An Efficient Algorithm for Distance-based Structural Graph Clustering

Anonymous Author(s)

ABSTRACT

Structural graph clustering (SCAN) is a classic graph clustering algorithm. In SCAN, a key step is to compute the structural similarity between vertices according to the overlap ratio of one-hop neighborhoods. Given two vertices u and v, existing studies only consider the case when u and v are neighbors. However, the structural similarity between non-neighboring vertices in SCAN is always zero, and using only one-hop neighbors on weighted graphs discards the weights on each edge. Both may not reflect the true closeness of two vertices and may fail to return high-quality clustering results.

To tackle this issue, we define and study the distance-based structural graph clustering problem. Given a distance threshold d and two vertices u and v, the structural similarity between u and v is defined as the ratio of their respective neighbors within a distance of no more than d. We show that the newly defined distance-based SCAN achieves better clustering results compared to the vanilla version of SCAN. However, the new definition brings challenges in the computation of final clustering results. To tackle this efficiency issue, we propose DistanceSCAN, an efficient approximate algorithm for solving the distance-based SCAN problem. The main idea of DistanceSCAN is to use *all-distances bottom-k sketches (ADS)* to speed up the computation of similarities. Given the ADS, we can derive the similarity between two vertices with a bounded cost of $O(k)$. However, to ensure that the estimated similarity has an approximation guarantee, the value of k still needs to be set to as large as thousands. This brings high computational costs when computing the similarities between neighboring vertices. To tackle this issue, we further construct histograms to prune the structural similarity computations of vertices pairs. Extensive experiments on real datasets validate the effectiveness and efficiency of DistanceSCAN.

ACM Reference Format:
Anonymous Author(s). 2022. An Efficient Algorithm for Distance-based Structural Graph Clustering. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (SIGMOD'23).* ACM, New York, NY, USA, 15 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Graph clustering is the task to group vertices into clusters by taking the topology into consideration and has wide applications, such as community detection in social networks [22], load balancing in distributed systems [2] and biological network analysis [14]. Different from many existing solutions, e.g., [11, 22], structural graph clustering (SCAN) [31] can not only detect clusters, but also hubs and outliers. Hubs and outliers do not belong to any cluster, but hubs bridge different clusters. It is of great significance to identify hubs in many fields such as viral marketing [12], epidemiology [28] and graph compression [19]. Therefore, a large number of researches on SCAN have recently emerged, e.g., [6, 24, 29].

The idea of SCAN is derived from the famous density-based clustering algorithm DBSCAN [13]. For a given vertex u and its neighbor v, the structural similarity is computed according to the ratio of the common neighbors in the one-hop neighborhoods of the two vertices. If a vertex u has many neighbors with high structural similarity, u will be classified as a core vertex. For the currently discovered core vertex, SCAN traverses its one-hop neighborhood and add neighbors with high structural similarity to the cluster containing the current core vertex. However, it may not be a good option to consider only one-hop neighbor on weighted graphs where the weight on each edge indicates the closeness between these two vertices. Thus, on such graphs, it is difficult for SCAN to make full use of the weight information, and the quality of the clustering results computed by SCAN is often unsatisfactory.

In order to obtain high-quality clustering results on weighted graphs, we define distance-based structural graph clustering. Given a distance threshold d and two vertices with distance no more than d, we compute the similarity of two vertices according to the overlap of their d-neighborhoods, where the d-neighborhood of a vertex u is the set of all vertices whose distance to the vertex u does not exceed d. The ratio of common neighbors in d-neighborhood better reflects the closeness between vertices on weighted graphs. Below we give several examples.

Example 1.1. According to the statistical results of the academic network DBLP [1] as of March 5, 2022, Christian S. Jensen and Richard T. Snodgrass have 114 co-published articles, but only 69 of their 590 neighbors are common neighbors, and the Jaccard similarity is 0.117 and the Cosine similarity is 0.234, which are tended to be regarded as dissimilar by SCAN. The more co-published articles, the closer the distance between the two vertices. If the structural similarity is calculated based on the d-neighborhood, the structural similarity of Christian S. Jensen and Richard T. Snodgrass tends to be high. Also according to DBLP, Philip S.Yu and Jiawei Han have 63 co-published articles, Xiang Ren and Jiawei Han have 76 co-published articles, but Philip S.Yu and Xiang Ren have no co-published articles. Since the structural similarity is only calculated for vertices in one-hop neighborhood, SCAN is difficult to deal with such structures, and can not give the proper structural similarity between Xiang Ren and Philip S. Yu. As Xiang Ren and Philip S. Yu are both close to Jiawei Han, their neighbors in the d-neighborhood tend to have a lot of overlap. Thus, the structural similarity between Xiang Ren and Philip S. Yu calculated by d-neighborhood tends to be high.

SIGMOD'23, June 18–23, 2023, Washington, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

[1] https://dblp.uni-trier.de/

Challenges. Existing work mainly studies SCAN by considering one-hop neighbors. To the best of our knowledge, this is the first work on distance-based structural graph clustering. There are several challenges to solve the distance-based structural graph clustering problem. The computations of structural similarity need to use the information of d-neighborhood. However, unlike the one-hop neighborhood, the d-neighborhoods of vertices in a weighted undirected graph are difficult to be obtained directly. Therefore, it is difficult to simply apply the structural graph clustering algorithms to the distance-based structural graph clustering problem. Computing d-neighborhoods for all vertices in an online manner requires performing shortest path computations for $O(n)$ vertices (n is the number of vertices in the graph), which incurs prohibitive computational costs and is infeasible on large graphs. If we precompute and save the d-neighborhoods of all vertices, we need to choose a large value of d to handle most queries, since the value of d for each query cannot be known in advance. In the worst case, the space complexity of the saved d-neighborhood of all vertices can reach $O(n^2)$. There is no explicit correlation between the structural similarity and the distance threshold d. As the value of the distance threshold d changes, the structural similarity between two vertices may increase or decrease, which makes it difficult to reuse the structural similarity results corresponding to different distance thresholds d. Therefore, the space complexity of directly caching the similarity results is also very high.

Our solution. To address the above challenges, we propose the DistanceSCAN algorithm. DistanceSCAN quickly calculates the clustering results of distance-based structural graph clustering through the following three aspects. First, in order to improve the efficiency of DistanceSCAN, we compute the structural similarity by bottom-k sketches [9] and all-distances bottom-k sketches [10]. Even though bottom-k sketches and all-distances bottom-k sketches are not new ideas, we make a connection to our studied distance-based SCAN problem and tackle the challenging task of structural similarity estimation. Second, we use histograms to quickly identify if the structural similarities of vertex pairs with small distances are above the given threshold, saving much computational cost of traversing the k sketch elements in bottom-k sketches. In each algorithm step, the structural similarity is preferentially calculated for vertex pairs with small distances. Finally, in order to reduce the number of structural similarity calculations, we immediately stop the correlation computation of each vertex after determining whether it is a core vertex. After all vertices are classified, the connectivity between the core vertices is checked to obtain the final clustering result.

Contributions. The contributions can be summarized as follows:

- The distance-based structural graph clustering is proposed and defined for the first time. By using the d-neighborhood as the basis for calculating the structural similarity, the distance information can be effectively utilized. The distance-based SCAN is a generalized definition of SCAN on weighted graphs. If all the weights of the graph and the value of d are equal, the distance-based SCAN can obtain the same clustering result as SCAN.
- We devised DistanceSCAN, an efficient algorithm for approximately computing clusters on weighted graphs. DistanceSCAN quickly computes structural similarities based on histograms and all-distances bottom-k sketches. After reading the graph and sketches, DistanceSCAN first detects core vertices and then classifies non-core vertices, hubs, and outliers. In order to reduce the calculation time of structural similarity, vertex pairs with small distances are preferentially calculated in each step.
- Extensive experiments show that the clustering results of distance-based SCAN achieve better clustering results compared to SCAN in terms of modularity and the proposed DistanceSCAN algorithm is more efficient than that of the baseline methods.

2 PRELIMINARIES

2.1 Problem Definition

In this paper, we focus on a weighted undirected graph $G = (V, E, w)$, where V is the set of vertices, $E \in V \times V$ is the set of edges, and $w : E \longrightarrow R^+$ is the weight function of edges. The sizes of vertices and edges are respectively represented by n, m, that is, $n = |V|, m = |E|$. The weight function w attaches a positive value to each edge to represent the distance between the two vertices. The smaller the distance, the closer the relationship between the two vertices. For a vertex $v \in V$, the one-hop neighborhood of v (denoted as $N(v)$) refers to the neighbors that are directly connected by edges, that is, $N(v) = \{u|(u, v) \in E\}$. In the following, for ease of presentation, we simply refer to a weighted undirected graph as a graph. The weight of a path is the sum of the weights of all the edges on the path. The distance $\delta(u, v)$ between vertices u and v is the weight of the shortest path between vertices u and v, which is formally defined as follows:

Definition 1 (Distance). *The distance between $u, v \in V$ is:*

$$\delta(u,v) = \begin{cases} \min\{w(p) : u \overset{p}{\rightsquigarrow} v\}, & \textit{if there is a path from } u \textit{ to } v. \\ \infty, & \textit{otherwise.} \end{cases}$$

In particular, we define the distance from the vertex to itself to be 0, that is, $\forall v \in V, \delta(v, v) = 0$. As mentioned before, computing structural similarity based on d-neighborhood tends to get better clustering results. The definition of d-neighborhood is as follows:

Definition 2 (d-neighborhood). *For a vertex $v \in V$, the d-neighborhood of v, $N_d(v)$, is the set of all vertices in the graph whose distance to vertex v does not exceed d, formally:*

$$N_d(v) = \{u \in V|\delta(u, v) \leq d\}. \tag{1}$$

Note that the vertex itself must be in the d-neighborhood of the vertex. In this paper, the structural similarity, $\sigma(u, v)$, of vertices u, v is measured using the Jaccard coefficient [18] of $N_d(u)$ and $N_d(v)$. In addition, the structural similarity can also be calculated using Cosine similarity or Dice similarity.

Definition 3 (Structural Similarity).

$$\sigma(u,v) = \frac{|N_d(u) \cap N_d(v)|}{|N_d(u) \cup N_d(v)|} \tag{2}$$

Given a threshold $\epsilon \in (0, 1)$, if the structural similarity between two vertices is no less than ϵ, we call these two vertices **structurally similar**, or simply **similar** if the context is clear. Structural similarity is also referred to simply as similarity. If the number of similar vertices in the d-neighborhood of a vertex v is large, the vertex v has a high "density" in the nearby area, and vertex v is likely to become a core vertex in a cluster.

Definition 4 (Core Vertex). *Given a similarity threshold $\epsilon \in R$, a threshold of similar neighbors $\mu \in N$ and a distance threshold $d \in R$, if there are no less than μ similar neighbors in $N_d(v)$, the vertex v is called a core vertex, denoted as $CORE_{d,\epsilon,\mu}(v)$. The formal definition is as follows:*

$$CORE_{d,\epsilon,\mu}(v) \iff |\{u|u \in N_d(v) \& \sigma(u,v) \geq \epsilon\}| \geq \mu \tag{3}$$

The definitions of direct structural reachability and structural reachability are given below. Note that in order to ensure the connectivity of clusters, only the one-hop neighbors of the core vertex can be directly structurally reachable from the core vertex.

Definition 5 (Direct Structural Reachability). *If the distance between a core vertex u and a one-hop neighbor v is no greater than d and they are similar, we say the vertex v is directly structurally reachable from the core vertex u, which is formally defined as follows:*

$$\begin{aligned} DirREACH_{d,\epsilon,\mu}(u,v) \quad &\iff CORE_{d,\epsilon,\mu}(u) \wedge \\ & v \in N_d(u) \cap N(u) \wedge \sigma(u,v) \geq \epsilon. \end{aligned} \tag{4}$$

Definition 6 (Structural Reachability). *If there is a directly structurally reachable path from a core vertex u to a vertex v, the vertex v is structurally reachable from the core vertex u, which is formally defined as follows:*

$$\begin{aligned} REACH_{d,\epsilon,\mu}(u,v) \iff & \quad \exists v_1,...,v_n \in V : v_1 = u \wedge v_n = v \\ \wedge \forall i \in 1,...,n-1 : & \quad DirREACH_{d,\epsilon,\mu}(v_i, v_{i+1}). \end{aligned} \tag{5}$$

Definition 7 (Cluster). *Given a similarity threshold $\epsilon \in R$, a threshold of similar neighbors $\mu \in N$ and a distance threshold $d \in R$, cluster $C \subseteq V$ satisfies:*

- *(Maximality) If a core vertex $u \in C$, all vertices structural reachable from the vertex u belong to the cluster C.*
- *(Connectivity) For any vertices $v_1, v_2 \in C$, there exists a core vertex $u \in C$ such that both v_1 and v_2 are structural reachable from u.*

A cluster contains core vertices and vertices structural reachable from the core vertices. The vertices that do not belong to any cluster are called hubs and outliers, which are defined as follows:

Definition 8 (Hubs and Outliers). *For a vertex $u \in G$ that does not belong to any cluster, if $N(u)$ contains vertices in two or more clusters, u is a hub, otherwise, u is an outlier.*

Problem definition. Given a weighted undirected graph $G = (V,E,w)$, a similarity threshold $\epsilon \in R$, a threshold of similar neighbors $\mu \in N$ and a distance threshold $d \in R$, the distance-based structural graph clustering problem aims to calculate the set $\mathbb{C}$ containing all clusters, hubs and outliers in G.

Note that SCAN is a special case of distance-based SCAN. When all weights in G are the same as d, distance-based SCAN and SCAN obtain the same clustering result. Similarly, when all weights are the same and d is a multiple of the weight, distance-based SCAN can perform graph clustering based on multi-hop neighbors.

2.2 Existing Solutions

2.2.1 Structural Graph Clustering. To the best of our knowledge, this is the first work to study the distance-based SCAN problem. The most relevant research work is algorithms for structural graph clustering. There are two state-of-the-art algorithms for solving structural graph clustering, namely GS*-Index [29] and pSCAN [6].

GS*-Index. GS*-Index pre-computes similarities and builds indices to quickly answer clustering queries. GS*-index takes advantage of the fact that the structural similarity on the unweighted graph is not related to μ, and caches the similarity of all edges and arranges them reasonably to deal with clustering queries. However, this method is not suitable for distance-based structural graph clustering. For different values of d, the structural similarity of vertices will change, which makes it difficult to reuse the structural similarity. If the structural similarity of all neighbors in d-neighborhoods is cached for each vertex in the graph for different values of d, the space complexity is unacceptable.

The pSCAN method. The pSCAN method designs pruning techniques to speed up the computation of clusters. It maintains the upper and lower bounds of the number of similar neighbors for each vertex. When the lower bound of vertex v is no lower than μ, v becomes a core vertex. When the upper bound of vertex v is lower than μ, v cannot be a core vertex. It detects core vertices in descending order of the upper bound of similar neighbors of vertices. If it can be determined whether the current vertex is a core vertex, the calculation of the structural similarity will be stopped. It reduces the number of edges required to calculate the structural similarity and improves efficiency. But it also can not be directly applied to distance-based structural graph clustering. Because both the similarity calculation and the initialization of the upper bound of similar neighbors need to calculate d-neighborhood. The worst-case time complexity of online computation of d-neighborhood is $O(n^2 \log(n))$. The worst-case space complexity for caching the d-neighborhood is $O(n^2)$. These time and space complexities are unacceptable for large networks.

WeightedSCAN. Some existing work, e.g. WeightedSCAN [16] generalizes structural graph clustering to weighted graphs. They use weighted Cosine similarity to calculate structural similarity. However, the clustering result of WeightedSCAN is not as good as the distance-based SCAN we defined, which is verified by the experimental results in Section 5.

2.2.2 Jaccard Coefficient Calculation. Bottom-k sketches [9] are often used to efficiently calculate the Jaccard coefficient. Bottom-k sketches use a single hash function to calculate the k elements with the smallest hash value in the given sets. The formal definition of bottom-k sketches is shown in Definition 9.

Definition 9 (Bottom-k sketches [9]). *Given a graph G, a hash function $hash(\cdot)$ and a distance threshold d, the bottom-k sketch $B(u)$ of a vertex $u \in G$ is a list of entries $\{hash(v_1), hash(v_2), ..., hash(v_k)\}$, where v_i is the vertex with i-th smallest hash value in $N_d(u)$.*

For distance-based structural graph clustering, different values of d correspond to different d-neighborhoods. The time complexity of creating sketches for all possible distances is very high. Edith Cohen et al. [10] propose the all-distance bottom-k sketches (ADS) to solve this problem. ADS can encode all neighborhoods with small space complexity, from which bottom-k sketches of d-neighborhood can be quickly retrieved. The definition of *all-distances bottom-k sketches* is given in Definition 10. We will make full use of ADS to improve our algorithm efficiency and optimize the construction and extraction of sketches.

Definition 10 (All-distances bottom-k sketches). *Given a graph G and a hash function hash, the all distances bottom-k sketches $\mathbb{B}(u)$ of a vertex $v \in G$ is a list of entries $\{(hash(v), \delta(u,v))\}$ where $hash(v)$ is smaller than the k-th lowest hash value amongst vertices within distance at most $\delta(u,v)$ from u.*

3 STRUCTURAL SIMILARITY CALCULATION

Structural similarity calculation is the main bottleneck of DistanceSCAN. In order to quickly calculate similarities between vertices pairs, we simultaneously construct histograms and ADS. Histograms can be used as the basis for pruning similarity calculation. If the similarity cannot be determined according to histograms, bottom-k sketches corresponding to any distance threshold can be extracted from ADS to quickly calculate the similarity. In this section, we first explain the construction process of the sketches, then introduce how to quickly calculate the similarity based on bottom-k sketches and histograms.

3.1 Construct Sketches

We first introduce algorithms for constructing histograms and ADS. Note that ADS cannot be directly used to calculate similarity. So we also provide an algorithm to extract bottom-k sketches from ADS. The upper and lower bounds of the similarity can be derived from the size of the d-neighborhood, which is described in detail in Section 3.2. In order to preserve the size information of the d-neighborhoods of each vertex, we construct histograms. The formal definition of the histogram index is as follows.

Definition 11 (Histogram). *Given the width of bin $\omega \in R$, for any vertex u in the graph, the histogram $H[u]$ contains the vector $(H[u][i]), i = 1, 2, 3...,$ where $H[u][i]$ is the size of the $(i \times \omega)$-neighborhood of the vertex u.*

The d-neighborhoods are the basis for the construction of histograms and ADS. We traverse the d-neighborhood of each vertex $v \in V$ based on the Dijkstra algorithm. During the traversal, we update the histograms and ADS by $\delta(u,v)$ and the hash value of the scanned vertex u. While building sketches, we can traverse all the vertices reachable by vertex v in the graph. However, if the value of d is too large, it will cause the d-neighborhoods sizes of most vertices in the graph to be $O(n)$, and most of the vertices are similar. Obviously, the clustering results obtained in this case have no practical significance. Therefore, we set the parameter d_{max}. For each vertex $v \in V$ in the graph, we only compute the vertices whose distance to vertex v does not exceed d_{max}.

Algorithm 1 shows the pseudo-code to construct sketches. It traverses the d-neighborhood of each vertex in the graph by the Dijkstra algorithm with pruning conditions. Firstly, it initializes histograms and ADS (Line 1). Then, it constructs sketches for each vertex (Lines 2-14). The algorithm for constructing ADS in [8] traverses vertices in ascending hash value order. However, this method cannot construct histograms that can greatly improve efficiency. Algorithm 1 utilizes the process of the Dijkstra algorithm to traverse the d-neighborhood in ascending order of distance (Lines 3-8,11-14), which can build histograms and ADS (Lines 9-10) at the same time. In particular, Algorithm 1 initializes the distances from the current vertex u to each vertex in the graph (Lines 3-5). Next, it pushes u into the empty priority queue PQ (Line 6). For each vertex v in PQ, it updates sketches (Lines 9-10) and pushes vertices whose distances to the vertex u are less than d_{max} into PQ (Lines 11-14).

```
Algorithm 1: ConstructSkeches
   Input: Graph $G(V,E,w)$ and parameters $d_{max}, k, \omega$
   Output: Histograms $H$, ADS $\mathbb{B}$
 1 $H \leftarrow \emptyset, \mathbb{B} \leftarrow \emptyset$;
 2 foreach $u \in V$ do
 3     foreach $v \in V$ do
 4         $\delta(u,v) \leftarrow \infty$;
 5     $\delta(u,u) \leftarrow 0$;
 6     $PQ.push(u)$;
 7     while $PQ \neq \emptyset$ do
 8         $v \leftarrow PQ.pop()$;
 9         Increase the frequencies of the bins corresponding
             to $\delta(u,v)$ in $H[u]$ by 1;
10         UpdateADS($\mathbb{B}, u, v, \delta(u,v)$) (Algo. 2);
11         foreach $v_{nei} \in N(v)$ do
12             if $\delta(u,v) + w(v,v_{nei}) < \min\{d_{max}, \delta(u,v_{nei})\}$
                 then
13                 $\delta(u,v_{nei}) \leftarrow \delta(u,v) + w(v,v_{nei})$;
14                 $PQ.push(v_{nei})$;
15 return $H, \mathbb{B}$;
```

The time complexity of traversing the d-neighborhood of each vertex is $O(n\Delta\log(\Delta))$, where Δ represents the maximum size of the d_{max}-neighborhood. Line 9 updates histograms. The number of bins h in the histogram of each vertex can be obtained from the ratio of d_{max} to the width of bins ω. Since h is much smaller than the number of vertices in the graph, that is, $h \ll n$, the space complexity of histograms is $O(n)$ and the time complexity of updating histograms is $O(1)$. Line 10 updates ADS. ADS can be stored using persistent balanced binary trees [10]. For a persistent balanced binary tree, the time and space complexity of creating a new tree is $O(1)$, and the time and space complexities of insertion and deletion are $O(\log(k))$. If the number of vertices in the current binary tree is less than k when a vertex v arrives, a new binary tree is directly created, and the vertex v is inserted into the new tree. Suppose the size of the current binary tree is equal to k and the hash value of vertex v is less than the maximum value in the current binary tree. In that case, we create a new tree, delete the vertex with the largest hash value in the new tree and insert vertex v into the new tree. Given a vertex $v \in V$, we use the maximum distance from the vertex in a bottom-k sketch to v to represent the distance from the bottom-k sketch to v. Bottom-k sketches in $\mathbb{B}(v)$ are arranged in increasing order of distance from bottom-k sketches to the vertex v. Given a distance threshold d, bottom-k sketches can be retrieved in $O(k)$ time after obtaining the indices by binary search from ADS.

Although bottom-k sketches can be quickly retrieved using persistent balanced binary trees, persistent balanced binary trees still result in high space complexity. In order to reduce the space complexity, we directly save the hash values and distances of vertices in bottom-k sketches as a list and use the max-heap to insert, modify,

Algorithm 2: UpdateADS

```
Input: ADS $\mathbb{B}$, vertices $u, v$ and distance $\delta(u, v)$
Output: ADS $\mathbb{B}$
1  Let $PQ$ be the priority queue saving the bottom-$k$ sketch of
     vertex $u$;
2  Let $hash(v)$ be the hash value of vertex $v$;
3  if $|PQ| < k$ then
4      $PQ$.push($hash(v)$);
5      $\mathbb{B}(u)$.push(($hash(v)$, $\delta(u, v)$));
6  else if $PQ.top() > hash(v)$ then
7      $PQ$.pop();
8      $PQ$.push($hash(v)$);
9      $\mathbb{B}(u)$.push(($hash(v)$, $\delta(u, v)$));
10 return $\mathbb{B}$;
```

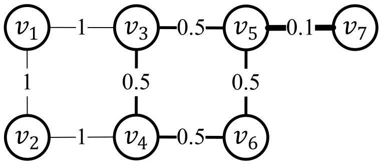


Figure 1: An example graph G_1

and return bottom-k sketches. Algorithm 2 shows the pseudo-code for updating ADS. PQ is the priority queue that holds the current bottom-k hash values (Line 1). Since the vertices arrive in the order of increasing distance, when the size of PQ is less than k, Lines 3-5 directly push the information of a vertex v into $\mathbb{B}(u)$ and PQ. When the size of PQ is equal to k and the hash value of v is less than the max value in PQ, Lines 6-9 first delete the vertex with the largest hash value in PQ and then push the information of vertex v to PQ and $\mathbb{B}(u)$. The space complexity of each update of ADS is $O(1)$ and the time complexity is $O(\log(k))$.

To compute the similarity, we need to get bottom-k sketches from ADS based on a given distance threshold. Getting the bottom-k sketches from ADS also utilizes the max-heap. Algorithm 3 shows the pseudo-code. Line 1 initializes an empty priority queue. For items in $\mathbb{B}(u)$ whose distance is not greater than d, Lines 3-9 take out the k items with the smallest hash value. Line 10 updates the bottom-k sketch of the vertex u. The time complexity of computing bottom-k sketches for all vertices in the graph is $O(n\Delta \log(k))$, where Δ represents the max size of d-neighborhood.

Example 3.1. Given an example graph G_1 as shown in Figure 1, Algorithm 1 constructs ADS and histograms for each vertex in G_1. Let $k = 4, d_{max} = 2, \omega = 0.5$ and assume that the hash value of each vertex is the same as the vertex id, namely $hash(v_i) = i$. Now, consider the sketches construction of the vertex v_7. Algorithm 1 traverses the d_{max}-neighborhood of v_7 according to the distance from small to large, the specific order is: $v_7, v_5, v_3, v_6, v_4, v_1$. Since $\delta(v_2, v_7) = 2.1 > d_{max}$, v_2 will not be traversed. When Algorithm 1 scans to the first vertex v_7, $H[v_7]$ changes from $\{0, 0, 0, 0\}$ to $\{1, 1, 1, 1\}$ as $\delta(v_7, v_7) = 0$ and $\mathbb{B}$ changes from $\emptyset$ to $\{v_7\}$. When Algorithm 1 scans to the last vertex v_1, $H[v_7]$ changes from $\{2, 4, 5, 5\}$ to $\{2, 4, 5, 6\}$ as $\delta(v_7, v_1) = 1.6$ and $\mathbb{B}$ changes from $\{v_7, v_5, v_3, v_6, v_4\}$ to $\{v_7, v_5, v_3, v_6, v_4, v_1\}$ by Algorithm 2 as $hash(v_1)$ is the smallest. Algorithm 3 compute the bottom-k sketches for each vertex. When $d = 1$, the bottom-k sketches of v_7 is $B[v_7] = \{v_7, v_5, v_3, v_6\}$.

3.2 Estimating Structural Similarity

Similar to SCAN, similarity calculation is the main procedure of the distance-based SCAN and the bottleneck of the algorithm efficiency. We speed up the computation of similarity by histograms and bottom-k sketches computed by Algorithm 1 and Algorithm 3 in Section 3.1, respectively. We first give some theoretical analysis of calculating similarity using histograms and bottom-k sketches, and then show the algorithm for similarity calculation. Lemma 3.2 and Lemma 3.3 give upper and lower bounds on the similarity.

Lemma 3.2. *Given a vertex $u \in V$ and a vertex $v \in N_d(u)$, the similarity of vertices u and v satisfies:*

$$\sigma(u, v) \le \min\left\{\frac{|N_d(u)|}{|N_d(v)|}, \frac{|N_d(v)|}{|N_d(u)|}\right\} \tag{6}$$

Proof. If the amount of vertices in $N_d(u)$ is not less than $N_d(v)$, when $N_d(v) \subseteq N_d(u)$, $|N_d(u) \cap N_d(v)|$ takes the maximum value $|N_d(v)|$ and $|N_d(u) \cup N_d(v)|$ takes the minimum value $N_d(u)$. Therefore $\sigma(u, v)$ is not greater than $|N_d(v)|/|N_d(u)|$. If the amount of vertices in $N_d(v)$ is not less than $N_d(u)$, when $N_d(u) \subseteq N_d(v)$, we have $\sigma(u, v) \le |N_d(u)|/|N_d(v)|$. □

Lemma 3.3. *Given a vertex $u \in V$ and a vertex $v \in N_d(u)$, the similarity of vertices u and v satisfies:*

$$\sigma(u, v) \ge \max \left\{\frac{|N_{d-\delta(u,v)}(u)|}{|N_d(u)|+|N_d(v)|-|N_{d-\delta(u,v)}(u)|}, \frac{|N_{d-\delta(u,v)}(v)|}{|N_d(u)|+|N_d(v)|-|N_{d-\delta(u,v)}(v)|}\right\} \tag{7}$$

Proof. As $v \in N_d(u)$, we have $\delta(u, v) \le d$. According to the triangle inequality of distance, for vertex $x \in N_{d-\delta(u,v)}(u)$, the distance from vertex x to v satisfies $\delta(x, v) \le \delta(x, u) + \delta(u, v) \le (d - \delta(u, v)) + \delta(u, v) = d$. It can be seen that the vertices in $N_{d-\delta(u,v)}(u)$ must also be in the d-neighborhood of v, that is, $N_{d-\delta(u,v)}(u) \subseteq N_d(v)$. Therefore, $N_{d-\delta(u,v)}(u) \subseteq N_d(u) \cap N_d(v)$. So, $|N_d(u) \cap N_d(v)| \ge |N_{d-\delta(u,v)}(u)|$, and $|N_d(u) \cup N_d(v)| \le |N_d(u)| + |N_d(v)| - |N_{d-\delta(u,v)}(u)|$. We have $\sigma(u, v) \ge |N_{d-\delta(u,v)}(u)|/(|N_d(u)| + |N_d(v)| - |N_{d-\delta(u,v)}(u)|)$. Similarly, $N_{d-\delta(u,v)}(v) \subseteq N_d(u) \cap N_d(v)$, and $\sigma(u, v) \ge |N_{d-\delta(u,v)}(v)|/(|N_d(u)| + |N_d(v)| - |N_{d-\delta(u,v)}(v)|)$. □

If the upper bound of $\sigma(u, v)$ is lower than the threshold ϵ, we can directly conclude that vertex u and vertex v are not similar. If the lower bound of $\sigma(u, v)$ is higher than the threshold ϵ, we can directly conclude that vertex u and vertex v are similar. The histograms save the number of neighbors at different distances for each vertex. We can query the upper and lower bounds of $N_d(u), N_d(v), N_{d-\delta(u,v)}(v), N_{d-\delta(u,v)}(u)$ in $O(1)$ time and then judge whether vertices u, v are similar. The size of d-neighborhood is monotonically non-decreasing with the growth of d. Combining with Lemma 3.3, it can be known that if the distance $\delta(u, v)$ between vertices u and v is small, the similarity may be high. In order to prune similarity computations based on the histogram as much as possible, DistanceSCAN at each step preferentially computes the similarity of pairs of vertices that are close together.

For vertices $u \in V, v \in N_d(u)$, when the upper and lower bounds of the similarity provided by histograms cannot directly determine

Algorithm 3: GetBottom-kSketches

```
Input: ADS $\mathbb{B}$, parameters $d, k$
Output: Bottom-$k$ sketches $B$
foreach $u \in V$ do
    $PQ \leftarrow \emptyset$;
    foreach $(hash(v), \delta(u, v)) \in \mathbb{B}(u)$ do
        if $\delta(u, v) \le d$ then
            if $|PQ| < k$ then
                $PQ$.push($v$);
            else if $PQ$.top()>$hash(v)$ then
                $PQ$.pop();
                $PQ$.push($v$);
    $B(u) \leftarrow$ hash values in $PQ$;
return $B$;
```

Algorithm 4: ComputeSimilarityBySketches

```
Input: Histograms $H$, bottom-$k$ sketches $B$, vertices $u, v$
Output: Vertices $u, v$ are similar or not
$N_d^{lb}(u), N_d^{ub}(u), N_{d-\delta(u,v)}^{lb}(u) \leftarrow H(u)$;
$N_d^{lb}(v), N_d^{ub}(v), N_{d-\delta(u,v)}^{lb}(v) \leftarrow H(u)$;
$\sigma^{lb}(u, v), \sigma^{ub}(u, v) \leftarrow$compute lower bound and upper
  bound of $\sigma(u, v)$(Lemma 3.2, 3.3);
if $\sigma^{lb}(u, v) \ge \epsilon$ then
    return True;
else if $\sigma^{ub}(u, v) < \epsilon$ then
    return False;
$\hat{\sigma}(u, v) \leftarrow$ compute similarity by $B(u), B(v)$;
if $\hat{\sigma}(u, v) \ge \epsilon$ then
    return True;
else
    return False;
```

whether they are similar or not, we get a more accurate estimate of the similarity using bottom-k sketches. Let $\hat{\sigma}(u, v) = |B(B(u) \cup B(v)) \cap B(u) \cap B(v)|/|B(B(u) \cup B(v))|$ represent the approximate similarity computed by bottom-k sketches, Theorem 3.4 shows that the $\hat{\sigma}(u, v)$ is an unbiased estimate of the similarity $\sigma(u, v)$. The complexity of estimating similarity using bottom-k sketches is $O(k)$. If k is set too small, it will bring a large error bound, and if k is too large, it will increase the computational cost. Given the relative error ρ and the failure probability p_f, according to Theorem 3.5, we can get the required size of k. Although Theorem 3.5 computes a high value of k, experiments in Section 5 show that very accurate clustering results can be obtained even when k is small.

Theorem 3.4. *Given sets A_1, A_2 and a hash function, let $B(A)$ denote the bottom-k sketch of set A. When the number of elements in A is less than k, $B(A)$ contains all the elements in A. An unbiased estimate of the Jaccard coefficient of sets A_1, A_2 can be obtained based on bottom-k sketches, namely:*

$$\mathbb{E}\left[\frac{|A_1 \cap A_2|}{|A_1 \cup A_2|}\right] = \mathbb{E}\left[\frac{|B(B(A_1) \cup B(A_2)) \cap B(A_1) \cap B(A_2)|}{|B(B(A_1) \cup B(A_2))|}\right] \quad (8)$$

Proof. Let $hash(\cdot)$ denote the hash function that generate bottom-k sketches and $hash(A)$ denote the set of hash values of elements in set A. Clearly,

$$\begin{aligned} B(B(A_1) \cup B(A_2)) &= MIN_k(hash(A_1) \cup hash(A_2)) \\ &= MIN_k(hash(A_1 \cup A_2)) \end{aligned} \quad (9)$$

where $MIN_k(\cdot)$ represents the k elements with the smallest value in the set. Let α be the smallest value in $hash(A_1 \cup A_2)$ and $hash^{-1}(\alpha)$ be the element in $A_1 \cup A_2$ whose hash value is α, then

$$\begin{aligned} \Pr[\alpha \in B(A_1) \cap B(A_2)] &= \Pr[hash^{-1}(\alpha) \in A_1 \cap A_2] \\ &= \tfrac{|A_1 \cap A_2|}{|A_1 \cup A_2|} \end{aligned} \quad (10)$$

Repeat this process for each element in $B(B(A_1) \cup B(A_2))$, and the theorem is proved. □

Theorem 3.5. *Given an relative error parameter $\rho > 0$, if $k \ge \frac{\log(1/p_f)(1/2+2\sigma_{min}/3)}{\rho^2\sigma_{min}^2}$,*

$$|\hat{\sigma}(u, v) - \sigma(u, v)| \le \rho \cdot \sigma(u, v) \quad (11)$$

holds with at least $1 - p_f$ probability for any $\sigma(u, v) > \sigma_{min}$, where $\hat{\sigma}(u, v)$ is the approximate similarity calculated by bottom-k sketches.

Proof. We use the following theorem to prove Theorem 3.5.

Theorem 3.6 (Bernstein's inequality). *[3] Let $X = (x_1, ..., x_n)$ be a finite population of n points and $X_1, ..., X_k$ be a random sample drawn without replacement from X. Let $a = \min_{1 \le i \le n} x_i$, $b = \max_{1 \le i \le n} x_i$ and $\mu = \frac{1}{n}\sum_{i=1}^n x_i$. The variance of X is $D[X] = \frac{1}{n}\sum_{i=1}^n (x_i - \mu)^2$. Then for all $\epsilon > 0$,*

$$\Pr\left[\frac{1}{k}\sum_{i=1}^k X_i - \mu \ge \epsilon\right] \le \exp\left(-\frac{k\epsilon^2}{2D[X] + (2/3)(b-a)\epsilon}\right). \quad (12)$$

Given two vertices u, v, the bottom-k sketch $B(B(u) \cup B(V))$ contains k random samples drawn without replacement from $N_d(u) \cup N_d(v)$. If $|N_d(u) \cup N_d(v)| \le k$, $\hat{\sigma}(u, v) = \sigma(u, v)$ and Theorem 3.5 holds. Otherwise, the error bound of $\hat{\sigma}(u, v)$ is proved as follows. Let x_i be a variable that takes value 1 if $N_d(u) \cap N_d(v)$ contains the i-th element in $N_d(u) \cup N_d(v)$, and value 0 otherwise. We have $a = \min_{1 \le i \le n} x_i = 0$, $b = \max_{1 \le i \le n} x_i = 1$, $\frac{1}{k}\sum_{i=1}^k x_i = \hat{\sigma}(u, v)$ and $\mu = \frac{1}{n}\sum_{i=1}^n x_i = \sigma(u, v)$. The variance of X is $D[X] = \sigma(u, v) - \sigma^2(u, v) \le 1/4$. Let $\epsilon = \rho \cdot \sigma_{min}$, we have

$$\begin{aligned} \Pr\left[\tfrac{1}{k}\sum_{i=1}^k X_i - \mu \ge \epsilon\right] &= \Pr[\hat{\sigma}(u, v) - \sigma(u, v) \ge \rho \cdot \sigma_{min}] \\ &\le \exp\left(-\tfrac{k \cdot \rho^2 \cdot \sigma_{min}^2}{2D[X] + (2/3)(b-a) \cdot \rho\sigma_{min}}\right) \\ &\le \exp\left(-\tfrac{k \cdot \rho^2 \cdot \sigma_{min}^2}{1/2 + (2/3) \cdot \rho\sigma_{min}}\right) \end{aligned}$$

Since $k \ge \frac{\log(1/p_f)(1/2+2\sigma_{min}/3)}{\rho^2\sigma_{min}^2}$, we can conclude that

$$\Pr[|\hat{\sigma}(u, v) - \sigma(u, v)| > \rho\sigma_{min}] \le 1 - p_f.$$

□

Algorithm 4 shows the pseudo-code for computing similarity using histograms and bottom-k sketches. Since histograms can determine whether the vertex pairs are similar within $O(1)$ time, the algorithm preferentially uses histograms for pruning. Lines 1-2 extract the upper and lower bounds of the d-neighborhood and the lower bound of $(d - \delta(u, v))$-neighborhood of the vertices u, v from histograms. Line 3 calculates the upper and lower bounds of the similarity according to Lemma 3.2 and Lemma 3.3. If Line 4 or Line 6 can directly judge whether vertices u, v are similar or not, Algorithm 4 returns the result. Otherwise, Lines 8-12 calculate the approximate similarity $\hat{\sigma}(u, v)$ using bottom-k sketches, and return the similarity result. It should be noted that our main purpose is to judge whether $\hat{\sigma}(u, v)$ is less than ϵ, not to calculate $\hat{\sigma}(u, v)$. Therefore, in the process of calculating $\hat{\sigma}(u, v)$, when it can be determined whether $\hat{\sigma}(u, v)$ is less than ϵ, the calculation can be terminated in advance and return the similarity result.

As a summary, we list the relationships between the algorithms in this section. Algorithm 1 builds histograms and ADS, which will be used in similarity computation (Algorithm 4). Algorithm 2 shows the key steps of updating ADS in the sketch construction (Algorithm 1). Given a distance threshold, Algorithm 3 gets the bottom-k sketches from ADS, which is to be used in similarity computation (Algorithm 4). Algorithm 4 calculates the similarity for vertex pairs according to the histogram constructed by Algorithm 1 and the bottom-k sketches obtained by Algorithm 3.

Example 3.7. Figure 1 shows an example graph G_1. Let $k = 4, d = 1, \omega = 0.5$ and $hash(v_i) = i$. Consider the vertex v_5. We can get sketches from Algorithm 1. $\mathbb{B}[v_5] = \{v_5, v_7, v_3, v_6, v_4, v_1, v_2\}$ and $H[v_5] = \{4, 5, 6, 7\}$. So the bottom-k sketch of v_5 computed by Algorithm 3 is $B[v_5] = \{v_3, v_4, v_5, v_6\}$. The bottom-$k$ sketch of v_7 is $B[v_7] = \{v_7, v_5, v_3, v_6\}$ which is shown in Example 3.1. Since $\delta(v_5, v_7) = 0.1$, we have $N_1(v_5) = H[v_5][2] = 5, N_1(v_7) = H[v_7][2] = 4, N_{0.9}(v_5) > H[v_5][1] = 4, N_{0.9}(v_7) > H[v_7][1] = 3$. $\sigma(v_5, v_7) \geq N_{0.9}(v_5)/(N_1(v_5) + N_1(v_7) - N_{0.9}(v_5)) = 0.8$. If $\epsilon \leq 0.8$, Lines 4-5 of Algorithm 4 directly determines that v_5 and v_7 are similar, without traversing bottom-k sketches of vertices v_5, v_7.

4 OUR APPROACH

In this section, we design an efficient approximation algorithm DistanceSCAN to solve the distance-based structural graph clustering problem. Structural similarity calculation is the performance bottleneck of the algorithm, so we use histograms to support pruning techniques and use bottom-k sketches for fast computation. The main idea of DistanceSCAN is to first identify core vertices, then check the connectivity between the core vertices, and finally classify the non-core vertices.

4.1 Algorithm Overview

As stated in Section 2.2, the main difficulty of the distance-based structural graph clustering problem is that the d-neighborhood is unknown. In large networks, it takes a long time to perform online computations, and the space complexity is too high to save d-neighbors directly. In order to quickly obtain d-neighborhood information to calculate similarity with low time and space complexities, we construct histograms and ADS for each vertex in the graph. Histograms store the size of the d-neighborhood of each vertex for different values of d. Given vertices u, v, we can quickly calculate the upper and lower bounds of the similarity based on histograms $H(u), H(v)$ and the distance between two vertices $\delta(u, v)$. When the similarity computation cannot be pruned according to the upper and lower bounds, we calculate the approximate similarity $\hat{\sigma}(u, v)$ based on bottom-k sketches retrieved from ADS. According to Section 3, it is easy to judge whether vertices with close distance are similar or not. So the main design principle of our DistanceSCAN algorithm is to preferentially calculate the similarity for vertices pairs with closer distance.

```
Algorithm 5: DistanceSCAN
   Input: A weighted graph $G(V, E, w)$ and parameters $d, \epsilon, \mu$
   Output: The clustering result of $G$
1  $H, \mathbb{B} \leftarrow$ InitSketches($G, d$) ; // Invoke Algo. 1 to
     construct sketches or load pre-stored sketches
2  $B \leftarrow$ GetBottom-$k$Sketches($\mathbb{B}, d$) (Algo. 3);
3  $S \leftarrow \emptyset$;
4  foreach $v \in V$ do
5      $lb(v) \leftarrow 0$;
6      $ub(v) \leftarrow N_d^{ub}(v)$;
7      Initialize the vertex type of $v$;
8  foreach unclassified $u \in V$ do
9      CheckCore($H, B, S, u$) (Algo. 6);
10 $\mathbb{C}$, hubs and outliers $\leftarrow$ GetClusters($H, B, S$);
11 return $\mathbb{C}$, hubs and outliers;
```

Algorithm 5 shows the pseudo-code for DistanceSCAN. Line 1 initializes histograms H and ADS $\mathbb{B}$. If the sketches are precomputed, histograms and ADS are read directly from memory or disk. Otherwise, histograms and ADS are constructed using Algorithm 1. Bottom-k sketches B are retrieved from ADS using Algorithm 3 (Line 2). In order to reduce repeated calculations and prevent the space complexity from being too high, if the vertex u is in the bottom-k sketch of vertex v, DistanceSCAN uses a hash table S initialized in Line 3 to cache $\sigma(u, v)$. For each vertex v, the upper bound of similar neighbors is the upper bound of $N_d(v)$, and the lower bound is 0 (Lines 4-6). If the size of $B(v)$ is less than k, $ub(v) = |B(v)|$. Otherwise, we get the upper bound of $N_d(v)$ by the histogram of vertex v. Line 7 initializes v as unclassified if $ub(v) \geq \lambda$, otherwise non-core. Lines 8-9 check whether each unclassified vertex of the graph is a core vertex. In this step, the algorithm calculates the similarity of the neighbors in the d-neighborhood according to the distance from small to large and updates the upper and lower bounds of similar neighbors and the vertex type. Line 10 first detects the connectivity between core vertices, then identifies non-core vertices in clusters, hubs, and outliers. Line 11 returns the clustering result containing clusters, hubs, and outliers.

4.2 Core Vertices Detection

Detection of core vertices in Line 9 of Algorithm 5 is a key step of DistanceSCAN. During the detection of core vertices, there are three types of vertices, namely core vertices, non-core vertices, and unclassified vertices. All vertices are initially unclassified. When calculating the similarities of edges, the upper and lower bounds of similar neighbors are updated at the same time. For a vertex $v \in V$,

Algorithm 6: CheckCore

```
Input: Histograms $H$, bottom-$k$ sketches $B$, similarity
       results $S$, vertex $u$
Output: the vertex type of $u$
1  Sort the vertices in $B(u)$ in ascending order of distance;
2  foreach $v \in B(u)$ do
3      CheckSimilarity($H, B, S, u, v$) (Algo. 7);
4      if $u$ is not unclassified then
5          return the vertex type of $u$;
6  Perform Dijkstra from $u$ to traverse $N_d(u)$ while $u$ is
     unclassified;
7  foreach $v \in N_d(u)$ in ascending order of $\delta(u, v)$ do
8      CheckSimilarity($H, B, S, u, v$) (Algo. 7);
9      if $u$ is core vertex or non-core vertex then
10         return the vertex type of $u$;
```

if the upper bound of similar neighbors is lower than μ, classify v as a non-core vertex. If the lower bound of similar neighbors is no less than μ, the vertex v is classified as a core vertex. In this step, we traverse each vertex in the graph to check if it is a core vertex. Detecting core vertices requires computing similarities with neighbors in the d-neighborhood, which is unknown in advance. We can compute the d-neighborhood using the Dijkstra algorithm, but it is relatively time-consuming. Note that some neighbors in the d-neighborhood are saved in bottom-k sketches, so we can first calculate the similarity with the neighbors in bottom-k sketches. Since it is easy to judge in $O(1)$ time whether neighbors with small distances are similar using a histogram, we sort the neighbors in bottom-k sketches according to their distances and calculate the neighbors with closer distances first. If the vertex type still cannot be determined, use the Dijkstra algorithm to traverse the neighbors in the d-neighborhood and calculate the similarity with the neighbors in ascending order of distance.

Algorithm 6 shows the pseudo-code for core vertex detection. To avoid traversing the d-neighborhood, Lines 1-5 calculate the similarity between the vertex u and its neighbors in the bottom-k sketch $B[u]$. If the type of the vertex still cannot be determined, Line 6 uses the Dijkstra algorithm to traverse the d-neighborhood. Lines 7-10 calculate the similarity of neighbors of vertex u in ascending order of distances. It terminates the traversal of the d-neighborhood of the current vertex after the vertex has been classified. Algorithm 6 is called $O(n)$ times by Algorithm 5 in total. Next, we analyze its total time complexity. Algorithm 6 is mainly divided into two parts. One part is the traversal of the d-neighborhood. For efficiency, Line 1 does a rough ordering by dividing neighbors in $B(u)$ into buckets, which can be done in $O(nk)$. Line 6 simply performs Dijkstra to traverse the d-neighborhood of vertex u. The time complexity is $O(n\Delta \log(\Delta))$, where Δ is the maximum size of d-neighborhood. The other part is the calculation of similarity. Combining with the amortized time complexity of Algorithm 7 in Section 4.3, we can know that its time complexity is $O((nk + m)\Delta)$. Since nk is much larger than m, the total time complexity of Algorithm 6 is $O(n\Delta \cdot (k + \log(\Delta))$. Note that since the algorithm will stop immediately after the vertex type can be determined by the upper and lower bounds of similar neighbors, Algorithm 6 is efficient in practice.

Algorithm 7: CheckSimilarity

```
Input: Histograms $H$, bottom-$k$ sketches $B$, similarity
       results $S$, vertices $u, v$
1  if $\sigma(u, v) \in S$ then
2      return ;
3  $S(u, v) \leftarrow$ ComputeSimilarityBySketches($H, B, u, v$) (Algo. 4);
4  Update the vertex types for $u, v$;
5  if $S(u, v)$ then
6      if $u, v$ are core vertices then
7          Merge the clusters containing $u$ and $v$;
8      if $u$ or $v$ becomes a core vertex then
9          Merge all clusters containing
             $\{x | x \in N_d(u) \cap N(u) \& \sigma(u, x) \geq \epsilon\} \cup \{u\}$ or
             $\{x | x \in N_d(v) \cap N(v) \& \sigma(v, x) \geq \epsilon\} \cup \{v\}$;
```

4.3 Structural Similarity Checking

Detecting core vertices requires calculating the similarity with neighbors. To reduce redundant computations, we not only use the hash table S to cache the similarity but also design pruning rules for similarity computation. The similarity between non-core vertices does not need to be calculated, and the similarity between core vertices in the same cluster also does not need to be calculated. To apply the above rules, we need to maintain the vertex types and clusters for each vertex when calculating the similarity.

Algorithm 7 shows the pseudo-code for checking similarity. Line 1 detects whether the similarity needs to be calculated. If there exists the similarity information of vertices u and v in the hash table S, it will be returned directly. Line 3 calculates the similarity of vertices u, v according to sketches and stores the result in S. If u, v are not structural similar, Line 4 classifies vertices whose upper bound of similar neighbors is lower than μ as non-core vertices. If the vertices u, v are similar and both are core vertices, Lines 6-7 merge the clusters containing u and v. The vertex u(or v) becomes a core vertex if the number of similar neighbors of u(or v) is exactly μ. If the vertex u(or v) is similar to some core vertices in the intersection of its d-neighbors and one-hop neighbors, Lines 8-9 merge the clusters where these core vertices and u(or v) are located. The time complexity of Line 3 is $O(k)$. The merging of clusters is implemented by a disjoint-set data structure, and the time complexity is $O(1)$. Lines 8-9 traverse the one-hop neighborhood of u or v. The amortized time complexity of Algorithm 7 is $O(k + m/n)$.

4.4 Clustering Result Calculation

In the core vertex detection step, core vertices in the same cluster may not be connected since we terminate the traversal of the d-neighborhood after classifying vertices. It is necessary to calculate the similarities of edges between core vertices of different clusters to ensure accurate clustering results. In this step, we also follow the principle of prioritizing the calculation of the similarity of vertex pairs that are close to each other. We first traverse all edges in the graph and extract edges that connect to core vertices in different clusters and have not calculated the similarities. The similarities of these edges are calculated in order of distances from the small to large. If two vertices of an edge are similar, merge the clusters where the two vertices are located. The time complexity of this part is $O(mk)$. Finally, we traverse non-core vertices in the graph. If there is a similar core vertex in the one-hop neighbor of the non-core vertex $v \in V$, assign v to the cluster where the core vertex is located. For a vertex $v \in V$ that is not in clusters, if the one-hop neighborhood of v has an intersection with two or more clusters, the vertex v will be classified as a hub, otherwise, it will be classified as an outlier. Since the similarities of all edges need to be calculated in the worst case, the time complexity of this part is also $O(mk)$.

4.5 Analysis of DistanceSCAN

DistanceSCAN needs to obtain bottom-k sketches from ADS using the distance threshold d by Algorithm 3. The time complexity is $O(n\Delta \log(k))$ and the space complexity is $O(nk)$. Since the number of bins of histograms is much smaller than the number n of vertices, the space complexity of histograms is $O(n)$. The hash table S only stores the similarities of vertices to their neighbors in bottom-k sketches, so the space complexity is $O(nk)$. Combining the time complexity and space complexity of each step of the algorithm, it can be seen that the time and space complexities of DistanceSCAN in the worst case are $O(n\Delta \cdot (\log(\Delta) + k) + mk)$ and $O(nk + m)$ respectively. ADS does not need to be stored in the memory. Hence, if they are in memory, the space complexity increases by $O(n\Delta)$. Otherwise the time complexity of reading bottom-k sketches from ADS increases by $O(n\Delta)$. Note that as our algorithm sets multiple pruning strategies, a large number of redundant calculations are reduced. So the efficiency of DistanceSCAN is very high in practice. In addition, except for some high-degree vertices, the d-neighborhood of vertices is often much smaller than k, so the space complexity of bottom-k sketches is also much lower than $O(nk)$.

To summarize the relationships between algorithms, the main algorithm DistanceSCAN (Algorithm 5) first uses Algorithm 1 and Algorithm 3 to get histograms and bottom-k sketches. Then, it invokes Algorithm 6 to classify vertices as core vertices or non-core vertices. Classifying vertices requires a lot of computation of the similarity between vertices. An important step of Algorithm 6 is to invoke Algorithm 7 to derive the similarity and integrate pruning rules to speed up the process of classifying core/non-core vertices.

Example 4.1. Given an example graph G_1 as shown in Figure 1, Algorithm 5 computes the clustering results. Let $k = 4, d = 1, \omega = 0.5$, $\lambda = 4$, $\epsilon = 0.5$ and $hash(v_i) = i$. As the upper bound of similar neighbors of v_1, v_2 is smaller than λ, these vertices are classified non-core vertices by Line 7 in Algorithm 5. Algorithm 5 calls Algorithms 6 to check whether v_3, v_4, v_5, v_6, v_7 are core vertices. For v_3, Lines 1-5 of Algorithms 6 call Algorithm 7 to calculate the similarity with neighbors in the order of $B[v_3] = \{v_3, v_4, v_5, v_1\}$. Since only v_1 and v_3 are not similar, we have $lb(v_3) = 3, ub(v_3) = 5$. Lines 6-9 of Algorithm 6 traverse the d-neighborhood of v_3. Since $v_7 \in N_d(v_3)$ and v_3 are similar, $lb(v_3) = 4 \geq \lambda$. v_3 is classified as a core vertex by algorithm 7. Finally, the clustering result returned by Algorithm 5 is that G_1 has a cluster consisting of five core vertices $\{v_3, v_4, v_5, v_6, v_7\}$ and v_1, v_2 are outliers.

5 EXPERIMENTS

In this section, we verify the effectiveness and efficiency of DistanceSCAN by comparing it with multiple baseline methods through extensive experiments. The algorithms are all implemented in C++. All experiments are conducted on a Linux machine with Intel(R) Xeon(R) Gold 6248 CPU @ 2.50GHz and 252GB main memory. This section has four parts. First, we introduce the datasets, parameters, and baseline methods used in the experiments. Second, we verify that our defined Distance-based SCAN can achieve high clustering quality, and the approximation algorithm DistanceSCAN has high accuracy. Third, we verify the efficiency of DistanceSCAN and the effect of sketches on efficiency. Finally, we compare our ADS construction algorithm with the ADS construction method based on persistent binary search trees.

5.1 Experimental Settings

This section introduces the basic settings of experiments, mainly including the introduction of the datasets, the value range of the parameters and their default values, and the baseline methods.

Datasets: Table 1 lists the statistical information such as the number of vertices, the number of edges, the average degree, the average clustering coefficient, and the type of weights of datasets used in experiments, where R indicates that the dataset has real weights and S indicates that the dataset has synthetic weights. TC-T24 is short for Topic-coauthor-T24, which is from ArnetMiner [26]. UK-2002 is from the Laboratory of Web Algorithmics [4]. Other datasets are from the Stanford Network Analysis Platform [20]. A directed graph can be converted to an undirected graph by adding edges in the other direction. TC-T24 contains researchers and their coauthorships in the fields of database/XML data. The weight on an edge is the number of co-published articles by the two researchers represented by the vertices. A large number of co-published articles mean that their research fields are similar and the distance between the vertices is close. We design a function $f = \frac{1}{\log(x)+1}$ to convert the number of articles into distance, where x is the number of co-published articles and f is the distance. For other datasets, we use the Jaccard similarity of adjacent vertices as synthesis weights. A high Jaccard similarity indicates a small distance between two vertices and vice versa. We also design a function $f = 1 - 0.9x$ to convert the synthetic weights into distances. The value range of the distances obtained by synthesis weights is $[0.1, 1)$. The value range does not start at 0 for two benefits. On the one hand, the size of the d-neighborhoods will not grow sharply with the increase of parameter d. On the other hand, it helps limit the maximum number of hops for neighbors.

Parameters: Referring to the previous work [23], our parameters are set as follows, in which the default values are shown in bold:

- $\mu = \underline{\mathbf{5}}$, $d_{\max} = \underline{\mathbf{1.4}}$
- $\epsilon \in \{0.1, 0.15, \underline{\mathbf{0.2}}, 0.25, 0.3\}$, $d \in \{0.6, 0.8, \underline{\mathbf{1}}, 1.2, 1.4\}$
- $k \in \{2^{13}, 2^{14}, 2^{15}, \underline{\mathbf{2^{16}}}\}$, $\omega \in \{0.05, \underline{\mathbf{0.01}}, 0.005, 0.001\}$

Method: DistanceSCAN is compared with the following methods:

- **SCAN [31]:** Structural graph clustering algorithm.
- **WeightedSCAN [16]:** the SCAN algorithm based on the weighted Cosine similarity.

Table 1: Network Statistics

Datasets	$\lvert V\rvert$	$\lvert E\rvert$	$\bar{d}$	c	t
TC-T24	1127	6690	11.87	0.3297	R
Facebook	4,039	88,234	43.69	0.6055	S
Brightkite	58,228	214,078	7.35	0.1723	S
Gowalla	196,591	950,327	9.66	0.2367	S
DBLP	317,080	1,049,866	6.62	0.6324	S
Flickr	105,938	2,316,948	43.74	0.0891	S
YouTube	1,134,890	2,987,624	5.27	0.0808	S
Pokec	1,632,803	30,622,564	37.51	0.1094	S
LiveJournal	3,997,962	34,681,189	17.35	0.2843	S
Orkut	3,072,441	117,185,083	76.28	0.1666	S
UK-2002	18,520,486	298,113,762	32.19	0.6891	S

- **EXACT:** Exact algorithm for distance-based structural graph clustering. EXACT first calculates the d-neighborhood of each vertex, performs clustering according to the process of SCAN, and uses the d-neighborhood to compute structural similarities.
- **pSCAN [6]:** pSCAN for distance-based structural graph clustering after calculating the d-neighborhood of each vertex.
- **DistanceSCAN-WH, DistanceSCAN-WB**: DistanceSCAN without using histogram to prune the similarity computation, and DistanceSCAN without using bottom-k sketches when traversing d-neighborhood in core vertex detection, respectively.

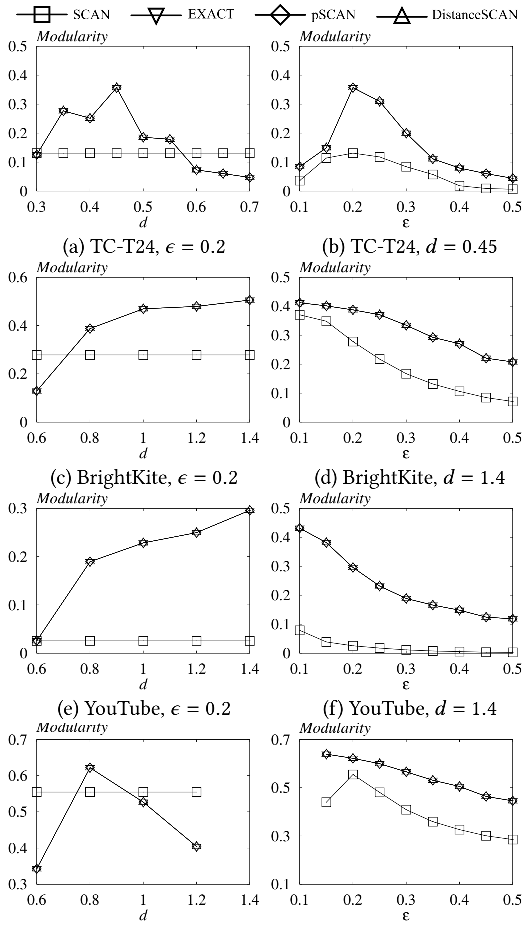
5.2 Cluster Quality Verification

The experiments in this section are mainly about the clustering quality of DistanceSCAN with two different similarity measures: Jaccard similarity and Cosine similarity, and the accuracy relative to EXACT. Following previous work [23, 27], we use modularity [21] to measure the quality of clusterings. One of the characteristics of modularity is that it does not need to be compared with the ground truth, so it has a wide range of applications. The modularity of a weighted graph is the proportion of weights of edges that fall within clusters minus the expected proportion of weights that fall within clusters in a random graph with the same degree distribution. The formal definition of modularity is given by Equation (13),

$$Q = \frac{1}{2m} \sum_{u,v \in V} \left(w(u,v) - \frac{k_u k_v}{2m} \right) I(u,v) \tag{13}$$

where k_u is the sum of weights of edges of vertex u, that is, $k_u = \sum_{v \in N(u)} w(u,v)$. m is the sum of the weights of all edges in the graph, that is, $m = \frac{1}{2} \sum_{u \in V} k_u$. $I(u,v)$ is an indicator function. When the vertices u and v are in the same cluster, $I(u,v)$ is 1. Otherwise, $I(u,v)$ is 0. The value range of modularity is $[0,1]$. Modularity close to 0 means that the weight distribution of the edges in clusters is similar to the weight distribution of randomly connected edges. A large modularity indicates a good clustering quality.

Modularity using Jaccard similarity. Figure 2 shows the clustering results of SCAN, pSCAN, EXACT, and DistanceSCAN when clustering based on Jaccard similarity. Because the WeightedSCAN algorithm is not suitable for Jaccard similarity, we did not compare it with EXACT. In order to get high-quality clustering results, we can fix ϵ, μ and find the distance threshold that can get high modularity results. Figures 2(a), (c), (e), (g) show the results of modularity varying distance threshold d when ϵ takes the default value of 0.2.



(a) TC-T24, ϵ = 0.2 (b) TC-T24, d = 0.45

(c) BrightKite, ϵ = 0.2 (d) BrightKite, d = 1.4

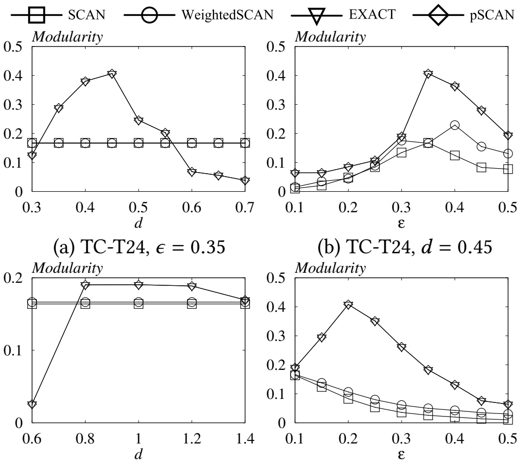
(e) YouTube, ϵ = 0.2 (f) YouTube, d = 1.4

(g) LiveJournal, ϵ = 0.2 (h) LiveJournal, d = 0.8

Figure 2: Evaluation of modularity using Jaccard similarity

Notice that the distance thresholds of TC-T24 and other datasets are set differently. To explain, the distances of TC-T24 are the real weights while the distances of other datasets are adopted according to the Jaccard similarity of adjacent vertices. Due to different physical meanings, the ranges of the weights are different. SCAN is not affected by distance. So the modularity of SCAN remains the same. When the distance threshold d is too small, there are too few neighbors around the vertex, and it is difficult to meet the requirement of the threshold μ, so there are fewer vertices in the cluster and the clustering quality is poor. When d is too large, it is easy for vertices to be similar, so that most of the vertices are in clusters, and the clustering quality is also poor. Hence, when d increases, the modularity tends to increase first and then decrease. Figures 2 (b),(d),(f),(h) show that when d takes an appropriate value, the modularity of EXACT, pSCAN, and DistanceSCAN are identical and are better than that of SCAN in most cases.

Modularity using Cosine similarity. Figure 3 reports the modularity of SCAN, WeightedSCAN, and EXACT when clustering based on Cosine similarity. For the interest of space, we only show the results for two graphs: TC-T24 and YouTube. Figures 3 (a),(c) find the most suitable distance threshold d under the similarity threshold ϵ where SCAN performs the best. The distance thresholds that



(a) TC-T24, $\epsilon = 0.35$ (b) TC-T24, $d = 0.45$

(c) YouTube, $\epsilon = 0.1$ (d) YouTube, $d = 1$

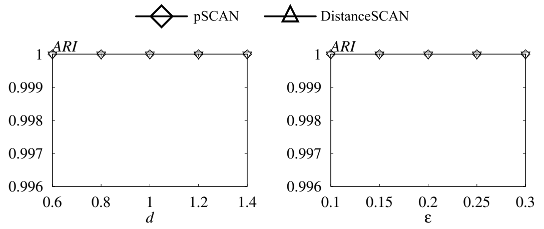
Figure 3: Evaluation of modularity using Cosine similarity

EXACT performs best on TC-T24 and YouTube are 0.45 and 1, respectively. In Figure 3 (b),(d), with the change of ϵ, EXACT achieves the highest modularity in most cases, while SCAN performs the worst. Combining results in the previous set of experiments, we find that the distance-based structural graph clustering shows identical best performance (in terms of modularity). Regardless of using Jaccard or Cosine similarity, the clustering results of distance-based SCAN are far better than WeightedSCAN. This is because WeightedSCAN considers the weight information of one-hop neighbors, but the utilization of the weight information is still not as sufficient as our proposed idea of distance-based structural graph clustering.

ARI. Since DistanceSCAN is an approximate algorithm, we further compare the clustering result derived by the approximation algorithm against the EXACT algorithm using the *Adjusted Rand Index* (ARI) [17]. ARI measures how consistent two non-overlapping clustering results are. Distance-based structural graph clustering results in overlapping clusters, but core vertices of clusters do not overlap. To be able to measure using ARI, we follow previous work [23] by setting vertices that belong to multiple clusters to the cluster where the core vertex in their one-hop neighborhood with the highest similarity is located. Given two clustering results, ARI counts how vertices are assigned to clusters in both clustering results. The formal definition of ARI is shown in Equation (14).

$$ARI = \frac{\sum_{i,j} C_{n_{ij}}^2 - \left[\sum_i C_{n_{i.}}^2 \sum_j C_{n_{.j}}^2\right] / C_n^2}{\frac{1}{2}\left[\sum_i C_{n_{i.}}^2 + \sum_j C_{n_{.j}}^2\right] - \left[\sum_i C_{n_{i.}}^2 \sum_j C_{n_{.j}}^2\right] / C_n^2} \tag{14}$$

where n_{ij} represents the number of vertices in the i-th cluster of the first clustering result and in the j-th cluster of the second clustering result, $n_{i.} = \sum_j n_{ij}$ represents the number of vertices of the i-th cluster of the first clustering result, $n_{.j} = \sum_i n_{ij}$ represents the number of vertices of the j-th cluster of the second clustering result. The value range of ARI is $[0, 1]$, where the closer it is to 1, the more consistent the two clustering results are. When k takes the default value of 2^{16}, for all datasets, the ARI of the clustering results obtained by DistanceSCAN under the default parameters and the clustering results of EXACT are all 1. Figure 4 shows the ARI of DistanceSCAN with varying d and ϵ on the dataset UK-2002. The ground truth clustering results are derived by pSCAN. We add pSCAN in Figure 4 as a reference. We can see that the ARI of DistanceSCAN is always 1, which means that the clustering results of DistanceSCAN are consistent with that of the ground truth. Although the default value of k is smaller than the value given by Theorem 3.5, the value of k is already larger than the size of d-neighborhood of most vertices in the datasets, which contributes to the accurate clustering results of DistanceSCAN.



(a) $\epsilon = 0.2$, vary d (b) $d = 1$,vary ϵ

Figure 4: The ARI of UK-2002

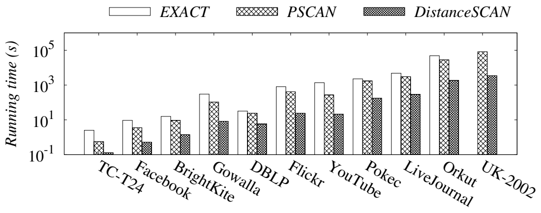


Figure 5: Running time on all datasets

5.3 Efficiency Evaluation

In this set of experiments, we verify the efficiency of DistanceSCAN. On one hand, the running time under various parameters is compared with EXACT and pSCAN to verify the efficiency of DistanceSCAN. On the other hand, we compare DistanceSCAN with DistanceSCAN-WH and DistanceSCAN-WB to verify the effectiveness of the optimization techniques.

Efficiency evaluation on all the datasets. Figure 5 shows the running time of EXACT, pSCAN, and DistanceSCAN on all datasets with default parameters, where results with running time longer than two days are omitted. Note that the y-axis is in log-scale. It can be seen that DistanceSCAN has good scalability and the shortest running time on all datasets. EXACT has the worst efficiency because it does not take any pruning techniques. The efficiency of the three algorithms is not only related to the number of edges in the graph, but also to the average degree of vertices. Gowalla with a higher average degree takes longer running time than DBLP. It is worth noting that on UK-2002 and Orkut, the running time of DistanceSCAN is only about 1/20 of pSCAN.

Efficiency with varying *d*. Figure 6 shows the efficiency of algorithms as the distance threshold d varies. Note that the y-axes of Figures 6 (b), (c), and (d) are in log-scale. The most obvious conclusion in Figure 6 is that DistanceSCAN outperforms pSCAN and EXACT in all cases. In addition, it can be seen that as the distance threshold d increases, the running time of pSCAN and DistanceSCAN both increase sharply. This is expected as the larger the d, the larger the size of the d-neighborhood, and the higher the time cost

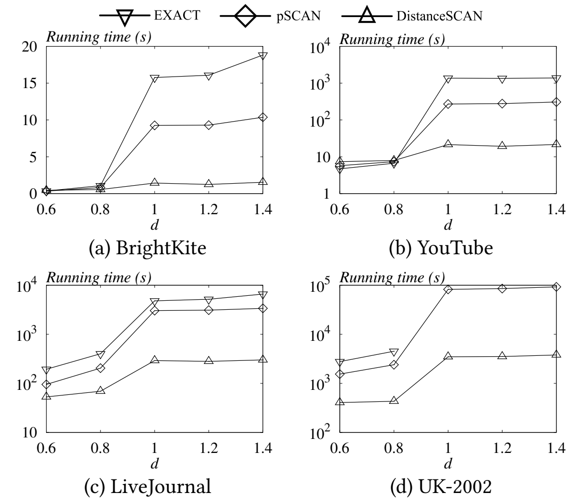


Figure 6: Evaluation of running time (vary d)

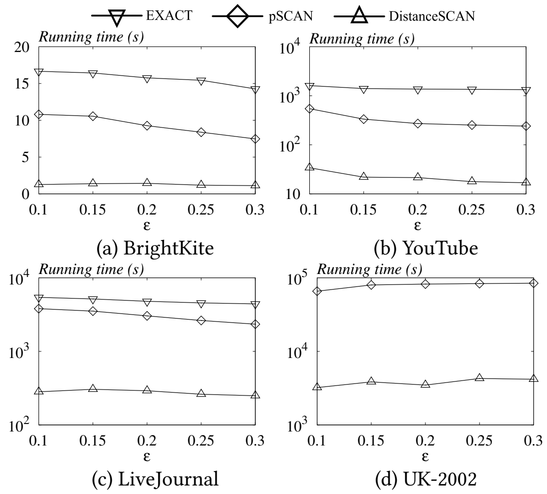


Figure 7: Evaluation of running time (vary ϵ)

of calculating the similarity. After d is greater than 1, the running time does not increase significantly with increasing distance threshold. The reason is that the distance distribution is not uniform and the proportion of edges with small distances is small. When d is greater than 0.8, the running time of EXACT is more than two days, so Figure 6 (d) omits the relevant experimental results of EXACT.

Efficiency with varying ϵ. Figure 7 shows the running time as the similarity threshold ϵ varies. The results of EXACT on UK-2002 are omitted because the running time of EXACT is more than two days. DistanceSCAN still achieves the best efficiency in all settings. As ϵ increases, the running time of DistanceSCAN decreases slightly. A large ϵ will lead to large-scale clusters, and the similarities between core vertices in the same cluster do not need to be calculated, which improves the efficiency of algorithms.

Efficiency with varying k. Figure 8 shows the effect of the sizes of bottom-k sketches on efficiency. DistanceSCAN still runs much faster than pSCAN and EXACT in all parameter settings. The change in the size of bottom-k sketches has little effect on DistanceSCAN. The main reason is that the size of the d-neighborhood

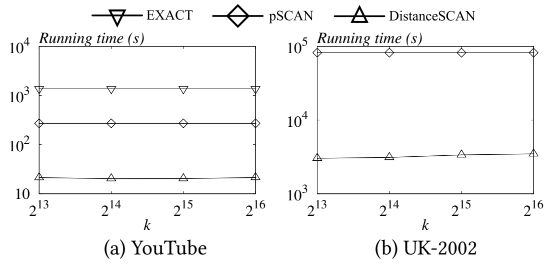


Figure 8: Evaluation of running time (vary k)

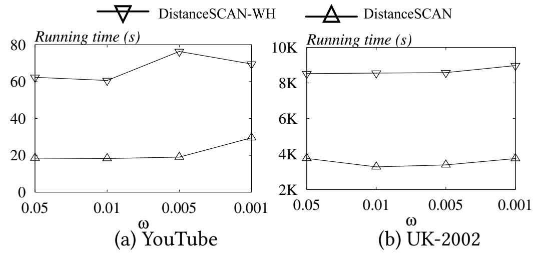


Figure 9: Evaluation of histogram (vary ω)

of most vertices in the graph is much smaller than the value of k. Therefore, in order to ensure the accuracy of DistanceSCAN, we set 2^{16} as the default size of bottom-k sketches.

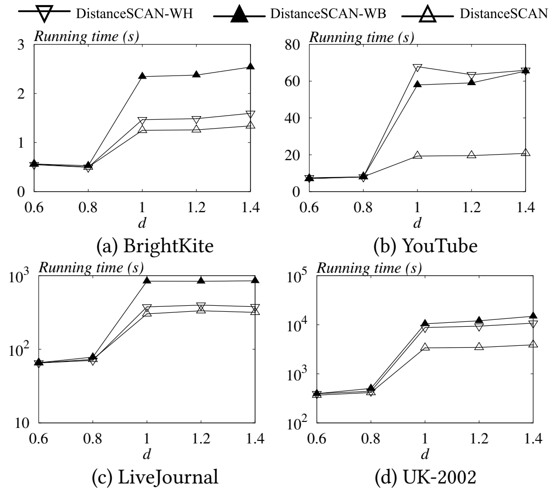
Efficiency with varying the width of the bins in histograms. Figure 9 shows the effect of the width of the bins on efficiency. As the width of the bins decreases, the efficiency increases slightly and then decreases. The main reason is that the number of similarity computations that can be pruned by histograms is limited, and the small width will increase the space complexity of histograms. Therefore, in order to balance the space complexity and efficiency, the default width of the bins is 0.01.

Evaluation of optimization techniques with varying d. When computing similarities, DistanceSCAN first uses histograms for pruning. In the step of checking core vertices, to reduce traversal of d-neighborhoods, DistanceSCAN first computes the similarity with neighbors in bottom-k sketches. To verify the validity of these optimizations, Figure 10 reports the running time of DistanceSCAN-WH, DistanceSCAN-WB and DistanceSCAN varing d. It can be seen that using histograms pruning and traversing the neighbors in bottom-k sketches greatly improves the efficiency. As d increases, DistanceSCAN has more obvious advantages in terms of efficiency.

Evaluation of optimization techniques with varying ϵ. Figure 11 presents the running time of DistanceSCAN-WH, DistanceSCAN and DistanceSCAN-WB varying ϵ. Again, DistanceSCAN runs faster than the other two for all values of ϵ. As ϵ increases, the advantage of DistanceSCAN in efficiency becomes more apparent.

5.4 Performance of Sketch construction

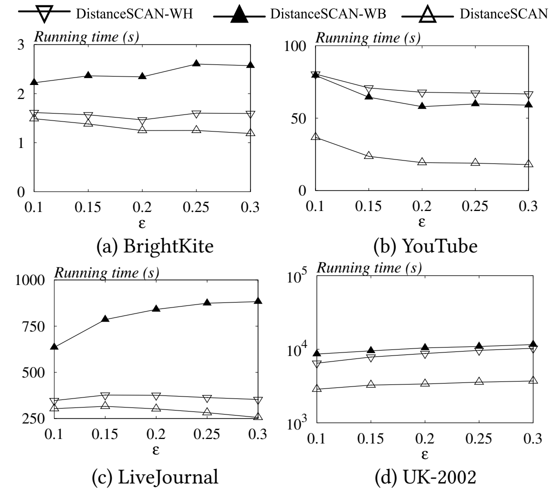
In this set of experiments, we compare the ADS construction method proposed in Section 3.1 with the ADS construction method based on persistent search trees [10], mainly including experiments on sketch size and construction time.



(a) BrightKite (b) YouTube

(c) LiveJournal (d) UK-2002

Figure 10: Evaluation of optimization technique (vary d)



(a) BrightKite (b) YouTube

(c) LiveJournal (d) UK-2002

Figure 11: Evaluation of optimization technique (vary ϵ)

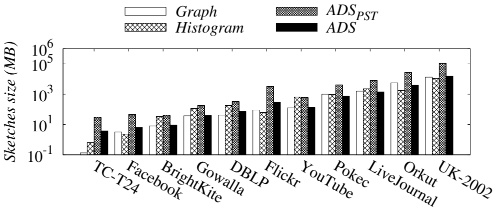
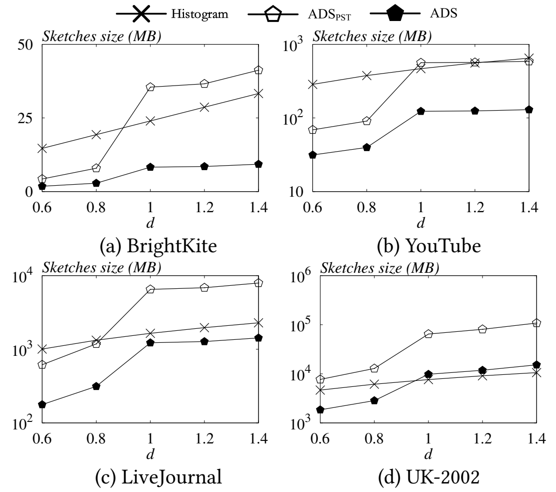


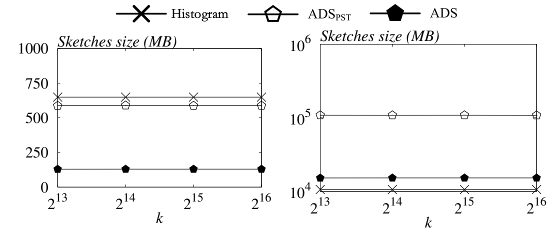
Figure 12: The sizes of sketches on all datasets



(a) BrightKite (b) YouTube

(c) LiveJournal (d) UK-2002

Figure 13: The sizes of sketches (vary d)



(a) YouTube (b) UK-2002

Figure 14: The sizes of sketches (vary k)

Sketch size on all datasets. Figure 12 shows the sizes of sketches constructed on all datasets. We also add the original size of the graph for comparison. ADS_{PST} saves all-distances bottom-k sketches by persistent balanced binary trees. The width of the bins in histograms is set to 0.01. It can be seen that the histogram size is generally comparable to the size of the original graph because the number of histogram bins is close to the average degree of most graphs. ADS stands for saving all-distances bottom-k sketches by our approach. It is clear that the space complexity of ADS is lower than that of ADS_{PST} on all datasets. The reason is that ADS reduces a lot of duplication in sketches.

Sketch size with varying d. Figure 13 shows the sizes of different sketches as the distance threshold increases. Note that the y-axes of Figures 13 (b), (c) and (d) are in log-scale. It can be seen that the sizes of two kinds of all-distances bottom-k sketches grow rapidly with the distance as the size of the d-neighborhood increases rapidly with the growth of d, which also causes frequent updates of all-distances bottom-k sketches. ADS still has lower space usage than ADS_{PST} under all parameters. The size of the histogram grows linearly with the distance threshold d since the width of the bins is fixed, and the number of bins grows linearly with the growth of d.

Sketch size with varying k. Figure 14 shows the sizes of different sketches as the sample number k of bottom-k sketches varies. The construction of histograms is independent of k and thus remains the same. The space usage of ADS is about an order of magnitude lower than that of ADS_{PST}. ADS and ADS_{PST} do not change significantly as k varies. On one hand, the d-neighborhoods of most vertices in the graph are much smaller than k. On the other hand, for some high-degree vertices, the increase of k makes the size of bottom-k sketches larger, and also reduces the update frequency of sketches, and keeps the sizes of sketches stable. Hence, using 2^{16} as the default value of k can not only ensure the accuracy of the algorithm but also not cause the size of sketches to be too large.

Construction time of sketches on all datasets. Figure 15 shows the running time of sketch construction on all datasets. Note that the y-axis is in log-scale. ADS is more efficient than ADS_{PST}. ADS can save more than 20% running time compared to ADS_{PST} on

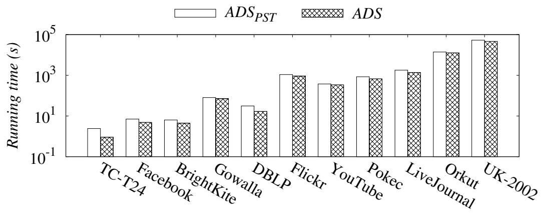
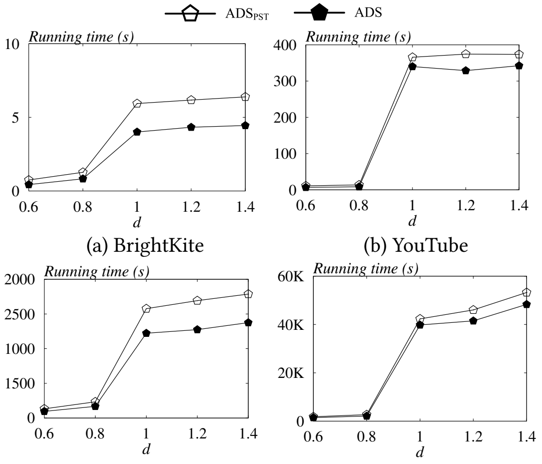


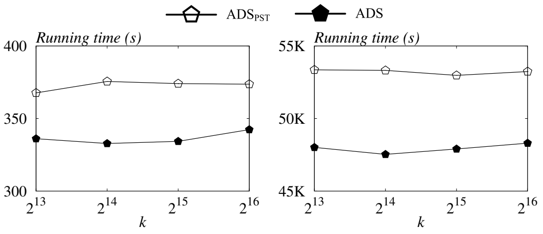
Figure 15: The construction time of sketches on all datasets



(a) BrightKite (b) YouTube

(c) LiveJournal (d) UK-2002

Figure 16: The construction time of sketches (vary d)



(a) YouTube (b) UK-2002

Figure 17: The construction time of sketches (vary k)

most datasets. While the time complexities of updating sketches for *ADS* and ADS_{PST} are both $O(\log(k))$, ADS_{PST} also needs to replicate hashes of $O(\log(k))$ vertices.

Construction time with varying d and k. Figure 16 shows the time of sketch construction as the distance threshold d increases. It can be seen that *ADS* still achieves better efficiency under all parameters. While both running time grows rapidly with d, *ADS* grows at a slower rate than ADS_{PST}. This is due to the fact that as d grows, the number of neighbors in d-neighborhoods increases, and ADS_{PST} needs to replicate the hash values of more vertices. Figure 17 shows the effect of k on the running time of sketch construction. It can be seen that consistent with the results of Figure 14, the running time of sketch construction does not change significantly as k varies.

6 RELATED WORK

After Xu et al. [31] propose structural graph clustering and give the original SCAN algorithm, a lot of research work on the optimization of SCAN emerged. One category of studies is devoted to designing parameter-free algorithms. Such work focuses on user-friendly design to avoid tuning input parameters but still find good clustering results. SCOT [5] draws on the idea of OPTICS [1], and outputs a vertex sequence for a given threshold μ. Vertices in the same cluster are adjacent to each other in the sequence. For example, SHRINK [15] uses modularity as the optimization objective and gradually merges vertex pairs with the highest similarity to obtain the hierarchical clustering result. DPSCAN [30] maps vertices into a two-dimensional space for clustering based on the distribution of the similarity between vertices and their neighbors.

Another category of studies is devoted to improving the efficiency of SCAN. SCAN++ [24] reduces redundant computation of similarity by determining the types of vertices and their two-hop neighbors. pSCAN [6] is currently the most advanced method based on pruning. It can speed up the calculation by adjusting the vertex calculation order and pruning in time after determining the type of vertices. GS*-Index [29] can quickly answer clustering queries by precomputing the similarity of all edges and saving them in order. SCAN-XP [25], ppSCAN [7] and GBBSIndexSCAN [27] accelerate the computation of clustering results by parallel computing. DynStrClu [23] focuses on structural graph clustering on dynamic graphs. Most of the existing work studies structural graph clustering by one-hop neighbors, which may result in poor quality clusters.

7 CONCLUSION

In this paper, the distance-based structural graph clustering (SCAN) problem is defined and studied. An efficient algorithm DistanceSCAN is proposed to derive the clustering result for distance-based SCAN. The experimental results show that the distance-based SCAN can obtain clustering results with higher modularity, and DistanceSCAN is far more efficient than baseline methods. For future work, we will work on the design of dynamic algorithms that can quickly return clustering results after edge additions, edge deletions, and weight changes occur. In addition, inspired by algorithms such as SHRINK, designing a parameter-free algorithm also has a wide range of application scenarios.

REFERENCES

[1] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. 1999. OPTICS: Ordering Points To Identify the Clustering Structure. In *SIGMOD*. 49–60.
[2] Kevin Aydin, MohammadHossein Bateni, and Vahab S. Mirrokni. 2016. Distributed Balanced Partitioning via Linear Embedding. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining, San Francisco, CA, USA, February 22-25, 2016*, Paul N. Bennett, Vanja Josifovski, Jennifer Neville, and Filip Radlinski (Eds.). ACM, 387–396.
[3] Rémi Bardenet and Odalric-Ambrym Maillard. 2015. Concentration inequalities for sampling without replacement. *Bernoulli* 21, 3 (2015), 1361–1385.
[4] Paolo Boldi and Sebastiano Vigna. 2004. The webgraph framework I: compression techniques. In *Proceedings of the 13th international conference on World Wide Web, WWW 2004, New York, NY, USA, May 17-20, 2004*, Stuart I. Feldman, Mike Uretsky, Marc Najork, and Craig E. Wills (Eds.). ACM, 595–602.
[5] Dustin Bortner and Jiawei Han. 2010. Progressive clustering of networks using Structure-Connected Order of Traversal. In *ICDE*. 653–656.
[6] Lijun Chang, Wei Li, Xuemin Lin, Lu Qin, and Wenjie Zhang. 2016. pSCAN: Fast and exact structural graph clustering. In *32nd IEEE International Conference on Data Engineering, ICDE 2016, Helsinki, Finland, May 16-20, 2016*. IEEE Computer Society, 253–264.
[7] Yulin Che, Shixuan Sun, and Qiong Luo. 2018. Parallelizing Pruning-based Graph Structural Clustering. In *ICPP*. 77:1–77:10.
[8] Edith Cohen. 2015. All-Distances Sketches, Revisited: HIP Estimators for Massive Graphs Analysis. *IEEE Trans. Knowl. Data Eng.* 27, 9 (2015), 2320–2334. https://doi.org/10.1109/TKDE.2015.2411606

[9] Edith Cohen, Mayur Datar, Shinji Fujiwara, Aristides Gionis, Piotr Indyk, Rajeev Motwani, Jeffrey D. Ullman, and Cheng Yang. 2001. Finding Interesting Associations without Support Pruning. *IEEE Trans. Knowl. Data Eng.* 13, 1 (2001), 64–78.

[10] Edith Cohen and Haim Kaplan. 2007. Summarizing data using bottom-k sketches. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing, PODC 2007, Portland, Oregon, USA, August 12-15, 2007*, Indranil Gupta and Roger Wattenhofer (Eds.). ACM, 225–234.

[11] Chris H. Q. Ding, Xiaofeng He, Hongyuan Zha, Ming Gu, and Horst D. Simon. 2001. A Min-max Cut Algorithm for Graph Partitioning and Data Clustering. In *Proceedings of the 2001 IEEE International Conference on Data Mining, 29 November - 2 December 2001, San Jose, California, USA*, Nick Cercone, Tsau Young Lin, and Xindong Wu (Eds.). IEEE Computer Society, 107–114.

[12] Pedro M. Domingos and Matthew Richardson. 2001. Mining the network value of customers. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, San Francisco, CA, USA, August 26-29, 2001*, Doheon Lee, Mario Schkolnick, Foster J. Provost, and Ramakrishnan Srikant (Eds.). ACM, 57–66.

[13] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. Density-based spatial clustering of applications with noise. In *Int. Conf. Knowledge Discovery and Data Mining*, Vol. 240. 6.

[14] Michelle Girvan and Mark EJ Newman. 2002. Community structure in social and biological networks. *Proceedings of the national academy of sciences* 99, 12 (2002), 7821–7826.

[15] Jianbin Huang, Heli Sun, Jiawei Han, Hongbo Deng, Yizhou Sun, and Yaguang Liu. 2010. SHRINK: a structural clustering algorithm for detecting hierarchical communities in networks. In *CIKM*. 219–228.

[16] Jianbin Huang, Heli Sun, Qinbao Song, Hongbo Deng, and Jiawei Han. 2013. Revealing Density-Based Clustering Structure from the Core-Connected Tree of a Network. *IEEE Trans. Knowl. Data Eng.* 25, 8 (2013), 1876–1889.

[17] Lawrence Hubert and Phipps Arabie. 1985. Comparing partitions. *Journal of classification* 2, 1 (1985), 193–218.

[18] Paul Jaccard. 1912. The distribution of the flora in the alpine zone. 1. *New phytologist* 11, 2 (1912), 37–50.

[19] U Kang and Christos Faloutsos. 2011. Beyond 'Caveman Communities': Hubs and Spokes for Graph Compression and Mining. In *11th IEEE International Conference on Data Mining, ICDM 2011, Vancouver, BC, Canada, December 11-14, 2011*, Diane J. Cook, Jian Pei, Wei Wang, Osmar R. Zaïane, and Xindong Wu (Eds.). IEEE Computer Society, 300–309.

[20] Jure Leskovec and Rok Sosic. 2016. SNAP: A General-Purpose Network Analysis and Graph-Mining Library. *ACM Trans. Intell. Syst. Technol.* 8, 1 (2016), 1:1–1:20.

[21] Mark EJ Newman. 2004. Analysis of weighted networks. *Physical review E* 70, 5 (2004), 056131.

[22] Mark EJ Newman. 2004. Fast algorithm for detecting community structure in networks. *Physical review E* 69, 6 (2004), 066133.

[23] Boyu Ruan, Junhao Gan, Hao Wu, and Anthony Wirth. 2021. Dynamic Structural Clustering on Graphs. In *SIGMOD*. 1491–1503.

[24] Hiroaki Shiokawa, Yasuhiro Fujiwara, and Makoto Onizuka. 2015. SCAN++: Efficient Algorithm for Finding Clusters, Hubs and Outliers on Large-scale Graphs. *Proc. VLDB Endow.* 8, 11 (2015), 1178–1189.

[25] Tomokatsu Takahashi, Hiroaki Shiokawa, and Hiroyuki Kitagawa. 2017. SCAN-XP: Parallel Structural Graph Clustering Algorithm on Intel Xeon Phi Coprocessors. In *NDA@SIGMOD*. 6:1–6:7.

[26] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. 2008. Arnet-Miner: extraction and mining of academic social networks. In *SIGKDD*. 990–998.

[27] Tom Tseng, Laxman Dhulipala, and Julian Shun. 2021. Parallel Index-Based Structural Graph Clustering and Its Approximation. In *SIGMOD*. 1851–1864.

[28] Yang Wang, Deepayan Chakrabarti, Chenxi Wang, and Christos Faloutsos. 2003. Epidemic Spreading in Real Networks: An Eigenvalue Viewpoint. In *22nd Symposium on Reliable Distributed Systems (SRDS 2003), 6-8 October 2003, Florence, Italy*. IEEE Computer Society, 25–34.

[29] Dong Wen, Lu Qin, Ying Zhang, Lijun Chang, and Xuemin Lin. 2019. Efficient structural graph clustering: an index-based approach. *VLDB J.* 28, 3 (2019), 377–399.

[30] Changfa Wu, Yu Gu, and Ge Yu. 2019. DPSCAN: Structural Graph Clustering Based on Density Peaks. In *DASFAA*, Vol. 11447. 626–641.

[31] Xiaowei Xu, Nurcan Yuruk, Zhidan Feng, and Thomas A. J. Schweiger. 2007. SCAN: a structural clustering algorithm for networks. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Jose, California, USA, August 12-15, 2007*, Pavel Berkhin, Rich Caruana, and Xindong Wu (Eds.). ACM, 824–833.