# Neighborhood Skyline on Graphs: Concepts, Algorithms and Applications

Qi Zhang[†], Rong-Hua Li[†], Hongchao Qin[†], Yongheng Dai[‡], Ye Yuan[†], Guoren Wang[†]

[†]*Beijing Institute of Technology, Beijing, China;* [‡]*Diankeyun Technologies Co. , Ltd.;*
qizhangcs@bit.edu.cn; lironghuabit@126.com; qhc.neu@gmail.com;
toyhdai@163.com; yuan-ye@bit.edu.cn; wanggrbit@126.com

***Abstract*—Neighborhood inclusion, representing that all the neighbors of a vertex are also adjacent to another vertex, has been recognized as an important relationship between two vertices in a graph. We call a vertex $u$ dominating $v$, denoted by $v \leq u$, if $N(v) \subseteq N(u) \cup \{u\}$ holds, where $N(v)$ denotes the set of neighbors of $v$. Based on such a domination relationship, we propose a concept called neighborhood skyline. The neighborhood skyline is a set of vertices in which any vertex $u$ cannot be dominated by the other nodes in the graph $G$, i.e., $\nexists v \in G, u \leq v$. We study a new problem, called neighborhood skyline computation, and develop a filter-refine search framework, FilterRefineSky, to efficiently find the neighborhood skyline by searching the vertices in a small candidate set instead of in the entire graph. We show that our neighborhood skyline technique can be used to speed up the computation of two well-studied group centrality maximization problems and the maximum clique search problem in graphs. We conduct extensive experiments to evaluate our algorithms on five large real-life datasets, and the results demonstrate the efficiency, effectiveness and scalability of our solutions.**

## I. Introduction

Neighborhood, perhaps, is the most basic concept in graphs, which represents the set of vertices that are adjacent to a given node. Most graph analysis tasks, such as shortest-path computation [1], reachability query [2], and cohesive community detection [3], frequently use the concept of neighborhood. Neighborhood inclusion is a basic relationship between two neighborhoods, representing that all neighbors of one node are also connected with another node.

Recently, the neighborhood inclusion relationship has been recognized as an important operator between two vertices for many graph analysis tasks. For example, in the problem of maximum independent set search, if the neighbors of a node are contained by that of others, then it can be safely pruned as it definitely not be in a maximum independent set. Such a reduction rule captured by neighborhood inclusion can be iteratively and incrementally applied to explore the maximum independent set [4], [5]. For shortest distance queries, a widely adopted approach is to construct an off-line index to speed up online query processing. Neighborhood inclusion arises an equivalence relation rule which can be used to compress the graph and further reduce the size of index structure [6]. In addition, a pre-order derived by neighborhood inclusion yields a novel graph, called threshold graph [7], [8], and much research has been done on threshold graphs, including spanning tree counting [9], characteristic polynomial computation [10] and so on.

In general, the set of neighbors of a vertex $u$, also known as the *open neighborhood*, is denoted by $N(u)$, i.e., $N(u) = \{v \in V | (u, v) \in E\}$. The *closed neighborhood* of $u$ is represented as $N[u] = N(u) \cup \{u\}$. Based on the neighborhood inclusion [7], given two vertices $u$ and $v$, a domination order $v \leq u$, as well as a vicinal pre-order, is defined which means that $v$'s open neighborhood is included in $u$'s close neighborhood, i.e., $N(v) \subseteq N[u]$. We say that $v$ is dominated by $u$ if $v \leq u$ holds for brevity. However, some vertices in a graph do not have this pre-order relation. That is, they cannot be dominated by other vertices. We define such a set of vertices as *neighborhood skyline*. Formally, a vertex $u$ belongs to a neighborhood skyline if and only if $\nexists v \in V, u \leq v$ holds. For example, consider the graph $G$ in Fig. 1, the neighborhood skyline $R$ is the set of all red vertices as there is no vertex in $V \backslash R$ can dominate them.

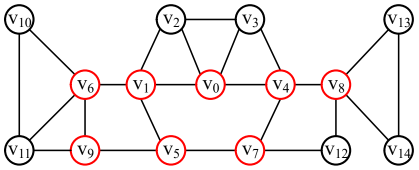


Fig. 1. An example graph $G$ (red vertex: the vertex in the neighborhood skyline set)

In this paper, we investigate a novel problem, namely, the neighborhood skyline computation problem, which is useful for many network analysis applications [11]–[26]. For instance, for the classic group closeness maximization problem [11], [12], we show that we only need to explore the vertices on the neighborhood skyline, instead of the entire graph, thus significantly improving the efficiency. Similarly, our technique can also be used to significantly prune the search space for the classic group betweenness maximization problem and the group harmonic maximization problem, which have been successfully applied in many network analysis related applications [13]–[16]. In addition, the neighborhood skyline technique can also be applied to speed up the maximum clique computation which is a fundamental problem in graph analysis [17]–[27].

To compute the neighborhood skyline on graphs, a basic idea is to calculate neighborhood inclusion relations among all vertices, and then select the neighborhood skyline according to those relations. As a key step, identifying neighborhood inclusion relations for all vertices is equivalent to the set containment join problem which finds all records in a data set $S$ that contain the record $q_i$ for each $q_i$ in a query set $Q$. Here the data set $S$ contains $n$ records and each record $S_i$ is the set $N(i) \cup \{i\}, i \in V$, and the query set $Q$ contains $n$ records where each record $q_i$ is the set $N(i), i \in V$. However, the state-of-the-art set containment join algorithms [28]–[33] are inefficient to compute neighborhood inclusion relations for two reasons. The first is that for each vertex $u$, these algorithms need to perform set containment query over $n$ records (i.e., $n$ vertices), which causes substantial unnecessary comparisons since the neighborhood inclusion relations occur only between $u$ and its 2-hop neighbors. Second, these algorithms usually construct an inverted index on the data set $S$ and a prefix tree on the query set $Q$ to improve efficiency because the size of $Q$ is significantly smaller than that of $S$ in the set containment join problem. While in our neighborhood skyline search problem, $Q$'s size is almost the same as $S$'s size and thus the memory overhead is unacceptable. Considering the specificity of our problem, i.e., a vertex can only be dominated by its 2-hop neighbors, a potential solution is to adapt the algorithm proposed by Brandes *et al.* [7] which is originally used to calculate neighborhood inclusion relations among all vertices. It is easy to derive that such a potential solution consumes $O(md_{max})$ time where $m$ is the number of edges and $d_{max}$ is the maximum degree of vertices in $G$. Clearly, such an algorithm is costly for large graphs because it requires identifying domination orders between each node and its neighbors within 2-hop. In addition, when determining the domination order for a vertex pair, a slight difference between their neighbors can break the neighborhood inclusion relationship, causing substantial unnecessary comparisons.

In this paper, we focus mainly on two aspects to improve the efficiency of such a basic algorithm: 1) reducing the search space of the neighborhood skyline, and 2) efficiently identifying the neighborhood inclusion relation of a vertex pair. To this end, we develop a novel filter-refine search framework which can capture approximate candidates of neighborhood skyline with low costs and calculate the exact skyline by using a bloom filter technique with low memory usage. In addition, we also investigate two applications on group centrality maximization problems and one application on maximum clique computation problem to show the power of our neighborhood skyline technique. To the best of our knowledge, this is the first work that studies the problem of neighborhood skyline search in a graph. Also, we are the first to apply the neighborhood skyline technique to speed up the group centrality maximization and maximum clique search problems. In summary, we make the following contributions.

A novel problem : neighborhood skyline computation. We propose a new concept, namely, neighborhood skyline, and investigate a novel problem of neighborhood skyline search. A filter-refine search framework, i.e., FilterRefineSky, is developed to address this problem which includes a filter and a refining phase. In the filter phase, we identify a small set of candidate vertices for the neighborhood skyline by introducing an edge-constraint on the neighborhood inclusion. We show that such a filter technique can significantly prune unpromising vertices. In the refining phase, we present a bloom-filter-based technique to further speed up the computation of the neighborhood inclusion relation between two vertices.

Applications of neighborhood skyline. We present three applications to exhibit the effectiveness of our neighborhood skyline technique. We show that the neighborhood skyline can be used to speed up the computation of group closeness maximization and group harmonic maximization problems. Our neighborhood skyline based pruning technique can work for most group centrality maximization problems in which the group centrality measure is based on the shortest-path distance, thus it could be of independent interests. We also show that the neighborhood skyline based pruning technique can accelerate the maximum clique computation.

Extensive experiments. We conduct comprehensive experimental studies to evaluate the proposed algorithms using five large real-world datasets. The results show that 1) the proposed FilterRefineSky algorithm is significantly superior to the baseline for finding the neighborhood skyline in a graph; 2) the size of the neighborhood skyline is significantly smaller than the number of vertices in the graph, thus the neighborhood skyline technique will be very effective for pruning for some downstream graph analysis applications; 3) the neighborhood skyline technique can substantially speed up the calculation of finding a group with the highest group closeness or group harmonic centrality measure and identifying the maximum clique search.

Reproducibility. For reproducibility, the source code of this paper is released at github: https://github.com/QiZhang1996/neighborhoodskylines.

**Organization.** We introduce some important notations and formulate our problem in Sec. II. Sec. III presents the neighborhood skyline search algorithms including the baseline and the filter-refine algorithms. The applications of neighborhood skyline on group centrality maximization problems and maximum clique computation problem are developed in Sec. IV. Sec. V reports the experimental results. We survey related studies in Sec. VI and conclude this work in Sec. VII.

## II. Preliminaries

Consider an undirected and unweighted graph $G = (V, E)$, where $V$ is the vertex set and $E$ is the edge set. Denote by $n = |V|$ the number of vertices and $m = |E|$ the number of edges in $G$. For each $u \in V$, $N(u)$ is the set of neighbors of $u$ which is also known as *open neighborhood*, i.e., $N(u) = \{v \in V|(u, v) \in E\}$. Denote the *closed neighborhood* of $u$ by $N[u] = N(u) \cup \{u\}$. Let $deg(u) = |N(u)|$ be the degree of $u$ and $d_{max}$ be the maximum degree of the vertices in $G$. A subgraph $G_S = (V_S, E_S)$ induced by a set of vertices $S$ is a subgraph of $G$ where $V_S = S$ and $E_S = \{(u, v)|u, v \in S, (u, v) \in E\}$.

Below, we give the concepts of *neighborhood inclusion* [7] and *domination order* [7], which are important to define the neighborhood skyline.

*Definition 1:* (Neighborhood Inclusion) Given two arbitrary different vertices $u, v \in V$, if $v$'s open neighborhood is included by $u$'s closed neighborhood, i.e., $N(v) \subseteq N[u]$, we say that $v$ is neighborhood-included by $u$.

*Definition 2:* (Domination Order) Given two arbitrary different vertices $u, v \in V$, the domination order $v \leq u$ is defined if and only if (1) $v$ is neighborhood-included by $u$ and $u$ is not neighborhood-included by $v$, i.e., $N(v) \subseteq N[u]$ and $N(u) \subsetneq N[v]$, or (2) $u$ and $v$ are neighborhood-included by each other and $u$ has a smaller ID than $v$, i.e., $N(v) \subseteq N[u]$, $N(u) \subseteq N[v]$ and $u_{id} < v_{id}$.

*Example 1:* Consider a graph $G$ in Fig. 1. For vertices $v_0$ and $v_2$, we have $N(v_2) \subseteq N[v_0]$ and $N(v_0) \subsetneq N[v_2]$, and thus $v_2$ is dominated by $v_0$ based on neighborhood inclusion, i.e., $v_2 \leq v_0$. While $v_1$ cannot be dominated by $v_0$ because of the presence of $v_5$ and $v_6$ in $v_1$'s neighbors, i.e., $N(v_1) \subsetneq N[v_0]$. For vertices $v_{13}$ and $v_{14}$, we can obtain $N(v_{14}) \subseteq N[v_{13}]$ and $N(v_{13}) \subseteq N[v_{14}]$, which satisfy the case (2) in Definition 2, and thus we denote $v_{14} \leq v_{13}$ since $v_{13}$ has a smaller ID. □

With Definition 1 and Definition 2, the neighborhood inclusion relations and domination orders can only exist between a vertex and its reachable vertices within two hops. In the following, we introduce a concept called *neighborhood skyline*.

*Definition 3:* (Neighborhood Skyline) Given a graph $G = (V, E)$, a vertex set $R \subseteq V$ is a neighborhood skyline if it is the largest set where each vertex $u \in R$ cannot be dominated by other vertices in $V$, i.e., $\forall u \in R, \nexists v \in V, u \leq v$.

According to Definition 3, we formulate the neighborhood skyline search problem as follows.

**Problem formulation.** Given a graph $G$, our goal is to find a vertex set $R$ that is a neighborhood skyline of $G$ based on the domination order.

The following example illustrates the definition of our problem.

*Example 2:* Reconsider the graph $G$ shown in Fig. 1. Take vertex $v_0$ as an example. Clearly, none of the vertices in $V$ can dominate $v_0$, and thus $v_0$ belongs to a neighborhood skyline according to Definition 3. While the vertex $v_{13}$ is not in a neighborhood skyline because we can find that the vertex $v_8$ can dominate $v_{13}$ based on neighborhood inclusion, i.e., $v_{13} \leq v_8$. With Definition 3, the vertex set $R = \{v_0, v_1, v_4, v_5, v_6, v_7, v_8, v_9\}$ is a neighborhood skyline of $G$. We can easily check that all the vertices in $R$ cannot be dominated by any vertices in $V$ and $R$ is the largest. □

**Challenges.** To solve the neighborhood skyline search problem, a basic algorithm is to compute whether a node is dominated by the others, i.e., identify neighborhood inclusion relationships, for each vertex in a graph. Clearly, computing neighborhood inclusion relationships can be considered as the set containment join problem which finds all records in a data set $S$ that contain the record $q_i$ for each $q_i$ in a query set $Q$. Here the data set $S = \{s_1, s_2, ..., s_n\}$ is the set of $n$ records in which $s_i = N(i) \cup \{i\}, i \in V$, and the query set $Q = \{q_1, q_2, ..., q_n\}$ contains $n$ records in which each record $q_i$ is $N(i), i \in V$. The state-of-the-art algorithms [28]–[33] for set containment join problem usually construct an inverted index on $S$ and a prefix tree on $Q$ to improve efficiency since the size of $Q$ is significantly smaller than that of $S$. Such algorithms are often very costly for our neighborhood skyline search problem because $Q$'s size is almost the same as $S$'s size and the memory overhead is unacceptable. In addition, those algorithms perform set containment query for $n$ records (i.e., $n$ vertices), which is inefficient for our problem since a vertex only maintains the neighborhood inclusion relations with its 2-hop neighbors rather than the whole vertex set. Considering this fact, another potential solution to calculate neighborhood inclusion relations is adapting the partial order computation algorithm with $O(md_{max})$ time [7], where $m$ is the number of edges and $d_{max}$ is the maximum degree of vertices in $G$. Such an approach, however, is costly for large graphs since it needs to check domination orders between each node and its 2-hop neighbors. Moreover, a slight difference between the neighbors of two vertices can violate the neighborhood inclusion, causing unnecessary comparisons for the common neighbors. In summary, the challenges of the neighborhood skyline search problem are twofold: (1) how to efficiently prune the vertices that are definitely not contained in the neighborhood skyline; and (2) how to efficiently identify whether there is a neighborhood inclusion relation between two vertices.

To tackle these challenges, in the following sections, we will propose a filter-refine search framework which first identifies a small set of candidate vertices with low costs and then calculates the exact neighborhood skyline on this small set. Equipped with the bloom filter technique, our framework can efficiently compute the neighborhood inclusion relation for a pair of vertices. In addition, we will also apply the neighborhood skyline technique to speed up the computation of two classical group centrality maximization problems as well as the maximum clique search problem.

**Remark.** It is worth remarking that our neighborhood skyline search problem is fundamentally different from the problem studied in [7]. Specifically, our neighborhood skyline focuses on whether a node is dominated by other vertices (we only need to identify those vertices that are not dominated by the others), while the problem studied in [7] aims to identify the entire partial order set (i.e., it requires finding the set of all domination relationships between any pair of nodes). As a result, our goal of developing neighborhood skyline search algorithms is to reduce the number of candidates as many as possible, and to identify neighborhood inclusion between two nodes as efficiently as possible. Further, the techniques for our problem are totally different from that for partial order computation [7]. Additionally, the neighborhood skyline based pruning technique can be very useful for two group centrality maximization problems and the maximum clique computation problem as shown in Sec. IV, thus it could be of independent interests.

## III. Neighborhood skyline search algorithms

In this section, we first give a baseline algorithm, called BaseSky, to find the neighborhood skyline in a graph. To improve efficiency, we then propose a filter-refine search framework, namely, FilterRefineSky, to solve our problem.

### A. A Baseline algorithm

To solve the problem of neighborhood skyline search, a straightforward method is to identify neighborhood inclusion relations for each vertex and then select those vertices that cannot be dominated by others. Brandes *et al.* proposed a subset partial order algorithm to calculate all neighborhood inclusion relationships among the vertices in a graph [7]. We slightly modify this algorithm to compute the neighborhood skyline. The pseudo-code of this algorithm is outlined in Algorithm 1, which is referred to as BaseSky.

As aforementioned, a vertex only keeps neighborhood inclusion relations and domination orders with the vertices that are reachable within two hops. For brevity, we employ $N_2(u)$ to denote those 2-hop reachable vertices from a vertex $u$ and $O(u)$ to indicate the vertex that can dominate $u$, i.e., $u \leq O(u)$. For each vertex $u$, BaseSky initializes the variable $O(u)$ to itself and then explores the vertices in $N_2(u)$ to find the relation of neighborhood inclusion (lines 7-17). For a vertex $w \in N_2(u)$, $T(w)$ records the size of the intersection of $u$'s open neighborhood and $w$'s close neighborhood, i.e., $T(w) = |N(u) \cap N[w]|$. Obviously, if $T(w) = deg(u)$ holds, that means the set $N(u) \cap N[w]$ is equal to the set $N(u)$, and further we have $N(u) \subseteq N[w]$. Thus, $u$ is neighborhood-included by $w$ and BaseSky maintains the domination relation depending on whether $u$ and $w$ are dominated by each other. If not, $u$ is definitely not contained in the neighborhood skyline. In this case, BaseSky sets $O(u)$ to $w$ and stops exploring the remaining vertices in $N_2(u)$ (lines 15-17). On the other hand, we assume that $u$ is the vertex with a smaller ID without losing generality. Since $u$ and $w$ dominate each other, BaseSky sets the $O(w)$ to $u$ according to Definition 3. Whether $u$ belongs to the neighborhood skyline is still unclear, thus BaseSky continues to check the next 2-hop reachable vertex to explore

**Algorithm 1:** BaseSky

```
   Input: $G = (V, E)$.
   Output: The neighborhood skyline vertex set $R$.
1  $R \leftarrow \emptyset$;
2  Let $O$ be an array with size $n$;
3  for $u \in V$ do $O(u) \leftarrow u$;
4  for $u \in V$ do
5      if $u \neq O(u)$ then continue;
6      Initialize an array $T$ with $T(i) = 0, 0 \leq i < n$;
7      for $v \in N(u)$ do
8          for $w \in N[v] \backslash \{u\}$ do
9              $T(w) \leftarrow T(w) + 1$;
10             if $T(w) = deg(u)$ then
11                 if $deg(w) = deg(u)$ then
12                     if $u_{id} > w_{id}$ and $O(u) = u$ then
13                         $O(u) \leftarrow w$;
14                     else if $O(w) = w$ then $O(w) \leftarrow u$;
15                 else
16                     if $O(u) = u$ then
17                         $O(u) \leftarrow w$; break;
18 for $u \in V$ do
19     if $u = O(u)$ then $R \leftarrow R \cup \{u\}$;
20 return $R$;
```

the domination orders (lines 11-14). Finally, for each vertex $u$, if $O(u) = u$, $u$ belongs to the neighborhood skyline, and BaseSky adds $u$ into the result set $R$. Thus, the set $R$ maintains the neighborhood skyline in the graph correctly. Note that in Algorithm 1, to improve efficiency, we only allow $O(u)$ to be updated once since vertex $u$ is definitely not contained in the skyline once we can find a vertex $v$ in $V$ satisfying $u \leq v$.

Below, we give the time and space complexity of Algorithm 1.

*Theorem 1:* Given a graph $G = (V, E)$, in the worst case, Algorithm 1 takes $O(md_{\max})$ time with $O(m + n)$ space [7].

### *B. A novel filter-refine framework*

BaseSky may be not very efficient for the problem of neighborhood skyline search. To improve the efficiency, we propose a filter-refine framework, called FilterRefineSky, to solve our problem. The main idea of FilterRefineSky is to yield a candidate vertex set to reduce the search space, and then compute the exact skyline vertices among them with the bloom filter technique. Below, we first introduce an efficient method to generate the candidate vertex set of the neighborhood skyline, followed by our filter-refine framework in detail.

#### *B.1 The candidate set for neighborhood skyline*

We first give a more stringent definition of neighborhood inclusion, called *edge-constrained neighborhood inclusion*, as follows.

*Definition 4:* (Edge-Constrained Neighborhood Inclusion) Given two arbitrary different vertices $u, v \in V$, we say $v$ is neighborhood-included with edge constraint by $u$ if and only if $v$ is neighborhood-included by $u$ and there is an edge between $u$ and $v$, i.e., $N[v] \subseteq N[u]$.

According to Definition 4, the edge-constrained neighborhood inclusion relations can only exist between a vertex and its neighbors. In the following, we define the *edge-constrained domination order* for two vertices in a graph.

*Definition 5:* (Edge-Constrained Domination Order) Given two vertices $u, v \in V$, the edge-constrained domination order $v \sqsubset u$ is defined if and only if (1) $v$ is edge-constrained neighborhood-included by $u$, and $u$ is not edge-constrained neighborhood-included by $v$, i.e., $N[v] \subsetneqq N[u]$, or (2) $u$ and $v$ are edge-constrained neighborhood-included by each other and $u$ has a smaller ID than $v$, i.e., $N[v] = N[u]$ and $u_{id} < v_{id}$.

Based on the edge-constrained domination order, we can calculate a set $C$, called *neighborhood candidates*, in which each vertex $u$ satisfies $\nexists v \in V, u \sqsubset v$. We have the following lemma which is useful for our FilterRefineSky algorithm to obtain the neighborhood candidates for the neighborhood skyline.

*Lemma 1:* Given a graph $G = (V, E)$, the neighborhood skyline set $R$ based on the domination order is a subset of the neighborhood candidates $C$ calculated by the edge-constrained domination order.

*Proof:* We prove this lemma by contradiction. Assume that vertex $u$ belongs to the neighborhood skyline of $G$, but is not a neighborhood candidate, i.e., $u \in R$ and $u \notin C$. Clearly, $u \notin C$ means that there exists a vertex $v$ satisfying $u \sqsubset v$. According to Definition 5, the following two cases should be considered. The first is when $N[u] \subsetneqq N[v]$, we have $N(u) \cup \{u\} \subsetneqq N[v]$ and further $N(u) \subsetneqq N[v]$ holds. Since $u$ and $v$ are connected, $N[u] - \{v\} \subsetneqq N[v] - \{v\} = N(v)$ holds which indicates that $N(v)$ cannot be included by $N[u]$. Therefore, $u$ is not contained in the neighborhood skyline by Definition 3, because vertex $v$ dominates $u$ which contradicts our assumption that $u \in R$. In the second case of $u \sqsubset v$ where $N[u] = N[v]$ and $v$ has a smaller ID than $u$, we have $N(u) \subseteq N[v]$ and $N(v) \subseteq N[u]$ (the case (2) in Definition 2). By considering the IDs of vertices, we can conclude that $u \leq v$, and thus $u$ does not belong to the neighborhood skyline $R$, which also violates the assumption that $u \in R$. Putting it all together, there is no vertex that belongs to $R$ but not to $C$, and thus $R$ is a subset of $C$. □

*Example 3:* Reconsider the graph $G$ illustrated in Fig. 1. According to Definition 3, vertex $v_{12}$ is not in the neighborhood skyline. This is because there exists a vertex $v_4$ that can dominate $v_{12}$, i.e., $v_{12} \leq v_4$. While from the perspective of edge-constrained domination order, $v_{12}$ is a neighborhood candidate as none of a vertex $v$ in $G$ satisfies $v_{12} \sqsubset v$. By considering the edge-constraint domination order, the neighborhood candidates in $G$ are the vertices in $C = \{v_0, v_1, v_4, v_5, v_6, v_7, v_8, v_9, v_{12}\}$. In previous examples, the neighborhood skyline in $G$ is $R = \{v_0, v_1, v_4, v_5, v_6, v_7, v_8, v_9\}$. Clearly, $R$ is indeed a subset of $C$ as the analysis in Lemma. 1. □

According to Lemma. 1, the neighborhood skyline $R$ is a subset of the set of neighborhood candidates $C$. Thus we can use $C$ as an approximation of $R$ because computing $C$ only requires maintaining the size of the intersection of close neighborhoods for two ends of an edge. The algorithm to calculate the neighborhood candidates $C$, called FilterPhase, is depicted in Algorithm 2, which is an important phase in our FilterRefineSky. The workflow of FilterPhase is similar to that of BaseSky. The difference is that we only consider the edge-constrained domination orders which only exist between a vertex and its neighbors. Thus, in Algorithm 2, we identify edge-constrained domination orders for each vertex by exploring its neighbors (lines 6-15), and FilterPhase finally returns the neighborhood candidates according to the indicators $O(*)$. Note that we also allow these indicators to be maintained once in FilterPhase like Algorithm 1 for improvement. The time and space complexity of Algorithm 2 are given in Theorem 2.

*Theorem 2:* Given a graph $G = (V, E)$, in the worst case, Algorithm 2 takes $O(m)$ time using $O(m + n)$ space.

**Algorithm 2:** FilterPhase

```
Input: $G = (V, E)$, an array $O$ with size $n$.
Output: The neighborhood candidate set $C$, the array $O$.
$C \leftarrow \emptyset$;
for $u \in V$ do $O(u) \leftarrow u$;
for $u \in V$ do
    if $u \neq O(u)$ then continue;
    Initialize an array $T$ with $T(i) = 0, 0 \leq i < n$;
    for $v \in N(u)$ do
        $T(v) \leftarrow T(v) + 1$;
        if $T(v) = deg(u)$ then
            if $deg(v) = deg(u)$ then
                if $u_{id} > v_{id}$ and $O(u) = u$ then
                    $O(u) \leftarrow v$;
                else if $O(v) = v$ then $O(v) \leftarrow u$;
            else
                if $O(u) = u$ then
                    $O(u) \leftarrow v$; break;
for $u \in V$ do
    if $u = O(u)$ then $C \leftarrow C \cup \{u\}$;
return $(C, O)$;
```

**Algorithm 3:** FilterRefineSky

```
   Input: $G = (V, E)$.
   Output: The neighborhood skyline vertex set $R$.
1  $R \leftarrow \emptyset$; $C \leftarrow \emptyset$;
2  Let $O$ be an array with size $n$;
3  $(C, O) \leftarrow$ FilterPhase$(G, O)$;
4  for $u \in C$ do Construct bloom filter $BF(u)$;
5  for $u \in C$ do
6      if $u \neq O(u)$ then continue;
7      $issky \leftarrow true$;
8      for $v \in N(u)$ do
9          if $issky = false$ then break;
10         for $w \in N(v) \backslash \{u\}$ do
11             if $issky = false$ then break;
12             if $deg(w) < deg(u)$ or $O(w) \neq w$ then
13                 continue;
14             if $BF(u) \& BF(w) \neq BF(u)$ then continue;
15             for $x \in N(u) \backslash \{v\}$ do
16                 if $BFcheck(u, w, x) = false$ then
17                     $issky \leftarrow false$; break;
18                 if $NBRcheck(w, x) = false$ then
19                     $issky \leftarrow false$; break;
20             if $issky = false$ then
21                 $issky \leftarrow true$; continue;
22             if $deg(w) = deg(u)$ then
23                 if $u_{id} > w_{id}$ and $O(u) = u$ then
24                     $O(u) \leftarrow w$;
25                 $issky \leftarrow true$;
26             else if $O(u) = u$ then
27                 $O(u) \leftarrow w$; $issky \leftarrow false$;
28 for $u \in V$ do
29     if $u = O(u)$ then $R \leftarrow R \cup \{u\}$;
30 return $R$;
```

*Proof:* In Algorithm 2, for each vertex $u$, it needs to find $u$'s neighbors, which consumes $\sum_{u \in V} deg(u) = O(m)$ time. We can easily derive that the space complexity of Algorithm 2 is $O(m+n)$, as the algorithm only needs to maintain a linear-size data structure $T(v)$ to record the number of common neighbors of $u$ and $v$. □

### *B.2 The filter-refine framework*

Here we propose a filter-refine framework to compute the neighborhood skyline, called FilterRefineSky. Before further processing, we first briefly introduce the bloom filter technique which is used in FilterRefineSky to efficiently identify the neighborhood inclusion relation of two sets.

Given a set of elements $X$, and a hash function that can randomly map an element $x \in X$ to a number $h(x)$ in $\{1, 2, ..., s\}$, denoted as $h : X \rightarrow \mathbb{Z}$. The bloom filter of $X$ is defined as $BF(X) = \{h(x) | x \in X\}$ [34]. In practical applications, the bloom filter $BF(X)$ can be implemented as a $b$-size bits array with each bit equal to 0, and then set the $(h(x)\ mod\ b)$-th bit to 1 for each $x$ in $X$. In this implementation, we use $BF_i(X)$ to represent the $i$-th bit of $BF(X)$ for brevity. Bloom filter can be used to determine whether an element belongs to a set. Consider a set of elements $X$ and an element $e$, if $h(e) \in BF(X)$ holds (or, the $BF_{(h(x)\ mod\ b)}(x)$ equals 1), then we can claim that $e$ belongs to $X$; otherwise $e$ cannot be in $X$. Note that the bloom filter is a probabilistic data structure; it is possible to obtain a false-positive result, but not to get a false-negative answer.

Recall that the key issue to compute the neighborhood skyline is to identify the domination order $u \leq v$, i.e., $N(u) \subseteq N[v]$. In FilterRefineSky, the candidate set of neighborhood skyline, $C$, is calculated first by FilterPhase based on the edge-constrained domination order (i.e., $N[u] \subseteq N[v]$). Therefore, determining whether a vertex $u$ belongs to the neighborhood skyline requires only identifying the relation $N(u) \subseteq N(v)$ between $u$ and $v \in N_2(u)$. To this end, we can construct a bloom filter $BF(u)$ with size equal to $b$ for each vertex $u \in C$ based on its neighbors $N(u)$ as aforementioned. With those bloom filters, if $w$ is a common neighbor of $u$ and $v$, then the $(h(w)\ mod\ b)$-th bit is 1 for both $BF(u)$ and $BF(v)$. For an arbitrary $i$-th bit, if $BF_i(u)$ is 1 while $BF_i(v)$ equals 0, that means at least one vertex $w$ is a neighbor of $u$ but not that of $v$ (i.e., $w \in N(u), w \notin N(v)$). Thus, $u$ cannot be dominated by $v$ based on Definition 2. Equipped with the bloom filters, we can quickly identify the domination order in the FilterRefineSky framework.

Algorithm 3 outlines the pseudo-code of FilterRefineSky. The algorithm first sets the collections $R$ and $C$ to empty, and initializes an array $O$ to maintain the domination relations (lines 1-2). It then invokes Algorithm 2 (FilterPhase) to calculate the neighborhood candidates $C$, and constructs the bloom filter structures for those candidates (lines 3-4). Since the neighborhood skyline is a subset of $C$ (Lemma. 1), we only need to identify the skyline vertices in $C$ by checking whether they can be dominated by the others based on neighborhood inclusion (lines 5-27). For each vertex $u \in C$, a boolean variable $issky$, initialized as true, is used to indicate the status of $u$. If $issky$ is false, $u$ does not belong to the neighborhood skyline. To determine the value of $issky$, the algorithm explores the domination orders between $u$ and its 2-hop neighbors because the 1-hop neighbors have been explored in FilterPhase. For each $w \in N_2(u)$, if $deg(w) < deg(u)$, $u$ is definitely not dominated by $w$, thus FilterRefineSky explores the next 2-hop vertex (lines 12-13). Otherwise, the algorithm checks the number of common neighbors of $u$ and $w$ by their bloom filter structures. If $BF(u) \& BF(w) \neq BF(u)$, there exists at least one neighbor of $u$ that is not connected to $w$ based on the property of bloom filter. Thus, $u$ cannot be dominated by $w$. FilterRefineSky stops the current loop and identifies the next vertex in $N_2(u)$ (line 14). Otherwise, the algorithm further verifies each neighbor $x$ using two constraints, namely, BFcheck and NBRcheck. BFcheck is coarse-grained to check high ← low

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $BF(v_4)$: | 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| $BF(v_5)$: | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| $BF(v_6)$: | 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| $BF(v_{12})$: | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

Fig. 2. The bloom filter structures of $v_4, v_5, v_6, v_{12}$

whether $x$ is also a neighbor of $w$ by the $BF_{(H(x)\ mod\ b)}(w)$ bit. Once this bit is 0, $u$ cannot be dominated by $w$ (lines 16-17). While if $x$ overcomes BFcheck, we employ NBRcheck to perform accurate identification to eliminate the false-positive answers after BFcheck. That is, NBRcheck checks if each neighbor $x$ of $u$ is also linked to $w$ with the adjacency list (lines 18-19). If one of the BFcheck and NBRcheck fails, $issky$ is set to false and FilterRefineSky stops searching the next common neighbor $x$. When $issky$ is false, the algorithm resets $issky$ to true and explores the next 2-hop vertex (lines 20-21). While $issky$ equals true indicates that $w$ can dominate $u$, and thus FilterRefineSky processes $u$ and $w$ based on their degrees (lines 22-27). Finally, FilterRefineSky outputs the neighborhood skyline set $R$ according to their indicators. Like Algorithm 1, in FilterRefineSky, for each vertex $u$, the $O(u)$ is also maintained only once. Note that in FilterRefineSky, for each neighbor $x$ of $u$, BFcheck may cause false-positive answer to the question $x \in N(w)$?. We further use NBRcheck to perform an exact validation by visiting the adjacency list when the answer of BFcheck is true. Thus, the FilterRefineSky algorithm can exactly calculate the neighborhood skyline.

*Example 4:* Still consider the graph $G$ in Fig. 1. Suppose that the hash function $h$ is defined as $h(x) = x\ mod\ 7$ and the size of our bloom filters is equal to 7, i.e., $b = 7$. Fig. 2 illustrates the bloom filter structures of $v_4, v_5, v_6$ and $v_{12}$. The set of neighborhood candidates in FilterRefineSky is $C = \{v_0, v_1, v_4, v_5, v_6, v_7, v_8, v_9, v_{12}\}$ and we only search skyline vertices in $C$ according to Lemma. 1. When determining whether $v_5$ belongs to the neighborhood skyline, $v_6$ is a 2-hop neighbor and we can derive that $BF(v_5)\&BF(v_6) = 0000110 \neq BF(v_5)$ with Fig. 2. That means $v_5$ has at least one neighbor that does not link to $v_6$ and thus it cannot be dominated by $v_6$. For vertex $v_{12}$, the 2-hop neighbor $v_5$ can pass through BFcheck due to $BF(v_{12})\&BF(v_5) = BF(v_{12})$ from Fig. 2. We further check the domination relation by NBRcheck since the bloom filter structure may obtain false-positive results. Obviously, $v_8$ is a neighbor of $v_{12}$ but not that of $v_5$, i.e., $N(v_{12}) \subsetneq N(v_5)$, thus $v_5$ cannot dominate $v_{12}$. While for vertex $v_4$, it passes the BFcheck (i.e., $BF(v_{12})\&BF(v_4) = BF(v_{12})$) and also the NBRcheck (i.e., $N(v_{12}) \subseteq N(v_4)$), thus we have $v_{12} \leq v_4$. Finally, Algorithm 3 outputs the neighborhood skyline as $R = C \backslash \{v_{12}\} = \{v_0, v_1, v_4, v_5, v_6, v_7, v_8, v_9\}$. □

In FilterRefineSky algorithm, the bloom filter technique is used to identify the relation $N(u) \subseteq N(v)$ between $u$ and its 2-hop neighbor $v$. To speed up the calculation, we only use one hash function based on bit-wise operations to construct bloom filter structures as used in [2]. Below, we analyze the probability of false-positive for $N(u) \subseteq N(v)$.

*Lemma 2:* Given a graph $G = (V, E)$ and two vertices $u$, $v \in N_2(u)$, the probability of false-positive for $N(u) \subseteq N(v)$ is $(1 - (1 - \frac{1}{d_{max}})^{d(v)})^{|N(u)-N(v)|}$.

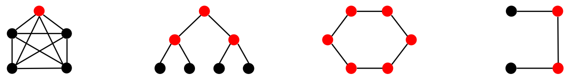

(a) Clique (b) Tree (c) Circle (d) Path

Fig. 3. Neighborhood skyline and neighborhood candidates in several special graphs (red vertex: the vertex in $R$ and $C$)

*Proof:* In FilterRefineSky, the size of bloom filter is $d_{max}$, and we use one hash function. For each $x \in N(v)$, we construct the bloom filter structure $BF(v)$. The probability of $h(x)$ not hitting a particular bit $BF_i(v)$ is $(1 - \frac{1}{d_{max}})$. Further, the probability that any particular bit $BF_i(v)$ is equal to 0 is $(1 - \frac{1}{d_{max}})^{d(v)}$ since it must be avoided by all $d(v)$ hash values. Therefore, the probability that a particular bit $BF_i(v)$ is set to 1 is $(1 - (1 - \frac{1}{d_{max}})^{d(v)})$. Consider a vertex $x \in N(u)$, we answer the query "Does $x$ belong to $N(v)$" by identifying whether the bit $BF_i(v)$ $h(x)$ hit is 1. Clearly, the probability of false-positive for $x \in N(v)$ is no larger than the probability of $BF_i(v) = 1$, i.e., $P(\text{error}) \leq (1 - (1 - \frac{1}{d_{max}})^{d(v)})$. Moreover, only the identifications for $x$ in the set $(N(u) - N(v))$ can yield false-positive results, and each $x \in (N(u) - N(v))$ is independent. Hence, the probability of false-positive for $N(u) \subseteq N(v)$ is $P(\text{error})^{|N(u)-N(v)|}$, i.e., $(1 - (1 - \frac{1}{d_{max}})^{d(v)})^{|N(u)-N(v)|}$. □

We analyze the time and space complexity of Algorithm 3 as follows.

*Theorem 3:* Given a graph $G = (V, E)$, the worst-case time complexity of Algorithm 3 is $O(m + d_{\max} \sum_{u \in C} deg(u)^2)$. Algorithm 3 outputs the neighborhood skyline of $G$ using $O(m + |C|d_{\max})$ space.

*Proof:* In Algorithm 3, the filter phase (Algorithm 2) takes $O(m)$ time in the worst case. The bloom filter structures are built by a hash function based on bit-wise operations which is very efficient, thus the time for bloom filter construction can be bounded by $O(m)$ time. In the refining phase, for each vertex $u$ in the neighborhood candidates $C$, Algorithm 3 needs to find $u$'s 2-hop neighbors, which consumes $deg(u) * d_{max}$ time. For a vertex pair $< u, w >$, the algorithm identifies whether it is also a neighbor of $w$ for each neighbor of $u$. By using a map structure for $w$'s neighbors, the time consumption can be bounded by $deg(u)$. Hence, the time complexity of Algorithm 3 is $O(m + d_{\max} \sum_{u \in C} deg(u)^2)$. Second, we analyze the space complexity of Algorithm 3. For each vertex $u$ in $C$, the algorithm constructs a bloom filter structure with size equal to $d_{max}$, thus taking at most $O(|C|d_{max})$. In addition, the algorithm needs to store the graph, thus the total space complexity is $O(m + |C|d_{\max})$. □

Note that in Theorem 3, the worst-case time complexity of FilterRefineSky relies on the size of $C$. Fig. 3 illustrates the skyline vertices (colored red) and neighborhood candidates (colored red) in several special graphs. For a clique (Fig. 3(a)), the sizes of neighborhood skyline $R$ and neighborhood candidate set $C$ equal 1 which are significantly less than the number of vertices. In a complete binary tree (Fig. 3(b)), $R$ and $C$ both include all non-leaf vertices. In a circle (Fig. 3(c)) and a path (Fig. 3(d)), we have $|R| = |C| = |V|$ and $|R| = |C| = |V| - 2$, respectively. Clearly, $|C|$ and $|R|$ are various for different graphs. In general, in power-law graphs where the degree distribution follows a power-law distribution [35], the sizes of $R$ and $C$ are often much smaller than $n$ because the vertices with small degrees are more easily dominated by others. Hence, the FilterRefineSky algorithm works efficiently in power-law graphs, which is also confirmed in our experiments.

**Remark.** Given two sets, the containment relationship between them is a deterministic concept. Therefore, we focus on the exact neighborhood skyline computation based on such a deterministic concept. It is also interesting to study the "approximate neighborhood skyline" based on approximate domination relationships, which clearly requires new definitions and new algorithms with different techniques. We leave this problem as an interesting future direction.

## IV. Applications of Neighborhood Skyline

In this section, we focus on two widely-used centrality measures, namely, closeness and harmonic centralities, and study two applications on group closeness maximization search and group harmonic maximization search to show the effectiveness of our neighborhood skyline technique. In addition, we also show that the neighborhood skyline technique can be applied to speed up the maximum clique computation.

### *A. Group closeness maximization*

The identification of important vertices in a network is a central task in graph analysis. To this end, various centrality measures are proposed as indicators for the importance of a vertex. Everett *et al.* further formalized the concepts of group centrality measures which allow determining the importance of a vertex set [36]. An inherent problem is how to elaborately pick the group of vertices with a given size $k$ that can yield the maximum score of a certain group centrality measure. Such NP-hard problems are known as group centrality maximization problems which arise in facility location, leader selection [37], and influence maximization [38], and so on. In this paper, we mainly revolve two widely-used measures, i.e, closeness and harmonic centralities, and study two applications on group closeness maximization search and group harmonic maximization search.

### *A.1 Problem formulation*

Let $d(u,v)$ represent the shortest-path distance between vertex $u$ and vertex $v$. The distance between vertex $u$ and a vertex set $S \subseteq V$ is denoted as $d(u,S) = \min_{s\in S} d(u,s)$. Below, we give the definitions of *vertex closeness centrality* and *group closeness centrality*.

*Definition 6:* (Vertex Closeness Centrality) For a vertex $u$ in $G=(V,E)$, the vertex closeness centrality of $u$ is defined as: $C(u) = \frac{n}{\sum_{v\in V\setminus\{u\}} d(v,u)}$.

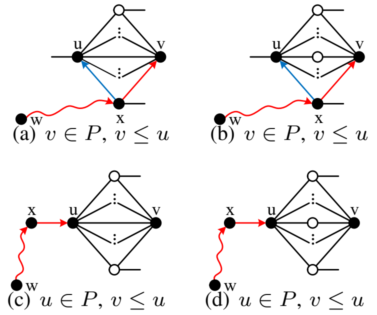
*Definition 7:* (Group Closeness Centrality) For a vertex set $S \subseteq V$, the group closeness centrality of $S$ is defined as: $GC(S) = \frac{n}{\sum_{v\in V\setminus S} d(v,S)}$.

Based on *vertex closeness centrality* and *group closeness centrality*, the problem of group closeness maximization is formulated as follows.

**Group Closeness Maximization Problem.** Given a graph $G$ and an integer $k$, the goal of Group Closeness Maximization (GCM) problem is to identify a group $S^* \subseteq V$ of size $k$ with maximum group closeness centrality, i.e., $S^* = \arg\max_{S\subseteq V} \{GC(S) : |S| = k\}$.

The following example illustrates the definition of the GCM problem.

*Example 5:* Consider the graph $G$ shown in Fig. 1. Suppose that the size of a group is $k=1$. From Fig. 1, we can easily check that $v_0$ is the answer with $GC(v_0) = 1/2$ because the sum of distances between vertex $v_0$ and the others is minimum. While when $k$ equals 3, the vertex set $S = \{v_0, v_6, v_8\}$ is the required group with maximum group closeness $GC(S) = 15/14$. □



(a) $v \in P$, $v \leq u$ (b) $v \in P$, $v \leq u$ (c) $u \in P$, $v \leq u$ (d) $u \in P$, $v \leq u$

Fig. 4. The cases of the shortest-path between $w$ and $S$

The GCM problem was shown to be NP-hard [11]. As a special case of $p$-median, the GCM problem can be solved by exact ILP (Integer Linear Programming) solvers [12]. These exact methods, however, can only handle very small graphs. They are not applicable for networks with millions of vertices and edges. To this end, recent research mainly investigates approximation algorithms following greedy strategies to solve this problem [11]–[13], [39]. In general, the main idea of these greedy frameworks is to select the vertex with the largest *marginal gain* of group closeness centrality into the result set $S$ at each round until the size of $S$ equals $k$. In each round, the marginal gain of every vertex $u \in V\setminus S$ can be calculated by $GC(S\cup\{u\}) - GC(S)$. We refer to such a simple implementation based on this idea as BaseGC. Given the size of the required group $k$, the marginal gain calculation is performed $k(2*n-k+1)/2$ times in the BaseGC algorithm where $n$ is the number of vertices. Clearly, calculating the marginal gain for each vertex $u \notin S$ in every round requires exploring the shortest-path distance from a vertex to the set $S\cup\{u\}$, which is very costly. In the following, we will propose a neighborhood skyline based pruning technique to speed up the BaseGC algorithm.

### *A.2 A neighborhood skyline based solution*

Below, we first derive a useful lemma, based on which we come up with a pruning technique to speed up the calculation for the GCM problem.

*Lemma 3:* Given a group $S \subseteq V$, for vertex $u$ and vertex $v$ in $G$ with $v \leq u$ and $u, v \notin S$, $GC(S\cup\{u\}) \geq GC(S\cup\{v\})$ holds.

*Proof:* According to Definition 7, we have the following equations for $GC(S\cup\{u\})$ and $GC(S\cup\{v\})$.

$$\begin{aligned} GC(S\cup\{u\}) &= \frac{n}{\sum_{w\in V\setminus(S\cup\{u\})} d(w, S\cup\{u\})} \\ = &\frac{n}{(\sum_{w\in V\setminus(S\cup\{u,v\})} d(w, S\cup\{u\})) + d(v, S\cup\{u\})} \end{aligned} \quad (1)$$

$$\begin{aligned} GC(S\cup\{v\}) &= \frac{n}{\sum_{w\in V\setminus(S\cup\{v\})} d(w, S\cup\{v\})} \\ = &\frac{n}{(\sum_{w\in V\setminus(S\cup\{u,v\})} d(w, S\cup\{v\})) + d(u, S\cup\{v\})} \end{aligned} \quad (2)$$

We compare Eq. (1) and Eq. (2) by considering the two parts of the denominators respectively. Consider a vertex $w \in$

**Algorithm 4:** NeiSkyGC $(G, k)$

```
    Input: $G = (V, E)$, an integer $k \geq 1$.
    Output: Set $S \subseteq V$ with $|S| = k$, s.t. $GC(S)$ is maximum.
1   Calculate the neighborhood skyline $R$;
2   $S \leftarrow \emptyset$;
3   while $|S| < k$ do
4       $v \leftarrow \arg\max_{u \in (R \setminus S)} (GC(S \cup \{u\}) - GC(S))$;
5       $S \leftarrow S \cup \{v\}$;
6   return $S$;
```

$V \backslash (S \cup \{u, v\})$ (the first part). Let $P =< w, ..., v_i, ..., s >$ be the shortest-path between $w$ and $S$ where $s$ is the vertex in $S$ with the minimum $d(w, s)$. If the path $P$ does not pass through $u$ and $v$, adding $u$ or $v$ to $S$ would not affect $d(w, s)$, thus we have $d(w, s) = d(w, S) = d(w, S \cup \{u\}) = d(w, S \cup \{v\})$. On the other hand, i.e., $P$ includes $u$ or $v$, we compare $d(w, S \cup \{u\})$ and $d(w, S \cup \{v\})$ as follows. In Fig. 4(a) and Fig. 4(b), $v$ is an intermediate vertex in $P$, and if we add $v$ into $S$, the shortest-path between $w$ and $S \cup \{v\}$ is $P' =< w, ..., x, v >$. Clearly, we can derive an alternative path $P'' =< w, ..., x, u >$ by replacing $v$ with $u$, as $x$ is also a neighbor of $u$ based on $v \leq u$. It follows that $d(w, u) = d(w, v)$ holds, and further we have $d(w, S \cup \{u\}) = d(w, S \cup \{v\})$. When adding $u$ into $S$ in the cases of Fig. 4(a) and Fig. 4(b), $d(w, S \cup \{u\}) = d(w, S \cup \{v\})$ still holds, which is consistent. Contrarily, for the cases of Fig. 4(c) and Fig. 4(d), when adding $u$ into $S$, the shortest-path between $w$ and $S \cup \{u\}$ is $P =< w, ..., x, u >$ and we have $d(w, S \cup \{u\}) = d(w, u)$. However, using $v$ instead of $u$ definitely enlarges the shortest-path distance between $w$ and the group since $x$ is not linked to $v$ directly. Specifically, in Fig. 4(c), we have $d(w, S \cup \{v\}) = d(w, v) = d(w, u) + 1 = d(w, S \cup \{u\}) + 1$, and for Fig. 4(d), $d(w, S \cup \{v\})$ is equal to $d(w, u) + 2 = d(w, S \cup \{u\}) + 2$. Thus, $d(w, S \cup \{u\}) < d(w, S \cup \{v\})$ holds. Considering the second part of the denominators in Eq.1 and Eq.2, we have $d(v, S \cup \{u\}) = d(u, S \cup \{v\})$ which are always equal to either 1 or 2. Putting it all together, the denominator of Eq.1 is no larger than that of Eq.2, thus $GC(S \cup \{u\}) \geq GC(S \cup \{v\})$ holds. □

*Example 6:* Consider the graph $G$ shown in Fig. 1. Suppose that the size of the desired group is $k = 1$ and the initial $S$ is empty. Take the vertices $v_0$ and $v_2$ as examples. As mentioned before, we have $v_2 \leq v_0$. With Fig. 1, the group closenesses when adding $v_0$ or $v_2$ into $S$ can be easily calculated, i.e., $GC(S \cup \{v_0\}) = \frac{n}{\sum_{w \in V \setminus \{v_0\}} d(w, v_0)} = 15/30 = 1/2$, $GC(S \cup \{v_2\}) = \frac{n}{\sum_{w \in V \setminus \{v_2\}} d(w, v_2)} = 15/36 = 5/12$. Clearly, selecting $v_0$ to the group $S$ yields a larger group closeness score than picking $v_2$ to $S$. This is because for each vertex $w$, we still have $d(w, v_0) \leq d(w, v_2)$ based on the domination order $v_2 \leq v_0$, which is consistent with Lemma. 3. □

Armed with Lemma. 3, we develop a general framework, called NeiSkyGC, to find $k$ vertices with the maximum group closeness. The pseudo-code of NeiSkyGC is depicted in Algorithm 4. The difference from BaseGC is that we only need to compute the marginal gains of group closeness centrality for the skyline vertices that are not included in $S$ instead of all vertices in $V \backslash S$ (line 4). For the desired group size $k$, the marginal gain calculation is only called $k(2 * r - k + 1)/2$ times in our NeiSkyGC algorithm in which $r$ is the size of the neighborhood skyline, i.e., $r = |R|$. Equipped with the neighborhood skyline based pruning technique, the NeiSkyGC can substantially reduce the number of marginal gain calculations compared to the BaseGC algorithm. As a result, our technique can significantly speed up the search of a $k$-size group with the maximum group closeness score.

*Example 7:* Reconsider the graph $G$ in Fig. 1. To yield a group with size $k = 3$, the BaseGC algorithm needs to compute the marginal gains for $15 + 14 + 13 = 42$ vertices in total. While utilizing our pruning technique, the NeiSkyGC algorithm only requires to compute the marginal gains of group closeness for the skyline vertices, thus it performs only $8 + 7 + 6 = 21$ marginal gain calculations. Obviously, NeiSkyGC significantly reduces the number of marginal gain calculations compared with BaseGC, thus making it more efficient. □

## *B. Group harmonic maximization*

### *B.1 Problem formulation*

Here we first introduce the concepts of *vertex harmonic centrality* and *group harmonic centrality*, which are essential to formulate the problem of group harmonic maximization.

*Definition 8:* (Vertex Harmonic Centrality) For a vertex $u$ in $G$ , the harmonic centrality of $u$, denoted by $H(u)$, is defined as $H(u) = \sum_{v \in V \setminus \{u\}} \frac{1}{d(v,u)}$.

*Definition 9:* (Group Harmonic Centrality) For a vertex set $S \subseteq V$, the group harmonic centrality of $S$, denoted by $GH(S)$, is defined as $GH(S) = \sum_{v \in V \setminus S} \frac{1}{d(v,S)}$.

With Definition 8 and Definition 9, the group harmonic maximization problem is defined as follows.

**Group Harmonic Maximization Problem.** Given a graph $G$ and an integer $k$, the problem of Group Harmonic Maximization (GHM) aims to find a group $S^* \subseteq V$ of size $k$ with maximum group harmonic centrality, i.e., $S^* = \arg\max_{S \subseteq V} \{GH(S) : |S| = k\}$.

The following example illustrates the definition of the GHM problem.

*Example 8:* Consider the graph $G$ shown in Fig. 1. When the size of a required group is $k = 1$, we can easily check that $v_4$ is the answer since the group harmonic score of $\{v_4\}$ equals 49/6 which is the highest among all vertices in Fig. 1. In the case of $k = 3$, the result of the GHM problem is $S = \{v_4, v_6, v_8\}$ with the maximum group harmonic $GH(S) = 11$. □

The GHM problem is also NP-hard [13]. By relating the GHM problem to the $p$-median problem, Angriman *et al.* showed that the greedy framework can be used to solve this problem with a 0.5-approximation guarantee, despite the non-monotonicity of $GH(\cdot)$ [13]. Similar to BaseGC for the GCM problem, such a greedy framework to solve the GHM problem, called BaseGH, works as follows. It first puts the vertex with the highest harmonic centrality to the group; and then iteratively adds the vertex with the highest marginal gain $GH(S \cup \{u\}) - GH(S)$ to the group. Given a desired size $k$, the marginal gain calculation is invoked $k(2 * n - k + 1)/2$ times, which is the main computational bottleneck in BaseGH.

### *B.2 A neighborhood skyline based algorithm*

In the following, we establish an important lemma, based on which we come up with an efficient pruning rule for the GHM problem.

*Lemma 4:* Given a group $S \subseteq V$, for vertex $u$ and vertex $v$ in $G$ with $v \leq u$ and $u, v \notin S$, $GH(S \cup \{u\}) \geq GH(S \cup \{v\})$ holds.

**Algorithm 5:** NeiSkyGH $(G, k)$

```
  Input: $G = (V, E)$, an integer $k \geq 1$.
  Output: Set $S \subseteq V$ with $|S| = k$, s.t. $GH(S)$ is maximum.
1 Calculate the neighborhood skyline $R$;
2 $S \leftarrow \emptyset$;
3 while $|S| < k$ do
4     $v \leftarrow \arg\max_{u \in (R \setminus S)} (GH(S \cup \{u\}) - GH(S))$;
5     $S \leftarrow S \cup \{v\}$;
6 return $S$;
```

*Proof:* Based on Definition 9, we have $GH(S \cup \{u\})$ and $GH(S \cup \{v\})$ as follows.

$$\begin{aligned} GH(S \cup \{u\}) &= \sum_{w \in V \setminus (S \cup \{u\})} \frac{1}{d(w, S \cup \{u\})} \\ = ( &\sum_{w \in V \setminus (S \cup \{u,v\})} \frac{1}{d(w, S \cup \{u\})}) + \frac{1}{d(v, S \cup \{u\})} \end{aligned} \tag{3}$$

$$\begin{aligned} GH(S \cup \{v\}) &= \sum_{w \in V \setminus (S \cup \{v\})} \frac{1}{d(w, S \cup \{v\})} \\ = ( &\sum_{w \in V \setminus (S \cup \{u,v\})} \frac{1}{d(w, S \cup \{v\})}) + \frac{1}{d(u, S \cup \{v\})} \end{aligned} \tag{4}$$

We prove this lemma by considering the vertex $w \in V \setminus (S \cup \{u, v\})$ and the vertex $v/u$ in Eq.3 and Eq.4. The proofs of $d(w, S \cup \{u\}) \leq d(w, S \cup \{v\})$ and $d(v, S \cup \{u\}) = d(u, S \cup \{v\})$ are similar to that of Lemma. 3. Further, we have $\frac{1}{d(\cdot, S \cup \{u\})} \geq \frac{1}{d(\cdot, S \cup \{v\})}$, thus $GH(S \cup \{u\}) \geq GC(S \cup \{v\})$ holds. □

*Example 9:* Consider the graph $G$ in Fig. 1. Suppose that the size of a desired group is $k = 1$ and the initial $S$ is empty. For the vertices $v_0$ and $v_2$ with $v_2 \leq v_0$, the group harmonic scores when adding $v_0$ or $v_2$ into $S$ are $GH(S \cup \{v_0\}) = \sum_{w \in V \setminus \{v_0\}} \frac{1}{d(w, v_0)} = 8$ and $GH(S \cup \{v_2\}) = \sum_{w \in V \setminus \{v_2\}} \frac{1}{d(w, v_2)} = 83/12$, respectively. Adding $v_0$ to the group $S$ obtains a larger group harmonic score than choosing $v_2$ to $S$. This is because for each vertex $w$, we have $\frac{1}{d(w, v_0)} \geq \frac{1}{d(w, v_2)}$ according to $v_2 \leq v_0$, which is consistent with Lemma. 4. □

Equipped with the pruning technique in Lemma. 4, we develop a general framework, namely, NeiSkyGH, to search $k$ vertices with the maximum group harmonic. The pseudo-code of NeiSkyGH is outlined in Algorithm 5. The difference between BaseGH and NeiSkyGH is that the former needs to compute the marginal gains of group harmonic score for all vertices in $V \setminus S$, while the latter only computes the marginal gains for the vertices located in the neighborhood skyline (line 4). Considering a required group size $k$, the marginal gain calculation is performed $k(2*r-k+1)/2$ times in NeiSkyGH, where $r$ is the number of skyline vertices. While BaseGH computes the marginal gains for $k(2*n-k+1)/2$ vertices, thus NeiSkyGH significantly reduces the computational costs, making it is superior to BaseGH. For instance, in Fig. 1, when the group size is 3, BaseGH needs to compute the marginal gains for $15 + 14 + 13 = 42$ vertices, while NeiSkyGH only computes the marginal gains for $8 + 7 + 6 = 21$ vertices.

**Algorithm 6:** NeiSkyMC $(G)$

```
  Input: $G = (V, E)$.
  Output: The maximum clique $H$.
1 Calculate the neighborhood skyline $R$;
2 $H \leftarrow \emptyset$;
3 for $u \in V$ do
4     if $u \in R$ then
5         $\hat{H} = \{u\}$; $\hat{X} = N(u) \cap V$;
6         Find a maximum clique by branch-and-bound method with $\hat{H}, \hat{X}$;
7 return $H$;
```

### *C. Maximum clique computation*

#### *C.1 Problem formulation*

Given a graph $G = (V, E)$, a vertex subset $H \subseteq V$ is a clique if every pair of vertices of $H$ is adjacent in $G$. The size of a clique $H$, denoted by $|H|$, is measured by its number of vertices. A clique $H$ of $G$ is a maximal clique if there is no clique $\hat{H}$ satisfying $\hat{H} \supset H$. A clique $H$ of $G$ is a maximum clique if its size is the largest among all (maximal) cliques of $G$. Based on these definitions, the maximum clique computation problem is formulated as follows.

**Maximum Clique Computation Problem.** Given a graph $G$, the problem of Maximum Clique Computation (MCC) aims to compute a maximum clique in $G$.

The MCC problem is NP-hard [40] and extensive algorithms [17]–[27] are proposed for solving MCC problem which follow a branch-and-bound framework. In particular, the framework extends an initially empty clique $H$ by moving vertices from a set $X$ of candidate vertices to $H$ to find a maximum clique. Here, $X$ is the vertex set in which each vertex is adjacent to all vertices of $H$, and is initialized by the vertex set $V$ of $G$. We refer to such a simple implementation based on the branch-and-bound framework as BaseMCC.

#### *C.2 A neighborhood skyline based algorithm*

Below, we introduce an useful lemma, based on which we derive an efficient pruning technique for the MCC problem.

*Lemma 5:* Given a graph $G = (V, E)$, for vertex $u$ and vertex $v$ in $G$ with $v \leq u$, if $v$ belongs to a maximum clique $H$, then $u$ must be in $H$.

Equipped with the pruning technique in Lemma. 5, we develop a general framework, namely, NeiSkyMC, to compute a maximum clique in a graph $G$. The pseudo-code of NeiSkyMC is outlined in Algorithm 6. The difference between BaseMCC and NeiSkyMC is that, when adding the first vertex into $H$, BaseMCC needs to select all vertices in $V$ as the first vertex and perform branch-and-bound search to output a maximum clique. While the latter only invokes the branch-and-bound search for the vertices located in the neighborhood skyline (line 4). That is to say, BaseMCC and NeiSkyMC perform the branch-and-bound search for $|V|$ vertices and $|R|$ vertices, respectively. In general, the size of $R$ is significantly less than that of $V$, and thus NeiSkyMC can substantially reduce the computational costs, making it superior to BaseMCC.

#### *C.3 Extending to finding k maximum cliques*

Here we extend the neighborhood skyline pruning to the problem of finding $k$ maximum cliques. For a vertex $u$, let $MC(u)$ be the maximum clique that includes $u$. The straightforward method of finding $k$ maximum cliques is to calculate the maximum clique for each vertex and then pick the $k$ cliques with the largest size. For brevity, we call such a method BaseTopkMCC. Clearly, this method needs to calculate maximum cliques for $|V|$ vertices. Below, we introduce the solution with the neighborhood skyline pruning technique, which is based on Lemma. 6.

*Lemma 6:* Given a graph $G = (V, E)$, for vertex $u$ and vertex $v$ in $G$ with $v \leq u$, $|MC(v)| \leq |MC(u)|$ holds.

Armed with the pruning technique in Lemma. 6, we propose an efficient algorithm, namely, NeiSkyTopkMCC, to compute $k$ cliques with the largest size in $G$. Specifically, in the first round, NeiSkyTopkMCC performs the NeiSkyMC algorithm to calculate a maximum clique. For the following $k - 1$ rounds, NeiSkyTopkMCC only computes maximum cliques for vertices in the neighborhood skyline, and selects the clique with the largest size as the answer of the current round. This is because if $v \leq u$ holds, we have $|MC(v)| \leq |MC(u)|$. Thus, in the same round, we do not need to calculate the maximum clique containing $v$. Note that in one round we pick a maximum clique which is calculated in $u$'s ego-network, i.e., $u$ is clearly a skyline vertex, we need to update the neighborhood skyline. This is because those vertices that are dominated by $u$ also have the potential to generate a maximum clique with the largest size in the next round. We omit the pseudo-code of NeiSkyTopkMCC due to the space limit.

### D. Discussions

In Sec. IV-A and Sec. IV-B, we propose the general greedy frameworks equipped with the neighborhood skyline based pruning technique to solve the GCM and GHM problems. Note that our pruning rules are orthogonal to all greedy algorithms for solving the GCM problem [11]–[13], [39] and the GHM problem [13], which iteratively select the vertex with the maximum marginal gain of group centrality. In the experiments, we will take Greedy++ [12] and Greedy-H [13] as examples to show the practical performance of our technique for the GCM and GHM problems.

Additionally, we can see that the key of Lemma. 3 and Lemma. 4 is to derive the inequalities, i.e., $d(w, S \cup \{u\}) \leq d(w, S \cup \{v\})$ and $d(v, S \cup \{u\}) = d(u, S \cup \{v\})$. Based on the domination order $v \leq u$, if the shortest path passes through $v$ from $w$ to $S$, an alternative path including $u$ with the same length must be obtained and not vice versa, thus these inequalities hold. In general, for the group centrality measures based on the shortest-path distance, the two inequalities always hold. Hence, our neighborhood skyline based pruning technique can also be used to handle the other related group centrality maximization problems, such as the group betweenness maximization. We leave this problem as an interesting future work.

In Sec. IV-C, we present a general framework with neighborhood skyline based pruning technique to speed up the MCC problem. Also, our pruning technique is orthogonal to all algorithms for solving the MCC problem [17]–[27]. In the experiments, we will take the state-of-the-art algorithm MC-BRB proposed by Chang [27] as an example to show the practical performance of our technique for the MCC problem. As many cohesive subgraph models are defined based on the neighborhood, our neighborhood skyline pruning technique is expected to speed up more of the maximum cohesive subgraph search problems, we leave this problem for future work.

## V. Experiments

### A. Experimental setup

In this section, we conduct extensive experiments to evaluate the efficiency and effectiveness of the proposed algorithms. We implement the baseline algorithm to compute the neighborhood skyline, namely, BaseSky (Algorithm 1). We also implement the filter-refine framework, i.e., FilterRefineSky (Algorithm 3), to solve the neighborhood skyline search problem. For comparison, we implement two neighborhood skyline computation algorithms, i.e., Base2Hop and BaseCSet, as baselines. The main idea of Base2Hop is to calculate all 2-hop neighbors for each vertex and then identify neighborhood skyline using the pruning technique and bloom filter technique in FilterRefineSky. And the BaseCSet algorithm first invokes FilterPhase (Algorithm 2) to compute the neighborhood candidate set $C$ for pruning and then performs BaseSky (Algorithm 1) for the vertices in $C$ instead of vertices in $V$ to derive the neighborhood skyline. Here the time complexity of BaseCSet is $O(d_{max} \sum_{u \in C} deg(u))$. As the neighborhood skyline search problem can be generalized as the set containment join problem, we also compare the proposed algorithm with the state-of-the-art set containment join algorithm: LC-Join [32]. In addition, we apply the neighborhood skyline technique to speed up the calculations of finding groups with the maximum closeness and harmonic measures and the computation of a maximum clique to show the effectiveness of our neighborhood skyline. To the best of our knowledge, the state-of-the-art algorithms for group closeness maximization, group harmonic maximization and maximum clique computation problems are Greedy++ proposed in [12], Greedy-H developed in [13] and MC-BRB presented in [27], respectively. Therefore, we use these three algorithms: Greedy++, Greedy-H and MC-BRB, as baselines for comparison. We implement the improved versions of the three algorithms with the neighborhood skyline pruning technique, i.e., NeiSkyGC, NeiSkyGH and NeiSkyMC, respectively. Note that our neighborhood skyline based pruning technique is orthogonal to all greedy algorithms for addressing the GCM problem and the GHM problem and all algorithms for the MCC problem as discussed in Sec. IV-D. We also implement the BaseTopkMCC and NeiSkyTopkMCC algorithms for finding $k$ maximum cliques. All algorithms are implemented in C++. All experiments are conducted on a PC with 3.3GHz CPU and 128GB memory running Ubuntu 20.04.1. In all experiments, the graph is resident in the main memory.

**Datasets.** We use 5 different types of real-life networks in the experiments, including web networks, social networks, communication networks and collaboration networks. The detailed statistics of the datasets are summarized in Table I. In Table I, $d_{\max}$ denotes the maximum degree of the graph. The dataset Flixster is downloaded from http://konect.cc, and the others are downloaded from snap.stanford.edu. In our experiments, we treat all datasets as undirected graphs.

**Parameters.** In the algorithms of group centrality maximization search, i.e., Greedy++, Greedy-H, NeiSkyGC and NeiSkyGH, the parameter $k$ is chosen from the set $\{50, 100, 150, 200, 250, 300\}$ with a default value of $k = 200$. We will study the performance of these four algorithms with varying $k$.

**Remark.** In FilterRefineSky, we only use one hash function based on bit-wise operations to construct bloom filter struc-

TABLE I
DATASETS

| Dataset | $n$ | $m$ | $d_{\max}$ | Desecription |
|---|---|---|---|---|
| Notredame | 325,731 | 1,090,109 | 10,721 | Web network |
| Youtube | 1,134,890 | 2,987,624 | 28,754 | Social network |
| WikiTalk | 2,394,385 | 4,659,565 | 100,029 | Communication network |
| Flixster | 2,523,386 | 7,918,801 | 1,474 | Social network |
| DBLP | 1,843,617 | 8,350,260 | 2,213 | Collaboration network |

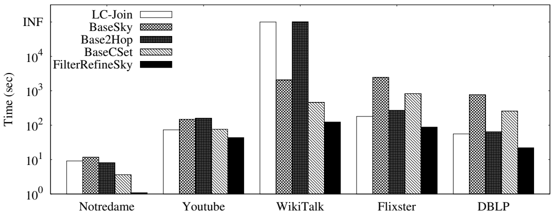


Fig. 5. Runtime of various neighborhood skyline computation algorithms

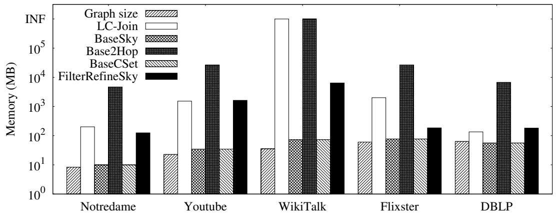


Fig. 6. Memory usages of various neighborhood skyline computation algorithms

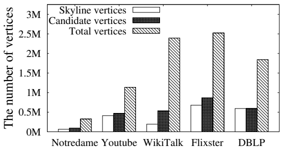
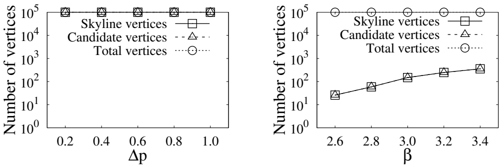


Fig. 7. Comparisons of the sizes of $R$, $C$ and $V$ on real-life graphs



(a) The ER graph (vary $\Delta p$) (b) The PL graph (vary $\beta$)

Fig. 8. Comparisons of the sizes of $R$, $C$ and $V$ on synthetic graphs

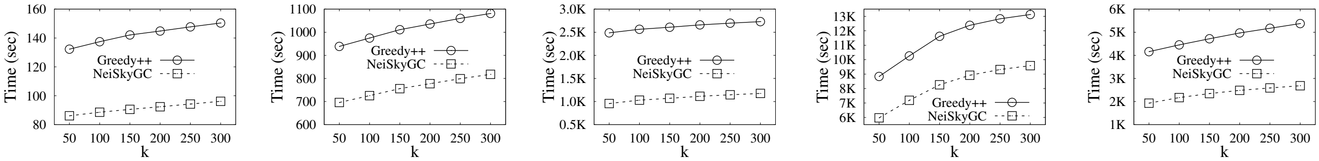
tures. The hash function is given as $BF_{h(v)>>5\%BK}(u)| = 1 << (h(v)\&31)$ where $BK$ is the number of bytes determined by $d_{max}$, which is originally used in [2]. As set containment query can be answered by bit-wise operations, such a hash function can be computed very fast.

### B. *Efficiency testing*

**Exp-1: Runtime of neighborhood skyline search algorithms.** Fig. 5 reports the runtime of LC-Join, BaseSky, Base2Hop, BaseCSet and FilterRefineSky algorithms on different datasets. Note that both LC-Join and Base2Hop algorithms are out of memory on WikiTalk, thus we denote their runtime as "INF". As expected, our FilterRefineSky algorithm achieves the lowest runtime among all algorithms over all datasets, which benefits from the powerful pruning technique and the bloom filter technique. In general, the running time of FilterRefineSky is 1.6-8.4 times and 4-35 times faster than that of LC-Join and BaseSky on all datasets, respectively. We can also see that the runtime of Base2Hop and BaseCSet is between that of BaseSky and FilterRefineSky. This is because the BaseCSet algorithm is equipped with the pruning technique based on BaseSky, and Base2Hop needs to calculate all 2-hop neighbors for each vertex first which causes additional running time. For example, on Notredame, FilterRefineSky takes 1 second to output the neighborhood skyline, while LC-Join and BaseSky consume 9 seconds and 11 seconds, respectively. And Base2Hop and BaseCSet take 8 seconds and 3.4 seconds to calculate the neighborhood skyline, respectively. Clearly, the runtime of FilterRefineSky is almost 8 times and 11 times faster than that of LC-Join and BaseSky. On DBLP, FilterRefineSky takes 22 seconds to calculate the neighborhood skyline, while LC-Join and BaseSky consume 56 seconds and 771 seconds respectively to obtain the results, which is at least 3 times and 29 times slower than FilterRefineSky. These results demonstrate that our filter-refine framework FilterRefineSky is substantially faster than the other baseline algorithms for neighborhood skyline computation on real-life graphs.

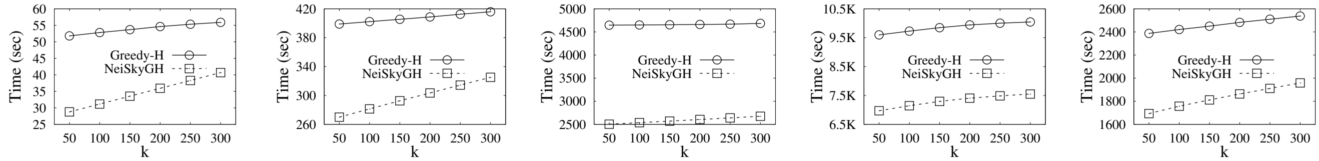
**Exp-2: Memory usages of neighborhood skyline search algorithms.** The memory costs of LC-Join, BaseSky, Base2Hop, BaseCSet and FilterRefineSky algorithms are shown in Fig. 6. Both LC-Join and Base2Hop are out of memory on WikiTalk, thus we denote their memory usages as "INF". As can be seen from Fig. 6, the Base2Hop algorithm occupies the maximum memory among all algorithms over all datasets as expected because it needs to maintain not only all 2-hop neighbors but also the bloom filter data structures for each vertex. The memory overheads of BaseSky and BaseCSet are slightly larger than the graph size on most datasets and LC-Join's memory usage is higher than the graph size over all datasets. This is because the former two algorithms only maintain several linear data structures, while LC-Join needs to construct an inverted index on data set $S$ and a prefix tree on query set $Q$ and the size of $Q$ is almost the same as that of $S$ in our neighborhood skyline search problem. Also, we can see that the memory usage of FilterRefineSky is higher than that of BaseSky, but it is still not very large especially for the graph with a relatively small $d_{\max}$. This is because FilterRefineSky needs to maintain the bloom filter structures for all skyline vertices which takes $|C|d_{\max}$. Since $|C|$ is often not very large (see Exp-3), the space overhead of FilterRefineSky is not very high on real-world graphs. Compared with LC-Join, FilterRefineSky uses less memory than LC-Join on most datasets, but achieves lower runtime of computing neighborhood skyline (see Exp-1). For instance, on Notredame, the original graph size is 8MB. The memory usages of BaseSky and BaseCSet on Notredame are both 10MB, while LC-Join and FilterRefineSky occupy 200MB and 120MB memory respectively. And the Base2Hop's memory overhead is 4,665MB which is the maximum among all algorithms. In general, FilterRefineSky uses less than 6GB memory to compute the neighborhood skyline over all datasets, which is acceptable for a modern computer. These results confirm our space complexity analysis in Sec. III.

**Exp-3: The size of the neighborhood skyline.** In this experiment, we compare the number of skyline vertices with the graph vertices $n$ and the candidate vertices on all datasets and the result is illustrated in Fig. 7. From Fig. 7, we can see that both the number of skyline vertices and candidate vertices in all datasets is significantly less than the number of graph



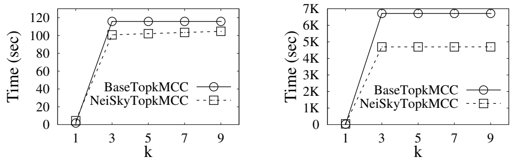
(a) Notredame (vary $k$) (b) Youtube (vary $k$) (c) WikiTalk (vary $k$) (d) Flixster (vary $k$) (e) DBLP (vary $k$)

Fig. 9. Comparisons of Greedy++ and NeiSkyGC on various datasets for group closeness maximization



(a) Notredame (vary $k$) (b) Youtube (vary $k$) (c) WikiTalk (vary $k$) (d) Flixster (vary $k$) (e) DBLP (vary $k$)

Fig. 10. Comparisons of Greedy-H and NeiSkyGH on various datasets for group harmonic maximization



(a) Pokec (vary $k$) (b) Orkut (vary $k$)

Fig. 11. Comparisons of BaseTopkMCC and NeiSkyTopkMCC

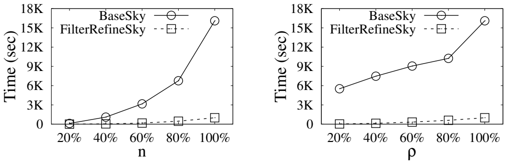
vertices. Also, there is a significant gap between the number of skyline vertices and that of candidate vertices. For example, on WikiTalk, the number of skyline vertices equals 194,629, while the number of candidate vertices and original vertices is 531,773 and 2,394,385, which is at least 2.7 times and 12 times larger than that of skyline vertices. On Flixster, the sizes of the neighborhood skyline, neighborhood candidates, and vertices are 679,201, 869,306 and 2,523,386, respectively.

In addition, we also evaluate the size of the neighborhood skyline on synthetic graphs. We generate five Erdos-Renyi (ER) random graphs and five Power-Law (PL) random graphs by Networkit (https://networkit.github.io/). In ER model, we fix the number of vertices as $1 * 10^5$ and generate random graphs by varying the expected probability of an edge $p = \Delta p * log(n)/n$. To generate PL graphs, we also set the number of vertices as $1 * 10^5$ and vary the growth exponent $\beta$. The results are illustrated in Fig. 8. As can be seen, both the number of skyline vertices and candidate vertices in ER graphs is very close to that of graph vertices with varying probability $p$. While for PL graphs, both the sizes of neighborhood skyline and candidate set are substantially less than that of vertex set for different growth exponent $\beta$. The results again confirm that the number of neighborhood skyline is less than that of vertices on real-life graphs because real-life graphs are usually power-law graphs where the degree distribution follows a power-law distribution [35]. If a graph is closer to an ER graph, then the size of neighborhood skyline may not be significantly smaller than that of vertex set. Meanwhile, these results also suggest that the neighborhood skyline technique can be very effective for pruning for some downstream graph analysis applications on real-life graphs (e.g., the group centrality maximization problems and the maximum clique search problem).

**Exp-4: The application of group closeness maximization.** Here we evaluate the Greedy++ and NeiSkyGC with varying parameter $k$. Fig. 9 shows the runtime of Greedy++ and NeiSkyGC on different datasets. As expected, the runtime of both Greedy++ and NeiSkyGC increases as $k$ increases for each dataset. This is because for a larger $k$, both the algorithms need to perform more iterations to form a $k$-size group, thus increasing the computational costs. From Fig. 9, we can also observe that on all datasets, the running time of NeiSkyGC is around 1.35-2.5 times faster than that of Greedy++ within almost all parameter settings. For example, when $k = 50$ on WikiTalk, NeiSkyGC takes 957 seconds to output all the $k$ vertices with the maximum group closeness, while Greedy++ consumes 2,487 seconds to yield the group, which is around 2.5 times slower than that of NeiSkyGC. On the DBLP dataset, the runtime of Greedy++ and NeiSkyGC takes 4,159 seconds and 1,930 seconds to output a 50-size group, respectively. These results indicate that the neighborhood skyline based pruning technique indeed can avoid calculating the marginal gains for the unpromising vertices during the search procedure, thus significantly speeding up the algorithms for the group closeness maximization problem. These results are also consistent with our analysis in Sec. IV-A.

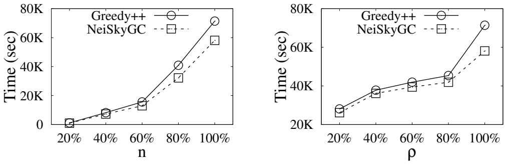
**Exp-5: The application of group harmonic maximization.** We evaluate the running time of Greedy-H and NeiSkyGH with varying parameter $k$ on different datasets. The results are depicted in Fig. 10. As can be seen, the runtime of both Greedy-H and NeiSkyGH increases with the increasing $k$ for each dataset because they need to explore more vertices to form a group for a larger $k$. In Fig. 10, the running time of NeiSkyGH is around 1.4-1.85 times faster than that of Greedy-H within almost all parameter settings overall datasets as expected. For instance, on WikiTalk, NeiSkyGH takes 2,503 seconds to output a 100-size group with the maximum group harmonic measure, while Greedy-H consumes 4,646 seconds which is 1.85 times slower than that of NeiSkyGH. These results confirm that the proposed neighborhood skyline technique can be used to reduce the number of marginal gain calculations in the group harmonic maximization search, which is consistent with our analysis in Sec. IV-B.

**Exp-6: The application of maximum clique computation.** We evaluate the BaseTopkMCC and NeiSkyTopkMCC algorithms with varying parameter $k$ on two large graphs: Pokec and Orkut. The result is depicted in Fig. 11, where the runtime includes the time of calculating the neighborhood skyline. Note that in the case of $k = 1$, the BaseTopkMCC and NeiSkyTopkMCC algorithms degenerate to MC-BRB and



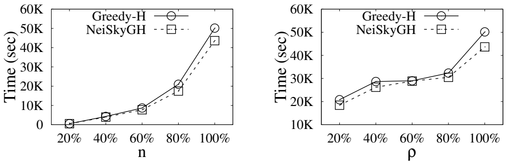
(a) LiveJournal (vary $n$) (b) LiveJournal (vary $\rho$)

Fig. 12. Scalability of BaseSky and FilterRefineSky



(a) LiveJournal (vary $n$) (b) LiveJournal (vary $\rho$)

Fig. 13. Scalability of Greedy++ and NeiSkyGC



(a) LiveJournal (vary $n$) (b) LiveJournal (vary $\rho$)

Fig. 14. Scalability of Greedy-H and NeiSkyGH

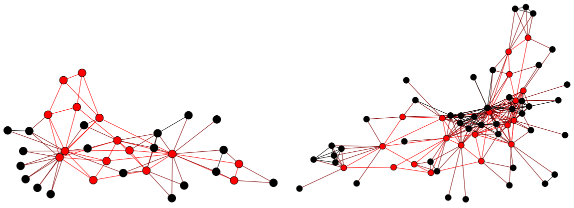
NeiSkyMC, respectively. From Fig. 11, we can see that the runtime of BaseTopkMCC and NeiSkyTopkMCC increases as $k$ increases as expected. Moreover, NeiSkyTopkMCC is slightly slower than BaseTopkMCC at $k = 1$, and when $k \geq 2$, the runtime of NeiSkyTopkMCC is less than that of BaseTopkMCC. This is because when $k = 1$, NeiSkyTopkMCC needs to calculate the neighborhood skyline first, while BaseTopkMCC can output the maximum clique in $G$ directly. For a larger $k$, except for the first round, NeiSkyTopkMCC only computes maximum cliques for vertices in the neighborhood skyline and maintains the skyline vertices. Compared to BaseTopkMCC, the pruning benefits of NeiSkyTopkMCC can dominate the costs for calculating and maintaining the neighborhood skyline. For example, when $k = 5$, NeiSkyTopkMCC takes 4696 seconds to calculate $k$ maximum cliques in Orkut, while BaseTopkMCC consumes 6717 seconds which is clearly slower than NeiSkyTopkMCC. These results show that compared to the baseline method, NeiSkyTopkMCC has no obvious advantage in finding one maximum clique, but it is more efficient to find top-$k$ maximum cliques.

**Exp-7: Scalability testing.** Here we evaluate the scalability of the proposed algorithms. To this end, we generate four subgraphs for each dataset by varying the number of vertices and density of the original graph to conduct experiments. We evaluate the runtime of BaseSky and FilterRefineSky on these generated subgraphs. Fig. 12 shows the results on a large graph LiveJournal and the results on the other datasets are similar. As expected, the runtime of FilterRefineSky increases very smoothly with increasing $n$ or $\rho$, while the runtime of BaseSky increases more sharply. Again, we can see that FilterRefineSky is significantly faster than BaseSky with all parameter settings, which is consistent with our previous findings.

We also evaluate the scalability of group centrality maximization algorithms and maximum clique search algorithm. The results on LiveJournal of group closeness maximization algorithms, i.e., Greedy++ and NeiSkyGC, are illustrated in Fig. 13; the results on LiveJournal of group harmonic maximization algorithms, Greedy-H and NeiSkyGH, are shown in Fig. 14; and Table II reports the running time of MC-BRB and NeiSkyMC algorithms on LiveJournal for maximum clique computation problem. Similar results can also be observed on the other datasets. From Fig. 13, we can see that the runtime of NeiSkyGC increases relatively smoothly with increasing $n$ or $\rho$ compared with that of Greedy++. Also, NeiSkyGC is significantly better than Greedy++ with all parameter settings as expected, which is consistent with our previous experiments. For the Greedy-H and NeiSkyGH algorithms, the runtime of NeiSkyGH also changes smoothly compared to that of Greedy-H when varying $n$ or $\rho$. Furthermore, consistent with our previous findings, NeiSkyGH is superior to Greedy-H under all experimental settings. From Table II, we can see that both the running time of NeiSkyMC and MC-BRB increases as $n$ increases because more vertices would lead in more search branches in the two branch-and-bound based algorithms. When varying density $\rho$, the running time of NeiSkyMC and MC-BRB do not increase with increasing $\rho$. This is because MC-BRB (NeiSkyMC) algorithm is equipped with many powerful pruning techniques and non-trivial heuristic search methods, thus can prune a lot of branches for a graph with larger density and reduce the running time. Consistent with our previous finding, NeiSkyMC is faster than MC-BRB over all datasets with varying $n$ or $\rho$. For example, when $\rho$ equals 80%, NeiSkyMC takes 4590.932 seconds to compute a maximum clique, while MC-BRB consumes 4871.153 seconds. These results demonstrate the high scalability of our group centrality maximization algorithms and maximum clique computation algorithm equipped with the neighborhood skyline pruning.

TABLE II
SCALABILITY OF MC-BRB AND NeiSkyMC ON LiveJournal

| vary $n$ | MC-BRB($\mu m$) | NeiSkyMC ($\mu m$) | vary $\rho$ | MC-BRB($\mu m$) | NeiSkyMC ($\mu m$) |
|---|---|---|---|---|---|
| 20% | 34,664 | **33,473** | 20% | 976,505 | **953,605** |
| 40% | 148,730 | **137,087** | 40% | 3,022,167 | **289,9143** |
| 60% | 404,485 | **401,014** | 60% | 13,178,465 | **12,469,524** |
| 80% | 627,427 | **617,582** | 80% | 4,871,153,877 | **4,590,932,416** |
| 100% | 1,063,380 | **1,055,273** | 100% | 1,063,380 | **1,055,273** |



(a) Karate (b) TrainBombing

Fig. 15. Case studies on Karate and TrainBombing (red vertex: the vertex in the neighborhood skyline set)

## C. Case study

Here we conduct case studies on two small datasets, Karate and TrainBombing, to evaluate the effectiveness of our neighbor skyline. Karate (34 nodes and 78 edges) is the well-known and much-used Zachary karate club network, and TrainBombing (64 nodes and 243 edges) contains contacts between suspected terrorists involved in the train bombing of Madrid on 2004. Both Karate and TrainBombing can be downloaded from https://konect.cc/networks. We invoke our FilterRefineSky algorithm to find the neighborhood skyline of Karate and TrainBombing. The results are depicted in Fig. 15 in which the vertices colored red are the vertices belonging to the neighborhood skyline. From Fig. 15, we can see that there are 15 nodes (44%) and 20 nodes (31%) in the neighborhood skyline sets of Karate and TrainBombing, respectively. Clearly, the size of neighborhood skyline is significantly less than the number of vertices for both of the graphs. Moreover, the nodes with smaller degrees are more easily dominated by other vertices. The degree distribution of real-life networks usually follows a power-law distribution; that is, there are many low-degree vertices and a few high-degree vertices. Hence, the size of neighborhood skyline in real-life networks is generally not very large, which is consistent with our previous findings.

## VI. Related work

**Set containment query.** Our work is closely related to the set containment query. Given a query record $Q$ and a dataset $S$, a set containment query of $Q$ over $S$ is to find all records in $S$ that are contained by $Q$. Many research investigates the problem of set containment query [41]–[44]. Helmer *et al.* studied the performance of different index structures and confirmed that the inverted list can show the best overall performance of all index structures for set-valued attributes of low cardinality [41]. Terrovitis *et al.* developed an OIF index that combines the inverted index with B-tree and proposed index-based query processing algorithms to address the queries for subset, set-equality and superset queries [42]. Zhu *et al.* presented an LSH ensemble index structure and a query algorithm which copes with the data volume and skew by means of data sketches using minwise hashing and domain partitioning [43]. Yang *et al.* investigated the problem of selectivity estimation on set containment search and proposed two sampling-based techniques, i.e., OT-Sampling and DCSampling, to track the challenges of set containment search [44]. In addition, as an important relationship, other studies based on set containment have also attracted much attention in recent years, such as set containment join [28]–[33] and set similarity join [45]–[49]. Based on set containment, the neighborhood skyline search problem is a subset of the above problems since the containment relations only exist between a vertex and its 2-hop reachable neighbors rather than all vertex pairs. To the best of our knowledge, the definition of neighborhood skyline is proposed for the first time, and also our work is the first to study the problem of finding skylines vertices in graphs.

**Skyline computation.** The skyline computation was originally investigated in computational theory. Kung *et al.* studied the problem of exploring the skyline for $d$-dimensional vectors, and developed an algorithm with $O(n \log n)$ time complexity for $d = 2, 3$ [50]. In the database field, Borzsonyi *et al.* first integrated the skyline operation into a database system [51]. After the seminal work, skyline computation has attracted much attention, and a large number of studies have been done to find the skyline with different definitions [52]–[57]. Recently, Liu *et al.* designed a novel structure, skyline diagram, which enables precomputation of three different skyline queries including quadrant, global, and dynamic skyline [55]. Li *et al.* proposed an $O(s(m+n))$ algorithm to find the skyline communities in a multi-valued graph for 2-dimension case. They also developed an efficient space-partition algorithm to find the skyline communities for the high-dimensional case [56]. Li *et al.* studied the skyline cohesive group query problem on road-social networks and presented a novel skyline group model with respect to the social and spatial cohesiveness [57]. In this work, the defined neighborhood skyline is fundamentally different from previous skyline concepts, thus all existing algorithms cannot be used to solve our neighborhood skyline search problem. Moreover, most of these existing solutions for skyline problems are based on divide-and-conquer, while we propose a filter-and-refine framework to address our problem which is totally different from the previous algorithms.

**Group centrality maximization.** The group centrality maximization problem aims to find the group of $k$ vertices with the maximum group centrality measure, which is typically NP-hard [11]. In particular, our work is closely related to the problems of group closeness maximization (a.k.a GCM) and group harmonic maximization (a.k.a GHM). A lot of algorithms have been devised to solve the GCM problem recently [11]–[13], [39]. Chen *et al.* proposed a greedy algorithm with approximation ratio $(1 - 1/e)$. Considering the expensive time complexity, they further developed an alternative heuristic sampling algorithm [11]. Bergamin *et al.* designed novel techniques to speed up the greedy algorithm without losing theoretical, which can handle the networks with tens or hundreds of millions of edges. Angriman *et al.* introduced new heuristics algorithms for group closeness maximization in large networks [39]. And then they provided approximation hardness results as well as approximation guarantees for the greedy algorithm [13]. In addition, Zhao *et al.* investigated the external-memory algorithm to solve the GCM problem [58]. For the GHM problem, Angriman *et al.*, for the first time, studied this problem and proved that the GHM problem on undirected graphs can be addressed by a constant-factor approximation algorithm [13]. To the best of our knowledge, we are the first to apply the neighborhood skyline pruning to speed up the GCM and GHM problems. In particular, our pruning technique is orthogonal to all greedy algorithms for solving the GCM and GHM problems. Moreover, the pruning derived by our neighborhood skyline is suitable for all group centrality maximization problems in which the group centrality measure is based on the shortest-path distance.

**Maximum clique computation.** Our work is also closely related to the maximum clique computation problem, which is NP-hard [40]. Extensive algorithms are proposed for maximum clique computation problem including exact algorithms [17]–[27] and heuristic algorithms [21], [22], [27]. All the exact solutions are based on the branch-and-bound method and equipped with various bounding techniques to prune a branch if its upper bound is not better than the currently found largest clique. For heuristic algorithms, maximum degree-based heuristic [21] and degeneracy order-based heuristic [22] are presented which greedily move to $C$ the vertex of $R$ that, has the maximum degree and has the highest rank according to the degeneracy ordering. The state-of-the-art algorithm for maximum clique computation is MC-BRB proposed by Chang [27], which transforms an instance of MCC-Sparse to instances of $k$-clique finding over dense subgraphs (KCF-Dense) that can be computed by the existing MCC-Dense methods. He also developed a new branch reduce-and-bound framework for KCF-Dense and an ego-centric algorithm MC-EGO for heuristically computing a near-maximum clique in near-linear time. In this work, we apply the neighborhood skyline pruning to speed up the MC-BRB algorithm, which can be considered as a new state-of-the-art algorithm. Also, our pruning technique is orthogonal to all branch-and-bound algorithms for solving the maximum clique computation problem.

## VII. Conclusion

In this paper, we propose a concept called neighborhood skyline, and study a novel problem of neighborhood skyline search on graphs. To solve this problem, we develop a filter-refine search framework, FilterRefineSky, which can efficiently calculate the neighborhood skyline in a graph. To show the effectiveness of our technique, we study two applications, i.e., group closeness maximization and group harmonic maximization, and prove that our neighborhood skyline can be used to speed up the computation of the group centrality maximization problems, in which the centrality measure is based on shortest-path distance. We also show that the neighborhood skyline technique can accelerate the maximum clique computation. We conduct extensive experiments using five real-life datasets to evaluate the proposed algorithms. The results demonstrate the efficiency, effectiveness, and scalability of our algorithms.

## References

[1] T. Akiba, Y. Iwata, and Y. Yoshida, "Fast exact shortest-path distance queries on large networks by pruned landmark labeling," in *Proceedings of SIGMOD*, pp. 349–360, ACM, 2013.

[2] H. Wei, J. X. Yu, C. Lu, and R. Jin, "Reachability querying: An independent permutation labeling approach," *Proceedings of VLDB Endow.*, vol. 7, no. 12, pp. 1191–1202, 2014.

[3] G. Palla, I. Derényi, I. Farkas, and T. Vicsek, "Uncovering the overlapping community structure of complex networks in nature and society," *nature*, vol. 435, no. 7043, pp. 814–818, 2005.

[4] L. Chang, W. Li, and W. Zhang, "Computing A near-maximum independent set in linear time by reducing-peeling," in *Proceedings of SIGMOD*, pp. 1181–1196, ACM, 2017.

[5] C. Piao, W. Zheng, Y. Rong, and H. Cheng, "Maximizing the reduction ability for near-maximum independent set computation," *Proceedings of VLDB Endow.*, vol. 13, no. 12, pp. 2466–2478, 2020.

[6] W. Li, M. Qiao, L. Qin, Y. Zhang, L. Chang, and X. Lin, "Scaling distance labeling on small-world networks," in *Proceedings of SIGMOD*, pp. 1060–1077, ACM, 2019.

[7] U. Brandes, M. Heine, J. Müller, and M. Ortmann, "Positional dominance: Concepts and algorithms," in *Proceedings of CALDAM*, vol. 10156, pp. 60–71, Springer, 2017.

[8] N. V. Mahadev and U. N. Peled, *Threshold graphs and related topics*. Elsevier, 1995.

[9] S. D. Nikolopoulos and C. Papadopoulos, "The number of spanning trees in k n-complements of quasi-threshold graphs," *Graphs and Combinatorics*, vol. 20, no. 3, pp. 383–397, 2004.

[10] M. Fürer, "Efficient computation of the characteristic polynomial of a threshold graph," *Theor. Comput. Sci.*, vol. 657, pp. 3–10, 2017.

[11] C. Chen, W. Wang, and X. Wang, "Efficient maximum closeness centrality group identification," in *Proceedings of Australasian Database Conference*, pp. 43–55, Springer, 2016.

[12] E. Bergamini, T. Gonser, and H. Meyerhenke, "Scaling up group closeness maximization," in *Proceedings of ALENEX*, pp. 209–222, SIAM, 2018.

[13] E. Angriman, R. Becker, G. D'Angelo, H. Gilbert, A. van der Grinten, and H. Meyerhenke, "Group-harmonic and group-closeness maximization–approximation and engineering," in *Proceedings of ALENEX*, pp. 154–168, SIAM, 2021.

[14] S. Dolev, Y. Elovici, R. Puzis, and P. Zilberman, "Incremental deployment of network monitors based on group betweenness centrality," *Inf. Process. Lett.*, vol. 109, no. 20, pp. 1172–1176, 2009.

[15] M. H. Chehreghani, A. Bifet, and T. Abdessalem, "An in-depth comparison of group betweenness centrality estimation algorithms," in *IEEE Big Data*, 2018.

[16] D. Dinler and M. K. Tural, "Faster computation of successive bounds on the group betweenness centrality," *Networks*, vol. 71, no. 4, pp. 358–380, 2018.

[17] R. Carraghan and P. M. Pardalos, "An exact algorithm for the maximum clique problem," *Operations Research Letters*, vol. 9, no. 6, pp. 375–382, 1990.

[18] C.-M. Li, Z. Fang, and K. Xu, "Combining maxsat reasoning and incremental upper bound for the maximum clique problem," in *2013 IEEE 25th International Conference on Tools with Artificial Intelligence*, pp. 939–946, IEEE, 2013.

[19] C.-M. Li, H. Jiang, and F. Manyà, "On minimization of the number of branches in branch-and-bound algorithms for the maximum clique problem," *Computers & Operations Research*, vol. 84, pp. 1–15, 2017.

[20] P. M. Pardalos and J. Xue, "The maximum clique problem," *Journal of global Optimization*, vol. 4, no. 3, pp. 301–328, 1994.

[21] B. Pattabiraman, M. Patwary, M. Ali, A. H. Gebremedhin, W.-k. Liao, and A. Choudhary, "Fast algorithms for the maximum clique problem on massive sparse graphs," in *International Workshop on Algorithms and Models for the Web-Graph*, pp. 156–169, Springer, 2013.

[22] R. A. Rossi, D. F. Gleich, and A. H. Gebremedhin, "Parallel maximum clique algorithms with applications to network analysis," *SIAM Journal on Scientific Computing*, vol. 37, no. 5, pp. C589–C616, 2015.

[23] P. San Segundo, A. Lopez, and P. M. Pardalos, "A new exact maximum clique algorithm for large and massive sparse graphs," *Computers & Operations Research*, vol. 66, pp. 81–94, 2016.

[24] E. Tomita, "Efficient algorithms for finding maximum and maximal cliques and their applications," in *International workshop on algorithms and computation*, pp. 3–15, Springer, 2017.

[25] E. Tomita, Y. Sutani, T. Higashi, S. Takahashi, and M. Wakatsuki, "A simple and faster branch-and-bound algorithm for finding a maximum clique," in *International Workshop on Algorithms and Computation*, pp. 191–203, Springer, 2010.

[26] J. Xiang, C. Guo, and A. Aboulnaga, "Scalable maximum clique computation using mapreduce," in *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pp. 74–85, IEEE, 2013.

[27] L. Chang, "Efficient maximum clique computation over large sparse graphs," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*, pp. 529–538, ACM, 2019.

[28] J. Yang, W. Zhang, S. Yang, Y. Zhang, and X. Lin, "Tt-join: Efficient set containment join," in *Proceedings of ICDE*, pp. 509–520, IEEE, 2017.

[29] J. Yang, W. Zhang, S. Yang, Y. Zhang, X. Lin, and L. Yuan, "Efficient set containment join," *VLDB Journal.*, vol. 27, no. 4, pp. 471–495, 2018.

[30] Y. Luo, G. H. Fletcher, J. Hidders, and P. De Bra, "Efficient and scalable trie-based algorithms for computing set containment relations," in *Proceedings of ICDE*, pp. 303–314, IEEE, 2015.

[31] A. Kunkel, A. Rheinländer, C. Schiefer, S. Helmer, P. Bouros, and U. Leser, "Piejoin: towards parallel set containment joins," in *Proceedings of SSDBM*, pp. 1–12, 2016.

[32] D. Deng, C. Yang, S. Shang, F. Zhu, L. Liu, and L. Shao, "Lcjoin: Set containment join via list crosscutting," in *Proceedings of ICDE*, pp. 362–373, IEEE, 2019.

[33] C. Yang, D. Deng, S. Shang, F. Zhu, L. Liu, and L. Shao, "Internal and external memory set containment join," *VLDB Journal.*, vol. 30, no. 3, pp. 447–470, 2021.

[34] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[35] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *science*, vol. 286, no. 5439, pp. 509–512, 1999.

[36] M. G. Everett and S. P. Borgatti, "The centrality of groups and classes," *The Journal of mathematical sociology*, vol. 23, no. 3, pp. 181–201, 1999.

[37] A. Clark, L. Bushnell, and R. Poovendran, "A supermodular optimization framework for leader selection under link noise in linear multi-agent systems," *IEEE TAC*, vol. 59, no. 2, pp. 283–296, 2013.

[38] S. Banerjee, M. Jenamani, and D. K. Pratihar, "A survey on influence maximization in a social network," *Knowledge and Information Systems*, vol. 62, no. 9, pp. 3417–3455, 2020.

[39] E. Angriman, A. van der Grinten, and H. Meyerhenke, "Local search for group closeness maximization on big graphs," in *IEEE Big Data*, pp. 711–720, IEEE, 2019.

[40] R. M. Karp, "Reducibility among combinatorial problems," in *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pp. 85–103, Plenum Press, New York, 1972.

[41] S. Helmer and G. Moerkotte, "A performance study of four index structures for set-valued attributes of low cardinality," *VLDB Journal.*, vol. 12, no. 3, pp. 244–261, 2003.

[42] M. Terrovitis, P. Bouros, P. Vassiliadis, T. K. Sellis, and N. Mamoulis, "Efficient answering of set containment queries for skewed item distributions," in *Proceedings of EDBT*, pp. 225–236, ACM, 2011.

[43] E. Zhu, F. Nargesian, K. Q. Pu, and R. J. Miller, "LSH ensemble: Internet-scale domain search," *Proceedings of VLDB Endow.*, vol. 9, no. 12, pp. 1185–1196, 2016.

[44] Y. Yang, W. Zhang, Y. Zhang, X. Lin, and L. Wang, "Selectivity estimation on set containment search," *Data Sci. Eng.*, vol. 4, no. 3, pp. 254–268, 2019.

[45] J. Wang, G. Li, and J. Feng, "Can we beat the prefix filtering?: an adaptive framework for similarity join and search," in *Proceedings of SIGMOD*, pp. 85–96, ACM, 2012.

[46] D. Deng, G. Li, H. Wen, and J. Feng, "An efficient partition based method for exact set similarity joins," *Proceedings of VLDB Endow.*, vol. 9, no. 4, pp. 360–371, 2015.

[47] W. Mann, N. Augsten, and P. Bouros, "An empirical evaluation of set similarity join techniques," *Proceedings of VLDB Endow.*, vol. 9, no. 9, pp. 636–647, 2016.

[48] X. Wang, L. Qin, X. Lin, Y. Zhang, and L. Chang, "Leveraging set relations in exact set similarity join," *Proceedings of VLDB Endow.*, vol. 10, no. 9, pp. 925–936, 2017.

[49] X. Wang, L. Qin, X. Lin, Y. Zhang, and L. Chang, "Leveraging set relations in exact and dynamic set similarity join," *VLDB Journal.*, vol. 28, no. 2, pp. 267–292, 2019.

[50] H.-T. Kung, F. Luccio, and F. P. Preparata, "On finding the maxima of a set of vectors," *Journal of the ACM (JACM)*, vol. 22, no. 4, pp. 469–476, 1975.

[51] S. Borzsony, D. Kossmann, and K. Stocker, "The skyline operator," in *Proceedings of ICDE*, pp. 421–430, IEEE, 2001.

[52] J. Pei, B. Jiang, X. Lin, and Y. Yuan, "Probabilistic skylines on uncertain data," in *Proceedings of VLDB Endow.*, pp. 15–26, Citeseer, 2007.

[53] C. Sheng and Y. Tao, "On finding skylines in external memory," in *Proceedings of PODS*, pp. 107–116, 2011.

[54] A. Asudeh, S. Thirumuruganathan, N. Zhang, and G. Das, "Discovering the skyline of web databases," *arXiv preprint arXiv:1512.02138*, 2015.

[55] J. Liu, J. Yang, L. Xiong, J. Pei, and J. Luo, "Skyline diagram: Finding the voronoi counterpart for skyline queries," in *Proceedings of ICDE*, pp. 653–664, IEEE Computer Society, 2018.

[56] R. Li, L. Qin, F. Ye, G. Wang, J. X. Yu, X. Xiao, N. Xiao, and Z. Zheng, "Finding skyline communities in multi-valued networks," *VLDB Journal.*, vol. 29, no. 6, pp. 1407–1432, 2020.

[57] Q. Li, Y. Zhu, and J. X. Yu, "Skyline cohesive group queries in large road-social networks," in *Proceedings of ICDE*, pp. 397–408, IEEE, 2020.

[58] J. Zhao, J. C. S. Lui, D. Towsley, and X. Guan, "Measuring and maximizing group closeness centrality over disk-resident graphs," in *Proceedings of WWW*, pp. 689–694, ACM, 2014.