Size-bounded Community Search over Large Bipartite Graphs

Yuting Zhang
University of New South Wales
Data61, CSIRO
Sydney, Australia
yutingz@cse.unsw.edu.au

Kai Wang
Shanghai Jiao Tong University
Shanghai, China
w.kai@sjtu.edu.cn

Wenjie Zhang
University of New South Wales
Sydney, Australia
zhangw@cse.unsw.edu.au

Wei Ni
Data61, CSIRO
Sydney, Australia
wei.ni@data61.csiro.au

Xuemin Lin
Shanghai Jiao Tong University
Shanghai, China
xuemin.lin@sjtu.edu.cn

ABSTRACT

In real-world applications, bipartite graphs are commonly used to represent relationships between two types of entities. However, existing studies in community discovery on bipartite graphs have overlooked the importance of considering the size of the resulting community, which can reflect practical constraints such as venue capacity or team size. In this paper, we address this limitation by introducing the size-bounded (α, β)-community (SABC) model, which incorporates both structural cohesiveness and size constraints in bipartite graphs. Our model has applications in various domains such as team formation, biological network analysis, and suspicious group identification. To enable efficient query processing on large graphs, we propose shrink-based and expand-based algorithms, complemented by reduction rules and vertex ordering strategies. Our extensive experiments conducted on 9 real-world datasets demonstrate the effectiveness and efficiency of our proposed model, query processing algorithms, and optimizations.

1 INTRODUCTION

Bipartite graphs are widely adopted to model relationships between two distinct types of entities. In bipartite graphs, vertices are divided into two disjoint sets (i.e., the upper and lower layers) and edges only exist between vertices from different layers. Such a structure allows bipartite graphs to capture many real-life interactions, such as author-paper networks [13], customer-product networks [31], and gene-property networks [30]. Community structures naturally reside in these real-life bipartite networks and community search, which retrieves densely connected subgraphs containing given query vertices, has recently been explored following different cohesive subgraph models of bipartite graphs (e.g., (α, β)-core [7, 18, 35, 36, 45], bitruss [28, 33, 39, 47], and biclique [6, 16, 17, 24, 42]). Community search over bipartite graphs has been proved useful in many real-world applications, such as personalized recommendations [19, 36], team formation [28], and fraud detection [3, 19]. Previous studies also reveal that the size of retrieved communities can be quite large and unpredictable [15, 23, 29], thus hindering the usefulness of community search models. For instance, on the MovieLense dataset (https://grouplens.org/datasets/movielens/25m/), when querying the (α, β)-core with $\alpha = \beta = 10$, the returned subgraph contains 162,539 users and 24,330 movies.

Size-bounded community search has been extensively studied on *general (unipartite) graphs* in recent years including size-bounded k-core model [23, 29, 41] and size-bounded k-truss model [20]. These studies demonstrate the size constraint of a community is natural and crucial in many real-world applications especially when query vertices are involved (e.g., a host wants to organize a party with 10 guests or forming a team with 12 group members). Despite the success of size-bounded community search models over unipartite graphs, these models cannot be directly applied to bipartite graphs since they do not consider the unique structure of bipartite graphs where two vertex sets with different semantics are involved and a single-value size constraint based models for unipartite graphs are no longer appropriate.

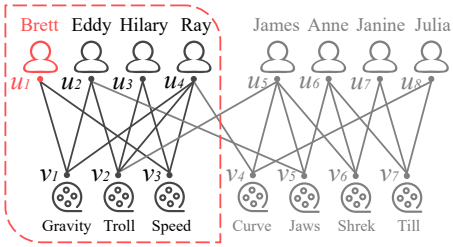


Figure 1: A user-movie network

Motivated by the observations above, in this paper, we are the first to study the size-bounded community search problem on bipartite graphs to find high-quality communities based on the well-studied (α, β)-core model. Specifically, given a bipartite graph G, a query vertex q, degree constraints α and β, and size constraints $\mathcal{S}_U$ and $\mathcal{S}_L$, we aim to find a size-bounded $(\underline{\alpha}, \underline{\beta})$-community (SABC) which is a connected subgraph $H \subseteq G$ containing q, satisfying the following conditions: (1) cohesiveness constraint: the degree of all upper layer vertices in H is at least α, and the degree of all lower layer vertices in H is at least β, (2) size constraint: the number of vertices in upper and lower layer of H is no larger than $\mathcal{S}_U$ and $\mathcal{S}_L$, respectively. The intuition behind the new SABC model is to utilize the widely adopted (α, β)-core model to capture structural cohesiveness and incorporate size constraints to restrict the size of the resultant community. Note that the two disjoint sets of vertices in a bipartite graph represent distinct entities and hold different application semantics, it is more appropriate to impose different size constraints on two layers. Consider the example in Figure 1. If the (α, β)-core model is applied to search a (2,3)-community of "Brett", the entire graph in the figure will be returned, which includes 8 users and 7 movies. However, if "Brett" only has time for at most three movies and wishes to engage in discussions with up to three other users due to venue capacity, our model with size constraints $\mathcal{S}_U = 4$

and $\mathcal{S}_L = 3$ can provide a more realistic solution. It can find the community in the red circle, which exclusively consists of users "Brett", "Eddy", "Hilary", and "Ray", along with movies "Gravity", "Troll", and "Speed". In addition, the (α, β)-core model naturally implies a lower bound over the number of upper and lower layer vertices, which are β and α, respectively.

Applications. Finding SABC has many real-world applications, and we list some of them below.

• *Team Formation.* In user-movie networks (e.g., MovieLense and IMDB), a host may wish to organize a group activity with users having similar interests in movies and share some movies together. As shown in [18], cohesive models such as (α, β)-core can be used to find the desired community. However, when considering factors such as venue capacity and activity duration, it becomes crucial to include size constraints alongside cohesive constraints when searching for the community. Thus, the SABC model can effectively provide personalized recommendations to the host, suggesting a group with a desired number of attendees and movies.

• *Biological Network Analysis.* In gene-property networks, the upper layer represents genes and the lower layer represents properties. A key task in biology is to identify functional modules composed of genes and corresponding properties, which are cohesive subgraphs in the gene-property network [26]. A recent study [30] demonstrates that the (α, β)-core model can be used to identify genes with similar properties. Li et al. [14] reveal that only small cohesive subgraphs can help to find a group with homophily property as the query vertex. Therefore, based on a given property or gene, the SABC model can effectively locate relevant properties or genes associated with disease susceptibility or drug response.

• *Suspicious Group Identification.* In online shopping platforms like Amazon or eBay, users and items naturally form a bipartite network. A group of malicious users tend to give a large number of fake ratings to some of the items they want to promote. A recent study [36] highlights the effectiveness of the (α, β)-core model in identifying such groups of malicious users and the associated products. However, one limitation of the (α, β)-core model is that it often generates large communities, which may include numerous innocent users. This, in turn, can increase the costs and efforts required for human checks and investigations. By constraining the size of the resulting community, our model allows a more targeted and efficient identification of malicious users and associated items.

Challenges and our solutions. In this paper, we aim to develop efficient algorithms for the size-bounded community (SABC) search problem over bipartite graphs.

We prove that the size-bounded (α, β)-community search problem is NP-hard. The high computational complexity makes it difficult to retrieve an exact solution for SABC model within a reasonable time. A naive solution to this problem is to evaluate all possible combinations of vertices in G to identify a community that satisfies the degree and size constraints. However, this approach is impractical as it will involve exploring many unpromising combinations. Therefore, it is crucial to design effective searching strategies and pruning techniques to avoid unpromising combinations and narrow down the search space.

Motivated by the above challenges, we propose both shrink-based and expand-based algorithms to efficiently obtain the results. The shrink-based search strategy quickly finds a feasible solution by starting from the entire (α, β)-core and removing vertices until the remaining graph forms a SABC. Following the observation that the result tends to be close to the query node, we also propose an expand-based search strategy that iteratively expands from the query vertex q to form a SABC. In addition, we propose three pruning rules to effectively limit the search space and eliminate unpromising branches during the search process. Furthermore, we propose two score functions based on the distribution of their neighbors in the current partial solution. This enables each iteration in the expand-based approach to select the most promising vertex, and each iteration in the shrink-based approach to remove the most unpromising vertex.

Contributions. Our principal contributions in this paper are summarized below.

• *A new community model.* Motivated by real-life requirements, we apply the size constraint to the (α, β)-community model. The paper is the first to study the problem of size-bounded community search on bipartite graphs.

• *Online search algorithm.* We develop efficient Shrink and Expand algorithms to query SABC with given size constraints, degree constraints and query vertex.

• *Novel pruning rules and vertex selection strategies.* To alleviate the expensive online searching cost, we design several pruning rules to limit the search space, reduce the number of search branches, and optimise the visiting order of vertices.

• *Extensive experimental studies.* We conduct extensive experiments over 9 real-world graphs to evaluate the effectiveness of the proposed model and the efficiency of our algorithms.

Organization. The rest of the paper is organized as follows. Section 2 defines the problem. Section 3 introduces the shrink-based algorithm and optimizes it with a pruning rule and the vertex selection technique. Section 4 introduces the expand-based algorithm and its optimizations. Section 5 reports experimental results. Section 6 reviews the related work. Section 7 concludes the paper.

2 PROBLEM DEFINITION

In this section, we formally introduce notations and definitions. Mathematical notations used throughout this paper are summarized in Table 1.

Table 1: The summary of notations

Notation	Definition
G	a bipartite graph
$V(G)/E(G)$	the vertex/edge set of G
$U(G)/L(G)$	the upper/lower layer of G
n, m	the number of vertices and edges in G ($m > n$)
u, v	a vertex in a bipartite graph
$(u, v), e$	an edge in a bipartite graph
H	the size-bounded (α, β)-community of q in G
$N_G(u)$	the set of neighbors of u in G
$deg_G(u)$	the degree of vertex u in G
$G_{D,q}$	the subgraph induced by q and D-hop neighbor of q in G
$dist_G(u, v)$	the shortest distance between u and v in G

Our problem is defined over a bipartite graph $G(V{=}(U, L), E)$, where $U(G)$ denotes the set of vertices in the upper layer, $L(G)$ denotes the set of vertices in the lower layer, and $E(G)$ denotes the set of edges. We have $E(G) \subseteq U(G) \times L(G)$, and $U(G) \cap L(G) = \emptyset$. An edge e between two vertices u and v in G is denoted as $e = (u, v)$ or (v, u). The set of neighbors of a vertex u in G

is denoted as $N_G(u) = \{v \in V(G) \mid (u,v) \in E(G)\}$, and the degree of u is denoted as $deg_G(u) = |N_G(u)|$. We use n and m to denote the number of vertices and edges in G, respectively. Before formally defining the problem, we introduce the following critical concepts.

Definition 2.1. **(α, β)-core.** Given a bipartite graph G and degree constraints α and β, a subgraph is an (α, β)-core of G if all the vertices in the upper layer have degree at least α and all the vertices in the lower layer have degree at least β; An (α, β)-core is maximal if any supergraph of it is not an (α, β)-core.

Based on the (α, β)-core model, our proposed model introduces additional size constraints to meet the query requirements in real-life scenarios. Since the upper layer $U(G)$ and lower layer $L(G)$ in a bipartite graph represent two distinct sets of entities, we use parameters $\mathcal{S}_U$ and $\mathcal{S}_L$ to represent the size constraints to the upper and lower layers respectively.

Definition 2.2. **Size-bounded (α, β)-community (SABC).** Given a bipartite graph G, a query vertex q, degree constraints α and β, and size constraints $\mathcal{S}_U$ and $\mathcal{S}_L$, a subgraph H is a size-bounded (α, β)-community of G if it satisfies the following constraints.
(1) Connectivity constraint. H is connected and $q \in H$;
(2) Cohesiveness constraint. $deg_H(u) \geq \alpha$ for each $u \in U(H)$ and $deg_H(v) \geq \beta$ for each $v \in L(H)$;
(3) Size constraint. $|U(H)| \leq \mathcal{S}_U$ and $|L(H)| \leq \mathcal{S}_L$.

Problem Statement (SABC Search). Given a bipartite graph G, a query vertex q, degree constraints α and β, and size constraints $\mathcal{S}_U$ and $\mathcal{S}_L$, we aim to find a size-bounded (α, β)-community in G.
Problem Hardness. We prove the NP-hardness of the SABC search.

Theorem 2.3. *The* SABC *search problem is NP-hard.*

Proof. We prove this by reducing the k-biclique search problem [17]. Given a bipartite graph G, and a integer k, the k-biclique search problem aims to find the biclique $K_{k,k}$ in G having $|U(K_{k,k})| = k$ and $|L(K_{k,k})| = k$. Now, given an instance of the k-biclique search problem which takes a graph $G(V{=}(U, L), E)$ and an integer k and aims to determine whether G has a k-biclique. We reduce it into a SABC problem as follows. We add the dummy vertex v_q and edges between v_q and every vertex in the other layer. Without loss of generality, we assume v_q is in the upper layer. Denote the resulting graph as G', i.e., $U(G') = U \cup \{v_q\}$, $L(G') = L(G)$ and $E(G') = E \cup \{(v_q, v) | v \in L(G)\}$. The query of SABC problem consists of a query vertex v_q, size constraints $(k+1, k)$, and degree constraints $(k, k+1)$. It is easy to verify that a subgraph $H \subseteq G$ is a k-biclique if and only if $H \cup v_q$ is a solution to the SABC problem. This is because the given constraints require every vertex in one layer to be connected to every vertex in the other layer in the SABC. And put v_q aside, the rest of the subgraph is a fully connected subgraph with k vertices in both layers. Therefore, since the k-biclique search problem is NP-hard, our SABC search problem is also NP-hard. □

Remark. Note that given a specific query vertex and constraints of degree and size, there can be multiple possible results that satisfy the given criteria. However, due to the inherent complexity as discussed above, it is impractical to cover all the possible results, and we aim to retrieve one of them. To simplify our discussion, we refer to a subgraph by its vertices set in the following.

3 SHRINK-BASED APPROACHES

3.1 A basic shrink-based approach

According to Definition 2.1 and Definition 2.2, a SABC must be a connected subgraph of the (α, β)-core containing q. Following this idea, we introduce a shrink-based approach that gradually peels the input graph to a connected subgraph of (α, β)-core containing q and checks whether it meets the cohesiveness and size constraints.

Let G' represent the connected subgraph of the (α, β)-core that contains q in G. We recursively select a vertex to remove and maintain G' as a connected subgraph satisfying the cohesiveness constraint and containing q. However, there is a situation where removing a vertex v would result in G' no longer satisfying the cohesiveness constraints. In such cases, we recover G' and avoid removing v in the subsequent recursions to ensure the degree constraints for q. We use set C to record the vertices that cannot be removed. This process gives our shrink-based algorithm a recursive nature, resembling a binary tree. During each recursion, we attempt to remove a vertex from G' that is not in C, while ensuring that all vertices in C remain in G'. The recursion continues until G' satisfies the size constraints, at this point, G' represents a SABC.

In Algorithm 1, we present details of the shrink-based approach. The algorithm begins by initializing a set C with q, which records the vertices that must be preserved to maintain q satisfying the degree constraints (Line 1). Then, we obtain G' as the connected component of (α, β)-core in G that contains q (Line 2). If G' is empty, an empty set is returned, indicating that no result exists (Lines 3 - 4). Otherwise, we initialize H as G' (Line 5) and find the result by calling the SSearch function (Line 6). SSearch recursively determines whether a vertex v can be removed (Line 8). In each iteration, a selected vertex is attempted to be removed from G' (Line 12). Note that removing v may lead to the removal of other vertices that do not satisfy the degree constraints, and we describe this process in Algorithm 2. If C remains a subset of G' after the removal and G' satisfies the size constraints, we return G' as a result (Lines 13 - 15). Otherwise, if the size of G' still exceeds the size constraints, the SSearch function is recursively called to further shrink G' (Line 16). Note that if the removal of selected vertex results in cascading removals of any vertex in C, the algorithm recovers G' by adding back all the vertices removed during the Peeling process (Line 17). The vertex v is then included in C as it must be preserved to ensure that q is part of the final result (Line 18).

Algorithm 2 shows the pseudo-code of the Peeling algorithm, which is invoked at Line 12 in Algorithm 1. Firstly, it initializes a queue Q and adds the selected vertex v to it (Line 1). Next, v is removed from G' (Line 2), and a Breadth First Search (BFS) approach is employed to iteratively remove vertices from G' based on the vertices in Q (Lines 3 - 10). In each iteration, it dequeues the first vertex u' from Q (Line 4). Then it visits the neighborhood vertices of u' and removes the one that does not meet the degree constraint (Lines 6 - 7). If u' is a vertex in C, which means u' cannot be removed to keep q still in the (α, β)-core. The algorithm terminates to avoid unnecessary peeling and back-track steps for the remaining vertices in G' and returns G' (Line 8). If v' can be removed from G', it added v' into Q (Line 10). Once the iterations are complete, the connected component in G' that contains the query vertex is returned.

Example 3.1. Consider the bipartite graph G in Figure 1. Suppose $q = u_1$, $\alpha = 2$, $\beta = 3$, $\mathcal{S}_U = 4$, and $\mathcal{S}_L = 3$. Initially, $C = \{u_1\}$,

Algorithm 1: Shrink

Input: G, q, α, β, $\mathcal{S}_U$, and $\mathcal{S}_L$;
Output: H;
1 $C \leftarrow \{q\}$;
2 $G' \leftarrow$ the connected component of (α, β)-core containing q;
3 **if** G' *is empty* **then**
4 | **return** $\emptyset$;
5 $H \leftarrow G'$;
6 $H \leftarrow$ SSearch(G', C);
7 **return** H;

8 **Function** SSearch(G', C)
9 | **if** *a valid solution* H *is found* **then**
10 | | **return** H;
11 | **foreach** $v \in V(G') \setminus C$ **do**
12 | | $G' \leftarrow$ Peeling(G', C, v);
13 | | **if** $C \subseteq V(G')$ **then**
14 | | | **if** $|U(G')| \leq \mathcal{S}_U$ *and* $|L(G'))| \leq \mathcal{S}_L$ **then**
15 | | | | **return** G';
16 | | | $H \leftarrow$ SSearch(G', C);
17 | | recover v and the vertices removed with v into G';
18 | | $C \leftarrow C \cup \{v\}$;
19 | | **if** $|U(C)| > \mathcal{S}_U$ *or* $|L(C)| > \mathcal{S}_L$ **then**
20 | | | **return** $\emptyset$;

Algorithm 2: Peeling

Input: a graph G', the set C of vertices that must be included, the vertex v to be removed;
Output: updated G';
1 $Q \leftarrow \emptyset$; $Q.push(v)$;
2 remove v from G';
3 **while** $Q \neq \emptyset$ **do**
4 | $u' \leftarrow Q.pop()$;
5 | **foreach** $v' \in N_{G'}(u')$ **do**
6 | | **if** v' *does not have enough degree* **then**
7 | | | remove v' from G';
8 | | | **if** $v' \in C$ **then**
9 | | | | **return** G';
10 | | | $Q.push(v')$;
11 remove isolated vertices from G';
12 **return** G';

and $G' = G$ since G is already a $(2, 3)$-core. Assume the visiting order of vertices at Line 11 is vertex ID and alternate between two layers. We first remove v_1 from G' using the Peeling algorithm. After v_1 is removed, the degree of u_1 does not meet the degree constraint to the upper layer, which is 2. So u_1 is also removed from G. Since $u_1 \in C$, which cannot be removed, the Peeling algorithm returns. Then we add all the vertex removed in this Peeling algorithm (v_1 and u_1) back to G' and also add v_1 into C. Then the same process is repeated for u_2, v_2, u_3, v_3, and u_4. When we select to remove u_5, vertices v_4, v_5, v_6, u_8, u_6, u_7, and v_7 are also be removed along with u_5. At this step, G' satisfies the size constraints $\mathcal{S}_U = 4$, and $\mathcal{S}_L = 3$. Thus the algorithm returns G' as H and terminates.

Correctness Analysis. Suppose there is a solution H containing C. There are two conditions for the vertex v to be removed at Line 11 in Algorithm 1, either it is in H (i.e., $v \in H$) or it is not in H (i.e., $v \in V(G') \setminus V(H)$). 1) If the selected vertex v is in H, the v will be added to C at Line 18 in Algorithm 1 unless there exist other solutions. Then it continues invoking itself at Line 16 in Algorithm 1 to compute a solution based on G' and C. 2) If the selected vertex v is not in H. Then removing v will not lead to the removal of any vertex in C by the degree constraints. Thus, before the remaining graph G' satisfies the size constraints, all the vertex $v \in V(G') \setminus V(H)$ will be examined recursively. Therefore, Algorithm 1 can correctly solve the SABC search problem.

Time Complexity. In Algorithm 1, the Peeling algorithm invoked at Line 11 takes $O(|E(G')|)$ time to peel G' to a connected subgraph satisfying the cohesiveness constraints. Let T_i denote the total time complexity of SSearch when $|C| = i$, where $|C|$ denotes the size of set C. Thus, the total time complexity of SSearch is T_1 where C contains only the query vertex q. Since Lines 10 - 20 is executed at most $V(G')$ times, the time complexity $T_i = |V(G')| \times T_{i+1} + O(|E(G')|)$. There are at most $\mathcal{S}_U + \mathcal{S}_L$ iterations for the SSearch process, so i is bounded by $\mathcal{S}_U + \mathcal{S}_L$. Then we have $T_{|C|} = O(|V(G')|^{\mathcal{S}_U + \mathcal{S}_L - |C|} \times |E(G')|)$. At Line 5 in Algorithm 1, we have $|C| = 1$, $|E(G')| = O(m)$, and $|V(G')| = O(n)$. Thus, the time cost of Algorithm 1 is $O(n^{\mathcal{S}_U + \mathcal{S}_L - 1} \times m)$.

3.2 Improving the shrink-based approach

Although the Shrink algorithm can correctly solve the SABC search problem, we can further explore opportunities for improvement in the following aspects.

- **Reduce the search space.** The size constraint inherently imposes an upper bound on the shortest distance between two vertices u and v in H. This is because if the distance between u and v becomes too large, it implies that u and v must be connected through a significant number of intermediate vertices, which will cause H to exceed the size constraint. Thus, it is possible for us to limit the search space with a distance upper bound.
- **Optimized vertex selection order.** In each iteration, we select a vertex from G' and remove it. As the removal of one vertex has a cascading effect on all subsequent vertices that will be removed, the choice of which vertex to remove can significantly impact the remaining subgraph G'. Therefore, it is essential to devise a proper strategy for vertex ordering.

Improvement 1: Distance-based reduction. To enhance the efficiency and restrict the search space, we propose a distance-based reduction to identify and prune unpromising vertices within a given bipartite graph. Let $dist_G(u, v)$ denote the shortest distance between vertices u and v in the bipartite graph G. The diameter of a graph G is defined as the maximum shortest distance between any two vertices, i.e., $max\{dist_G(u, v)\}$. We can calculate the value of a diameter D and obtain a subgraph based on D that can cover all the possible results. To determine the appropriate diameter upper bound, we first introduce the following lemma, which establishes the relationship between diameter D and the given degree constraints α, β.

Lemma 3.2. *Consider a connected bipartite graph g, in which $deg_g(u) \geq \alpha$ for each $u \in U(g)$ and $deg_g(v) \geq \beta$ for each $v \in L(g)$. Let D denote the diameter of g. When $\alpha > 1$, $\beta > 1$, and $D > 1$, we must have $|U(g)| \geq \lfloor \frac{D-1}{4} \rfloor (\beta - 2) + \lfloor \frac{D}{2} \rfloor + \beta - 1$ and $|L(g)| \geq \lfloor \frac{D-1}{4} \rfloor (\alpha - 2) + \lfloor \frac{D}{2} \rfloor + \alpha - 1$.*

Proof. Let g be a connected bipartite graph with $deg_g(u) > 1$ for each $u \in U(g)$, $deg_g(v) > 1$ for each $v \in L(g)$, and $D > 1$. Let p be a path in the graph g such that the shortest distance between the head vertex and the tail vertex in p is D. p must exist

as the diameter of g is D. Let $u_0, u_1, ...u_x$ denote the upper layer vertices in p, and $v_0, v_1, ..., v_y$ denote the lower layer vertices in p. Let NU_s be the set of u_i's neighbors that are not in p, $s = \lfloor \frac{i}{2} \rfloor$, i.e. $NU_{\lfloor \frac{i}{2} \rfloor} = N_g(u_i) \cup N_g(u_{i+1})$, where $N_g(u)$ denotes the set of neighbors of u in g. Let NL_t be the set of v_j's neighbors that are not in p, $t = \lfloor \frac{j}{2} \rfloor$, i.e. $NL_{\lfloor \frac{j}{2} \rfloor} = N_g(v_j) \cup N_g(v_{j+1})$. Note that, the number of upper layer vertex and the lower layer vertex in p may not be divisible by 2. In such cases, the last neighbor set only includes the neighbors of the last vertex in p. Since p is the shortest distance between the head vertex and the tail vertex of this diameter, we must have $NL_t \bigcap NL_{t'} = \emptyset, \forall t \neq t'$ and $NU_s \bigcap NU_{s'} = \emptyset, \forall s \neq s'$. Otherwise, there must exist a shorter path between the head vertex and tail vertex of p with a length smaller than D. In a bipartite graph, p falls into one of the following 4 cases.

• Case 1: Start with an upper layer vertex and end with an upper layer vertex.
• Case 2: Start with an upper layer vertex and end with a lower layer vertex.
• Case 3: Start with a lower layer vertex and end with a lower layer vertex.
• Case 4: Start with a lower layer vertex and end with an upper layer vertex.

Without loss of generality, we focus on discussing Case 1 and Case 2. Case 3 and Case 4 can be proved similarly by interchanging upper vertices and lower vertices. In both Case 1 and Case 2, we have $x = \lfloor \frac{D}{2} \rfloor$, $y = \lfloor \frac{D-1}{2} \rfloor$, $s = \lfloor \frac{D}{4} \rfloor$, $t = \lfloor \frac{D-1}{4} \rfloor$.

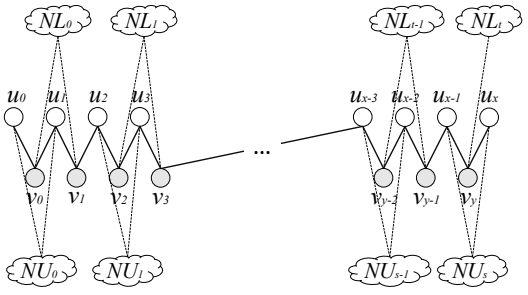


Figure 2: Proof of Lemma 3.2 - Case 1

For Case 1, $p = (u_0, v_0, u_1, v_1, ..., v_{\lfloor \frac{D-1}{2} \rfloor}, u_{\lfloor \frac{D}{2} \rfloor})$. For the vertex sets $NU_0, NU_1, ..., NU_s$, we have the following observations.

• $NU_0 \geq \alpha - 1$.
• $NU_s \geq \alpha - 2$, for $0 < s < \lfloor \frac{D}{4} \rfloor$.
• $NU_{\lfloor \frac{D}{4} \rfloor} \geq \alpha - 1$.

Thus, we can have the number of lower layer vertices in g in Case 1 is lower bounded by the number of vertices in $NU_0, NU_1, ..., NU_s$ and the lower layer vertices in path p, i.e., $|L(g)| \geq \lfloor \frac{D-1}{2} \rfloor + 1 + \sum_{s=0}^{\lfloor \frac{D}{4} \rfloor} |NU_s| \geq \lfloor \frac{D}{4} \rfloor(\alpha - 2) + \lfloor \frac{D-1}{2} \rfloor + \alpha + 1$.

For the vertex sets $NL_0, NL_1, ..., NL_t$, we observe that:

• $NL_t \geq \beta - 2$, for $0 \leq t \leq \lfloor \frac{D-1}{4} \rfloor$.

Following the logic applied to the lower layer vertices, the number of upper layer vertices in g in Case 1 is lower bounded by the number of vertices in $NL_0, NL_1, ..., NL_t$ and the upper layer vertices in path p, i.e., $|U(g)| \geq \lfloor \frac{D}{2} \rfloor + 1 + \sum_{t=0}^{\lfloor \frac{D-1}{4} \rfloor} |NL_t| \geq \lfloor \frac{D-1}{4} \rfloor(\beta - 2) + \lfloor \frac{D}{2} \rfloor + \beta - 1$.

For Case 2, $p = (u_0, v_0, u_1, v_1, ..., u_{\lfloor \frac{D}{2} \rfloor}, v_{\lfloor \frac{D-1}{2} \rfloor})$. For the vertex sets $NU_0, NU_1, ..., NU_s$, we have the following observations.

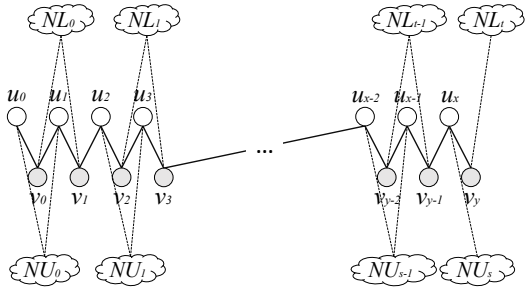


Figure 3: Proof of Lemma 3.2 - Case 2

• $NU_0 \geq \alpha - 1$.
• $NU_s \geq \alpha - 2$, for $0 < s \leq \lfloor \frac{D}{4} \rfloor$.

Same as Case 1, the number of vertices in NU_0, NU_1, ..., NU_s, along with the lower layer vertices in the path p, serves as the minimum value for the number of vertices in $L(g)$. Thus, we can have $|L(g)| \geq \lfloor \frac{D-1}{2} \rfloor + 1 + \sum_{s=0}^{\lfloor \frac{D}{4} \rfloor} |NU_s| \geq \lfloor \frac{D}{4} \rfloor(\alpha - 2) + \lfloor \frac{D-1}{2} \rfloor + \alpha$.

For the vertex sets $NL_0, NL_1, ..., NL_t$, we observe that:

• $NL_t \geq \beta - 2$, for $0 \leq t < \lfloor \frac{D-1}{4} \rfloor$.
• $NL_{\lfloor \frac{D-1}{4} \rfloor} \geq \beta - 1$.

In line with Case 1, the number of upper layer vertices of g in Case 2 is at least $\lfloor \frac{D}{2} \rfloor + 1 + \sum_{t=0}^{\lfloor \frac{D-1}{4} \rfloor} |NL_t| \geq \lfloor \frac{D}{4} \rfloor(\beta - 2) + \lfloor \frac{D}{2} \rfloor + \beta$.

Following the same logic, the same inequalities hold true in Case 3 and Case 4 as well. Thus, lemma 3.2 holds. □

Following the above lemma, we can know that if a vertex v with $dist_G(v, q)$ satisfies $\lfloor dist_G(v, q) - 1/4 \rfloor(\beta - 2) + \lfloor dist_G(v, q)/2 \rfloor + \beta - 1 > \mathcal{S}_U$ and $\lfloor dist_G(v, q) - 1/4 \rfloor(\alpha - 2) + \lfloor dist_G(v, q)/2 \rfloor + \alpha - 1 > \mathcal{S}_L$, then it can be deemed unpromising since it cannot be included in any SABC. Thus, to reduce the search space of Shrink, we can compute the minimum D value to remove such unpromising vertices while preserving all the valid SABC. To compute the minimum D value, when $\alpha > 1$ and $\beta > 1$, we can incrementally increase D from 2, and check whether the size computed using Lemma 3.2 exceeds the given size constraints. Once $\lfloor \frac{D-1}{4} \rfloor(\beta - 2) + \lfloor \frac{D}{2} \rfloor + \beta - 1$ or $\lfloor \frac{D-1}{4} \rfloor(\alpha - 2) + \lfloor \frac{D}{2} \rfloor + \alpha - 1$ exceeds the given size constraints, the D value is the minimum value that can let $G_{D,q}$ cover all the possible SABC. Here $G_{D,q}$ denotes the subgraph containing q and each vertex v in $G_{D,q}$ satisfies $dist_G(v, q) \leq D$. Note that, when $\alpha = 1$ and $\beta = 1$, the minimum possible value of D is 1. In this case, we only keep vertex v having $dist_g(v, q) = 1$. When $q \in U(G), \beta = 1$ or $q \in L(G), \alpha = 1$, we only keep vertex v having $dist_g(v, q) = 1$. And when $q \in U(G), \alpha = 1$ or $q \in L(G), \beta = 1$, we keep vertex v having $dist_g(v, q) = 2$.

Example 3.3. Let's consider the bipartite graph G shown in Figure 1. Given $q = u_1$, $\alpha = 2, \beta = 3$, $\mathcal{S}_U = 4$, and $\mathcal{S}_L = 3$, we apply the distance-based reduction technique. To begin, we first obtain the minimum D value using Lemma 3.2. First, we set D as 2, we can get $|U(G)| \geq 3, |L(G)| \geq 2$. Since the size of G does not satisfy the size constraints, we increase D to 3. At this point, we still have $|U(G)| \geq 3$ and $|L(G)| \geq 2$. Continuing, we increase D to 4, then we have $|U(G)| \geq 4, |L(G)| \geq 3$, which can cover the given size constraints. Thus, we can safely remove the vertices having distances larger than 4. To compute the distance of each vertex to q, we employ the BFS algorithm. Since $dist_G(v_6, u_1) = dist_G(v_7, u_1) = 5$ and $dist_G(u_7, u_1) = 6$, we can remove v_6, v_7, and u_7 from G.

Improvement 2: Degree-based vertex ordering. In the shrink-based approach, it removes one vertex from G' in each iteration. If a vertex v is required to be included in the resulting community and be removed during the process, it must be added back after the algorithm completes all subsequent recursions. Thus, the time consumption in the subsequent recursions can be reduced if we postpone the removal of v and remove some unpromising vertices first. To achieve this, we propose a function to measure the "goodness" of each unvisited vertex and select the most promising one to explore as follows.

Definition 3.4. **Degree-based vertex scoring function.** Given a graph G' and a vertex set C, the score function for a vertex $u \in V(G') \setminus C$ is defined as

$$f(u) = deg_{G'}(u)$$

Intuitively, to meet the cohesiveness constraints, the vertices with higher degrees are more likely to be retained since they have more connections with other vertices. Thus, the degree-based vertex scoring function mainly considers the vertices degree in the remaining subgraph G'. By applying the degree-based vertex scoring function, in each iteration, we can select the vertex having a minimum degree in $G' \setminus C$, and the most unpromising vertex can be removed first.

Algorithm 3: `Shrink-SP`

Input: G, q, α, β, $\mathcal{S}_U$, and $\mathcal{S}_L$;
Output: H;
1 $C \leftarrow \{q\}$;
2 Compute distance from q to each vertices in $G \setminus q$;
3 $D \leftarrow$ the minimum D value computed using Lemma 3.2;
4 $G' \leftarrow$ the connected subgraph in $G_{D,q}$ satisfying the cohesiveness constraint and $q \in G'$; ▷ `/* Distance-based reduction */`
5 Run Algorithm 1 Lines 3 - 7;

6 **Function** `SSearch`(G', C)
7 **if** *a valid solution H is found* **then**
8 **return** H;
9 **foreach** *v has the lowest score in $V(G') \setminus C$* **do**
10 ▷ `/* Degree-based vertex ordering */`
11 Run Algorithm 1 Lines 12 - 20;

The `Shrink-SP` algorithm. Based on the techniques discussed above, we propose the `Shrink-SP` algorithm for solving the `SABC` problem. The pseudocode is shown in Algorithm 3, which is similar to Algorithm 1 but incorporates the distance-based reduction (Line 2) and the vertex scoring function (Line 9). Before initializing G', it first applies Lemma 3.2 to obtain a minimum value of D (Line 3). Then it initializes G' as the connected subgraph in $G_{D,q}$ satisfying the cohesiveness constraint and containing q (Line 4). Then in the `SSearch` function, we select the vertex with the lowest score in $V(G') \setminus C$ as v, so that the vertex with the smallest degree can be removed from G' first (Line 9).

4 EXPAND-BASED APPROACHES

The shrink-based approaches proposed in the above section can find the exact result correctly. Nevertheless, when processing queries with relatively small size constraints, they may need to execute the `Peeling` algorithm many times to meet the size constraints. It may lead to more back-track operations until a `SABC` is found. Additionally, the `Peeling` algorithm takes $O(|E(G')|)$ times, which can be time-consuming to execute in each recursion. Motivated by this, we explore expand-based approaches in this section.

4.1 A basic expand-based approach

The expand-based approach aims to explore the local subgraph around the query vertex q. It enumerates all possible combinations of vertices that the subgraph induced by them can satisfy the size constraints and evaluates whether the subgraph is connected and satisfies the cohesiveness constraint. Let C denote the candidate vertex set and G' denote the remaining graph. For each unvisited vertex in $V(G') \setminus C$, there are two choices: (1) the vertex belongs to the resulting community, in which case we add it to C; (2) otherwise, we remove it from G'. Considering these two situations, we create two instances for each vertex v: $(C \cup \{v\}, G')$ and $(C, G' \setminus v)$. The algorithm continues the enumeration process and returns when encounters a subgraph induced by C that can meet the constraints of `SABC`.

Algorithm 4: `Expand`

Input: G, q, α, β, $\mathcal{S}_U$, and $\mathcal{S}_L$;
Output: H;
1 $C \leftarrow \{q\}$;
2 $G' \leftarrow$ the connected component of (α, β)-core containing q in G;
3 $H \leftarrow \emptyset$;
4 **if** *G' is not empty* **then**
5 $H \leftarrow$ `ESearch`(C, G');
6 **return** H;

7 **Function** `ESearch`(C, G')
8 **if** $|U(C)| > \mathcal{S}_U$ *or* $|L(C)| > \mathcal{S}_L$ **then**
9 **return** $\emptyset$;
10 $H' \leftarrow$ the subgraph induced by C;
11 **if** *H' is a* `SABC` **then**
12 **return** H';
13 $v \leftarrow$ an vertex in $V(G') \setminus C$;
14 `ESearch`$(C \cup \{v\}, G')$;
15 `ESearch`$(C, G' \setminus v)$;

The pseudo-code of the basic expand-based approach is given in Algorithm 4. It first initializes a set C as a candidate set (Line 1). Then G' is initialized as the connected component of (α, β)-core that contains the query vertex q (Line 2). H is initialized as an empty set (Line 3). If G' is not empty (Line 4), the algorithm invokes the `ESearch` function to recursively maintain the candidate set C and a subgraph G' (Line 5). In the `ESearch` function, if C is still under the size constraints (Lines 8 - 9), it obtains the subgraph induced by C as H' (Line 10). If H' can meet the constraints of `SABC` then it returns (Lines 11 - 12). Here we check whether H is a `SABC` by first checking the degree constraints for each vertex. If all the vertex in H can meet the degree constraint we then further check the connectivity. If H is still not a valid result, `ESearch` continues to visit other combinations and selects an unvisited vertex v, i.e. $v \in V(G') \setminus C$ (Line 13). For the selected vertex v, `ESearch` first considers the possibility of including v in the resulting community and visits the branch where v is added to C (Line 14). If a valid solution is not found when v is included in C, the algorithm proceeds to explore the branch where v is removed from G' (Line 15).

Example 4.1. Consider the bipartite graph G shown in Figure 1. Suppose the query is $q = u_1$, $\alpha = 2$, $\beta = 3$, $\mathcal{S}_U = 4$, and $\mathcal{S}_L = 3$. In

the algorithm, we select vertices at Line 15 following the order of their IDs and alternating between the two layers. Figure 4 shows part of the search process of the ESearch function. The candidate set C in each reduction is shown as the tree node. The dashed part is not visited since the algorithm visits the left subtree first, and a result is already found and returned by the algorithm. Initially, $C = u_1$ and $G' = G$. For the first branch, we choose v_1, and we visit the left as $C = u_1, v_1$. Then we process forward by selecting the next unvisited vertices in $V(G') \setminus C$. The search continues in a similar way until a result is obtained.

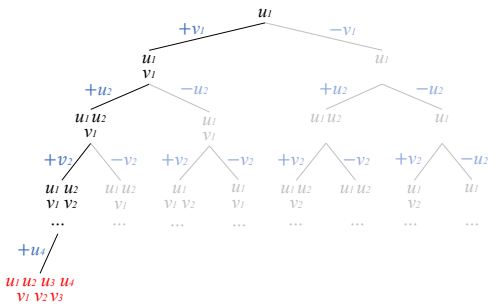


Figure 4: An example of Expand algorithm

Correctness Analysis. The Expand algorithm determines the result by exploring all possible combinations of candidates that meet the specified size constraints. For each candidate set, it verifies whether a subgraph induced by the candidate can also satisfy the prescribed cohesiveness constraints. When a subgraph induced by the candidate set fulfills the cohesiveness constraints, the algorithm returns the result. Therefore, as long as a valid result exists, the Expand algorithm is capable of identifying it through its exhaustive search approach.

Complexity Analysis. The time complexity of Expand is $O(n^{\mathcal{S}_U+\mathcal{S}_L})$ since the depth of recursion is bounded by $O(\mathcal{S}_U + \mathcal{S}_L)$. Here n is the number of vertices in G and $\mathcal{S}_U, \mathcal{S}_L$ are the size constraints.

4.2 Improving the expand-based approach

In this part, we propose techniques to enhance the performance of the basic expand-based algorithm presented in Section 4.1. The expand-based approach can also be improved by limiting the search space and selecting appropriate vertices to visit in each iteration. Thus, the improvements proposed in Section 3.2 are also applicable to the expand-based approach as well. Furthermore, there exist additional opportunities for improvement to examine the existence of an SABC in the candidate graph G'.

- **Prune the unpromising vertices.** In our expand-based approach, the goal is to quickly extend the candidate set. Peeling the candidate subgraph G' to a connected subgraph that satisfies the cohesiveness constraint and contains q can be time-consuming. Therefore, we need to find a more efficient method to prune the unpromising vertices.
- **Reduce the search branches.** In each recursion, our expand-based approach selects one vertex in $V(G') \setminus C$ and generates two branches. This can result in a significant number of recursions. Effective techniques to reduce the number of generated branches will help mitigate this issue.
- **Optimized vertex selection order.** In the expand-based approach, the candidate sets do not satisfy the given cohesiveness constraints until a result is obtained. Therefore, it becomes crucial to develop an appropriate vertex order that not only considers the cohesiveness constraints but also takes into account the contribution of each vertex in forming a subgraph that satisfies the cohesiveness constraint.

Improvement 1: Degree-based reduction. We propose the degree-based reduction to prune the unpromising vertices during the search process. In the expand-based approach, after removing an invalid vertices v from the candidate subgraph G', G' may no longer satisfy the cohesiveness constraint. To optimize the prune effectiveness, an intuitive way is to execute the Peeling process on G' and keep G' as a connected subgraph that satisfies the cohesiveness constraint and contains q. However, peeling G' requires $O(|E(G')|)$ times. While it effectively reduces the graph size for the subsequent iteration, it does not improve the efficiency of the whole algorithm. Thus, to compromise the effectiveness and efficiency, here we only remove the vertices that do not have enough degree, i.e. $u \in U(G') \setminus C$ with $deg_{G'}(u) < \alpha$, or $v \in L(G') \setminus C$ with $deg_{G'}(v) < \beta$.

The above observation can be proved as follows. As the degree of a vertex exhibits a monotonically increasing property in the expand-based approach, it is guaranteed that $deg_{H'}(v) \leq deg_{G'}(v)$ for all feasible communities H'. According to the definition of SABC, it is imperative that any vertex $u \in U(H') \setminus C$ satisfies $deg_{H'}(u) \geq \alpha$, and any vertex $v \in L(H') \setminus C$ satisfies $deg_{H'}(v) \geq \beta$. Consequently, all upper layer vertices in G' must possess degree at least α, while lower layer vertices in G' must have degree at least β.

Improvement 2: Size and structure-based reductions. Give a subgraph G' and a subgraph H' induced by candidate set C. Let α' denote the minimum degree value among all the degree values of the upper layer vertex in H', i.e., $\alpha' = min\{deg_{H'}(u)|u \in U(H')\}$, and β' denote the minimum degree value among all the degree values of the lower layer vertex in H', i.e., $\beta' = min\{deg_{H'}(v)|v \in L(H')\}$. We can safely prune the brunch if $\beta - \beta' > \mathcal{S}_U - |U(H')|$ or $\alpha - \alpha' > \mathcal{S}_L - |L(H')|$. If H' satisfies $\mathcal{S}_U - |U(H')| \geq \beta - \beta'$, we can further prune the brunches if it does not have $\beta - \beta'$ upper vertices having degree larger than $\alpha - \alpha'$. Also, If H' satisfies $\mathcal{S}_L - |L(H')| \geq \alpha - \alpha'$, we can further prune the brunches if it does not have $\alpha - \alpha'$ upper vertices having degree larger than $\beta - \beta'$.

According to Definition 2.2, a feasible community H' must adhere to both size and degree constraints. Consequently, the minimum degree value for the upper-layer vertices, denoted as α', should reach at least α, and the minimum degree value for the lower-layer vertices, denoted as β', should reach at least β. Considering that there can be at most $\mathcal{S}_U - |U(H')|$ upper-layer vertices and $\mathcal{S}_L - |L(H')|$ lower-layer vertices that can still be added to H', the increments of α' and β' are, at most, $\mathcal{S}_L - |L(H')|$ and $\mathcal{S}_U - |U(H')|$, respectively. Therefore, in order to satisfy the specified constraints, H' must fulfill the conditions $\beta - \beta' \leq \mathcal{S}_U - |U(H')|$ and $\alpha - \alpha' \leq \mathcal{S}_L - |L(H')|$.

Improvement 3: Neighbor-based vertex ordering. Similar to the shrink-based algorithm, to form a subgraph that satisfies the cohesiveness constraint, vertices with high degrees should be given priority to add into the candidate set since they have more connections with other vertices and are more likely to be included in a subgraph that satisfies the cohesiveness constraint. Thus, the degree-based vertex scoring function proposed in Section 3.2 also applies to the expand-based approach. The difference is the expand-based approach needs to explore the most promising vertex first, and the vertex with the highest score is selected in each recursion.

For the expand-based approach, given a candidate set C, the selected vertex should be closely connected to the vertices in C, so that these vertices can form a feasible community quickly. The intuition that vertices with a large degree are more likely to be included in a subgraph that satisfies the cohesiveness constraint is generally correct. However, it is not always the case that keeping only high-degree vertices will result in a SABC. For instance, if a large degree vertex v is all connected to some "peripheral" vertices, then it also can not help with forming a resulting community. By considering this, we propose the neighbor-based vertex scoring function as follows.

Definition 4.2. **Neighbor-based vertex scoring function.** Give a subgraph G' and a candidate set C, the score function for a vertex $u \in U(G') \setminus C$ is defined as

$$f(u) = min\{deg_{G'}(u), \mathcal{S}_L\} + \{v' | v' \in N_{G'}(u) \wedge deg_{G'}(v') \geq \beta\}$$

the score function for a vertex $v \in L(G') \setminus C$ is defined as

$$f(v) = min\{deg_{G'}(v), \mathcal{S}_U\} + \{u' | u' \in N_{G'}(v) \wedge deg_{G'}(u') \geq \alpha\}$$

We focus on two criteria in the neighbor-based vertex scoring function. The first criterion measures the number of edges that the selected vertex itself can contribute to the resulting community. We consider this criterion because if the degree of the selected vertex exceeds the size constraint of the other layer, the number of neighbors that the selected vertex can introduce to the resulting community is still limited by the size constraint. The second criterion assesses the number of additional vertices that the selected vertex can add to set C to help form a subgraph satisfying the cohesiveness constraint. Specifically, it counts the number of neighbors of the selected vertex that can meet the degree constraints. By employing the neighbor-based vertex scoring function to determine the vertex with the highest score, we can identify the vertex that is most likely to constitute a subgraph that satisfies the cohesiveness constraint.

The Expand-SP algorithm. Based on the improvements introduced in Section 4.2, we propose the Expand-SP algorithm as shown in Algorithm 5. The difference from Algorithm 4 are as follows. Firstly, when initializing G', we apply the distance-based reduction to narrow down the search space (Lines 2 - 3). Secondly, in the ESearch function, we further prune unpromising branches using the size and structure-based reduction. So that it returns to the previous recursion if the current instance and it following brunches are not able to generate a result (Lines 7 - 8). Then we apply Degree-based reduction, where we remove all the vertices that do not meet the degree constraints and record them in the set RV (Lines 9 - 13). To determine the next vertex to explore, we select a vertex with the highest score in $V(G') \setminus C$ using the neighbor-based vertex scoring function (Line 14). After returning from the following recursion, we add the vertex stored in RV back to G' (Line 17).

Algorithm 5: Expand-SP

```
Input: G, q, α, β, S_U and S_L;
Output: H;
1  C ← q;
2  D ← the diameter upper bound computed using Lemma 3.2;
3  G' ← the connected subgraph in G_{D,q} satisfying the
     cohesiveness constraint and containing q;        ▷ /*
     Distance-based reduction */
4  Run Algorithm 4 Lines 3 - 5;

5  Function ESearch(C, G')
6      Run Algorithm 4 Lines 8 - 12;
7      if not satisfy the size and structure-based reduction then
8          return; ▷ /* Size and structure-based reduction
             */
9      RV ← ∅;
10     foreach v' ∈ V(G') \ C do
11         if v' does not have enough degree then
12             Remove v' from G';
13             RV.push(v'); ▷ /* Degree-based reduction */
14     v ← the vertex in V(G') \ C with the highest score;   ▷ /*
         Neighbor-based vertex ordering */
15     ESearch(C ∪ {v}, G');
16     ESearch(C, G' \ v);
17     Add vertices in RV back to G';          ▷ /* Degree-based
         reduction */
```

5 EXPERIMENTS

In this section, we conduct experiments to evaluate the effectiveness of the SABC model and the efficiency of our proposed algorithms.

Algorithms. To the best of our knowledge, no existing work investigates the SABC search problem and corresponding algorithms. We mainly evaluate the algorithms of both the shirk-based approach and the expand-based approach with all the improvements. We denote our proposed improvements as follows.

- P1: Distance-based reduction in Section 3.2.
- P2: Degree-based reduction in Section 4.2.
- P3: Size and structure-based reduction in Section 4.2.
- S1: Degree-based vertex scoring function in Definition 3.4.
- S2: Neighbor-based vertex scoring function in Definition 4.2.

Our experiments are conducted against the following algorithms.

- Shrink: Basic shrink-based algorithm in Algorithm 1.
- Expand: Basic expand-based algorithm in Algorithm 4.
- Shrink-P: Shrink + P1.
- Expand-P: Expand + P1 + P2+ P3.
- Shrink-S: Shrink + S1.
- Expand-S: Expand + S2.
- Shrink-SP: Improved shrink-based algorithm in Algorithm 3, i.e., Shrink + P1 + S1.
- Expand-SP: Improved expand-based algorithm in Algorithm 5,i.e., Expand +P1 + P2 + P3 + S2.

All algorithms are implemented in C++ and the experiments are run on a Linux server with an Intel Xeon E3-1231 processor (3.40GHz, 4 Cores) and 16GB main memory.

Table 2: Summary of Datasets

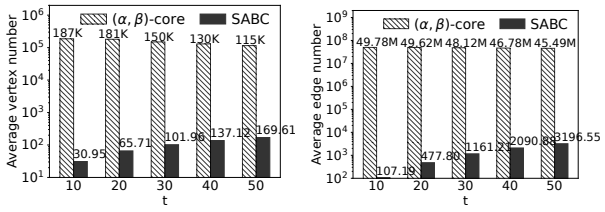
Dataset	$\|U\|$	$\|L\|$	$\|E\|$	deg_{avg}	density
YouTube	94.24 K	30.09 K	293.36 K	4.72	5.51
Github	56.52 K	120.87 K	440.24 K	4.96	5.33
Teams	901.13 K	34.46 K	1.37 M	2.92	7.75
Citeseer	337.12 K	196.13 K	1.75 M	6.57	6.81
Livemocha	61.45 K	104.10 K	2.19 M	21.07	27.42
MovieLens	162.54 K	59.05 K	25.00 M	225.64	255.18
Wikipedia	1.85 M	1.67 M	39.95 M	22.68	22.71
BagPubmed	1.00 M	624.96 K	256.80 M	115.92	449.54
YahooSong	8.2M	14.1 K	483.45 M	315.88	324.68

Datasets. In our experiments, we use 9 real-world datasets, which can be found in KONECT (http://konect.cc/). Table 2 shows the statistics of datasets. $|U|$ and $|L|$ denote the number of vertices in the upper and lower layers, and $|E|$ represents the number of edges. deg_{avg} denotes the average degree, which is computed as $\frac{2\times|E(G)|}{|U(G)|+|L(G)|}$. Density is computed as $\frac{|E(G)|}{\sqrt{|U(G)||L(G)|}}$ [36].

Parameters. For each experiment, we randomly generate 19 queries. The query vertex is randomly selected from the (α, β)-core to ensure that there is a meaningful community containing the query vertex. The default values of the query parameters are $\alpha = 4$, $\beta = 4$, $\mathcal{S}_U = 9$, and $\mathcal{S}_L = 9$. The average result quality and processing time of the 19 queries are reported. For each test, we set the time limit as 1000 seconds, and we record its running time as 1000 seconds if it does not finish within the time limit.

5.1 Effectiveness Evaluation

In this subsection, we validate the effectiveness of our proposed model. Here we compare our model with the (α, β)-core model on MovieLens which contains 25M ratings from 162K users (U) on 59K movies (M).



(a) Average vertex number (b) Average edge number

Figure 5: Effect of the community size, setting $\alpha, \beta = t$

Evaluation of the community quality. We conduct a comparison between the (α, β)-core model and our SABC model in terms of size. The degree constraints for both models were set in the range of 10 to 50, with increments of 10. Moreover, the size constraint was defined as 1.2 times the value of the degree constraint. The statistics of 20 query results are presented in Figure 5, which demonstrates our SABC model can effectively limit the size of the resulting community. As we can see, since the (α, β)-core model only adopts the degree constraints, the size of an (α, β)-community can become exceptionally large, making it unpredictable for some applications that have capacity constraints. By incorporating size constraints, our SABC model can identify communities with fewer vertices and edges compared to the (α, β)-core model. This makes the SABC model more apt for tackling real-world challenges that come with capacity limitations.

Case study. We conduct queries using the movie "Richard III" (ID: 41) with degree constraints $\alpha = 6$, $\beta = 6$ and size constraints $\mathcal{S}_U = 12$, $\mathcal{S}_L = 12$. The community generated by the Shrink-SP algorithm is denoted as $SABC_s$, while the one generated by the Expand-SP algorithm is denoted as $SABC_e$. Additionally, community derived from the Shrink-SP algorithm utilizing the vertex scoring function as maximum average movie rating is denoted as $SABC_m$. Table 3 presents the statistics of all query results. In the table, $|U|$ and $|M|$ represent the total number of users and movies in the community, respectively, while $|E|$ denotes the total number of ratings in the community. As shown in Figure 6, the (α, β)-core community encompasses all query results. The average movie rating of each movie is marked in blue. It can be observed that Shrink-SP and Expand-SP can obtain different communities. The communities obtained by Shrink-SP are more likely to have a large size than the communities obtained by Expand-SP. That is because the shrink-based algorithm keeps peeling the subgraph until the size shrinks to the size constraints, while the expand-based algorithm expands the candidate subgraph and terminates when it meets the degree constraints. This inherent flexibility allows users to select an algorithm that aligns with their specific requirements. Moreover, by incorporating additional evaluation criteria, the SABC model is capable of returning a more refined selection compared to the (α, β)-core model. For example, as illustrated in Figure 6, when we apply the maximum average movie rating as the vertex selection function, all the movies in the $SABC_m$ community, except for the query vertex, have an average rating greater than 4.25. In contrast, for $SABC_s$, the average rating is 3.35, and for $SABC_e$, it is 3.41.

Table 3: Statistics of query results, $q = 41$

Models	$\|U\|$	$\|M\|$	$\|E\|$
$SABC_e$	8	9	68
$SABC_s$	12	12	144
$SABC_m$	12	12	126
(α, β)-core	162,541	30,207	24,933,300

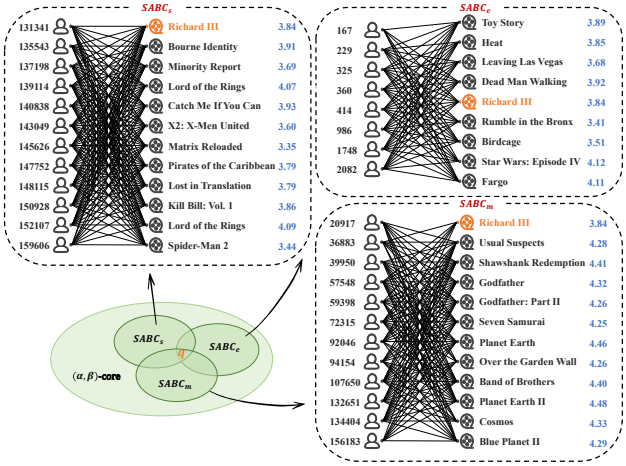


Figure 6: Case study

5.2 Performance Evaluation

Here we evaluate the processing time and success ratio of our proposed algorithms. We also investigate the query time with different α, β, $\mathcal{S}_U$, and $\mathcal{S}_L$ values. Furthermore, we validate the reduction rules and vertex scoring functions. Due to the NP-hardness of the SABC problem, there is a slight portion of hard cases with a huge search space that cannot be computed within the time limit. Thus we also report the success ratio, which is the number of queries successfully returned before timeout over the total number of queries.

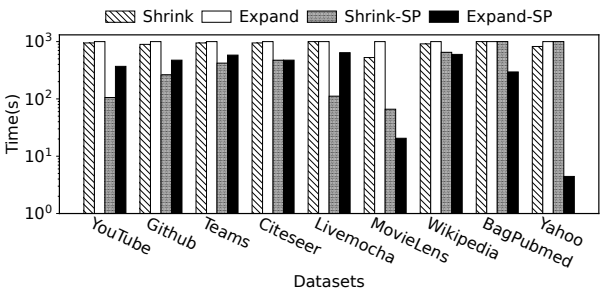


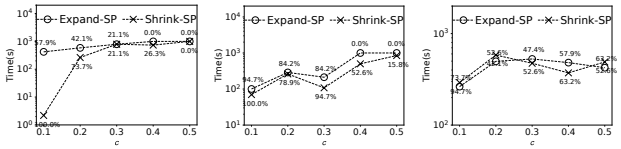
Figure 7: Evaluation of SABC query time

Exp1: Processing time and success ratio on all datasets. In this experiment, we report the average processing time and success ratio of query algorithms (Shrink, Shrink-SP, Expand,

Table 4: Success ratio of queries

DataSet	Shrink	Expand	Shrink-SP	Expand-SP
YouTube	5.26%	0.00%	89.47%	63.16%
Github	10.53%	0.00%	73.68%	52.63%
Teams	5.26%	0.00%	57.89%	42.11%
Citeseer	5.26%	0.00%	52.63%	52.63%
Livemocha	0.00%	0.00%	89.47%	36.84%
MovieLens	47.37%	0.00%	100.00%	100.00%
Wikipedia	10.53%	0.00%	47.37%	42.11%
BagPubmed	0.00%	0.00%	0.00%	78.95%
YahooSong	21.05%	0.00%	0.00%	100.00%

Expand-SP) to retrieve SABC with degree constraints $\alpha = 4, \beta = 4$, and size constraints $\mathcal{S}_U = 9$, $\mathcal{S}_L = 9$. The query time of each dataset is shown in Figure 7 and the success ratios are reported in Table 4. In general, with the help of our proposed reduction rules and vertex scoring function, Expand-SP outperforms Expand and Shrink-SP outperforms Shrink in both processing time and success ratio. Additionally, it is observed that Shrink usually performs better on smaller datasets, whereas Expand performs better on larger datasets. This trend can be attributed to two primary factors. Firstly, Shrink-SP requires more time to gradually peel the entire graph and is more likely to encounter unpromising branches on larger datasets. Secondly, larger datasets tend to contain a higher number of vertices with large degrees. By leveraging our proposed vertex scoring function, Expand-SP can prioritize visiting these high-degree vertices, facilitating the formation of an (α, β)-core more efficiently. In certain special cases, we can observe that Shrink and Expand outperform Shrink-SP and Expand-SP. The reason for this is that both Shrink-SP and Expand-SP utilize the vertex ordering improvements, which prioritizes the removal of unpromising vertices, as they are less likely to appear in the result. Although these improvements generally expedite result formation, there exist cases where low-degree vertices do end up in the result. Removing these vertices can cause the search to venture into unpromising branches, potentially causing the algorithm to exceed the time limit.

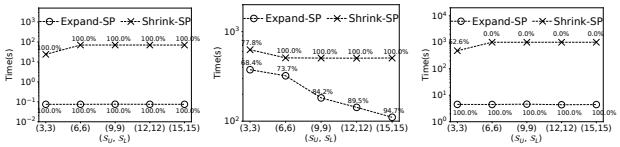


(a) GitHub (b) MovieLens (c) Citeseer

Figure 8: Effect of α and β, setting $\alpha, \beta = c \cdot \delta$

Exp2: Evaluation of varying α, β. In this experiment, we report the average processing time and success ratio of query algorithms Shrink-SP and Expand-SP with both α and β from $0.1 \cdot \delta$ to $0.5 \cdot \delta$ and $\mathcal{S}_U = \delta$, $\mathcal{S}_L = \delta$. Figure 8 illustrates the results of our experiment. The graph shows that, in general, increasing the degree constraints leads to longer query times for both the Shrink-SP and Expand-SP algorithms. Nevertheless, the query time for each individual query depends on the graph's structure and the neighbors of the query vertices. It should be acknowledged that our selection of query vertices only represents a small portion of the entire graph, and therefore, experimental results may slightly deviate from the overall trend in certain cases.

Exp3: Evaluation of varying $\mathcal{S}_U, \mathcal{S}_L$. In this experiment, we report the average processing time and success ratio of query algorithms Shrink-SP and Expand-SP with both $\mathcal{S}_U$ and $\mathcal{S}_L$



(a) MovieLens (b) Livemocha (c) BagPubmed

Figure 9: Effect of $\mathcal{S}_U$ and $\mathcal{S}_L$

from 3 to 15 and $\alpha = 2$, $\beta = 2$. We chose this set of parameters based on practical considerations, as in real-world applications, size-bounded community search tends to favor smaller sizes. The results are shown in Figure 9. As a general trend, both the Shrink-SP and Expand-SP Algorithms tend to have shorter query times as the size constraints increase. This is because larger size constraints make it easier for the algorithms to find results that satisfy the degree constraints. This pattern is more pronounced in smaller graphs, such as Livemocha in Figure 9(b). In Figure 9(c), we observe a decline in the success rate with increasing size constraints. This can be attributed to the distance-based reduction improvement. As the size constraints grow, the search space also expands accordingly, possibly leading the search into less promising branches and increasing the risk of exceeding the time limit.

Table 5: Effect of different optimization techniques

DataSet		Expand	+S2	+S2+P1	+S2+P1+P2	+S2+P1+P2+P3
YouTube	time	1000	789.48	672.80	646.88	474.74
	rate	0%	21.05%	36.84%	36.84%	52.63%
	num	13.37 M	8.76 M	5.80 M	3.04 M	3.00 M
Citeseer	time	1000	842.11	634.10	580.59	528.02
	rate	0%	15.79%	36.84%	42.11%	47.37%
	num	3.14 M	1.93 M	1.60 M	681.76 K	606.29 K
Teams	time	1000	947.37	768.17	756.64	581.34
	rate	0%	5.26%	26.32%	26.32%	42.11%
	num	1.76 M	1.17 M	1.09 M	505.37 K	480.71 K

Exp4: Evaluation of different optimization techniques. In this experiment, we aim to assess the efficiency of various reduction rules and vertex scoring functions in improving the performance of the Expand algorithm. We have chosen the Expand algorithm for this experiment because it allows us to implement all the optimization techniques we have proposed. To simplify the discussion, we refer to the three reduction rules being evaluated as P1, P2, and P3. As for the vertex selection techniques, we focus on S2 in this experiment, as it has been specifically designed for the Expand algorithm. Table 5 presents the results in terms of running time, success rate, and the number of generated branches. The data in the table clearly demonstrates that the implementation of our optimization techniques leads to a significant reduction in the number of iterations and running time, while simultaneously increasing the success rate.

Exp5: Evaluation of different vertex selection strategies. In this experiment, we validate the vertex scoring functions S1 and S2 on Shrink-P and Expand-P, respectively. The results of different scoring functions applied on Expand-P and Shrink-P are shown in Figure 10. The results show that, across most of the datasets, S2 outperforms S1 in Expand-P. Regarding Shrink-P, S1 performs better than S2. This is because S1 considers the vertex's degree while S2 examines the vertex's contribution in forming a subgraph that satisfies cohesiveness constraints, and they are suitable for different approaches.

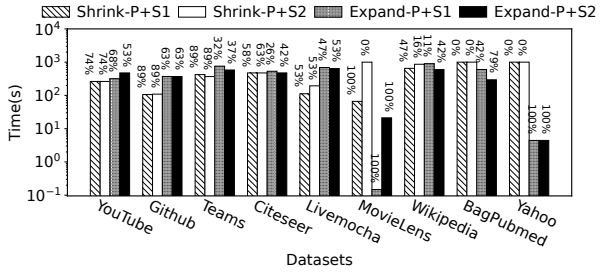
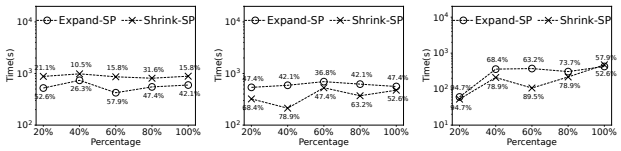


Figure 10: Effect of Different Vertex Selection Strategies

Exp6: Evaluation of scalability. In this experiment, we study the scalability of Expand-SP and Shrink-SP algorithms by altering the graph size on datasets Wikipedia, Teams, and Citeseer. When varying graph size, we randomly sample 20% to 100% edges of the original graphs. Figure 11 demonstrates that both Expand-SP and Shrink-SP algorithms are scalable. The computational cost trends on the Citeseer dataset generally rise with an increase in the percentage of edges. This can be attributed to the fact that increasing the graph size will increase the number of vertex that need to be considered and also increase the peeling time of the (α, β)-core. However, the computational costs remain relatively stable for the Wikipedia and Teams datasets. The reason is that, in certain instances, enlarging the graph size can increase the number of neighborhoods of some vertices, making them more likely candidates for inclusion in the final result. As a result, the algorithm can more efficiently construct the desired outcome.



(a) Wikipedia (b) Teams (c) Citeseer

Figure 11: Effect of graph size

6 RELATED WORK

To the best of our knowledge, this paper is the first to study size-bounded community search over bipartite graphs. Below we review three closely related areas, size-bounded community search on unipartite graphs, cohesive subgraph models on bipartite graphs, and community search on bipartite graphs.

Size-bounded community search on unipartite graphs. On unipartite graphs, size-bounded community search is conducted based on different cohesiveness models such as k-core and k-truss. Based on the k-core model and taking the size constraint into consideration, Yao et al. [41] study the size-bounded k-core community search problem, with a query vertex q and size constraint $[l, h]$ and aims to find a community with an optional degree. Li et al.[14] present the minimum k-core model, which aims to find the smallest k-core subgraph containing the query set. For the truss-based model, a size-bounded community search model is first proposed in [20], incorporating both shrink-based and expand-based algorithms. In addition, the authors of [43] introduce a size-constrained truss community model that aims to maximize the cohesiveness of the resulting community and provide an efficient solution for handling the hard case. These models cannot be applied to solve our problem due to the intrinsic differences between unipartite graphs from bipartite graphs.

Cohesive model on bipartite graphs. On bipartite graphs, several existing studies explore the (α, β)-core model [7, 10, 18, 21, 38]. Ding et al. [7] propose an efficient algorithm with linear time complexity for querying the (α, β)-core in an online manner. Liu et al. [18] design a BiCore-Index to store and retrieve the (α, β)-core. The BiCore-Index maintenance technique is presented in [19] to handle dynamic bipartite graphs. Additionally, the authors of [28, 33, 39, 47] focus on the k-bitruss model in bipartite graphs, which is the maximal subgraph where each edge is contained in at least k butterflies [40]. In these works, zou et al. [47], is the first to propose the k-bitruss model. By adding a butterfly connectivity constraint, Sariyuce et al. [28] further introduce the k-tip/k-wing model and provide efficient peeling algorithms. Here, k-tip is the maximal butterfly-connected subgraph where each vertex is contained in at least k butterflies, and k-wing is the maximal butterfly-connected subgraph where each edge is contained in at least k butterflies. The parallel tip decomposition algorithm is also recently studied in [11]. Wang et al. [33] propose a novel online index and a new bitruss decomposition algorithm. Biclique is a complete subgraph of a given bipartite graph [22]. Based on the biclique model, the maximal biclique enumeration problem was studied in many recent works [5, 25, 32, 44, 46], which aims to find all the maximal bicliques. In addition, various variants of the maximal biclique model have been investigated recently, including maximum vertex biclique [12], maximum edge biclique [27], and maximum balanced biclique [4].

Community search on bipartite graphs. Community search aims to search a subgraph that consists of internally connected vertices based on a query vertex and specific query parameters [8, 9]. While various community search algorithms have been proposed for bipartite graphs, these works often overlook the consideration of size constraints. Wang et al. [36] study the (α, β)-community search problem and provide both online and index-based approaches. In [37], the authors propose an index maintenance technique to support searching (α, β)-core communities on dynamic bipartite graphs, allowing efficient updating of the index after the insertion and deletion of an edge. The authors of [2] study the community search problem following the k-wing model and further provide solutions for the k-wing community search on the dynamic bipartite graph [1]. The authors of [34] investigate the personalized maximum biclique search problem and proposed the PMBC-Index enabling both efficient construction and query processing.

7 CONCLUSION

In this paper, we study the SABC search problem that aims to find an (α, β)-core-based community containing the query vertex with the number of the upper and lower layer verities in the subgraph not exceeding a given size threshold. We prove the problem is NP-hard. Practically efficient exact solutions are developed that employ novel pruning rules and effective search strategies. Extensive experiments verify the effectiveness of SABC and the superiority of the techniques. In the future, we will explore approximate solutions to address the extreme cases caused by the hardness of the problem and explore the possibility of applying Graph Nural Network to solve the extreme cases.

8 ACKNOWLEDGMENT

Kai Wang is supported by NSFC U2241211 and NSFC 62302294.
Wenjie Zhang is supported by ARC FT210100303 and ARC DP230101445.
Xuemin Lin is supported by NSFC U2241211 and NSFC U20B2046.
Kai Wang is the corresponding author.

REFERENCES

[1] Aman Abidi, Lu Chen, Rui Zhou, and Chengfei Liu. 2021. Searching Personalized k-wing in Large and Dynamic Bipartite Graphs. *CoRR* abs/2101.00810 (2021). arXiv:2101.00810 https://arxiv.org/abs/2101.00810

[2] Aman Abidi, Lu Chen, Rui Zhou, and Chengfei Liu. 2022. Searching Personalized k-wing in Bipartite Graphs. *IEEE Transactions on Knowledge and Data Engineering* (2022).

[3] Alex Beutel, Wanhong Xu, Venkatesan Guruswami, Christopher Palow, and Christos Faloutsos. 2013. CopyCatch: Stopping Group Attacks by Spotting Lockstep Behavior in Social Networks. In *Proceedings of the 22nd International Conference on World Wide Web (WWW '13)*. Association for Computing Machinery, New York, NY, USA, 119–130.

[4] Lu Chen, Chengfei Liu, Rui Zhou, Jiajie Xu, and Jianxin Li. 2021. Efficient Exact Algorithms for Maximum Balanced Biclique Search in Bipartite Graphs. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava (Eds.). ACM, 248–260.

[5] Lu Chen, Chengfei Liu, Rui Zhou, Jiajie Xu, and Jianxin Li. 2022. Efficient maximal biclique enumeration for large sparse bipartite graphs. *Proceedings of the VLDB Endowment* 15, 8 (2022), 1559–1571.

[6] Apurba Das and Srikanta Tirthapura. 2018. Incremental maintenance of maximal bicliques in a dynamic bipartite graph. *IEEE Transactions on Multi-Scale Computing Systems* 4, 3 (2018), 231–242.

[7] Danhao Ding, Hui Li, Zhipeng Huang, and Nikos Mamoulis. 2017. Efficient Fault-Tolerant Group Recommendation Using alpha-beta-core. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, CIKM 2017, Singapore, November 06 - 10, 2017*. ACM, 2047–2050.

[8] Yixiang Fang, Xin Huang, Lu Qin, Ying Zhang, Wenjie Zhang, Reynold Cheng, and Xuemin Lin. 2020. A survey of community search over big graphs. *VLDB J.* 29, 1 (2020), 353–392.

[9] Yixiang Fang, Zhongran Wang, Reynold Cheng, Hongzhi Wang, and Jiafeng Hu. 2018. Effective and efficient community search over large directed graphs. *IEEE Transactions on Knowledge and Data Engineering* 31, 11 (2018), 2093–2107.

[10] Yizhang He, Kai Wang, Wenjie Zhang, Xuemin Lin, and Ying Zhang. 2021. Exploring cohesive subgraphs with vertex engagement and tie strength in bipartite graphs. *Inf. Sci.* 572 (2021), 277–296.

[11] Kartik Lakhotia, Rajgopal Kannan, Viktor K. Prasanna, and César A. F. De Rose. 2020. RECEIPT: REfine CoarsE-grained IndePendent Tasks for Parallel Tip decomposition of Bipartite Graphs. *Proc. VLDB Endow.* 14, 3 (2020), 404–417.

[12] Harry R Lewis. 1983. Michael R. ΠGarey and David S. Johnson. Computers and intractability. A guide to the theory of NP-completeness. WH Freeman and Company, San Francisco1979, x+ 338 pp. *The Journal of Symbolic Logic* 48, 2 (1983), 498–500.

[13] Michael Ley. 2002. The DBLP Computer Science Bibliography: Evolution, Research Issues, Perspectives. In *String Processing and Information Retrieval, 9th International Symposium, SPIRE 2002, Lisbon, Portugal, September 11-13, 2002, Proceedings (Lecture Notes in Computer Science)*, Alberto H. F. Laender and Arlindo L. Oliveira (Eds.), Vol. 2476. Springer, 1–10.

[14] Conggai Li, Fan Zhang, Ying Zhang, Lu Qin, Wenjie Zhang, and Xuemin Lin. 2019. Efficient Progressive Minimum k-core Search. *Proc. VLDB Endow.* 13, 3 (2019), 362–375.

[15] Jianxin Li, Taotao Cai, Ke Deng, Xinjue Wang, Timos Sellis, and Feng Xia. 2020. Community-diversified influence maximization in social networks. *Inf. Syst.* 92 (2020), 101522.

[16] Jinyan Li, Guimei Liu, Haiquan Li, and Limsoon Wong. 2007. Maximal biclique subgraphs and closed pattern pairs of the adjacency matrix: A one-to-one correspondence and mining algorithms. *IEEE Transactions on Knowledge and Data Engineering* 19, 12 (2007), 1625–1637.

[17] Bingkai Lin. 2018. The Parameterized Complexity of the K-Biclique Problem. *J. ACM* 65, 5, Article 34 (aug 2018), 23 pages.

[18] Boge Liu, Long Yuan, Xuemin Lin, Lu Qin, Wenjie Zhang, and Jingren Zhou. 2019. Efficient (α, β)-core computation: An index-based approach. In *The World Wide Web Conference*. Association for Computing Machinery, New York, NY, USA, 1130–1141.

[19] Boge Liu, Long Yuan, Xuemin Lin, Lu Qin, Wenjie Zhang, and Jingren Zhou. 2020. Efficient (α, β)-core computation in bipartite graphs. *VLDB J.* 29, 5 (2020), 1075–1099.

[20] Boge Liu, Fan Zhang, Wenjie Zhang, Xuemin Lin, and Ying Zhang. 2021. Efficient community search with size constraint. In *37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021*. IEEE, 97–108.

[21] Qing Liu, Xuankun Liao, Xin Huang, Jianliang Xu, and Yunjun Gao. 2023. Distributed (α, β)-Core Decomposition over Bipartite Graphs. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 909–921.

[22] Bingqing Lyu, Lu Qin, Xuemin Lin, Ying Zhang, Zhengping Qian, and Jingren Zhou. 2020. Maximum Biclique Search at Billion Scale. *Proc. VLDB Endow.* 13, 9 (2020), 1359–1372.

[23] Yuliang Ma, Ye Yuan, Feida Zhu, Guoren Wang, Jing Xiao, and Jianzong Wang. 2019. Who Should Be Invited to My Party: A Size-Constrained k-Core Problem in Social Networks. *J. Comput. Sci. Technol.* 34, 1 (2019), 170–184.

[24] Ziyi Ma, Yuling Liu, Yikun Hu, Jianye Yang, Chubo Liu, and Huadong Dai. 2021. Efficient maintenance for maximal bicliques in bipartite graph streams. *World Wide Web* (2021), 1–21.

[25] Arko Provo Mukherjee and Srikanta Tirthapura. 2017. Enumerating Maximal Bicliques from a Large Graph Using MapReduce. *IEEE Trans. Serv. Comput.* 10, 5 (2017), 771–784.

[26] Georgios A Pavlopoulos, Panagiota I Kontou, Athanasia Pavlopoulou, Costas Bouyioukos, Evripides Markou, and Pantelis G Bagos. 2018. Bipartite graphs in systems biology and medicine: a survey of methods and applications. *GigaScience* 7, 4 (2018), giy014.

[27] René Peeters. 2003. The maximum edge biclique problem is NP-complete. *Discrete Applied Mathematics* 131, 3 (2003), 651–654.

[28] Ahmet Erdem Sariyüce and Ali Pinar. 2018. Peeling Bipartite Networks for Dense Subgraph Discovery. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, WSDM 2018, Marina Del Rey, CA, USA, February 5-9, 2018*. ACM, 504–512.

[29] Mauro Sozio and Aristides Gionis. 2010. The community-search problem and how to plan a successful cocktail party. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, July 25-28, 2010*. ACM, 939–948.

[30] Amos Tanay, Roded Sharan, Martin Kupiec, and Ron Shamir. 2004. Revealing modularity and organization in the yeast molecular network by integrated analysis of highly heterogeneous genomewide data. *Proceedings of the National Academy of Sciences* 101, 9 (2004), 2981–2986.

[31] Jun Wang, Arjen P. de Vries, and Marcel J. T. Reinders. 2006. Unifying user-based and item-based collaborative filtering approaches by similarity fusion. In *SIGIR 2006: Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Seattle, Washington, USA, August 6-11, 2006*. ACM, 501–508.

[32] Jianhua Wang, Jianye Yang, Ziyi Ma, Chengyuan Zhang, Shiyu Yang, and Wenjie Zhang. 2023. Efficient Maximal Biclique Enumeration on Large Uncertain Bipartite Graphs. *IEEE Transactions on Knowledge and Data Engineering* (2023).

[33] Kai Wang, Xuemin Lin, Lu Qin, Wenjie Zhang, and Ying Zhang. 2020. Efficient Bitruss Decomposition for Large-scale Bipartite Graphs. In *36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020*. IEEE, 661–672.

[34] Kai Wang, Wenjie Zhang, Xuemin Lin, Lu Qin, and Alexander Zhou. 2022. Efficient Personalized Maximum Biclique Search. In *38th IEEE International Conference on Data Engineering, ICDE 2022, Kuala Lumpur, Malaysia, May 9-12, 2022*. IEEE, 498–511.

[35] Kai Wang, Wenjie Zhang, Xuemin Lin, Ying Zhang, and Shunyang Li. 2022. Discovering Hierarchy of Bipartite Graphs with Cohesive Subgraphs. In *38th IEEE International Conference on Data Engineering, ICDE 2022, Kuala Lumpur, Malaysia, May 9-12, 2022*. IEEE, 2291–2305.

[36] Kai Wang, Wenjie Zhang, Xuemin Lin, Ying Zhang, Lu Qin, and Yuting Zhang. 2021. Efficient and Effective Community Search on Large-scale Bipartite Graphs. In *37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021*. IEEE, 85–96.

[37] Kai Wang, Wenjie Zhang, Ying Zhang, Lu Qin, and Yuting Zhang. 2023. Discovering Significant Communities on Bipartite Graphs: An Index-Based Approach. *IEEE Trans. Knowl. Data Eng.* 35, 3 (2023), 2471–2485.

[38] Kai Wang, Gengda Zhao, Wenjie Zhang, Xuemin Lin, Ying Zhang, Yizhang He, and Chunxiao Li. 2023. Cohesive Subgraph Discovery over Uncertain Bipartite Graphs. *IEEE Transactions on Knowledge and Data Engineering* (2023).

[39] Yue Wang, Ruiqi Xu, Xun Jian, Alexander Zhou, and Lei Chen. 2022. Towards distributed bitruss decomposition on bipartite graphs. *Proceedings of the VLDB Endowment* 15, 9 (2022), 1889–1901.

[40] Yixing Yang, Yixiang Fang, Maria E Orlowska, Wenjie Zhang, and Xuemin Lin. 2021. Efficient bi-triangle counting for large bipartite networks. *Proceedings of the VLDB Endowment* 14, 6 (2021), 984–996.

[41] Kai Yao and Lijun Chang. 2021. Efficient Size-Bounded Community Search over Large Networks. *Proc. VLDB Endow.* 14, 8 (apr 2021), 1441–1453.

[42] Long Yuan, Lu Qin, Wenjie Zhang, Lijun Chang, and Jianye Yang. 2017. Index-based densest clique percolation community search in networks. *TKDE* 30, 5 (2017), 922–935.

[43] Fan Zhang, Haicheng Guo, Dian Ouyang, Shiyu Yang, Xuemin Lin, and Zhihong Tian. 2023. Size-constrained Community Search on Large Networks: An Effective and Efficient Solution. *IEEE Transactions on Knowledge and Data Engineering* (2023).

[44] Yun Zhang, Charles A Phillips, Gary L Rogers, Erich J Baker, Elissa J Chesler, and Michael A Langston. 2014. On finding bicliques in bipartite graphs: a novel algorithm and its application to the integration of diverse biological data types. *BMC bioinformatics* 15, 1 (2014), 110.

[45] Yuting Zhang, Kai Wang, Wenjie Zhang, Xuemin Lin, and Ying Zhang. 2021. Pareto-optimal community search on large bipartite graphs. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 2647–2656.

[46] Yiwei Zhao, Zi Chen, Chen Chen, Xiaoyang Wang, Xuemin Lin, and Wenjie Zhang. 2023. Finding the Maximum k-Balanced Biclique on Weighted Bipartite Graphs. *IEEE Transactions on Knowledge and Data Engineering* 35, 8 (2023), 7994–8007.

[47] Zhaonian Zou. 2016. Bitruss Decomposition of Bipartite Graphs. In *Database Systems for Advanced Applications - 21st International Conference, DASFAA 2016, Dallas, TX, USA, April 16-19, 2016, Proceedings, Part II (Lecture Notes in Computer Science)*, Vol. 9643. Springer, 218–233.