Author/s:
Li, M;Borovica-Gajic, R;Choudhury, FM;Cui, N;Ding, L

Title:
Maximal size constraint community search over bipartite graphs

Date:
2024-08-03

Citation:
Li, M., Borovica-Gajic, R., Choudhury, F. M., Cui, N. & Ding, L. (2024). Maximal size constraint community search over bipartite graphs. Knowledge-Based Systems, 297, https://doi.org/10.1016/j.knosys.2024.111961.

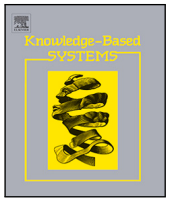
Persistent Link:
https://hdl.handle.net/11343/355748

Knowledge-Based Systems 297 (2024) 111961

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys



Maximal size constraint community search over bipartite graphs

Mo Li [a], Renata Borovica-Gajic [b], Farhana M. Choudhury [b], Ningning Cui [c], Linlin Ding [a,*]

[a] School of Information, Liaoning University, Shenyang, 110036, China
[b] School of Computing and Information Systems, The University of Melbourne, Melbourne, VIC 3010, Australia
[c] School of Data Science and Artificial Intelligence, Chang'an University, Xi'an, 710064, China

ARTICLE INFO

Keywords:
Community search
Bipartite graph
Size constraint
Vertex ordering

ABSTRACT

Community search over bipartite graphs is a crucial issue with significant applications in various domains, including group recommendations, fraud detection and representation learning. Current research primarily focuses on the structural cohesiveness constraint between two sets of vertices, often neglecting the community's size constraint. Unfortunately, this oversight may lead to communities of sizes ranging from excessively small to substantially large, thereby affecting their applicability. Therefore, in this paper, we focus on the maximal size constraint community search over bipartite graphs, which not only takes into account the (α, β) degree constraints for each set of vertices but also ensures that the number of vertices in each set is limited to ξ and ζ, respectively. To solve this problem, we first introduce two competitive algorithms, namely Ordering-based Expansion Algorithm (OEA) and Followers-based Peeling Algorithm (FPA), wherein they greedily expand or peel vertices, respectively, to identify the maximal one in a breadth-first manner, utilizing a novel vertex ordering and followers concept. To further accelerate the computation, we propose an advanced Expand-and-Filter Algorithm (EFA++) that first acquires a slightly larger community and subsequently executes a filtering phase employing a dynamic programming technique. Furthermore, we define the "Affected Area" for edge insertion or deletion that caters for dynamic scenarios. Finally, the theoretical analysis and extensive experimental evaluation on several real-life datasets demonstrate the effectiveness and efficiency of the proposed algorithms.

1. Introduction

Bipartite graphs are widely adopted to model the relationships between two distinct sets of entities, such as networks of users and locations, or authors and their publications [1–3]. Community structures are inherent within these practical networks, highlighting the importance of the community search problem. This problem has captivated researchers across various fields, owing to its broad applications ranging from personalized recommendations, social event organization to the representation learning [4,5]. Among them, (α, β)-core has gained recent attention [6–8], due to its potential to delineate more complex community structures within bipartite graphs. Specifically, an (α, β)-core is characterized as the largest subgraph in which each vertex in the upper layer connects to at least α neighbors, while each vertex in the lower layer connects to at least β neighbors.

The majority of current research on community search within bipartite graphs [8–10] has concentrated on analyzing the network's structural aspects and weight constraints. For instance, Zhang et al. [10] focused on the Pareto-optimal (α, β)-community, which considers both the degree constraints α, β and the weight of vertices in bipartite graphs. Meanwhile, Wang et al. [9] studied the significant (α, β)-community search over edge-weighted bipartite graphs. However, because these communities are defined by user-selected parameters α and β, there is a significant variation in the size of the resultant communities, ranging from excessively large to impracticably diminutive dimensions, which may limit their practical applications. Moreover, the size constraint of communities over bipartite graphs has not yet been explored, but attempts have been made to address the issue in unipartite graphs. For example, Ma et al. [11] investigated the problem of a size-constraint k-core group query in social networks, aiming to find a user group with size h containing the query user that constitutes a k-core structure. Yao et al. [12] designed exact algorithms for the size-bounded community search that aim to find a subgraph with the largest min-degree among all subgraphs containing the query vertex and satisfying the size interval.

Applications. In bipartite graphs, the size constraint is an important factor in many applications, such as event organization and GNN information aggregation. For instance, the scenario of event organization: envision an academic exchange platform wherein researchers and seminars are represented as a bipartite graph, with the edges symbolizing

* Corresponding author.
E-mail address: dinglinlin@lnu.edu.cn (L. Ding).

https://doi.org/10.1016/j.knosys.2024.111961
Received 15 April 2024; Received in revised form 11 May 2024; Accepted 14 May 2024
Available online 17 May 2024

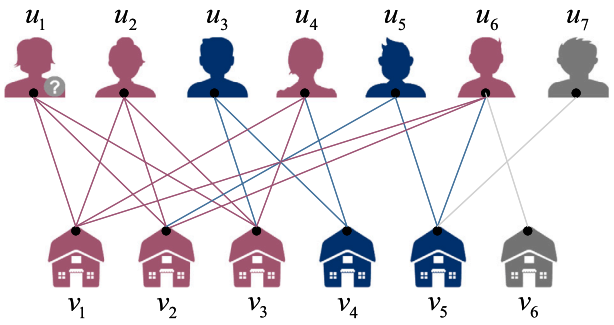


Fig. 1. A toy bipartite graph.

a researcher's interest in attending a specific seminar. Considering the bipartite graph shown in Fig. 1, let us assume u_1 is a conference organizer tasked with planning a themed seminar day. This involves making decisions about which researchers to invite and which seminars to schedule.

- *Degree Constraint*: Each researcher wishes to have at least 2 seminars of interest to choose from, and each seminar needs to attract at least 2 researchers to ensure an active discussion. In Fig. 1, a community highlighted by vertices and edges in blue and pink meets this constraint.
- *Size Constraint*: Due to the venue capacity and the goal of fostering in-depth discussions, the organizer decides to limit the number of researchers participating in the seminar day to exactly 4, and to arrange for exactly 3 distinct seminars.
- *Maximization Constraint*: To enhance the overall experience, u_1 aims to prioritize both the interests of the participants and the appeal of the selected seminars. The goal is to ensure that the seminar day's agenda aligns with the interests of its attendees while maximizing the attractiveness of the chosen seminars.

Additionally, the community with size constraint can also be used to accelerate the GNN information aggregation. Graph Neural Networks (GNNs) have gained significant attention due to their powerful ability to model relationships and interactions in graphs [13–15]. A crucial operation in GNNs involves recursively aggregating information from each vertex's neighbors. Typically, this process uses full connections, which can lead to a substantial amount of redundant computations. By applying a size-constrained community search, where information aggregation is limited to vertices within a specific community rather than all neighbors, we can significantly enhance the efficiency and reduce information loss during information aggregation.

Motivated by these, we introduce the novel concept of maximal size constraint community search over bipartite graphs. Specifically, given a bipartite graph and a query vertex, our objective is to identify a community that includes the query vertex. This community must not only fulfill the (α, β)-core criteria but also adhere to specified size constraints for the number of vertices in each layer of the bipartite graph. Moreover, it is imperative that the identified community exhibits the highest degree of cohesiveness, thereby enhancing its practical applicability and relevance. As shown in Fig. 1, the community represented by the vertices and edges in pink adheres to these constraints.

Challenges. The maximal size constraint community search is demonstrated to be a NP-hard problem (Section 2). Given a bipartite graph $G = (U, V, E)$, a naive solution involves enumerating all combinations $C^{\xi}_{|U|} \cdot C^{\zeta}_{|V|}$ of vertex sets containing query vertex q over the largest (α, β)-core. Subsequently, we verify whether each of them is a maximal size constraint community and then obtain the most cohesive one. However, this approach is inefficient when the size of (α, β)-core is large.

Addressing this challenge with existing solutions continues to involve significant computational expenses. The state-of-the-art algorithm for the (p, q)-biclique counting and enumeration challenge was introduced by Yang et al. [16]. Their goal was to detect all bicliques composed of p vertices in the upper layer and q vertices in the lower layer of a bipartite graph. To this end, they utilized a branch-and-bound method (BCList++) in conjunction with an iterative algorithm, achieving a time complexity of $O(a(H)^p \cdot |E(H)| \cdot deg_{max})$, where H represents the 2-hop graph and $a(H)$ the graph's arboricity. However, adapting BCList++ for our specific problem does not mitigate the computational intensity issue, primarily because its design is focused on identifying vertex sets that meet biclique cohesion criteria rather than enabling the search for personalized communities tailored to individual users.

Furthermore, Wang et al. [9] presented another cutting-edge algorithm aimed at identifying the significant (α, β)-core with the highest minimum weight. Their method proves to be efficient with a complexity of $O(\delta \cdot m)$, where δ is bounded by $\sqrt{m}$. Yet, it merely identifies the largest (α, β)-core. To pinpoint the community that meets the criteria of size, connectivity, and maximal constraints, it still need to enumerate all combinations within this core, which remains a less than optimal approach in terms of computational efficiency.

Contributions. To tackle the maximal size constraint community search problem, we propose several efficient algorithms. Expanding and peeling techniques are extensively utilized to deal with the community search problem [11,17]. Inspired by this, we first propose an Ordering-based Expanding Algorithm (OEA) and a Followers-based Peeling Algorithm (FPA). (i) OEA employs a breadth-first search from the query vertex, guided by a greedy strategy to expand outward until the community reaches the size constraint. Notably, the greedy strategy incorporates a unique vertex ordering technique named "vulnerable ordering". Unlike widely adopted methods such as degree ordering [18] and core ordering [19], vulnerable ordering is tailored for bipartite graphs. This approach offers insights into the vulnerability or stability of each vertex within bipartite graphs (as discussed in Section 3.1). (ii) FPA iteratively removes vertices from the entire (α, β)-community based on their followers until the size constraint is satisfied. Here, followers of a vertex u represent the vertices that will leave the community when u is removed (Section 3.2).

To further improve the query efficiency, we also design an advanced Expand-and-Filter Algorithm (EFA++). Firstly, we transform the initial graph to align with its vulnerable ordering. Subsequently, it extends from the query vertex q within the transformed network to adjacent vertices with higher vulnerable orderings. This incremental process leads to the construction of a marginally larger subgraph G', namely the boundary community. Moreover, this problem can be equivalently converted to a classical dynamic programming problem, exploiting the inherent property of the transformed network. As a result, this optimization strategy leads to a reduced time complexity of $O(|E|deg_{max} + |U + V| \cdot \max\{wx - \xi, wy - \zeta\})$, where wx and wy are the number of each layer vertices in G' (Section 4). In addition, our proposed method naturally possesses an advantage in extending support to the size constraint community search over *dynamic* bipartite graphs (Section 5).

Our main contributions are summarized as below:

- To the best of our knowledge, we are the first to define the maximal size constraint community search problem over bipartite graphs, and prove its NP hardness.
- We propose a novel Ordering-based Expanding Algorithm (OEA) and a Followers-based Peeling Algorithm (FPA) to solve the maximal size constraint community search problem.
- We also design an advanced Expand-and-Filter Algorithm (EFA++) to further improve the query efficiency. Additionally, an incremental online search algorithm is introduced by exploring the "Affected Area" of dynamic changes.
- Extensive experiments demonstrate the effectiveness and efficiency of the proposed algorithms over ten real-world datasets.

Table 1
Summary of symbol notations.

Notation	Description
$G((U,V),E)$	A bipartite graph with vertex sets (U,V) and edge set E
α, β	The degree constraints of each vertex set
ξ, ζ	The size constraints of each vertex set
$G_{\alpha,\beta}$	The (α,β)-core
$\mathcal{R}$	The maximal size constraint community
$\mathcal{L}$	The vulnerable ordering of each vertex
$F(u)$, $FC(u)$	The followers and followers count of u
$N_G(u)$	The neighbors of u in G
$deg(u,G)$	The degree of u in G

Organization. The remainder of this paper is organized as follows. First, we formalize our problem in Section 2 and develop two baseline methods in Section 3. In Section 4, an advanced Expand-and-Filter is presented. The incremental algorithm is introduced in Section 5. Experimental evaluation are discussed in Section 6. Finally, we discuss the related work in Section 7 and conclude the work in Section 8.

2. Problem definition

We first introduce bipartite graphs denoted as $G = ((U,V),E)$, where U and V represent two different sets of vertices of the bipartite graph, respectively. $E \subseteq U \times V$ denotes the set of edges. The set of neighbors of u in G is denoted as $N_G(u) = \{v \in V | (u,v) \in E\}$, and the degree of u is $deg(u,G) = |N_G(u)|$. The frequently used notations are given in Table 1.

Definition 1 (*Induced Subgraph*)**.** Given a bipartite graph $G = ((U,V),E)$, for any subset (U',V') of (U,V) and edge set $E' = \{(u,v)|u \in U', v \in V', (u,v) \in E\}$, $C = ((U',V'),E')$ is a subgraph of G induced by (U',V').

Definition 2 ((α,β)-*Core*)**.** Given a bipartite graph G, the degree constraints α and β, an induced subgraph $G_{\alpha,\beta}$ is an (α,β)-core, if it satisfies the following constraints:

- **degree constraint**: Each vertex $u \in U_{\mathcal{R}}$ satisfies $deg(u,\mathcal{R}) \geq \alpha$ and each vertex $v \in L_{\mathcal{R}}$ satisfies $deg(v,\mathcal{R}) \geq \beta$.
- **connectivity constraint**: $G_{\alpha,\beta}$ is connected.
- **Maximization Constraint**: $G_{\alpha,\beta}$ is maximal, where no supergraph $G' \supset G_{\alpha,\beta}$ is an (α,β)-core.

Definition 3 ((α,β)-*Community*)**.** Given a query vertex q, the (α,β)-core containing q is called as (α,β)-community, i.e., $G_{\alpha,\beta}(q)$.

Definition 4 (*Coreness*)**.** If a vertex u in the upper layer U belongs to an (α,β)-core, but does not belong to an $(\alpha+1,\beta)$-core, we call α the coreness of u. Similarly, we have the coreness of v in V.

With the help of the definition of (α,β)-core and (α,β)-community, we can formally define the maximal size constraint community over bipartite graphs.

Definition 5 (*Maximal Size Constraint Community*)**.** Given a bipartite graph G, a query vertex q, the degree constraints α and β, the size constraint of U and V, i.e., ξ and ζ, $\mathcal{R}$ is an (α,β)-community with additional constraints as follows:

- **Size Constraint**. The total number of vertices in $U_{\mathcal{R}}$ satisfies $|U_{\mathcal{R}}| = \xi$ and the total number of vertices in $L_{\mathcal{R}}$ satisfies $|L_{\mathcal{R}}| = \zeta$.
- **Maximization Constraint**. There exists no other $G' \subseteq G_{\alpha,\beta}$ satisfying connectivity, cohesiveness and size constraints. In addition, $\mathcal{R}$ has the highest total Coreness.

Problem Statement (Maximal Size Constraint Community Problem, MSCC) Given a bipartite graph G, the degree constraints α and β, the size constraints of each layer ξ and ζ, a query vertex q, the maximal size constraint community search aims to find the maximal size constraint community $\mathcal{R}$ in G.

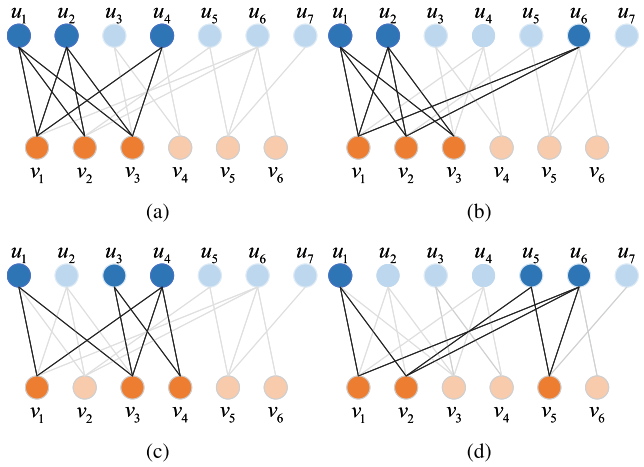


Fig. 2. An example of maximal size constraint community over bipartite graphs.

Example 1. Fig. 2 provides an illustrative example of maximal size constraint community search over a user-location network. Let us consider the scenario when the query vertex for the maximal size constraint community search is u_1, and we adhere to specific degree constraints (2, 2) along with size constraints (3, 3). Under these conditions, Fig. 2(a) and (b) emerge as the optimal result. Although all Fig. 2(a-d) satisfy the predefined criteria of connectivity, cohesiveness, and size constraints, it is clear that Fig. 2(a) and (b) showcase a higher total coreness.

Theorem 1. *The MSCC problem is NP-hard.*

Proof. We establish the NP-hardness of the maximal size constraint community search problem over bipartite graphs (MSCC) by employing a reduction from the size-constraint k-core query problem in static graphs (SCCGQ) [11]. SCCGQ's NP-hardness is well-established through a reduction from the online Steiner tree (OST) problem.

Notably, our MSCC encompass scenarios where the cohesive constraints, represented by α and β, are equal, and the size constraints, denoted as ξ and ζ, are also equal. This scenario is inherently more complex than SCCGQ since SCCGQ addresses unipartite graphs and involves a single size constraint. Even in cases where α and β, as well as ξ and ζ, differ, our MSCC remains more intricate than SCCGQ. Since SCCGQ is NP-hard, this implies that the MSCC problem is NP-hard as well. □

3. The baseline solutions

The naive solution for MSCC problem comprises two parts: the first part relates to computing the (α,β)-community, while the second part enumerates all the possible combinations and checks whether a maximal size constraint community is satisfied.

In the first phrase, we leverage the (α,β)-core decomposition algorithm [6] to obtain an (α,β)-core of the original graph G, denoted as $G_{\alpha,\beta}$, in which the degree of each node in the upper layer is no less than α, and the degree of each node in the lower layer is no less than β. In the second phrase, we examine each subgraph constructed by every subset of $U_{\alpha,\beta}$ of size ξ and every subset of $V_{\alpha,\beta}$ of size ζ. The subgraphs that contain the query vertex q and satisfy the connectivity, cohesiveness, size constraints will be seen as the candidates. Finally, the candidate with the highest total coreness will be regarded as the maximal size constraint community.

Table 2
Different ordering.

Vertex	u_1	u_2	u_3	u_4	u_5	u_6	v_1	v_2	v_3	v_4	v_5
Degree	3	3	2	3	2	3	4	4	4	2	2
Core	2	2	2	2	2	2	2	2	2	2	2
Vulnerable	4	4	1	2	1	2	3	3	3	1	1

The time complexity of the naive solution can also be divided into two parts. The first process is to get the (α, β)-core, i.e., $G_{\alpha,\beta}$ by utilizing the core decomposition algorithm, which time complexity is $O(m)$, where m represents the number of edges. Assume the numbers of nodes in each layer of $G_{\alpha,\beta}$ is $|U_{\alpha,\beta}|$ and $|V_{\alpha,\beta}|$, respectively. The time complexity of enumeration and justifying the community is $O(C^{\xi}_{|U_{\alpha,\beta}|} \cdot C^{\zeta}_{|V_{\alpha,\beta}|} \cdot (\xi + \zeta))$. It is evident that the naive solution manifests an exponential time complexity, rendering it particularly time-consuming, especially when the number of nodes within the induced (α, β)-core is substantial.

Alternatively, we can extend the BS algorithm in [11] or the BCList++ algorithm in [16] to solve our MSCC problem. However, they are still inefficient, since the extension of BS algorithm expands from the query vertex in a breath-first search, listing an extensive number of combinations, whilst the BCList++ algorithm follows a branch-and-bound method in an iterative way, being suboptimal for maximal size constraint community search.

In this section, we introduce two key algorithms. Firstly, the Ordering-based Expanding (OEA) algorithm is proposed, which employs a greedy expansion strategy from the query vertex based on a novel vertex ordering. Secondly, the Followers-based Peeling (FPA) algorithm is presented, utilizing a greedy removal strategy according to the followers of vertices.

3.1. Ordering-based expanding algorithm

The Ordering-based Expanding Algorithm (OEA) initiates a breadth-first search from the query vertex q, employing a greedy strategy to expand towards vertices that contribute to increased cohesiveness in the subgraph G'. This strategy leverages a distinctive vertex ordering technique called "vulnerable ordering", and during the expansion, it selects connected vertices with higher vulnerable ordering.

Vulnerable Ordering. Vertex ordering plays a pivotal role in enhancing graph query performance [18–20]. Among the array of strategies, degree ordering [18,20] and core ordering [19] stand as widely adopted techniques. In this paper, we introduce a novel vertex ordering technique, coined *vulnerable ordering*, over bipartite graphs that integrates both degree and core ordering, providing insights into the vulnerability or stability of each vertex.

The primary concept behind vertex vulnerability ordering is to dynamically identify the most vulnerable vertices, where the most vulnerable vertices are the vertices with the lowest difference between their current degree and the degree constraints α or β.

We propose an efficient algorithm to solve the vertex vulnerable ordering in linear time. The distinction between vulnerable ordering and core ordering can be summarized in two key aspects:

- *Target Focus*: Vulnerable ordering emphasizes the differences between the current degree of a vertex with degree constraints α or β, while core ordering focuses on the degree of each vertex.
- *Processing Procedure*: In vulnerable ordering, the procedure entails identifying a series of nodes with the smallest degree differences and establishing their ordering. Subsequently, it involves reducing the degrees of their neighbors. This process is iterated until all nodes are assigned an ordering. In contrast, core ordering commences by identifying a single node v with the smallest degree, and then decreases the degrees of neighbors u with degrees greater than $deg(v)$. This process is reiterated until all remaining nodes share the same degree.

Algorithm 1: Vertex Vulnerable Ordering Algorithm

Input : G: a bipartite graph
α,β: degree constraints
ξ,ζ: size constraints
Output: $\mathcal{L}$: the order of each vertex

1 $G_{\alpha,\beta} \leftarrow G$
2 **while** $\exists deg(u, G_{\alpha,\beta}) < \alpha$ *or* $\exists deg(v, G_{\alpha,\beta}) < \beta$ **do**
3 remove u or v and its incident edges from $G_{\alpha,\beta}$
4 **if** $|U_{\alpha,\beta}| < \xi$ *or* $|V_{\alpha,\beta}| < \zeta$ *or* $q \notin G_{\alpha,\beta}$ **then**
5 return $\emptyset$
6 Initialize a vector df to record the degree difference
7 **for** *each* $u \in U_{G_{\alpha,\beta}}$ **do**
8 $df(u, G_{\alpha,\beta}) = deg(u, G_{\alpha,\beta}) - \alpha$
9 **for** *each* $v \in V_{G_{\alpha,\beta}}$ **do**
10 $df(v, G_{\alpha,\beta}) = deg(v, G_{\alpha,\beta}) - \beta$
11 $level \leftarrow 1$
12 **while** $G_{\alpha,\beta} \neq \emptyset$ **do**
13 $\gamma \leftarrow min_{u \in (U_{G_{\alpha,\beta}} \cup V_{G_{\alpha,\beta}})} df(u, G_{\alpha,\beta})$
14 **while** $\forall u \in G_{\alpha,\beta} : df(u, G_{\alpha,\beta}) = \gamma$ **do**
15 $\mathcal{L}(u) \leftarrow level$
16 **while** $\forall u \in G_{\alpha,\beta} : df(u, G_{\alpha,\beta}) = \gamma$ **do**
17 remove u and its incident edges from $G_{\alpha,\beta}$
18 $level \leftarrow level + 1$
19 **return** $\mathcal{L}$;

The pseudo-code of the vulnerable ordering is illustrated in Algorithm 1. We begin by employing the core decomposition algorithm to obtain the (α, β)-core, denoted as $G_{\alpha,\beta}$. This involves a cascading removal of vertices from two layers: vertices with a degree less than α in the upper layer and those with a degree less than β in the lower layer (Lines 1–3). Subsequently, we examine whether the query vertex q is absent from $G_{\alpha,\beta}$, and assess if the current size of the (α, β)-core exceeds the specified size constraints. If these conditions are not met, the algorithm will terminate and return NULL (Lines 4–5). Then, we initialize df to record the degree difference of each vertex $u \in U_{G_{\alpha,\beta}} \cup V_{G_{\alpha,\beta}}$, and its initial value is set as the difference between the current degree and α or β (Lines 6–10).

Next, we will conduct the peeling onion procedure to derive the final vulnerable ordering. Specifically, we use a variable $level$ to keep track of the current vulnerable ordering (Line 11). Subsequently, for each vertex $u \in G_{\alpha,\beta}$ with the lowest degree difference df, we assign its vulnerable ordering as $level$ (Lines 14–15). Following the assignment process, we perform the peeling operation by removing u and its incident edges from $G_{\alpha,\beta}$ (Lines 16–17). At the end of the loop, we update the value of $level$ (Line 18). This process differs from core ordering, where Lines 15 and 17 are combined in a single iterative step. This subtle change results in distinct ordering, as our vulnerable ordering emphasizes the degree to which each vertex departs from others.

Example 2. Consider the bipartite graph depicted in Fig. 3. Among the vertices, we observe that u_3, u_5, v_4 and v_5 have the smallest difference between $\alpha = 2$ or $\beta = 2$. Consequently, we assign an ordering of '1' to these vertices. Following this, we identify u_4 and u_6 as having the lowest residue degree difference, and thus, they are assigned an ordering of '2'. This process is applied iteratively until all vertices have been ranked. The resulting degree, core, and vulnerable orderings are presented in Table 2.

Expanding. Leveraging the vulnerable ordering, our ordering-based expanding algorithm (OEA) designates the query vertex as the center for outward expansions. In each expansion, the center vertex expands to its neighbors with the highest vulnerable ordering to form a smallest (α, β)-community. The algorithm then checks whether the current

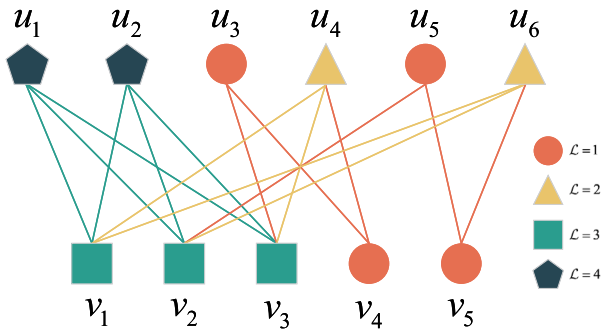


Fig. 3. A $(2,2)$-core of the toy graph with its ordering.

community meets the size constraints. If not, it selects the vertex with the highest vulnerable ordering as the new center to initiate another expansion. If the current size exceeds the size constraints, the algorithm rolls back to select another vertex as a center for a new expansion. It is worth noting that there may be multiple vertices with the same vulnerable ordering. To handle this scenario, we utilize a breadth-first search approach to obtain all possible expansions and record the community cohesiveness during this process. Finally, the returned community satisfies the degree, size, and maximal constraints, forming a maximal size constraint community. For instance, as depicted in Fig. 3, if the query vertex is u_1, we initially identify the smallest $(2,2)$-community consisting of $\{u_1,u_2,v_1,v_2,v_3\}$. If the size constraint is $(2,3)$, this community is accepted as the final result. However, if the size constraint is $(3,3)$, the expansion continues to include vertices u_4 or u_5 from the upper layer, who have the highest vulnerable ordering. Consequently, the communities formed by $\{u_1,u_2,u_4,v_1,v_2,v_3\}$ and $\{u_1,u_2,u_5,v_1,v_2,v_3\}$ are regarded as the final maximal size constraint communities for u_1.

Note that if we discover the result community with the center vertex vulnerable ordering as l, we do not explore expansions with the center vertex being $l-1$. This truncated strategy ensures the discovery of the most cohesive community because the vulnerable ordering reflects the cohesive contribution of each vertex. At the same time, this strategy avoids listing all combinations of vertices.

Time Complexity. OEA algorithm has two phrases. Vulnerable ordering can be computed in $O(|E|)$ time, where $|E|$ represents the total edges in G by employing a strategy similar to core decomposition. Expanding process requires $O(2^{\xi+\zeta}\cdot deg_{max})$. In total, the time complexity of OEA algorithm is $O(|E|+2^{\xi+\zeta}\cdot deg_{max})$.

3.2. Followers-based peeling algorithm

The Followers-based Peeling algorithm (FPA) employing a greedy strategy to remove vertices from the (α,β)-community based on their number of followers until the size constraints are satisfied. Here, "follower" denotes the set of vertices that will exit the community upon the removal of a specified vertex u, effectively capturing the impact of a vertex's departure on the community's size.

Definition 6 (*Followers and Followers Count*)**.** Given an (α,β)-core, i.e., $G_{\alpha,\beta}$, and a vertex u, the followers $F(u)$ of u are the vertices in $G_{\alpha,\beta}$ that will depart from the community with u due to the degree constraints imposed by the (α,β)-core, i.e. $G_{\alpha,\beta}-G_{\alpha,\beta}[G_{\alpha,\beta}\setminus u]$. And the followers count FC can be represented as (nu,nv), where nu means the number of the upper layer vertices whose degree is less than α following the deletion of u, nv means the number of the lower layer vertices whose degree is less than β following the deletion of u.

Example 3. Consider the $(2,2)$-core illustrated in Fig. 3. The followers of u_4 are 2 vertices (u_4 and u_3) in the upper layer and 1 vertex (v_4) in the lower layer, i.e., $F(u_4)=\{u_4,u_3,v_4\}$. Then the followers count is $(2,1)$, i.e., $FC(u_4)=(2,1)$.

The way to compute the followers count for a given vertex u in $G_{\alpha,\beta}$ involves executing a breadth-first search (BFS) starting from u to carry out expansion. Initially, we record u along with its layer, and enqueue its neighbors into a queue Q. We dequeue it and decrement the degree of the head vertex qt by 1. Subsequently, we verify whether it still adheres to the degree constraints. If not, we record qt along with its layer in the result set and enqueue its neighbors into Q. This process is repeated until the queue is empty. Consequently, we obtain the followers and their count for u.

Peeling. Our FPA algorithm removes vertices from the entire (α,β)-community according to their followers count until the remaining (α,β)-community satisfies the size constraint. In each peeling step, FPA selects a vertex based on its followers count and removes its followers from the current community. The algorithm then checks whether it meets the size constraints. If not, it selects another vertex to initiate another peeling. If the current size is less than the size constraint, the algorithm rolls back to select another vertex for a new peeling. The process is the opposite of the OEA algorithm, and the difference lies in the greedy selection. The greedy strategy in the FPA algorithm is to select the vertex with a higher followers count, as it can contribute more to the difference between the current community size and the size constraints.

Time Complexity. FPA has two phrases. Followers computation can be computed in $O(|E|\cdot|U+V|)$ time. Peeling process requires $O(2^{\xi+\zeta}\cdot deg_{max})$. In total, the time complexity of FPA is $O(|E|\cdot|U+V|+2^{\xi+\zeta}\cdot deg_{max})$.

4. The advanced approach

While the OEA and FPA provide a valuable computational framework for solving our problem, our empirical study suggests that they still suffer from the drawback of a large search space. In detail, the breadth-first search approach in both expansion and peeling lists several combinations to find the most cohesive one. Consequently, these two algorithms are inefficient when the size constraints ξ and ζ are large, as their time complexity is exponential to $\xi+\zeta$.

To alleviate the drawbacks, we propose an advanced Expand-and-Filter Algorithm (EFA++). This algorithm can be divided into 4 parts. Specifically, (i) we initially transform the original graph into transformed networks based on their vulnerable ordering. (ii) we then expand the community from the query vertex q to acquire a marginally larger community according to the transformed networks. (iii) we delve into the inherent properties of the transformed networks, which help accelerate followers computation in near linear time. (iv) by leveraging the followers count and transformed network, we elegantly transform the community search problem into a classical dynamic programming problem. Next, we will introduce each of the four sections in sequence.

Transformed Networks. After applying vulnerable ordering to the vertices, two types of transformed networks emerge: (i) the followers network, which elucidates relationships between vertices with distinct vulnerable orderings across different layers; and (ii) the inner network, delineating relationships between vertices sharing the same ordering within a given layer. Specifically, followers networks include a followers right network and a followers left network, where the followers right network represents connections from vertices with lower ordering to higher ones, while the followers left network reflects connections from vertices with higher ordering to lower ones. The inner network reflects the relationships between vertices with the same ordering, and a pointer is used to indicate the different vulnerable ordering.

In detail, given a bipartite graph G, the followers right network $\tilde{G}$ and followers left network $\hat{G}$ are specifically induced by the vulnerable ordering $\mathcal{L}$. The vertex sets of these networks are identical. An edge $u\to v$ exists in $\tilde{G}$ iff $\mathcal{L}(u)<\mathcal{L}(v)$ and $(u,v)\in E(G)$. An edge $u\to v$ exists in $\hat{G}$ iff $\mathcal{L}(u)>\mathcal{L}(v)$ and $(u,v)\in E(G)$. In contrast, for a bipartite graph G, the inner network $\bar{G}$ is similarly induced by the vulnerable ordering $\mathcal{L}$. While the vertex sets remain identical, an edge $u\to v$ exists in $\bar{G}$ iff $\mathcal{L}(u)=\mathcal{L}(v)$ and $(u,v)\in E(G)$.

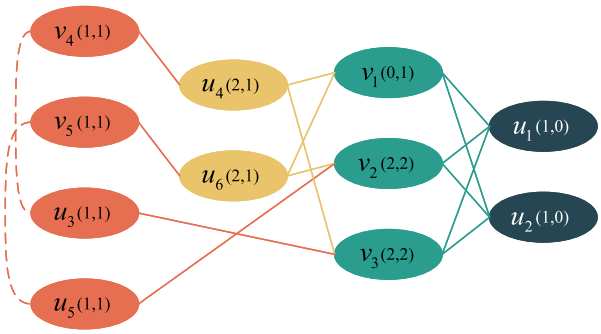


Fig. 4. The followers network.

Example 4. Following the example in Fig. 3. The followers network is illustrated in Fig. 4 with solid line, while the inner network is represented with the dotted line.

The followers network can also be utilized for followers computation, and its additional properties will be introduced in the following section.

Expanding to a Boundary Community. Initiating from q with the objective of obtaining the most cohesive community, we initially expand to the vertices directly or indirectly connected to q with a higher vulnerable ordering. Note that, the difference between this expansion with the OEA algorithm lies in that this expansion encompasses vertices with the same vulnerable ordering together. If the size constraints are met during this process, we can directly obtain the maximal size constrained community. However, if the right community size is not achieved, we need to identify the boundary size. This occurs when, upon adding vertices with a vulnerable ordering of $i-1$, the size is smaller than the given size constraint, but upon adding vertices with a vulnerable ordering of i, the size exceeds the given constraint. What is more, the community with boundary size is called boundary community.

In situations where including all related vertices with a higher vulnerable ordering fails to meet the size constraint, we continue to incorporate vertices with a lower vulnerable ordering, expanding until we identify the boundary community.

The pseudo-code of expanding to a boundary community is shown as Algorithm 2. We begin by computing the vulnerable ordering of each vertex (Line 1). Subsequently, we expand from q based on the vulnerable ordering. During the expanding process, we first expand from vertex v in the followers right network $\tilde{G}$ to vertices with a higher vulnerable ordering. If the size of the current subgraph meets the size constraint (ξ,ζ), the expand-peeling framework terminates, and the final results are obtained directly (Lines 8–10). If the size of the current subgraph exceeds (ξ,ζ), the subgraph with the boundary size is returned (Lines 11–12). A similar operation is then conducted on the vertices in the inner network, followed by the followers left network.

Followers Computation. If we apply the aforementioned method from the FPA algorithm directly to compute the followers count for each vertex in G', it incurs a high computational complexity of $O(|E|\cdot|U+V|)$, rendering it excessively time-consuming. The elevated complexity stems from the non-reusability of this approach, as it neglects the inter-vertex relationships during followers count computation. To illustrate, consider the followers count of v_4 as $(1,1)$ and compare it to the followers count of its adjacent neighbor u_4, which is $(2,1)$. In this scenario, the followers count $FC(v_4)=(1,1)$ can be reused, presenting an opportunity for accelerating the followers computation process.

Within the followers network, vertices can be categorized into three types: (i) *Exposed vertex*: vertices with the minimum vulnerable ordering. (ii) *Influenced vertex*: vertices connected to the exposed vertices. (iii) *Steady vertex*: vertices not connected to the most vulnerable vertices.

Algorithm 2: Expansion Algorithm

Input : G: a bipartite graph
α,β: degree constraints
ξ,ζ: size constraints
q: a query vertex
Output: G': subgraph G' with the boundary size

1 invoke Algorithm 1 to get vulnerable ordering $\mathcal{L}$
2 $G' \leftarrow q$
3 Initialize a queue $Q \leftarrow q$
4 **while** $Q \neq \emptyset$ **do**
5 $h = Q$.dequeue()
6 **for** *each* $v = nbr(h, \tilde{G})$ **do**
7 $G' \leftarrow v$, $Q \leftarrow v$
8 **if** $(G'.size(u), G'.size(v) = (\xi,\zeta)$ **then**
9 terminate the expand-peeling framework
10 **return** G';
11 **if** $(G'.size(u), G'.size(v) > (\xi,\zeta)$ **then**
12 **return** G';
13 expand G' to the vertices with an equal vulnerable ordering like Lines 3-12
14 expand G' to the vertices with a lower vulnerable ordering like Lines 3-12
15 **return** G';

Example 5. Following the example in Fig. 3. Within followers network, u_3, u_5, v_4, v_5 are exposed vertices, u_4, u_6, v_2, v_3 are influenced vertices, and the residue vertex are steady vertices.

We use different methods depending on the type of node, i.e., *exposed*, *influenced* and *steady* vertex, to compute their followers and followers count.

Exposed vertices uniformly possess the lowest vulnerable ordering and appear in the initial layer of the followers network logically. Obtaining the followers of exposed vertices is simplified by counting their neighbors in the inner networks. It is noteworthy that connected exposed vertices share identical followers and follower counts. This equivalence arises from the fact that they are cascade-deleted together with the removal of a single vertex, with only 1 residue degree difference to the given degree constraints. Consequently, we can expedite the process by assigning the followers and their counts to the adjacent vertices in the inner network.

Steady vertices possess a relatively high vulnerable ordering and do not connect to exposed vertices. Consequently, the deletion of a steady vertex does not impact the satisfaction of degree constraints for other vertices. This is because all remaining vertices have at least a two-degree difference from the specified degree constraints, and the removal of a single vertex will not cause them to violate the degree constraints.

The computation of followers and their counts for influenced vertices directly influences the overall efficiency of the followers computation algorithm. Therefore, we first delve into the properties of the followers network.

Property 1. *When deleting an exposed vertex u and an influenced vertex v, if there exists a connection between u and v in the followers network $(\mathcal{L}(u)<\mathcal{L}(v))$, then the total followers count of u and v equals the followers count of v, i.e., $FC(u+v)=FC(v)$.*

Property 2. *In the followers network, a vertex with a vulnerable ordering $\mathcal{L}=i$ exists in the ith layer of the followers network logically. It connects to $i-1$ neighbors with lower ordering and i neighbors with higher ordering.*

Property 3. *When deleting an influenced vertex u and a steady vertex v, if there exists a connection between u and v in the followers network, then the*

total followers count of u and v is equal to the sum of the followers count of u and the followers count of v, i.e., $FC(u+v) = FC(u) + FC(v)$.

Through the property of followers network, we can compute the followers of influenced vertex in different cases. For a given influenced vertex, **Case (i)** occurs when it only connects to exposed vertices, and **Case (ii)** occurs when it connects to both exposed vertices and other influenced vertices.

Case (i) The vertex with a vulnerable ordering of 2 in the second layer logically connects to only one exposed vertex, as indicated by Property 2. Assuming a vertex u with a vulnerable ordering of 2 connects to an exposed vertex v, the followers of u can be directly obtained through $FC(v)$ by adding u itself.

Case (ii) For a vertex u in a higher layer ($L(u) > 2$), it connects to only one exposed vertex, denoted as w, in the followers network. Otherwise, the exposed vertex would no longer have a vulnerable ordering of 1. We can consider the neighbors of w in the inner network as affected exposed vertices and push them into a hash map. If a neighbor of u, denoted as v in the second layer, has a connection with these affected exposed vertices, v needs to be further added to the affected hash map. The vertices in the hash map will be the followers of u. Otherwise, only the exposed vertices and u will be impacted by the deletion of u.

Algorithm 3 presents the details of computing the followers count in a bipartite graph. Initially, we set the followers count $FC = (fu, fv)$ of each vertex to 0, where fu and fv are the followers count in the upper and lower layers, respectively. For each exposed vertex u, we first check whether it has been assigned followers; if so, we proceed to the next exposed vertex (Lines 3–4). This is because connected exposed vertices share identical followers and follower count. Subsequently, the neighbors of u in the inner network will be regarded as its followers, and we can record the number of vertices in each layer. At the end of the exposed vertex's followers computation, we assign u's followers count to its neighbors in the inner network (Lines 6–12).

In the case of steady vertex u, we treat them as the follower and determine the followers count accordingly (Lines 13–15). For each influenced vertex u, we traverse its neighbors in followers right network $\tilde{G}$ (Lines 16–25). The first neighbor must be an exposed vertex v, and we can expand the affected vertices by adding the adjacent neighbors of v in the inner network $\bar{G}$. Subsequently, we search for the vertices with $\mathcal{L}(v) = 2$ and assess whether they have a connection with the affected vertices; if so, they are added to the affected vertices. Regarding the vertices with $\mathcal{L}(v) > 2$, they cannot have two edges with the exposed vertices, thus remaining stable even after the removal of one edge. In conclusion, we obtain the affected vertices of an influenced vertex u and determine the followers count accordingly.

Dynamic Programming. The goal of this procedure is to identify combinations of vertices within the boundary community G' with lowest vulnerable ordering. This process utilize a dynamic programming techniques according to followers counts and size differences. Finally, the maximal size constraint community will be constructed through this vertex combination and the boundary community G'.

Recall that in the inner networks, if u and v are connected, their followers are the same because their follower sets are identical. Consequently, we can treat them equally. Moreover, if two vertices have no connections, their followers can be directly added according to Property 3. This allows us to convert the size constraint (α, β)-community search problem into the combination of followers of each vertex to meet the size constraint difference. Interestingly, this problem can be solved by the Knapsack problem.

Given a size difference (dfu, dfv) and a set of followers counts for each vertex FC, we aim to identify a set of vertices whose total followers count equals the size difference (dfu, dfv). Fortunately, this problem can be converted into the 0-1 Knapsack problem. Suppose there is a knapsack with a capacity of dfu, and $|U_{G'}|$ items where each item (each vertex u in G') has a weight $FC(u).size(u)$ (the followers count of vertices in the upper layer) and a profit associated with it $FC(u).size(v)$ (the followers count of vertices in the lower layer). The goal of the 0-1 Knapsack problem is to place the items into the knapsack to maximize the sum of profits. Since we obtain all possible total profits during the dynamic programming procedure, we can search whether a total profit equals dfv, and then roll back to determine the combination of followers count, ultimately obtaining the corresponding set of vertices.

Algorithm 3: Followers Computation Algorithm

Input : G: a bipartite graph
$\tilde{G}$: the corresponding followers network
$\bar{G}$: the corresponding inner network
Output: FC: followers count

```
1  Initialize FC = (fu, fv) with (0,0) for each vertex
2  for each u ∈Exposed(U ∪ V) do
3      if FC(u) ≠ (0,0) then
4          continue
5      Initialize a queue Q ← u, a hashmap F ← u
6      while Q ≠ ∅ do
7          w ← Q.dequeue()
8          for each h ∈ nbr(w, G̃) do
9              Q ← h, F ← h
10     FC(u) = (F.size(u), F.size(v))
11     for each w ∈ F do
12         FC(w) = FC(u)
13 for each u ∈Steady(U ∪ V) do
14     Initialize a hashmap F ← u
15     FC(u) = (F.size(u), F.size(v))
16 for each u ∈Influenced(U ∪ V) do
17     Initialize a queue Q ← u, a hashmap F ← u
18     for each v = nbr(u, G̃) do
19         if L(v) = 1 then
20             Q ← v for each w = nbr(v, G̃) do
21                 Q ← w
22         L(v) = 2 for each w = nbr(v, G̃) do
23             if w ∈ Q then
24                 Q ← v
25     FC(u) = (F.size(u), F.size(v))
26 return FC;
```

During the dynamic programming procedure, the key point is to determine whether to include the vertex u with a weight $FC(u).size(u)$ in the knapsack. The following dynamic programming equation captures this decision:

$$dp(i,j) = max \begin{cases} dp(i-1,j) \\ dp(i-1, j - FC(u).size(u)) + FC(u).size(v) \end{cases}$$

The variable $dp(i,j)$ represents the maximum total profit of i items when the remaining capacity is j. Using Equation (1), for each vertex u in the subgraph with boundary size, if the total profit increases by adding u, i.e., $dp(i-1,j) < dp(i-1, j - FC(u).size(u)) + FC(u).size(v)$, we update $dp(i,j)$ and record the path for subsequent backtracking to obtain the combination. It is important to note that we need to modify the dynamic programming to obtain the maximum total profit exactly equal to the residue degree in the lower layer dfv. For this purpose, we initialize $dp(i,0) = 0$ for all $i < |U_{G'}|$, and $dp(i,j) = -\infty$ for all $i < |U_{G'}|$ and $j > 0$.

The pseudo-code of the dynamic programming algorithm is illustrated in Algorithm 4. We first reorder the vertex in the community G', and invoke Algorithm 3 to obtain the followers count of each vertex in G' (Lines 1–2). Then we compute the size difference (dfu, dfv) with the constraint (ξ, ζ). Next, we initialize the candidate vertex CV

as the unconnected vertices of G' with lowest vulnerable ordering $\mathcal{L}$, excepting for the query vertex q or the vertices whose followers count exceeds the size difference (Lines 4–7). After that, we conduct the dynamic programming over the vertices in CV with their followers count of lower layers vertices and upper layers vertices. During the dynamic programming (Lines 10–15), we first initialize the variable $dp(i,j)$ to obtain the exact dfv as the total maximum profits. According to the dynamic programming equation, we have all $dp(i,j)$ for each $1 \le i \le |CV|$ and $1 \le j \le dfu$. Then we search for the dfv through dp, and roll back according to $path$ to obtain the set of vertices SC (Lines 16–20). Let PC donates the community without the vertex in lowest ordering. The final MSCC will be obtained by intersecting SV and PC.

Algorithm 4: Dynamic Programming Algorithm

```
Input  : G': subgraph with the boundary size
         ξ,ζ: size constraints
         q: a query vertex
Output: R: maximal size constraint community
1  invoke Algorithm 1 over G' to get vulnerable ordering L'
2  invoke Algorithm 3 over G' to obtain followers count
3  (dfu, dfv) = (|U_G'|, |V_G'|) − (ξ, ζ)
4  CV ← unconnected vertices of Ḡ' with lowest ordering L'
5  for each vertex u ∈ CV do
6      if FC(u) > (dfu, dfv) or u = q then
7          CV ← CV \ u
8  Initialize 2d array dp, path
9  dp(i,0) = 0, dp(i,j) = −∞∀j > 0
10 for i = 1 ··· |CV| do
11     for j = 1 ··· dfu do
12         if dp(i−1,j) < dp(i−1, j − FC(u).size(u)) + FC(u).size(v) then
13             dp(i,j) = p(i−1, j − FC(u).size(u)) + FC(u).size(v)
14             path(i,j) = 1
15         dp(i,j) = dp(i−1,j)
16 Search dp(i,j) = dfv
17 while i > 0, j > 0 do
18     if path(i,j) = 1 then
19         SV ← u_i, j ← j − F(u_i).dfu
20     i ← i − 1
21 PC ← G' \ u for each u with L(u) = minL'
22 R ← PC ∪ SV
23 return R;
```

Time Complexity. The EFA++ algorithm comprises graph transformation, expanding to the boundary community, followers computation and dynamic programming. Graph transformation and expanding will be completed in linear time. The time complexity of followers computation is $O(|E_{G'}| \cdot deg_{max}(\bar{G}))$, where $|E_{G'}|$ is the total edge number of boundary community G', and $deg_{max}(\bar{G})$ is the maximum degree of inner network $\bar{G}$. However, the dynamic programming algorithm visits each vertex in G' at most $\{wx-\xi, wy-\zeta\}$ times, so its time complexity can be denoted as $O(|U+V| \cdot \max\{wx-\xi, wy-\zeta\})$, where wx and wy are the number of each layer vertices in G'. In total, the time complexity is $O(|E|_{G'} deg_{max}(\bar{G})) + |U+V| \cdot \max\{wx-\xi, wy-\zeta\}$. ***Correctness***. The community aims to identify the maximal (α,β)-community adhering to predefined size constraints. The vulnerable ordering assigns ranks to vertices based on their cohesiveness within the graph. The ordering-based expanding algorithm ensures the acquisition of the most cohesive boundary community. Additionally, the dynamic programming algorithm seeks vertex combinations with the lowest vulnerable ordering, thereby guaranteeing that the returned community attains the highest total vulnerable ordering, indicative of its maximum cohesiveness.

5. Incremental online algorithm

When graphs are dynamically updated, a straightforward solution to solve the maximal size constraint community search problem is to re-compute it, which is inefficient for large graphs due to a large amount of recomputation. In this section, we discuss the incremental algorithms for maintaining vulnerable ordering, and transformed networks, which can accelerate the process of MSCC search over dynamic bipartite graphs. We mainly focus on edge insertion and deletion, because node insertions/deletions can be treated as a sequence of edge insertions/deletions.

Edge Insertion. Suppose an edge (u,v) is inserted. If we recompute the followers of each vertex, then conducting the EFA++ framework to obtain the updated maximal size constraint community will yield unnecessary recomputation. To avoid this phenomenon, we can recognize the "Affected Area" of an edge insertion. We can then reorder all vertices in $O(m)$ time complexity, and compute the followers of the "Affected Area" rather than the entire vertex set.

Property 4 (*Affected Area*)**.** *Suppose an edge (u,v) is inserted, the followers of u and v remains unchanged. The "Affected Area" of (u,v) is denoted as the connected influenced vertices.*

Proof. The computation of followers for a given vertex u is akin to peeling an onion. When an edge (u,v) is inserted, to compute the followers of u or v, the procedure involves peeling the vertex along with its incident edges. Consequently, the added edges have no impact on the followers of u or v. On the contrary, the added edge (u,v) will indeed influence the followers of its connected influenced vertices. This is because these influenced vertices are connected to the exposed vertices, and the presence of the edge (u,v) may prevent the deletion of exposed vertices. □

The affected area resulting from the deletion of an edge (u,v) is analogous to that resulting from its addition. When a sequence of edges are inserted/deleted, we can batch update their "Affected Area", and recompute their followers. It is essential to note that the most computationally intensive part of the EFA++ framework is the followers computation. By identifying the "Affected Area", only the vertices within this region need to undergo followers count recomputation, rather than re-evaluating each vertex. This accelerates the process of obtaining the maximal size constraint community.

6. Experiments

In this section, we empirically evaluate the performance of the proposed algorithms. We conduct all experiments on a Windows 10 server with Intel Xeon Silver 4210R 2.4 GHz processor and 128 GB main memory, and all of the algorithms are implemented in C++. We terminate an algorithm if the running time is more than 10^4 seconds.

Datasets. We conducted the algorithms on ten real-world networks datasets collected from KONECT.[1] The summary of datasets is shown in Table 3. $|U|$ and $|V|$ are the number of vertices in each layer, $|E|$ is the number of edges. $\bar{d_U}$ and $\bar{d_V}$ are the average vertex degree in each layer, $\bar{d}$ is the total average vertex degree. For brevity, K denotes 10^3 and M denotes 10^6 multipliers.

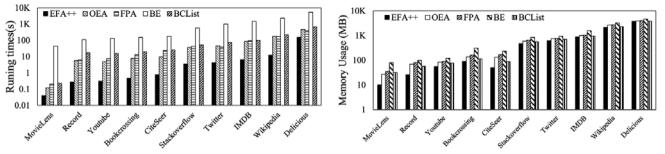
Algorithms. We implement and compare following algorithms:

- EFA++: our proposed advanced approach, i.e., Expand-and-Filter Algorithm.
- OEA: our proposed Ordering-based Expand Algorithm.
- FPA: our proposed Followers-based Peeling Algorithm.
- BE [11]: a state-of-art solution for size constraint community search over unipartite graphs, and we adjusted it to the bipartite graph.
- BCList [16]: a state-of-art solution for (p,q)-bicluque enumeration algorithm, which is a size constraint community modeled as biclique.

[1] http://www.konect.cc/networks/

Table 3
Summary of datasets.

Datasets	$\|U\|$	$\|V\|$	$\|E\|$	d_U^-	d_V^-	$\bar{d}$
MovieLens	24K	16K	7K	5.78	12.57	7.92
Record	168K	18K	233K	1.39	12.66	2.50
Youtube	94K	30K	293K	3.11	9.75	4.72
Bookcrossing	77K	185K	433K	5.57	2.33	3.29
CiteSeer	105K	181K	512K	4.86	2.82	3.57
Stackoverflow	545K	96K	1.3M	2.39	13.47	4.06
Twitter	175K	530K	1.89M	10.80	3.56	5.36
IMDB	303K	896K	3.78M	12.46	4.22	6.30
Wikipedia	1.8M	1.8K	3.79M	2.07	20.75	3.72
Delicious	833K	33.7M	301M	361.533	8.91	17.4



(a) Evaluation of running time (b) Evaluation of memory usage

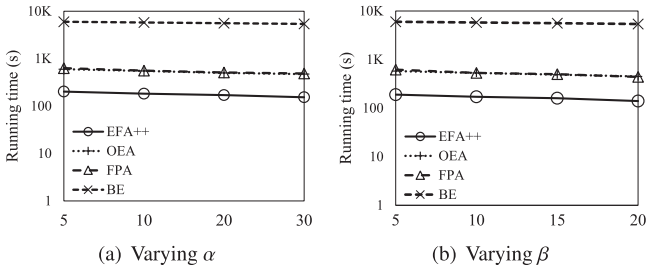
Fig. 5. Query performance on different datasets.

6.1. Efficiency evaluation

Exp-1: Experiments over all datasets. In Fig. 5, we evaluate the performance of EFA++, OEA, FPA, BE and BCList across all datasets in terms of processing time and memory usage. From Fig. 6(a) we can see that EFA++, OEA, EFA, and BCList outperform BE. This is attributed to the fact that the BE algorithm expands from the query vertex while considering the degree of each vertex, listing all results in a breadth-first search manner without fully leveraging the bipartite graph features. EFA++, OEA, and FPA demonstrate higher efficiency compared to BCList, thanks to their tight cohesive models and iterative approaches. Among these algorithms, EFA++ stands out as the most effective. It addresses the limitations of both pure expanding and pure peeling, delving into the properties of maximal size constraint communities. This allows EFA++ to avoid listing all results in a non-iterative manner, contributing to its superior performance. **Memory Usage**. The memory usage statistics for the EFA++, OEA, FPA, BE, and BCList algorithms are reported in Fig. 6(b). In general, EFA++ exhibits the lowest memory footprint. This discrepancy arises from EFA++'s utilization of a boundary community, which enables it to circumvent listing all combinations and, consequently, incurs lower memory costs.

Exp2: Varying values of α and β. In Fig. 6, we vary α and β on dataset *Delicious*, respectively. The reason for selecting this dataset is its possession of the largest average degree, rendering it more suitable for varying α and β. In addition, we exclude BCList algorithm since it computes the biclique, without parameter α and β. From Fig. 6(a), we set $\beta = 5$, $(\xi, \zeta) = (30, 30)$, and varying α from 5, 10, 20 to 30. We can see that, the running time of each algorithm decreases as α increases. The efficiency of each algorithm depends on the size of (α, β)-community. When α increases, the size of (α, β)-community decreases, boosting performance of all algorithms. Among them, EFA++ consistently outperforms other algorithms, particularly when varying α. This superiority is attributed to the linear time complexity of EFA++, which is proportional to the product of the total number of edges and the total number of vertices. This efficiency stands in contrast to other algorithms whose time complexity of exponential to the size constraint. In Fig. 6(b), α is fixed at 10, $(\xi, \zeta) = (30, 30)$, and β is varied from 5, 10, 15 to 20. The results mirror those obtained from varying α. As β increases, the size of the (α, β)-community decreases, resulting in enhanced efficiency for each algorithm.

Exp3: Varying values of ξ and ζ. To assess the impact of ξ and ζ values, we conduct experiments on the *YouTube* dataset, varying values of ξ and ζ from 20 to 60. We set degree constraints $\alpha = \beta = 5$. Fig. 7 presents the experimental results. The results indicate that, in comparison to baseline algorithms, the EFA++ algorithm is more accommodating to queries when the ξ and ζ is large. For instance, when $\xi = \zeta = 40$, EFA++ exhibits the smallest execution time among the tested settings. This is attributed to the time complexity of the Expand-and-Filter algorithm, which depends on the size difference between the (α, β)-community and the size constraints (ξ, ζ). Since the time complexities of EPF and FPA are exponential concerning the sum of ξ and ζ, the computational demand increases as ξ and ζ increase. Even when queries have values of ξ and ζ are small (i.e., $\xi = \zeta = 20$), EFA++ remains more efficient than its competitors.



(a) Varying α (b) Varying β

Fig. 6. Varying α and β.

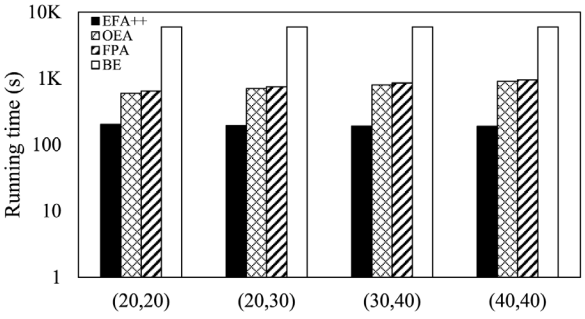


Fig. 7. Varying ξ and ζ.

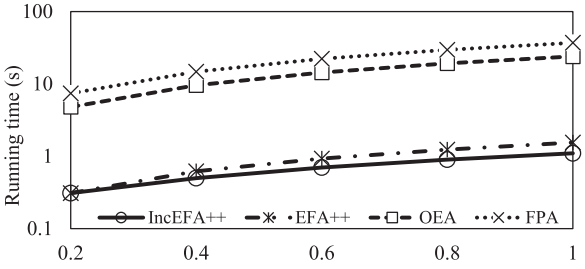


Fig. 8. Performance of incremental edge insertion.

Exp4: Experiments on the incremental algorithm. For the maintenance of the size constraint (α, β)-community, we conducted experiments on the *Youtube* Dataset. Specifically, we randomly removed 5000 distinct edges from the graph and documented the average processing time for each edge removal. In Fig. 8, the depicted running times of various algorithms exhibit variations as the changing ratio ranges from 0.2, 0.4, 0.6, 0.8 to 1.0. Notably, incremental EFA++ consistently outperforms other algorithms. This superior performance is attributed to its ability to identify the Affected Area and thereby avoid the recomputation of all updates.

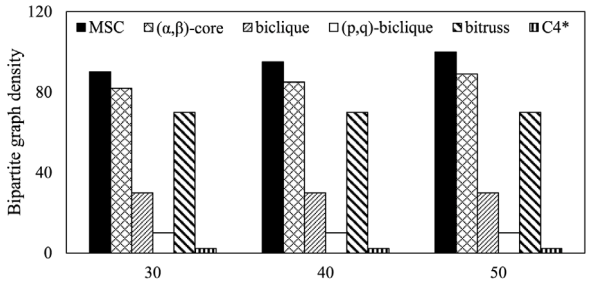


Fig. 9. Bipartite graph density.

Table 4
Statistics of query results.

Model	$\|U\|$	$\|V\|$	Co
MSCC	50	50	17.3
(α, β)-core	1,925	609	14.4
biclique	27	32	14.6
(p,q)-biclique	10	10	10
bitruss	55	47	13.4
C_{4*}	18,836	954	3.7

6.2. Effectiveness evaluation

In this section, we evaluate the effectiveness of our model on *Youtube* dataset, which contains 293 K memberships between 94 K users subscribed to 30 K Youtubers. We compare the maximal size constraint community with the (α, β)-core, k-bitruss, maximal biclique, and (p, q)-biclique models. We also add a community C_{4*} similar with [9], which is the induced subgraph of all the movies with average ratings at least 4. Note that, when considering different models, the connected component will be regarded as the result.

Exp5: Evaluating the community quality. Suppose a user on YouTube wants to find the top 30, 40, and 50 friends who share common interests with a YouTuber. Fig. 9 illustrates the bipartite graph density, computed as $d(G) = |E(G)|/\sqrt{|U(G)||V(G)|}$. Since only our model incorporates size constraints, we return the community with the size constraints of other models without considering the maximal cohesiveness constraint. We observe that the community produced by our MSCC has the best average density when varying from the top 30 vertices to the top 50. This indicates that the maximal size constraint community can consistently yield a group of users with greater cohesiveness while adhering to the size and degree constraints.

Exp6: Case study. We also perform a query using degree constraints $\alpha = 10$ and $\beta = 20$ on *YouTube*, along with size constraints $\xi = 50$ and $\zeta = 50$. The statistics of query results with $q = 2423$ are presented in Table 4. Here, $|U|$ and $|V|$ represent the total number of users and YouTubers in the corresponding community model, respectively, and Co indicates the cohesiveness of each query result. Our community model successfully identifies a community that satisfies both the degree and size constraints. In contrast, other community models yield communities with thousands and tens of thousands of vertices, violating the specified size constraints and rendering the results impractical.

7. Related work

To the best of our knowledge, this paper is the first work to study the maximal size constraint community search over bipartite graphs. Below we review closely related areas, including community search over unipartite graphs and community search over bipartite graphs.

Community Search over Unipartite Graphs. Local community search has been studied extensively in recent years [2], and the community can be modeled as k-core, k-truss [21–23] and k-clique [1,24]. In this work, we consider the size constraint over the bipartite graphs and its maintenance, so we will introduce the community search problem considering size and dynamic changing. Li et al. [25] developed an efficient progressive algorithm, namely PSA to tackle the minimum k-core search. This algorithm is designed to locate the smallest community that incorporates a series of query vertices. On dynamic changing scenario, Li et al. [26] investigated the identification of persistent communities within temporal networks. Li et al. [27] introduced the most active community search problem, aiming to identify a maximally expanded community, subject to specified connectivity criteria. Li et al. [28] delved into the cohesive subgraph search problem within large temporal graphs. Tang et al. [29] focused on the reliable community search problem in dynamic networks. As for the community search with size constraint, Ma et al. [11] investigated the problem of size-constraint k-core group query in social networks. Yao et al. [12] presented the size-bounded community search problem, which aims to find the community with largest min-degree containing a query vertex, and the community size falls within a certain range. However, their focus on unipartite graphs differs from our problem; thus, their solutions are not directly applicable to our bipartite graph context. Even if their methods were adapted to our scenario, they would still lack efficiency due to their reliance on an iterative approach, which leads to exponential time complexity.

Community Search over Bipartite Graphs. (α, β)-core was first introduced in [30] to describe the cohesiveness of the nodes in bipartite graphs, which extends the concept of k-core model in unipartite graphs. [31] discussed the properties of (α, β)-core and [6] proposed an improved fault-tolerant group recommendation method based on (α, β)-core. Liu et al. [7,8] proposed a BiCore-index to support efficient and frequently optimal (α, β)-core computation over large bipartite graphs, where BiCore-index recorded all (α, β)-core with every possible α and β in an efficient way. Motivated by skyline queries, Zhang et al. [10] designed a novel Pareto-optimal (α, β)-community model on vertex-weighted bipartite graphs. Wang et al. [9] first focused on the significant community search problem over bipartite graphs, where the significant community satisfied the cohesiveness constraints, i.e., α, β degree and maximal constraints, i.e., the community with the largest minimum edge weights, and proposed an index-based method to first solve the community search, then based on this index, a peeling and expand method is utilized to solve the significant community search. Li et al. [32] considered the keyword cohesiveness (i.e., its vertices share common keywords) and the structural cohesiveness (i.e., (α, β)-core) of the community over bipartite graphs. Xu et al. [33] focused on the attributed community search over edge-weighted bipartite graphs. Guan et al. [34] considered the privacy problem of (α, β)-core queries. They constructed two privacy preserving schemes with different levels of security to handle (α, β)-core queries over the bipartite graph under the two-server setting. Apart from the (α, β)-core, there are some related works focus on biclique [35–40], and bitruss [17,18,41,42]. Among them, He et al. [43] proposed a novel τ-strengthened (α, β)-core model to consider both tie strength and vertex engagement on bipartite graphs. Another noteworthy study by Yang et al. in [16] examined the problem of enumerating and counting (p, q)-bicliques, where p and q represent the sizes of each layer in a bipartite graph. To address this problem, they proposed a branch-and-bound approach. However, the worst-case time complexity of their method is exponential in terms of either p or q. Adapting this solution to our problem would clearly be inefficient due to its prohibitive computational demands. Overall, most existing studies neglect the size constraint community search in bipartite graphs, a critical aspect in applications like personalized recommendation, fraud detection, and team formation. The few that address size constraints focus on bicliques and suffer from significant inefficiencies.

8. Conclusion

In this paper, we study the problem of maximal size constraint community search over bipartite graphs. To efficiently solve this problem, we first propose two competitive algorithms, namely OEA and FPA. To further improve the efficiency, we propose an advanced expand-and-filter algorithm. This innovative algorithm converts the community search problem into a dynamic programming problem. The process involves the utilization of transformed networks, expanding to a boundary community, followers computation, and dynamic programming techniques. Extensive experiments conducted over 10 real datasets demonstrate the superior performance of our proposed methods against the state-of-the-art approaches.

CRediT authorship contribution statement

Mo Li: Writing – original draft, Methodology, Investigation, Formal analysis, Conceptualization. **Renata Borovica-Gajic:** Writing – review & editing, Validation, Investigation. **Farhana M. Choudhury:** Writing – review & editing, Validation, Formal analysis. **Ningning Cui:** Writing – review & editing, Investigation. **Linlin Ding:** Funding acquisition, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This study was funded by the National Key Research and Development Program of China (No. 2022YFC3004603); National Natural Science Foundation of China (No. 62072220); Natural Science Foundation of Liaoning Province, China (Nos.2022-KF-13-06, 2022-BS-111); the Youth Talent Support Program of 'Xing Liao Talent Program', China (No. XLYC2203003); Liaoning Provincial Department of Education Youth Project, China (No. JYTQN2023189). We would also like to thank the authors of [16] for providing the baseline source code.

References

[1] L. Yuan, L. Qin, W. Zhang, L. Chang, J. Yang, Index-based densest clique percolation community search in networks, IEEE Trans. Knowl. Data Eng. 30 (5) (2018) 922–935.
[2] Y. Fang, X. Huang, L. Qin, Y. Zhang, W. Zhang, R. Cheng, X. Lin, A survey of community search over big graphs, VLDB J. 29 (1) (2020) 353–392.
[3] L. Sun, X. Huang, R. Li, B. Choi, J. Xu, Index-based intimate-core community search in large weighted graphs, IEEE Trans. Knowl. Data Eng. 34 (9) (2022) 4313–4327.
[4] H. Zhao, P. Rui, J. Chen, Y. Zhang, Y. Wang, S. Zhao, J. Tang, Hinchip: Heterogeneous information network representation with community hierarchy preserving, Knowl.-Based Syst. 264 (2023) 110343.
[5] J. Park, K. Lee, H. Kwon, Two-level graph representation learning with community-as-a-node graphs, in: ICDM, 2023, pp. 1259–1264.
[6] D. Ding, H. Li, Z. Huang, N. Mamoulis, Efficient fault-tolerant group recommendation using alpha-beta-core, in: CIKM, 2017, pp. 2047–2050.
[7] B. Liu, L. Yuan, X. Lin, L. Qin, W. Zhang, J. Zhou, Efficient (α, β)-core computation: An index-based approach, in: WWW, 2019, pp. 1130–1141.
[8] B. Liu, L. Yuan, X. Lin, L. Qin, W. Zhang, J. Zhou, Efficient (α, β)-core computation in bipartite graphs, VLDB J. 29 (5) (2020) 1075–1099.
[9] K. Wang, W. Zhang, X. Lin, Y. Zhang, L. Qin, Y. Zhang, Efficient and effective community search on large-scale bipartite graphs, in: ICDE, 2021, pp. 85–96.
[10] Y. Zhang, K. Wang, W. Zhang, X. Lin, Y. Zhang, Pareto-optimal community search on large bipartite graphs, in: CIKM, 2021, pp. 2647–2656.
[11] Y. Ma, Y. Yuan, F. Zhu, G. Wang, J. Xiao, J. Wang, Who should be invited to my party: A size-constrained k-core problem in social networks, J. Comput. Sci. Tech. 34 (1) (2019) 170–184.
[12] K. Yao, L. Chang, Efficient size-bounded community search over large networks, in: PVLDB, vol. 14, (8) 2021, pp. 1441–1453.
[13] C. Xu, J. Si, Z. Guan, W. Zhao, Y. Wu, X. Gao, Reliable conflictive multi-view learning, in: AAAI, 2024, pp. 16129–16137.
[14] Y. Liang, Q. Song, Z. Zhao, H. Zhou, M. Gong, BA-GNN: behavior-aware graph neural network for session-based recommendation, Front. Comput. Sci. 17 (6) (2023) 176613.
[15] C. Li, X. Guo, W. Lin, Z. Tang, J. Cao, Y. Zhang, Multiplex network community detection algorithm based on motif awareness, Knowl.-Based Syst. 260 (2023) 110136.
[16] J. Yang, Y. Peng, W. Zhang, (p,q)-biclique counting and enumeration for large sparse bipartite graphs, in: PVLDB, vol. 15, (2) 2022, pp. 141–153.
[17] K. Wang, X. Lin, L. Qin, W. Zhang, Y. Zhang, Towards efficient solutions of bitruss decomposition for large-scale bipartite graphs, VLDB J. 31 (2) (2022) 203–226.
[18] K. Wang, X. Lin, L. Qin, W. Zhang, Y. Zhang, Vertex priority based butterfly counting for large-scale bipartite networks, in: PVLDB, vol. 12, (10) 2019, pp. 1139–1152.
[19] M. Danisch, O. Balalau, M. Sozio, Listing k-cliques in sparse real-world graphs, in: WWW, 2018, pp. 589–598.
[20] N. Chiba, T. Nishizeki, Arboricity and subgraph listing algorithms, SIAM J. Comput. 14 (1) (1985) 210–223.
[21] Z. Zheng, F. Ye, R. Li, G. Ling, T. Jin, Finding weighted k-truss communities in large networks, Inform. Sci. 417 (2017) 344–360.
[22] B. Liu, F. Zhang, W. Zhang, X. Lin, Y. Zhang, Efficient community search with size constraint, in: ICDE, 2021, pp. 97–108.
[23] T. Cai, J. Li, N.A.H. Haldar, A. Mian, J. Yearwood, T. Sellis, Anchored vertex exploration for community engagement in social networks, in: ICDE, 2020, pp. 409–420.
[24] Y. Fang, K. Yu, R. Cheng, L.V.S. Lakshmanan, X. Lin, Efficient algorithms for densest subgraph discovery, in: PVLDB, vol. 12, (11) 2019, pp. 1719–1732.
[25] C. Li, F. Zhang, Y. Zhang, L. Qin, W. Zhang, X. Lin, Efficient progressive minimum k-core search, in: PVLDB, vol. 13, (3) 2019, pp. 362–375.
[26] R. Li, J. Su, L. Qin, J.X. Yu, Q. Dai, Persistent community search in temporal networks, in: ICDE, 2018, pp. 797–808.
[27] L. Li, Y. Zhao, Y. Li, F. Wahab, Z. Wang, The most active community search in large temporal graphs, Knowl.-Based Syst. 250 (2022) 109101.
[28] Y. Li, J. Liu, H. Zhao, J. Sun, Y. Zhao, G. Wang, Efficient continual cohesive subgraph search in large temporal graphs, WWW J. 24 (5) (2021) 1483–1509.
[29] Y. Tang, J. Li, N.A.H. Haldar, Z. Guan, J. Xu, C. Liu, Reliable community search in dynamic networks, in: PVLDB, vol. 15, (11) 2022, pp. 2826–2838.
[30] A. Ahmed, V. Batagelj, X. Fu, S.-H. Hong, D. Merrick, A. Mrvar, Visualisation and analysis of the internet movie database, in: APVIS, 2007, pp. 17–24.
[31] M. Cerinšek, V. Batagelj, Generalized two-mode cores, Social Networks 42 (2015) 80–87.
[32] D. Li, X. Liang, R. Hu, L. Zeng, X. Wang, (α, β)-AWCS: (α, β)-attributed weighted community search on bipartite graphs, in: IJCNN, 2022, pp. 1–8.
[33] Z. Xu, Y. Zhang, L. Yuan, Y. Qian, Z. Chen, M. Zhou, Q. Mao, W. Pan, Effective community search on large attributed bipartite graphs, Int. J. Pattern Recognit. Artif. Intell. 37 (2) (2023) 2359002:1–2359002:25.
[34] Y. Guan, R. Lu, Y. Zheng, S. Zhang, J. Shao, G. Wei, Achieving efficient and privacy-preserving (α, β)-core query over bipartite graphs in cloud, IEEE Trans. Dependable Secur. Comput. 20 (3) (2023) 1979–1993.
[35] L. Chen, C. Liu, R. Zhou, J. Xu, J. Li, Efficient maximal biclique enumeration for large sparse bipartite graphs, in: PVLDB, vol. 15, (8) 2022, pp. 1559–1571.
[36] B. Lyu, L. Qin, X. Lin, Y. Zhang, Z. Qian, J. Zhou, Maximum and top-k diversified biclique search at scale, VLDB J. 31 (6) (2022) 1365–1389.
[37] K. Wang, W. Zhang, X. Lin, L. Qin, A. Zhou, Efficient personalized maximum biclique search, in: ICDE, 2022, pp. 498–511.
[38] L. Chen, C. Liu, R. Zhou, J. Xu, J. Li, Efficient exact algorithms for maximum balanced biclique search in bipartite graphs, in: SIGMOD, 2020, pp. 248–260.
[39] B. Lyu, L. Qin, X. Lin, Y. Zhang, Z. Qian, J. Zhou, Maximum biclique search at billion scale, in: PVLDB, vol. 13, (9) 2020, pp. 1359–1372.
[40] Y. Wang, S. Cai, M. Yin, New heuristic approaches for maximum balanced biclique problem, Inform. Sci. 432 (2018) 362–375.
[41] K. Wang, X. Lin, L. Qin, W. Zhang, Y. Zhang, Efficient bitruss decomposition for large-scale bipartite graphs, in: ICDE, 2020, pp. 661–672.
[42] Y. Wang, R. Xu, X. Jian, A. Zhou, L. Chen, Towards distributed bitruss decomposition on bipartite graphs, in: PVLDB, vol. 15, (9) 2022, pp. 1889–1901.
[43] Y. He, K. Wang, W. Zhang, X. Lin, Y. Zhang, Exploring cohesive subgraphs with vertex engagement and tie strength in bipartite graphs, Inform. Sci. 572 (2021) 277–296.