The VLDB Journal (2020) 29:1407–1432
https://doi.org/10.1007/s00778-020-00618-5

REGULAR PAPER

Finding skyline communities in multi-valued networks

Rong-Hua Li[1] · Lu Qin[2] · Fanghua Ye[3] · Guoren Wang[1] · Jeffrey Xu Yu[4] · Xiaokui Xiao[5] · Nong Xiao[3] · Zibin Zheng[3]

Received: 2 August 2019 / Revised: 4 March 2020 / Accepted: 12 May 2020 / Published online: 8 June 2020

Abstract

Given a scientific collaboration network, how can we find a group of collaborators with high research indicator (e.g., h-index) and diverse research interests? Given a social network, how can we identify the communities that have high influence (e.g., PageRank) and also have similar interests to a specified user? In such settings, the network can be modeled as a multi-valued network where each node has d ($d \geq 1$) numerical attributes (i.e., h-index, diversity, PageRank, similarity score, etc.). In the multi-valued network, we want to find communities that are not dominated by the other communities in terms of d numerical attributes. Most existing community search algorithms either completely ignore the 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. We develop an elegant space-partition algorithm to efficiently compute the skyline communities. Two striking advantages of our algorithm are that (1) its time complexity relies mainly on the size of the answer s (i.e., the number of skyline communities), and thus, it is very efficient if s is small; and (2) it can progressively output the skyline communities, which is very useful for applications that only require part of the skyline communities. In addition, we also develop three efficient graph reduction techniques to further speed up the proposed algorithms. Extensive experiments on both synthetic and real-world networks demonstrate the efficiency, scalability, and effectiveness of the proposed algorithm.

Keywords Community search · Skyline community · k-core · Multi-valued networks

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 [13,14,16,17,20,30]. 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 numerical attributes. Such numerical attributes can be obtained from the profiles of the nodes or the statistical information of the nodes computed by

✉ Guoren Wang
wanggrbit@126.com

Rong-Hua Li
lironghuabit@126.com

Lu Qin
Lu.Qin@uts.edu.au

Fanghua Ye
smartyfh@outlook.com

Jeffrey Xu Yu
yu@se.cuhk.edu.hk

Xiaokui Xiao
xkxiao@nus.edu.sg

Nong Xiao
zhzibin@mail.sysu.edu.cn

Zibin Zheng
xiaon6@mail.sysu.edu.cn

[1] Beijing Institute of Technology, Beijing, China

[2] University of Technology, Sydney, Australia

[3] Sun Yat-Sen University, Guangzhou, China

[4] Chinese University of Hong Kong, Hong Kong, China

[5] National University of Singapore, Singapore, Singapore

different network analysis methods (e.g., the degree, PageRank, influence, etc.). For example, in the Aminer scientific collaboration network (http://aminer.org), each author has several numerical attributes, including the number of published papers, h-index, activity, diversity, sociability, etc. Such network data are typically modeled as a multi-valued network where each node is associated with d ($d \geq 1$) numerical attributes.

Given a multi-valued network, how can we find the communities that are not *dominated* by the other communities in terms of d numerical attributes? For instance, consider a pair of numerical attributes (h-index, diversity) in the Aminer scientific collaboration network. How can we find a group of collaborators with high h-index and diverse research interests in the Aminer network? Similarly, consider two numerical attributes (#papers, activity). How can we find a community in the Aminer network so that its members not only have a number of publications, but also they are very active in their research areas in recent years?

Most existing community search algorithms either completely ignore the numerical attributes or only consider one numerical attribute of the nodes [20], and therefore, they cannot be directly used to answer these questions. A naive approach to address these questions is described as follows. First, we can compute the average value (or other linear combinations) over all d numeric attributes for each node in the multi-valued network. Then, based on the average value of each node, we apply the previous community search algorithm for one numerical attribute [20] to identify communities. This naive method, however, cannot fully capture all the interesting communities in the d-dimensional 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 [28] and skyline [5]. A skyline community is a maximal connected k-core (not necessary the maximal k-core as defined in [28]) that is not dominated by the other connected k-cores in the d-dimensional attribute space (the detailed definition can be found in Sect. 3). Except for finding interesting communities in a scientific collaboration network, our skyline community model can also be used for many other 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 wish to find a team so that its members not only have high h-index values, 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 h-index and activity. Similarly, by finding the skyline communities with three attributes including activity, the number of publications, and diversity, we can also 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, and thus, it is very difficult to determine which community is located on the skyline. To overcome this challenge, we develop several novel algorithms to find the skyline communities efficiently. The main contributions of this paper are summarized below.

New community model. We propose the skyline community model which can be applied to discover the communities that are not dominated by the other communities in a multi-valued network. To the best of our knowledge, the skyline

community model is the first community model for multi-valued networks, and our work is also the first to introduce skyline for community modeling.

Novel algorithms. We first develop an efficient algorithm, called SkylineComm2D, to find all the skyline communities in the 2D case, i.e., $d = 2$. The time complexity of SkylineComm2D is $O(s(m+n))$ where s denotes the number of 2D skyline communities (i.e., the answer size), and the space complexity of SkylineComm2D is $O(m+n+s)$, which is linear with respect to (w.r.t.) the graph and answer size. To handle the high-dimensional case (i.e., $d \geq 3$), we propose a basic algorithm and an elegant space-partition algorithm to find the skyline communities efficiently. Two striking features of the space-partition algorithm are that (1) its worst-case time complexity is dependent mainly on the answer size, and thus, it is very efficient when the answer size is not very large; and (2) it is able to progressively output the skyline communities during the execution of the algorithm, and therefore, it is very useful for applications that only require part of the skyline communities. Additionally, we also propose three new graph reduction techniques to further speed up the proposed algorithms. The general idea of our graph reduction techniques is that we first identify a connected k-core H based on some fast heuristic algorithms and then prune the unpromising nodes that are dominated by H, since those nodes cannot be contained in any skyline community.

Extensive experiments. We conduct extensive experiments over both synthetic and real-world networks to evaluate the proposed algorithms. The results show that SkylineComm2D is very efficient which takes less than 3.5 s to compute all the skyline communities in a real-world network with 2.5 million nodes and 7.9 million edges. The results also demonstrate the high efficiency and scalability of the space-partition algorithm. For example, in the same million-scale network, the space-partition algorithm is able to derive all the skyline communities within 500 s when $d = 3$. The space-partition algorithm is also scalable to a power-law random graph with more than 100 million edges even when $d = 6$. 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. Section 2 reviews the previous studies that are related to this work. We propose the skyline community model and formulate the skyline community search problem in Sect. 3. We develop an efficient algorithm to find all the 2D skyline communities in Sect. 4. The basic algorithm and the space-partition algorithm for the $d \geq 3$ case are proposed in Sects. 5 and 6, respectively. The graph reduction techniques are shown in Sect. 7. We report the experiments in Sect. 8 and conclude this paper in Sect. 9.

2 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 [8], k-core [13,22,28], k-truss [10,16,17,32], k-core-truss [23], maximal k-edge connected subgraph [1,6,34], k-plex [3,11], nucleus [27], quasi-clique [12], locally densest subgraph [26,31], query-biased density [33], 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. [14] proposed an attributed community model that is tailored to the graphs with textual attributes. Li et al. [20,21] introduced an influential community model, which takes the node's influence into consideration. More recently, several improved online search algorithms for the influential community computation problem had been proposed by Chen et al. [7] and Bi et al. [4]. Note that all those improved techniques are based on the sorted order of nodes by their influence values. Since each node has d ($d > 1$) numerical attributes for the skyline community search problem, there is no sorted ordering of the nodes by considering all d ($d > 1$) attributes. As a result, all the existing influential community search techniques cannot be used for the skyline community search problem with $d > 1$. We note that 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. [19] is a short version of this paper. In this paper, we extensively extend our previous work [19] and develop three new graph reduction techniques to further speed up the proposed algorithms. Moreover, the experimental evaluations have been substantially enhanced and all sections have been carefully revised.

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. [30] studied the community search problem in social networks based on the k-core model. Huang et al. [16] introduced a k-truss community model and proposed several efficient k-truss community search algorithms. Cui et al. [12] investigated an overlap community search problem based on the quasi-clique model. More recently, Huang et al. [17] proposed the closest truss community model to find the k-truss community with small diameter. An excellent survey on community search models and algorithms can be found in [15].

Skyline computation. The skyline computation problem was originally studied in the theory community. Kung et al. [18] 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. [5]. A large number of algorithms have since been devised to efficiently find the skyline under different settings [2,9,24,25,29]. For example, Papadias et al. [24] proposed an efficient algorithm for progressively finding the skyline. Sheng and Tao [29] proposed an external-memory algorithm to compute the skyline efficiently. Pei et al. [25] 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.

3 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 $X_v = (x_1^v, \ldots, x_d^v)$, where $X_v \in X$ and $x_i^v \in \mathbb{R}$. For convenience, we refer to the i-th dimension ($i = 1, \ldots, d$) as the x_i dimension. Suppose 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.

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-$\widehat{core}$.

Let $H = (V_H, E_H)$ be an induced subgraph of G. Following the definition of the influence value of a community in [20], we define the value of H on the x_i dimension (for $i = 1, 2, \ldots, d$) as

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

By Eq. (1), we define a *domination* relationship between two subgraphs, which will be used to define our skyline community model.

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 $i = 1, \ldots, d$, and there exists $f_i(H) < f_i(H')$ for a certain i, we call that H' dominates H, denoted by $H \prec H'$.

Based on Definition 1 , we propose a new community model, called skyline community, by using the concepts of k-core [28] and skyline [5].

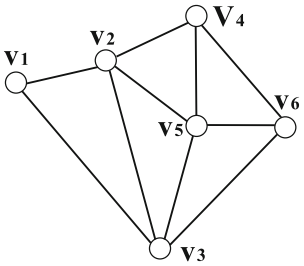
Definition 2 Given a multi-valued graph $G = (V, E, X)$ and an integer k. A skyline community with a parameter k is an induced subgraph $H = (V_H, E_H, X_H)$ of G such that it satisfies the following properties.

- 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 $i = 1, \ldots, d$.

Note that 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. 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.

In this paper, we aim at computing all the skyline communities as defined in Definition 2. Note that if $d = 1$, the skyline community search problem is equivalent to the problem of finding the top-1 influential community [20]. Therefore, if $d = 1$, we can use the algorithms proposed in [20] to find the skyline community efficiently. However, when $d > 1$, the



	x_1	x_2	x_3
v_1	8	14	7
v_2	9	15	12
v_3	11	17	3
v_4	6	16	4
v_5	7	8	5
v_6	10	9	10

Fig. 1 Running example

problem becomes much harder and the algorithms proposed in [20] 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 very costly to enumerate all the k-$\widehat{cores}$ to identify the skyline communities. Second, unlike the traditional d-dimensional skyline computation problem [5], 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.

4 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 $i = 1, \ldots, d$). For example, in the Aminer network, the numerical attributes such as h-index and the number of papers are positive. Note that if this assumption does not hold, we can revise x_i^u by $\tilde{x}_i^u = x_i^u - \min_{v \in V}\{x_i^v\} + \epsilon > 0$ for all $i = 1, \ldots, d$ and $u \in V$ which does not affect the correctness of all the proposed algorithms (ϵ is a positive constant).

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, and 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.

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.*

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 $x_1^u \leq f_1(H_1)$. Then, 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. □

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 Sects. 5 and 6), which cannot be deleted by the algorithm. To find all the 2D skyline communities, we can invoke

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 following example illustrates how Algorithm 2 works.

Example 2 Consider the graph shown in Fig. 1. To illustrate the SkylineComm2D algorithm, we consider two dimensions x_1 and x_3 for each node (i.e., the first dimension is x_1, and the second dimension is x_3). First, the algorithm computes the maximal f_3 by invoking the DimMax algorithm. In this example, it is easy to check that $f_3 = 4$. Then, the algorithm refines the constraint by $x_3 \geq 4$ (line 3) and calculates the maximal f_1 value. In this example, the maximal f_1 value is 6. Subsequently, the algorithm adds $(6, 4)$ into the answer set which corresponds to the skyline community $\{v_2, v_4, v_5, v_6\}$. After that, the algorithm refines the constraint by $x_1 > 6$, which corresponds to delete all the nodes with $x_1 \leq 6$. And then, the algorithm computes the *next* maximal f_3 based on the refined constraints. It is also easy to verify that $f_3 = 3$. Using the same procedure, we can obtain the next skyline community $\{v_1, v_2, v_3\}$ with values $(8, 3)$. Similarly, the algorithm refines the constraint in line 5 by $x_1 > 8$, and then the algorithm invokes DimMax to compute f_3. In this case, $f_3 = 0$; thus, the algorithm terminates.

Algorithm 1 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: $\quad H \leftarrow$ 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: $\quad$ Let $u \in H$ be the smallest-value node on the x_d dimension;
9: $\quad flag \leftarrow 1$; $flag \leftarrow$ DFS(u);
10: $\quad$ **if** $flag = 0$ **then break**;
11: $\quad$ **if** $\mathcal{F} \neq \emptyset$ **then**
12: $\quad\quad H \leftarrow$ the maximal k-$\widehat{core}$ in H that contains $\mathcal{F}$;
13: $\quad f_d \leftarrow \max\{f_d, f_d(H)\}$;
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: $\quad \delta(v, H) \leftarrow \delta(v, H) - 1$;
22: $\quad$ **if** $\delta(v, H) < k$ **then** DFS(v);
23: Delete u from H;
24: **return** 1;

Algorithm 2 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: $f_2 \leftarrow$ DimMax ($G, \mathcal{I}, \mathcal{F}, 2$); $\mathcal{R} \leftarrow \emptyset$;
2: **while** $f_2 > 0$ **do**
3: $\quad \tilde{\mathcal{I}} \leftarrow \mathcal{I}\bar{\cap}\{x_2 \geq f_2\}$; $f_1 \leftarrow$ DimMax ($G, \tilde{\mathcal{I}}, \mathcal{F}, 1$);
4: $\quad \mathcal{R} \leftarrow \mathcal{R} \cup \{(f_1, f_2)\}$;
5: $\quad \tilde{\mathcal{I}} \leftarrow \mathcal{I}\bar{\cap}\{x_1 > f_1\}$; $f_2 \leftarrow$ DimMax ($G, \tilde{\mathcal{I}}, \mathcal{F}, 2$);
6: **return** $\mathcal{R}$;

The correctness of Algorithm 2 is shown in the following theorem.

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 the 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 $H_1, H_2, \ldots, H_s$. 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 connected k-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 connected k-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 can 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 ($i = 1, \ldots, s-1$). 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 $i = 1, \ldots, s-1$ 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 $j = 1, \ldots, s-1$ and $j \neq i$ such that $f_2(H_j) > f_2(H) > f_2(H_{j+1})$. Assume without loss of generality that $i < j$. Then, we 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 connected k-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 to $f_2(H_i) > f_2(H) > f_2(H_{i+1})$. Putting it all together, we can conclude that Algorithm 2 outputs all 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.*

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 some 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. □

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, and thus, Algorithm 2 is very efficient in practice.

5 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.

5.1 Handling the $d = 3$ case

The dimension reduction idea. Our algorithm is based on a *dimension reduction* idea which involves three steps. First, we derive all the possible f_3 values that the skyline communities may have on the x_3 dimension. Second, for each possible f_3 value, denoted by f_3^*, we find all the 2D skyline communities on the x_1 and x_2 dimensions such that the f_3 values of these skyline communities equal f_3^*. Here we refer to a skyline community based on the x_1 and x_2 dimensions as a 2D skyline community, and all those based on three dimensions as 3D skyline communities. Finally, we merge the resulting skyline communities for all possible f_3 values, and invoke a traditional skyline algorithm [5,18] to determine all the 3D skyline communities.

Let F_3 be the set of all the possible f_3 values. For the first step, a naive solution is to set F_3 to be the set of all the x_3 values of the nodes in G, because the f_3 values of all the skyline communities must take from the set of all the x_3 values of nodes. The second step can be implemented as follows. 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 only one node u with $x_3^u = f_3^*$ can be fixed, because all the x_3 values form a total order by our assumption. We invoke SkylineComm2D with constraint $\mathcal{I} = \{x_3 \geq f_3^*\}$ and fixed-point set $\mathcal{F} = \{u\}$ to compute the 2D skyline communities on the x_1 and x_2 dimensions. It can be easily shown that the resulting communities are 2D skyline communities (on the x_1 and x_2 dimensions) with f_3 values equaling f_3^*.

An improved implementation. The naive implementation is inefficient because it needs to invoke the SkylineComm2D algorithm $|F_3| = n$ times. Here, we propose an improved

implementation based on an interesting connection between our problem and the influential community search problem [20].

Recall that the influential community model is tailored to the network with only one numerical attribute [20]. In a multi-valued network with d numerical attributes, the influential community can be defined on each dimension x_i ($i = 1, \ldots, d$). Specifically, on the x_i dimension, a community H is called an influential community [20] if (1) it is a connected k-core (i.e., k-$\widehat{core}$), and (2) there does not exist a k-$\widehat{core}$ H' such that H' contains H and $f_i(H') = f_i(H)$ (see Eq. 1). Let T_i be the set of values of all the influential communities defined on the x_i dimension. T_i can be computed using the influential community search algorithm described in Algorithm 3 [20]. In particular, Algorithm 3 iteratively deletes the smallest-value node on the x_i dimension, and records the f_i values of the current maximal k-$\widehat{core}$ which corresponds to the value of an influential community [20].

Note that both the skyline communities and influential communities are k-$\widehat{cores}$ satisfying a *maximal* property; and both the f_i values of the skyline communities and the values of the influential communities on the x_i dimension (i.e., T_i) are defined by the "min" operator (Eq. 1). Intuitively, the f_i values of the skyline communities should be contained in T_i because T_i consists of all the possible values of the *maximal* k-$\widehat{cores}$ defined by the "min" operator. Indeed, Lemma 2 shows that the f_i values of the skyline communities must be taken from T_i.

Lemma 2 *For each dimension x_i ($i = 1, \ldots, d$), 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 that $f_i(H) \notin T_i$. 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 $T_i = \{f_i^1, \ldots, f_i^t\}$ with $f_i^1 <, \cdots, < f_i^t$. Then, by definition, it is easy to show that $f_i^1 < f_i(H) < f_i^t$. Thus, there exists j ($1 \le j \le t-1$) 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 and it does not contain u, and thus, it must be contained in H_{j+1}, which is a contradiction. □

Algorithm 3 [20]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;

Algorithm 4 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: $F_3 \leftarrow$ 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: $\tilde{\mathcal{I}} \leftarrow \mathcal{I} \cap \{x_3 \ge f_3\}$;
5: $\mathcal{T} \leftarrow$ 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** Skyline($\mathcal{R}$);

Based on Lemma 2, we can set $F_3 = T_3$ in our algorithm. Since $|T_3| \le n$ and can be substantially smaller than n in practice [20], this improved implementation is much more efficient than the naive implementation.

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 $\{x_3 \ge f_3\}$ 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.

Example 3 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 $\mathcal{I} = \{x_3 \ge 3\}$ 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

Algorithm 5 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** Basic3D$(G, \mathcal{I}, \mathcal{F})$;
2: $F_d \leftarrow$ 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: $\tilde{\mathcal{I}} \leftarrow \mathcal{I} \cap \{x_d \geq f_d\}$;
6: $\mathcal{T} \leftarrow$ 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** Skyline$(\mathcal{R})$;

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.

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 $O(n^2 \log n)$ [5,18], 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 $O(n^2)$. □

5.2 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.*

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 [18] (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 $O(n^{d-1})$. □

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 well for many real-world networks as shown in the experiments.

5.3 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., $X_u = \{x_1^u, \ldots, x_d^u\}$. 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 $(x_1^u, \ldots, x_{d-1}^u)$ 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.

6 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.

6.1 Handling the $d = 3$ case

The key idea. The basic idea of our new algorithm is that we compute the skyline communities following the decreasing order of the f_3 values of the 3D skyline communities. In other words, we first compute the set of 3D skyline communities that have the largest f_3 value, and then calculate the 3D skyline communities having the second-largest f_3 value, etc. The challenge is how to implement this procedure without computing invalid skyline communities. Our solution is detailed as follows.

Let $\mathcal{H}$ be the set of 3D skyline communities that have the maximum f_3 value. $\mathcal{H}$ can be easily computed by the following procedure. First, we invoke the DimMax algorithm (Algorithm 1) with constraint $\mathcal{I} = \{x_1 > 0, x_2 > 0\}$ to derive the largest f_3 value in G, denoted by f_3^*. Then, we fix the node u with $x_3^u = f_3^*$ and invoke SkylineComm2D with constraint $\mathcal{I} = \{x_1 > 0, x_2 > 0\}$ and fixed-point set $\mathcal{F} = \{u\}$ to compute the 2D skyline communities on the x_1 and x_2 dimensions. Clearly, all the resulting communities are valid 3D skyline communities having the largest f_3 value.

Since f_3^* is maximum, the f_3 values of the remaining 3D skyline communities in G must be smaller than f_3^*. Hence, the (f_1, f_2) values of the remaining 3D skyline communities cannot be dominated by those of the skyline communities in $\mathcal{H}$. By the skyline property, all the (f_1, f_2) values of the 3D skyline communities in $\mathcal{H}$ form a *staircase-like shape* in the 2D space. For ease of understanding, we use an example shown in Fig. 2a to illustrate our idea. In this example, we have three 3D skyline communities in $\mathcal{H} = \{H_1, H_2, H_3\}$, and the label "*" denotes the 3D skyline communities on the x_1 and x_2 dimensions. Clearly, the space below the staircase-like shape is dominated by the skyline communities in $\mathcal{H}$ which can be safely pruned. The (f_1, f_2) values of the remaining 3D skyline communities must be located on the top of the staircase-like shape.

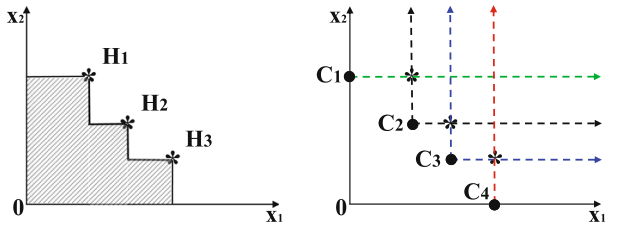
Obviously, the maximum f_3 value of the remaining 3D skyline communities is the second-largest f_3 value over all the 3D skyline communities. However, it is challenging to derive the maximum f_3 value of the remaining 3D skyline communities. This is because (1) the (f_1, f_2) values of the remaining 3D skyline communities are located on the top of the staircase-like shape which forms an irregular 2D space (see Fig. 2a), and (2) we cannot directly apply DimMax to compute the maximum f_3 value given that the (f_1, f_2) values are located in such an irregular 2D space.

To overcome this challenge, we propose a space-partition approach. The key step of our approach is to partition the *irregular* 2D space (the 2D space on the top of the staircase-like shape) into several overlapped *regular* 2D subspaces, in which the maximum f_3 value can be computed by DimMax. Formally, the regular 2D space is defined as follows.

Definition 3 Given two dimensions x_1 and x_2, a 2D space is called a regular 2D space if and only if it can be represented by a pair of constraints $(x_1 > f_1, x_2 > f_2)$, where (f_1, f_2) is a 2D point.

Note that the above definition of the regular 2D space can be directly extended to the high-dimensional case. Again, we use the example shown in Fig. 2 to illustrate the space-partition idea. In this example, the *irregular* 2D space in Fig. 2a is divided into four overlapped regular subspaces as shown in Fig. 2b where each 2D point C_i corresponds to a regular subspace.

For a regular 2D space represented by $(x_1 > f_1, x_2 > f_2)$, we can compute the maximum f_3 value in that space by invoking DimMax with constraint $\mathcal{I} = \{x_1 > f_1, x_2 > f_2\}$. As a result, we are able to derive the maximum f_3 value in the *irregular* 2D space, denoted by $\tilde{f}_3^*$, using such a space-partition method. Furthermore, we can also identify the regular 2D subspaces in which the maximum f_3 value achieves $\tilde{f}_3^*$. After obtaining $\tilde{f}_3^*$ and the corresponding regular 2D subspaces, the SkylineComm2D algorithm can be used to compute the 2D skyline communities in that regular 2D subspace. We claim that the computed 2D skyline communities are also the 3D skyline communities. The reasons are as follows. First, the (f_1, f_2) values of these 2D skyline communities cannot be dominated by the previously com-



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

Fig. 2 Illustration of the space-partition idea (color online)

Algorithm 6 The Space-Partition Framework

```
1: Let P be the initial 2D space represented by (x1 > 0, x2 > 0);
2: R ← ∅;
3: while P ≠ ∅ do
4:     S ← partition P into a set of overlapped regular subspaces;
5:     (f3*, T) ← identify the largest f3 value (f3*) and the corresponding regular
       subspaces (T) in S by the DimMax algorithm;
6:     H ← compute the set of 2D skyline communities in T by SkylineComm2D;
7:     R ← R ∪ H;
8:     P ← prune the 2D space dominated by H in P;
9: return R;
```

puted skyline communities (i.e., $\mathcal{H}$), because they are located on the top of the staircase-like shape formed by the already computed 3D skyline communities (based on the x_1 and x_2 dimensions). Second, since our algorithm computes the 3D skyline communities following the decreasing order of the f_3 values, the f_3 values of the undiscovered 3D skyline communities must be smaller than $\tilde{f}_3^*$. As a result, all the computed 2D skyline communities are valid 3D skyline communities. Once we obtain a set of new 3D skyline communities, we can iteratively use the same space-partition method to compute the remaining 3D skyline communities. The general framework of our space-partition method is shown in Algorithm 6.

To implement our framework, the remaining question is how can we divide the irregular 2D space into several overlapped regular 2D subspaces? Below, we define two important concepts called MIN skyline and corner point which will be used to partition the irregular 2D space.

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 $x = (x_1, \ldots, x_d) \in \mathcal{A}$, there does not exist a point $y = (y_1, \ldots, y_d) \in \mathcal{L} \backslash \mathcal{A}$ such that $y_i \leq x_i$ for all $i = 1, \ldots, d$ and $y_i < x_i$ for a certain $i = 1, \ldots, d$.

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. 2a. 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

Algorithm 7 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 R ← ∅; Priority Queue Q ← ∅; C ← {(0, 0)};
2: if DimMax(G, I, F, 3) > 0 then
3:     Q.Push((0, 0), DimMax(G, I, F, 3));
4: while Q ≠ ∅ do
5:     c3 ← Q.MaxVal(); S' ← ∅;
6:     while Q.MaxVal() = c3 do
7:         ((c1, c2), c3) ← Q.Pop(); {// c3 is the priority of (c1, c2) ∈ Q}
8:         Ĩ ← I ∩ {x1 > c1, x2 > c2};
9:         Let u be the node that x3^u = c3; F̃ ← F ∪ {u};
10:        S_tmp ← SkylineComm2D(G, Ĩ, F̃);
11:        S' ← S' ∪ S_tmp;
12:    for all (c1, c2) ∈ S' do
13:        R ← R ∪ {(c1, c2, c3)};
14:    for all s' ∈ S' do C ← UpdateCornerPoints(C, s', 2);
15:    for all (c1, c2) ∈ C do
16:        Ĩ ← I ∩ {x1 > c1, x2 > c2};
17:        if (c1, c2) ∉ Q and DimMax(G, Ĩ, F, 3) > 0 then
18:            Q.Push((c1, c2), DimMax(G, Ĩ, F, 3));
19: return R;
```

Algorithm 8 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:     C'_i ← ∅;
3:     for all (c = (x1, ..., xd)) ∈ C s.t. x_j < x'_j for 1 ≤ j ≤ d do
4:         C ← C \ {c}; replace x_i with x'_i in c;
5:         C'_i ← C'_i ∪ {c};
6:     C'_i ← Skyline(C'_i, d, MIN); {// computed by classic skyline algorithms}
7: return C ∪ C'_1 ∪ ... ∪ C'_d;
```

corner points which are labeled by "•" in Fig. 2b. Note that the coordinates of the corner points can be determined by the (f_1, f_2) values of the 3D skyline communities. For example, in Fig. 2b, the coordinates of the corner point C_3 can be determined by the 3D skyline communities H_2 and H_3, which are $(f_1(H_2), f_2(H_3))$. Based on the corner points, we can easily divide the irregular 2D space into several overlapped regular 2D subspaces as illustrated in Fig. 2b. Note that each corner point corresponds to a regular 2D subspace. For the corner point $C_3 = (f_1(H_2), f_2(H_3))$ for example, the corresponding regular 2D subspace can be represented by $(x_1 > f_1(H_2), x_2 > f_2(H_3))$.

Implementation details. The detailed implementation of our algorithm is shown in Algorithm 7. In Algorithm 7, we use a priority queue $\mathcal{Q}$ to maintain all the regular 2D subspaces where the priority of the subspace is the maximum f_3 value in that subspace. Specifically, in the priority queue $\mathcal{Q}$, we use a pair $((c_1, c_2), c_3)$ to denote a regular 2D subspace, where (c_1, c_2) denotes the corner point corresponding to the regular 2D 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 regular 2D space into $\mathcal{Q}$ (lines 1–3). Then, the algorithm iteratively computes the skyline communities based on the best-first strategy (lines 4–18). Note that

the algorithm can derive skyline communities following a decreasing order of the f_3 values based on the best-first strategy. In each iteration, the algorithm first finds the maximum priority from $\mathcal{Q}$ and sets c_3 as the maximum priority (line 5). The algorithm then iteratively pops the regular 2D space whose priority equals c_3 from $\mathcal{Q}$ (line 7). For a popped regular 2D 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). All the computed 2D skyline communities are recorded 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 8. 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 7) by invoking Algorithm 8. Clearly, if the previously 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 $x = (x_1, \ldots, x_d)$ completely dominating a point $y = (y_1, \ldots, y_d)$ if and only if $x_i > y_i$ for all $i = 1, \ldots, d$. For example, consider the corner points shown in Fig. 3a. 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. 3b. Algorithm 8 details this procedure. In Algorithm 8, we compute the MIN skyline in each dimension (line 7 in Algorithm 8), because the cross points 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.

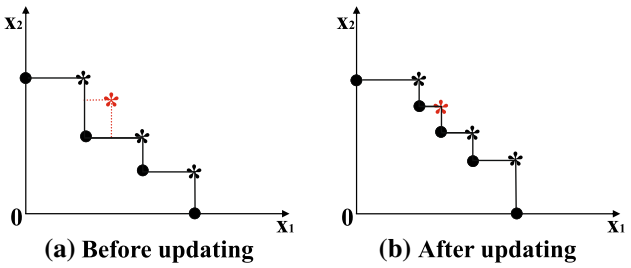


Fig. 3 Illustration of the corner points updating

After updating $\mathcal{C}$, Algorithm 7 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 7 works.

Example 4 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 regular space (i.e., $(x_1 > 0, x_2 > 0)$) 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 7 is analyzed below.

Theorem 5 *Algorithm 7 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 shows the complexity of Algorithm 7.

Theorem 6 *Let s be the number of 3D skyline communities. The worst-case time and space complexity of Algorithm 7 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 8, 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 7 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. □

An improved 3D algorithm. Due to the overlapped space-partition method, a skyline community may be recomputed in Algorithm 7 if its (f_1, f_2) values are located in two regular 2D subspaces with the same priority (see lines 6–11 in Algorithm 7). To avoid such redundant computations, we propose an improved algorithm to ensure that no skyline community is recomputed.

The skyline community clearly cannot be recomputed in two regular 2D subspaces with different priorities in Algorithm 7; thus, we need to avoid redundant computations when the regular 2D subspaces have the same priority. Let $\mathcal{P} = \{((c_1^1, c_2^1), c_3), \ldots, ((c_1^t, c_2^t), c_3)\}$ be the set of regular 2D spaces with the same priority c_3. Suppose without loss of generality that c_3 is the current maximum 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 maximum 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 maximum 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 2D spaces, which thus avoids recomputing the skyline communities.

Algorithm 9 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 R ← ∅; Priority Queue Q ← ∅; C ← {(0, 0)};
2: if DimMax(G, I, F, 3) > 0 then
3:     Q.Push((0, 0), DimMax(G, I, F, 3));
4: while Q ≠ ∅ do
5:     c3 ← Q.MaxVal(); c2' ← 0; S' ← ∅;
6:     while Q.MaxVal() = c3 do
7:         ((c1, c2), c3) ← Q.Pop();
           {// following the decreasing order of the c1 value}
8:         c2' ← max(c2', c2); Ĩ ← I ∩̃ {x1 > c1, x2 > c2'};
9:         Let u be the node that x3^u = c3; F̃ ← F ∪ {u};
10:        S_tmp ← SkylineComm2D(G, Ĩ, F̃);
11:        c2' ← max(c2', max{x2' | (x1', x2') ∈ S_tmp});
12:        S' ← S' ∪ S_tmp;
13:    for all (c1, c2) ∈ S' do
14:        R ← R ∪ {(c1, c2, c3)};
15:    for all s' ∈ S' do C ← UpdateCornerPoints(C, s', 2);
16:    for all (c1, c2) ∈ C do
17:        Ĩ ← I ∩̃ {x1 > c1, x2 > c2};
18:        if (c1, c2) ∉ Q and DimMax(G, Ĩ, F, 3) > 0 then
19:            Q.Push((c1, c2), DimMax(G, Ĩ, F, 3));
20: return 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. 4a. 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, three skyline communities $\mathcal{S}_{tmp} = \{H_1, H_2, H_3\}$ have been obtained in the regular space C_1. The algorithm updates c_2' by $f_2(H_1)$, because $f_2(H_1)$ is the largest value among all the f_2 values of 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 the regular space C_2. Due to the constraint $x_2 > c_2'$, the shading area in the regular space C_2 in Fig. 4b is pruned. As a result, the skyline communities H_1 and H_2 will not be recomputed in the second iteration. Algorithm 9 shows the details of our algorithm.

In lines 6–12 of Algorithm 9, since c_2' does not decrease, none of the computed skyline communities are recomputed in the subsequent iterations. On the other hand, a skyline

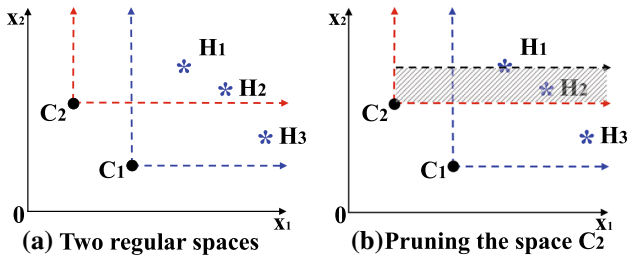


Fig. 4 Illustration of the idea of the improved algorithm

community also cannot be recomputed in spaces with different priorities (i.e., different c_3 values), and thus, each skyline community is only calculated once by Algorithm 9. We can apply a similar argument used in Theorem 5 to prove the correctness of Algorithm 9. Below, we analyze the time and space complexity of the algorithm.

Theorem 7 *The time and space complexity of Algorithm 9 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 9, the total time cost of computing all the skyline communities in line 10 is $O(s(m+n))$. Similar to Algorithm 7, the total number of corner points generated by the algorithm is $O(s)$, and 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 8 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 9 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, and thus, the total cost of computing the MIN skyline in each dimension (line 7 in Algorithm 8) is $O(s)$ time. In the 3D case, s is bounded by n^2, and 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 9 is $O(s(m+n))$. Second, we can easily show that the space complexity of Algorithm 9 is $O(m+n+s)$, which is the same as Algorithm 7. □

Note that the worst-case time complexity of Algorithm 9 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 9 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.

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

6.2 Handling the $d > 3$ case

We extend Algorithm 7 to handle the $d > 3$ case in Algorithm 10. The general procedure of Algorithm 10 is very similar to that of Algorithm 7. 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 9 cannot be applied to the $d > 3$ case. The reason is as follows. For the $d > 3$ case, the regular space is a $(d-1)$-dimensional space. For each regular $(d-1)$-dimensional space ($d > 3$), we cannot use a similar method to that illustrated in Fig. 4 to prune the dominated $(d-1)$-dimensional space. This is because if we prune the dominated $(d-1)$-dimensional space, the resulting space is no longer a regular $(d-1)$-dimensional space when $d > 3$. The correctness analysis of Algorithm 10 is also very similar to the analysis of Algorithm 7; 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 10 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 10 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 10 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. □

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 $d \leq 5$), and 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 10 is a progressive algorithm which is very useful for applications that require only part of the skyline communities.

Remark. It is worth remarking that all algorithms presented in this paper focus mainly on computing the d-dimensional f vectors for all skyline communities. Note that we can easily determine the skyline communities using such f vectors. Specifically, for each skyline community H, if we know $(f_1(H), \ldots, f_d(H))$, we are able to extract H from G by the following procedure. First, for any node $u \in G$ that satisfies $x_i^u < f_i(H)$ for a certain $i = 1, \ldots, d$, we delete u from G. This is because such a node u cannot be contained in H by definition. Then, we compute the connected k-core on the remaining graph which is exactly the skyline community H. Note that the total time cost of this procedure to derive all skyline communities is $O(s(m+n))$ (s is the number of skyline communities), which does not increase the time complexity of all the proposed algorithms.

7 Graph reduction techniques

In this section, we develop several graph reduction techniques to further speed up the proposed algorithms. The idea is that we first apply the proposed graph reduction techniques to prune the unpromising nodes which are definitely not contained in any skyline community. Then, we invoke the proposed skyline community search algorithms on the pruned graph to identify all skyline communities.

Our graph reduction techniques are based on the following idea. First, we identify a connected k-core (k-$\widehat{core}$) H from the multi-valued graph G using some fast heuristic algorithms. Then, we prune all the nodes that are *dominated* by H, since these nodes cannot be contained in any skyline community in terms of Definition 2. Note that a node u is dominated by H if and only if $x_i^u \leq f_i(H)$ for all $i = 1, \ldots, d$ and there exists $x_i^u < f_i(H)$ for a certain i. The remaining question is how can we quickly find a k-$\widehat{core}$ that can dominate as many nodes as possible. Below, we propose three efficient heuristic algorithms to achieve this goal.

Pruning rule 1. The first pruning algorithm is shown in Algorithm 11. For each node u, the algorithm first computes the minimum value $\underline{x}^u$ among the d-dimensional values of u (lines 1–2). Then, the algorithm calculates the 1D skyline community H on the graph G based on this *one-dimensional* value (line 3). Recall that when $d = 1$, there is only one 1D skyline community in a graph which is the *influential community* with the highest influence value [20]. Thus, the 1D skyline community can be computed in linear time (w.r.t. the graph size) using the algorithm proposed in [20]. Since H is a k-$\widehat{core}$, the algorithm prunes all nodes dominated by H (line 4).

The rationale of Algorithm 11 is described as follows. By taking the minimum value $\underline{x}^u$ as the *one-dimensional* value, the values of the k-$\widehat{core}$ H, i.e., $f_1(H), \ldots, f_d(H)$, are no less than the minimum value among all $\underline{x}^u$. As a result, many nodes having small values on d dimensions could be pruned by H. Moreover, the striking feature of Algorithm 11 is that the k-$\widehat{core}$ H is also a valid skyline community in the d-dimensional attribute space. Therefore, all the nodes that are dominated by such a skyline community H can be pruned by Algorithm 11.

Theorem 9 *The 1D skyline community H obtained by Algorithm 11 is a valid skyline community in the d-dimensional attribute space.*

Proof To prove the theorem, we need to show H satisfying the cohesive, skyline, and maximal properties as defined in Definition 2. First, H is clearly a connected k-core; thus, the cohesive property is satisfied. Second, we prove that H satisfies the skyline property. Suppose to the contrary that there is a k-$\widehat{core}$ H' that dominates H. Then, by definition, we have $f_i(H) \leq f_i(H')$ for all $i = 1, \ldots, d$, and there exists $f_i(H) < f_i(H')$ for a certain i. Let $\underline{f}(H) \triangleq \min_{u \in H}\{\underline{x}^u\}$. Since H is the 1D skyline community based on the minimum value $\underline{x}^u$ for each $u \in V$, we have $\underline{f}(H) > \underline{f}(H')$.

Algorithm 11 Rule1($G = (V, E), k$)

1: **for** each $u \in V$ **do**
2: $\quad \underline{x}^u \leftarrow \min\{x_1^u, x_2^u, \cdots, x_d^u\}$;
3: $H \leftarrow$ compute the 1D skyline community based on the value $\underline{x}^u$ for each $u \in V$;
4: $V_p \leftarrow$ the set of nodes dominated by H;
5: **return** the subgraph induced by $V \setminus V_p$;

Since $f_i(H) \leq f_i(H')$ for each $i \in \{1, \ldots, d\}$ (by assumption), we have $\min\limits_{i=1,\ldots,d}\{\min\limits_{u\in H}\{x_i^u\}\} \leq \min\limits_{i=1,\ldots,d}\{\min\limits_{u\in H'}\{x_i^u\}\}$. As a result, we have $\min\limits_{u\in H}\{\min\limits_{i=1,\ldots,d}\{x_i^u\}\} \leq \min\limits_{u\in H'}\{\min\limits_{i=1,\ldots,d}\{x_i^u\}\}$, and therefore $\underline{f}(H) \leq \underline{f}(H')$ by definition, which is a contradiction. Third, we show that H also meets the maximal property. Since H is a 1D skyline community, we have $\underline{f}(H) > \underline{f}(H')$ by definition. Assume that there is a k-$\widehat{core}$ H' with $f_i(H) = f_i(H')$ for each $i = 1, \ldots d$ that contains H. Then, we have $\min\limits_{u\in H}\{\min\limits_{i=1,\ldots,d}\{x_i^u\}\} = \min\limits_{u\in H'}\{\min\limits_{i=1,\ldots,d}\{x_i^u\}\}$. Therefore, we can obtain that $\underline{f}(H) = \underline{f}(H')$, which is a contradiction. □

We analyze the time and space complexity of Algorithm 11 as follows. First, the algorithm takes $O(nd)$ time to compute $\underline{x}^u$ for all $u \in V$. Second, the algorithm takes $O(m + n)$ time to compute the 1D skyline community H by using the algorithm proposed in [20]. Therefore, the total time cost of Algorithm 11 is $O(m+nd)$, which is linear with respect to (w.r.t.) the graph size. In addition, it is easy to show that the space overhead of Algorithm 11 is also linear w.r.t. the graph size.

Pruning rule 2. Our second pruning algorithm is described in Algorithm 12. Algorithm 12 iteratively identifies a k-$\widehat{core}$ to prune unpromising nodes. In each iteration, the algorithm first finds the set of nodes V_i that have the minimum value on the x_i dimension for each $i = 1, \ldots, d$ (lines 3–4). Then, the algorithm deletes all the nodes in V_i, and computes the maximal k-core H on the remaining graph (lines 5–7). After that, the algorithm prunes all the nodes that are dominated by H (lines 8–10). If there is no node that can be dominated by H (or the number of iterations is larger than a given threshold τ), the algorithm terminates (lines 8–9). The rationale of Algorithm 11 is described below. By iteratively removing the nodes that have the minimum value on a certain dimension, the algorithm can guarantee that the $f_i(H)$ value of the resulting k-core H is no less than the minimum value on the dimension x_i for each $i = 1, \ldots d$. As a consequence, many nodes with small values on each dimension are expected to be dominated by H, which will be pruned by Algorithm 12.

It can be shown that the time complexity of Algorithm 12 is $O(\tau(m + nd))$ in the worst case. This is because in each iteration, the algorithm takes $O(nd)$ time to determine V_i for each $i = 1, \ldots, d$ (lines 3–4) and $O(m+n)$ time to calculate the maximal k-core (line 7). Thus, the total time complexity of Algorithm 12 is $O(\tau(m + nd))$ in the worst case. Also, it is easy to derive that the space overhead of Algorithm 12 is liner w.r.t. the graph size.

Algorithm 12 Rule2($G = (V, E), k$)

1: $iter \leftarrow 0$;
2: **while** $iter < \tau$ **do**
3: $\quad$ **for** $i = 1$ **to** d **do**
4: $\quad\quad V_i \leftarrow \arg\min_{u\in V}\{x_i^u\}$;
5: $\quad V_{\min} \leftarrow \bigcup_{i\in\{1,\cdots,d\}} V_i$;
6: $\quad$ Let G' be the subgraph induced by $V \setminus V_{\min}$;
7: $\quad H \leftarrow$ compute the maximal k-core in G';
8: $\quad V_p \leftarrow$ the set of nodes in V that are dominated by H;
9: $\quad$ **if** $V_p = \emptyset$ **then break**;
10: $\quad V \leftarrow V \setminus V_p$; $iter \leftarrow iter + 1$;
11: **return** the subgraph induced by V;

Algorithm 13 Rule3($G = (V, E), k$)

1: $iter \leftarrow 0$;
2: **while** $iter < \tau$ **do**
3: $\quad V_s \leftarrow$ Skyline(V, d, MIN);
4: $\quad$ Let G' be the subgraph induced by $V \setminus V_s$;
5: $\quad H \leftarrow$ compute the maximal k-core in G';
6: $\quad V_p \leftarrow$ the set of nodes in V that are dominated by H;
7: $\quad$ **if** $V_p = \emptyset$ **then break**;
8: $\quad V \leftarrow V \setminus V_p$; $iter \leftarrow iter + 1$;
9: **return** the subgraph induced by V;

Pruning rule 3. Our third pruning technique is shown in Algorithm 13. Unlike the previous pruning rules, Algorithm 13 first deletes all the nodes that are located in the MIN skyline (see Definition 4) in the d-dimensional attribute space (lines 3–4), and then computes the maximal k-core H on the remaining graph (line 5). The algorithm iteratively performs this procedure to prune the multi-valued graph. In each iteration, the nodes dominated by H can be safely pruned, as those nodes cannot be contained in any skyline community (lines 6–8). If no node can be dominated by H, the algorithm terminates (line 7). The rationale of Algorithm 13 is that the nodes located in the MIN skyline are likely dominated by a skyline community; thus, they are more likely pruned by H.

The time complexity of Algorithm 13 is analyzed as follows. First, the algorithm takes $O(n \log^{d-2} n)$ time to compute the MIN skyline using the algorithm proposed in [18]. Second, the algorithm consumes $O(m + n)$ time to calculate the maximal k-core. Since there are at most τ iterations, the total time costs of Algorithm 13 is $O(\tau(m + n \log^{d-2} n))$ in the worst case. Similar to the previous pruning algorithms, we can easily derive that the space cost of Algorithm 13 is also linear w.r.t. to the graph size.

8 Experiments

We conduct comprehensive experiments to evaluate the proposed model and algorithms. For the $d = 2$ case, we implement two algorithms: SkylineComm2D (Algorithm 2) and NewPrune. NewPrune is the SkylineComm2D algorithm equipped with all pruning techniques developed in

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

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

Sect. 7. For the $d \geq 3$ case, we implement three algorithms: Basic (Algorithm 5), New (Algorithm 10), and NewPrune. Here NewPrune is the New algorithm with all pruning techniques developed in Sect. 7. Note that for the pruning rules 2 and 3 developed in Sect. 7, we set the parameter τ to be a sufficient large number so that the algorithm terminates with no node can be pruned. We will study the effect of the parameter τ in Sect. 8.3. In the Basic algorithm, we have integrated the pruning rule proposed in Sect. 5.3. 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 [5] to generate the numerical attributes for our datasets. We use the same method proposed in [5] 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.

8.1 Performance studies for $d = 2$

Exp-1: Efficiency of the 2D algorithms. We vary the core number k from 5 to 25 and evaluate the efficiency of the SkylineComm2D and NewPrune algorithms. The results on the networks with independent attributes are shown in Fig. 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. 5d, when $k = 15$, SkylineComm2D takes 2.8 s to output all the skyline communities, whereas it only uses 2.15 s if $k = 25$. Also, we can see that the running time of NewPrune is not very sensitive w.r.t. k. Moreover, NewPrune is much faster than SkylineComm2D on all the datasets due to the powerful pruning rules developed in Sect. 7. For instance, in Fig. 5d, NewPrune takes 0.2 seconds to compute all skyline communities given that $k = 15$, while SkylineComm2D takes 2.8 s using the same parameter setting. Note that on all datasets, SkylineComm2D and NewPrune take less than 4 and 0.5 s to output all the results, respectively. These results indicate that both SkylineComm2D and NewPrune are very efficient in practice, which confirm our complexity analysis shown in Sects. 4 and 7.

Figures 6 and 7 show the efficiency of our 2D algorithms on the networks with correlated and anti-correlated attributes, respectively. As can be seen, on these two types of attributed networks, the running time of SkylineComm2D decreases with increasing k, because the graph size (i.e., the maximal k-core) decreases as k increases. Similarly, the running time of NewPrune is much less than that of SkylineComm2D on all datasets. Compared to the results shown in Fig. 5, the running time of SkylineComm2D and NewPrune on the networks with correlated attributes is much less than the running time of SkylineComm2D and NewPrune on the networks with independent and anti-correlated attributes, respectively. For example, when $k = 5$, SkylineComm2D takes 0.96 s to find all skyline communities on the Flixster network with correlated attributes (Fig. 6d), while it consumes 3.45 and 50 s on the same network with independent and anti-correlated attributes, respectively (Figs. 5d and 7d). 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 shown in Sects. 4 and 7.

Exp-2: Memory overhead of the 2D algorithms. We show the memory costs of SkylineComm2D and NewPrune with varying k on the independent attributed networks. Similar results can be observed on the other types of attributed networks. Figure 8 shows our results. From Fig. 8, we can see that the memory overheads of both SkylineComm2D and NewPrune decrease with increasing k on all datasets. This

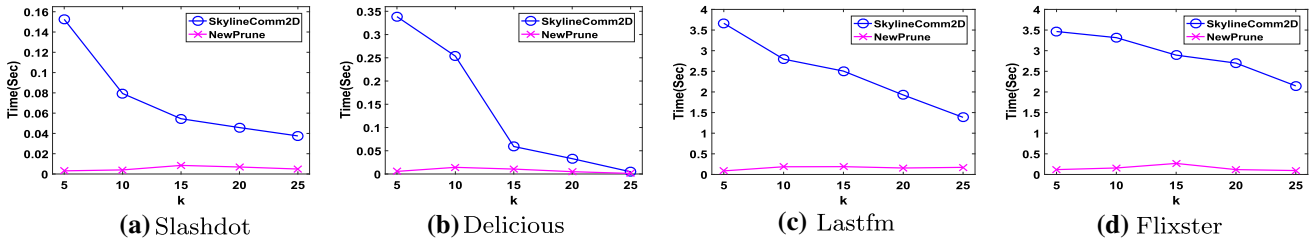


Fig. 5 Efficiency of SkylineComm2D and NewPrune in networks with independent attributes (vary k)

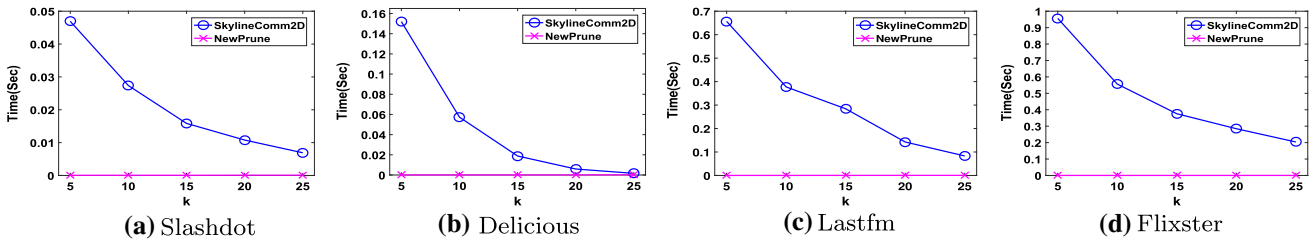


Fig. 6 Efficiency of SkylineComm2D and NewPrune in networks with correlated attributes (vary k)

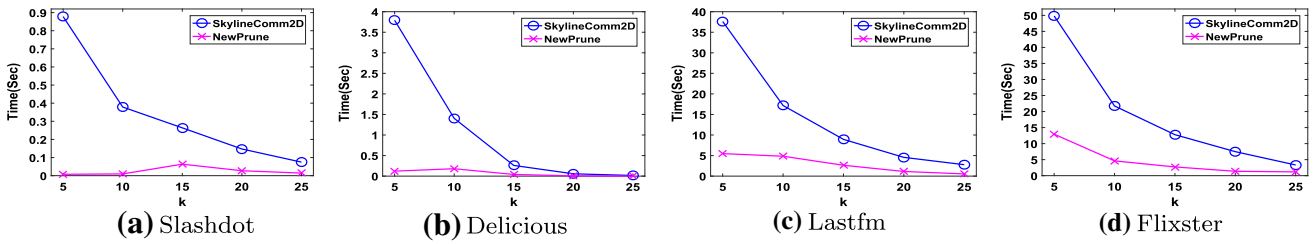


Fig. 7 Efficiency of SkylineComm2D and NewPrune in networks with anti-correlated attributes (vary k)

is because the graph size decreases as k increases. Also, we can see that the memory costs of SkylineComm2D and NewPrune are at most 3 times the graph size, indicating that both SkylineComm2D and NewPrune are memory-efficient. These results are consistent with the space complexity of the 2D algorithms.

Exp-3: Scalability of the 2D algorithms. We vary the number of nodes (n) and edges (m) on the Lastfm network with independent attributes to evaluate the scalability of the SkylineComm2D and NewPrune algorithm. Similar scalability results can also be obtained on the other datasets with various types of attributes. The results are shown in Fig. 9. As can be seen, both SkylineComm2D and NewPrune scale near linearly with varying n or m. This is because the number of 2D skyline communities is typically much smaller than n, and thus, the complexity of SkylineComm2D and NewPrune is near linear w.r.t. the graph size. These results confirm the complexity analysis in Sects. 4 and 7.

8.2 Performance studies for $d \geq 3$

Exp-4: Efficiency ($d = 3$). Fig. 10 shows the efficiency results of Basic, New, and NewPrune on the networks with independent attributes when $d = 3$. As can be seen, if $k \geq 15$, the running time of Basic, New, and NewPrune decreases as k increases. However, if $k < 15$, the running time slightly increases when k increases. On all the datasets, New is at least one order of magnitude faster than Basic, and NewPrune is around 3 times faster than New. For instance, in Fig. 10a, when $k = 20$, NewPrune takes 1.7 s, New takes 5.3 s, whereas Basic takes 100.6 s. Also, as shown in Fig. 10c, d, Basic is very time consuming on the Lastfm and Flixster datasets ("Inf" means that the algorithm cannot terminate in 50,000 s). Both New and NewPrune, however, still run very fast on these datasets. For example, in Fig. 10d, New and NewPrune only take 200 and 80 s to find all 3D skyline communities on Flixster, respectively. These results confirm our theoretical analysis in Sects. 5, 6, and 7.

For $d = 3$, Figs. 11 and 12 report the efficiency of different algorithms on the networks with correlated and anti-correlated attributes, respectively. As can be observed, on both of these two types of datasets, New is at least one order of

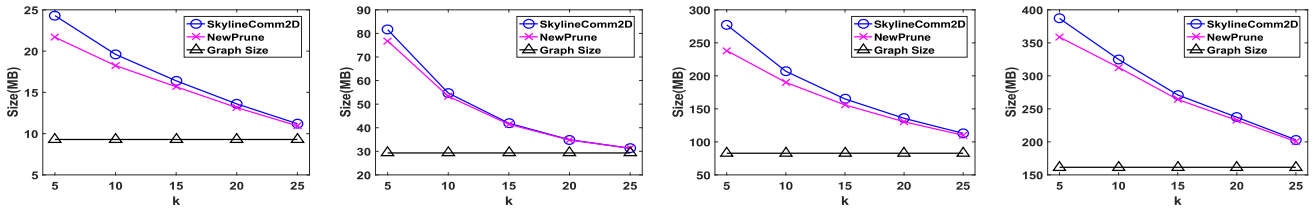


Fig. 8 Memory overheads of SkylineComm2D and NewPrune in networks with independent attributes (vary k)

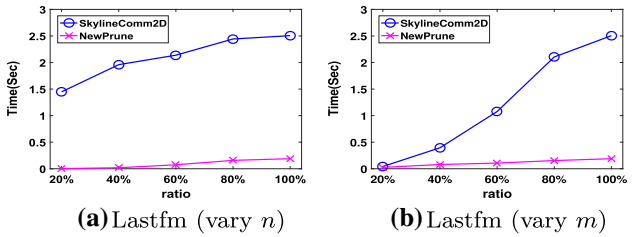


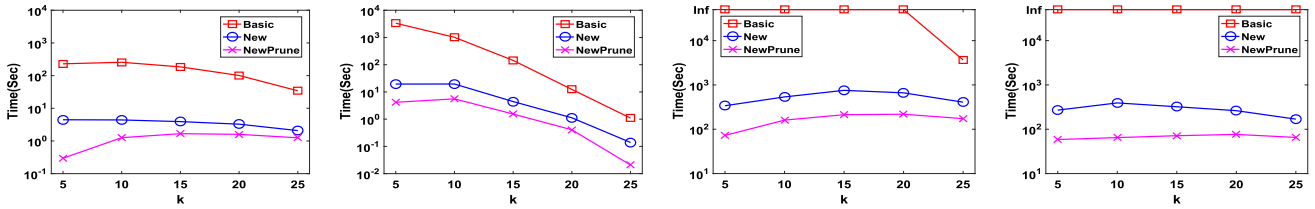
Fig. 9 Scalability of SkylineComm2D and NewPrune ($k = 15$)

magnitude faster than Basic in most testings, and NewPrune is at least twice faster than New. Basic is very costly on the Lastfm and Flixster networks with anti-correlated attributes (Fig. 12c, d), but New and NewPrune still perform very well on these datasets. However, on all the networks with correlated attributes, all three algorithms work very well. This is because the number of skyline communities in the networks with correlated attributes is not very large. Furthermore, we can see that NewPrune runs extremely fast on the networks with correlated attributes which is three orders of magnitude faster than New. The reason is that our pruning rules developed in Sect. 7 can substantially prune the original multi-valued network if its attributes are dependent. These results further confirm the time complexity analysis of our algorithms in Sects. 5, 6, and 7.

Exp-5: Memory overhead. Figure 13 depicts the memory costs of Basic, New, and NewPrune on the networks with independent attributes given that $d = 3$. Similar results can also be observed in different types of attributed networks and also for the other d values. As can be seen, the space usage of NewPrune is comparable to that of New. Both New and NewPrune consume much less memory than Basic on all datasets. This is because Basic needs to maintain a large number of invalid skyline communities. Generally, the memory size of Basic, New and NewPrune decreases with increasing k. When $k = 25$, the space costs of New and NewPrune are close to that of the graph size, as these algorithms significantly reduce the graph size when k is large. These results demonstrate that both New and NewPrune are memory-efficient, which are consistent with the space complexity analysis shown in Sects. 6 and 7.

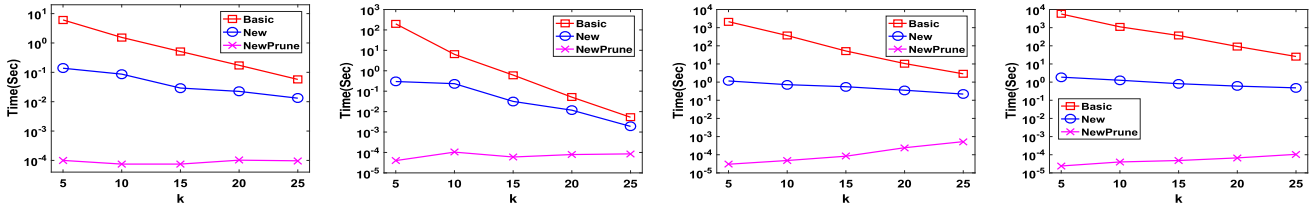
Exp-6: Efficiency (Vary d). We evaluate the efficiency of Basic, New, and NewPrune by varying d from 2 to 5. Note that when $d = 2$, both Basic and New are referred to as the SkylineComm2D algorithm. The results on the Delicious and Flixster networks are reported in Fig. 14. Similar results can be observed on the other datasets. As desired, the running time of all algorithms increases as d increases. NewPrune is consistently faster than New on all datasets, and both New and NewPrune are at least one order of magnitude faster than Basic when $d \geq 3$. As shown in Fig. 14c, d, NewPrune is at least two orders of magnitude faster than New on the networks with correlated attributes, which are consistent with our previous results. Also, we can see that all algorithms work very well for all d on the networks with correlated attributes, because the number of skyline communities in these networks does not increase very quickly when d increases. The Basic algorithm is typically very expensive, and it is even impractical when $d \geq 3$ on the Flixster network with independent or anti-correlated attributes. For both New and NewPrune, their running time typically increase by 10 times when d increases by 1 on the network with independent or anti-correlated attributes. The reason is that the number of skyline communities increases quickly when d increases by 1 on these networks. 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 often much smaller than the total number of numeric attributes. Therefore, in this sense, our New and NewPrune algorithms are tractable for handling most real-world applications.

Exp-7: Scalability. We evaluate the scalability of New and NewPrune when $d \geq 5$. To this end, we vary n and m on the Lastfm network with independent attributes. The results for other types of datasets are consistent. The results for $d = 6$ are reported in Fig. 15. Similar curves can also be observed for other d values. As can be seen, the running time of New and NewPrune increases smoothly with varying m and n,



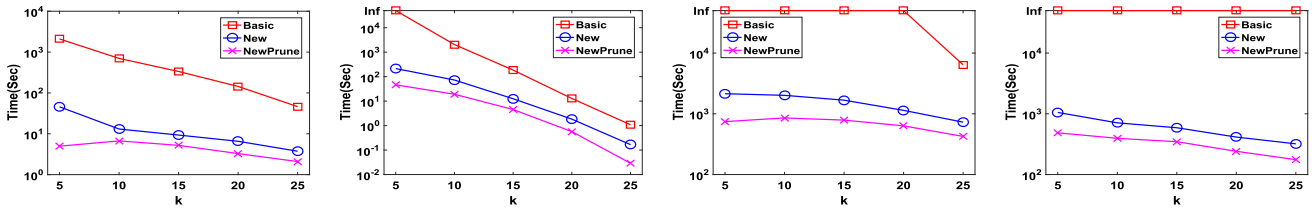
(a) Slashdot **(b)** Delicious **(c)** Lastfm **(d)** Flixster

Fig. 10 Efficiency of Basic, New, and NewPrune in networks with independent attributes (vary k, $d = 3$)



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

Fig. 11 Efficiency of Basic, New, and NewPrune in networks with correlated attributes (vary k, $d = 3$)



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

Fig. 12 Efficiency of Basic, New, and NewPrune in networks with anti-correlated attributes (vary k, $d = 3$)

implying that our algorithms scale well w.r.t. the graph size. These results indicate that both New and NewPrune are scalable to handle large real-world graphs given that $d = 6$. Again, as d is often very small (e.g., $d \leq 5$), our algorithms are scalable to handle most real-world applications.

Exp-8: Number of skyline communities. We show the number of skyline communities identified by our algorithms with varying d. The results on Delicious and Flixster networks with independent attributes are depicted in Fig. 16. Similar results can also be observed on the other datasets. From Fig. 16, we can see that the number of skyline communities, denoted by s, increases by 10 times when d increases by 1. These results are consistent with the efficiency results of our algorithms.

Figure 17 shows the number of skyline communities (denoted by s) with varying k on the Delicious and Flixster datasets with independent attributes. As can be seen, if $k < 15$, s slightly increases with increasing k. However, if $k > 15$, s decreases as k increases. This is because, if k is large, the number of k-$\widehat{cores}$ may decrease with increasing k.

Figure 18 depicts the number of skyline communities on the Delicious and Flixster datasets with different types of attributes. As desired, the number of skyline communities on the network with anti-correlated attributes is the largest among all the three types of attributed networks. Also, we can see that the number of skyline communities on 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 further confirm the efficiency results of our algorithms.

Exp-9: Progressive performance. Recall that our best algorithm NewPrune can progressively output skyline communities. In this experiment, we evaluate the progressive performance of NewPrune. Figure 19 shows the results on the Flixster network with independent attributes when $d = 2$ and $d = 5$. The results for the other d values are consistent. From Fig. 19, we can see that the running time of NewPrune is proportional to the number of skyline communities. When $d = 2$, NewPrune can progressively output all the results in less than 0.5 s. When $d = 5$, the NewPrune algorithm finds 100 skyline communities in less than 25 s, and outputs 1000 skyline communities in less than 200 s in a 2.5 million nodes

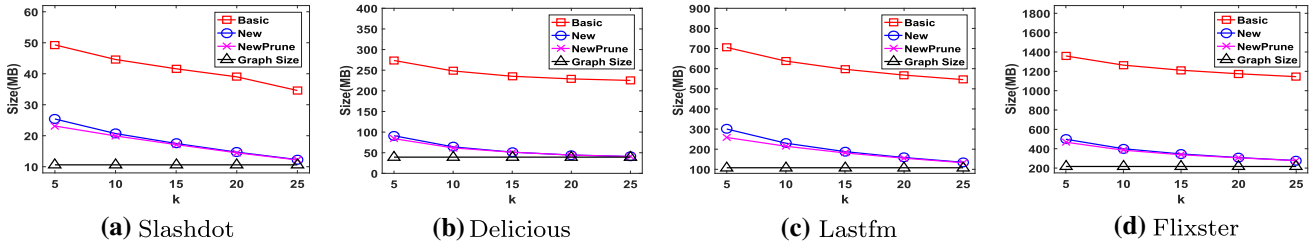


Fig. 13 Memory overheads of Basic, New, and NewPrune in networks with independent attributes (vary k, $d = 3$)

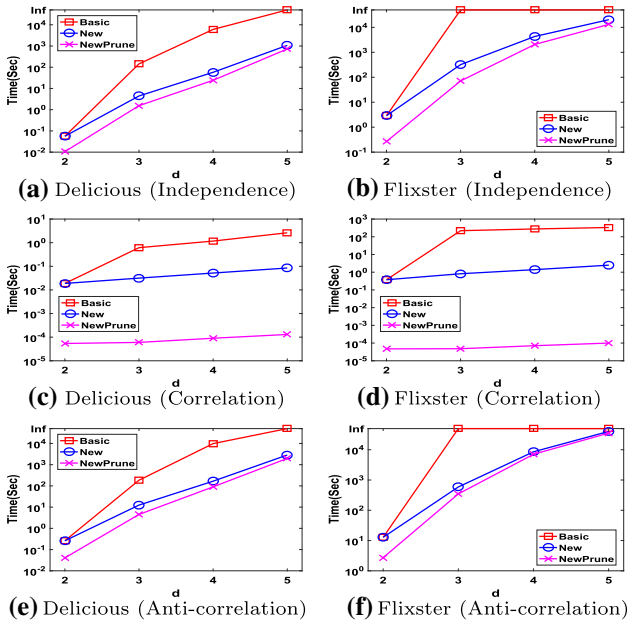


Fig. 14 Efficiency of Basic, New, and NewPrune ($k = 15$)

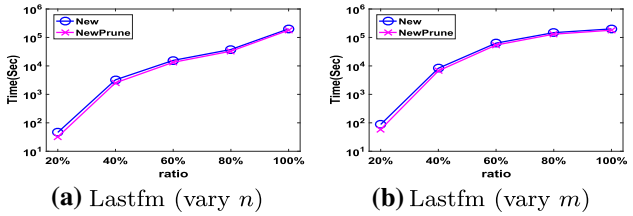


Fig. 15 Scalability of New and NewPrune ($k = 15$, $d = 6$)

graph. These results demonstrate that our algorithm is very efficient for applications that only need to find part of the skyline communities.

Exp-10: Results on power-law random graphs. Here we evaluate the performance of our algorithms on large-scale power-law random graphs. To this end, we generate a power-law graph with $n = 5$M and $m = 7.3$M and a set of power-law graphs with more than 10 million nodes and edges. All these power-law graphs are generated by a random graph generator developed in SNAP (http://snap.stanford.edu) with a power-law degree exponent $\gamma = 2.5$. For each node in the power-law graph, we randomly generate d independent numerical attributes using a uniform distribution. The results are reported in Fig. 20. From Fig. 20a, all three algorithms are scalable to a million-scale graph for a large d value (e.g., $d = 10$). The running time of Basic, New, and NewPrune increases as d increases. Figure 20b shows that the running time of all algorithms decreases with increasing k. Generally, New is around one order of magnitude faster than Basic, and NewPrune is at least twice faster than New under most parameter settings. Figure 20c, d shows the running time of New and NewPrune with varying m. As can be seen, both New and NewPrune exhibit very good scalability performance with respect to m. Even when $d = 6$, New and NewPrune take round 20,000 and 10,000 s in a graph with 100 million edges, respectively. These results further demonstrate the high efficiency and scalability of the proposed algorithms.

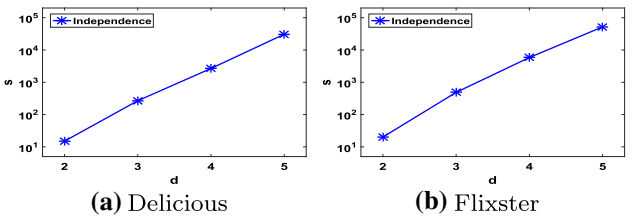


Fig. 16 Number of skyline communities (vary d, $k = 15$)

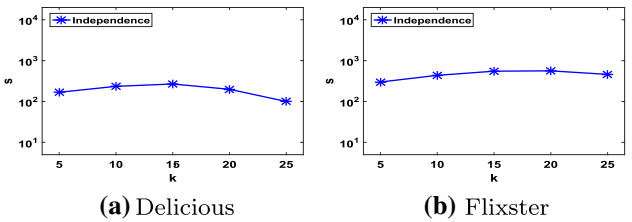
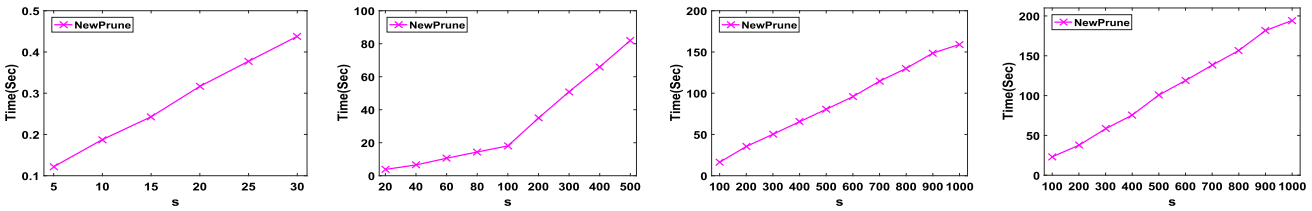


Fig. 17 Number of skyline communities (vary k, $d = 3$)

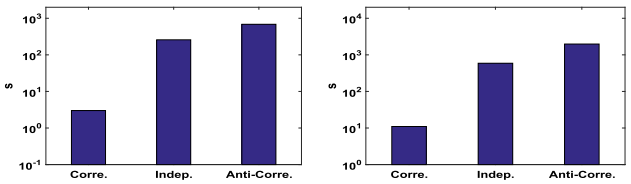
8.3 Results on a graph with real numeric attributes

In this subsection, we evaluate our algorithms using an additional dataset Aminer which has real number attributes. The Aminer dataset is a scientific collaboration network which is downloaded from http://aminer.org. Specifically, the Aminer



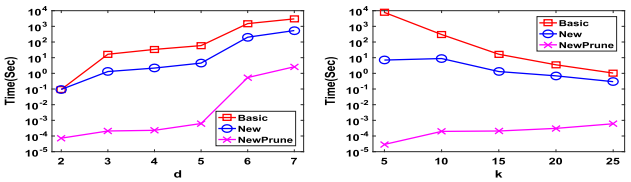
(a) $d = 2$ **(b)** $d = 3$ **(c)** $d = 4$ **(d)** $d = 5$

Fig. 19 Progressive performance testing on the Flixster dataset ($k = 15$)



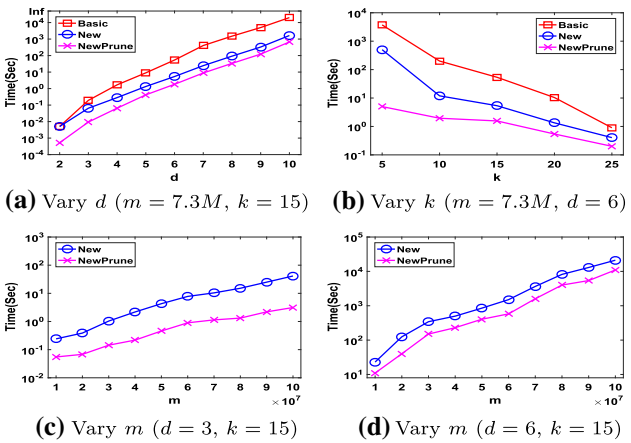
(a) Delicious ($d = 3$, $k = 15$) **(b)** Flixster ($d = 3$, $k = 15$)

Fig. 18 Number of skyline communities (various attributes)



(a) Vary d ($k = 15$) **(b)** Vary k ($d = 3$)

Fig. 21 Efficiency results on the Aminer dataset



(a) Vary d ($m = 7.3M$, $k = 15$) **(b)** Vary k ($m = 7.3M$, $d = 6$)

(c) Vary m ($d = 3$, $k = 15$) **(d)** Vary m ($d = 6$, $k = 15$)

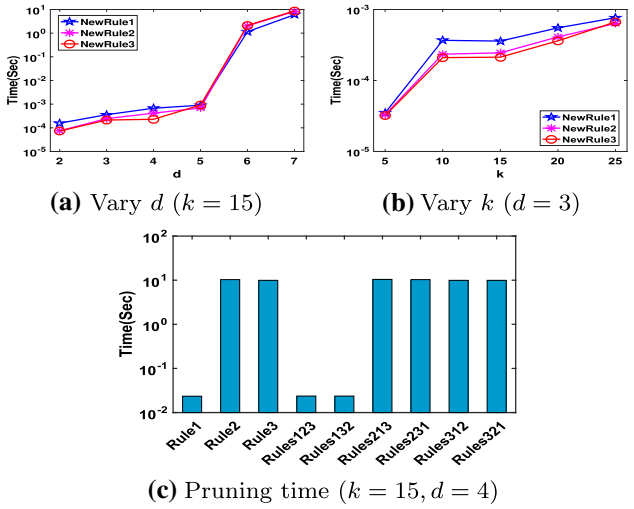
Fig. 20 Results on the power-law random graphs

dataset contains 1,712,433 authors and 4,258,615 collaboration relationships, where each node in Aminer has 7 numeric attributes (*#papers*, *#citations*, *h-index*, *g-index*, *sociability*, *diversity*, and *activity*).

Exp-11: Efficiency results on Aminer. Figure 21 shows the runtime of Basic, New, and NewPrune on Aminer with varying d and k. As expected, NewPrune is consistently faster than Basic and New with varying k or d values, which further confirms the high effectiveness of our pruning techniques. Moreover, we can see that when $d \leq 5$, NewPrune is around 3 orders of magnitude faster than New. The reason could be that the first 5 attributes of Aminer ((*#papers*, *#citations*, *h-index*, *g-index*, *sociability*) are highly correlated, and thus, our pruning rules can prune a large number of unpromising nodes. Such results are consistent with the results shown in Fig. 14c, d.

Exp-12: Comparison of different pruning rules. Here, we compare the performance of different pruning rules. We refer to NewRule1, NewRule2, NewRule3 as the New algorithm integrated with the pruning rules 1, 2, and 3, respectively. Figure 22a, b shows the runtime of NewRule1, NewRule2, NewRule3 with varying d and k. As can be seen, the runtime of NewRule1, NewRule2, NewRule3 are comparable, indicating that all three pruning rules exhibit similar pruning effect for the New algorithm. Figure 22c shows the runtime of different pruning rules. As expected, Rule1 takes much less time than Rule2 and Rule3, but its pruning power should be lower than that of Rule2 and Rule3 as indicated in Fig. 22a, b. Figure 22c also shows how the order of applying different rules affects the pruning performance. As can be seen, when we first apply Rule1 followed by Rule2 or Rule3 (i.e., Rules123 and Rules132 in Fig. 22c), we can achieve the best pruning performance. The reason is that Rule1 is much faster than Rule2 and Rule3, thus using such a *lightweight* rule first to prune the graph and then invoking Rule2 and Rule3 on the pruned graph can significantly reduce the time overhead. It is worth noting that in our experiments, the NewPrune algorithm first applies Rule1 followed by Rule2 and then Rule3 for pruning.

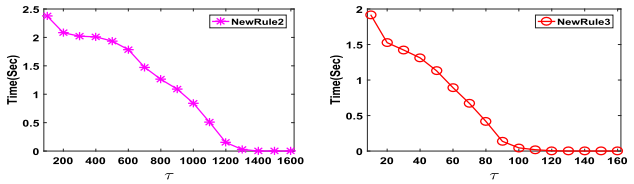
Exp-13: Effect of the parameter τ. Here, we study the effect of the parameter τ in NewRule2 and NewRule3. From Fig. 23, we can observe that the runtime of NewRule2 or NewRule3 decreases as τ increases, suggesting that a larger τ is more preferable for both NewRule2 and NewRule3. Recall that a large τ may increase the pruning costs of both Rule2 and Rule3, but the benefits are that many unpromising nodes can be pruned by Rule2 and Rule3 with a large τ. The results in Fig. 23 indicate that such benefits are greater than the prun-



(a) Vary d $(k = 15)$ **(b)** Vary k $(d = 3)$

(c) Pruning time $(k = 15, d = 4)$

Fig. 22 Comparison of different pruning rules on Aminer



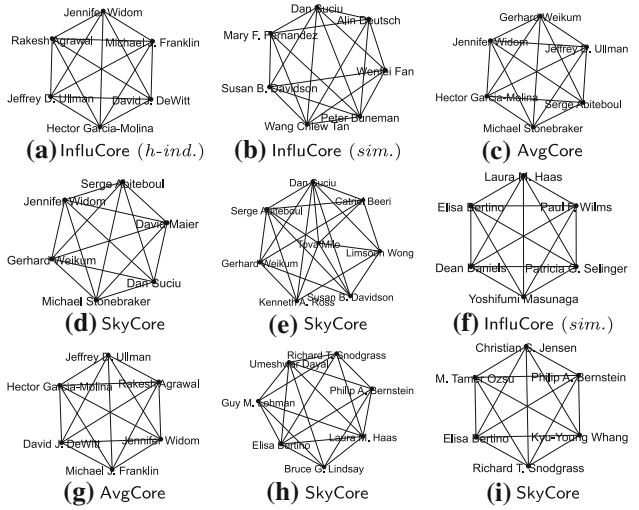
(a) Vary τ $(k = 15, d = 4)$ **(b)** Vary τ $(k = 15, d = 4)$

Fig. 23 Effect of the parameter τ on Aminer

ing costs for the NewRule2 and NewRule3 algorithms. This is why we set τ to be a sufficient large number for both Rule2 and Rule3 in all our experiments.

8.4 Case studies

We use two real-world datasets for case studies: AminerSmall and Gowalla. The AminerSmall dataset is a subgraph of the Aminer dataset used in Sect. 8.3 which contains the authors in database, data mining, machine learning, and information retrieval areas. AminerSmall comprises 5,411 nodes and 17,477 edges. We select 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 (i.e., the skyline community model), denoted by SkyCore, with three baseline methods: InfluCore, AvgCore, and MergeCore. InfluCore denotes the influential community search algorithm [20] which only considers one numeric attribute. AvgCore first takes the average value over the d



(a) InfluCore $(h\text{-}ind.)$ **(b)** InfluCore $(sim.)$ **(c)** AvgCore

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

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

Fig. 24 Comparison of different methods on the AminerSmall dataset $(k = 5)$; Fig. (**a**–**e**) shows the results of the query "Prof. Dan Suciu," and Fig. (**a**, **f**–**i**) depicts the results of the query "Prof. Elisa Bertino"

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. Below, we conduct three different case studies to evaluate the proposed method.

Exp-14: Finding similar and influential communities. 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. Figure 24a–e reports the results of professor Dan Suciu's communities, and Fig. 24a, f–i shows the results of professor Elisa Bertino's communities. In Fig. 24a, b, f, we can see that the results obtained by InfluCore only capture one attribute. For example, in Fig. 24a, 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. 24b, 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,

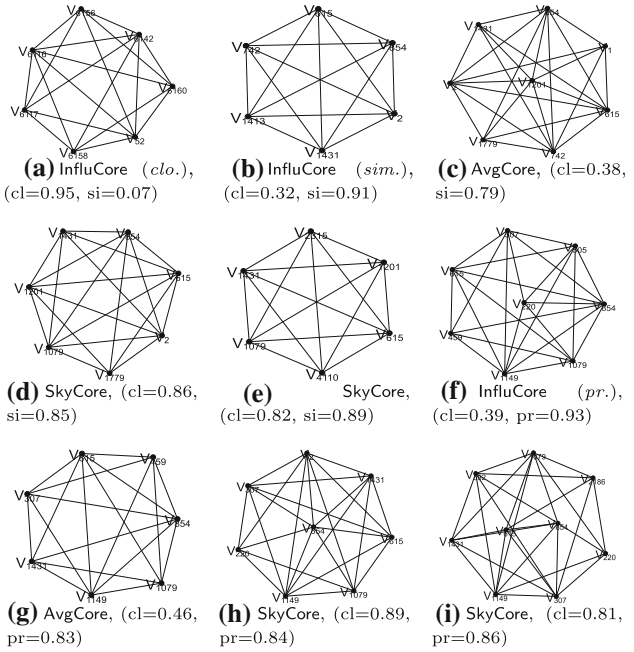


Fig. 25 Comparison of different methods on the Gowalla dataset with a query node "v_{615}"($k = 5$); Figs. (**a**–**e**) use two attributes: closeness and similarity ; Figs. (**f**–**i**) use two attributes: closeness and PageRank; "cl," "si," and "pr," denote the closeness, similarity, and PageRank values, respectively

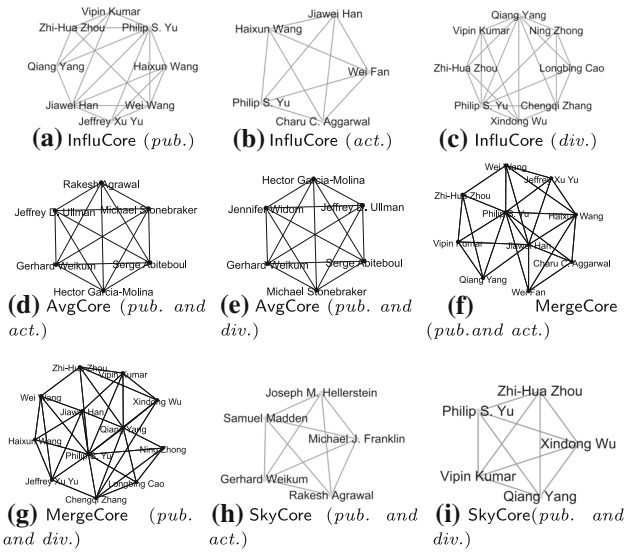


Fig. 26 Comparison of different methods for team search on the AminerSmall dataset

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 Fig. 24c, g, the resulting communities obtained by AvgCore also cannot capture both influence and similarity. For example, in Fig. 24g, 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 Fig. 24d, e, h, i, our approach (SkyCore) performs much better than all the baseline methods. For example, in Fig. 24d, 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-15: 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. 25. Similar results can be observed by the other query nodes. In Fig. 25, 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. For example, in Fig. 25a, the community contains the nodes that are close to the query node v_{615}, but the nodes are not similar to v_{615}. As can be seen, 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 Fig. 25c, d, SkyCore dominates InfluCore in both closeness and similarity. Likewise, when using the closeness and PageRank attributes, SkyCore is also the winner among all the algorithms as shown in Fig. 25f–i. These results indicate that the skyline community approach is very effective in applications of finding "close and similar" communities (or "close and influential" communities) in LBSN.

Exp-16: Versatile team search in AminerSmall. We compare our algorithm with three baseline methods for versatile team search on the AminerSmall 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]. Figure 26a–c reports the results obtained by InfluCore based on the attributes *the number of papers*, *activity*, and *diversity*, respectively. As desired, the team in Fig. 26a mainly comprises the researchers who publish a large number of papers. Similarly, the teams in Fig. 26b, c focus 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 Fig. 26f, 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. Figure 26d, e shows 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. 26d contains the researchers that have many publications, and they are also active in recent years. The similar team in Fig. 26e, 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. 26i, the team obtained by SkyCore consists of the scholars that have many 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.

9 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. In addition, we also develop three new graph reduction techniques to further accelerate the proposed algorithms. Extensive experiments in both real-world and synthetic multi-valued networks demonstrate the efficiency, scalability, and effectiveness of our solutions.

Acknowledgements Rong-Hua Li was partially supported by the NSFC Grants 61772346 and U1809206. Lu Qin was supported by ARC DP 160101513. Jeffrey Xu Yu was supported by the Research Grants Council of the Hong Kong SAR, China No. 14221716. Xiaokui Xiao was partially supported by NUS, Singapore, under an SUG grant.

References

1. Akiba, T., Iwata, Y., Yoshida, Y.: Linear-time enumeration of maximal k-edge-connected subgraphs in large networks by random contraction. In: CIKM, pp. 909–918 (2013)
2. Asudeh, A., Thirumuruganathan, S., Zhang, N., Das, G.: Discovering the skyline of web databases. PVLDB **9**(7), 600–611 (2016)
3. Berlowitz, D., Cohen, S., Kimelfeld, B.: Efficient enumeration of maximal k-plexes. In: SIGMOD (2015)
4. Bi, F., Chang, L., Lin, X., Zhang, W.: An optimal and progressive approach to online search of top-k influential communities. PVLDB **11**(9), 1056–1068 (2018)
5. Börzsönyi, S., Kossmann, D., Stocker, K.: The skyline operator. In: ICDE, pp. 421–430 (2001)
6. Chang, L., Yu, J.X., Qin, L., Lin, X., Liu, C., Liang, W.: Efficiently computing k-edge connected components via graph decomposition. In: SIGMOD, pp. 205–216 (2013)
7. Chen, S., Wei, R., Popova, D., Thomo, A.: Efficient computation of importance based communities in web-scale networks using a single machine. In: CIKM (2016)
8. Cheng, J., Ke, Y., Fu, A.W.C., Yu, J.X., Zhu, L.: Finding maximal cliques in massive networks. ACM Trans. Database Syst. **36**(4), 21:1–21:34 (2011)
9. Chomicki, J., Godfrey, P., Gryz, J., Liang, D.: Skyline with presorting. In: ICDE, pp. 717–719 (2003)
10. Cohen, J.: Trusses: Cohesive subgraphs for social network analysis. Technical report, National Security Agency (2005)
11. Conte, A., Firmani, D., Mordente, C., Patrignani, M., Torlone, R.: Fast enumeration of large k-plexes. In: KDD (2017)
12. Cui, W., Xiao, Y., Wang, H., Lu, Y., Wang, W.: Online search of overlapping communities. In: SIGMOD, pp. 277–288 (2013)
13. Cui, W., Xiao, Y., Wang, H., Wang, W.: Local search of communities in large graphs. In: SIGMOD, pp. 991–1002 (2014)
14. Fang, Y., Cheng, R., Luo, S., Hu, J.: Effective community search for large attributed graphs. PVLDB **9**(12), 1233–1244 (2016)
15. Fang, Y., Huang, X., Qin, L., Zhang, Y., Zhang, W., Cheng, R., Lin, X.: A survey of community search over big graphs. VLDB J. **29**, 353–392 (2020)
16. Huang, X., Cheng, H., Qin, L., Tian, W., Yu, J.X.: Querying k-truss community in large and dynamic graphs. In: SIGMOD pp. 1311–1322 (2014)
17. Huang, X., Lakshmanan, L.V.S., Yu, J.X., Cheng, H.: Approximate closest community search in networks. PVLDB **9**(4), 276–287 (2015)
18. Kung, H.T., Luccio, F., Preparata, F.P.: On finding the maxima of a set of vectors. J. ACM **22**(4), 469–476 (1975)
19. Li, R., Qin, L., Ye, F., Yu, J.X., Xiao, X., Xiao, N., Zheng, Z.: Skyline community search in multi-valued networks. In: SIGMOD (2018)
20. Li, R., Qin, L., Yu, J.X., Mao, R.: Influential community search in large networks. PVLDB **8**(5), 509–520 (2015)
21. Li, R., Qin, L., Yu, J.X., Mao, R.: Finding influential communities in massive networks. VLDB J. **26**(6), 751–776 (2017)
22. Li, R., Yu, J.X., Mao, R.: Efficient core maintenance in large dynamic graphs. IEEE Trans. Knowl. Data Eng. **26**(10), 2453–2465 (2014)
23. Li, Z., Lu, Y., Zhang, W., Li, R., Guo, J., Huang, X., Mao, R.: Discovering hierarchical subgraphs of k-core-truss. Data Sci. Eng. **3**(2), 136–149 (2018)

24. Papadias, D., Tao, Y., Fu, G., Seeger, B.: An optimal and progressive algorithm for skyline queries. In: SIGMOD, pp. 467–478 (2003)
25. Pei, J., Jiang, B., Lin, X., Yuan, Y.: Probabilistic skylines on uncertain data. In: VLDB, pp. 15–26 (2007)
26. Qin, L., Li, R., Chang, L., Zhang, C.: Locally densest subgraph discovery. In: KDD, pp. 965–974 (2015)
27. Sariyüce, A.E., Seshadhri, C., Pinar, A., Çatalyürek, Ü.V.: Finding the hierarchy of dense subgraphs using nucleus decompositions. In: WWW (2015)
28. Seidman, S.B.: Network structure and minimum degree. Soc. Netw. **5**(3), 269–287 (1983)
29. Sheng, C., Tao, Y.: On finding skylines in external memory. In: PODS, pp. 107–116 (2011)
30. Sozio, M., Gionis, A.: The community-search problem and how to plan a successful cocktail party. In: KDD, pp. 939–948 (2010)
31. Tatti, N., Gionis, A.: Density-friendly graph decomposition. In: WWW (2015)
32. Wang, J., Cheng, J.: Truss decomposition in massive networks. PVLDB **5**(9), 812–823 (2012)
33. Wu, Y., Jin, R., Li, J., Zhang, X.: Robust local community detection: on free rider effect and its elimination. PVLDB **8**(7), 798–809 (2015)
34. Zhou, R., Liu, C., Yu, J.X., Liang, W., Chen, B., Li, J.: Finding maximal k-edge-connected subgraphs from a large graph. In: EDBT, pp. 480–491 (2012)