions vertices of a graph into clusters with an objective to optimize the intuitive notions of *intra-cluster density* and *inter-cluster sparsity*. In many real-world applications, however, the sheer sizes and inherent complexity of graphs may render existing graph clustering methods inefficient or incapable of yielding quality graph clusters. In this paper, we propose gSparsify, a graph sparsification method, to preferentially retain a small subset of edges from a graph which are more likely to be within clusters, while eliminating others with less or no structure correlation to clusters. The resultant simplified graph is succinct in size with core cluster structures well preserved, thus enabling faster graph clustering without a compromise to clustering quality. We consider a quantitative approach to modeling the evidence that edges within densely knitted clusters are frequently involved in small-size graph motifs, which are adopted as prime features to differentiate edges with varied cluster significance. Path-based indexes and path-join algorithms are further designed to compute graph-motif based cluster significance of edges for graph sparsification. We perform experimental studies in real-world graphs, and results demonstrate that gSparsify can bring significant speedup to existing graph clustering methods with an improvement to graph clustering quality.

Categories and Subject Descriptors

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

Keywords

Graph Sparsification; Graph Clustering; Graph Motif

1. INTRODUCTION

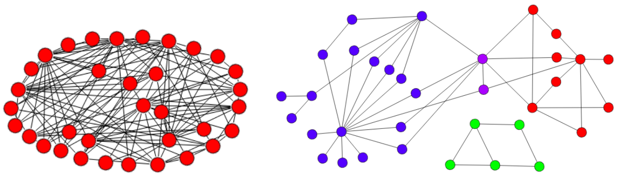
Recent years have witnessed a growing trend in business intelligence and scientific exploration that models and interprets structured data as graphs [1, 9]. As a ubiquitous abstraction depicting relationships among entities, graphs have become an increasingly common focus of data sciences, and have fused a wide range of applications in social networks, biological networks, and the Web. Out of many graph analytical and mining tasks, *graph clustering* is a fundamental building block with extensive applications in community detection [13], network visualization [31], ranking [38], and keyword search [11], to name a few. The objective of graph clustering is to partition vertices of a graph into clusters such that vertices within the same cluster are densely connected, while those in different clusters are sparsely interlinked. As a result, there have been an array of graph clustering methods toward optimizing the intuitive notions of *intra-cluster density* and *inter-cluster sparsity* for graph clusters [14, 33].

However, the problem of graph clustering remains challenging due in particular to the following reasons. First, graphs drawn from real-world applications become massive in scale. The sheer sizes of graphs may hinder a direct application of existing graph clustering methods that are mainly applicable to small or medium-size graphs. Second, and more interestingly, when graphs get larger and more complex, a great percentage of interactions (edges) have been witnessed accompanying redundant and spurious information [32]. The existence of extremely tangled, noisy edges can easily obfuscate intrinsic properties of graphs, thus leading to low-quality graph clusters on the one hand, and inducing fruitless computation on the other.

To address the aforementioned issues, we design in this paper a new graph sparsification method, gSparsify. The goal of gSparsify is to simplify a graph G in a way that the *cluster-significant* edges (*w.r.t.* the graph clustering objectives) are well retained, while edges with little or no cluster structure insight can be filtered without sacrificing the clustering quality significantly. This way, the salient structures of G are approximately preserved, or even enhanced, in the sparsified graph G', which is much smaller and more amenable to effective graph clustering. gSparsify can be employed as a pre-processing step for existing graph clustering methods with significant speedup (2x to 37x in our experimental studies) and no compromise on the quality of graph clusters. gSparsify is also of practical interest as an individual tool in many graph analytical tasks, such as graph summarization, graph backbone detection, and large-scale graph visualization.

EXAMPLE 1. *Figure 1a illustrates the famous karate club social network with* 34 *vertices and* 78 *edges. We further add random edges to complicate the graph structure with* 127 *edges in total. The graph exhibits a "hair-ball" structure*

CIKM'15, October 19–23, 2015, Melbourne, Australia.
© 2015 ACM. ISBN 978-1-4503-3794-6/15/10 ...$15.00.
DOI: http://dx.doi.org/10.1145/2806416.2806543.



(a) Karate Club Network (b) The Sparsified Graph

Figure 1: Sparsifying the Karate Club Network with 34 Vertices and 127 Edges into a Simplified Graph with 48 edges Exhibiting 4 Clusters.

which is hard to cluster. After applying the proposed graph sparsification method, gSparsify*, we get a sparsified graph (Figure 1b) with* 48 *edges (*62.2% *edge reduction). It is immediately clear that there exist* 4 *clusters in the graph.* □

To accurately identify cluster-significant edges that need to be retained during graph sparsification, we propose a quantitative approach to associating each edge $e = (u, v)$ with *structure-aware* weights denoting the degrees of relevance of two constituent vertices, u and v, to be in the same cluster. We start our reasoning with an important observation that vertices of an *intra-cluster* edge e are usually densely connected within a cluster. As a result, e is frequently involved in many basic, localized graph primitives, termed as *graph motifs*. Graph motifs are small, connected subgraphs occurring in significantly higher frequencies than would be expected in random graphs [28], and thus play a key role in uncovering structural design principles from real-world graphs. We examine a series of small-size graph motifs, and select short-length cycles as prime features to quantify cluster significance for edges. Specifically, a vector of *cluster-significance scores* is associated with each edge $e \in E$, with each component representing the number (or ratio) of every pre-selected cycle motif e lies in. Cluster-significance scores can be further aggregated to synthesize the collective significance of e in participating in and forming all specified graph motifs. The higher the cluster-significance scores of e, the more occurrences of e witnessed in graph motifs, and thus more probably e is an intra-cluster edge that should be preserved in graph sparsification.

To compute the cluster-significance scores of an edge $e = (u, v)$ in terms of short-length cycle motifs, we need enumerate all cycles encompassing e as a constituent edge, which turns out to be very time-consuming. We design a path-based indexing approach to maintaining short-length paths emanating from vertices. This way the cycle-motif enumeration problem boils down to a set of *path-join* operations on all indexed paths originated from u and v, respectively. We further use another level of inverted index to facilitate path-join evaluation, thus resulting in fast enumeration of cycle motifs in large graphs. Specifically, the contribution of our work is summarized as follows,

1. We propose a graph sparsification principle based on graph motifs with an objective to improve the efficiency and quality for graph clustering in real-world graphs. We justify the effectiveness of graph motifs, more specifically short-length cycle motifs, in encoding local cluster information of graphs, and propose a motif-based approach to computing cluster significance of edges in graphs (Section 4);
2. We design a path-based indexing approach to enumerating all short-length cycles for the computation of cluster-significance scores of edges in graphs. This approach is efficient and can be extended to enumerate other graph motifs in large graphs (Section 5);
3. We design the motif-based graph sparsification algorithm, gSparsify, to simplify real-world graphs. The experimental studies demonstrate that gSparsify can improve state-of-the-art graph clustering methods, in terms of clustering efficiency and quality (Section 6,7).

The remainder of this paper is organized as follows. We first elaborate on the related work in Section 2, then define preliminary concepts and notations in Section 3. Section 4 examines graph motifs, more specifically short length-cycles, in quantifying cluster significance of edges in graphs. In Section 5 we discuss a path-based indexing approach to enumerating short-length cycles for computing cluster significance of edges. Section 6 summarizes the graph sparsification algorithm, gSparsify. We discuss experimental studies in Section 7, and provide concluding remarks in Section 8.

2. RELATED WORK

In this section, we discuss related work for graph clustering, graph sparsification, and graph motifs. We then brief an existing graph sparsification method, L-Spar [32], with similar goals as gSparsify.

Graph clustering, also referred to as *community detection* in the social network community, has been extensively studied, and there exist rich literature on a wide range of graph clustering methods, including network flow based approaches [29], spectral clustering approaches [25], modularity based approaches [34], hierarchical graph partitioning approaches [21, 12], and local clustering approaches [10, 36, 16] (see surveys [14, 33] for a comprehensive list of graph clustering algorithms). Most of existing graph clustering solutions aim to optimize intra-cluster density, or inter-cluster sparsity, or both, based on various definitions of "density" and "sparsity" of graph clusters. Note that gSparsify is not restricted to any specific graph clustering method. Instead, it is to complement existing graph clustering methods.

Graph sparsification is to approximate a graph $G = (V, E)$ by another sparse graph $G' = (V, E')$ in achieving desired graph metrics within reasonable error bounds. As $|E'| \ll |E|$, the computation cost of related problems upon G', as opposed to G, is expected to be cheaper. Cut sparsifiers [15] are simplified graphs in which the weight of every cut agrees up to a multiplicative factor of $(1 \pm \epsilon)$ with that of the corresponding cut in the original graph. Graph spanners [7] are sparsified graphs to approximate pairwise distances of vertices. Spectral sparsifiers [3, 35] are to approximate eigenvalues to an arbitrarily small multiplicative error. SPINE [26] is to identify backbones of graphs for information propagation. However, existing graph sparsification methods are not primarily proposed to optimize graph clustering, which is the goal of our work.

Graph motifs, also known as *graphlets*, are small subgraphs defined as interaction patterns occurring at numbers significantly higher than those in randomized graphs [28]. As elementary structures of complex networks, graph motifs carry out key functionalities and represent a broad range of natural phenomena. Triangles [4] and related clustering coefficients [23] are used to detect the presence of spamming ac-

$G = (V, E)$	An input graph
$G' = (V, E')$	The sparsified graph
$\mathcal{C} = \{c_1, c_2, \ldots, c_k\}$	A clustering of k graph clusters
l_0	The user-specified length threshold
l-path	A path with l edges
l-cycle	A cycle with l edges
d_u	The degree of the vertex $u \in V$
d	The maximum degree of vertices in G
$\mathbb{F}(\cdot)$	An aggregate function
γ	The local sparsification exponent
$\varphi(c)$	The graph conductance of the cluster c

Table 1: A Primer of Notations

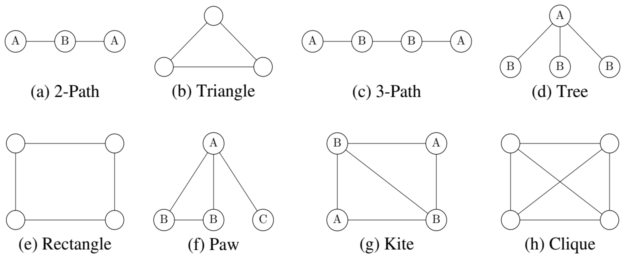
tivities in large-scale Web graphs. The distribution of graph motifs in which a vertex is part of is used as an indicator for network classification [27]. Graph motifs are also used to analyze protein-protein interaction networks [8]. However, to the best of our knowledge, there is no existing work that considers graph motifs for promoting the efficiency and effectiveness of graphs clustering.

Enumerating graph motifs is nontrivial in large graphs [18]. Both exact and approximate solutions have been proposed under various paradigms including exact counting [41], sampling [28], frequent pattern mining [8], and color coding [17]. Specifically, triangle enumeration has been extensively studied, including memory-resident algorithms [24] with the optimal complexity of $O(|E|^{3/2})$ in the worst case, and disk-resident algorithms [30, 22] for massive graphs. Counting the number of short-length cycles can be accomplished in $O(|V|^{\omega})$ where $\omega < 2.376$ is the exponent for matrix multiplication [2]. Enumerating maximal cliques from a graph has proven to be optimal in $O(3^{|V|/3})$ in the worst-case [39].

L-Spar [**32**] (short for **L**ocal **Spar**sifier) is most similar to our method, gSparsify, for graph sparsification. L-Spar takes advantage of a similarity-based heuristic that an edge $e = (u, v)$ that is a part of many triangles is probably an intra-cluster edge [19]. L-Spar adopts Jaccard coefficient, $|\Gamma(u) \cap \Gamma(v)|/|\Gamma(u) \cup \Gamma(v)|$, where $\Gamma(\cdot)$ is the adjacency list of a given vertex, to model this heuristic and uses minwise hashing [5] to approximate Jaccard coefficient, thus achieving excellent scalability for graph sparsification. The key difference between gSparsify and L-Spar is two-fold. First, instead of using a single triangle motif, gSparsify considers a group of short-length cycle motifs including triangles (a special kind of cycle motif of length 3), to model cluster significance of edges. Evaluation based on multiple graph motifs can help identify intra-cluster edges that are not necessarily involved solely in triangles, and will significantly enhance the effectiveness of graph sparsification. From this perspective, L-Spar is just a special case of gSparsify. Second, the modeling of Jaccard coefficient and minwise hashing in L-Spar fails to generalize if non-triangle motifs are considered for graph sparsification. As a result, we propose a path-based indexing approach in gSparsify to enumerate all short-length cycles based on path-join operations. This path-join method can also be extended to enumerate other graph motifs in large graphs.

3. PRELIMINARIES

We consider in this paper the graph sparsification problem on simple, undirected, and connected graphs, while the proposed algorithms can be easily extended to other types of graphs.



(a) 2-Path (b) Triangle (c) 3-Path (d) Tree

(e) Rectangle (f) Paw (g) Kite (h) Clique

Figure 2: Graph Motifs with 3 and 4 Vertices. Different Non-isomorphic Positions of Vertices in Graph Motifs are Marked by Different Alphabet Letters.

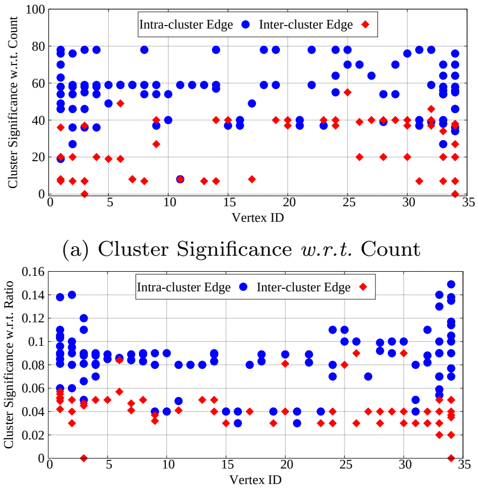
Given a graph $G = (V, E)$, we consider some basic structures that are of special interest for graph motifs and graph sparsification. A *path* $p = (v_1, \ldots, v_{l+1})$ is a sequence of vertices where $(v_i, v_{i+1}) \in E, 1 \leq i \leq l$. When the context is clear, we consider p a *simple path* in which all vertices are distinct and can be indexed and represented by $p[i], 1 \leq i \leq l+1$. A path containing $l+1$ vertices, or l edges, is of length l, denoted as an l-path for brevity. If $v_1 = v_{l+1}$, the path p is *closed* and it turns out to be a *cycle*. A cycle containing l vertices (edges) is denoted as an l-cycle. The shortest cycles are 3-cycles, also known as *triangles*. Table 1 summarizes the key notations in this paper.

4. GRAPH MOTIF-BASED CLUSTER SIGNIFICANCE

The goal of graph sparsification towards optimizing graph clustering is to preferentially retain intra-cluster edges of a graph, such that cluster structures can be well preserved in the resultant spersified graph. The critical problem is to accurately identify edges that are more likely to be intra-cluster edges. In this section, we discuss how graph motifs can be used as prime features to encode cluster structures of graphs and help identify intra-cluster edges from a graph.

In comparison to global features of graphs, such as degree distributions, diameters, and community structures [6], graph motifs are small, connected graphs captured in the local vicinity of vertices or edges, and are primarily used as elementary features representing key functionalities of graphs. Figure 2 illustrates graph motifs with 3 and 4 vertices, which can be broadly classified into two categories. The *position-sensitive* graph motifs contain vertices in different equivalence classes *w.r.t.* graph isomorphism. For example, in Figure 2, paths ((a) and (c)), tree (d), paw (f), and kite (g) are position-sensitive motifs, and the alphabet letters denote different equivalence classes of vertices *w.r.t.* graph isomorphism. In contrast, in *position-insensitive* graph motifs, all vertices are isomorphic to each other. For example, in Figure 2, cycles ((b) and (e)) and clique (h) are position-insensitive motifs.

Based on the intuitive definition of graph clustering, we note that clusters of a graph are relatively dense subgraphs. Therefore, the chances of small-size graph motifs occurring in a graph cluster are high. Equivalently, an intra-cluster edge $e \in E$ is very likely to lie frequently in different graph motifs. As a result, we evaluate the *cluster significance* of e by examining the distribution of graph motifs that include e as a constituent edge. The higher the cluster-significance value of e is, the more probably e is involved in a closely knitted local structure, and hence e is more likely to be an intra-cluster edge in the graph.



(a) Cluster Significance *w.r.t.* Count

(b) Cluster Significance *w.r.t.* Ratio

Figure 3: Cluster Significance Scores of Edges in the Karate Club Network.

However, not all graph motifs play an equal role in representing local cluster structures of graphs. Path motifs (Figure 2(a) and (c)) are often used to identify cuts in sparse regions of graphs [20]. Similarly, all the edges of tree motifs (Figure 2(d)) and the singular edge (A, C) of the paw motif (Figure 2(f)) are likely to be cuts straddling different clusters. We therefore consider short-length *cycle motifs* as prime features for cluster significance evaluation in that:

1. Graph clusters are characterized by a high density of edges, so it is reasonable to expect that intra-cluster edges form a significant number of *closed* graph structures like short-length cycles. In contrast, edges across different clusters are hardly part of cycles. It turns out that edges within many cycles are more likely to be intra-cluster edges than inter-cluster ones;

2. Cycles are the simplest position-insensitive graph motifs, so edges within a cycle can be treated uniformly *w.r.t.* graph isomorphism. We thus can design graph sparsification algorithms irrespective of the exact position an edge lies in cycles;

3. For other complex position-insensitive graph motifs such as cliques, they can be treated approximately as a composition of cycle motifs. For instance, the fact that an edge e is in a 4-clique can be simulated as e is both in two triangles (length-3 cycles) and in two rectangles (length-4 cycles).

Example 2. *We consider the four clusters in the karate club network in Example* 1 *(Figure 1). Each edge in the original graph G can be classified as intra-cluster edges if two end-vertices are in the same cluster, or inter-cluster edges otherwise. We quantify the cluster significance of an edge e in two different ways: (1)* ***Count:*** *we compute the absolute number of* 3*-cycles (triangles),* 4*-cycles (quadrangles) and* 5*-cycles (pentagons) in the graph containing e in these cycles; (2)* ***Ratio:*** *we compute the accumulative ratio of* 3*-cycles,* 4*-cycles, and* 5*-cycles containing e relative to* 3*-paths,* 4*-paths, and* 5*-paths, respectively, which go through e. For each vertex in G, its incident edges are sorted non-increasingly in terms of cluster-significance scores, and results are shown in Figure 3a and Figure 3b, respectively. Note that intra-cluster edges (blue circles) are consistently ranked higher than, and well separated from, inter-cluster edges (red diamonds), because intra-cluster edges are more frequently involved in short-length cycles. Therefore, short-length cycle motifs lend themselves well as representative candidates to model cluster significance of edges in graphs.* □

Henceforth, we focus on short-length cycles as prime graph motifs to evaluate the cluster significance of edges. We first define the notion of "cluster significance" upon which edges can be ranked *w.r.t.* short-length cycle motifs. Given a graph $G = (V, E)$ and an edge $e \in E$, if $c^l(e)$ denotes the number of l-cycles ($l \geq 3$), each of which includes e as a constituent edge, we define the *cluster significance* of e *w.r.t. the l-cycle motif* as

$$CS^l_{\mathbb{C}}(e) = c^l(e) \tag{1}$$

Here the subscript $\mathbb{C}$ stands for *absolute counting*. Given a user-specified cycle length threshold $l_0 \geq 3$, we can further maintain a *cluster-significance vector* for the edge e as

$$CSV_{\mathbb{C}}(e) = (CS^3_{\mathbb{C}}(e), \ldots, CS^{l_0}_{\mathbb{C}}(e)) \tag{2}$$

$CSV_{\mathbb{C}}(e)$ is a vector of $(l_0 - 2)$ cluster-significance scores of e *w.r.t.* a set of cycles of lengths ranging from 3 up to l_0. We further define the *cluster significance* of e *w.r.t.* counting as

Definition 1 (**CS *w.r.t.* COUNTING**). *Given $e \in E$ and a cycle length threshold l_0, the cluster significance of e* w.r.t. *counting is an aggregation of $CSV_{\mathbb{C}}(e)$:*

$$CS_{\mathbb{C}}(e) = \mathbb{F}(CS^3_{\mathbb{C}}(e), \ldots, CS^{l_0}_{\mathbb{C}}(e)) \tag{3}$$

where $\mathbb{F}(\cdot)$ is an aggregate function like SUM *or* AVG*.*

Alternatively, if $p^l(e)$ denotes the number of l-paths, each of which passes through the edge e, and we allow *closed paths*, *i.e.*, vertices within paths can occur repeatedly, we define the *cluster significance* of e *w.r.t.* l-cycle motif as

$$CS^l_{\mathbb{R}}(e) = \frac{c^l(e)}{p^l(e)} \tag{4}$$

Here the subscript $\mathbb{R}$ stands for *ratio*. Analogously, we can define the cluster-significance vector $CSV_{\mathbb{R}}(e)$ as

$$CSV_{\mathbb{R}}(e) = (CS^3_{\mathbb{R}}(e), \ldots, CS^{l_0}_{\mathbb{R}}(e)) \tag{5}$$

The cluster-significance *w.r.t.* ratio can be defined as

Definition 2 (**CS *w.r.t.* RATIO**). *Given $e \in E$ and a cycle length threshold l_0, the cluster significance of e* w.r.t. *ratio is an aggregation of $CSV_{\mathbb{R}}(e)$:*

$$CS_{\mathbb{R}}(e) = \mathbb{F}(CS^3_{\mathbb{R}}(e), \ldots, CS^{l_0}_{\mathbb{R}}(e)) \tag{6}$$

Example 3. *We consider a sample graph shown in Figure 4, the cycle motif length threshold $l_0 = 5$, and the aggregate function is SUM(·). For the edge $(1, 2)$, the cluster-significance vector* w.r.t. *counting is $CSV_{\mathbb{C}}(1, 2) = (2, 4, 4)$, meaning that the edge $(1, 2)$ is contained in two 3-cycles, four 4-cycles, and four 5-cycles. So the cluster significance of $(1, 2)$* w.r.t. *counting, $CS_{\mathbb{C}}(1, 2)$, is 10. Similarly, the cluster-significance vector* w.r.t. *ratio is $CSV_{\mathbb{R}}(1, 2) = (0.167, 0.1, 0.036)$, and the cluster significance of $(1, 2)$* w.r.t. *ratio, $CS_{\mathbb{R}}(1, 2)$, is 0.303. If we consider the edge $(5, 6)$, both*

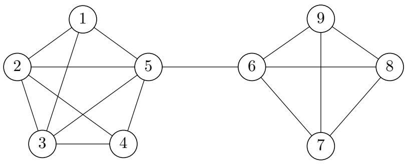


Figure 4: A Sample Graph G

$CS_{\mathbb{C}}(5,6)$ and $CS_{\mathbb{R}}(5,6)$ are 0. It means that this edge is not contained in any cycle motifs, and most probably it is an inter-cluster edge. □

5. COMPUTING CLUSTER SIGNIFICANCE

In this section, we discuss the computation of cluster-significance scores for edges in a graph. For either the count-based model, $CS_{\mathbb{C}}(e)$, or the ratio-based model, $CS_{\mathbb{R}}(e)$, we need enumerate all short-length cycles that contain e as a constituent edge. However, even enumerating the simplest triangle motifs turns out to be time-consuming in large graphs. We thus design an efficient path-based indexing approach to facilitating the computation.

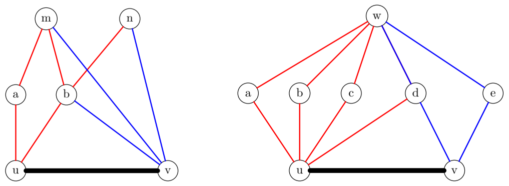
Consider an edge $e = (u, v) \in E$, any cycle including e as a constituent edge can be decomposed into three sub-parts: a path $p_1 = (u, \ldots, w)$ originating from u, another path $p_2 = (v, \ldots, w)$ originating from v, and the edge $e = (u, v)$, where p_1 and p_2 are two vertex-disjoint paths except that they share one common end-vertex $w \in V$. Specifically, we choose w as the median point to u and v in the cycle, such that either $|p_1| = |p_2|$ or $|p_1| = |p_2| \pm 1$. As a result, an l-cycle that contains the edge $e = (u, v)$ can be identified as follows. We enumerate every path p_1 of length $\lceil \frac{l-1}{2} \rceil$ emanating from u, and every path p_2 of length $\lfloor \frac{l-1}{2} \rfloor$ emanating from v, respectively. if p_1 and p_2 share no common vertices except the end-vertex other than u and v, a cycle is identified from the graph. We formulate this idea based on the notion of *path join*, as follows,

Definition 3 (**PATH JOIN**). *For an edge $e = (u, v)$, l-cycles containing e can be identified by joining two paths p_1 and p_2, $p_1 \bowtie_\theta p_2$, where the join condition θ is*

1. $|p_1| = \lceil \frac{l-1}{2} \rceil$, $|p_2| = \lfloor \frac{l-1}{2} \rfloor$;
2. $p_1[1] = u$, $p_2[1] = v$ *(p_1 and p_2 originate from u and v, respectively);*
3. $p_1[\lceil \frac{l-1}{2} \rceil] = p_2[\lfloor \frac{l-1}{2} \rfloor]$ *(p_1 and p_2 share one common end-vertex, other than u and v);*
4. $p_1[i] \neq p_2[j]$ *for* $1 < i < \lceil \frac{l-1}{2} \rceil$, $1 < j < \lfloor \frac{l-1}{2} \rfloor$ *(p_1 and p_2 are vertex-disjoint).*

Theorem 1. *All cycle motifs of length l containing the edge $e = (u, v) \in E$ as an constituent edge can be enumerated based on path join, as defined in Definition 3.*

Example 4. *Figure 5 illustrates two examples for enumerating cycle motifs that contain the edge (u, v) (in thick black color), based on path join. In Figure 5a, we enumerate all 4-cycles. The 2-paths emanating from u (color in red), and 1-paths enumerating from v (color in blue) are joined together, thus resulting in three 4-cycles. In Figure 5b, we enumerate all 5-cycles. The 2-paths emanating from u and v, respectively, are considered for path join, thus resulting in seven 5-cycles. Note that the paths (u, d, w) and (v, d, w) fail to join as both share a common intermediate vertex d, so the path-join condition does not hold.* □



(a) 4-cycles (b) 5-cycles

Figure 5: Enumerating Cycle Motifs for the Edge (u, v)

```
Algorithm 1: Computing $CS^l_{\mathbb{C}}(e)$
Input: An edge $e = (u, v) \in E$, the motif length $l$
Output: The cluster-significance score $CS^l_{\mathbb{C}}(e)$
begin
    for $w \in V$ do
        $\mathcal{H}_1(w) \leftarrow \emptyset$;    $\mathcal{H}_2(w) \leftarrow \emptyset$;
    for $p_i \in \mathcal{P}_{\lceil \frac{l-1}{2} \rceil}(u)$ do
        $\mathcal{H}_1(p_i[\lceil \frac{l-1}{2} \rceil]) \leftarrow \mathcal{H}_1(p_i[\lceil \frac{l-1}{2} \rceil]) \cup \{i\}$;
    for $p_j \in \mathcal{P}_{\lfloor \frac{l-1}{2} \rfloor}(v)$ do
        $\mathcal{H}_2(p_j[\lfloor \frac{l-1}{2} \rfloor]) \leftarrow \mathcal{H}_2(p_i[\lfloor \frac{l-1}{2} \rfloor]) \cup \{j\}$;
    $CS^l_{\mathbb{C}}(e) \leftarrow 0$;
    for $w \in V$ do
        if $\mathcal{H}_1(w) \neq \emptyset$ and $\mathcal{H}_2(w) \neq \emptyset$ then
            for $i \in \mathcal{H}_1(w)$ do
                for $j \in \mathcal{H}_2(w)$ do
                    if $\theta : (p_i \bowtie_\theta p_j)$ then
                        $CS^l_{\mathbb{C}}(e) \leftarrow CS^l_{\mathbb{C}}(e) + 1$;
    return $CS^l_{\mathbb{C}}(e)$;
```

To this end, we design a path-based indexing approach to computing cluster significance of edges in G. Given the motif length threshold, $l_0 \geq 3$, we maintain, for each vertex $v \in V$, all distinct l-paths emanating from v into a vector, $\mathcal{P}_l(v)$, where $1 \leq l \leq \lceil \frac{l_0 - 1}{2} \rceil$. Specifically, $\mathcal{P}_1(v)$ is just the adjacency list of v. If the maximum degree of vertices in G is denoted d, both the time and space complexity for path-based index construction are $O(d^{\lceil \frac{l_0 - 1}{2} \rceil} \cdot |V|)$. Note that real-world graphs often follow power-law degree distributions, so in practice a majority of vertices have degrees that are significantly smaller than d. Another important factor is that, the length threshold of cycle motifs, l_0, is often set small, as longer cycles in cluster-significance evaluation will bring marginal improvement for graph sparsification. Therefore, this path-based index can be pre-built offline effectively in the index construction phase.

Based on Theorem 1, in order to enumerate all cycle motifs of length l ($l \leq l_0$) containing an edge $e = (u, v)$, we employ all pre-indexed paths $p_i \in \mathcal{P}_{\lceil \frac{l-1}{2} \rceil}(u)$ and $p_j \in \mathcal{P}_{\lfloor \frac{l-1}{2} \rfloor}(v)$ to perform a path join, $p_i \bowtie_\theta p_j$, as detailed in Algorithm 1. To facilitate the evaluation of path join, we maintain an inverted index $\mathcal{H}_1(\cdot) : V \longmapsto 2^{\mathbb{N}}$, such that for each vertex $w \in V$, if w is the last end-vertex (other

Algorithm 2: gSparsify

```
   Input: Graph $G = (V, E)$, the motif length threshold
          $l_0$, the local sparsification exponent $\gamma$, an
          aggregate function $\mathbb{F}(\cdot)$
   Output: Sparsified graph $G'$
 1 begin
 2     $G' \leftarrow \emptyset$;
 3     for $e = (u, v) \in E$ do
 4         $CS_{\mathbb{C}}(e) \leftarrow 0$;   $CS_{\mathbb{P}}(e) \leftarrow 0$;
 5         foreach $l : 3 \leq l \leq l_0$ do
 6             compute $CS^l_{\mathbb{C}}(e)$ (Algorithm 1);
 7             compute $CS^l_{\mathbb{R}}(e)$ (Equation 7);
 8         $CS_{\mathbb{C}}(e) \leftarrow \mathbb{F}(CS^3_{\mathbb{C}}(e), \ldots, CS^{l_0}_{\mathbb{C}}(e))$;
           $CS_{\mathbb{R}}(e) \leftarrow \mathbb{F}(CS^3_{\mathbb{R}}(e), \ldots, CS^{l_0}_{\mathbb{R}}(e))$;
 9     foreach $u \in V$ do
10         Sort all incident edges $e = (u, v) \in E$ by $CS_{\mathbb{C}}(e)$
           (or by $CS_{\mathbb{R}}(e)$);
11         Add top $d_u^{\gamma}$ edges to $G'$;
12     return $G'$;
```

than u) of a path $p_i \in \mathcal{P}_{\lceil \frac{l-1}{2} \rceil}(u)$, then w together with the path index i of p_i are stored into $\mathcal{H}_1(w)$. This way, all the paths of $\mathcal{P}_{\lceil \frac{l-1}{2} \rceil}(u)$ that share a common end-vertex, w, are grouped together. Analogously, we build another inverted index $\mathcal{H}_2(\cdot)$, in which all paths of $\mathcal{P}_{\lfloor \frac{l-1}{2} \rfloor}(v)$ sharing common end-vertices (other than v) are grouped together. Both inverted indexes are first initialized (Lines 2-3), and updated in terms of the last end-vertex of each path in $\mathcal{P}_{\lceil \frac{l-1}{2} \rceil}(u)$ and $\mathcal{P}_{\lfloor \frac{l-1}{2} \rfloor}(v)$, respectively (Lines 4-7). To compute the cluster significance *w.r.t.* counting, $CS^l_{\mathbb{C}}(e)$, we consider the paths $p_i \in \mathcal{P}_{\lceil \frac{l-1}{2} \rceil}(u)$, and $p_j \in \mathcal{P}_{\lfloor \frac{l-1}{2} \rfloor}(v)$, both of which sharing a common end-vertex (Line 10). If p_i and p_j satisfy the path-join condition θ, as defined in Definition 3, we successfully identify an l-cycle from the graph (Lines 13-14).

Assume paths originating from u may end at any vertex of G, then $|\mathcal{H}_1(\cdot)| = |\mathcal{P}_{\lceil \frac{l-1}{2} \rceil}(u)|/|V|$. Similarly, $|\mathcal{H}_2(\cdot)| = |\mathcal{P}_{\lfloor \frac{l-1}{2} \rfloor}(v)|/|V|$. The time complexity of Algorithm 1 turns out to be $O(|\mathcal{P}_{\lceil \frac{l-1}{2} \rceil}(u)| \cdot |\mathcal{P}_{\lfloor \frac{l-1}{2} \rfloor}(v)| \cdot ((l-1)/2)^2/|V|)$, where the term $((l-1)/2)^2)$ denotes the time to evaluate the path-join condition θ for p_i and p_j. Note that $|\mathcal{P}_{\lceil \frac{l-1}{2} \rceil}(u)|$ is the number of $(\lceil \frac{l-1}{2} \rceil)$-paths originating from u, bounded up by $d^{\lceil \frac{l-1}{2} \rceil}$, so the worst-case complexity of Algorithm 1 is $O(\frac{d^l}{|V|})$.

Once the cluster significance *w.r.t.* counting, $CS^l_{\mathbb{C}}(e)$, has been computed, it is straightforward to derive the cluster significance *w.r.t.* ratio, $CS^l_{\mathbb{R}}(e)$, which is to normalize $CS^l_{\mathbb{C}}(e)$ by considering all possible paths passing through e:

$$CS^l_{\mathbb{R}}(e) = \frac{CS^l_{\mathbb{C}}(e)}{|\mathcal{P}_{\lceil \frac{l-1}{2} \rceil}(u)| \cdot |\mathcal{P}_{\lfloor \frac{l-1}{2} \rfloor}(v)|} \tag{7}$$

Note that all $(\lceil \frac{l-1}{2} \rceil)$-paths originating from u are maintained in the index $\mathcal{P}_{\lceil \frac{l-1}{2} \rceil}(u)$, and analogously all $(\lfloor \frac{l-1}{2} \rfloor)$-paths originating from v are maintained in $\mathcal{P}_{\lfloor \frac{l-1}{2} \rfloor}(v)$. Once $CS^l_{\mathbb{C}}(e)$ is known, $CS^l_{\mathbb{P}}(e)$ can be computed in $O(1)$.

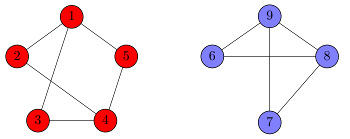


Figure 6: The Sparsified Graph, G'

6. GRAPH SPARSIFICATION: gSparsify

In this section, we detail our graph motif-based sparsification method, gSparsify, which uses short-length cycle motifs to model the cluster significance of edges. The edges with highly ranked cluster-significance scores are preferably retained. This way, the core cluster structures can be well preserved in the sparsified graph.

Algorithm 2 describes the whole process for graph sparsification. The sparsified graph, G', is first initialized as an empty graph (Line 2). For each edge $e \in G$, we compute its cluster-significance scores, $CS_{\mathbb{C}}(e)$ and $CS_{\mathbb{R}}(e)$, based on Algorithm 1 (Lines 3-8). We further choose an aggregate function, $\mathbb{F}$, to synthesize cluster-significance scores of edges based on a series of cycle motifs. After that, a localized, *vertex-centric* sparsification approach is employed to retain edges with high cluster significance in the neighborhood of vertices. Specifically, for each vertex u of the graph G, we select the top d_u^{γ} $(0 \leq \gamma < 1)$ incident edges, ranked on either $CS_{\mathbb{C}}(e)$ or $CS_{\mathbb{R}}(e)$, where d_u is the degree of u, into G' for sparsification (Lines 9-11). In principle, we sparsify high-degree vertices more aggressively than low-degree ones, because high-degree vertices are more likely to be hubs that tend to connect multiple clusters. As a result, their incident edges are likely to be inter-cluster edges. In implementation, we choose a strictly concave function, d_u^{γ}, for graph sparsification. When $\gamma \rightarrow 0$, the sparsified graph G' contains very few edges but each vertex u still preserves at least one edge, making the resultant cluster containing u connected. When $\gamma \rightarrow 1$, the sparsified graph G' is almost identical to G as most edges are retained.

The complexity of Algorithm 2 is formulated with two components. The first is to compute cluster significance of edges in the graph, which is $O(d^{l_0}|E|/|V|)$. The second is to sort, for each vertex, its incident edges based on cluster-significance scores, which is $\sum_u O(d_u \text{log} d_u) \leq \sum_u O(d_u \text{log} d) = O(\text{log} d \cdot |E|)$. So the overall complexity of Algorithm 2 is $O(d^{l_0}|E|/|V| + \text{log} d \cdot |E|)$.

Example 5. *We apply the graph sparsification algorithm,* gSparsify*, on the sample graph G shown in Figure 4 by setting $l_0 = 5$, $\gamma = 0.1$, and $\mathbb{F} =$SUM. The sparsified graph G' is illustrated in Figure 6. G' contains* 11 *edges, so* 5 *edges of G are filtered out during graph sparsification. It can be witnessed that there are two clusters (colored in red and blue, respectively) in the sparsified graph G'.* □

7. EXPERIMENTS

In this section, we present our experimental studies for gSparsify. We make use of gSparsify as a pre-processing step to sparsify a series of real-world graphs, and examine both the benefit and cost of gSparsify toward improving the effectiveness and efficiency of state-of-the-art graph clustering methods. We implement gSparsify in C and compile it with GCC 4.4.7. All experiments were carried out in a Linux machine running RedHat Enterprise Server 6.5, with 12 AMD Opteron 2.3GHz CPUs and 96GB of memory.

7.1 Datasets

We consider three real-world graphs in our experimental studies, including a protein-protein interaction network, a co-authorship network, and a social network:

1. **Yeast PPI Network:** This is a protein-protein interaction (PPI) network in the BioGrid database [37]. This graph contains $4,531$ vertices representing yeast genes, and $22,736$ edges depicting protein interactions. The average degree of vertices is 10 and the maximum degree is $1,549$. We cluster this graph into 450 clusters, each of which contains approximately 10 vertices in average, because the sizes of most protein complexes are in the range of 5 to 15;

2. **DBLP[1]:** This is a co-authorship graph where two authors are connected if they publish as co-authors at least one paper. This co-authorship graph contains $317,080$ vertices and $1,049,866$ edges. The publication venues, *e.g.*, journals or conferences, define the ground-truth clusters. We consider $5,000$ clusters in the experiments;

3. **Orkut[2]:** This is an online social network where vertices represent anonymized individuals, and edges illustrate friendship relations between individuals. The graph has $3,072,441$ vertices and $117,185,083$ edges. Orkut allows users to form groups, which are used as ground-truth clusters. We consider $5,000$ clusters in the experiments.

7.2 Graph Clustering Methods

We consider three well-known graph clustering methods in our experimental studies:

1. **METIS**[21] is a high-quality graph partitioning toolkit implemented based on multilevel recursive-bisection, multilevel k-way, and constraint-based graph partitioning schemes. We used METIS 5.1.0 with default parameter settings;

2. **Graclus**[12] is a fast graph clustering tool that computes normalized cut and ratio association for a graph without costly eigenvector computation. We use Graclus 1.2 and choose the default parameter settings for graph clustering;

3. **MCL**[40] is a scalable graph clustering method based on simulation of stochastic flows in graphs. Note that the number of clusters are indirectly controlled by the inflation parameter. By tuning this parameter, we can generate the desired number of graph clusters in the experiments.

It is worth noting that gSparsify is not restricted to the above-mentioned graph clustering algorithms. Any graph clustering technique toward optimizing the notions of intra-cluster density and inter-cluster sparsity of graph clusters can benefit from the proposed method, gSparsify.

Dataset	**Graph Motif Length**					
	$l_0 = 3$		$l_0 = 4$		$l_0 = 5$	
	Time	Space	Time	Space	Time	Space
Yeast	0	0.22	0.11	2.66	0.13	2.93
DBLP	0	10.93	47.3	349.5	49.2	371.5
Orkut	0	762.1	$1,159$	$2,716$	$1,174$	$2,755$

Table 2: Index Construction Cost of gSparsify (Time in Seconds, and Space in Megabytes).

7.3 Evaluation Metrics

We evaluate the graph sparsification method from multiple perspectives. The foremost evaluation is to examine to what extent gSparsify can help improve the graph clustering effectiveness. We apply graph clustering algorithms on the original graph G, and the sparsified graph G', respectively, and assess the clustering quality *w.r.t.* ground truth or graph clustering metrics.

If we have ground truth for clusters, $\mathcal{C} = \{c_1, c_2, \dots, c_k\}$ (*e.g.*, in DBLP and Orkut), we evaluate clustering quality based on the *average F-score* of graph clusters. The F-score of a predicted graph cluster $\bar{c}$ *w.r.t.* a ground-truth cluster $c_i \in \mathcal{C}$, denoted $F(\bar{c})|_{c_i}$, is the harmonic mean of the precision and recall. Furthermore, the F-score of $\bar{c}$ is

$$F(\bar{c}) = \max_{c_i \in \mathcal{C}} F(\bar{c})|_{c_i} \quad \forall i : 1 \leq i \leq k$$

It is the F-score of the predicted cluster $\bar{c}$ *w.r.t.* the ground truth cluster $c^* \in \mathcal{C}$ to which $\bar{c}$ approximates best. The average F-score of all the predicted clusters is the weighted average of all F-scores, each of which is weighted by the cluster size. Empirically, the higher the average F-score, the better the clustering quality.

We also consider the *average graph conductance* to evaluate the quality of graph clusters in all graph datasets. Given a cluster $c \in \mathcal{C}$ of G, the graph conductance of c is

$$\varphi(c) = \frac{\sum_{i \in c, j \in G/c} I_{ij}}{\min(I(c), I(G/c))} \quad \forall i, j \in V$$

where I_{ij} is an indicator function that equals 1 if there exists an edge (i, j) between vertices i and j, and 0 otherwise, and

$$I(c) = \sum_{i \in c} \sum_{j \in V} I_{ij}$$

Graph conductance lies between 0 and 1 with lower values indicating better clustering quality. The average graph conductance is the average of $\varphi(c)$ for all clusters $c \in \mathcal{C}$. Note that the evaluation of clustering quality based on graph conductance should be performed on the original graph G. That is, for clusters derived from the sparsified graph G', we need map all such clusters back into G and compute graph conductance in G. This way, it will reflect how well graph sparsification can retain the cluster structures of G.

Another evaluation metric is the speedup gSparsify offers for graph clustering. We apply different graph clustering methods on the original graph G and the sparsified graph G', respectively, and examine the runtime performance gain gSparsify provides. As G' is much smaller than G, clustering on G' is expected to be significantly faster.

There exist several critical parameters for gSparsify, including the cycle length threshold l_0, the local sparsification exponent γ, and the aggregate function $\mathbb{F}$. We also examine how these parameters regulate the performance of graph sparsification. If not specified otherwise, we choose the cluster significance *w.r.t.* ratio, $CS_{\mathbb{R}}$, as the default model for

[1] DBLP datasets: http://snap.stanford.edu/data/index.html

[2] Orkut datasets: http://snap.stanford.edu/data/index.html

Metis

Dataset	Original			**Sparse Ratio**	**Speedup**	L-Spar		gSparsify	
	F-score	φ	Time(s)			F-score	φ	F-score	φ
Yeast	N.A.	0.89	2.47	**0.26**	**21x**	N.A.	0.84	**N.A.**	**0.76**
DBLP	16.04	0.65	116.45	**0.21**	**18x**	16.67	0.65	**18.83**	**0.60**
Orkut	11.08	0.85	13,858	**0.17**	**37x**	10.97	0.79	**15.45**	**0.72**

Table 3: Graph Clustering Using METIS

Graclus

Dataset	Original			**Sparse Ratio**	**Speedup**	L-Spar		gSparsify	
	F-score	φ	Time(s)			F-score	φ	F-score	φ
Yeast	N.A.	0.83	0.25	**0.26**	**2x**	N.A.	0.80	**N.A.**	**0.73**
DBLP	16.72	0.61	73.9	**0.21**	**13x**	17.15	0.55	**17.36**	**0.59**

Table 4: Graph Clustering Using Graclus

MCL

Dataset	Original			**Sparse Ratio**	**Speedup**	L-Spar		gSparsify	
	F-score	φ	Time(s)			F-score	φ	F-score	φ
Yeast	N.A.	0.84	6.56	**0.26**	**18x**	N.A.	0.83	**N.A.**	**0.74**
DBLP	16.50	0.77	133.79	**0.21**	**17x**	16.52	0.75	**18.04**	**0.67**
Orkut	11.65	0.72	20,519	**0.17**	**21x**	10.79	0.81	**12.85**	**0.66**

Table 5: Graph Clustering Using MCL

graph sparsification, and set the key parameters with the following default values: $l_0 = 5$, $\gamma = 0.5$ and $\mathbb{F} = \mathsf{AVG}(\cdot)$. Meanwhile, as an indexing approach, gSparsify needs to build path-based indexes and inverted indexes, we also evaluate the time and space cost of index construction in the experimental studies.

We compare gSparsify with L-Spar [32] that only uses the triangle motif for graph sparsification. The results will demonstrate that gSparsify is a generalized method where a series of graph motifs can be incorporated together to improve the clustering quality of real-world graphs.

7.4 Index Construction Cost

We first report the index construction cost of gSparsify in different graph datasets, shown in Table 2. We choose different lengths of cycle motifs ranging from 3 to 5. When $l_0 = 3$, only the one-level neighborhood index $\mathcal{P}_1$, *i.e.*, the adjacency list, is built. When $l_0 = 4$, we need to build the path index $\mathcal{P}_2$ and use the inverted index $\mathcal{H}_1$, and when $l_0 = 5$, both the path index $\mathcal{P}_2$ and inverted indexes of all vertices are required. When the graph is small, for example, in the Yeast dataset, the index can be constructed efficiently with very small memory consumption. When the graph becomes excessively large, for example, in the Orkut dataset, the index construction needs more time and memory. However, this path-based index is constructed offline only once. Such cost is much affordable compared with the performance gain obtained for graph sparsification.

7.5 Improvement on Graph Clustering

We consider both gSparsify and L-Spar to sparsify the three graph datasets, and examine the clustering quality and speedup by applying three graph clustering algorithms on the original graph G and the sparsified graph G', respectively. As to the clustering quality, we evaluate both average F-scores, if graphs have ground truth for clusters, and the average graph conductance. For speedup, we consider the runtime cost of graph clustering on G' against G. The graph sparsification ratio, $|E'|/|E|$, is controlled by tuning the local sparsification exponent, γ.

The experimental results of using METIS are shown in Table 3. For all the three datasets, gSparsify provides better clustering quality results than the plain graph clustering method applied on the original graph, in terms of both F-scores and the graph conductance. gSparsify also outperforms L-Spar by offering better clustering quality results. For example, in the largest Orkut dataset, the F-score is enhanced from 10.97 to 15.45, and the graph conductance is further reduced from 0.79 to 0.72, both indicating that gSparsify preserves more intra-cluster edges during graph sparsification. The main reason is that gSparsify leverages short-length cycle motifs for cluster significance quantification, such that intra-cluster edges can be effectively identified from within large graphs. However, L-Spar only uses the 3-cycle motif, thus leading to less effective results. Another benefit of graph sparsification is the speedup it offers for graph clustering. We apply METIS on the original graph G, and the sparsified graph G', which is about $1/4$ to $1/5$ the size of G. The speedups for graph clustering are 21x, 18x, and 37x, respectively, for the Yeast, DBLP and Orkut datasets. This indicates that, gSparsify can significantly speedup graph clustering on large graphs while preserving, or even enhancing, the clustering quality.

We perform the same experimental studies using Graclus and MCL clustering algorithms, and the results are presented in Table 4, and Table 5, respectively. Note that Graclus cannot finish successfully in the largest Orkut dataset, so the results are not included. Our graph sparsification method, gSparsify, can improve clustering quality in different graph datasets. Specifically, gSparsify results in better clustering quality than L-Spar, in terms of both F-scores and the graph conductance. Meanwhile, gSparsify brings significant speedup for these two clustering methods. This is important because clustering in large graphs is very resource-intensive and time-consuming, while graph sparsification is an effective means to shorten the gap of the application of many graph clustering solutions that are designed mainly for small or medium-size graphs.

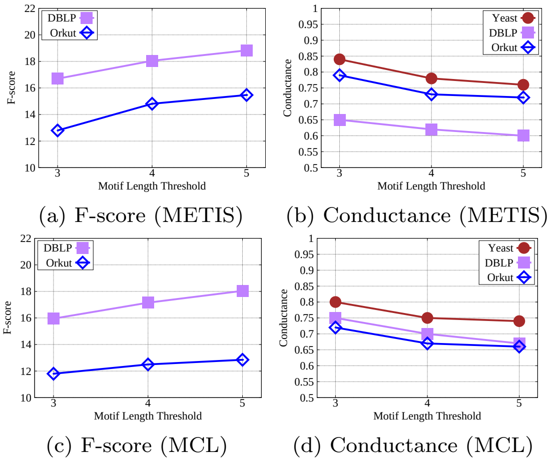
7.6 Parametric Analysis

The graph sparsification algorithm, gSparsify, is critical to a series of algorithmic parameters. We then perform experimental studies to examine how these key parameters affect the overall graph sparsification performance.

7.6.1 Graph Spasification w.r.t. l_0

We first examine the motif length threshold, l_0. By tuning the values of l_0, we adopt different short-length cycle motifs for graph sparsification. For example, if $l_0 = 3$, we only use 3-cycles in graph sparsification, while if $l_0 = 4$, both 3-cycles and 4-cycles will be employed. As shown in Figure 7a and Figure 7b, we choose l_0 ranging from 3 to 5 and evaluate the clustering quality *w.r.t.* F-scores and the graph conductance, respectively, using the METIS method. When more short-length cycle motifs are considered, intra-cluster edges are more likely to be identified during graph sparsification, and thus improve the graph clustering quality. We perform the same experiments and witness similar evidences using the MCL method, and the results are illustrated in Figure 7c and Figure 7d, corresponding to F-scores and the graph conductance, respectively.

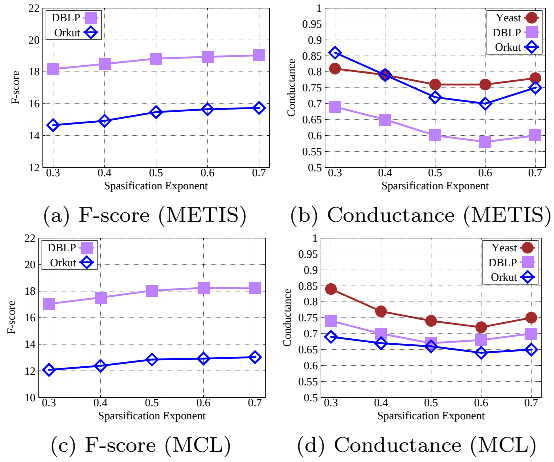
However, when l_0 is set high, we have to take more time for graph sparsification. Figure 8 illustrates the sparsification time for three different datasets, in terms of l_0. Here a common trade-off needs to be made between sparsification time and effectiveness. Considering graph sparsification can be performed offline, we still can afford more time for graph sparsification to trade better graph clustering quality, which is the goal of our work.



(a) F-score (METIS) (b) Conductance (METIS)

(c) F-score (MCL) (d) Conductance (MCL)

Figure 7: Graph Clustering Quality *w.r.t.* l_0



(a) F-score (METIS) (b) Conductance (METIS)

(c) F-score (MCL) (d) Conductance (MCL)

Figure 9: Graph Clustering Quality *w.r.t.* γ

7.6.2 *Graph Spasification* w.r.t. γ

We then examine the local sparsification exponent, γ, which controls the number of top ranked edges to be retained during graph sparsification. By tuning $0 \leq \gamma < 1$, we can derive a series of sparsified graphs with varied sizes. As shown in Figure 9a, we generate five different sparsified graphs for the graph datasets by choosing different γ ranging from 0.3 to 0.7, and evaluate clustering quality using METIS. We note that, in terms of F-scores, if we retain more edges during graph sparsification ($\gamma = 0.7$), the clustering quality can be slightly improved. If we sparsify the graph avidly be setting $\gamma = 0.3$, more edges will be filtered thus resulting in a slightly worse clustering result. However, the F-score for $\gamma = 0.3$ is still better than that generated from the original graph. It indicates that although we filter out most edges from G and retain about 10% of the edges into the sparsified graph G', the core cluster structures are still largely preserved with most of the noisy edges eliminated.

Similarly, we evaluate the clustering quality *w.r.t.* the graph conductance, and results are shown in Figure 9b. When γ is set low ($\gamma = 0.3$), we filter out many edges that may hurt the clustering effectiveness. However, when γ is set high ($\gamma = 0.7$), many noisy edges are retained in G', which also bring side effects for graph clustering and lead to inferior results. We conduct the same experiments using MCL, and results are shown in Figure 9c and Figure 9d, respectively. The patterns between the clustering quality and γ are very similar to the cases where METIS is adopted.

7.6.3 *Graph Spasification* w.r.t. $\mathbb{F}$

If multiple short-length cycle motifs are adopted to quantify the cluster significance of edges, we use an aggregate function $\mathbb{F}$ to synthesize cluster-significance scores. In this experiment, we choose two aggregate functions, SUM and AVG, and examine the effect of such aggregate functions for graph sparsification. Note that with different aggregate functions, the top ranked edges that are to be retained during graph sparsification will be different, and thus the final sparsified graph G' may vary. We test the clustering quality results, in terms of both F-scores and the graph conductance, on the DBLP dataset, as shown in Figure 10. For different graph clustering methods, AVG offers better clustering quality results than SUM in most cases. Note that it is inappropriate to choose MIN or MAX as the aggregate function in our framework, as they only choose the cluster-significance score based on one type of cycle motifs, which usually offers inferior sparsification results.

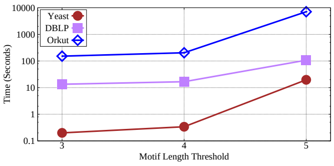


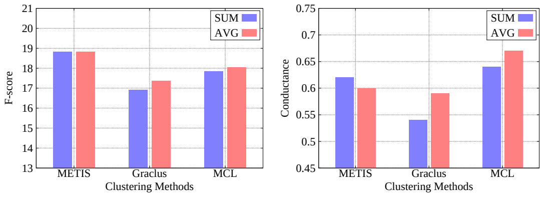
Figure 8: Graph Sparsification Time *w.r.t.* l_0

7.6.4 $CS_{\mathbb{C}}$ vs. $CS_{\mathbb{R}}$

When modeling cluster significance of edges in a graph, we consider two approaches: $CS_{\mathbb{C}}$ and $CS_{\mathbb{R}}$. The first one counts the absolute number of motifs that contain the edge to be examined, while the second one computes the relative ratio *w.r.t.* the total number of possible paths passing through the edge. We perform another experiment to evaluate these two models, each of which is chosen as the underlying method for the computation of cluster-significance scores of edges. We apply gSparsify with $CS_{\mathbb{C}}$, termed as COUNT, and gSparsify with $CS_{\mathbb{R}}$, termed as RATIO, in the DBLP dataset and examine the clustering quality of different graph clustering methods. The results are shown in Figure 11. We notice that RATIO provides consistently better quality results than COUNT, in terms of both F-scores and the graph conductance. This is especially true when dealing with edges incident to high-degree vertices.

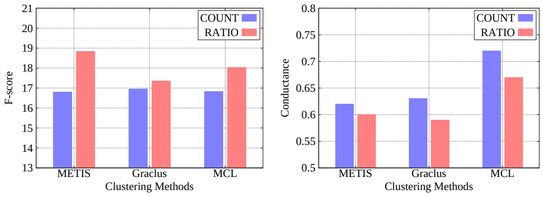
8. CONCLUSIONS

In this paper, we designed a new graph sparsification method, gSparsify, toward enabling efficient and cost-effective graph clustering on real-world graphs. The goal of gSparsify is to identify and preferentially retain cluster-significance edges of a graph G into a sparsified graph G', such that edges with little or no cluster structure insight can be effectively pruned before costly graph clustering is performed. The main idea of gSparsify is to make use of a group of short-length cycle motifs to model cluster significance of edges. To facilitate the computation of cluster significance, we devised a path-based indexing approach such that the costly graph-motif enumeration can be cast into a systemic path-join process. Our experimental studies demonstrated that, gSparsify generalizes and outperforms the state-of-the-art graph spar-



(a) F-score (DBLP) (b) Conductance (DBLP)

Figure 10: Graph Clustering Quality *w.r.t.* $\mathbb{F}$



(a) F-score (DBLP) (b) Conductance (DBLP)

Figure 11: Graph Clustering Quality *w.r.t.* Cluster-significance Modeling

sification method, L-Spar. More importantly, gSparsify enables more efficient graph clustering without a compromise of clustering quality, and therefore can be effectively employed to cluster real-world large graphs.

9. REFERENCES

[1] C. C. Aggarwal and H. Wang. *Managing and Mining Graph Data*. Springer, 2010.

[2] N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles. In *ESA'94*, pages 354–364, 1994.

[3] J. Batson, D. A. Spielman, N. Srivastava, and S.-H. Teng. Spectral sparsification of graphs: Theory and algorithms. *Commun. ACM*, 56(8):87–94, 2013.

[4] L. Becchetti, P. Boldi, C. Castillo, and A. Gionis. Efficient algorithms for large-scale local triangle counting. *ACM Trans. Knowl. Discov. Data*, 4(3):13:1–13:28, 2010.

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

[6] D. Chakrabarti and C. Faloutsos. Graph mining: Laws, generators, and algorithms. *ACM Comput. Surv.*, 38(1), 2006.

[7] S. Chechik, M. Langberg, D. Peleg, and L. Roditty. Fault-tolerant spanners for general graphs. In *STOC'09*, pages 435–444, 2009.

[8] J. Chen, W. Hsu, M. L. Lee, and S.-K. Ng. Nemofinder: Dissecting genome-wide protein-protein interactions with meso-scale network motifs. In *KDD'06*, pages 106–115, 2006.

[9] D. J. Cook and L. B. Holder. *Mining Graph Data*. John Wiley & Sons, 2006.

[10] W. Cui, Y. Xiao, H. Wang, and W. Wang. Local search of communities in large graphs. In *SIGMOD'14*, pages 991–1002, 2014.

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

[12] I. S. Dhillon, Y. Guan, and B. Kulis. Weighted graph cuts without eigenvectors: A multilevel approach. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(11):1944–1957, 2007.

[13] H. N. Djidjev and M. Onus. Scalable and accurate graph clustering and community structure detection. *IEEE Trans. Parallel Distrib. Syst.*, 24(5):1022–1029, 2013.

[14] S. Fortunato. Community detection in graphs. *CoRR*, 2009.

[15] W. S. Fung, R. Hariharan, N. J. Harvey, and D. Panigrahi. A general framework for graph sparsification. In *STOC '11*, pages 71–80, 2011.

[16] D. F. Gleich and C. Seshadhri. Vertex neighborhoods, low conductance cuts, and good seeds for local community methods. In *KDD'12*, pages 597–605, 2012.

[17] M. Gonen and Y. Shavitt. Approximating the number of network motifs. In *WAW'09*, pages 13–24, 2009.

[18] J. A. Grochow and M. Kellis. Network motif discovery using subgraph enumeration and symmetry-breaking. In *RECOMB'07*, pages 92–106, 2007.

[19] R. Gupta, T. Roughgarden, and C. Seshadhri. Decompositions of triangle-dense graphs. In *ITCS '14*, pages 471–482, 2014.

[20] M. Haghir Chehreghani. Effective co-betweenness centrality computation. In *WSDM'14*, pages 423–432, 2014.

[21] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, 1998.

[22] J. Kim, W.-S. Han, S. Lee, K. Park, and H. Yu. OPT: A new framework for overlapped and parallel triangulation in large-scale graphs. In *SIGMOD'14*, pages 637–648, 2014.

[23] K. Kutzkov and R. Pagh. On the streaming complexity of computing local clustering coefficients. In *WSDM'13*, pages 677–686, 2013.

[24] M. Latapy. Main-memory triangle computations for very large (sparse (power-law)) graphs. *Theor. Comput. Sci.*, 407(1-3):458–473, 2008.

[25] J. Liu, C. Wang, M. Danilevsky, and J. Han. Large-scale spectral clustering on graphs. In *IJCAI'13*, pages 1486–1492, 2013.

[26] M. Mathioudakis, F. Bonchi, C. Castillo, A. Gionis, and A. Ukkonen. Sparsification of influence networks. In *KDD'11*, pages 529–537, 2011.

[27] T. Milenkovic and N. Przulj. Uncovering biological network function via graphlet degree signatures. *Cancer Inform.*, 6:257–273, 2008.

[28] R. Milo and et al. Network motifs: simple building blocks of complex networks. *Science*, 298(5595):824–827, 2002.

[29] L. Orecchia and Z. A. Zhu. Flow-based algorithms for local graph clustering. In *SODA'14*, 2014.

[30] R. Pagh and F. Silvestri. The input/output complexity of triangle enumeration. In *PODS'14*, pages 224–233, 2014.

[31] T. Praneenararat, T. Takagi, and W. Iwasaki. Interactive, multiscale navigation of large and complicated biological networks. *Bioinformatics*, 27(8):1121–1127, 2011.

[32] V. Satuluri, S. Parthasarathy, and Y. Ruan. Local graph sparsification for scalable clustering. In *SIGMOD '11*, pages 721–732, 2011.

[33] S. E. Schaeffer. Survey: Graph clustering. *Comput. Sci. Rev.*, 1(1):27–64, 2007.

[34] H. Shiokawa, Y. Fujiwara, and M. Onizuka. Fast algorithm for modularity-based graph clustering. In *AAAI'13*, 2013.

[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.-H. Teng. A local clustering algorithm for massive graphs and its application to nearly linear time graph partitioning. *SIAM J. Comput.*, 42(1):1–26, 2013.

[37] C. Stark, B.-J. Breitkreutz, T. Reguly, L. Boucher, A. Breitkreutz, and M. Tyers. BioGRID: a general repository for interaction datasets. *Nucleic Acids Research*, 34:535–539, 2006.

[38] Y. Sun, J. Han, P. Zhao, Z. Yin, H. Cheng, and T. Wu. RankClus: Integrating clustering with ranking for heterogeneous information network analysis. In *EDBT '09*, pages 565–576, 2009.

[39] E. Tomita, A. Tanaka, and H. Takahashi. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theor. Comput. Sci.*, 363(1):28–42, 2006.

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

[41] S. Wernicke. Efficient detection of network motifs. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 3(4):347–359, 2006.