SCAN++: Efficient Algorithm for Finding Clusters, Hubs and Outliers on Large-scale Graphs

Hiroaki Shiokawa†‡, Yasuhiro Fujiwara†, Makoto Onizuka§
† NTT Software Innovation Center, 3-9-11 Midori-cho, Musashino, Tokyo, Japan
‡ University of Tsukuba, 1-1-1 Tennodai, Tsukuba, Ibaraki, Japan
§ Osaka University, 1-5 Yamadaoka, Suita, Osaka, Japan
†{shiokawa.hiroaki, fujiwara.yasuhiro}@lab.ntt.co.jp, §onizuka@ist.osaka-u.ac.jp

ABSTRACT

Graph clustering is one of the key techniques for understanding the structures present in graphs. Besides cluster detection, identifying hubs and outliers is also a key task, since they have important roles to play in graph data mining. The structural clustering algorithm *SCAN*, proposed by Xu *et al.*, is successfully used in many application because it not only detects densely connected nodes as clusters but also identifies sparsely connected nodes as hubs or outliers. However, it is difficult to apply SCAN to large-scale graphs due to its high time complexity. This is because it evaluates the density for all adjacent nodes included in the given graphs. In this paper, we propose a novel graph clustering algorithm named *SCAN++*. In order to reduce time complexity, we introduce new data structure of *directly two-hop-away reachable node set (DTAR)*. DTAR is the set of two-hop-away nodes from a given node that are likely to be in the same cluster as the given node. SCAN++ employs two approaches for efficient clustering by using DTARs without sacrificing clustering quality. First, it reduces the number of the density evaluations by computing the density only for the adjacent nodes such as indicated by DTARs. Second, by sharing a part of the density evaluations for DTARs, it offers efficient density evaluations of adjacent nodes. As a result, SCAN++ detects exactly the same clusters, hubs, and outliers from large-scale graphs as SCAN with much shorter computation time. Extensive experiments on both real-world and synthetic graphs demonstrate the performance superiority of SCAN++ over existing approaches.

1. INTRODUCTION

Recent advances in social and information science have shown that large-scale graphs are becoming increasingly important to represent complicated structures and schema-less data such as is generated by Twitter, Facebook and various complex networks. To understand these complex networks, graph cluster analysis (*a.k.a.* community detection) is one of the most important techniques in various research areas such as data mining [11, 37] and social science [23]. A cluster can be regarded as a group of nodes that are densely connected within a group and sparsely connected to those of other groups. Besides extracting clusters, finding special role nodes, *hubs* and *outliers*, is also a worthwhile task for understanding the structures of large-scale graphs [16]. Hubs are generally thought of as bridging different clusters. In the context of graph data mining, they are often considered as representative or influential nodes. In contrast, outliers are the nodes that are neither clusters nor hubs; they are treated as noise. The hubs and outliers provide useful insights in mining graphs. For instance, hubs in web graphs act like authoritative web pages that link similar topics [17]. The detection of outliers in web graphs is useful in stripping spam pages from web pages [35]. As well as web analysis, hubs and outliers play important roles in various applications such as marketing [7] and epidemiology [32]. That is why identifying hubs and outliers has become an interesting and important problem.

Most traditional clustering algorithms such as graph partitioning [27, 30], modularity-based method [26, 28], and density-based method [15] only study the problem of cluster detection and so ignore hubs and outliers. One of the most successful clustering methods is *structural clustering algorithm* (*SCAN*) proposed by Xu *et al.* [36]. Similar to density-based clustering, the main concept of SCAN is that densely connected adjacent nodes should be in the same cluster. However, unlike the traditional algorithms, SCAN successfully finds, at not insignificant cost, not only clusters but also hubs and outliers. As a result, SCAN has been used in many applications including bioinfomatics [6], social analyses [20], clinical analyses [22], and so on.

Although SCAN's effectiveness in detecting hubs and outliers as well as clusters is known in many applications, SCAN has, unfortunately, a serious weakness; it requires high computation costs for large-scale graphs. This is because SCAN has to find all clusters prior to identifying hubs and outliers; it first finds densely connected node sets as clusters. It then classifies the remaining non-clustered nodes into hubs or outliers. This clustering procedure entails exhaustive density evaluations for all adjacent nodes in large-scale graphs. Several methods have been proposed to improve its clustering speed. For example, LinkSCAN*, proposed by Lim *et al.*, is a state-of-the-art algorithm that uses SCAN to find overlapping communities from large-scale graphs [21]. They improve the efficiency of SCAN by employing an *edge sampling technique* [21] in the clustering process. By sampling edges from the graphs, they reduce the number of edges that require density evaluations. However, this approach produces approximated clustering results; it does not guarantee the same clustering results as the original algorithm, and so loses the superiority of SCAN [21].

1.1 Contributions

In this paper, we present a novel algorithm, *SCAN++*. SCAN++ can efficiently handle graphs with more than 100 million nodes and edges without sacrificing the clustering quality compared to SCAN.

Proceedings of the VLDB Endowment, Vol. 8, No. 11
Copyright 2015 VLDB Endowment 2150-8097/15/07.

SCAN++ is based on the property of real-world graphs; real-world graphs such as web graphs have high scores of *clustering coefficients* [34]. The clustering coefficient of a node is a measure of node density. If a node and its neighbor nodes approach a complete graph (*a.k.a.* a clique), the score of clustering coefficient becomes high. That is, a node and its two-hop-away node especially in real-world graphs are expected to share large parts of their neighborhoods. Based on this property, SCAN++ prunes the density evaluation for the nodes that are shared between a node and its two-hop-away node. Specifically, SCAN++ employs the following techniques: (1) it uses a new data structure, *directly two-hop-away reachable node set (DTAR)*, the set of nodes that are two hops away from a given node, (2) it reduces the cost of clustering by avoiding unnecessary density evaluations if the nodes are not included in DTARs, and (3) its density evaluation is efficient since DTAR allows the reusing of density evaluation results. After identifying clusters, SCAN++ classifies the remaining nodes, which do not belong to clusters, as hubs or outliers. Instead of the exhaustive computation performed by the original algorithm, SCAN++ can find clusters in an efficient manner in large-scale real-world graphs.

SCAN++ has the following attractive characteristics:

- **Efficient**: SCAN++ achieves higher clustering speeds than SCAN as well as the edge sampling technique of LinkSCAN* (Section 5.1.1). Although SCAN significantly increases its clustering time as the number of edges increases, SCAN++ has near-linear clustering time against the number of edges (Section 5.2.2).
- **Exact**: SCAN++ theoretically guarantees the same clustering results as SCAN, even though it drops unnecessary density evaluations (Section 4.3 and 5.1.4).
- **Effective**: As described above, real-world graphs have high scores in terms of clustering coefficients. SCAN++ offers efficient clustering for large-scale real-world graphs that exhibit high clustering coefficients (Section 5.2.1).

To the best of our knowledge, SCAN++ is the first solution to achieve both high efficiency and clustering results guarantees at the same time. Our experiments confirm that SCAN++ computes clusters, hubs and outliers 20.4 times faster than SCAN on average without sacrificing the clustering quality. Even though SCAN is effective in enhancing application quality, it has been difficult to apply to large-scale graphs due to its performance limitation. However, by providing a sophisticated approach that suits the identification of clusters, hubs, and outliers, SCAN++ will help to improve the effectiveness of a wider range of applications.

The remainder of this paper is organized as follows. Section 2 describes the background of this work. Section 3 and 4 introduce the main ideas of SCAN++ and their theoretical assessment, respectively. Section 5 reviews the results of our experiments. Section 6 shows related work. Section 7 provides our brief conclusion.

2. PRELIMINARY

In this section, we formally define the notations and introduce the background of this paper. Let $\mathbb{G} = \{\mathbb{V}, \mathbb{E}\}$ be an unweighted and undirected graph, where $\mathbb{V}$ and $\mathbb{E}$ are a set of nodes and edges, respectively. We assume graphs are undirected and unweighted only to simplify the representations. Other types of graphs such as directed and weighted, can be handled with only slight modifications. Table 1 lists the main symbols and their definitions.

We briefly review the original algorithm SCAN proposed by Xu *et al.* [36]. SCAN is one of the most popular graph clustering methods; it successfully detects not only clusters $\mathbb{C}$ but also hubs $\mathbb{H}$ and outliers $\mathbb{O}$ unlike traditional methods. SCAN extracts clusters

Table 1: Definition of main symbols.

Symbol	Definition
ϵ	Threshold of the structural similarity, $0 \leq \epsilon \leq 1$
μ	Minimal number of nodes in a cluster
c_u	Cluster ID of the cluster to which node u belongs
$\mathbb{G}$	Given graph
$\mathbb{V}$	Set of nodes in $\mathbb{G}$
$\mathbb{E}$	Set of edges in $\mathbb{G}$
$\mathbb{C}$	Set of clusters in $\mathbb{G}$
$\mathbb{H}$	Set of hubs in $\mathbb{H}$
$\mathbb{O}$	Set of outliers in $\mathbb{G}$
$\mathbb{P}$	Set of pivots in $\mathbb{G}$
$\mathbb{B}$	Set of bridges in $\mathbb{G}$
$\mathbb{N}[u]$	Set of nodes in the structure neighborhoods of node u
$\mathbb{N}_\epsilon[u]$	Set of nodes in the ϵ-neighborhoods of node u
$\mathbb{D}[u]$	Set of directly structure-reachable nodes of node u
$\mathbb{C}[u]$	Set of nodes that belong to the same cluster as node u
$\mathbb{T}[u]$	Set of nodes in the DTAR of node u
$\mathbb{T}_u$	Set of nodes in the converged DTAR of node u
$\mathbb{L}[u]$	Set of nodes in the local cluster of node u
$\mathbb{V}_{\mathbb{T}_u}$	Set of candidate nodes of clusters derived from $\mathbb{T}_u$
$\mathbb{P}_\epsilon[b]$	Set of pivots in the ϵ-neighborhood pivots of bridge b
$\lvert \cdot \rvert$	Number of nodes or edges included in a given set
$\sigma(u, v)$	Structural similarity between node u and v

as sets of nodes that have dense internal connections; it identifies the other non-clustered nodes (*i.e.* nodes that belong to none of the clusters) as hubs or outliers. Thus, prior to identifying hubs and outliers, it finds all clusters in a given graph.

In order to find clusters, SCAN first detects a special node, called *core*. Core is a node that has a lot of neighbor nodes with highly dense connections; the core is regarded as the seed of a cluster. SCAN uses the *structural neighborhood* [36] to evaluate density. The structural neighborhood of a node is a node set composed of the node itself and all its adjacent nodes.

DEFINITION 1 (STRUCTURAL NEIGHBORHOOD). *The definition of structural neighborhood of node u, denoted by* $\mathbb{N}[u]$*, is given by* $\mathbb{N}[u] = \{v \in \mathbb{V} : (u, v) \in \mathbb{E}\} \cup \{u\}$.

The density of adjacent nodes is computed by the common nodes in the structural neighborhoods. SCAN measures the number of common nodes in two structural neighborhoods normalized by the geometric mean of their structural neighborhood sizes. This measurement is called *structural similarity* and is defined as follows:

DEFINITION 2 (STRUCTURAL SIMILARITY). *The structural similarity between node u and v, denoted by* $\sigma(u, v)$*, is defined as* $\sigma(u, v) = |\mathbb{N}[u] \cap \mathbb{N}[v]| / \sqrt{|\mathbb{N}[u]||\mathbb{N}[v]|}$.

The structural similarity is a score varying from 0 to 1 that indicates the scale of matching degree of structural neighborhoods. When adjacent nodes share many members of their structural neighborhoods, their structural similarity becomes large.

From Definition 2, SCAN detects the core by evaluating structural similarities for all neighborhoods. In order to specify core metrics, SCAN requires two user-specified parameters. First is the minimum score of the structural similarity to neighbor nodes, denoted by ϵ. Second is the minimum number of neighborhoods, denoted by μ, all of whose structural similarities exceed ϵ. SCAN regards a node as core when it has at least μ neighbors with structural similarities greater than ϵ:

DEFINITION 3 (CORE). *Node u is core iff* $|\mathbb{N}_\epsilon[u]| \geq \mu$*, where* $\mathbb{N}_\epsilon$*, called* ϵ*-neighborhood, is* $\mathbb{N}_\epsilon[u] = \{v \in \mathbb{N}[u] : \sigma(u, v) \geq \epsilon\}$.

Once SCAN finds core, SCAN expands a cluster from the core. Specifically, nodes included in the ϵ-neighborhood of the core are assigned to the same cluster as the core. The ϵ-neighborhood nodes of core node u are called *directly structure-reachable nodes*, denoted by $\mathbb{D}[u]$ (*e.g.* in Figure 1(a), $\mathbb{D}[u_0] = \{u_0, u_1, u_5, u_6\}$ since

u_0 is core.) When node u is core and $\mathbb{D}[u] \neq \emptyset$, SCAN assigns all nodes in $\mathbb{D}[u]$ to the same cluster as node u.

SCAN recursively expands the cluster by checking whether each node, which is included in the cluster, satisfies core condition defined by Definition 3 or not. Specifically, if (1) node v is included in $\mathbb{D}[u]$ and (2) node v is core, SCAN assigns nodes in $\mathbb{D}[v]$ to the same cluster as node u. These directly structure-reachable nodes (*i.e.* $\mathbb{D}[v]$) are expanded from a member node of the cluster (*i.e.* $\mathbb{D}[u]$). These expanded directly structure-reachable nodes $\mathbb{D}[v]$ are called *structure-reachable nodes* of node u. If node $v \in \mathbb{D}[u]$ is *not* core, it does not expand the cluster from node v. All nodes in a cluster, except the core node, are called *border* nodes. SCAN recursively finds cores and expands the clusters from the cores until there are no undiscovered cores in the structure-reachable nodes of node u. After completion of cluster expansion, SCAN obtains the structure-reachable nodes of node u, which are composed of cores and borders. The original algorithm determines the obtained nodes as being in the same cluster as node u. Formally, the cluster that has node u is defined as follows:

DEFINITION 4 (CLUSTER). *The cluster by node u, denoted by $\mathbb{C}[u]$, is defined as $\mathbb{C}[u] = \{w \in \mathbb{D}[v] : v \in \mathbb{C}[u]\}$, where $\mathbb{C}[u]$ is initially set to $\mathbb{C}[u] = \{u\}$.*

After termination of cluster expansion, SCAN randomly selects a new node from the nodes that have yet to be checked. SCAN continues this procedure until there are no undiscovered cores.

Finally, SCAN identifies non-clustered nodes (*i.e.* nodes that belong to no cluster) as hubs or outliers.

DEFINITION 5 (HUB AND OUTLIER). *Assume node u does not belong to any cluster. $u \in \mathbb{H}$ iff node v and w exist in $\mathbb{N}[u]$ such that $\mathbb{C}[v] \neq \mathbb{C}[w]$. Otherwise $u \in \mathbb{O}$.*

Note that, as described in the literature [36], Definition 5 is flexible enough for practical application. For example, it may be more appropriate than Definition 5 for some applications to determine a non-clustered node with extremely high degree as a hub. This point should be discussed in future when we consider actual applications.

As a result, SCAN finds all clusters, hubs, and outliers in a graph. However, despite its effectiveness in finding the hidden structure of graphs, it is difficult to apply SCAN to large-scale graphs since it requires high time complexity. This is because the clustering procedure entails exhaustive similarity evaluations for all adjacent nodes in the given graph; Thus, if $\mathbb{V} = \{u_1, u_2, \ldots, u_{|\mathbb{V}|}\}$, the running cost of SCAN is of the order of $O(|\mathbb{N}[u_1]| + |\mathbb{N}[u_2]| + \cdots + |\mathbb{N}[u_{|\mathbb{V}|}]|) = O(|\mathbb{E}|)$. In addition to the cost of clustering, each structural similarity computation (*e.g.* $\sigma(u, v)$) takes at least $O(\min(|\mathbb{N}[u]|, |\mathbb{N}[v]|))$ time since the computation of structural similarity defined in Definition 2 enumerates all common nodes between $\mathbb{N}[u]$ and $\mathbb{N}[v]$. Therefore, the total running cost of SCAN is $O(\min(|\mathbb{N}[u]|, |\mathbb{N}[v]|)|\mathbb{E}|)$. The average and the largest size of degree are $|\mathbb{E}|/|\mathbb{V}|$ and $|\mathbb{V}|$, respectively. Hence, the average and the worst running cost of SCAN are given by $O(|\mathbb{E}|^2/|\mathbb{V}|)$ and $O(|\mathbb{V}|^3)$, respectively. Also, SCAN needs to hold the structural similarity scores of all adjacent nodes in $\mathbb{E}$ and the cluster IDs of all nodes. Thus, the space complexity of the entire clustering task of SCAN is $O(|\mathbb{E}| + |\mathbb{V}|)$.

3. PROPOSED METHOD: SCAN++

Our goal is to find exactly the same clusters, hubs, and outliers as SCAN from large-scale graphs within short computation time. In this section, we present details of our proposal, SCAN++. We first overview the ideas underlying SCAN++ and then give a full description of the graph clustering algorithm.

3.1 Overview of SCAN++

In order to efficiently find exactly same clusters as SCAN, we use an observation of real-world graphs: *if node u is two hops away from node v, their structural neighborhoods,* $\mathbb{N}[u]$ *and* $\mathbb{N}[v]$*, are likely to share large portion of nodes*. This observation is based on a well-known property of real-world graphs: *real-world graphs are expected to have high clustering coefficients* [34]. For nodes that have high clustering coefficients, the topology among a node and its neighboring nodes is likely to be a clique [34]. Thus, nodes u and v are expected to share most of their neighborhoods if they are two hops apart. For example, in a social network, if a user and friends of his/her friends are in the same community, they are likely to share a lot of common friends even if they do not have direct friendships with each other.

In order to reduce the computation costs, SCAN++ uses a new data structure based on the observation, called *directly two-hop-away reachable node set* (*DTAR* for short), instead of the directly structure-reachable nodes of SCAN. Intuitively, DTAR is a set of nodes such that (1) it includes two-hop-away nodes from a given node, and (2) the nodes in DTAR are likely to be lie in the same cluster of the given node. By selecting two-hop-away nodes from the given node, we share the computation of clustering among the given nodes and nodes in DTAR. By using DTAR, we consider two approaches to defeating the exhaustive computation of SCAN. First approach is the *two-phase clustering*. In this method, we reduce the number of similarity computations for clustering without sacrificing the quality of clusters. Specifically, the method first roughly detects subsets of clusters by computing structural similarity only for the pairs of the pivot in DTAR and its adjacent node. It then refines the subsets of clusters to find exactly the same clusters as SCAN. The exactness of clustering results is proved in Section 4.3. The second approach is the *similarity sharing*. In this method, we reduce the computation cost for each similarity computation from $O(|\mathbb{E}|/|\mathbb{V}|)$ by sharing the scores of each similarity computation. We give a detailed definition of DTAR in Section 3.2. Also, we discuss the details of the two-phase clustering method and similarity sharing method in Section 3.3 and 3.4, respectively.

In the following sections, we present our method by using the running example in Figure 1 where $\epsilon = 0.6$ and $\mu = 3$.

3.2 Directly Two-hop-away Reachable (DTAR)

We introduce the data structure called DTAR. DTAR is a set of nodes that (1) it includes two-hop-away nodes from a given node, and (2) the nodes in DTAR are likely to lie in the same cluster as the given node. The formal definition of DTAR is as follows:

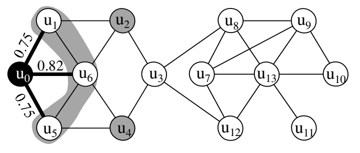
DEFINITION 6 (DIRECTLY TWO-HOP-AWAY REACHABLE). *The definition of DTAR of node u, denoted by $\mathbb{T}[u]$, is given by $\mathbb{T}[u] = \{v \in \mathbb{V} : v \notin \mathbb{N}_\epsilon[u]$ and $\mathbb{N}_\epsilon[u] \cap \mathbb{N}[v] \neq \emptyset\}$.*

In addition, we define two classes of nodes as follows:

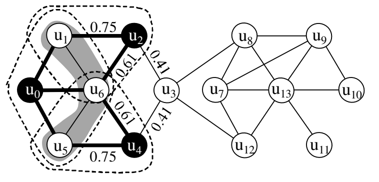
DEFINITION 7 (PIVOT AND BRIDGE). *Let $\mathbb{T}[u]$ be DTAR of node u, node u is a pivot if it acts the starting point of DTAR $\mathbb{T}[u]$. Also, non-pivot nodes in the ϵ-neighborhoods of a pivot (i.e. $\mathbb{N}_\epsilon[u]\backslash\{u\}$ for pivot u) are referred to as bridges.*

Figure 1(a) shows an example of DTAR of u_0. In this example, u_0 is a pivot, and u_1, u_5 and u_6 are bridges since $\mathbb{N}_\epsilon[u_0]\backslash\{u_0\} = \{u_1, u_5, u_6\}$. Clearly, $u_2, u_4 \notin \mathbb{N}_\epsilon[u_0]$, $\mathbb{N}_\epsilon[u_0] \cap \mathbb{N}[u_2] \neq \emptyset$, and $\mathbb{N}_\epsilon[u_0] \cap \mathbb{N}[u_4] \neq \emptyset$. Thus, $\mathbb{T}[u_0] = \{u_2, u_4\}$.

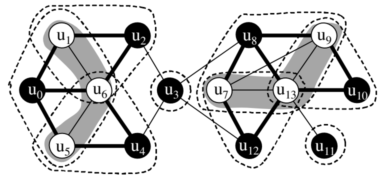
Similar to the directly structure-reachable nodes, DTAR is recursively expanded by selecting a new pivot. Let nodes u and $\mathbb{T}[u]$ be a pivot and a DTAR of node u, respectively; SCAN++ selects node $v \in \mathbb{T}[u]$ as a new pivot and then assigns all nodes in $\mathbb{T}[v]$ to a new



(a) Output of computing DTAR for u_0



(b) Output of computing converged TAR for u_0



(c) Output of the local clustering phase

Figure 1: Running example ($\epsilon = 0.6, \mu = 3$). ***Black and gray nodes denote pivots and DTARs of the pivots, respectively. Nodes circled by grayed area and circled by dotted line denote bridges and the local clusters of the pivots, respectively. Real number of the edge between u_i and u_j denotes the score of $\sigma(u_i, u_j)$ and bold lines denote $\sigma(u_i, u_j) \geq \epsilon$.***

DTAR expanded from $\mathbb{T}[u]$. This DTAR, $\mathbb{T}[v]$, expanded from a new pivot in $\mathbb{T}[u]$, is called the *two-hop-away reachable node set* (*TAR* for short). Our proposal recursively finds new pivots and expands DTARs from the pivots until there are no undiscovered pivots and bridges. After the expansions terminate, SCAN++ obtains a converged TAR rooted at a given node. Formally, the converged TAR, which is rooted at node u, is defined as follows:

DEFINITION 8 (CONVERGED TAR). *The converged TAR of pivot u, denoted by $\mathbb{T}_u$, is defined as $\mathbb{T}_u = \{w \in \mathbb{T}[v] : v \in \mathbb{T}_u$ and w is not bridge$\}$, where $\mathbb{T}_u$ is initially set to $\mathbb{T}_u = \{u\}$.*

Figure 1(b) shows an example of converged TAR of u_0. Since $\mathbb{T}[u_0] = \{u_2, u_4\}$, our method expands TAR of u_0 from u_2 and u_4 by selecting u_2 and u_4 as new pivots. Since $\mathbb{T}[u_2]$ and $\mathbb{T}[u_4]$ have no undiscovered nodes, our method stops the expansion and obtains converged TAR $\mathbb{T}_{u_0} = \{u_0, u_2, u_4\}$.

3.3 Two-phase Clustering

SCAN++ detects the clusters by constructing converged TARs and running the two-phase clustering method simultaneously. The two-phase clustering method allows us to efficiently find clusters while matching the exactness of the SCAN results. In this section, we formally introduce this two-phase clustering method.

We overview the two-phase clustering below. The two-phase clustering consists of (1) *local clustering phase* and (2) *cluster refinement phase*. In the local clustering phase, SCAN++ roughly clusters the given graph, and identifies local clusters for each converged TAR. In our algorithm, local clusters are obtained from a converged TAR. The local clusters act as a subset of clusters that are potentially included in the converged TAR. After finding the local clusters, SCAN++ obtains clusters by merging the local clusters in the cluster refinement phase. This refinement phase enables SCAN++ to produce exactly same clustering results as SCAN but with much shorter computation time. We detail each phase does in the following sections.

3.3.1 Local clustering phase

At the beginning of clustering, SCAN++ finds a converged TAR in Definition 8 and then extracts local clusters from the converged TAR, in the bottom-up clustering manner. By finding local clusters, SCAN++ captures the rough cluster structures of the given graph. The formal definition of the local cluster is given as follows:

DEFINITION 9 (LOCAL CLUSTER). *Let node u be a pivot. If $|\mathbb{N}_\epsilon[u]| \geq \mu$, the definition of the local cluster of node u, denoted by $\mathbb{L}[u]$, is given by $\mathbb{L}[u] = \mathbb{N}_\epsilon[u]$. Otherwise, $\mathbb{L}[u] = \{u\}$.*

For instance, the nodes circled by the dotted line in Figure 1(b) show an example of local clusters of the converged TAR $\mathbb{T}_{u_0}$. Since pivots u_0, u_2 and u_4 in $\mathbb{T}_{u_0}$ are cores, we have local clusters $\mathbb{L}[u_0] = \{u_0, u_1, u_5, u_6\}$, $\mathbb{L}[u_2] = \{u_1, u_2, u_6\}$, and $\mathbb{L}[u_4] = \{u_4, u_5, u_6\}$. Note that each local cluster in a converged TAR is connected to the other local clusters via bridges. The goal of this phase is to enumerate all local clusters for each pivot in each converged TAR.

Concrete details of the procedure of the local clustering phase are as follows: First, SCAN++ selects arbitrary node $u \in \mathbb{V}$ as a pivot of a DTAR. Next, SCAN++ evaluates the structural similarity defined in Definition 2 for the pivot and its adjacent node that are included in $\mathbb{N}[u]$. By applying Definition 3, SCAN++ then checks whether node u satisfies the requirement of core or not; if $|\mathbb{N}_\epsilon[u]| \geq \mu$, then node u is core. Thus SCAN++ assigns all nodes in $\mathbb{N}_\epsilon[u]$ to $\mathbb{L}[u]$ to the local cluster of node u by applying Definition 9. Otherwise, it only assigns node u to $\mathbb{L}[u]$. Then, SCAN++ obtains the DTAR rooted from node u and selects a new pivot from $\mathbb{T}[u]$. SCAN++ recursively continues this procedure until it finds converged TAR $\mathbb{T}_u$ that is rooted at node u. After that, SCAN++ selects a new pivot, a node that has not been a pivot or bridge in any converged TAR. SCAN++ terminates the local clustering phase if all nodes are assigned as pivots or bridges.

Efficiency of the local clustering phase: The local clustering evaluates the structural similarities only for the pivots; that is, it skips the similarity computations for adjacent node pairs that are lying between bridges. For example, in Figure 1(b), our method does not compute similarities for pairs (u_1, u_6) and (u_5, u_6) since u_1, u_5, and u_6 are bridges. More precisely, our approach skips $|\{(u, v) \in \mathbb{E} : u, v \in \hat{\mathbb{B}}\}|$ computations for each pivot by letting $\hat{\mathbb{B}}$ be the set of bridges of each pivot. Actually, the size of $\hat{\mathbb{B}}$ depends on the clustering coefficient of graphs. From Definition 6 and 7, we obtain $|\hat{\mathbb{B}}| = |\mathbb{N}_\epsilon[u] \cap \mathbb{N}[v]|$, where node u and v are pivots such that $v \in \mathbb{T}[u]$. Since Latapy *et al.* [19] defines the clustering coefficient of a node pair (u, v) as $c = |\mathbb{N}[u] \cap \mathbb{N}[v]|/|\mathbb{N}[u] \cup \mathbb{N}[v]|$, we have $0 \leq |\hat{\mathbb{B}}| \leq c|\mathbb{N}[u] \cup \mathbb{N}[v]|$. Recall that $|\mathbb{N}[u] \cap \mathbb{N}[v]| \leq d$ and $d \leq |\mathbb{N}[u] \cup \mathbb{N}[v]|$ where d is the average degree, thus we have $0 \leq |\hat{\mathbb{B}}| \leq cd$. As a result, the average size of $|\hat{\mathbb{B}}|$ is $cd/2$, and $|\hat{\mathbb{B}}|$ clearly depends on the clustering coefficient c. Hence, $|\hat{\mathbb{B}}|$ becomes large if graphs increase c; SCAN++ increases the number of computations that are avoided by the local clustering phase. In practice, as shown in Table 2, many real-world graphs show high clustering coefficients; thus our method successfully prunes the candidates subjected to computations for the real-world graphs. We theoretically and experimentally verify the effect of our method in Section 4.1 and 5, respectively.

3.3.2 Cluster refinement phase

After identifying the local clusters, SCAN++ then refines them to find exactly the same clusters as SCAN. From Definition 4, we introduce a necessary and sufficient condition for merging local clusters in the following lemma:

LEMMA 1 (MERGING LOCAL CLUSTERS). *Let nodes u and v lie in the same converged TAR. We have, $\exists w \in \mathbb{N}_\epsilon[u] \cap \mathbb{N}_\epsilon[v]$ s.t. $|\mathbb{N}_\epsilon[w]| \geq \mu$ iff $\mathbb{L}[u] \cup \mathbb{L}[v] \subseteq \mathbb{C}[w]$.*

PROOF. We first prove the necessary condition of Lemma 1. Since $w \in \mathbb{N}_\epsilon[u] \cap \mathbb{N}_\epsilon[v]$ *s.t.* $|\mathbb{N}_\epsilon[w]| \geq \mu$, node w is core and we have $u, v \in \mathbb{D}[w]$. From Definition 9, $\mathbb{L}[u] = \mathbb{N}_\epsilon[u] = \mathbb{D}[u]$ and $\mathbb{L}[v] = \mathbb{N}_\epsilon[v] = \mathbb{D}[v]$, when node u and v are core. Otherwise, $\mathbb{L}[u]$ and $\mathbb{L}[v]$ contain only node u and node v, respectively. Thus we have $\mathbb{L}[u] \cup \mathbb{L}[v] \subseteq \mathbb{D}[w] \cup \mathbb{D}[u] \cup \mathbb{D}[v]$. From Definition 4, we have $\mathbb{C}[w] = \{w \in \mathbb{D}[v] : v \in \mathbb{C}[w]\}$ where $\mathbb{C}[w]$ is initially set to $\mathbb{C}[w] = \{w\}$. Hence, $\mathbb{L}[u] \cup \mathbb{L}[v] \subseteq \mathbb{D}[w] \cup \mathbb{D}[u] \cup \mathbb{D}[v] \subseteq \mathbb{C}[w]$. Therefore, we have the necessary condition of Lemma 1.

Next, we prove the sufficient condition of Lemma 1. Since $\mathbb{L}[u] \cup \mathbb{L}[v] \subseteq \mathbb{C}[w]$, we have $c_u = c_v$. Hence, from Definition 4, we have node w such that $u, v \in \mathbb{C}[w]$ and $|\mathbb{N}_\epsilon[w]| \geq \mu$. Additionally, from Definition 9, nodes u and v are pivots. Recall Definitions 6 and 8, two pivots (*i.e.* node u and v) only share the nodes in $\mathbb{N}_\epsilon[u] \cap \mathbb{N}[v]$ (or $\mathbb{N}[u] \cap \mathbb{N}_\epsilon[v]$). Therefore, node w must be in $\mathbb{N}_\epsilon[u] \cap \mathbb{N}[v]$ (or $\mathbb{N}[u] \cap \mathbb{N}_\epsilon[v]$). Since $u, v \in \mathbb{C}[w]$, nodes u and v have $\sigma(u, w) \geq \epsilon$ and $\sigma(v, w) \geq \epsilon$, respectively. Thus $w \in \mathbb{N}_\epsilon[u] \cap \mathbb{N}_\epsilon[v]$, which yields the sufficient condition of Lemma 1. □

From Lemma 1, if we have core in $\mathbb{N}_\epsilon[u] \cap \mathbb{N}_\epsilon[v]$, $\mathbb{L}[u]$ and $\mathbb{L}[v]$ are assigned to the same cluster. From Definition 9, a local cluster is adjacent to other local clusters via bridges. Hence, if a bridge satisfies the core condition in Definition 3, SCAN++ merges the local clusters adjacent to the bridge into the same cluster. Figure 1(c) shows an example of the output of the local clustering phase. In Figure 1(c), u_6 and u_{13} can be cores since u_6 and u_{13} are adjacent to at least $\mu = 3$ pivots with structural similarity greater than ϵ. Thus, u_6 and u_{13} can merge their adjacent local clusters, $\{\mathbb{L}[u_0], \mathbb{L}[u_2], \mathbb{L}[u_4]\}$ and $\{\mathbb{L}[u_8], \mathbb{L}[u_{10}], \mathbb{L}[u_{12}]\}$, respectively.

Intuitively, to find local clusters that are merged into the same cluster, we check all bridges to determine whether they can be cores or not. This is because Lemma 1 implies that we may be able to merge local clusters if a bridge has more than two pivots in its ϵ-neighborhoods. However, this straightforward approach incurs high computation costs since we have to compute similarities among cores and bridges. To avoid this inefficiency, SCAN++ reuses the results of the local clustering phase. We first define a set of pivots that are included in ϵ-neighborhood of a bridge.

DEFINITION 10 (ϵ-NEIGHBORHOOD PIVOTS). *Let node b be a bridge extracted in the local clustering phase, the ϵ-neighborhood pivots of node b, denoted by $\mathbb{P}_\epsilon[b]$, are defined as $\mathbb{P}_\epsilon[b] = \{p \in \mathbb{N}[b] : \sigma(b, p) \geq \epsilon$ and p is a pivot$\}$.*

For example, the six bridges u_1, u_5, u_6, u_7, u_9, and u_{13} in Figure 1(c) have the following ϵ-neighborhood pivots, $\mathbb{P}_\epsilon[u_1]=\{u_0, u_2\}$, $\mathbb{P}_\epsilon[u_5]=\{u_0, u_4\}$, $\mathbb{P}_\epsilon[u_6]=\{u_0, u_2, u_4\}$, $\mathbb{P}_\epsilon[u_7]=\{u_8, u_{12}\}$, $\mathbb{P}_\epsilon[u_9]$ $=\{u_8, u_{10}\}$, and $\mathbb{P}_\epsilon[u_{13}]=\{u_8, u_{10}, u_{12}\}$.

From Lemma 1, we have to extract cores from bridges such that $|\mathbb{P}_\epsilon[b]| \geq 2$ since such bridges connects two or more pivots (and local clusters) with the structural similarity greater than ϵ. However, if the ϵ-neighborhood pivots of a bridge already satisfy the core condition in Definition 3 (*i.e.* $|\mathbb{P}_\epsilon[b]| \geq \mu$) by the local clustering phase, we can determine that the bridge is core without computing similarities. In addition, from Lemma 1 and Definition 10, we can introduce prunable bridges given by the following lemma.

LEMMA 2 (PRUNABLE BRIDGES). *Let bridge b be core, and $\bigcup_{p \in \mathbb{P}_\epsilon[b]} \mathbb{L}[p]$ be the merged cluster by Lemma 1. The following set shows prunable bridges that are merged into clusters without computing similarities in the subsequent cluster refinement process:*

$$\{b' \in \textstyle\bigcup_{p \in \mathbb{P}_\epsilon[b]} \mathbb{L}[p] : |\{p' \in \mathbb{P}_\epsilon[b'] : c_{p'} \neq c_b\}| = 0\}, \quad (1)$$

where $c_{p'}$ and c_b are clusters of pivot p' and bridge b, respectively.

PROOF. From Definition 10, prunable bridges have neighborhood pivots whose cluster ids are the same as c_b. This implies that all neighboring local clusters have already been merged in the same cluster by Lemma 1. Hence, the prunable bridges do not merge any local clusters in the subsequent cluster refinement process. □

Lemma 2 implies that we can skip the process to determine the prunable bridges are core nodes at the cluster refinement process. For example, in Figure 1(c), bridge u_6 and u_{13} are clearly cores since $|\mathbb{P}_\epsilon[u_6]| \geq \mu$ and $|\mathbb{P}_\epsilon[u_{13}]| \geq \mu$, respectively. Thus, we obtain the prunable bridges $\{u_1, u_5\}$ for u_6 and $\{u_7, u_9\}$ for u_{13} since u_6 and u_{13} merge the local clusters $\{\mathbb{L}[u_0], \mathbb{L}[u_2], \mathbb{L}[u_4]\}$ and $\{\mathbb{L}[u_8], \mathbb{L}[u_{10}], \mathbb{L}[u_{12}]\}$, respectively.

By using Lemma 1, 2 and Definition 10, we introduce a concrete procedure for the cluster refinement phase as follows: First, SCAN++ obtains a set of bridges $\mathbb{B}$ as a result of the local clustering phase. Next, it selects bridge $b \in \mathbb{B}$ that maximizes $|\mathbb{P}_\epsilon[b]|$ so that we can merge a lot of local clusters and remove many prunable bridges from $\mathbb{B}$ by Lemma 2 if bridge b is core. Then, it determines whether bridge b is core or not. If bridge b is core, SCAN++ merges all nodes in $\bigcup_{p \in \mathbb{P}_\epsilon[b]} \mathbb{L}[p]$ into the same cluster based on Lemma 1. Then, SCAN++ obtains all prunable bridges included in $\{b' \in \bigcup_{p \in \mathbb{P}_\epsilon[b]} \mathbb{L}[p] : |\{p' \in \mathbb{P}_\epsilon[b'] : c_{p'} \neq c_b\}| = 0\}$ by Lemma 2, and removes them from $\mathbb{B}$. These processes are continued until there are no bridges that have more than μ local clusters.

After the above procedure, we can divide the remaining bridges into two groups by their degree: (1) bridges with $|\mathbb{N}[b]| < \mu$, or (2) bridges with $|\mathbb{N}[b]| \geq \mu$ and $2 \leq |\mathbb{P}_\epsilon[b]| < \mu$. From Definition 3, the former case trivially has no cores, hence SCAN++ removes them from $\mathbb{B}$. The latter case may have some cores, so SCAN++ computes the structural similarities only for the bridges in the latter case. Finally, SCAN++ terminates the cluster refinement when there are no unevaluated bridges in $\mathbb{B}$.

Efficiency of the cluster refinement phase: Our cluster refinement phase has short computation time for two reasons: First is that SCAN++ does not require exhaustive structural similarity computations for all bridges. In practice, two local clusters in a converged TAR tend to share a lot of bridges due to the high clustering coefficients of real-world graphs. This implies that we can merge several local clusters at the same time by checking only one of the bridges, and thus prune a lot of computations for prunable bridges included in the merged local clusters (Lemma 2). Therefore, we can reduce the computation time by merging local clusters. Second reason is that structural similarity computations are not required for bridges if the parameter settings are effective. This is based on the observations on the effective parameters (*i.e.* ϵ and μ) for real-world graphs as revealed by Xu *et al.* [36] and Lim *et al.* [21]. In the literature [36], they revealed the following effective parameter setting, given the goal of reasonable clustering results for real-world graphs: *"an ϵ value between 0.5 and 0.8 is normally sufficient to achieve a good clustering result. We recommend a value for μ, of 2."* Also, in the literature [21], Lim *et al.* revealed that clustering quality parameter is less sensitive to μ than ϵ. These observations imply that desirable clustering results can be obtained by properly choosing the above parameters. In practice, if we set parameter μ = 2 based on the observation of the literature [36], the bridges have the following attractive property for efficient computations:

LEMMA 3 (PROPERTY OF BRIDGES FOR $\mu = 2$). *If we set μ = 2, bridges always satisfy the core condition.*

PROOF. From the definition of DTAR in Definition 6, SCAN++ always selects bridges from ϵ-neighborhoods of a pivot. In addition, from the definitions of the structural similarity in Definition 2, each node always has the structural similarity that is equal to 1 with itself (*e.g.* $\sigma(u,u) = 1$). As a result, bridges have $|\mathbb{P}_\epsilon[b]| \geq 2$, therefore they always satisfy the core condition when $\mu = 2$. $\square$

That is, bridges in real-world graphs are cores and so structural similarities do not need to be calculated for bridges.

As a result, SCAN++ lowers the computation cost by cluster refinement. We will show that cluster refinement has small, practical computation time for real-world graphs in Section 5.1.1.

3.4 Similarity Sharing

In this section, we describe our approach to reducing the cost of structural similarity computation. As shown in Section 2, the original algorithm enumerates all common nodes in the structural neighborhoods of two adjacent nodes. This approach is expensive since its time complexity is $O(|\mathbb{E}|/|\mathbb{V}|)$ on average. Hence, we introduce an efficient method for computing the structural similarity by sharing the intermediate results of structural similarities in DTAR. We first introduce a topological property of DTAR, and then we detail our approach based on the property. In order to show the property, we first define pivot subgraph $\mathbb{G}_w$ by using $\mathbb{T}[u]$ as follows:

DEFINITION 11 (PIVOT SUBGRAPH). *If node v is a two-hop-away node from node u (i.e. $v \in \mathbb{T}[u]$) given in Definition 6 and $\mathbb{G}_w = \{\mathbb{V}_w, \mathbb{E}_w\}$ is the pivot subgraph of node w where $\mathbb{V}_w \subseteq \mathbb{V}$ and $\mathbb{E}_w \subseteq \mathbb{E}$, $\mathbb{V}_w$ and $\mathbb{E}_w$ are defined as $\mathbb{V}_w = \mathbb{N}[u] \cap \mathbb{N}[v] \cup \{w\}$ and $\mathbb{E}_w = \{(x,y) \in \mathbb{E} : x, y \in \mathbb{V}_w\}$, respectively.*

Definition 11 indicates that if node v is included in $\mathbb{T}[u]$, we have two pivot subgraphs $\mathbb{G}_u$ and $\mathbb{G}_v$ for node u and v, respectively. For example, since $u_2 \in \mathbb{T}[u_0]$, Figure 1(a) has two pivot subgraphs $\mathbb{G}_{u_0}$ and $\mathbb{G}_{u_2}$ consisting of $\mathbb{V}_{u_0} = \{u_0, u_1, u_6\}$ and $\mathbb{V}_{u_2} = \{u_1, u_2, u_6\}$.

Definition 11 provides the following lemma that shows a topological property of DTAR suggested in Definition 6.

LEMMA 4 (SUBGRAPH ISOMORPHISM OF DTAR). *If node v is a directly two-hop away reachable from node u (i.e. $v \in \mathbb{T}[u]$) given in Definition 6, the pivot subgraphs of node u and v (i.e. $\mathbb{G}_u$ and $\mathbb{G}_v$) are always isomorphic [5].*

PROOF. From Definition 1 and 6, $\mathbb{N}[u] \cap \mathbb{N}[v] = \{w \in \mathbb{V} : (u,w) \in \mathbb{E} \wedge (v,w) \in \mathbb{E}\} \neq \emptyset$ if $v \in \mathbb{T}[u]$. Hence, if mapping $\varphi(u) = v$ and $\varphi(w) = w$ where $w \in \mathbb{N}[u] \cap \mathbb{N}[v]$, trivially we have isomorphism mapping $\varphi : \mathbb{V}_u \to \mathbb{V}_v$ with $(x,y) \in \mathbb{E}_u \Leftrightarrow (\varphi(x), \varphi(y)) \in \mathbb{E}_v$. Therefore, $\mathbb{G}_u$ and $\mathbb{G}_v$ are isomorphic. $\square$

This lemma implies that if node u is a pivot and node v is a node in $\mathbb{T}[u]$ given by Definition 6, node v and the nodes in $\mathbb{N}[u] \cap \mathbb{N}[v]$ always have the same subgraph topology as the subgraph of node u and nodes in $\mathbb{N}[u] \cap \mathbb{N}[v]$. For instance, in Figure 1(a), the two pivot subgraphs consisting of $\mathbb{V}_{u_0} = \{u_0, u_1, u_6\}$ and $\mathbb{V}_{u_2} = \{u_1, u_2, u_6\}$ are clearly isomorphic. Thus, by using Lemma 4, we introduce the following lemma for efficient structural similarity computation.

LEMMA 5 (SIMILARITY SHARING). *If we have nodes u, v and w such that $v \in \mathbb{T}[u]$ and $w \in \mathbb{N}[u] \cap \mathbb{N}[v]$, we can compute structural similarity $\sigma(v,w)$ by using the result of the structural similarity $\sigma(u,w)$ as follows:*

$$\sigma(v,w) = \frac{\sqrt{|\mathbb{N}[u]||\mathbb{N}[w]|}\sigma(u,w) - |(\mathbb{N}[u] \backslash \mathbb{N}[v]) \cap \mathbb{N}[w]| + |(\mathbb{N}[v] \backslash \mathbb{N}[u]) \cap \mathbb{N}[w]|}{\sqrt{|\mathbb{N}[v]||\mathbb{N}[w]|}}. \quad (2)$$

PROOF. From Definition 11 and Lemma 4, we have two pivot isomorphic subgraphs $\mathbb{G}_u$ and $\mathbb{G}_v$ for node u and v, respectively. Therefore, $\mathbb{N}[u] \cap \mathbb{N}[w]$ shares $\mathbb{N}[u] \cap \mathbb{N}[v] \cap \mathbb{N}[w] \neq \emptyset$ with $\mathbb{N}[v] \cap \mathbb{N}[w]$ since $\mathbb{N}[u] \cap \mathbb{N}[v] \neq \emptyset$ and $w \in \mathbb{N}[u] \cap \mathbb{N}[v]$ for $v \in \mathbb{T}[u]$ given by Definition 6. Hence, if we decompose $|\mathbb{N}[u] \cap \mathbb{N}[w]|$ and $|\mathbb{N}[v] \cap \mathbb{N}[w]|$ by using $\mathbb{N}[u] \cap \mathbb{N}[v] \cap \mathbb{N}[w]$ into $|\mathbb{N}[v] \cap \mathbb{N}[w]| = |\mathbb{N}[u] \cap \mathbb{N}[v] \cap \mathbb{N}[w]| + |(\mathbb{N}[v] \backslash \mathbb{N}[u]) \cap \mathbb{N}[w]|$ and $|\mathbb{N}[u] \cap \mathbb{N}[w]| = |\mathbb{N}[u] \cap \mathbb{N}[v] \cap \mathbb{N}[w]| + |(\mathbb{N}[u] \backslash \mathbb{N}[v]) \cap \mathbb{N}[w]|$, we have,

$$|\mathbb{N}[v] \cap \mathbb{N}[w]| = |\mathbb{N}[u] \cap \mathbb{N}[w]| - |(\mathbb{N}[u] \backslash \mathbb{N}[v]) \cap \mathbb{N}[w]| + |(\mathbb{N}[v] \backslash \mathbb{N}[u]) \cap \mathbb{N}[w]|. \quad (3)$$

From Definition 2, structural similarity is as follows:

$$\sigma(v,w) = \frac{|\mathbb{N}[v] \cap \mathbb{N}[w]|}{\sqrt{|\mathbb{N}[v]||\mathbb{N}[w]|}} \quad (4) \qquad \sigma(u,w) = \frac{|\mathbb{N}[u] \cap \mathbb{N}[w]|}{\sqrt{|\mathbb{N}[u]||\mathbb{N}[w]|}} \quad (5)$$

Hence, from Eq. (3) and (5),

$$\text{Eq. (4)} = \frac{|\mathbb{N}[u] \cap \mathbb{N}[w]| - |(\mathbb{N}[u] \backslash \mathbb{N}[v]) \cap \mathbb{N}[w]| + |(\mathbb{N}[v] \backslash \mathbb{N}[u]) \cap \mathbb{N}[w]|}{\sqrt{|\mathbb{N}[v]||\mathbb{N}[w]|}}$$

$$= \frac{\sqrt{|\mathbb{N}[u]||\mathbb{N}[w]|}\sigma(u,w) - |(\mathbb{N}[u] \backslash \mathbb{N}[v]) \cap \mathbb{N}[w]| + |(\mathbb{N}[v] \backslash \mathbb{N}[u]) \cap \mathbb{N}[w]|}{\sqrt{|\mathbb{N}[v]||\mathbb{N}[w]|}}. \quad (6)$$

Therefore, we have Lemma 5. $\square$

Lemma 5 implies that we can reuse the result of the similarity computation $\sigma(u,w)$ for obtaining $\sigma(v,w)$ where node v is a two-hop-away node from node u (*i.e.* $v \in \mathbb{T}[u]$) and $w \in \mathbb{N}[u] \cap \mathbb{N}[v]$.

Efficiency of similarity sharing method: As shown in Lemma 5, SCAN++ shares the scores of structural similarity computations between a node and a node in the DTAR. Hence, SCAN++ reduces the cost of structural similarity computation. From Lemma 4 and 5, the efficiency of the similarity sharing method is as follows:

LEMMA 6 (COMPLEXITY OF LEMMA 5). *Let $v \in \mathbb{T}[u]$ and $w \in \mathbb{N}[u] \cap \mathbb{N}[v]$. The computation of $\sigma(v,w)$ in Lemma 5 requires $O(\min(|\mathbb{N}[v] \backslash \mathbb{N}[u]|, |\mathbb{N}[w]|))$ if $\sigma(u,w)$ has been obtained.*

PROOF. From Lemma 5, we can obtain the score of $\sigma(v,w)$ by computing $\sigma(u,v)$, $|(\mathbb{N}[u] \backslash \mathbb{N}[v]) \cap \mathbb{N}[w]|$ and $|(\mathbb{N}[v] \backslash \mathbb{N}[u]) \cap \mathbb{N}[w]|$. Given $v \in \mathbb{T}[u]$, we have already had the score of $\sigma(u,w)$ by Definition 6. Additionally, since $(\mathbb{N}[u] \backslash \mathbb{N}[v]) \cap \mathbb{N}[w] \subseteq \mathbb{N}[u] \cap \mathbb{N}[w]$, $|(\mathbb{N}[u] \backslash \mathbb{N}[v]) \cap \mathbb{N}[w]|$ was also obtained when SCAN++ computed $|\mathbb{N}[u] \cap \mathbb{N}[w]|$ for $\sigma(u,w)$. The remaining term of Eq. (2) is just $|(\mathbb{N}[v] \backslash \mathbb{N}[u]) \cap \mathbb{N}[w]|$. Therefore the similarity sharing requires the computational cost $O(\min(|\mathbb{N}[v] \backslash \mathbb{N}[u]|, |\mathbb{N}[w]|))$. $\square$

Lemma 6 shows the similarity sharing incurs $O(\min(|\mathbb{N}[v] \backslash \mathbb{N}[u]|, |\mathbb{N}[w]|))$ when $\sigma(u,w)$ has been computed. In Figure 1(a), for computing $\sigma(u_1, u_2)$ it is enough to confirm whether $\mathbb{N}[u_2] \backslash \mathbb{N}[u_0] = \{u_3\}$ is included in $\mathbb{N}[u_1]$ or not since $u_2 \in \mathbb{T}[u_0]$ and $\sigma(u_0, u_1)$ have already been obtained. In contrast, as shown in Definition 2, the original computation form of structural similarity incurs $O(\min(|\mathbb{N}[v]|, |\mathbb{N}[w]|))$ times. Hence, in Figure 1(a), we need to check whether all nodes in $\{u_1, u_3, u_6\}$ are included in $\mathbb{N}[u_1]$ or not. Thus, if $\sigma(u,w)$ has been computed, the similarity sharing reduces the cost of computing $\sigma(v,w)$ such that $v \in \mathbb{T}[u]$ and $w \in \mathbb{N}[u]$ compared to the original computation form.

3.5 Algorithm of SCAN++

We can efficiently extract the clustering results by using two-phase clustering and similarity sharing. The pseudo-code of our proposal, SCAN++, is given in Algorithm 1. Algorithm 1 consists of three parts: local clustering phase given by Section 3.3.1 (line 2-17), cluster refinement phase given by Section 3.3.2 (line 18-37), and classification of hubs and outliers (line 38-44). Initially all the nodes are labeled with their own cluster-id (*i.e.* c_u for node u). First, SCAN++ runs local clustering phase (line 2-17). It selects a node as a pivot of a DTAR (line 3-6). Then, SCAN++ computes the structural similarities for the pivot by using Lemma 5

Algorithm 1 SCAN++

```
Input: $\mathbb{G} = (\mathbb{V}, \mathbb{E})$, $\epsilon \in \mathbb{R}$, $\mu \in \mathbb{N}$;
Output: clusters $\mathbb{C}$, hubs $\mathbb{H}$, and outliers $\mathbb{O}$;
 1: $\mathbb{U} = \mathbb{V}$, $\mathbb{B} = \emptyset$;
 2: while $\mathbb{U} \neq \emptyset$ do
 3:     select a node $u \in \mathbb{U}$;
 4:     $\mathbb{T}_u = \{u\}$;
 5:     while we have unvisited pivots in $\mathbb{T}_u$ do
 6:         select node $p \in \mathbb{T}_u$;
 7:         for each node $v \in \mathbb{N}[p]$ do
 8:             evaluate $\sigma(p, v)$ by Lemma 5;
 9:         end for
10:         get $\mathbb{L}[p]$ by Definition 9;
11:         label all nodes in $\mathbb{L}[p]$ as $c_p$;
12:         get $\mathbb{T}[p]$ by Definition 6;
13:         expand $\mathbb{T}_u$ using $\mathbb{T}[p]$ by Definition 8
14:     end while
15:     get $\mathbb{V}_{\mathbb{T}_u}$ by Definition 12;
16:     $\mathbb{U} = \mathbb{U}\backslash\mathbb{V}_{\mathbb{T}_u}$, $\mathbb{B} = \mathbb{B} \cup \{\mathbb{N}_\epsilon[p]\backslash\{p\}\}$;
17: end while
18: while $\mathbb{B} \neq \emptyset$ do
19:     get node $b \in \mathbb{B}$ s.t. $\arg\max |\mathbb{P}_\epsilon[b]|$;
20:     if $|\mathbb{N}[b]| < \mu$ then
21:         $\mathbb{B} = \mathbb{B}\backslash\{b\}$;
22:     else
23:         if $2 \leq |\mathbb{P}_\epsilon[b]| < \mu$ or $b$ has already been visited then
24:             evaluate $\sigma(b, b')$ for $b' \in \mathbb{N}[b]\backslash\mathbb{P}_\epsilon[b]$, $\mathbb{B} = \mathbb{B}\backslash\{b\}$;
25:         end if
26:         if node $b$ is core then
27:             merge $\bigcup_{p\in\mathbb{P}_\epsilon[b]} \mathbb{L}[p]$ in to the same cluster;
28:             label all nodes in $\bigcup_{p\in\mathbb{P}_\epsilon[b]} \mathbb{L}[p]$ as $c_b$;
29:             for each bridge $b'$ in $\bigcup_{p\in\mathbb{P}_\epsilon[b]} \mathbb{L}[p]$ do
30:                 if $|\{p \in \mathbb{P}_\epsilon[b'] : c_p \neq c_b\}| = 0$ then
31:                     $\mathbb{B} = \mathbb{B}\backslash\{b'\}$ by Lemma 2;
32:                 end if
33:             end for
34:         end if
35:     end if
36: end while
37: insert all clusters into $\mathbb{C}$;
38: for each singleton node $u \in \mathbb{V}$ do
39:     if $\exists x, y \in \mathbb{N}[u]$ s.t. $c_x \neq c_y$ then
40:         label node $u$ as hub and $u \in \mathbb{H}$;
41:     else
42:         label node $u$ as outlier and $u \in \mathbb{O}$;
43:     end if
44: end for
```

(line 7-9). After that, it finds local clusters from the pivot by Definition 9 (line 10-11). Finally, it expands $\mathbb{T}_u$ by Definition 6 (line 12-13), and continues this procedure until there are no unvisited pivots in $\mathbb{T}_u$. Then, the cluster refinement phase starts. SCAN++ refines local clusters (line 18-37). First, SCAN++ selects bridge b that maximizes $|\mathbb{P}_\epsilon[b]|$ (line 19). If $|\mathbb{N}[b]| < \mu$, the bridge can not be core, and hence it is removed from $\mathbb{B}$ (line 20-21). Otherwise, when $2 \leq |\mathbb{P}_\epsilon[b]| < \mu$ or bridge b has already been visited, SCAN++ computes the structural similarity of bridge b until SCAN++ can identify node b as core or border (line 23-25). Then, SCAN++ checks if bridge b satisfies the core condition in Definition 3 (line 26). If the bridge is core, SCAN++ merges local clusters by Lemma 1 (line 27-28) and removes prunable bridges from $\mathbb{B}$ based on Lemma 2 (line 29-33). Finally, SCAN++ adds the clusters derived in this phase to $\mathbb{C}$ (line 37). After the cluster refinement, SCAN++ classifies the singleton nodes that do not belong to any cluster, as either hubs or outliers (line 38-44). This phase is based on Definition 5. If a singleton node is adjacent to multiple clusters, it regards the node as a hub (line 38). Otherwise, it regards the node as an outlier (line 40). After assigning all nodes to clusters $\mathbb{C}$, hubs $\mathbb{H}$ or outliers $\mathbb{O}$, SCAN++ terminates the clustering procedure.

3.6 Parallel Extension of SCAN++

The previous sections (Section 3.1 to 3.5) assumed that SCAN++ was implemented as a single-threaded program. Recent studies have shown the increased availability of parallel graph processings such as Apache Giraph [1] and Pregel [24]. Thus, we introduce an extension of SCAN++ for MapReduce [4], which is one of the most standard parallel processing frameworks.

The basic idea of the MapReduce-based SCAN++ is as follows. At first, we assume that node-ids are assigned by breadth first search. The local clustering phase (Section 3.3.1) and the cluster refinement phase (Section 3.3.2) correspond to the map and reduce functions, respectively. More specifically, the map function takes key-value pairs $\langle u, \mathbb{N}[u]\rangle$ as input, where u is a node in $\mathbb{V}$. Then, it starts the local clustering phase from a node u whose neighbor nodes are in the same map function. Next, the map function outputs $\texttt{list}(\langle b, \mathbb{L}[p_i]\rangle)$ where b is a bridge node and $p_i \in \mathbb{P}_\epsilon[b]$. The reduce function takes $\langle b, \texttt{list}(\mathbb{L}[p_i])\rangle$ as input, and outputs merged local clusters by running the cluster refinement phase (Section 3.3.2). Finally, we obtain clustering results by assigning a same cluster-id to nodes included in a same cluster and classifying the singleton nodes into hubs or outliers.

4. THEORETICAL ANALYSES OF SCAN++

In this section, we theoretically discuss the efficiency, space complexity, and exactness of SCAN++.

4.1 Efficiency of SCAN++

We analyze the computational complexity of algorithm SCAN++. Given a graph with $|\mathbb{V}|$ nodes and $|\mathbb{E}|$ edges, SCAN++ finds all clusters *w.r.t.* given parameter settings. This theoretically entails the following time complexity:

THEOREM 1 (TIME COMPLEXITY OF SCAN++). *SCAN++ incurs time complexity of $O(\frac{2-c}{2\delta+c}|\mathbb{E}|)$ for clustering where $\delta = |\mathbb{V}|/|\mathbb{E}|$ and c is the average pairwise clustering coefficient [19].*

PROOF. Let $|\mathbb{P}|$, d and s be the average number of pivots, the average degree, and the average computation cost of each similarity computation, respectively. The total computation cost of SCAN++ can be represented as $O(|\mathbb{P}|ds)$ since SCAN++ computes structural similarities for each pivot (Section 3). In order to prove Theorem 1, we specify the cost of $|\mathbb{P}|$ and s below.

First, we specify $|\mathbb{P}|$. As we described in Section 3.3.1, each pivot has $cd/2$ bridges on average. Hence, from Definition 9, the average size of a local cluster is $cd/2+1$. Thus, the average number of pivots is expected to $O(|\mathbb{P}|) = O(\frac{|\mathbb{V}|}{cd/2+1})$.

Next, we specify s. Lemma 6 shows that we can obtain structural similarity on DTARs by computing just $|(\mathbb{N}[v]\backslash\mathbb{N}[u]) \cap \mathbb{N}[w]|$ where $v \in \mathbb{T}[u]$ and $w \in \mathbb{N}[u] \cap \mathbb{N}[v]$. Hence, the time complexity of each similarity computation is $O(\min(|\mathbb{N}[v]\backslash\mathbb{N}[u]|, |\mathbb{N}[w]|))$. Recall that a pivot shares $cd/2$ neighborhoods with the other pivots, hence we have $O(s) = O(\min(|\mathbb{N}[v]\backslash\mathbb{N}[u]|, |\mathbb{N}[w]|)) = O(d - cd/2) = O((2-c)d)$ times for each similarity computation.

Therefore, we have $O(|\mathbb{P}|ds) = O(\frac{(2-c)|\mathbb{V}|d}{c+2/d})$. Recall that $d = |\mathbb{E}|/|\mathbb{V}| = \delta^{-1}$, hence SCAN++ has $O(|\mathbb{P}|ds) = O(\frac{2-c}{2\delta+c}|\mathbb{E}|)$. □

In practice, real-world graphs show $|\mathbb{V}| \ll |\mathbb{E}|$, $0 < \delta \ll 1$, and $0 < c < 1$. Also, it is known that the clustering coefficient c of most real-world graphs tends to be high [34]. Thus, Theorem 1 indicates that SCAN++ can find clusters much faster than SCAN since it needs $O(|\mathbb{E}|^2/|\mathbb{V}|)$ and we obtain $|\mathbb{E}|^2/|\mathbb{V}| - \frac{2-c}{2\delta+c}|\mathbb{E}| > 0$. In Section 5, we experimentally verify the efficiency of SCAN++.

4.2 Space Complexity of SCAN++

We theoretically analyze the space complexity of our proposed algorithm SCAN++ as follows:

THEOREM 2 (SPACE COMPLEXITY OF SCAN++). *SCAN++ requires space complexity of $O(|\mathbb{E}|+|\mathbb{V}|+l)$ to obtain clustering results, where l is the sum of all local cluster sizes that are detected by the local clustering phase (i.e. $l=\sum_p |\mathbb{L}[p]|$).*

PROOF. Let $\mathbb{P}$ and $\mathbb{B}$ be the sets of pivots and bridges extracted by SCAN++, respectively. From Algorithm 1, SCAN++ extracts $|\mathbb{P}|$ pivots, $|\mathbb{B}|$ bridges, and $|\mathbb{V}|$ cluster-ids in the local clustering phase. In addition, it computes $|\mathbb{E}|/|\mathbb{V}|$ structural similarities for each pivot and holds $|\mathbb{P}_\epsilon[b]|$ for each bridges. Furthermore, SCAN++ holds local clusters of all pivots such that the total size of local clusters is defined as l. Thus, the total space complexity is $O(|\mathbb{P}|+|\mathbb{B}|+|\mathbb{V}|+\frac{|\mathbb{P}||\mathbb{E}|}{|\mathbb{V}|}+|\mathbb{B}||\mathbb{P}_\epsilon[b]|+l)$. Since all nodes are included in either $\mathbb{P}$ or $\mathbb{B}$, we have $|\mathbb{V}|=|\mathbb{P}|+|\mathbb{B}|$. Also, from Definition 10, $|\mathbb{P}_\epsilon[b]|$ is at most $|\mathbb{E}|/|\mathbb{V}|$. Hence, the space complexity is $O(|\mathbb{P}|+|\mathbb{B}|+|\mathbb{V}|+\frac{|\mathbb{P}||\mathbb{E}|}{|\mathbb{V}|}+|\mathbb{B}||\mathbb{P}_\epsilon[b]|+l)=O(|\mathbb{E}|+|\mathbb{V}|+l)$. □

Theorem 2 indicates that SCAN++ needs larger space than SCAN, which requires $O(|\mathbb{E}|+|\mathbb{V}|)$ space for clustering. In addition, in the worst case, SCAN++ incurs $O(2|\mathbb{E}|+|\mathbb{V}|)$ space complexity since $l\approx|\mathbb{P}||\mathbb{E}|/|\mathbb{V}|\approx|\mathbb{V}||\mathbb{E}|/|\mathbb{V}|=|\mathbb{E}|$. However, from Definition 9, the size of each local cluster relies on the parameter ϵ; thus, the space cost of l can be small for specific parameter ϵ settings in practice. In Section 5.1.3, we experimentally verify the actual impacts of l by varying parameter ϵ values.

4.3 Exactness of SCAN++

We analyze the exactness of clustering results of SCAN++ compared to SCAN. Prior to discussing the exactness, we define a set of nodes that are cluster candidates derived from a converged TAR.

DEFINITION 12 (CANDIDATE CLUSTERS). *Let $\mathbb{T}_u$ be a converged TAR obtained by SCAN++, the candidate cluster $\mathbb{V}_{\mathbb{T}_u}$ derived from $\mathbb{T}_u$ is defined as $\mathbb{V}_{\mathbb{T}_u}=\mathbb{T}_u\cup\{\bigcup_{\forall v\in\mathbb{T}_u}\mathbb{N}_\epsilon[v]\}$.*

From Definition 12, we have the following lemma:

LEMMA 7 (NON-DIRECTLY STRUCTURE-REACHABILITY). *Let $\overline{\mathbb{V}}_{\mathbb{T}_u}$ be nodes that do not belong to $\mathbb{V}_{\mathbb{T}_u}$ (i.e. $\overline{\mathbb{V}}_{\mathbb{T}_u}=\mathbb{V}\backslash\mathbb{V}_{\mathbb{T}_u}$). SCAN++ has the following property for adjacent node pair (u,v).*

$$u\in\mathbb{V}_{\mathbb{T}_u} \text{ and } v\in\overline{\mathbb{V}}_{\mathbb{T}_u}\Rightarrow\sigma(u,v)<\epsilon. \quad (7)$$

PROOF. We prove Lemma 7 by contradiction. We assume that adjacent node pair (v,w) has $\sigma(v,w)\geq\epsilon$ if $v\in\mathbb{V}_{\mathbb{T}_u}$ and $w\in\overline{\mathbb{V}}_{\mathbb{T}_u}$. From Definition 6 and 8, all bridges are adjacent to only the nodes in $\mathbb{V}_{\mathbb{T}_u}$. Thus node v must be a pivot of $\mathbb{T}_u$ since node v is adjacent to node w which belongs to $\overline{\mathbb{V}}_{\mathbb{T}_u}$. Recall Definition 6 that SCAN++ regards ϵ-neighborhoods of a pivot as bridges that are included in $\mathbb{V}_{\mathbb{T}_u}$. Hence, node w is a member of $\mathbb{N}_\epsilon[v]\subseteq\mathbb{V}_{\mathbb{T}_u}$, and this contradicts $w\in\overline{\mathbb{V}}_{\mathbb{T}_u}$. This yields Lemma 7. □

Lemma 7 implies $\mathbb{V}_{\mathbb{T}_u}$ is always surrounded by adjacent nodes whose similarities are less than ϵ. According to Lemma 7, we introduce the following property of clusters derived from $\mathbb{V}_{\mathbb{T}_u}$.

LEMMA 8 (CLUSTER COMPREHENSIBILITY). *Let $\mathbb{C}[v]$ be a cluster where node $v\in\mathbb{V}_{\mathbb{T}_u}$. All member nodes included in $\mathbb{C}[v]$ satisfy the following condition:*

$$v\in\mathbb{V}_{\mathbb{T}_u}\Rightarrow\forall w\in\mathbb{C}[v],w\in\mathbb{V}_{\mathbb{T}_u}. \quad (8)$$

PROOF. We prove Lemma 8 by contradiction. We first have the following assumption: $v\in\mathbb{V}_{\mathbb{T}_u}\Rightarrow\exists w\in\mathbb{C}[v],w\notin\mathbb{V}_{\mathbb{T}_u}$. From Definition 4, node w is included in the structure-reachable node set of node v since $w\in\mathbb{C}[v]$. However, Lemma 7 shows that node w has similarity less than ϵ for all adjacent nodes in $\mathbb{V}_{\mathbb{T}_u}$ since $w\notin\mathbb{V}_{\mathbb{T}_u}$. Hence, node w is not structure-reachable from node v. This contradicts the assumption, which yields Lemma 8. □

Lemma 8 indicates no clusters and local clusters cross several converged TARs; all local clusters in $\mathbb{V}_{\mathbb{T}_u}$ only partition $\mathbb{V}_{\mathbb{T}_u}$ derived from a single converged TAR. For example, in Figure 1(c), we have a candidate clusters $\mathbb{V}_{\mathbb{T}_{u_0}}=\{u_0,u_1,u_2,u_4,u_5,u_6\}$ that has three local clusters $\mathbb{L}[u_0]=\{u_0,u_1,u_5,u_6\}$, $\mathbb{L}[u_2]=\{u_1,u_2,u_6\}$, and $\mathbb{L}[u_4]=\{u_4,u_5,u_6\}$. Since $\mathbb{V}_{\mathbb{T}_{u_0}}$ is adjacent to only u_3 in $\overline{\mathbb{V}}_{\mathbb{T}_{u_0}}$ with $\sigma(u_2,u_3)=0.41<\epsilon$ and $\sigma(u_3,u_4)=0.41<\epsilon$, it clearly satisfies Lemma 7 and all of the local clusters ($\mathbb{L}[u_0]$, $\mathbb{L}[u_2]$, and $\mathbb{L}[u_4]$) are included in $\mathbb{V}_{\mathbb{T}_{u_0}}$.

From Lemma 8 and Definition 4, which defines clusters derived by SCAN, SCAN++ finds the same clusters as SCAN if it satisfies the following conditions in each candidate clusters $\mathbb{V}_{\mathbb{T}_u}$: (1) SCAN++ finds all cores in $\mathbb{V}_{\mathbb{T}_u}$, (2) SCAN++ finds all nodes in $\mathbb{D}[u]$, where node u is core, as the structure-reachable nodes of node u on $\mathbb{V}_{\mathbb{T}_u}$. We prove SCAN++ satisfies the conditions below.

(1) Core completeness: The following lemma demonstrates that SCAN++ finds all cores from a candidate clusters:

LEMMA 9 (CORE COMPLETENESS). *SCAN++ finds all cores included in candidate clusters $\mathbb{V}_{\mathbb{T}_u}$.*

PROOF. From Definition 12, nodes in $\mathbb{V}_{\mathbb{T}_u}$ are divided into pivots and bridges. As shown in Section 3.3.1, SCAN++ finds all cores from pivots since it computes the similarities for all adjacent nodes of the pivots in the local clustering phase. In addition, SCAN++ finds all cores from the bridges. There are two reasons: First, as we described in Section 3.3.2, if bridges are adjacent to more than μ pivots with structural similarity that exceeds ϵ, the bridges are regarded as core in the cluster refinement phase. Second, as shown in Algorithm 1 (line 23-25), SCAN++ computes the similarities for all remaining bridges. Thus, SCAN++ finds all cores in $\mathbb{V}_{\mathbb{T}_u}$. □

(2) Structure reachability: To demonstrate the condition (2), which we described above, we introduce the following property:

LEMMA 10 (BRIDGE CONNECTIVITY). *Let $\mathbb{P}$ and $\mathbb{B}$ be sets of pivots and bridges,respectively. We have the following property:*

$$b_i,b_j\in\mathbb{B} \text{ and } b_j\in\mathbb{N}[b_i]\Rightarrow\exists p\in\mathbb{P} \text{ s.t. } b_i,b_j\in\mathbb{N}_\epsilon[p]. \quad (9)$$

PROOF. We prove by contradiction. We assume that we have $b_i,b_j\in\mathbb{B}$ *and* $b_j\in\mathbb{N}[b_i]\Rightarrow p\notin\mathbb{P}$ *s.t.* $b_i,b_j\in\mathbb{N}_\epsilon[p]$. From Definition 6, either b_i or b_j should be a pivot if $p\notin\mathbb{P}$ *s.t.* $b_i,b_j\in\mathbb{N}_\epsilon[p]$. This contradict $b_i,b_j\in\mathbb{B}$, hence we have Lemma 10. □

From Lemma 10, we prove the condition (2) as follows:

LEMMA 11 (STRUCTURE REACHABILITY). *Let node u be a core in $\mathbb{V}_{\mathbb{T}_u}$ and $\mathbb{D}[u]$ be directly structure-reachable nodes of node u derived by SCAN. All nodes in $\mathbb{D}[u]$ are included in the structure-reachable nodes of node u on $\mathbb{V}_{\mathbb{T}_u}$.*

PROOF. If node u is a pivot, SCAN++ clearly satisfies Lemma 11 from the DTAR definition in Definition 6. Next, if node u is a bridge, nodes in $\mathbb{D}[u]$ are divided into pivots and bridges on $\mathbb{V}_{\mathbb{T}_u}$. From Definition 6, SCAN++ clearly satisfies Lemma 11 for pivots in $\mathbb{D}[u]$. Similarly, from Lemma 10, SCAN++ satisfies Lemma 11 for the bridge b in $\mathbb{D}[u]$. This is because we have a node $p\in\mathbb{P}$ *s.t.* $u,b\in\mathbb{N}_\epsilon[p]$ from Lemma 10. Hence, if node p is a core, node b is included in the structure-reachable nodes of node u by Definition 6; otherwise, node b is a directly structure-reachable node on $\mathbb{V}_{\mathbb{T}_u}$ by Algorithm 1 (line 23-25). Thus, if node u is a bridge, SCAN++ satisfies Lemma 11, which proves Lemma 11. □

Finally, we have the following theorem from Lemma 9 and 11:

THEOREM 3 (EXACTNESS OF SCAN++). *SCAN++ always has exactly the same clustering results as SCAN.*

PROOF. From Lemma 9, 11 and Definition 4, it is clear that SCAN++ has exactly same clustering results as SCAN. □

Table 2: Real-world datasets

Dataset	$\lvert\mathbb{V}\rvert$	$\lvert\mathbb{E}\rvert$	Graph type	c
condmat	23,133	186,936	Social network	0.6334
slashdot	77,360	905,468	Social network	0.0555
amazon	334,863	925,872	Web graph	0.3967
dblp	317,080	1,049,866	Social network	0.6324
road	1,379,917	3,843,320	Road network	0.0470
google	875,713	5,105,039	Web graph	0.5143
cnr	325,557	5,477,938	Web graph	0.5586
skitter	1,696,415	11,095,298	Computer network	0.2581
uk-2002	18,520,486	298,113,762	Web graph	0.6891
webbase	118,142,155	1,019,903,190	Web graph	0.5533

5. EXPERIMENTS

We compared the effectiveness of four algorithms including our proposed method SCAN++.

- **SCAN++**: our proposal.
- **SCAN***: a simple variation of SCAN that produces approximate results by utilizing the edge sampling technique proposed by the state-of-the-art method LinkSCAN* [21]. Based on LinkSCAN*, SCAN* samples $\min\{d_u, \alpha+\beta \ln d_u\}$ edges for each node, where d_u is the degree of a node and both α and β are user-specified parameters. We set $\alpha = 2|\mathbb{E}|/|\mathbb{V}|$ and $\beta = 1$ as recommended by LinkSCAN*.
- **SCAN**: the original algorithm [36].
- **gSkeletonClu**: a state-of-the-art algorithm extended from SCAN that provides us parameter-free structural clustering [14]. gSkeletonClu employs the tree-decomposition-based algorithm and it searches clustering results that maximize the score of modularity [26].

All experiments were conducted on a Linux 2.6.18 server with one CPU (Intel Xeon Processor L5640 2.27GHz) and 144GBytes of main memory. SCAN++, SCAN* and SCAN were implemented in C/C++ as single-threaded programs, which use a single CPU core with the entire graph held in the main memory. Also, we used gcc-g++ 4.8.1 compiler with optimization parameter "-O2" for each algorithm. To evaluate the other algorithm, we used the program of gSkeletonClu published on their authors' sites[1].

The experiments used 10 public datasets published by Standard Network Analysis Project[2] and Laboratory of Web Algorithmics[3]. The statistics of each dataset are shown in Table 2. In the right most column, c shows the average clustering coefficient. Additionally, in order to evaluate the effectiveness of our algorithm, we also used synthetic datasets generated by LFR benchmark [18], which is considered as the *de facto standard* model for generating graphs. The settings will be detailed later.

5.1 Evaluation on Real-world Datasets

5.1.1 Efficiency

We evaluated the clustering performance of each method through wall clock time for the real-world datasets. In this evaluation, we fixed the parameter $\mu = 5$ and varied the parameter ϵ as 0.2, 0.4, 0.6 and 0.8 for each algorithm. Figure 2 shows the running time for each real-world dataset. Since existing algorithm show almost same results under all parameter settings, we omitted the results of them from Figure 2 except for $\epsilon = 0.6$. In addition, we omitted the results of gSkeletonClu, SCAN*, and SCAN for several large datasets since they cannot compute clusters in a day.

[1] `http://web.xidian.edu.cn/jbhuang/en/publications.html`
[2] `http://snap.stanford.edu`
[3] `http://law.di.unimi.it/datasets.php`

Figure 2 shows that SCAN++ is much faster than existing approaches under all conditions examined. Of particular interest, SCAN++ is 20.4 times faster than SCAN on average, and it is also a few orders of magnitude faster than gSkeletonClu. As described in Section 2, SCAN subjects all adjacent nodes in the given graph to structural similarity computations. Furthermore, SCAN incurs average computation time of $O(|\mathbb{E}|/|\mathbb{V}|)$ for each structural similarity computation. Hence, SCAN requires $O(|\mathbb{E}|^2/|\mathbb{V}|)$ time on average. Similar to SCAN, for finding clustering results that maximizes modularity, gSkeletonClu has to extract spanning trees from the graph by computing structural similarities for all adjacent nodes. Therefore, as shown in Figure 2, gSkeletonClu requires significantly larger computation times for clustering than SCAN++ or SCAN. In contrast to both SCAN and gSkeletonClu, as shown in Section 3, SCAN++ employs two efficient clustering approaches, (1) two-phase clustering and (2) similarity sharing that utilizes the clustering coefficient. As a result, as shown in Theorem 1, SCAN++ only requires time complexity $O(\frac{2-c}{2\delta+c}|\mathbb{E}|)$. Therefore, SCAN++ finds clustering results much more efficiently than SCAN*, SCAN and gSkeletonClu.

Figure 2 also shows that SCAN* could be competitive with our proposal SCAN++ for slashdot in terms of efficiency. This is due to former's use of the clustering coefficient of the given graph. As shown in Table 2, slashdot has a significantly lower clustering coefficient than the other datasets. SCAN++ could not reduce the running time enough by using two-phase clustering and similarity sharing since the small graphs had low clustering coefficients. Even though road and skitter have relatively lower clustering coefficients than the other datasets, SCAN++ was much faster than SCAN*. There are two reasons. First, road and skitter are much larger graphs than slashdot. If graph size is large enough, SCAN++ can reduce the computation time even if the clustering coefficients are small. Second, each node in road has almost the same degree while slashdot has a skewed degree distribution. As we described, SCAN* eliminates edges from the graph when the degree of each node is large enough. However, the nodes in road have almost the same degree; hence SCAN* could not effectively eliminate edges from the dataset. Therefore, SCAN++ ran faster than SCAN* for road and skitter. Although, SCAN* is efficient for small graphs with lower clustering coefficients, it is an approximation approach based on SCAN and so can not match the clustering performance of the other methods. We will discuss this point in Section 5.1.4.

In all conditions examined, the running time of the cluster refinement phase described in Section 3.3 is negligible. Specifically, SCAN++ consumed less than 1% of its running time for merging clusters under all conditions examined. This is because, in the real-world datasets with high clustering coefficients, each bridge is adjacent to many pivots with high structural similarity scores. Hence, as shown in Lemma 2, most bridges are prunable bridges, and they do not require additional similarity computations for merging local clusters. We omit the detailed results of the running time for merging local clusters due to space limitations.

Our experiments also considered different parameter μ settings. Figure 3 shows the running time of SCAN++ for the five small datasets for various values of μ. As shown in Figure 3, the values of μ have no significant impact for the running time of SCAN++. This is because the running time of the cluster refinement phase consumes at most 1% of the total running time. Although we omit the results of the other algorithms from Figure 3 and the results of the large datasets due to space limitations, all the other methods shows almost same results as Figure 2. SCAN++ can find clusters, hubs, and outliers more efficiently than the existing approaches even under different parameter settings.

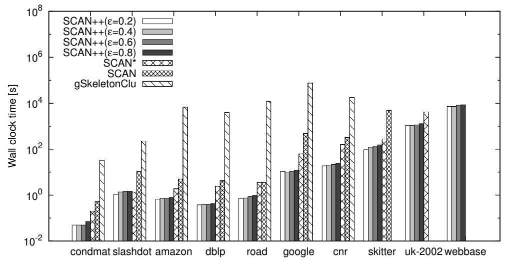


Figure 2: Runtime for real-world datasets

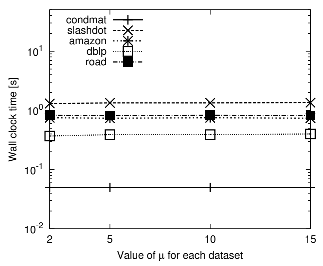


Figure 3: Effect of parameter μ

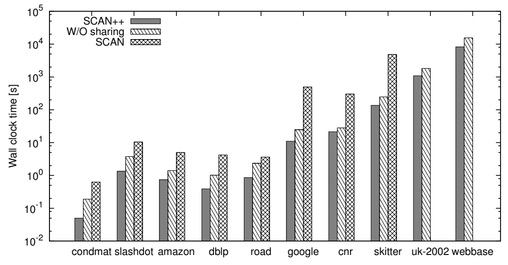


Figure 4: Effect of key techniques

Table 3: Memory usage for real-world datasets

Dataset	SCAN++ $\epsilon = 0.2$	SCAN++ $\epsilon = 0.4$	SCAN++ $\epsilon = 0.6$	SCAN++ $\epsilon = 0.8$	SCAN
condmat	8.0M	7.4M	7.0M	6.8M	6.4M
slashdot	12M	11M	10M	10M	9.3M
amazon	44M	37M	31M	30M	29M
dblp	40M	29M	27M	26M	25M
road	98M	97M	93M	93M	92M
google	119M	103M	98M	96M	96M
cnr	59M	57M	57M	55M	52M
skitter	319M	235M	212M	209M	207M
uk-2002	4.1G	3.7G	3.6G	3.6G	-
webbase	17G	15G	15G	15G	-

5.1.2 Effectiveness of the Key Techniques

As mentioned in Section 3.3 and 3.4, we used two-phase clustering and similarity sharing to prune unnecessary computations. In order to show the effectiveness of our approach, we evaluated the running time of a variant method of SCAN++ that did not use similarity sharing. In this evaluation, we fixed parameter $\epsilon = 0.6$ and $\mu = 5$ for each algorithm. Figure 4 shows the runtime of each algorithm to find clusters, hubs, and outliers from the real-world datasets. In this figure, "W/O sharing" represents the results of the variant method of SCAN++ that did not use similarity sharing. Figure 4 shows that SCAN++ without similarity sharing is 8.76 times faster than the original algorithm SCAN on average. On the other hand, the comparison of "SCAN++" and "W/O sharing" in Figure 4 shows that the runtime can be cut further; SCAN++ is 2.33 times faster than W/O sharing on average. This indicates that two-phase clustering contributes most of the improvement of SCAN++, as demonstrated in Section 5.1.1. As shown in Figure 4, SCAN++ more efficiently reduce the computation cost than "W/O sharing" by combining two-phase clustering and similarity sharing.

5.1.3 Memory usage

Section 4.2 theoretically shows SCAN++ incurs $O(|\mathbb{E}| + |\mathbb{V}| + l)$ space complexity. This section experimentally verifies our theoretical analysis for real-world datasets. Table 3 summarizes the measured memory usages of SCAN++ and SCAN. Table 3 demonstrates that SCAN++ increases its memory usage by the sizes of $|\mathbb{V}|$ and $|\mathbb{E}|$. In addition, Table 3 shows that SCAN can be competitive with SCAN++ according to parameter ϵ. This is because, as shown in Definition 9, the size of each local cluster is determined by the value of ϵ. If we set ϵ to a small value, SCAN++ produces a lot of large size of local clusters. Thus, the size of l becomes large and SCAN++ increases its memory usage for the parameter settings. These results verify our theoretical analysis in Section 4.2.

5.1.4 Exactness

One major contribution of SCAN++ is that it outputs exactly same clustering results as SCAN. To demonstrate the exactness of the clustering results, we evaluated accuracy of obtaining cores and clustering results against SCAN for each dataset. In this experiment, we used two measures of accuracy, *F-measure* [25] and *adjusted rand index (ARI)* [25], for cores and clusters, respectively. F-measure quantifies the accuracy of the clustering results by calculating the harmonic mean of precision and recall. Hence, we defined precision and recall as follows: precision is the fraction of cores by each method that matches those of SCAN, and recall is the fraction of cores obtained by SCAN that are also extracted by each method. F-measure takes a value between 0 and 1, and F-measure is 1 if the obtained cores exactly match those by SCAN. ARI is a measure of the similarity between two clustering results. ARI has a value between 0 and 1, and it shows 1 if the two clustering results are completely same. Figure 5 and Figure 6 show F-measure and ARI scores of each algorithm compared to the clustering results of SCAN, respectively. In this evaluation, we set the parameters of each algorithm as $\epsilon = 0.6$ and $\mu = 5$. We omitted the results of uk-2002 and webbase since, as shown in Section 5.1.1, SCAN did not return the results in a day.

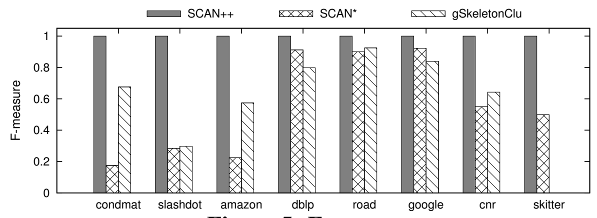


Figure 5: F-measure

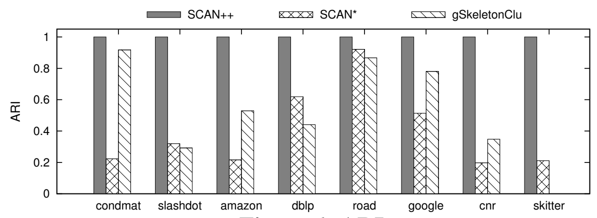


Figure 6: ARI

Figure 5 indicates that SCAN++ obtains exactly same cores as SCAN. In addition, Figure 6 indicates that SCAN++ extracts the same clustering results as SCAN since it shows the ARI scores equal to 1 for all datasets. As shown in Theorem 3, SCAN++ guarantees of outputting the same clustering results as SCAN even though we drops unnecessary similarity computations. Hence, F-measure and ARI of SCAN++ were 1 as shown in Figure 5 and Figure 6, respectively. On the other hand, SCAN* and gSkeletonClu output clustering results that differ from those of SCAN. This is because SCAN* is an approximation method that samples subset of edges from the given graph and gSkeletonClu employs the clustering results that maximize modularity [26]. Figure 5 and 6, as well as Figure 2, confirms that SCAN++ is superior to existing methods since SCAN++ is more efficient than existing methods without sacrificing the accuracy compared to SCAN.

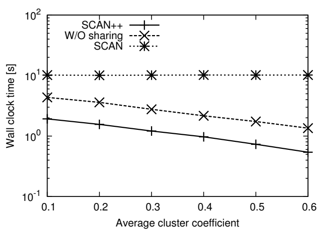


Figure 7: Effect of c

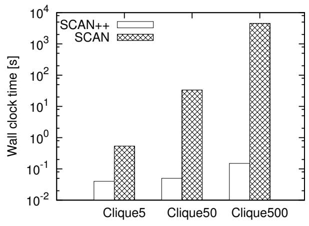


Figure 8: Runtime of cliques

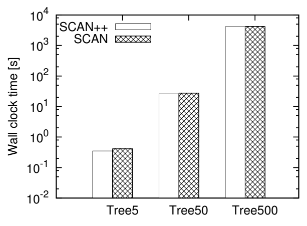


Figure 9: Runtime of trees

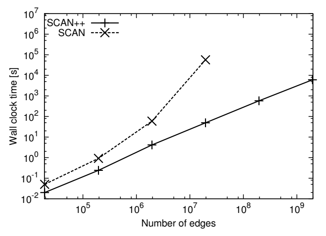


Figure 10: Scalability

5.2 Evaluation on Synthetic Datasets

5.2.1 Effectiveness of Clustering Coefficient

We evaluated the effectiveness of SCAN++ in terms of high clustering coefficients by using synthetic graphs. We generate LFR benchmark graphs with 100 thousand nodes; the average clustering coefficient was varied from 0.1 to 0.6 following the real-world datasets in Table 2. The other parameters, average degree and maximum degree, were fixed at 20 and 50, respectively. Figure 7 shows runtimes of SCAN++, W/O sharing in Section 5.1.2 and SCAN for different c scores. As shown in Figure 7, SCAN shows almost constant computation time under all conditions examined. Unlike SCAN, SCAN++ and W/O sharing increased their clustering speed as c increased. In the most efficient case (*i.e.* $c = 0.6$) our proposals were up to three times faster than the result of the worst case (*i.e.* $c = 0.1$). These results imply that our two-hop-away node based algorithm effectively prunes the candidates that are assessed. Thus, our algorithm outperforms SCAN when the given graph has high c scores as is likely with real-world graphs.

As shown in Figure 7, the runtime of SCAN++ depends on the clustering coefficient, hence we additionally evaluated two special graphs, "clique" and "tree", whose clustering coefficients are close to $c = 1$ and $c = 0$, respectively. We evaluated "clique" based on the *connected caveman graph* [33]. The connected caveman graph is a graph consisting of a ring of several k-cliques. We generated three connected caveman graphs with 100 thousand nodes; the clique size k of the three graphs were fixed as 5, 50, and 500 that are referred as Clique5, Clique50, and Clique500, respectively. Also, we generated "trees" based on the balanced tree with 100 thousand nodes; each node had at most k children. We generated three trees by varying k as 5, 50, and 500 that are referred as Tree5, Tree50, and Tree500, respectively. Figure 8 and 9 show runtimes for the cliques and the trees, respectively. Figure 8 shows that SCAN++ runs significantly faster than SCAN for the cliques since the graphs have high clustering coefficient of close to 1. In this case, $c \approx 1$ and $\delta \approx 0$ since the average degree closes to $|\mathbb{V}|$. Thus, from Theorem 1, the time complexity of SCAN++ becomes $O(\frac{2-c}{2\delta+1}|\mathbb{E}|) = O(|\mathbb{E}|)$. Therefore, SCAN++ can run significantly faster than SCAN for the cliques. On the other hand, Figure 9 shows that SCAN++ is competitive with SCAN for the trees since they have low clustering coefficient of close to 0. As shown in Theorem 1, if the clustering coefficient is 0, the time complexity of SCAN++ is $O(|\mathbb{E}|^2/|\mathbb{V}|)$, which is the same as SCAN. Thus, SCAN++ requires almost the same runtime as SCAN for the trees.

5.2.2 Scalability

We evaluated the scalability of SCAN++ and SCAN. We generated LFR benchmark graphs with various numbers of nodes from 1 thousand to 100 million. The other parameters, average degree, maximum degree and clustering coefficient, were fixed at 20, 50 and 0.4, respectively. We omitted the results of SCAN for the graphs with 10 million and 100 million nodes since it did not return the clustering results in a day. The running times of the algorithms in Figure 10, show that SCAN++ has near-linear runtime in terms of number of edges. In contrast, SCAN significantly increases its running time as the size of edges increases. This result verifies our theoretical analysis in Section 4.1, hence, our proposals are scalable for large-scale graphs.

6. RELATED WORK

The problem of finding clusters in a graph has been studied for some decades in many fields, particularity in computer science and physics [12]. Graph partitioning algorithms [27, 30] are natural choices for this problem. Since cluster structures are highly complex, several clustering algorithms have been recently introduced. Here we review some of the more successful methods.

Modularity-based algorithms: Modularity, proposed by Newman and Girvan [26], is widely used to evaluate the cluster structure of a graph from global perspective. Modularity is a quality metric of graph clustering; it measures the difference of the graph structure from that of a random graph. The main idea of modularity is to find groups of nodes that have a lot of inner-group edges and few inter-group edges; optimal clustering is achieved when modularity is maximized. Although modularity-based algorithms are effective in many applications, finding the maximum modularity is an NP-complete problem. This has led to the introduction of approximation approaches. Instead of performing exhaustive computations, various greedy algorithms based on hierarchical agglomeration clustering have been proposed such as CNM [3] and BGLL [2]. Recently, Shiokawa *et al.* proposed an incremental aggregation based clustering algorithm for finding clusters in large-scale graphs [28]. However, despite the efficiency of the approach, these methods cannot identify hubs and outliers in graphs. Furthermore, recent research pointed out that modularity is not a scale-invariant measure [10]. Hence, it is difficult for modularity-based methods to find small clusters hidden in large-scale graphs; these methods fail to fully reproduce the ground-truth [36]. This serious problem is famously known as the *resolution limit* of modularity [10]. Unlike traditional modularity-based algorithms, we use structural similarity [36], which overcomes the resolution limit, as the clustering measure. Hence, our proposal can obtain better clustering results than modularity-based methods.

Structural clustering algorithms: Due to the resolution limit of modularity-based methods, structural clustering algorithms [13, 14, 21, 36] have been widely used in many applications in the last few years. SCAN [36], proposed by Xu *et al.*, is the most popular method based on structural similarity. It is an extension of the traditional density based clustering method DBSCAN [8]. Unlike traditional density-based algorithms [15] and clique detection methods [29, 31], this algorithm can successfully find clusters as well as hubs and outliers in a graph by specifying two parameters ϵ and μ. Moreover, Xu *et al.* reported that SCAN outperforms modularity-based methods in producing clustering results that resemble the ground-truth [36]. However, as described in Section 2, this algorithm incurs an average cost of $O(|\mathbb{E}|^2/|\mathbb{V}|)$. Hence, SCAN incurs large computation time for large-scale graphs.

Huang *et al.,* proposed the two parameter-free methods named SHRINK [13] and gSkeletonClu [14]. SHRINK is a hierarchical clustering algorithm that combines the advantages of structural similarity and modularity-based methods. It first computes similarities for all adjacent nodes. Then it aggregates densely connected nodes into the same clusters if the aggregation improves the modularity score. In this way, SHRINK achieves parameter-free and hierarchical clustering. In contrast, gSkeletonClu tries to find clustering results that maximize the modularity by using the tree-decomposition-based algorithm. As with SHRINK, it first computes the structural similarities for all adjacent nodes in the graph. After that, it extracts maximum spanning trees from the graph by using the scores of the structural similarities; and then it searches better clustering results in terms of modularity from the trees. SHRINK and gSkeletonClu are user-friendly algorithms since they do not require user-specified parameters. However, these methods requires exhaustive similarity computations for all adjacent nodes; hence the time complexities of SHRINK and gSkeletonClu are at least $O((|\mathbb{E}|^2 \log|\mathbb{V}|)/|\mathbb{V}|)$ and $O(|\mathbb{E}|^2/|\mathbb{V}| + |\mathbb{V}| \log|\mathbb{V}|)$, respectively. Hence, as well as SCAN, both methods incur large computation time for large-scale graphs.

Recently, Lim *et al.* proposed LinkSCAN*, which uses SCAN to find overlapping communities. For detecting overlapping communities, LinkSCAN* transforms the graph into a link-space graph which combines the advantages of the graph and line graph [9]. This transformation entails an increase of the size of the graph for clustering, hence they introduced a graph sampling step, which we used for SCAN* in Section 5. This approach is certainly efficient in reducing the computation time; however, as shown in Figure 5, it degrades the clustering results compared to SCAN since sampling involves approximation. By using SCAN++ instead of the graph sampling approach, we can improve the performance of LinkSCAN* since our proposal is not only efficient but also exact.

Our work is different from these traditional algorithms in that it provides efficient clustering with no loss in clustering quality from SCAN. Our theoretical analyses and experiments show that SCAN++ incurs $O(\frac{2-c}{2\delta+c}|\mathbb{E}|)$ time complexity and much faster clustering than the traditional methods.

7. CONCLUSION

This paper addressed the problem of efficiently finding clusters, hubs, and outliers in large-scale graphs. Our proposal, SCAN++, is based on three ideas: (1) it introduces a new data structure, called directly two-hop-away reachable (DTAR), that contains only nodes that two hops away from a given node, (2) it drops unnecessary density evaluations for adjacent nodes in the clustering procedure by using DTARs, and (3) its density evaluation method is highly efficient since it shares some of the density evaluation results of DTARs. As far as we know, this is the first study to introduce a graph clustering algorithm that achieves both high efficiency and exactly same clustering results as SCAN at the same time. Graph clustering algorithms that extract not only clusters but also hubs and outliers are essential for many applications. The proposal will help to improve the effectiveness of current and future applications.

8. REFERENCES

[1] Apache Giraph. `http://giraph.apache.org/`.

[2] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast Unfolding of Communities in Large Networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008:P10008, October 2008.

[3] A. Clauset, M. E. J. Newman, and C. Moore. Finding Community Structure in Very Large Networks. *Phys. Rev. E*, 70:066111, Dec 2004.

[4] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Proc. OSDI*, pages 107–113, 2004.

[5] R. Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

[6] Y. Ding, M. Chen, Z. Liu, D. Ding, Y. Ye, M. Zhang, R. Kelly, L. Guo, Z. Su, S. Harris, F. Qian, W. Ge, H. Fang, X. Xu, and W. Tong. atBioNet An Integrated Network Analysis Tool for Genomics and Biomarker Discovery. *BMC Genomics*, 13(1), 2012.

[7] P. Domingos and M. Richardson. Mining the Network Value of Customers. In *Proc. KDD*, pages 57–66, 2001.

[8] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proc. KDD*, pages 226–231, 1996.

[9] T. S. Evans and R. Lambiotte. Line Graphs, Link Partitions, and Overlapping Communities. *Phys. Rev. E*, 80(1):016105, July 2009.

[10] S. Fortunato and M. Barthélemy. Resolution Limit in Community Detection. *Proceedings of the National Academy of Sciences*, 104(1):36–41, 1 2007.

[11] Y. Fujiwara, G. Irie, S. Kuroyama, and M. Onizuka. Scaling Manifold Ranking Based Image Retrieval. *PVLDB*, 8(4):341–352, 2014.

[12] Y. Fujiwara, M. Nakatsuji, H. Shiokawa, Y. Ida, and M. Toyoda. Adaptive Message Update for Fast Affinity Propgation. In *Proc. KDD*, 2015.

[13] J. Huang, H. Deng, H. Sun, Y. Sun, J. Han, and Y. Liu. SHRINK: A Structural Clustering Algorithm for Detecting Hierarchical Communities in Networks. In *Proc. CIKM*, pages 219–228, 2010.

[14] J. Huang, H. Sun, Q. Song, H. Deng, and J. Han. Revealing Density-Based Clustering Structure from the Core-Connected Tree of a Network. *IEEE TKDE*, 25(8):1876–1889, 2013.

[15] P. Jiang and M. Singh. SPICi: A Fast Clustering Algorithm for Large Biological Networks. *Bioinformatics*, 26(8):1105–1111, 2010.

[16] U. Kang and C. Faloutsos. Beyond 'Caveman Communities': Hubs and Spokes for Graph Compression and Mining. In *Proc. ICDM*, pages 300–309, 2011.

[17] J. M. Kleinberg. Authoritative Sources in a Hyperlinked Environment. *J. ACM*, 46(5):604–632, Sept. 1999.

[18] A. Lancichinetti and S. Fortunato. Community Detection Algorithms: A Comparative Analysis. *Phys. Rev. E*, 80:056117, Nov 2009.

[19] M. Latapy, C. Magnien, and N. D. Vecchio. Basic Notions for the Analysis of Large Two-mode Networks. *Social Networks*, 30(1):31–48, 2008.

[20] P. Lee, L. V. S. Lakshmanan, and E. E. Milios. Incremental Cluster Evolution Tracking from Highly Dynamic Network Data. In *Proc. ICDE*, pages 3–14, 2014.

[21] S. Lim, S. Ryu, S. Kwon, K. Jung, and J.-G. Lee. LinkSCAN*: Overlapping Community Detection Using the Link-space Transformation. In *Proc. ICDE*, pages 292–303, 2014.

[22] Z. Liu, Q. Shi, D. Ding, R. Kelly, H. Fang, and W. Tong. Translating Clinical Findings into Knowledge in Drug Safety Evaluation - Drug Induced Liver Injury Prediction System (DILIps). *PLoS Computational Biology*, 7(12), 2011.

[23] T. Lou and J. Tang. Mining Structural Hole Spanners Through Information Diffusion in Social Networks. In *Proc. WWW*, pages 825–836, 2013.

[24] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: A System for Large-scale Graph Processing. In *Proc. SIGMOD*, pages 135–146, 2010.

[25] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[26] M. E. J. Newman and M. Girvan. Finding and Evaluating Community Structure in Networks. *Phys. Rev. E*, 69:026113, Feb 2004.

[27] J. Shi and J. Malik. Normalized Cuts and Image Segmentation. *IEEE TPAMI*, 22(8):888–905, 2000.

[28] H. Shiokawa, Y. Fujiwara, and M. Onizuka. Fast Algorithm for Modularity-based Graph Clustering. In *Proc. AAAI*, pages 1170–1176, 2013.

[29] J. Wang and J. Cheng. Truss Decomposition in Massive Networks. *PVLDB*, 5(9):812–823, 2012.

[30] L. Wang, Y. Xiao, B. Shao, and H. Wang. How to Partition a Billion-Node Graph. In *Proc. ICDE*, pages 568–579, 2014.

[31] N. Wang, J. Zhang, K. Tan, and A. K. H. Tung. On Triangulation-based Dense Neighborhood Graphs Discovery. *PVLDB*, 4(2):58–68, 2010.

[32] Y. Wang, D. Chakrabarti, C. Wang, and C. Faloutsos. Epidemic Spreading in Real Networks: an Eigenvalue Viewpoint. In *Proc. SRDS*, pages 25–34, 2003.

[33] D. J. Watts. *Small Worlds: The Dynamics of Networks Between Order and Randomness*. Princeton studies in complexity. Princeton University Press, 1999.

[34] D. J. Watts and S. H. Strogatz. Collective Dynamics of 'Small-World' Networks. *Nature*, 393(6684):409–10, 1998.

[35] B. Wu and B. D. Davison. Identifying Link Farm Spam Pages. In *Proc. WWW*, pages 820–829, 2005.

[36] X. Xu, N. Yuruk, Z. Geng, and T. A. J. Schweiger. SCAN: A Structural Clustering Algorithm for Networks. In *Proc. KDD*, pages 824–833, 2007.

[37] Y. Zhou, H. Cheng, and J. X. Yu. Clustering Large Attributed Graphs: an Efficient Incremental Approach. In *Proc. ICDE*, pages 689–698, 2010.