Skyline Community Search in Multi-valued Networks

Rong-Hua Li[†], Lu Qin[‡], Fanghua Ye[#], Jeffrey Xu Yu[*], Xiaokui Xiao[§], Nong Xiao[#], Zibin Zheng[#]

[†] *Shenzhen University, China;* [‡]*University of Technology, Australia;* [#]*Sun Yat-Sen University, China*
[*]*The Chinese University of Hong Kong, Hong Kong;* [§]*Nanyang Technological University, Singapore*
{rhli,yu}@se.cuhk.edu.hk; Lu.Qin@uts.edu.au; smartyfh@outlook.com;
xkxiao@ntu.edu.sg; xiao-n@vip.sina.com; zibin.gil@gmail.com

ABSTRACT

Community search is a problem of finding cohesive subgraphs that meet the query conditions in a network. In this work, we study a community search problem in a general type of network, called a multi-valued network, where each node in the network is associated with d ($d \geq 1$) numerical attributes. These numerical attributes can typically be obtained from the profiles of the nodes or the statistical information of the nodes computed by a variety of network analysis methods (e.g., the PageRank of the nodes). Unfortunately, existing community search algorithms either completely ignore these numerical attributes or only consider one numerical attribute of the nodes. To capture d numerical attributes, we propose a novel community model, called skyline community, based on the concepts of k-core and skyline. A skyline community is a maximal connected k-core that cannot be dominated by the other connected k-cores in the d-dimensional attribute space. However, finding all the skyline communities is a very challenging problem, because the search space can be exponentially large. To overcome this issue, we propose a basic algorithm and an elegant space-partition algorithm to find the skyline communities efficiently. The striking feature of the space-partition algorithm is that its time complexity relies mainly on the size of the answer s (i.e., the number of skyline communities), thus it is very efficient if s is small. Moreover, the space-partition algorithm can progressively output the skyline communities, which is very useful for applications that only require part of the skyline communities. We conduct extensive experiments in several real-world large networks, and the results demonstrate the efficiency and effectiveness of our algorithms.

1. INTRODUCTION

Many real-world networks such as social networks consist of community structures. Discovering the communities from a network is a fundamental problem in network analysis. Recently, a query-dependent community discovery problem called community search has attracted much attention in the database community due to a large number of applications [22, 12, 10, 15, 13, 11]. The goal of the community search problem is to find those densely-connected subgraphs in a network that satisfy the query conditions.

In many real-world applications, the nodes in a network are often associated with a set of attributes (including numerical and textual attributes) which may be very important in characterizing and understanding the community structure [15, 11]. For example, Li et al. [15] studied a community search problem in an attributed graph in which each node has a numerical attribute called influence. Based on this numerical attribute, they proposed an influential community model to characterize the influence of the community. More recently, Fang et al. [11] investigated a community search problem in an attributed graph in which each node is associated with a set of textual attributes. They proposed an attribute community model to capture both the structural and attribute cohesiveness of the community.

In this paper, we focus on a particular type of node attribute in the network: the numerical attributes. Instead of considering only one numerical attribute as studied in [15], we study a more general case, where every node in the network has d ($d \geq 1$) different numerical attributes. These numerical attributes can typically be obtained from the profiles of the nodes or the statistical information of the nodes computed by different network analysis methods (e.g., the influence and PageRank of the node). For example, in the Aminer scientific network (aminer.org), each author is associated with several numerical attributes, including the number of published papers, h-index, activity, diversity, sociability and so on.

The community in a multi-valued network should take the d-dimensional numerical attributes into consideration. Intuitively, we may wish to find the interesting communities that are not *completely dominated* by the other communities in the d-dimension attribute space. A naive approach to this end is described as follows. First, we can compute the average value (or other linear combinations) over all d numeric attributes for each node. Then, based on the average value of each node, we apply the previous community search algorithm for one numerical attribute [15] to identify communities. This naive method, however, cannot fully capture all the interesting communities in the d-dimension attribute space. This is because a community with high average value in each dimension could also be *dominated* by the other communities (as confirmed in our experiments). To fully characterize all those interesting communities, we propose a novel community model called skyline community based on the concepts of k-core [20] and skyline [4]. A skyline community is a maximal connected k-core (not necessary the maximal k-core as defined in [20]) that is not dominated by the other connected k-cores in the d-dimensional attribute space (the detailed definition can be found in Section 2). Finding the skyline communities in a multi-valued network motivates many interesting applications, three of which are introduced as follows.

Personalized influential community search. In an online social network, a user may want to discover the influential communities with similar interests. For example, in the Facebook social network, a football-fan user would like to find the influential football-fan groups, as these groups play important roles for football information dissemination in the network. In this application, we can extract two numeric attributes for each user: the influence and similarity (i.e., the similarities between the query user and the other users in the social network). By discovering the skyline communities on these numeric attributes, we can obtain the communities that are not dominated by the others in terms of both influence and similarity. Therefore, from the skyline communities, the query user can get the desired communities that are not only influential, but also have similar interests to him.

Close social groups discovery in LBSN. The location-based social network (LBSN) is a special social network in which each user is associated with a location. To join similar and close social groups, a user in an LBSN may wish to find the social groups such that they not only have similar interests, but they are also close to him. Similar query may also help the companies to perform marketing or promotion activities. For example, the fast food company KFC may want to identify the social groups that are not only interested in KFC's food, but they are also close to the location of KFC. In these applications, we can extract two numeric attributes for each user in the LBSN: (1) the similarity between the query and the user, and (2) the distance between the query location and the user's location. By mining the skyline communities on these numeric attributes, we are able to obtain the desired social groups.

Versatile team search. Consider a team search application in the Aminer scientific network. We may want to find a team so that its members not only have a number of publications, but also they are very active in their research areas in recent years. Such teams can be detected by mining the skyline communities from the Aminer network using two numeric attributes: the number of publications and activity. Similarly, by finding the skyline communities with three attributes including activity, the number of publications, and diversity, we are able to identify the teams whose members are active in their research areas, and also they have numerous publications and diverse research interests.

Technical challenge and our contributions. Finding the skyline communities is a challenging task. The challenge we face is fundamentally different from the traditional skyline discovery problem. This is because in our problem, the communities are not given. Moreover, the number of potential communities (i.e., the number of connected k-cores) can be exponentially large in a multi-valued graph, thus it is very difficult to determine which community is located on the skyline.

To overcome this challenge, we first propose an efficient algorithm, called SkylineComm2D, to find all the skyline communities in the 2D case, i.e., $d = 2$. The time complexity of the SkylineComm2D algorithm is $O(s(m+n))$ where s denotes the number of 2D skyline communities (i.e., the answer size), and the space complexity of the SkylineComm2D algorithm is $O(m+n+s)$, which is linear w.r.t. the graph and answer size. To handle the high-dimensional case (i.e., $d \geq 3$), we first propose a basic algorithm based on an interesting connection between our community model and the influential community model [15]. To improve the efficiency, we develop an elegant algorithm based on a space-partition idea. The space-partition algorithm first prunes the dominated space by the computed skyline communities and then recursively partitions the remaining space into several *regular* subspaces, in which the skyline communities can be computed by the $(d-1)$-dimensional skyline community search algorithm (more details can be found in Section 5). The striking feature of the space-partition algorithm is that its worst-case time complexity is dependent mainly on the answer size, thus it is very efficient when the answer size is not very large. Moreover, for $d = 3$, the space-partition algorithm achieves the same time and space complexity as the SkylineComm2D algorithm. Compared to the basic algorithm, the space-partition algorithm is a progressive algorithm which progressively outputs the skyline communities during the execution of the algorithm. This progressive feature is very useful for applications that only require part of the skyline communities. We conduct extensive experiments over several real-world large networks to evaluate the proposed algorithms. The results show that the space-partition algorithm is at least one order of magnitude faster than the basic algorithm, and uses much less memory. In addition, we conduct comprehensive case studies to evaluate the effectiveness of the proposed skyline community model. The results show that many interesting and meaningful communities can be discovered using our model that cannot be found by other methods.

Organization. We propose the skyline community model and formulate the skyline community search problem in Section 2. We develop an efficient algorithm to find all the 2D skyline communities in Section 3. The basic algorithm and the space-partition algorithm for the $d \geq 3$ case are proposed in Section 4 and Section 5 respectively. Extensive experiments are reported in Section 6. Finally, we review the related work and conclude this paper in Section 7 and Section 8 respectively.

2. PROBLEM STATEMENT

We model a graph with d numerical attributes as a multi-valued graph $G = (V, E, X)$, where V ($|V| = n$) and E ($|E| = m$) denote the set of nodes and edges respectively, and X ($|X| = n$) is a set of d-dimensional vectors. In a multi-valued graph, each node $v \in V$ is associated with a d-dimensional real-valued vector denoted by $X_v = (x_1^v, \cdots, x_d^v)$, where $X_v \in X$ and $x_i^v \in \mathbb{R}$. For convenience, we refer to the i-th dimension ($i = 1, \cdots, d$) as the x_i dimension. In addition, we assume without loss of generality that on the x_i dimension, x_i^v for all $v \in V$ form a strict total order, i.e., $x_i^v \neq x_i^u$ for any $u \neq v$. It is important to note that if this assumption does not hold, we can easily construct a strict total order by using the node identity to break ties for any $x_i^v = x_i^u$. The d-dimensional vector X_v represents the values of the node v w.r.t. d different numerical attributes.

Based on the multi-valued graph model, we study the community search problem in a network with numerical attributes. To model the structural cohesiveness, we use the widely-used k-core model to represent the communities [20, 3, 22, 10, 15]. Specifically, denote by $\delta(v, G)$ the degree of node v in the multi-valued graph G. Let $H = (V_H, E_H)$ be an induced subgraph of G, i.e., $V_H \subseteq V$ and $E_H = \{(u,v) | u, v \in V_H, (u,v) \in E\}$. A k-core H is an induced subgraph where each node $v \in H$ has a degree at least k, i.e., $\delta(v, H) \geq k$. The maximal k-core $\tilde{H}$ is a k-core such that there is no super k-core containing $\tilde{H}$. For each node $v \in V$, the core number of v is the maximal k such that a k-core contains v. Note that the maximal k-core is not necessarily connected. To avoid confusion, we refer to a connected k-core as a k-$\widehat{core}$.

Clearly, the traditional k-core model cannot capture the d-dimensional numerical attributes of a community. Li et al. [15] recently proposed an influential community model which can capture the influence of a community. Each node in their model is associated with an influence value, and the influence of a community is defined as the minimum influence value among all the values of its members. In the context of multi-valued graph, the influential community model only works for the $d = 1$ case, because it considers the one-dimensional (1D) numerical attribute of a community. In this paper, we focus on the community search problem for the d-dimensional case ($d \geq 1$), where each node is associated with d real values. Below, we first give a novel definition to characterize a community for the d-dimensional case ($d > 1$).

The skyline community model. Note that it is nontrivial to generalize the existing community model for the 1D case (i.e., the influential community model) [15] to the d-dimensional case ($d > 1$). The definition in [15] of the influential community model is based on the comparison of the influence values of the communities. However, unlike the 1D case, it is not easy to compare two communities because each community may have d ($d > 1$) values w.r.t. d different dimensions. As a result, the influential community model cannot be directly extended to the d-

dimensional case ($d > 1$). To overcome this issue, we introduce the *domination* relationship between two communities, which will be used to define our skyline community model.

Let $H = (V_H, E_H)$ be an induced subgraph of G. Following the definition of the influence value of a community in [15], we define the value of H on the x_i dimension (for $i = 1, 2, \cdots, d$) as

$$f_i(H) \triangleq \min_{v \in V_H}\{x_i^v\}. \quad (1)$$

Below, we briefly discuss why we use the "min" operator in Eq. (1) to define $f_i(H)$. The motivation of this definition is the same as that of the influential community model [15]. By using the "min" operator, we can ensure that all the members in H on the x_i dimension have a value no less than $f_i(H)$. That is to say, if $f_i(H)$ is large, each node in H must have a large value on the x_i dimension. This is very useful for excluding *outliers* in H (i.e., the nodes with small values on the x_i dimension). For example, assume that the x_i dimension denotes the PageRank values of the nodes and we want to find an *important* community whose members have high PageRank values. Clearly, we can use the above definition of $f_i(H)$ to capture the *importance* of the community.

DEFINITION 1. *Let $H = (V_H, E_H)$ and $H' = (V_{H'}, E_{H'})$ be two communities. If $f_i(H) \leq f_i(H')$ for all $i = 1, \cdots, d$, and there exists $f_i(H) < f_i(H')$ for a certain i, we call that H' dominates H, denoted by $H \prec H'$.*

Intuitively, an interesting community in a multi-valued graph G should be a cohesive subgraph which also cannot be dominated by other communities. For example, in the Aminer scientific network, assume that we consider two numerical attributes: activity and diversity. In this example, we may want to find a community that is not dominated by other communities in both activity and diversity.

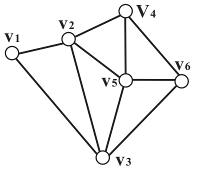
Based on this intuition, we use the concepts of k-core [20] and skyline [4] to define a new community model in the multi-valued graph, called skyline community. To the best of our knowledge, we are the first to use the concept of skyline for community modeling. In our model, we make use of k-core to represent the cohesive property of a community, as it is successfully used for community search applications [22, 10, 15, 11].

DEFINITION 2. *Given a multi-valued graph $G = (V, E, X)$ and an integer k. A skyline community with a parameter k is an induced subgraph $H = (V_H, E_H, X_H)$ of G such that it satisfies the following properties.*

- *Cohesive property: H is a k-$\widehat{core}$ (i.e., H is a connected k-core);*
- *Skyline property: there does not exist an induced subgraph H' of G such that H' is a k-$\widehat{core}$ and $H \prec H'$;*
- *Maximal property: there does not exist an induced subgraph H' of G such that (1) H' is a k-$\widehat{core}$, (2) H' contains H, and (3) $f_i(H') = f_i(H)$ for all $i = 1, \cdots, d$.*

Note that without the maximal property, there could be a large number of skyline communities with the same f values on the d dimensions. The maximal property in Definition 2 ensures that a skyline community is not contained in a larger skyline community with the same f values on the d dimensions, and therefore avoid redundancy. It is worth noting that the k-$\widehat{core}$ in our definition is not necessarily the maximal k-core as defined in [20, 16], and the number of k-$\widehat{cores}$ could be exponentially large. The following example illustrates the definition of the skyline community.

EXAMPLE 1. *Consider the graph shown in Fig. 1. The left panel is a graph with 6 nodes, and the right panel shows the values of these nodes in three different dimensions. Suppose for instance that $k = 2$. Then, by Definition 2, $H_1 = \{v_1, v_2, v_3\}$ is a skyline community with values $f(H_1) = (8, 14, 3)$, because there does not exist a 2-$\widehat{core}$ that can dominate it, and it is also the maximal subgraph that satisfies the cohesive and skyline properties. Similarly, $H_2 = \{v_2, v_4, v_5, v_6\}$ is a skyline community with $f(H_2) = (6, 8, 4)$. The subgraph $H_3 = \{v_4, v_5, v_6\}$ is not a skyline community, because it is contained in $H_2 = \{v_2, v_4, v_5, v_6\}$ which has the same f values as H_3. The subgraph $H_4 = \{v_2, v_3, v_4, v_5, v_6\}$ is also not a skyline community, as $f(H_4) = (6, 8, 3)$ is dominated by H_1 and H_2.*



	x1	x2	x3
v1	8	14	7
v2	9	15	12
v3	11	17	3
v4	6	16	4
v5	7	8	5
v6	10	9	10

Figure 1: Running example

Discussions. Apart from the "min" operator used in Eq. (1), another two possible operators may be "max" and "sum". These two operators, however, are not appropriate for skyline community modeling. The reason is that unlike the "min" operator, these two operators are *monotonic* w.r.t. the community size, i.e., a community has a larger f value than its sub-communities. As a result, the answers are always the set of maximal k-$\widehat{cores}$ of the original multi-valued graph, which are independent of the numerical attributes of the nodes in the graph.

The skyline community search problem. Given a multi-valued graph $G = (V, E, X)$ and an integer k, the problem is to find all the skyline communities from G with the parameter k.

Note that if $d = 1$, there is only one skyline community by Definition 2. Moreover, we can easily show that when $d = 1$, the skyline community search problem is equivalent to the problem of finding the top-1 influential community [15]. Therefore, if $d = 1$, we can use the algorithms proposed in [15] to find the skyline community efficiently. However, when $d > 1$, the problem becomes much harder and the algorithms proposed in [15] cannot be used. Below, we discuss the challenges of our problem.

Challenges. The challenges to solve our problem are twofold. First, the number of k-$\widehat{cores}$ (i.e., connected k-cores) in a multi-valued network can be exponentially large. Thus, it is intractable to enumerate all the k-$\widehat{cores}$ to identify the skyline communities. Second, unlike the traditional d-dimensional skyline computation problem [4], the *d-dimensional data points* in our problem, which correspond to the k-$\widehat{cores}$, are not given. As a result, it is challenging to devise an efficient algorithm to detect the skyline communities without enumerating all the k-$\widehat{cores}$. In the following sections, we will develop several efficient algorithms to overcome these challenges.

3. ALGORITHM FOR $d = 2$

In this section, we propose an efficient algorithm to find all skyline communities in the 2D case (i.e., $d = 2$). The algorithm for the 2D case will be used as the building-block when we process the $d > 2$ case. In the rest of this paper, we assume without loss of generality that the values of the nodes on all d dimensions are positive (i.e., $x_i^u > 0$ for all $u \in V$ and $i = 1, \cdots, d$). Recall that in the 2D case each skyline community H has two values $(f_1(H), f_2(H))$. If $H = \emptyset$, we define $f_i(H) = 0$ for $i = 1, 2$. For each skyline community H, we mainly focus on devising an algorithm to compute $(f_1(H), f_2(H))$, because we can easily extract the community from G based on these two values.

The basic idea of our algorithm is as follows. First, we only consider the x_2 dimension in graph G, and compute the maximal f_2 value, denoted by f_2^*, among all the k-$\widehat{cores}$. We find a maximal k-$\widehat{core}$ (denoted by $\tilde{H}$) which achieves f_2^* by recursively deleting the node with the smallest x_2 value until the graph contains no k-core. Note that the maximal k-$\widehat{core}$ $\tilde{H}$ may not be a skyline community. This is because $\tilde{H}$ could be dominated by a community H which has the same f_2 value, but a larger f_1 value than $\tilde{H}$. However, such a community H must be contained in $\tilde{H}$, because it has the same f_2 value as $\tilde{H}$, which is maximal over all the k-$\widehat{cores}$. Therefore, to find a skyline community, we can apply the same procedure to compute the maximal f_1 value, denoted by f_1^*, over all the connected sub-k-cores contained in $\tilde{H}$. The resulting k-$\widehat{core}$ denoted by H_1 must be a skyline community with values $(f_1(H_1), f_2(H_1))$, where $f_1(H_1) = f_1^*$ and $f_2(H_1) = f_2^*$. This is because f_2^* is maximal among all the k-$\widehat{cores}$, thus no other k-$\widehat{core}$ that can dominate it on the x_2 dimension. On the other hand, f_1^* is maximal over all the k-$\widehat{cores}$ with the same f_2^* value, thus no k-$\widehat{core}$ exists that can dominate it. Since the above recursive procedure ensures that the resulting k-$\widehat{core}$ is maximal, it must be a skyline community.

Using the above method, we can find one skyline community which has the maximal f_2 value of all the skyline communities. The challenge is how to find the other skyline communities. We can tackle this challenge based on the following result. All the proofs in this paper can be found in the Appendix.

LEMMA 1. *Let H_1 with values $(f_1(H_1), f_2(H_1))$ be the skyline community that has the maximal f_2 value over all the skyline communities. The nodes in G whose x_1 values are no larger than $f_1(H_1)$ cannot then be contained in the other skyline communities.*

Based on Lemma 1, we can shrink the graph G by removing all the nodes whose x_1 values are no larger than $f_1(H_1)$. We invoke the above procedure in the reduced graph to find the next skyline community H_2. It should be noted that H_2 must be different from H_1, because its f_1 value is larger than $f_1(H_1)$. We can iteratively invoke this procedure to find all the skyline communities until the reduced graph contains no k-core.

Algorithm 2 implements the above procedure. In Algorithm 2, $\mathcal{I}$ denotes the set of constraints. Initially, $\mathcal{I} = \{x_1 > 0, x_2 > 0\}$, which means that no constraint is active (because $x_i^u > 0$ for all $u \in V$ and $i = 1, 2$ by our assumption). $\mathcal{F}$ denotes the set of fixed nodes. For the 2D case, there is no need to fix nodes, thus $\mathcal{F} = \emptyset$. However, for the $d > 2$ case, we will use the set $\mathcal{F}$ to maintain the fixed nodes (see Sections 4 and 5), which cannot be deleted by the algorithm. To find all the 2D skyline communities, we can invoke $\mathsf{SkylineComm2D}(G, \{x_1 > 0, x_2 > 0\}, \emptyset)$. The detailed algorithm is described as follows.

First, Algorithm 2 invokes Algorithm 1 to calculate the maximal f_2 over all the skyline communities (line 1). Specifically, Algorithm 1 first deletes all the invalid nodes (i.e., shrinks the graph, line 1 in Algorithm 1), and then computes the maximal k-core H (line 2 in Algorithm 1). The algorithm then recursively deletes the node with the smallest x_2 value until $H = \emptyset$ (lines 6-12 in Algorithm 1). The algorithm returns the maximal f_2 value over all the k-$\widehat{cores}$ subject to the constraints $\mathcal{I}$.

After determining f_2, Algorithm 2 iteratively computes the skyline communities in lines 2-5. In line 3, Algorithm 2 first refines $\mathcal{I}$ by the constraint $x_2 \geq f_2$. Here we use a notation $\bar{\cap}$ to denote the *refinement* operator. In particular, if $\mathcal{I} = \{x_1 > 0, x_2 > 0\}$, then $\tilde{\mathcal{I}} = \mathcal{I} \bar{\cap} \{x_2 \geq f_2\} = \{x_1 > 0, x_2 \geq f_2\}$, because the constraint $x_2 > 0$ in $\mathcal{I}$ is refined by $x_2 \geq f_2$. Then, Algorithm 2 calls Algorithm 1 with the refined constraints $\tilde{\mathcal{I}}$ to calculate the maximal f_1 value (line 3). It should be noted that in Algorithm 1, the constraint $x_2 \geq f_2$ ensures that all the nodes with x_2 values smaller than f_2 are deleted. Therefore, the obtained f_1 value in line 3 (Algorithm 2) is the maximal f_1 value over all the k-$\widehat{cores}$ with the same f_2 value. By definition, there is a skyline community that has values (f_1, f_2). In line 4, the algorithm adds (f_1, f_2) into the answer set. Subsequently, in line 5, the algorithm refines the constraint by $(x_1 > f_1)$, because nodes with x_1 values no larger than f_1 cannot be included in the undiscovered skyline communities (see Lemma 1). Then, the algorithm calculates the maximal f_2 value subject to the refined constraints $\tilde{\mathcal{I}}$. After obtaining f_2, the algorithm iteratively applies the same procedure to compute the next skyline community. The algorithm terminates when $f_2 = 0$, which means that no k-core exist that satisfies the refined constraints. The correctness of Algorithm 2 is shown in the following theorem.

Algorithm 1 $\mathsf{DimMax}(G, \mathcal{I}, \mathcal{F}, d)$

```
Input:   A multi-valued graph $G$, constraints $\mathcal{I}$, fixed nodes set $\mathcal{F}$, $d$.
Output:  The maximal value on the $d$-th dimension.
1: $G \leftarrow$ delete all the nodes in $G$ that violate the constraints $\mathcal{I}$;
2: $H \leftarrow$ compute the maximal $k$-core in $G$;
3: if $\mathcal{F} \neq \emptyset$ then
4:     $H \leftarrow$ the maximal $k$-$\widehat{core}$ in $H$ that contains $\mathcal{F}$;
5: Compute $f_d(H)$ based on Eq. (1);
6: $f_d \leftarrow f_d(H)$; $visit(u) \leftarrow 0$ for all $u \in H$;
7: while $H \neq \emptyset$ do
8:     Let $u \in H$ be the smallest-value node on the $x_d$ dimension;
9:     $flag \leftarrow 1$; $flag \leftarrow$ DFS($u$);
10:    if $flag = 0$ then break;
11:    if $\mathcal{F} \neq \emptyset$ then
12:        $H \leftarrow$ the maximal $k$-$\widehat{core}$ in $H$ that contains $\mathcal{F}$;
13:    $f_d \leftarrow \max\{f_d, f_d(H)\}$;
14: return $f_d$;
15: Procedure DFS ($u$)
16: if $u \in \mathcal{F}$ then return 0; {// the fixed node cannot be deleted}
17: $visit(u) \leftarrow 1$;
18: Let $N(u, H)$ be the neighborhood of $u$ in $H$;
19: Let $\delta(u, H)$ be the degree of $u$ in $H$;
20: for all $v \in N(u, H)$ and $visit(v) = 0$ do
21:     $\delta(v, H) \leftarrow \delta(v, H) - 1$;
22:     if $\delta(v, H) < k$ then DFS($v$);
23: Delete $u$ from $H$;
24: return 1;
```

Algorithm 2 $\mathsf{SkylineComm2D}(G, \mathcal{I}, \mathcal{F})$

```
Input:   A multi-valued graph $G$, constraints $\mathcal{I}$, fixed nodes set $\mathcal{F}$.
Output:  Skyline Communities in $G$.
1: $f_2 \leftarrow \mathsf{DimMax}(G, \mathcal{I}, \mathcal{F}, 2)$; $\mathcal{R} \leftarrow \emptyset$;
2: while $f_2 > 0$ do
3:     $\tilde{\mathcal{I}} \leftarrow \mathcal{I} \bar{\cap} \{x_2 \geq f_2\}$; $f_1 \leftarrow \mathsf{DimMax}(G, \tilde{\mathcal{I}}, \mathcal{F}, 1)$;
4:     $\mathcal{R} \leftarrow \mathcal{R} \cup \{(f_1, f_2)\}$;
5:     $\tilde{\mathcal{I}} \leftarrow \mathcal{I} \bar{\cap} \{x_1 > f_1\}$; $f_2 \leftarrow \mathsf{DimMax}(G, \tilde{\mathcal{I}}, \mathcal{F}, 2)$;
6: return $\mathcal{R}$;
```

THEOREM 1. *Algorithm 2 correctly computes all the 2D skyline communities.*

We analyze the time and space complexity of Algorithm 2 in the following theorem.

THEOREM 2. *Let s be the number of skyline communities in G. Then, the worst case time and space complexity of Algorithm 2 is $O(s(m + n))$ and $O(m + n + s)$ respectively.*

Note that in the 2D case, the total number of skyline communities s is bounded by n, because the number of f_2 values of the skyline communities is bounded by n. Thus, the time and space complexity of Algorithm 2 is also bounded by $O(n(m + n))$ and $O(m + n)$ respectively. In our experiments, we will show that s is typically very small, thus our algorithm is very efficient in practice.

4. THE BASIC ALGORITHM FOR $d \geq 3$

Recall that Algorithm 2 can iteratively compute all the 2D skyline communities once it has found the *first* skyline community. To find the *first* skyline community, Algorithm 2 computes the maximal f_2 value, and applies a similar procedure to determine the f_1 value. Unfortunately, this idea does not work in the case of $d \geq 3$. This is because for the $d \geq 3$ case, we do not know how to determine the remaining values (f_1 and f_2) of a skyline community after computing the maximal f_3 value. Furthermore, even if we can find the *first* skyline community for the $d \geq 3$ case, it is still quite nontrivial to find all the remaining skyline communities. Below, we develop a basic algorithm to tackle these challenges based on an in-depth analysis of the skyline community model. For convenience, we first devise a basic algorithm to handle the 3D case (i.e., $d = 3$), and then we extend this algorithm to handle the $d > 3$ case.

Algorithm 3 [15]$\mathsf{InfComm}(G, d)$

```
Input:  A multi-valued graph $G$, $d$.
Output: All the $f_d$ values.
1:  $H \leftarrow$ the maximal $k$-core of $G$; $T_d \leftarrow \emptyset$;
2:  while $H \neq \emptyset$ do
3:      Let $\tilde{H}$ be the maximal connected component of $H$ with smallest $f_d$ value, denoted by $f_d^*$;
4:      $T_d \leftarrow T_d \cup \{f_d^*\}$; Let $u \in \tilde{H}$ be the node that $x_d^u = f_d^*$;
5:      InfCommDFS($u$);
6:  return $T_d$;
7:  Procedure InfCommDFS($u$)
8:  for all $v \in N(u, H)$ do
9:      Delete edge $(u, v)$ from $H$;
10:     if $\delta(v, H) < k$ then InfCommDFS($v$);
11: Delete node $u$ from $H$;
```

4.1 Handling the $d = 3$ case

The key idea of the basic algorithm is that we first compute all the possible f_3 values that the skyline communities may have. Second, for each possible f_3 value, denoted by f_3^*, we remove all the nodes whose x_3 values are smaller than f_3^* and fix the node u with $x_3^u = f_3^*$. A fixed node denotes that the node cannot be deleted by the algorithm. Note that since we assume that all the x_3 values form a total order, there is only one node u with $x_3^u = f_3^*$ that can be fixed. We invoke Algorithm 2 with constraint $\mathcal{I} = \{x_3 \geq f_3^*\}$ and fixed-point set $\mathcal{F} = \{u\}$ to compute the 2D skyline communities based on the first two dimensions (i.e., the x_1 and x_2 dimensions). Here we refer to a skyline community based on the first two dimensions as a 2D skyline community, and all those based on three dimensions as a 3D skyline community. Following this procedure, we obtain a set of 2D skyline communities denoted by $\mathcal{T}$. Clearly, each community $H \in \mathcal{T}$ must contain u (because u is fixed), and also $f_3(H) = f_3^*$. It is important to note that $H \in \mathcal{T}$ is not necessarily a valid 3D skyline community because it is computed by using the first two dimensions (not all three dimensions). However, it is easy to show that the 3D skyline communities whose f_3 values equal f_3^* must be contained in $\mathcal{T}$. Finally, we merge all the resulting 2D skyline communities for every possible f_3 value, and invoke a traditional skyline algorithm [14, 4] to determine all the 3D skyline communities.

Let F_3 be the set of all the possible f_3 values. The remaining question is how to determine F_3. A naive solution is to set F_3 to be the set of all the x_3 values of the nodes in G. Clearly, this naive solution is correct, because the f_3 values of all the skyline communities must take from the set of all the x_3 values of nodes. However, this naive solution is inefficient, because we need to invoke the $\mathsf{SkylineComm2D}$ algorithm $|F_3| = n$ times. Below, we propose a much more efficient solution to determine F_3.

Our solution is based on an interesting connection between our problem and the influential community search problem [15]. We only consider one dimension x_i ($i = 1, 2, 3$), and then compute all the influential communities [15] based on the x_i dimension. Let T_i be the set of values of all those influential communities. We find that the f_i values of the skyline communities must be taken from T_i (see Lemma 2). As a result, we can set $F_3 = T_3$ in our algorithm. Since $|T_3| \leq n$ and can be substantially smaller than n in practice [15], our solution is much more efficient than the naive solution.

For completeness, we describe the influential community search algorithm [15] to compute T_d based on the x_d dimension in Algorithm 3. In particular, the algorithm iteratively deletes the smallest-value node on the x_d dimension and records the f_d values of the current maximal k-$\widehat{core}$, which corresponds to the value of an influential community [15].

Based on F_3, we can enumerate every $f_3 \in F_3$, and invoke $\mathsf{SkylineComm2D}$ to compute all the 2D skyline communities whose values on the x_3 dimension are equal to f_3. Our algorithm is depicted in Algorithm 4. In line 1, we compute F_3 by invoking Algorithm 3 based on the x_3 dimension. In lines 2-7, we calculate the 2D skyline communities for each $f_3 \in F_3$. The algorithm first fixes the node u that $x_3^u = f_3$ (line 3), because the node u must be contained in all the 2D skyline communities whose values on the x_3 dimension are equal to f_3. In line 4, the algorithm refines the constraint $\mathcal{I}$ by $\{x_3 \geq f_3\}$ which indicates that the nodes whose x_3 values are smaller than f_3 will be removed. The algorithm then invokes Algorithm 2 to compute the 2D skyline communities based on the x_1 and x_2 dimensions (line 5). Lastly, the algorithm combines the results (lines 6-7), and applies a traditional skyline algorithm to determine all the 3D skyline communities (line 8). The following example illustrates how Algorithm 4 works.

Algorithm 4 $\mathsf{Basic3D}(G, \mathcal{I}, \mathcal{F})$

```
Input:  A multi-valued graph $G$, constraints $\mathcal{I}$, fixed nodes set $\mathcal{F}$.
Output: Skyline Communities in $G$.
1: $F_3 \leftarrow \mathsf{InfComm}(G, 3)$; $\mathcal{R} \leftarrow \emptyset$;
2: for all $f_3 \in F_3$ do
3:     Let $u$ be the node that $x_3^u = f_3$; $\tilde{\mathcal{F}} \leftarrow \mathcal{F} \cup \{u\}$;
4:     $\tilde{\mathcal{I}} \leftarrow \mathcal{I} \tilde{\cap} \{x_3 \geq f_3\}$;
5:     $\mathcal{T} \leftarrow \mathsf{SkylineComm2D}(G, \tilde{\mathcal{I}}, \tilde{\mathcal{F}})$; {// Compute skyline communities based on the first two dimensions.}
6:     for all $(f_1, f_2) \in \mathcal{T}$ do
7:         $\mathcal{R} \leftarrow \mathcal{R} \cup \{f_1, f_2, f_3\}$;
8: return $\mathsf{Skyline}(\mathcal{R})$;
```

Algorithm 5 $\mathsf{BasicHighD}(G, \mathcal{I}, \mathcal{F}, d)$

```
Input:  A multi-valued graph $G$, constraints $\mathcal{I}$, fixed nodes set $\mathcal{F}$.
Output: Skyline Communities in $G$.
1: if $d = 3$ then return $\mathsf{Basic3D}(G, \mathcal{I}, \mathcal{F})$;
2: $F_d \leftarrow \mathsf{InfComm}(G, d)$; $\mathcal{R} \leftarrow \emptyset$;
3: for all $f_d \in F_d$ do
4:     Let $u$ be the node with $x_d^u = f_d$; $\tilde{\mathcal{F}} \leftarrow \mathcal{F} \cup \{u\}$;
5:     $\tilde{\mathcal{I}} \leftarrow \mathcal{I} \tilde{\cap} \{x_d \geq f_d\}$;
6:     $\mathcal{T} \leftarrow \mathsf{BasicHighD}(G, \tilde{\mathcal{I}}, \tilde{\mathcal{F}}, d-1)$;
7:     for all $(f_1, \cdots, f_{d-1}) \in \mathcal{T}$ do
8:         $\mathcal{R} \leftarrow \mathcal{R} \cup \{f_1, \cdots, f_{d-1}, f_d\}$;
9: return $\mathsf{Skyline}(\mathcal{R})$;
```

EXAMPLE 2. *Consider the graph shown in Fig. 1. The algorithm first obtains $F_3 = \{3, 4\}$ by invoking Algorithm 3. Then, for $f_3 = 3$, the algorithm invokes Algorithm 2 with constraint $\mathcal{I} = \{x_3 \geq 3\}$ and $\mathcal{F} = \{v_3\}$ to compute the 2D skyline communities based on the x_1 and x_2 dimensions (lines 3-5). In this example, we can obtain only one 2D skyline community which is $\{v_1, v_2, v_3\}$ with values $(8, 14)$. The algorithm adds $(8, 14, 3)$ into the answer set $\mathcal{R}$ (lines 6-7). Similarly, for $f_3 = 4$, the algorithm also obtains one 2D skyline community which is $\{v_2, v_4, v_5, v_6\}$ with values $(6, 8)$. The algorithm adds $(6, 8, 4)$ into $\mathcal{R}$. Finally, the algorithm computes the skyline over $\mathcal{R}$ which is the set $\{(8, 14, 3), (6, 8, 4)\}$. Hence, in the graph in Fig. 1, we obtain two 3D skyline communities.*

We analyze the correctness and complexity of Algorithm 4 in Theorem 3. We first give a lemma which will be used to prove Theorem 3.

LEMMA 2. *For each dimension x_i ($i = 1, \cdots, d$), the f_i values of all the skyline communities are contained in the set T_i which is computed by Algorithm 3.*

THEOREM 3. *Algorithm 4 correctly finds all the 3D skyline communities, and the worst-case time and space complexity of Algorithm 4 is $O(n^2(m + n))$ and $O(n^2)$ respectively.*

4.2 Handling the $d > 3$ case

We generalize Algorithm 4 to handle the $d > 3$ case in Algorithm 5. The general procedure is very similar to Algorithm 4. The main difference is that the algorithm recursively invokes itself with a parameter $d-1$ to compute the $(d-1)$-dimensional skyline communities (line 6). The recursive procedure terminates when $d = 3$ (line 1), because we invoke Algorithm 4 to compute the 3D skyline communities. The correctness analysis of Algorithm 5 is also similar to that of Algorithm 4, thus we omit the details for brevity. Below, we analyze the complexity of Algorithm 5.

THEOREM 4. *For $d \geq 3$, the worst-case time and space complexity of Algorithm 5 is $O(n^{d-1}(m+n+(d-1)\log^{d-3} n))$ and $O(n^{d-1})$ respectively.*

Note that the above complexity is the worst-case complexity. In practice, since the number of skyline communities in the d-dimensional space is much smaller than $O(n^{d-1})$ and also d is very small (e.g., $d \leq 5$), our basic algorithm still works for many real-world networks as shown in the experiments.

4.3 A pruning rule

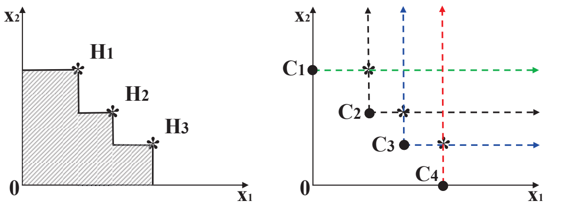
We present a simple but efficient pruning rule to speed up the basic algorithms. When we fix the node u with $x_d^u = f_d$ in both Algorithm 4 (line 3) and Algorithm 5 (line 4), we can use the d-dimensional values of node u for pruning, i.e., $X_u = \{x_1^u, \cdots, x_d^u\}$. Since all the $(d-1)$-dimensional skyline communities computed by fixing u must contain u, the values of u form an upper bound of all those skyline communities. Therefore, when fixing u, we first check whether u is dominated by the already computed skyline communities. If this is the case, we do not need to recursively invoke the algorithm to compute the $(d-1)$-dimensional skyline communities (line 5 in Algorithm 4 and line 6 in Algorithm 5), because those communities are definitely dominated by the already computed skyline communities. Using this pruning rule, we can save a number of recursive calls in the basic algorithms. To implement this pruning rule, we first sort the set F_d in a decreasing order, and then compute the skyline communities for each $f_d \in F_d$ following this order. When we fix u (line 3 in Algorithm 4 and line 4 in Algorithm 5), we check whether $(x_1^u, \cdots, x_{d-1}^u)$ is dominated by the already computed $(d-1)$-dimensional skyline communities. If this is the case, u is dominated because the f_d values of all the already computed $(d-1)$-dimensional skyline communities are larger than x_d^u, and thus there is no need to recursively invoke the algorithm to calculate the $(d-1)$-dimensional skyline communities with fixed u.

5. THE SPACE-PARTITION ALGORITHM

Although the pruning rule significantly accelerates the basic algorithm, it is still inefficient for the $d > 3$ case because it needs to compute a large number of invalid skyline communities. In this section, we propose a more efficient algorithm based on a novel space-partition idea. The worst-case time complexity of our new algorithm relies mainly on the number of skyline communities, i.e., the answer size, thus it is very efficient if the answer size is not very large. Unlike the basic algorithm, the new algorithm outputs the skyline communities progressively, and no invalid skyline community is generated. This progressive feature is very useful when the applications only need to compute part of the skyline communities. Below, we first consider the $d = 3$ case, and then we extend our algorithm to handle the $d > 3$ case.

5.1 Handling the $d = 3$ case

The key idea. The basic idea of our new algorithm is that if we find a set of valid 3D skyline communities $\mathcal{H}$, the 3D space that is dominated by $(f_1(H), f_2(H), f_3(H))$ for all $H \in \mathcal{H}$ can be pruned, because there is no skyline community in that space. After pruning the dominated space, we recursively compute the skyline communities in the remaining space. However, computing the skyline communities in the remaining space is a very challenging task, because all the computed skyline communities in the 3D space form a *staircase-like shape*, thus the remaining space is an *irregular* space. We do not know how to compute the maximal f_3 value of the skyline communities in this *irregular* space, thus the previous proposed algorithms do not work.



(a) A skyline example (b) Corner points and subspaces

Figure 2: Illustration of the space-partition idea (color online)

To overcome this challenge, we propose a novel space-partition approach to partition the remaining space into several overlapped *regular* subspaces. Here the regular subspace is defined as follows.

DEFINITION 3. *A d-dimensional space is called a regular space if and only if (1) the maximal f_d value over all the d-dimensional skyline communities in that space can be computed by the* DimMax *algorithm, and (2) $f_d > 0$.*

Note that by Definition 3, a regular space must contain a skyline community because $f_d > 0$. Initially, the 3D space can be represented by $(x_1 > 0, x_2 > 0, x_3 > 0)$ (the first quadrant of the 3D coordinate system). We compute the maximal f_3 value of the skyline communities in the initial space, which is denoted by f_3^*. If $f_3^* > 0$, we know that a skyline community exists in the initial space (otherwise there is no skyline community). Thus, in this case, the initial space is a regular space by definition. We fix the node u with $x_3^u = f_3^*$, and invoke the SkylineComm2D algorithm to compute the set of all the 2D skyline communities $\mathcal{R}$. Since all these 2D skyline communities $\mathcal{R}$ have value f_3^* on x_3, they cannot be dominated by the other k-cores in the 3D space. Therefore, they are also 3D skyline communities. Since f_3^* is maximal, the f_3 values of the remaining skyline communities in G must be smaller than f_3^*. Therefore, the (f_1, f_2) values of the remaining skyline communities cannot be dominated by those of the skyline communities in $\mathcal{R}$. For ease of understanding, we use an example to illustrate our idea. Consider the example shown in Fig. 2(a), where the label '*' denotes the skyline communities on the x_1 and x_2 dimensions. In this example, we have three skyline communities in $\mathcal{R} = \{H_1, H_2, H_3\}$. Clearly, the space below the staircase-like shape is dominated by the skyline communities in $\mathcal{R}$. The (f_1, f_2) values of the remaining skyline communities must be located on the top of the staircase-like shape. Before we proceed further, we first define two useful concepts called MIN skyline and corner point as follows.

DEFINITION 4. *Let $\mathcal{L}$ be a set of d-dimensional points. The* MIN *skyline of $\mathcal{L}$, denoted by $\mathcal{A}$, contains all the points in $\mathcal{L}$ that satisfy the following condition. For any point $x = (x_1, \cdots, x_d) \in \mathcal{A}$, there does not exist a point $y = (y_1, \cdots, y_d) \in \mathcal{L} \backslash \mathcal{A}$ such that $y_i \leq x_i$ for all $i = 1, \cdots, d$ and $y_i < x_i$ for a certain $i = 1, \cdots, d$.*

DEFINITION 5. *Let $\mathcal{R}$ be a set of skyline points in the d-dimensional space. Let $\mathcal{B}$ be the set of all the cross points in the boundary of the d-dimensional staircase-like shape formed by the skyline. The corner point set $\mathcal{C}$ is the* MIN *skyline computed over all the cross points in $\mathcal{B}$.*

Reconsider the graph shown in Fig. 2(a). There are seven cross points in the boundary of the staircase-like shape (including three skyline points). We compute the MIN skyline over all the cross points. Clearly, we can obtain four corner points which are labeled by '•' in Fig. 2(b). Note that the coordinates of the corner points can be determined by the coordinates of the skyline communities. For example, in Fig. 2(b), the coordinates of the corner point C_3 can be determined by the skyline communities H_2 and H_3, which are $(f_1(H_2), f_2(H_3))$.

Based on the corner points, we divide the remaining space into several overlapped subspaces. For each corner point C_i in $\mathcal{C}$, we obtain a subspace using the following method. First, we set C_i as the origin of the coordinate system. The subspace is the first quadrant of this coordinate system. The division approach is illustrated in Fig. 2(b). In this example, we have four corner points (i.e., $\{C_1, \cdots, C_4\}$), and thus we obtain four subspaces (the different subspaces are depicted by dashed lines in various colors). Clearly, the subspaces are overlapped, and the union of all the subspaces covers the remaining space (i.e., the space that is not dominated by $\mathcal{R}$). More importantly, each subspace can be easily characterized by the corresponding corner point. For the corner point $C_3 = (f_1(H_2), f_2(H_3))$ for example, we can represent the subspace by $(x_1 > f_1(H_2), x_2 > f_2(H_3))$. Note that the subspace does not contain the corner point, i.e., $(f_1(H_2), f_2(H_3))$, because the corner point is dominated by the skyline point in $\mathcal{R}$. To determine whether a subspace contains a skyline community, we invoke the DimMax algorithm to compute the maximal f_3 in that subspace. For the subspace corresponding to C_3, we invoke the DimMax algorithm with constraint $\mathcal{I} = (x_1 > f_1(H_2), x_2 > f_2(H_3))$ to compute the maximal f_3 value. Therefore, if $f_3 > 0$, the subspace is a regular space by Definition 3. Otherwise, we can safely prune this subspace, because there is no skyline community in that subspace.

Clearly, all the remaining skyline communities (i.e., the skyline communities with f_3 values smaller than f_3^*) must, by our space-partition method, be located in the regular subspaces. Thus, we can use a priority queue $\mathcal{Q}$ to maintain all those regular subspaces where the priority of the subspace is the maximal f_3 value in that subspace. We then make use of a best-first strategy to iteratively compute the remaining skyline communities. In each iteration, we pop the regular subspaces from $\mathcal{Q}$ that have the maximal priority denoted by $\tilde{f}_3^*$. Note that several regular subspaces may exist that have the same maximal priority. For each popped regular subspace, we fix the node u with $x_3^u = \tilde{f}_3^*$, and invoke the SkylineComm2D algorithm to compute the 2D skyline communities in that subspace. We claim that the computed 2D skyline communities are also the 3D skyline communities. The reason is as follows. On the one hand, these skyline communities cannot be dominated by the previously computed skyline communities, because they are located on the top of the staircase-like shape formed by the already computed skyline communities (based on the first two dimensions). On the other hand, the f_3 values of the undiscovered skyline communities must be smaller than $\tilde{f}_3^*$ by the best-first strategy, thus they also cannot be dominated by the undiscovered skyline communities. As a result, all the computed 2D skyline communities are valid 3D skyline communities. Once we obtain a set of valid 3D skyline communities, we can iteratively use the same method to partition the remaining space into several regular subspaces, and push the regular subspaces into $\mathcal{Q}$. The algorithm terminates when $\mathcal{Q} = \emptyset$.

Algorithm 6 New3D$(G, \mathcal{I}, \mathcal{F})$

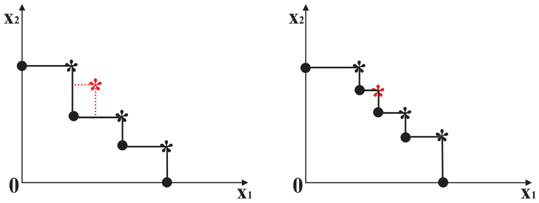
```
Input:   A multi-valued graph $G$, constraints $\mathcal{I}$, fixed nodes set $\mathcal{F}$.
Output:  Skyline Communities in $G$.
1: Result $\mathcal{R} \leftarrow \emptyset$; Priority Queue $\mathcal{Q} \leftarrow \emptyset$; $\mathcal{C} \leftarrow \{(0,0)\}$;
2: if DimMax$(G, \mathcal{I}, \mathcal{F}, 3) > 0$ then
3:     $\mathcal{Q}$.Push($(0,0)$, DimMax$(G, \mathcal{I}, \mathcal{F}, 3)$);
4: while $\mathcal{Q} \neq \emptyset$ do
5:     $c_3 \leftarrow \mathcal{Q}$.MaxVal(); $\mathcal{S}' \leftarrow \emptyset$;
6:     while $\mathcal{Q}$.MaxVal() $= c_3$ do
7:         $((c_1, c_2), c_3) \leftarrow \mathcal{Q}$.Pop(); {// $c_3$ is the priority of $(c_1, c_2) \in \mathcal{Q}$}
8:         $\tilde{\mathcal{I}} \leftarrow \mathcal{I} \bar{\cap} \{x_1 > c_1, x_2 > c_2\}$;
9:         Let $u$ be the node that $x_3^u = c_3$; $\tilde{\mathcal{F}} \leftarrow \mathcal{F} \cup \{u\}$;
10:        $\mathcal{S}_{tmp} \leftarrow$ SkylineComm2D$(G, \tilde{\mathcal{I}}, \tilde{\mathcal{F}})$;
11:        $\mathcal{S}' \leftarrow \mathcal{S}' \cup \mathcal{S}_{tmp}$;
12:    for all $(c_1, c_2) \in \mathcal{S}'$ do
13:        $\mathcal{R} \leftarrow \mathcal{R} \cup \{(c_1, c_2, c_3)\}$;
14:    for all $s' \in \mathcal{S}'$ do $\mathcal{C} \leftarrow$ UpdateCornerPoints$(\mathcal{C}, s', 2)$;
15:    for all $(c_1, c_2) \in \mathcal{C}$ do
16:        $\tilde{\mathcal{I}} \leftarrow \mathcal{I} \bar{\cap} \{x_1 > c_1, x_2 > c_2\}$;
17:        if $(c_1, c_2) \notin \mathcal{Q}$ and DimMax$(G, \tilde{\mathcal{I}}, \mathcal{F}, 3) > 0$ then
18:            $\mathcal{Q}$.Push($(c_1, c_2)$, DimMax$(G, \tilde{\mathcal{I}}, \mathcal{F}, 3)$);
19: return $\mathcal{R}$;
```

Algorithm 7 UpdateCornerPoints$(\mathcal{C}, s', d)$

```
Input:   Corner Points $\mathcal{C}$, Skyline Point $s' = (x'_1, \ldots, x'_d)$, $d$.
Output:  Updated Corner Points by Adding $s'$.
1: for $i = 1$ to $d$ do
2:     $\mathcal{C}'_i \leftarrow \emptyset$;
3:     for all $(c = (x_1, \ldots, x_d)) \in \mathcal{C}$ s.t. $x_j < x'_j$ for $1 \leq j \leq d$ do
4:         $\mathcal{C} \leftarrow \mathcal{C} \setminus \{c\}$; replace $x_i$ with $x'_i$ in $c$;
5:         $\mathcal{C}'_i \leftarrow \mathcal{C}'_i \cup \{c\}$;
6:     $\mathcal{C}'_i \leftarrow$ Skyline$(\mathcal{C}'_i, d$, MIN); {// computed by classic skyline algorithms}
7: return $\mathcal{C} \cup \mathcal{C}'_i \cup \ldots \cup \mathcal{C}'_d$;
```

Implementation details. Our algorithm for the $d = 3$ case is detailed in Algorithm 6. In the priority queue $\mathcal{Q}$, we use a pair $((c_1, c_2), c_3)$ to denote a regular space, where (c_1, c_2) denotes the corner point corresponding to the subspace and c_3 is the priority of that subspace (i.e., the maximal f_3 value in that subspace). Initially, the algorithm pushes the initial space into $\mathcal{Q}$ if it is a regular space (lines 1-3). Then, the algorithm iteratively computes the skyline communities based on the best-first strategy (lines 4-18). In each iteration, the algorithm first finds the maximal priority from $\mathcal{Q}$ and sets c_3 as the maximal priority (line 5). The algorithm then iteratively pops the regular space whose priority equals c_3 from $\mathcal{Q}$ (line 7). For a popped regular space $((c_1, c_2), c_3)$, the algorithm refines the constraint $\mathcal{I}$ by $\{x_1 > c_1, x_2 > c_2\}$ (line 8), and fixes the node u with $x_3^u = c_3$ (line 9). The algorithm invokes SkylineComm2D with the refined constraint and fixed node u to compute the 2D skyline communities (line 10). The algorithm records all the computed 2D skyline communities in $\mathcal{S}'$ (lines 10-11). Since all the computed 2D skyline communities must be 3D skyline communities by the best-first strategy, the algorithm adds all these computed 2D skyline communities into the answer set $\mathcal{R}$ (lines 12-13). The algorithm then updates the corner points based on the newly-calculated skyline communities in this iteration (line 14).

To compute the corner points, we devise an incremental algorithm which is depicted in Algorithm 7. Specifically, for each skyline community $s' \in \mathcal{S}'$, the algorithm incrementally updates the previously-computed corner points set $\mathcal{C}$ (line 14 in Algorithm 6) by invoking Algorithm 7. Clearly, if the previous-computed corner point c is *completely dominated* by the skyline point s', this corner point must be below the staircase-like shape formed by the updated skyline after adding s'. Here we call a point $x = (x_1, \cdots, x_d)$ completely dominating a point $y = (y_1, \cdots, y_d)$ if and only if $x_i > y_i$ for all $i = 1, \cdots, d$. For example, consider the corner points shown in Fig. 3(a). The red '*' denotes the newly-added skyline point s'. In this example, there is one corner point that is completely dominated by s'. Let $\bar{\mathcal{C}}$ be the set of corner points completely dominated by s'. We remove all the corner points in $\bar{\mathcal{C}}$, because these corner points are no longer the cross points. The completely-dominated corner points in $\bar{\mathcal{C}}$ can be used to compute the new cross points generated by adding s'. For each dominated corner point $\bar{c} \in \bar{\mathcal{C}}$, we obtain a cross point by replacing the x_i coordinate of $\bar{c}$ with that of s' and keeping the other coordinates of $\bar{c}$ unchanged. Clearly, for each completely-dominated corner point, we obtain d new cross points. After obtaining all the cross points, we compute the MIN skyline to get the updated corner points. Reconsider the example shown in Fig. 3. In this example, we obtain two cross points which are also the corner points as shown in Fig. 3(b). Algorithm 7 details this procedure. In Algorithm 7, we compute the MIN skyline in each dimension (line 7 in Algorithm 7), because the cross points



(a) Before updating (b) After updating

Figure 3: Illustration of the corner points updating

generated in different dimensions cannot be dominated w.r.t. each other. Moreover, the remaining corner points in $\mathcal{C}$ (the corner points that are not completely dominated by s') cannot be dominated by the newly-computed corner points. Thus, the algorithm outputs the union of all corner points, forming a MIN skyline.

After updating $\mathcal{C}$, Algorithm 6 pushes the newly-generated regular spaces into $\mathcal{Q}$ (lines 15-18), and then iteratively computes the skyline communities based on the best-first strategy, until $\mathcal{Q} = \emptyset$ and the algorithm terminates. The following example illustrates how Algorithm 6 works.

Example 3. *Reconsider the graph shown in Fig. 1. First, the algorithm pushes $((0,0),4)$ into $\mathcal{Q}$, as the maximal f_3 value in the initial space is 4. Then, the algorithm pops $((0,0),4)$ from $\mathcal{Q}$ (line 7). Subsequently, the algorithm fixes node v_4 and invokes the* SkylineComm2D *algorithm with constraint $(x_1 > 0, x_2 > 0)$ to calculate the 2D skyline communities. In this case, we obtain one 2D skyline community which is $\{v_2, v_4, v_5, v_6\}$ with value $(6,8)$. Then, the algorithm adds $(6,8,4)$ into the answer set as $\{v_2, v_4, v_5, v_6\}$ is also a 3D skyline community. The algorithm then updates the corner points $\mathcal{C}$ (line 14). In this example, we obtain two corner points which are $(6,0)$ and $(0,8)$, i.e., $\mathcal{C} = \{(6,0),(0,8)\}$. For the corner point $(6,0)$, the algorithm invokes* DimMax *with constraint $(x_1 > 6, x_2 > 0)$ to compute the maximal f_3 value. In this case, we get the maximal f_3 value of 3. Similarly, for the corner point $(0,8)$, we obtain the maximal f_3 value of 3. Then, the algorithm pushes $((6,0),3)$ and $((0,8),3)$ into $\mathcal{Q}$. Likewise, in the second iteration, we can obtain a skyline community (v_1, v_2, v_3). After the second iteration, the algorithm terminates as $\mathcal{Q} = \emptyset$. Therefore, we find two skyline communities $\{v_2, v_4, v_5, v_6\}$ and (v_1, v_2, v_3). This result is consistent with our previous result obtained by Algorithm 4.*

The correctness of Algorithm 6 is analyzed in Theorem 5.

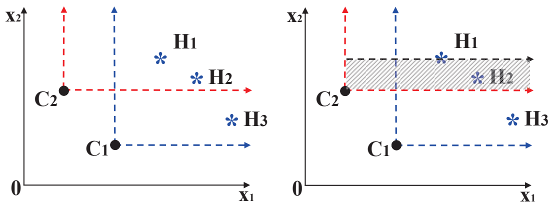
Theorem 5. *Algorithm 6 correctly computes all the 3D skyline communities.*

We analyze the complexity of Algorithm 6 in Theorem 6.

Theorem 6. *Let s be the number of 3D skyline communities. The worst-case time and space complexity of Algorithm 6 is $O(s^2(m+n))$ and $O(m+n+s)$ respectively.*

An improved 3D algorithm. Due to the overlapped space-partition method, a skyline community may be recomputed in Algorithm 6 if it is located in two regular subspaces with the same priority (see lines 6-11 in Algorithm 6). To avoid such redundant computations, we propose an improved algorithm to ensure that no skyline community will be recomputed.

The skyline community clearly cannot be recomputed in two regular subspaces with different priorities in Algorithm 6, thus we need to avoid redundant computations when the regular subspaces have the same priority. Let $\mathcal{P} = \{((c_1^1, c_2^1), c_3), \cdots, ((c_1^t, c_2^t), c_3)\}$ be the set of regular spaces with the same priority c_3. Suppose without loss of generality that c_3 is the current maximal priority in $\mathcal{Q}$ and $c_1^1 > \cdots > c_1^t$. Then, the improved algorithm iteratively computes the skyline communities in the regular spaces in $\mathcal{P}$ following the decreasing order of the c_1 values. To avoid redundant computations, the algorithm maintains the maximal c_2 value denoted by c_2' that it has found so far. Note that since the c_1 values of the regular spaces follow decreasing order, the c_2 values must follow increasing order (because the corner points form a skyline). Then, the algorithm fixes node u with $x_3^u = c_3$, and invokes the SkylineComm2D algorithm with constraint $(x_1 > c_1^i, x_2 > c_2')$ and fixed node u to compute the 2D skyline communities. For all the computed 2D skyline communities $\mathcal{S}_{tmp}$, the algorithm finds the maximal c_2 value in $\mathcal{S}_{tmp}$ and updates c_2' if it is larger than c_2'. Based on this, the algorithm can prune the dominated space in the subsequent regular spaces, which thus avoids recomputing the skyline communities.



(a) Two regular spaces (b) Pruning the space C2

Figure 4: Illustration of the idea of the improved algorithm

Algorithm 8 ImprovedNew3D$(G, \mathcal{I}, \mathcal{F})$

Input: A multi-valued graph G, constraints $\mathcal{I}$, fixed nodes set $\mathcal{F}$.
Output: Skyline Communities in G.

```
1: Result $\mathcal{R} \leftarrow \emptyset$; Priority Queue $\mathcal{Q} \leftarrow \emptyset$; $\mathcal{C} \leftarrow \{(0,0)\}$;
2: if DimMax$(G,\mathcal{I},\mathcal{F},3) > 0$ then
3:     $\mathcal{Q}$.Push$((0,0),$ DimMax$(G,\mathcal{I},\mathcal{F},3))$;
4: while $\mathcal{Q} \neq \emptyset$ do
5:     $c_3 \leftarrow \mathcal{Q}$.MaxVal(); $c_2' \leftarrow 0$; $\mathcal{S}' \leftarrow \emptyset$;
6:     while $\mathcal{Q}$.MaxVal() $= c_3$ do
7:         $((c_1,c_2),c_3) \leftarrow \mathcal{Q}$.Pop();
           {// following the decreasing order of the $c_1$ value}
8:         $c_2' \leftarrow \max(c_2', c_2)$; $\tilde{\mathcal{I}} \leftarrow \mathcal{I} \hat{\cap} \{x_1 > c_1, x_2 > c_2'\}$;
9:         Let $u$ be the node that $x_3^u = c_3$; $\tilde{\mathcal{F}} \leftarrow \mathcal{F} \cup \{u\}$;
10:        $\mathcal{S}_{tmp} \leftarrow$ SkylineComm2D$(G,\tilde{\mathcal{I}},\tilde{\mathcal{F}})$;
11:        $c_2' \leftarrow \max(c_2', \max\{x_2' | (x_1',x_2') \in \mathcal{S}_{tmp}\})$;
12:        $\mathcal{S}' \leftarrow \mathcal{S}' \cup \mathcal{S}_{tmp}$;
13:    for all $(c_1,c_2) \in \mathcal{S}'$ do
14:        $\mathcal{R} \leftarrow \mathcal{R} \cup \{(c_1,c_2,c_3)\}$;
15:    for all $s' \in \mathcal{S}'$ do $\mathcal{C} \leftarrow$ UpdateCornerPoints$(\mathcal{C}, s', 2)$;
16:    for all $(c_1,c_2) \in \mathcal{C}$ do
17:        $\tilde{\mathcal{I}} \leftarrow \mathcal{I} \hat{\cap} \{x_1 > c_1, x_2 > c_2\}$;
18:        if $(c_1,c_2) \notin \mathcal{Q}$ and DimMax$(G,\tilde{\mathcal{I}},\mathcal{F},3) > 0$ then
19:            $\mathcal{Q}$.Push$((c_1,c_2),$ DimMax$(G,\tilde{\mathcal{I}},\mathcal{F},3))$;
20: return $\mathcal{R}$;
```

For ease of understanding, we use an example to illustrate the idea of our improved algorithm. Consider the example shown in Fig. 4. Suppose that we have two regular spaces that have the same priority as shown in Fig. 4(a). Let $C_1 = (c_1^1, c_1^2)$ and $C_2 = (c_2^1, c_2^2)$ be the corner points of these two regular spaces respectively. For convenience, we refer to these two regular spaces as space C_1 and space C_2. Following the decreasing order of c_1 value, the algorithm first pops C_1 from the priority queue $\mathcal{Q}$, and then computes the skyline communities in the space C_1. In this example, there are three skyline communities $\mathcal{S}_{tmp} = \{H_1, H_2, H_3\}$ calculated in the space C_1. The algorithm updates c_2' by $f_2(H_1)$, because $f_2(H_1)$ is maximal among all the skyline communities. In the second iteration, the algorithm pops C_2 from $\mathcal{Q}$. Since $c_2^2 < c_2'$, the algorithm invokes SkylineComm2D with constraint $(x_1 > c_2^1, x_2 > c_2')$ to compute the skyline communities in space C_2. Due to the constraint $x_2 > c_2'$, the shading area in the space C_2 in Fig. 4(b) is pruned. As a result, the skyline communities H_1 and H_2 will not be recomputed in the second iteration. The detailed description of our algorithm is shown in Algorithm 8.

In lines 6-12 of Algorithm 8, since c_2' does not decrease, none of the computed skyline communities are recomputed in the subsequent iterations. On the other hand, a skyline community also cannot be recomputed in spaces with different priorities (i.e.,

Algorithm 9 $\mathsf{NewHighD}(G, \mathcal{I}, \mathcal{F}, d)$

Input: A multi-valued graph G, constraints $\mathcal{I}$, fixed nodes set $\mathcal{F}$, $d \geq 3$.
Output: Skyline Communities in G.

```
1: if $d = 3$ then return $\mathsf{ImprovedNew3D}(G, \mathcal{I}, \mathcal{F})$;
2: Result $\mathcal{R} \leftarrow \emptyset$; Priority Queue $\mathcal{Q} \leftarrow \emptyset$; $\mathcal{C} \leftarrow \{(0, \ldots, 0)_{d-1}\}$;
3: if $\mathsf{DimMax}(G, \mathcal{I}, \mathcal{F}, d) > 0$ then
4:     $\mathcal{Q}.\mathsf{Push}((0, \ldots, 0)_{d-1}, \mathsf{DimMax}(G, \mathcal{I}, \mathcal{F}, d))$;
5: while $\mathcal{Q} \neq \emptyset$ do
6:     $c_d \leftarrow \mathcal{Q}.\mathsf{MaxVal}()$; $\mathcal{S}' \leftarrow \emptyset$;
7:     while $\mathcal{Q}.\mathsf{MaxVal}() = c_d$ do
8:         $((c_1, \ldots, c_{d-1}), c_d) \leftarrow \mathcal{Q}.\mathsf{Pop}()$;
9:         $\tilde{\mathcal{I}} \leftarrow \mathcal{I}\bar{\cap}\{x_1 > c_1, \ldots, x_{d-1} > c_{d-1}\}$;
10:        Let $u$ be the node that $x_d^u = c_d$; $\tilde{\mathcal{F}} \leftarrow \mathcal{F} \cup \{u\}$;
11:        $\mathcal{S}_{tmp} \leftarrow \mathsf{NewHighD}(G, \tilde{\mathcal{I}}, \tilde{\mathcal{F}}, d-1)$;
12:        $\mathcal{S}' \leftarrow \mathcal{S}' \cup \mathcal{S}_{tmp}$;
13:     for all $(c_1, \ldots, c_{d-1}) \in \mathcal{S}'$ do
14:         $\mathcal{R} \leftarrow \mathcal{R} \cup \{(c_1, \ldots, c_{d-1}, c_d)\}$;
15:     for all $s' \in \mathcal{S}'$ do $\mathcal{C} \leftarrow \mathsf{UpdateCornerPoints}(\mathcal{C}, s', d-1)$;
16:     for all $(c_1, \ldots, c_{d-1}) \in \mathcal{C}$ do
17:         $\tilde{\mathcal{I}} \leftarrow \mathcal{I}\bar{\cap}\{x_1 > c_1, \ldots, x_{d-1} > c_{d-1}\}$;
18:         if $(c_1, \ldots, c_{d-1}) \notin \mathcal{Q}$ and $\mathsf{DimMax}(G, \tilde{\mathcal{I}}, \mathcal{F}, d) > 0$ then
19:             $\mathcal{Q}.\mathsf{Push}((c_1, \ldots, c_{d-1}), \mathsf{DimMax}(G, \tilde{\mathcal{I}}, \mathcal{F}, d))$;
20: return $\mathcal{R}$;
```

different c_3 values), thus each skyline community is only calculated once by Algorithm 8. We can apply a similar argument used in Theorem 5 to prove the correctness of Algorithm 8. Below, we analyze the time and space complexity of the algorithm.

THEOREM 7. *The time and space complexity of Algorithm 8 is $O(s(m + n))$ and $O(m + n + s)$ respectively, where s is the total number of 3D skyline communities.*

Note that the worst-case time complexity of Algorithm 8 can be dominated by $O(n^2(m + n))$, because the total number of 3D skyline communities is bounded by n^2. Therefore, even in the worst case, Algorithm 8 is also better than Algorithm 4. In our experiments, we will show that the ImprovedNew3D algorithm is at least one order of magnitude faster than the Basic3D algorithm, and uses much less memory. Furthermore, since the ImprovedNew3D algorithm outputs the skyline communities progressively, it is very useful when the application only needs part of the skyline communities. However, the Basic3D algorithm may generate invalid results, thus it is not a progressive algorithm.

5.2 Handling the $d > 3$ case

We extend Algorithm 6 to handle the $d > 3$ case in Algorithm 9. The general procedure of Algorithm 9 is very similar to that of Algorithm 6. The main difference is that the algorithm recursively invokes itself with a parameter $d - 1$ to compute all the $(d - 1)$-dimensional skyline communities (line 11). In addition, the pruning idea used in Algorithm 8 cannot be applied to the $d > 3$ case. This is because in the $d > 3$ case, each regular space is represented by a $(d - 1)$-dimensional point ($d - 1 \geq 3$). In the $(d - 1)$-dimensional space ($d - 1 \geq 3$), we cannot use a similar method to that illustrated in Fig. 4 to prune the dominated space, so the resulting space is still regular. The correctness analysis of Algorithm 9 is also very similar to the analysis of Algorithm 6, thus we omit the details for brevity. Below, we analyze the time and space complexity of the algorithm.

THEOREM 8. *The worst-case time and space complexity of Algorithm 9 is $O((d - 1)!s^{d-2}(m + n))$ and $O(m + n + ds)$ respectively, where s denotes the number of d-dimensional skyline communities.*

Note that the time complexity analysis in Theorem 8 is the worst-case complexity. In practice, the time cost of our algorithm is much lower than the worst-case complexity, because our algorithm substantially prunes the dominated space. Moreover, s and d are typically not very large in practice (e.g., $s \leq 10^5$ and $d \leq 5$), thus our algorithm can be very efficient. In the experiments, we will show that our algorithm is at least one order of magnitude faster than the basic algorithm, and it can also be scaled to handle large graphs. Compared to Algorithm 5, Algorithm 9 is a progressive algorithm which is very useful for applications that require only part of the skyline communities.

Network	n	m	$d_{\max}$	$k_{\max}$
Slashdot	79K	0.5M	2507	53
Delicious	536K	1.4M	3216	33
Lastfm	1.2M	4.5M	5150	70
Flixster	2.5M	7.9M	1474	68

Table 1: Datasets (K = 10^3 and M = 10^6)

6. EXPERIMENTS

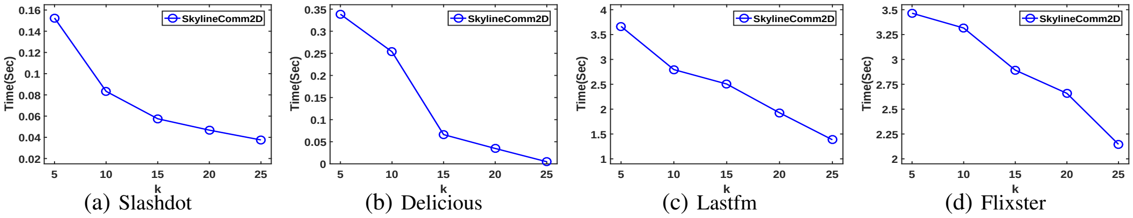
We conduct comprehensive experiments to evaluate the proposed model and algorithms. For the $d = 2$ case, we implement the SkylineComm2D algorithm (Algorithm 2). For the $d \geq 3$ case, we implement two algorithms: Basic (Algorithm 5) and New (Algorithm 9). Note that in the Basic algorithm, we have integrated the pruning rule proposed in Section 4.3. For convenience, when $d = 2$, both Basic and New are the same as the SkylineComm2D algorithm. Since all the existing community search algorithms cannot be used for skyline community search, we use the Basic algorithm as the baseline algorithm for performance studies. All the algorithms are implemented in C++, and all experiments are conducted on a PC with two 2.4GHz Intel Xeon CPUs and 64GB main memory running Ubuntu 14.04.5 (64-bit).

Datasets. We use four real-world networks in our experiments. The statistics of the datasets are summarized in Table 1. In Table 1, $d_{\max}$ and $k_{\max}$ denote the maximal degree and the maximal core number of the network, respectively. All four datasets are social networks, downloaded from (`http://networkrepository.com/`). Note that the original datasets do not contain numerical attributes. To evaluate the performance of our algorithms, we apply a widely-used method in the skyline processing literature [4] to generate the numerical attributes for our datasets. We use the same method proposed in [4] to generate three different types of numerical attributes in each network: 1) *independence*, 2) *correlation*, and 3) *anti-correlation*. *Independence* implies that the attribute values are generated independently using a uniform distribution. *Correlation* means that if a node is good in one dimension (attribute), then it is also good in the other dimensions. *Anti-correlation* indicates that if a node is good in one dimension, then it is bad in one or all of the other dimensions. Intuitively, the number of skyline communities in the network with correlated attributes should be much smaller than the number in the same network with independent attributes or anti-correlated attributes, and among them, the number of skyline communities in the networks with anti-correlated attributes is maximal. Due to space limit, we mainly report the results obtained from the networks with independent attributes, and the results for the other types of attributed networks are reported in Appendix A.2.

6.1 Performance studies for $d = 2$

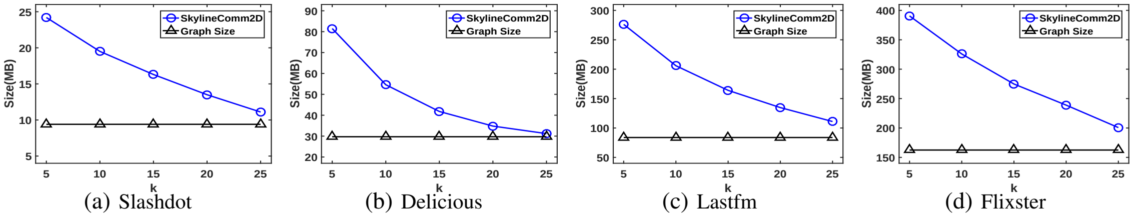
Exp-1: Efficiency of SkylineComm2D**.** We vary the core number k from 5 to 25, and evaluate the efficiency of the SkylineComm2D algorithm. The results in the networks with independent attributes are shown in Figs. 5. As can be seen, the running time of SkylineComm2D decreases with increasing k. This is because the graph size after pruning decrease with increasing k. For example, in Fig. 5(d), when $k = 15$ SkylineComm2D takes 2.8 seconds to output all the skyline communities, whereas it only uses 2.15 seconds if $k = 25$. In all the datasets, SkylineComm2D takes less than 4 seconds to output all the results. These results indicate that SkylineComm2D is very efficient in practice, which confirm the complexity analysis of SkylineComm2D in Section 3.

Exp-2: Memory overhead of SkylineComm2D**.** We show the



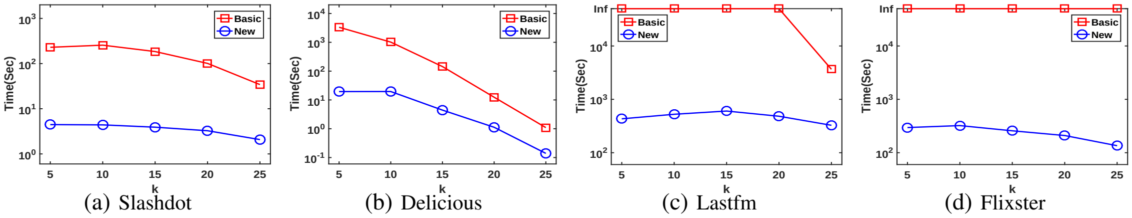
(a) Slashdot (b) Delicious (c) Lastfm (d) Flixster

Figure 5: Efficiency of SkylineComm2D in networks with independent attributes (vary k)



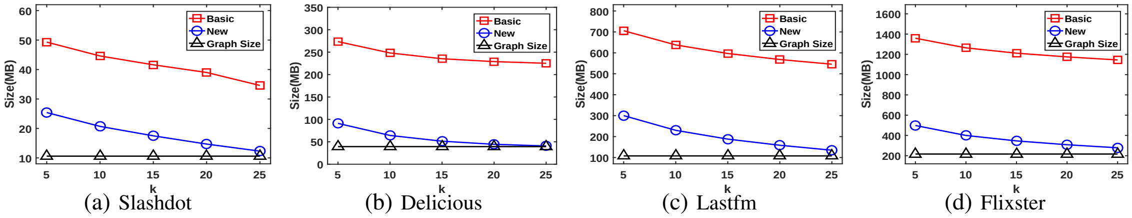
(a) Slashdot (b) Delicious (c) Lastfm (d) Flixster

Figure 6: Memory overhead of SkylineComm2D in networks with independent attributes (vary k)



(a) Slashdot (b) Delicious (c) Lastfm (d) Flixster

Figure 7: Efficiency of Basic and New in networks with independent attributes (vary k, $d = 3$)



(a) Slashdot (b) Delicious (c) Lastfm (d) Flixster

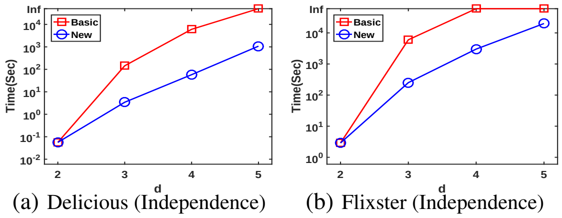
Figure 8: Memory overhead of Basic and New in networks with independent attributes (vary k, $d = 3$)

memory cost of SkylineComm2D with varying k in the independent attributed networks. Similar results can be observed in the other types of attributed networks. Fig. 6 depicts our results. From Fig. 6, we can see that the memory cost of SkylineComm2D decreases with increasing k in all the datasets. This is because the graph size decreases with increasing k. Also, we can see that the memory overhead of our algorithm is at most 3 times the graph size, indicating that SkylineComm2D is memory-efficient. These results are consistent with the space complexity of SkylineComm2D.

6.2 Performance studies for $d \geq 3$

Exp-3: Efficiency ($d = 3$). For $d = 3$, the efficiency testings of Basic and New in the networks with independent attributes are reported in Fig. 7. Also, for the other types of attributed networks, the results can be found in Appendix A.2. As can be seen, if $k \geq 15$, the running time of both Basic and New decreases with increasing k. However, if $k < 15$, the running time slightly increases when k increases. In all the datasets, New is at least one order of magnitude faster than Basic. For instance, in Fig. 7(a), when $k = 20$, New takes 5.3 seconds, whereas Basic takes 100.6 seconds. Also, as shown in Figs. 7(c) and (d), Basic is intractable in the Lastfm and Flixster datasets ('Inf' means that the algorithm cannot terminate in 50,000 seconds). New, however, still runs very fast in these datasets. For example, in Fig. 7(d), New only takes 200 seconds to find all 3D skyline communities in Flixster. These results confirm our theoretical analysis in Sections 4 and 5.

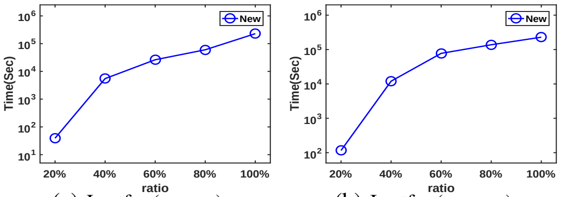
Exp-4: Memory overhead. For $d = 3$, the memory cost of Basic and New is shown in Fig. 8 in the networks with independent attributes. Similar results can also be observed in different types of attributed networks and also for the other d values. As can be seen, New consumes much less memory than Basic in all the datasets. This is because Basic needs to maintain a large number of invalid skyline communities. Generally, the memory size of both Basic and New decreases with increasing k. When $k = 25$, the space cost of New is close to that of the graph size, as the algorithm significantly reduces the graph size when k is large. These results demonstrate that New is memory-efficient, which are consistent with the space complexity analysis in Section 5.



(a) Delicious (Independence) (b) Flixster (Independence)

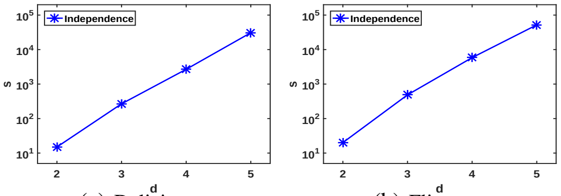
Figure 9: Efficiency of Basic and New (Vary d, $k = 15$)

Exp-5: Efficiency (Vary d). We evaluate the efficiency of Basic and New by varying d from 2 to 5. Note that when $d = 2$, we refer to both Basic and New as the SkylineComm2D algorithm. The results in the Delicious and Flixster networks are reported in Fig. 9, and similar results can be observed in the other datasets. As desired, the running time of both Basic and New increases



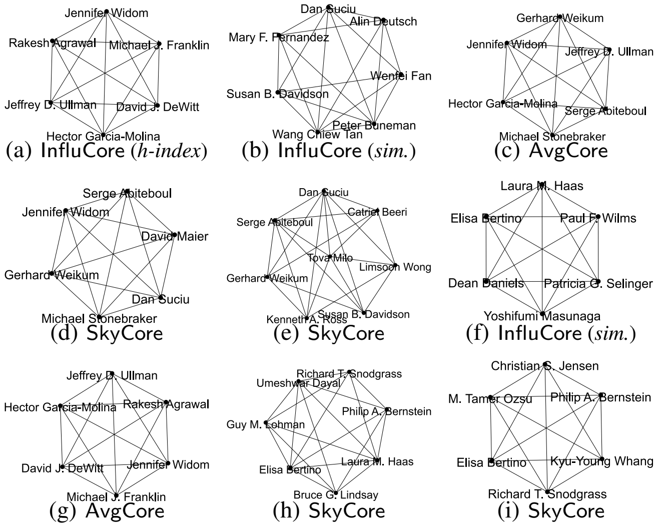
(a) Lastfm (vary n) (b) Lastfm (vary m)

Figure 10: Scalability of New ($k = 15$, $d = 6$)



(a) Delicious (b) Flixster

Figure 11: Number of skyline communities (vary d, $k = 15$)



(a) InfluCore (*h-index*) (b) InfluCore (*sim.*) (c) AvgCore

(d) SkyCore (e) SkyCore (f) InfluCore (*sim.*)

(g) AvgCore (h) SkyCore (i) SkyCore

Figure 12: Comparison of different methods in the Aminer dataset ($k = 5$); Figs. (a-e) show the results of the query "Prof. Dan Suciu", and Figs. (a, f-i) depict the results of the query "Prof. Elisa Bertino".

when d increases, and New is at least one order of magnitude faster than Basic when $d \geq 3$. Also, we can see that Basic is costly, and it is intractable when $d \geq 4$ in the Flixster dataset. For the New algorithm, the running time typically increases by 10 times when d increases by 1, because the number of skyline communities increases quickly when d increases by 1. In practice, d is often very small (e.g., $d \leq 5$). This is because the nodes in most real-world networks do not have too many numeric attributes. For example, in the Aminer scientific network (`http://aminer.org/`), each node has 7 numeric attributes. To the best of our knowledge, Aminer is the publicly available network that has the largest number of numeric attributes. On the other hand, in the skyline community model, d is equal to the number of *selected* numeric attributes which is typically much smaller than the total number of numeric attributes. Therefore, in this sense, our New algorithm is tractable for handling most real-world applications.

Exp-6: Scalability. We evaluate the scalability of New when $d > 5$. To this end, we vary n and m in the Lastfm network with independent attributes. The results for $d = 6$ are reported in Fig. 10. Similar results can also be observed for other d values. As can be seen, the running time of New increases smoothly with varying m and n, implying that the algorithm scales well w.r.t. the graph size. These results indicate that New is scalable to handle large real-world graphs given that $d = 6$. Again, as d is often very small (e.g., $d \leq 5$), our algorithm is scalable to handle most real-world applications. In addition, we also study the scalability of New for a query-dependent skyline community search problem when $d > 6$. Due to space limit, the results are reported in the Appendix (see Additional Exp-6).

Exp-7: Number of skyline communities (Vary d). We show the number of skyline communities found by our algorithm with varying d. The results in Delicious and Flixster networks with independent attributes are depicted in Fig. 11. Similar results can also be observed in the other datasets. From Fig. 11, we can see that the number of skyline communities, denoted by s, increases by 10 times when d increase by 1. These results are consistent with the efficiency results of our algorithms.

6.3 Case studies

We use two additional datasets for case studies: Aminer and Gowalla. The Aminer dataset is a scientific network collected from (`aminer.org`) which contains the authors in database, data mining, machine learning, and information retrieval areas. The dataset comprises 5,411 nodes and 17,477 edges. We crawl four numeric attributes for each author: *h-index*, *the number of papers*, *activity*, and *diversity*. Here *h-index* measures the academic influence of an author, *activity* measures whether an author is active or not in recent years, and *diversity* measures the diversity of the author's research topics. The Gowalla dataset is a location-based social network (LBSN). We download this dataset from (`http://snap.stanford.edu`). For each node in the Gowalla, we extract a location from its check-in records. We compare our approach (denoted by SkyCore) with three baseline methods: InfluCore, AvgCore, and MergeCore. InfluCore denotes the influential community search algorithm [15] which only considers one numeric attribute. AvgCore first takes the average value over the d numeric attributes for each node, and then invokes InfluCore to compute the communities based on the average values. MergeCore first finds the top-1 influential communities on each numeric attribute, and then merges the d resulting communities. We perform three different case studies to evaluate the proposed method, two of which are reported below. The case study on team search is reported in the Appendix A.2 due to space limit.

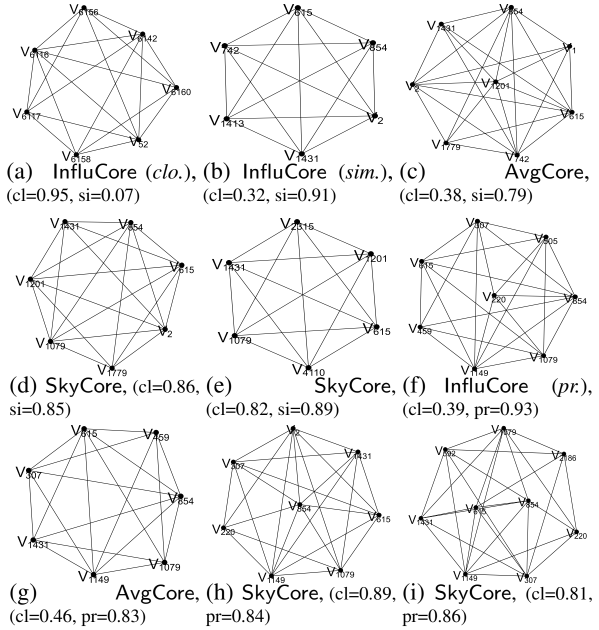
Exp-8: Finding similar and influential communities in Aminer. In this case study, we aim to find the influential communities such that their members are similar to a given query node u based on the Jaccard similarity. For each node in Aminer, we use the h-index to measure the influence. We compute the Jaccard similarity between u and the other nodes in the network (for a node v, the Jaccard similarity between u and v is $|N(u) \cap N(v)|/|N(u) \cup N(v)|$). As a result, we can obtain two numeric attributes for each node (the h-index and the Jaccard similarity). For a fair comparison, we normalize each numeric attribute into the range $[0, 1]$ in all case studies. Based on these two normalized numeric attributes, we apply the above four different methods with parameter $k = 5$ to compute the communities. Figs. 12(a-e) report the results of professor Dan Suciu's communities, and Figs. 12(a, f-i) show the results of professor Elisa Bertino's communities. In Figs. 12(a-b, f), We can see that the results obtained by InfluCore only capture one attribute. For example, in Fig. 12(a), the community mainly contains the influential authors in the database community which are not necessarily similar to professor Dan Suciu (by Jaccard similarity). On the other hand, in Fig. 12(b), the community comprises the authors that are similar to professor Dan Suciu, but their h-index values are not necessarily very high. Also, we can observe that there is no overlap among the communities (a), (b), and (f), thus MergeCore cannot obtain a connected community. In effect, we find that MergeCore fails to find a connected community in most of the case studies. This is because the resulting communities on different attributes are typically uncorrelated, and therefore the merged community is often disconnected. From Figs. 12(c) and (g), the resulting communities obtained by AvgCore also cannot capture both influence and similarity. For example, in Fig. 12(g), the community includes many high influential researchers, but they are dissimilar to professor Elisa Bertino. Moreover, the community also does not contain professor Elisa Bertino. As shown in Figs. 12(d-e, h-i), our approach (SkyCore) performs much better than all the baseline methods. For example, in Fig. 12(d), the community comprises many high influential researchers who are also very similar to professor Dan Suciu based on the Jaccard similarity. These results indicate that the proposed skyline community approach can indeed capture both influence and similarity of a community. Thus, we believe that our approach is very useful for such a personalized influential community search application.

Exp-9: Finding close communities in Gowalla. In this case study, we aim to identify the "close and similar" (or "close and influential") communities for a query node in the LBSN. For each node v, we compute three numeric attributes: the Euclidian distance and the Jaccard similarity between the query node u and v ($|N(u) \cap N(v)|/|N(u) \cup N(v)|$), and the PageRank of v. We normalize all the attributes into the range $[0, 1]$. For the normalized Euclidian distance x_v of v, we use $1 - x_v$ to measure the closeness between u and v. The PageRank of v is used to measure the influence of v. The results based on the query node v_{615} are reported in Fig. 13. Similar results can be observed by the other query nodes. In Fig. 13, each reported community is associated with two values, denoting the computed f values of the community (Eq. (1)). Similar to the previous case study, the InfluCore only captures one attribute. From example, in Fig. 13(a), the community contains the nodes that are close to the query node v_{615}. The closeness value of this community is 0.95, while the similarity value is only 0.07. Also, we can see that this community does not contain the query node. The AvgCore performs better than InfluCore, but it is significantly worse than SkyCore. For example, in Figs. 13(c-d), SkyCore dominates InfluCore in both closeness and similarity. Similarly, when using the closeness and PageRank attributes, SkyCore is also the winner among all the algorithms as shown in Figs. 13(f-i). These results indicate that our algorithm is very effective in applications of finding "close and similar" communities (or "close and influential" communities) in LBSN.

7. RELATED WORK

Community model and search. Community in a graph is typically represented by a cohesive subgraph. A large number of community models have been proposed, such as maximal clique [6], k-core [20, 16, 10], k-truss [8, 23, 12, 13], maximal k-edge connected subgraph [25, 5, 1], quasi-clique [9], locally densest subgraph [19], query-biased density [24], and so on. All these community models only consider the graph structural information and ignore the attributes (numerical and textual attributes) associated with the nodes. Recently, Fang et al. [11] proposed an attributed community model that is tailored to the graphs with textual attributes. Li et al. [15] introduced an influential community model, which takes the node's influence into consideration. However, their model only considers one numerical attribute (i.e., the influence), thus the techniques proposed in [15] cannot be used for our problem when $d > 1$. Our skyline community model is the first community model that can capture d-dimensional numerical attributes, and our work is also the first to introduce skyline for community modeling.

Another related line of work is on community search, where the goal is to find a cohesive subgraph that includes the query nodes. Sozio et al. [22] studied the community search problem in social networks based on the k-core model. Huang et al. [12] introduced a k-truss community model, and proposed several efficient k-truss community search algorithms. Cui et al. [9] investigated an overlap community search problem based on the quasi-clique model. More recently, Huang et al. [13] proposed the closest truss community model to find the k-truss community with small diameter. In this paper, we study the skyline community search problem, and the proposed techniques are dramatically different from all the previous community search algorithms.



(a) InfluCore (*clo.*), (cl=0.95, si=0.07)
(b) InfluCore (*sim.*), (cl=0.32, si=0.91)
(c) AvgCore, (cl=0.38, si=0.79)
(d) SkyCore, (cl=0.86, si=0.85)
(e) SkyCore, (cl=0.82, si=0.89)
(f) InfluCore (*pr.*), (cl=0.39, pr=0.93)
(g) AvgCore, (cl=0.46, pr=0.83)
(h) SkyCore, (cl=0.89, pr=0.84)
(i) SkyCore, (cl=0.81, pr=0.86)

Figure 13: Comparison of different methods in the Gowalla dataset with a query node "v_{615}"($k = 5$); Figs. (a-e) use two attributes: closeness and similarity , Figs. (f-i) use two attributes: closeness and PageRank; 'cl', 'si', 'pr', denotes the closeness, similarity, PageRank values respectively.

Skyline computation. The skyline computation problem was originally studied in the theory community. Kung et al. [14] proposed an $O(n \log n)$ algorithm to find the skyline in 2D space. For the d-dimensional space, they also proposed an $O(n \log^{d-2} n)$ algorithm. In the database community, the skyline operator was first introduced by Borzsonyi et al. [4]. A large number of algorithms have since been devised to efficiently find the skyline under different settings [17, 7, 18, 21, 2]. For example, Papadias et al. [17] proposed an efficient algorithm for progressively finding the skyline. Sheng and Tao [21] proposed an external-memory algorithm to compute the skyline efficiently. Pei et al. [18] studied the skyline computation problem for uncertain data. Asudeh et al. [2] studied the skyline computation problem for hidden web data. In this work, we are the first to study the skyline community search problem, where the skyline operator is defined over all the communities in a multi-valued graph. Since our problem is fundamentally different from previous skyline problems, all the existing algorithms cannot be used for skyline community search.

8. CONCLUSION

In this paper, we propose a novel skyline community model to detect interesting communities in a multi-valued network, where each node is associated with d numerical attributes. The resulting communities identified by our model cannot be dominated by the other communities in the d-dimensional attribute space. We develop a basic and a novel space-partition algorithm to find all the skyline communities efficiently. The worst-case time complexity of the space-partition algorithm relies mainly on the number of skyline communities, thus it is very efficient if the size of the answer is not very large. Extensive experiments in real-world multi-valued networks demonstrate the efficiency, scalability and effectiveness of our solutions.

9. REFERENCES

[1] T. Akiba, Y. Iwata, and Y. Yoshida. Linear-time enumeration of maximal k-edge-connected subgraphs in large networks by random contraction. In *CIKM*, pages 909–918, 2013.

[2] A. Asudeh, S. Thirumuruganathan, N. Zhang, and G. Das. Discovering the skyline of web databases. *PVLDB*, 9(7):600–611, 2016.

[3] V. Batagelj and M. Zaversnik. An O(m) algorithm for cores decomposition of networks. *CoRR*, cs.DS/0310049, 2003.

[4] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.

[5] L. Chang, J. X. Yu, L. Qin, X. Lin, C. Liu, and W. Liang. Efficiently computing k-edge connected components via graph decomposition. In *SIGMOD*, pages 205–216, 2013.

[6] J. Cheng, Y. Ke, A. W.-C. Fu, J. X. Yu, and L. Zhu. Finding maximal cliques in massive networks. *ACM Trans. Database Syst.*, 36(4):21:1–21:34, 2011.

[7] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with presorting. In *ICDE*, pages 717–719, 2003.

[8] J. Cohen. Trusses: Cohesive subgraphs for social network analysis. *Technical report, National Security Agency*, 2005.

[9] W. Cui, Y. Xiao, H. Wang, Y. Lu, and W. Wang. Online search of overlapping communities. In *SIGMOD*, pages 277–288, 2013.

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

[11] Y. Fang, R. Cheng, S. Luo, and J. Hu. Effective community search for large attributed graphs. *PVLDB*, 9(12):1233–1244, 2016.

[12] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu. Querying k-truss community in large and dynamic graphs. *SIGMOD*, pages 1311–1322, 2014.

[13] X. Huang, L. V. S. Lakshmanan, J. X. Yu, and H. Cheng. Approximate closest community search in networks. *PVLDB*, 9(4):276–287, 2015.

[14] H. T. Kung, F. Luccio, and F. P. Preparata. On finding the maxima of a set of vectors. *J. ACM*, 22(4):469–476, 1975.

[15] R. Li, L. Qin, J. X. Yu, and R. Mao. Influential community search in large networks. *PVLDB*, 8(5):509–520, 2015.

[16] R. Li, J. X. Yu, and R. Mao. Efficient core maintenance in large dynamic graphs. *IEEE Trans. Knowl. Data Eng.*, 26(10):2453–2465, 2014.

[17] D. Papadias, Y. Tao, G. Fu, and B. Seeger. An optimal and progressive algorithm for skyline queries. In *SIGMOD*, pages 467–478, 2003.

[18] J. Pei, B. Jiang, X. Lin, and Y. Yuan. Probabilistic skylines on uncertain data. In *VLDB*, pages 15–26, 2007.

[19] L. Qin, R. Li, L. Chang, and C. Zhang. Locally densest subgraph discovery. In *KDD*, pages 965–974, 2015.

[20] S. B. Seidman. Network structure and minimum degree. *Social Networks*, 5(3):269–287, 1983.

[21] C. Sheng and Y. Tao. On finding skylines in external memory. In *PODS*, pages 107–116, 2011.

[22] M. Sozio and A. Gionis. The community-search problem and how to plan a successful cocktail party. In *KDD*, pages 939–948, 2010.

[23] J. Wang and J. Cheng. Truss decomposition in massive networks. *PVLDB*, 5(9):812–823, 2012.

[24] Y. Wu, R. Jin, J. Li, and X. Zhang. Robust local community detection: On free rider effect and its elimination. *PVLDB*, 8(7):798–809, 2015.

[25] R. Zhou, C. Liu, J. X. Yu, W. Liang, B. Chen, and J. Li. Finding maximal k-edge-connected subgraphs from a large graph. In *EDBT*, pages 480–491, 2012.

A. APPENDIX

A.1 Missing proofs

LEMMA 1. *Let H_1 with values $(f_1(H_1), f_2(H_1))$ be the skyline community that has the maximal f_2 value over all the skyline communities. Then, the nodes in G whose x_1 values are no larger than $f_1(H_1)$ cannot be contained in the other skyline communities.*

PROOF. We prove this lemma by contradiction. Suppose that there is a skyline community H ($H \neq H_1$) that contains a node u with $x_1^u \leq f_1(H_1)$. H can be dominated by H_1 because $f_1(H) \leq f_1(H_1)$ and $f_2(H) < f_2(H_1)$, which is a contradiction. □

THEOREM 1. *Algorithm 2 correctly computes all the 2D skyline communities.*

PROOF. It is easy to show that the communities returned by Algorithm 2 must be skyline communities. To prove the theorem, we need to show that all the skyline communities have been computed by Algorithm 2. Suppose to the contrary that there is a skyline community H with values $(f_1(H), f_2(H))$ that cannot be obtained by Algorithm 2. We assume that Algorithm 2 iteratively outputs s skyline communities which are H_1, $H_2, \cdots, H_s$. Clearly, by Algorithm 2, we have $f_2(H_1) > f_2(H_2) > \cdots > f_2(H_s)$. Also, by definition, we have $f_1(H_1) < f_1(H_2) < \cdots < f_1(H_s)$. Since H is a skyline community, it is a k-$\widehat{core}$ and it must be contained in the maximal k-core of graph G. By our algorithm, $f_2(H_1)$ is the maximal f_2 value over all the k-$\widehat{cores}$ in G, thus $f_2(H) < f_2(H_1)$. On the other hand, H_s is the last skyline community computed by Algorithm 2, thus invoking Algorithm 1 with constraint $x_1 > f_1(H_s)$ (lines 7-8 in Algorithm 2) results in $f_2 = 0$. That is to say, the graph G cannot contain a k-core with f_1 value larger than $f_1(H_s)$. Therefore, we conclude that $f_1(H) < f_1(H_s)$. Since H is a skyline community, we have $f_1(H_1) < f_1(H) < f_1(H_s)$ and $f_2(H_1) > f_2(H) > f_2(H_s)$.

Furthermore, we claim that $f_1(H)$ and $f_2(H)$ must satisfy that $f_1(H_i) < f_1(H) < f_1(H_{i+1})$ and $f_2(H_i) > f_2(H) > f_2(H_{i+1})$ for a certain H_i ($i = 1, \cdots, s-1$). We can prove this statement by a contradiction. Since $f_1(H_1) < f_1(H) < f_1(H_s)$, there exists a skyline community H_i for $i = 1, \cdots, s-1$ such that $f_1(H_i) < f_1(H) < f_1(H_{i+1})$. Suppose to the contrary that $f_2(H_i) > f_2(H) > f_2(H_{i+1})$ does not hold. Since $f_2(H_1) > f_2(H) > f_2(H_s)$, there exists a skyline community H_j for $j = 1, \cdots, s-1$ and $j \neq i$ such that $f_2(H_j) > f_2(H) > f_2(H_{j+1})$. Assume without loss of generality that $i < j$. We then have $f_1(H_i) < f_1(H) < f_1(H_j)$ and $f_2(H_j) > f_2(H)$. As a result, H_j dominates H, which is a contradiction.

After computing H_i (line 6 in Algorithm 2), the algorithm invokes Algorithm 1 to calculate $f_2(H_{i+1})$ with constraint $x_1 > f_1(H_i)$ (lines 7-8 in Algorithm 2). By Algorithm 1, we know that $f_2(H_{i+1})$ is the maximal f_2 value over all k-$\widehat{cores}$ whose f_1 values are larger than $f_1(H_i)$. Since H is a skyline community with $f_1(H) > f_1(H_i)$, we have $f_2(H) < f_2(H_{i+1})$, which contradicts $f_2(H_i) > f_2(H) > f_2(H_{i+1})$. We can therefore conclude that Algorithm 2 outputs all skyline communities. This completes the proof. □

THEOREM 2. *Let s be the number of skyline communities in G. Then, the worst case time and space complexity of Algorithm 2 are $O(s(m+n))$ and $O(m+n+s)$ respectively.*

PROOF. First, the time complexity of Algorithm 1 is $O(m+n)$, because we only need to scan the graph once. Since Algorithm 2 invokes Algorithm 1 s times, the time complexity of Algorithm 2 is $O(s(m+n))$. For the space complexity, the algorithm only needs to store the graph and several auxiliary arrays (e.g., $visit$) which consume $O(m+n)$ space, and also the algorithm needs to maintain the results which use $O(s)$ space. Therefore, the space complexity of Algorithm 2 is $O(m+n+s)$, which is linear to the graph size and answer size. □

LEMMA 2. *For each dimension x_i ($i = 1, \cdots, d$), the f_i values of all the skyline communities are contained in the set T_i which is computed by Algorithm 3.*

PROOF. We prove the lemma by contradiction. Suppose to the contrary that there is a skyline community H in which $f_i(H) \notin T_i$. Recall that T_i denotes the set of the values of all the influential communities that are computed based on the x_i dimension (see Algorithm 3). We assume without loss of generality that there are t different elements in T_i and $T_i = \{f_i^1, \cdots, f_i^t\}$ with $f_i^1 <, \cdots, < f_i^t$. By definition, it is easy to show that $f_i^1 < f_i(H) < f_i^t$. Thus, there exists j ($1 \leq j \leq t-1$) such that $f_i^j < f_i(H) < f_i^{j+1}$. Let G_j be the graph that is obtained by removing all the nodes whose x_d values are smaller than f_i^j, and H_j be the maximal k-core of G_j. Let $u \in H_j$ be the node with $x_i^u = f_i^j$ (i.e., $u \in H_j$ is the smallest-value node in the x_i dimension). Since H is a k-core with $f_i(H) > f_i^j$, H must be contained in H_j and also H cannot contain node u. On the other hand, since $f_i(H) < f_i^{j+1}$, H cannot be contained in H_{j+1}. By definition, if we remove u from H_j and then compute the maximal k-core, we can obtain H_{j+1}. Since H is a k-core which does not contain u, it must be contained in H_{j+1}, which is a contradiction. This completes the proof. □

THEOREM 3. *Algorithm 4 correctly finds all the 3D skyline communities, and the worst-case time and space complexity of Algorithm 4 is $O(n^2(m+n))$ and $O(n^2)$ respectively.*

PROOF. First, we prove the correctness of Algorithm 4. By Lemma 2, F_3 ($F_3 = T_3$) contains all the possible f_3 values that the 3D skyline communities may have. For each $f_3 \in F_3$, the algorithm calculates all the 2D skyline communities whose values in the x_3 dimension equal f_3. As a result, all the 3D skyline communities must be contained in the union of the sets of all those 2D skyline communities. By computing the skyline in the union of these sets, the algorithm can obtain all the 3D skyline communities.

Second, we analyze the complexity of Algorithm 4. For the time complexity, the algorithm takes $O(m+n)$ time to compute F_3. Then, for each $f_3 \in F_3$, the algorithm invokes a SkylineComm2D algorithm which takes at most $O(n(m+n))$ time. The total time cost taken in the '*for*' loop (line 3) is therefore $O(n^2(m+n))$ in the worst case. Since the size of $\mathcal{T}$ is bounded by n, the total size of $\mathcal{R}$ in line 7 is bounded by $O(n^2)$. Finally, the algorithm calls a traditional skyline algorithm to compute the final results which consumes $O(n^2 \log n)$ [14, 4], because there are at most $O(n^2)$ 3D points recorded in $\mathcal{R}$. The total time complexity is $O(n^2(m+n))$. For the space complexity, we can easily show that the algorithm uses $O(m+n+n^2)$ space, which is dominated by $O(n^2)$. □

THEOREM 4. *For $d \geq 3$, the worst-case time and space complexity of Algorithm 5 is $O(n^{d-1}(m+n+(d-1)\log^{d-3} n))$ and $O(n^{d-1})$ respectively.*

PROOF. We start by analyzing the time complexity. It is easy to show that the total number of skyline points in the d-dimensional discrete space is bounded by $O(n^{d-1})$ for $d \geq 2$. Here the discrete space means that the skyline points in each dimension can only take n discrete values. Let $T(d)$ be the worst-case time complexity of Algorithm 5. Then, $T(d) = n \times T(d-1) + n^{d-1} \log^{d-3}(n^{d-1})$, where $n^{d-1} \log^{d-3}(n^{d-1})$ denotes the time cost of computing the final skyline communities by using the traditional skyline algorithm [14] (for $d \geq 3$). Then, we can obtain that $T(d) = O(n^{d-1}(m+n+(d-1)\log^{d-3} n))$. The space complexity is dominated by the total number of all the $(d-1)$-dimensional skyline communities that are recorded in $\mathcal{R}$, which is $O(n^{d-1})$. This completes the proof. □

THEOREM 5. *Algorithm 6 correctly computes all the 3D skyline communities.*

PROOF. First, we prove that the computed skyline communities are correct 3D skyline communities. Let $\mathcal{R}_i$ be the set of skyline communities computed in the i-th iteration. By the best-first strategy, the algorithm computes the 3D skyline communities following the decreasing order of the f_3 values. Hence, in the i-th iteration, the skyline communities in $\mathcal{R}_i$ cannot be dominated by the undiscovered skyline communities (because the f_3 values of the skyline communities in $\mathcal{R}_i$ must be larger than those of the undiscovered skyline communities). On the other hand, the skyline communities in $\mathcal{R}_i$ cannot be dominated by the skyline communities in $\mathcal{R}_j$ with $j < i$, because the previously-calculated skyline communities cannot dominate the skyline communities in $\mathcal{R}_i$ in terms of the first two dimensions. Second, since the proposed space-partition algorithm does not miss any subspace, all the skyline communities must be discovered by our algorithm. □

THEOREM 6. *Let s be the number of 3D skyline communities. The worst-case time and space complexity of Algorithm 6 is $O(s^2(m+n))$ and $O(m+n+s)$ respectively.*

PROOF. First, we analyze the time complexity of the algorithm. Since each skyline point generates at most two new corner points in the 2D space by Algorithm 7, the total number of corner points generated by our algorithm is bounded by $O(s)$. For each corner point, the algorithm invokes the SkylineComm2D algorithm at most once, which takes at most $O(s(m+n))$ time. Thus, the total cost taken in lines 6-11 is bounded by $O(s^2(m+n))$. In addition, for each skyline point, the time cost to update the corner points in line 14 is $O(s)$. Thus, the total cost taken in line 14 can be bounded by $O(s^2)$, which is dominated by $O(s^2(m+n))$. It is also easy to show that the total cost taken in lines 15-18 is bounded by $O(s^2(m+n))$. It can thus be seen that the worst-case time complexity of Algorithm 6 is $O(s^2(m+n))$. For the space complexity, we need to maintain the graph and the priority $\mathcal{Q}$, which consume $O(m+n+s)$ space in total. This completes the proof. □

THEOREM 7. *The time and space complexity of Algorithm 8 is $O(s(m+n))$ and $O(m+n+s)$ respectively, where s is the total number of 3D skyline communities.*

PROOF. First, we analyze the time complexity of the algorithm. Since no skyline community is recomputed by Algorithm 8, the total time cost of computing all the skyline communities in line 10 is $O(s(m+n))$. Similar to Algorithm 6, the total number of corner points generated by the algorithm is $O(s)$, thus the total time cost taken in lines 16-19 is bounded by $O(s(m+n))$. Finally, we analyze the total time cost of computing the corner points in line 15. A straightforward implementation of Algorithm 7 results in $O(s)$ time complexity. As a result, the total cost of maintaining the corner points set is $O(s^2)$ in the worst case. Recall that Algorithm 8 needs to dynamically maintain the corner points set $\mathcal{C}$ in each iteration. Since all the corner points in $\mathcal{C}$ form a skyline, it is easy to develop a tree-like structure to maintain all the corner points such that finding a completely-dominated corner point can be done in $O(\log s)$ and updating the tree structure can also be done in $O(\log s)$. Since there are $O(s)$ corner points in total, the total maintenance cost is $O(s \log s)$ time. Additionally, in the 3D case, the corner points are 2D points, thus the total cost of computing the MIN skyline in each dimension (line 7 in Algorithm 7) is $O(s)$ time. In the 3D case, s is bounded by n^2, thus the time cost of maintaining the corner points set $\mathcal{C}$ is also dominated by $O(s(m+n))$. Ultimately, the time complexity of Algorithm 8 is $O(s(m+n))$. Second, we can easily show that the space complexity of Algorithm 8 is $O(m+n+s)$, which is the same as Algorithm 6. □

THEOREM 8. *The worst-case time and space complexity of Algorithm 9 is $O((d-1)!s^{d-2}(m+n))$ and $O(m+n+ds)$ respectively, where s denotes the number of d-dimensional skyline communities.*

PROOF. The most time-consuming step in Algorithm 9 is in lines 7-12, because the algorithm needs to recursively invoke itself with a parameter $d-1$. Below, we analyze the time complexity in this recursion procedure. Note that each d-dimensional skyline community generates at most $(d-1)$ corner points by our algorithm. Hence, the total number of $(d-1)$-dimensional corner points is bounded by $(d-1)s$. By recursive analysis, we derive that the algorithm invokes the ImprovedNew3D algorithm at most $(d-1)s \times (d-2)s \cdots \times 3s$ times. Thus, the total time cost is $O((d-1)!s^{d-2}(m+n))$. Note that by a similar recursive analysis, we can see that the total time cost to update the corner points in line 15 and the total time cost to push the corner points into $\mathcal{Q}$ in lines 16-19 can be dominated by $O((d-1)!s^{d-2})$ and $O((d-1)!s^{d-2}(m+n))$ respectively. Thus, the worst-case time complexity of Algorithm 9 is $O((d-1)!s^{d-2}(m+n))$. For the space complexity, the algorithm needs to maintain the graph and the total number of corner points which use $O(m+n+ds)$ space. This completes the proof. □

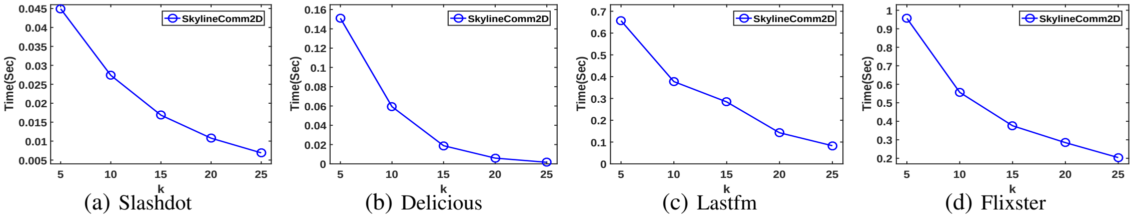
A.2 Additional experiments

Additional Exp-1: Scalability of SkylineComm2D. We vary the number of nodes (n) and edges (m) in the Lastfm network with independent attributes to evaluate the scalability of the SkylineComm2D algorithm. Similar scalability results can also be observed in the other datasets with various types of attributes. The results are shown in Fig. 18. As can be seen, SkylineComm2D scales near linearly with varying m or n. This is because the number of 2D skyline communities is typically much smaller than n, and thus the complexity of SkylineComm2D is near linear w.r.t. $O(m+n)$. These results confirm the complexity analysis in Section 3.

Additional Exp-2: Efficiency of SkylineComm2D. Figs. 14 and 15 show the efficiency of the SkylineComm2D algorithm in the networks with correlated and anti-correlated attributes, respectively. As can be seen, in these two types of attributed networks, the running time of SkylineComm2D decreases with increasing k. This is because the graph size (i.e., the maximal k-core) decreases when k increases. Compared to the results shown in Exp-1 (Fig. 5), the running time of SkylineComm2D in the networks with correlated attributes is much less than that of SkylineComm2D in the networks with independent and anti-correlated attributes. For example, when $k = 5$, SkylineComm2D takes 0.96 seconds to find all skyline communities in the Flixster network with correlated attributes (Fig. 14(d)), while it consumes 3.45 and 50 seconds in the same network with independent and anti-correlated attributes respectively (Fig. 5(d) and Fig. 15(d)). This is because the number of skyline communities in the correlated attributed network is much smaller than the number of skyline communities in the independent or anti-correlated attributed network. These results are consistent with the complexity analysis of SkylineComm2D in Section 3.

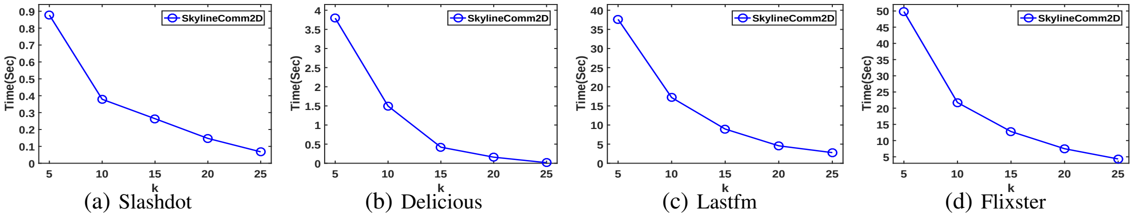
Additional Exp-3: Efficiency of Basic and New ($d = 3$). For $d = 3$, Figs. 16 and 17 report the efficiency of Basic and New in the networks with correlated and anti-correlated attributes, respectively. As can be observed, in both of these two types of datasets, New is at least one order of magnitude faster than Basic in most testings. Basic is intractable in the Lastfm and Flixster networks with anti-correlated attributes (Figs. 17(c) and (d)), but New still performs very well in these datasets. However, in all the networks with correlated attributes, both Basic and New work very well. This is because in the networks with correlated attributes, the number of skyline communities is not very large. These results further confirm the time complexity analysis of our algorithms in Sections 4 and 5.

Additional Exp-4: Efficiency of Basic and New (vary d). In this experiment, we vary d and evaluate Basic and New in the Delicious and Flixster networks with correlated and anti-correlated attributes respectively. Note that when $d = 2$, we refer to both Basic and New as the SkylineComm2D algorithm. The results are shown in Figs. 19. We can observe that the running time of our algorithms increases when d



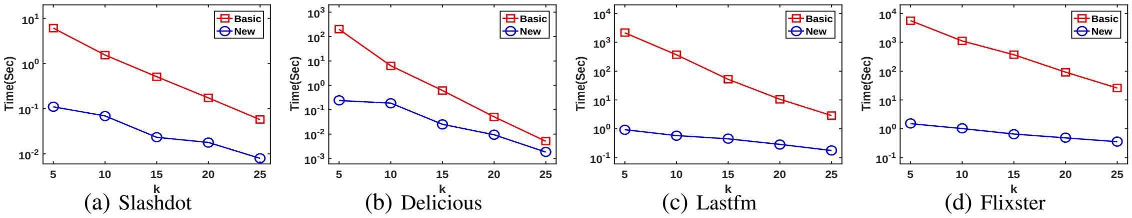
(a) Slashdot (b) Delicious (c) Lastfm (d) Flixster

Figure 14: Efficiency of SkylineComm2D in networks with correlated attributes (vary k)



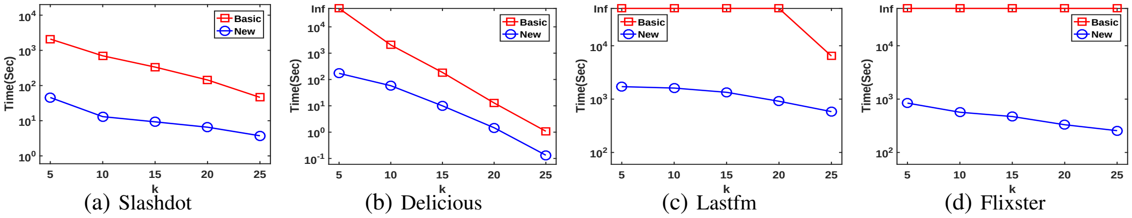
(a) Slashdot (b) Delicious (c) Lastfm (d) Flixster

Figure 15: Efficiency of SkylineComm2D in networks with anti-correlated attributes (vary k)



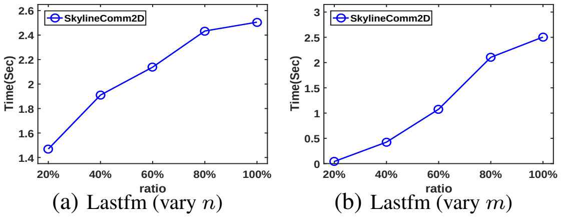
(a) Slashdot (b) Delicious (c) Lastfm (d) Flixster

Figure 16: Efficiency of Basic and New in networks with correlated attributes (vary k, $d = 3$)



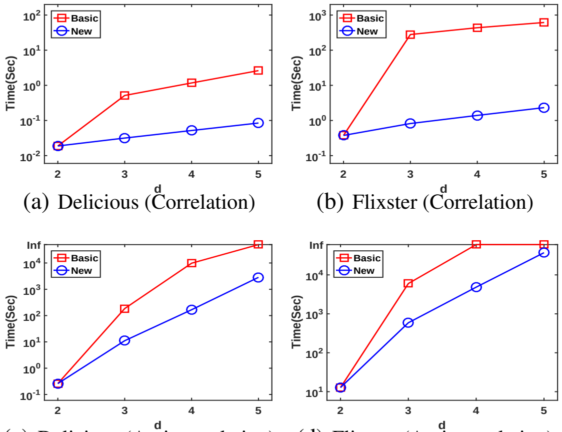
(a) Slashdot (b) Delicious (c) Lastfm (d) Flixster

Figure 17: Efficiency of Basic and New in networks with anti-correlated attributes (vary k, $d = 3$)



(a) Lastfm (vary n) (b) Lastfm (vary m)

Figure 18: Scalability of SkylineComm2D ($k = 15$)



(a) Delicious (Correlation) (b) Flixster (Correlation)

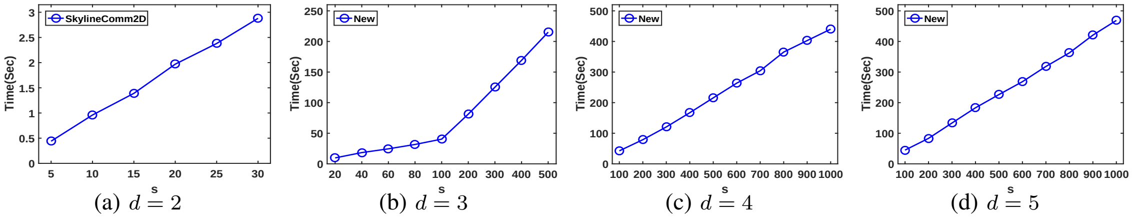
(c) Delicious (Anti-correlation) (d) Flixster (Anti-correlation)

Figure 19: Efficiency of Basic and New (Vary d, $k = 15$)

increases. Basic is intractable when $d \geq 4$ in the Flixster network with anti-correlated attributes. Also, New is at least one order of magnitude faster than Basic when $d \geq 3$. Similar to the results in the independent attributed networks, the running time of New increases by 10 times when d increases by 1 in the networks with anti-correlated attributes, as the number of skyline communities increases very fast with increasing d. However, in the networks with correlated attributes, both Basic and New work very well for all d, because in this case the number of skyline communities does not increases very quickly when d increases.

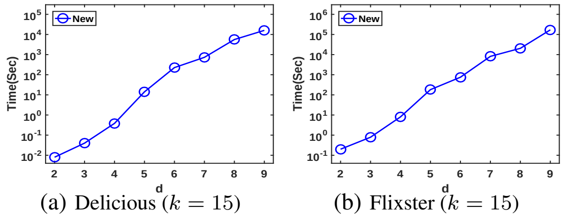
Additional Exp-5: Progressive performance. We evaluate the progressive performance of SkylineComm2D and New, as both are progressive algorithms. Fig. 20 shows the results in the Flixster network with independent attributes. From Fig. 20, we can observe that the running time of SkylineComm2D and New are proportional to the number of skyline communities. When $d = 2$ and $d = 3$, SkylineComm2D and New can progressively output all the results in less than 3 and 250 seconds respectively. When $d = 4$ or $d = 5$, the New algorithm finds 100 skyline communities in less than 50 seconds, and outputs 1000 skyline communities in less than 500 seconds in a 2.5 million node graph. These results demonstrate that our algorithm is very useful for applications that only need to find part of the skyline communities.

Additional Exp-6: Scalability of New for a given query (vary d). Here we study the scalability of New for a query-dependent skyline community search problem, where the problem is to find all the skyline communities that contain the query node. Note that we can first fix the query node, and



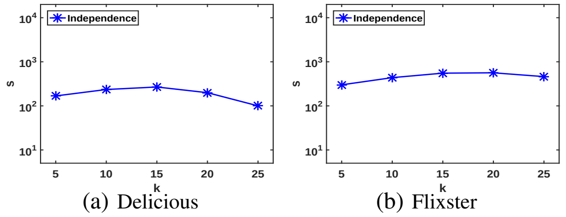
(a) $d = 2$ (b) $d = 3$ (c) $d = 4$ (d) $d = 5$

Figure 20: Progressive performance testing in the Flixster dataset ($k = 15$)



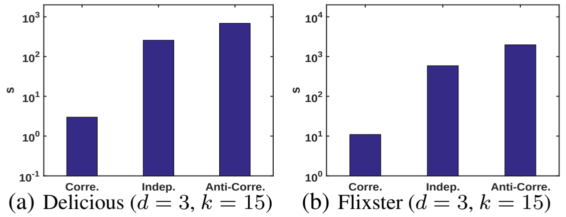
(a) Delicious ($k = 15$) (b) Flixster ($k = 15$)

Figure 21: Scalability of New for a given query (vary d)



(a) Delicious (b) Flixster

Figure 22: Number of skyline communities (vary k, $d = 3$)



(a) Delicious ($d = 3$, $k = 15$) (b) Flixster ($d = 3$, $k = 15$)

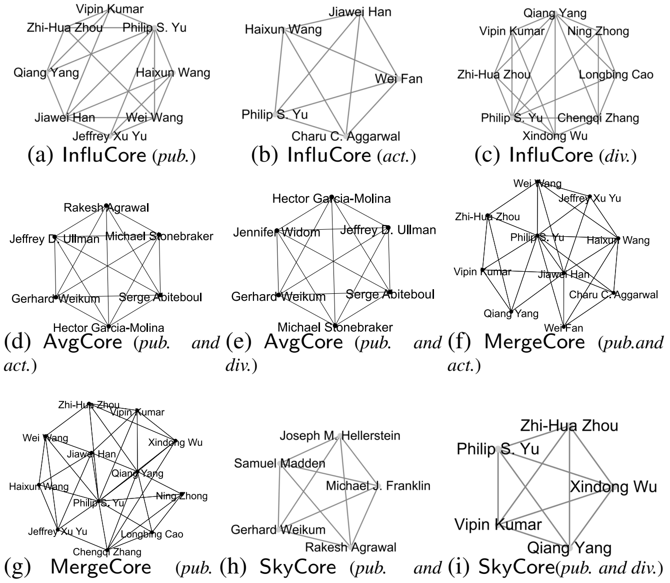
Figure 23: Number of skyline communities (various attributes)

then invoke the New algorithm to find all these communities. We randomly select 10 query nodes, and the results are the average running time of New over the 10 query nodes. Fig. 21 depicts the results with varying d (from 2 to 9) in the Delicious and Flixster datasets. Similar results can also be observed in the other datasets. From Fig. 21, we can see that when $d = 9$, New takes around 10^5 seconds in a large-scale graph with 2.5 million nodes and 7.9 million edges. Also, we can observe that New scales near linearly w.r.t. d. These results further indicate that the New algorithm is scalable to handle large real-world graphs.

Additional Exp-7: Number of skyline communities (Vary k). Fig. 22 reports the number of skyline communities (denoted by s) with varying k in the Delicious and Flixster datasets with independent attributes. Similar results can also be observed in the other datasets. As can be seen, if $k < 15$, s slightly increases with increasing k. However, if $k > 15$, s decreases when k increases. This is because, if k is large, the number of k-$\widehat{cores}$ may decrease with increasing k. These results are consistent with the results observed in Exp-3.

Additional Exp-8: Number of skyline communities (various attributes). Fig. 23 depicts the number of skyline communities in the Delicious and Flixster datasets with different types of attributes. As desired, the number of skyline communities in the network with anti-correlated attributes is the largest among all the three types of attributed network. Also, we can see that the number of skyline communities in the independent and anti-correlated attributed networks is at least one order of magnitude larger than the number of skyline communities in the correlated attributed network. These results confirm the efficiency results of our algorithms.

Additional Exp-9: Versatile team search in Aminer. We compare our algorithm with three baseline methods for versatile team search in the Aminer network. We use two sets of attributes which are $\mathcal{A} = \{$*the number of papers*, *activity*$\}$ and $\mathcal{B} = \{$ *the number of papers*, *diversity*$\}$. For the attribute set $\mathcal{A}$, we aim to find the teams from Aminer such that its members not only publish a large number of papers, but they are also active in recent years. Similarly, for set $\mathcal{B}$, our goal is to identify the teams, in which the members have numerous publications and diverse research interests. We set $k = 4$ in this case study, and similar results can be observed for other k values. For each numeric attribute, we also normalize the values of the nodes into the range $[0, 1]$. Figs. 24(a-c) report the results obtained by InfluCore based on the attributes *the number of papers*, *activity*, and *diversity* respectively. As desired, the team in Fig. 24(a) mainly comprises the researchers who publish a large number of papers. Similarly, the teams in Figs. 24(b-c) fucus mainly on the activity and diversity respectively. Note that unlike our previous case studies, the teams obtained by InfluCore on different attributes have overlapping members. Thus, the MergeCore method can obtain a connected team as shown in Figs. 24(f-g). The reason could be that the three numeric attributes used in this case study may be correlated with each other, and therefore some nodes may be contained in various top-1 influential communities with different attributes. Figs. 24(d-e) show the results by AvgCore. As can be seen, the two resulting teams w.r.t. the attribute sets $\mathcal{A}$ and $\mathcal{B}$ are highly similar. The team in Fig. 24(d) contains the researchers that have many publications and they are also active in recent years. The similar team in Fig. 24(e), however, cannot capture the diversity, as it mainly contains database researchers. Compared to AvgCore, our approach SkyCore can well capture the two attributes simultaneously. For example, in Fig. 24(i), the team obtained by SkyCore consists of the scholars that have a number of publications and diverse research topics (including machine learning and data mining). Compared to MergeCore, SkyCore tends to find more compact teams. Moreover, by our definition of skyline community, the results obtained by SkyCore can dominate the results obtained by MergeCore. These results demonstrate that our approach is more effective than the baseline methods for the application of versatile team search.



(a) InfluCore (*pub.*) (b) InfluCore (*act.*) (c) InfluCore (*div.*)

(d) AvgCore (*pub. and act.*) (e) AvgCore (*pub. and div.*) (f) MergeCore (*pub.and act.*)

(g) MergeCore (*pub. and div.*) (h) SkyCore (*pub. and act.*) (i) SkyCore(*pub. and div.*)

Figure 24: Comparison of different methods for team search