Effective Community Search on Large Attributed Bipartite Graphs

Zongyu Xu
Nanjing University of Science and Technology
zongyu.xu@njust.edu.cn

Yihao Zhang
Nanjing University of Science and Technology
yhzhangeg@163.com

Long Yuan*
Nanjing University of Science and Technology
longyuan@njust.edu.cn

Yuwen Qian
Nanjing University of Science and Technology
admon@njust.edu.cn

Zi Chen
East China Normal University
zchen@sei.ecnu.edu.cn

Mingliang Zhou†
Chongqing University
mingliangzhou@cqu.edu.cn

Qin Mao
Qiannan Normal Coll Nationalities
345379197@qq.com

Weibin Pan
North Information Control Research
Academy Group Co.
nnupwb@163.com

ABSTRACT

Community search over bipartite graphs has attracted significant interest recently. In many applications such as user-item bipartite graph in E-commerce, customer-movie bipartite graph in movie rating website, nodes tend to have attributes, while previous community search algorithm on bipartite graphs ignore attributes, which makes the returned results with poor cohesion with respect to their node attributes. In this paper, we study the community search problem on attributed bipartite graphs. Given a query vertex q, we aim to find attributed (α, β)-communities of G, where the structure cohesiveness of the community is described by an (α, β)-core model, and the attribute similarity of two groups of nodes in the subgraph is maximized. In order to retrieve attributed communities from bipartite graphs, we first propose a basic algorithm composed of two steps: the generation and verification of candidate keyword sets, and then two improved query algorithms Inc and Dec are proposed. Inc is proposed considering the anti-monotonity property of attributed bipartite graphs, then we adopt different generating method and verifying order of candidate keyword sets and propose the Dec algorithm. After evaluating our solutions on eight large graphs, the experimental results demonstrate that our methods are effective and efficient in querying the attributed communities on bipartite graphs.

KEYWORDS

Community search; Bipartite graphs; Attributed graphs.

1 INTRODUCTION

With the proliferation of graph data, research efforts have been devoted to many fundamental problems in managing and analyzing graph data [9–11, 21, 24, 45, 47–52, 54]. Bipartite graphs are widely used to represent the relationships between two different types of entities in many real-world applications, such as user-page networks [7, 36], customer-product networks [35, 40], collaboration networks [8, 30], gene co-expression networks [28, 57]. In these practical networks, community structure naturally exists, and a number of cohesive subgraph models (e.g., (α, β)-core [31], bitruss [43], and biclique [33]) are proposed to capture the communities in the bipartite graphs. Following these models, community search over bipartite graphs that aims to find densely connected subgraphs satisfying specified structural cohesiveness conditions has been studied in applications such as anomaly detection [33], personalized recommendation [29], and gene expression analysis [34].

In the aforementioned real-world applications, the entities modeled by the vertices of bipartite graphs often have properties represented by text strings or keywords. When performing community search over such bipartite graphs, previous studies often only focus on the structural cohesiveness of communities but ignore the attributes of the vertices. However, these attributes are important for making sense of communities[4–6], and taking the attributes into consideration provides more personalization and interpretation regarding the returned results[17, 26], while there are few researches on community search based on attributed bipartite graphs.
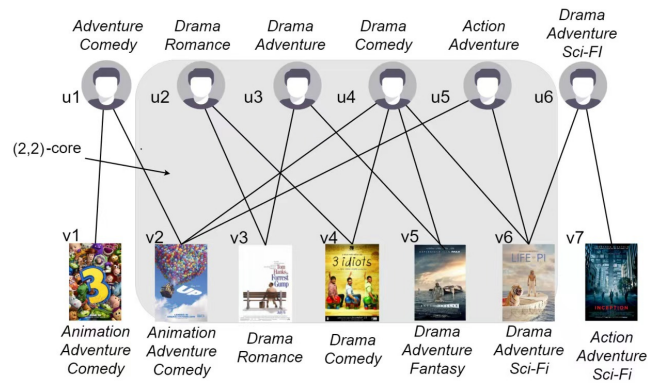

Figure 1: A customer-movie network

**Corresponding authors
†*Corresponding authors

Motivated by this, we study the *attributed* (α, β)-*community search* problem on attributed bipartite graphs in this paper. Specifically, given an attributed bipartite graph G and a query vertex $q \in G$, we aim to find one or more attributed communities in G such that these communities meet both structure cohesiveness (e.g., each vertex in upper layer has at least α neighbors and each vertex in lower layer has at least β neighbors) and keyword cohesiveness (e.g., vertices in the same layer share the most keywords).

Applications. Attributed (α, β)-community has many real-world applications. For example,

- Personalized product recommendation. Attributed (α, β)-community can be used to recommend personalized products. Consider the sub customer-movie subnetwork of IMDB (https://www.imdb.com), where the vertices in the upper layer represent the consumers and the associated attributes describe his or her preference for movies, the vertices in the lower layer represent the movies and the associated attributes describe its genres. The platforms can utilize the attributed (α, β)-community model to provide personalized recommendation. For example, as Fig.1 shows, if we regard u_2 as the query customer, we can find a (2,2)-community composed of viewers $\{u2, u3, u4, u5\}$ and movies $\{v2, v3, v4, v5, v6\}$. In this community "u2" who prefer "Drama" and "Romance" movies may not be interested in "v2". We further consider the keyword cohesiveness of this community and find an attributed (2,2)-community containing viewers $\{u2, u3, u4\}$ who share the same preference for "Drama" movies and the movies $\{v3, v4, v5\}$ with genre "Drama". We can recommend the movie "v5" which the user is likely to be interested in to the query viewer "u2".

- Team Formation. In a bipartite graph composed of developers and projects, an edge between a developer and a project indicates that the developer participates in the project, the keywords of developers show their skills while that of projects indicate the technology it requires. When there is a new project to complete, a developer may wish to form a team as cohesive as possible with all developers in this team having the skills that the project requires, which can be supported by an attributed (α, β)-community search over the bipartite graph through specifying keywords of the new project.

Although attributed (α, β)-community search is useful in real applications. it is still inapplicable if the search cannot be finished efficiently, considering that attributed bipartite graph can be very large, and the (structure and keyword) cohesiveness criteria can be complex to handle. A simple way is first to consider all the possible attribute combinations, and then return the corresponding (α, β)-community that have the most shared attributes. However, the possible number of attribute combinations is exponential, which makes this approach infeasible in practice.

To address this problem, we observe that the attributed (α, β)-community owns the anti-monotonicity property, namely, for a given set $\mathcal{A}$ of attributes, if it appears in every vertex of an attributed (α, β)-community, then every subset $\mathcal{A}'$ of $\mathcal{A}$, there exists an attributed (α, β)-community in which every vertex contains $\mathcal{A}'$. Following this observation, we devise efficient algorithms which can significantly reduce the search space when compute the results.

Contributions. In this paper, we make the following contributions.

- The first work on attributed (α, β)-community search over attributed bipartite graphs. In this paper, we propose the (α, β)-community search problem. To the best of our knowledge, this is the first work on attributed (α, β)-community search.
- Efficient algorithms to conduct the (α, β)-community search. Based on the anti-monotonicity property, we devise efficient algorithms to conduct the (α, β)-community search.
- Extensive experiments on real datasets. We conduct extensive experiments to evaluate the performance of the proposed algorithms. The experimental results demonstrates the efficiency of our proposed algorithms.

Outline. The remainder of this paper is organized as follows. Section 2 presents some related works. Section 3 describes the proposed problem and definitions. A basic solution, enumerating all possible keyword sets and searching for (α, β)-communities with the most shared keywords, is described in Section 4. Section 5 describes two more efficient algorithms generating and verifying candidate keyword sets in different ways. Section 6 discusses the obtained results with our approaches. Finally, conclusion will be found in Section 7.

2 RELATED WORK

2.1 Community search on unipartite graphs.

Community search performed on unipartite graphs usually using different cohesiveness models such as k-core[38], k-truss [12], clique[20]. For a detailed survey, see Ref. [18]. Based on k-core, two online algorithms and one index-based algorithm for k-core community search on unipartite graphs are studied, Cui et al.[13] propose a local search algorithm, Sozio et al.[39]propose a global search algorithm, Barbieri et al.[3] propose a tree-like index structure, and Wu et al.[46] study the maximal personalized influential community search. Using k-core, Fang et al.[15, 17, 19] further integrate the attributes of vertices to identify community and then the spatial locations of vertices are also considered to identify community[16, 27, 41]. For the truss-based community search, Huang et al.[23] propose the triangle-connected k-truss community model and then study the closest model.[25], Akbas et al. [2] also study the triangle-connected k-truss community model and propose an index-based search algorithm. Acquisti et al.[1] present an efficient k-clique component detection algorithm and Yuan et al.[53] study the problem of densest clique percolation community search.

2.2 Community search/detection on bipartite graphs.

On bipartite graphs, several existing works [14, 22, 31, 32] extend the k-core model on unipartite graph to the (α, β)-core model. Ding et al.[14]extend the linear k-core mining algorithm to compute (α, β)-core. He et al.[22] first consider both tie strength and vertex engagement on bipartite graphs and propose a novel cohesive subgraph model. Liu et al.[31, 32] present an efficient algorithm based on a novel index to compute (α, β)-core in linear time regarding the result size. Based on the butterfly structure, Sariyuce et al.[37], Wang et al.[42, 43], Zou et al.[58] study the bitruss model in bipartite graphs which is the maximal subgraph where each edge is contained in at least k butterflies. Zhang et al.[55] study the biclique
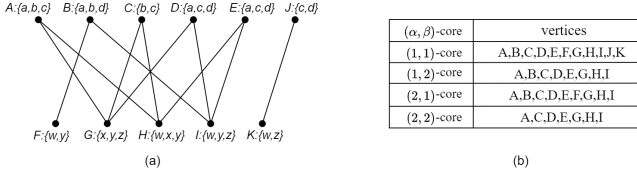
Figure 2: Illustrating the (α, β)-core

enumeration problem. zhang et al.[56] are the first to consider both structure cohesiveness and weight of vertices on bipartite graphs and then propose a novel cohesive subgraph model. Wang et al.[44] present a novel index structure and study the significant community search problem on weighted bipartite graphs, which is the first to study community search on bipartite graphs. However, community search on attributed bipartite graphs remains largely unexplored.

3 PROBLEM DEFINITION

Our problem is defined over an undirected attributed bipartite graph $G = (U, V, E)$, which consists of nodes divided into two separate sets, U and V, such that every edge connects one node in U to another node in V. We use $U(G)$ and $V(G)$ to denote the two disjoint node sets of G and $E(G)$ to represent the edge set of G. Each vertex $u \in U(G)$ ($v \in V(G)$) is associated with a set of keywords denoted by $W_U(u)$ ($W_V(v)$). An edge e between two vertices u and v in G is denoted as (u, v). We denote the number of nodes in $U(G)$ and $V(G)$ as n_u and n_v, the total number of nodes as n and the number of edges in $E(G)$ as m. The set of neighbors of a vertex u in G is denoted as $N(u, G) = \{v \in V(G)|(u, v) \in E(G)\}$, and the degree of u is denoted as $deg(u, G) = |N(u, G)|$. Table 1 lists the symbols used in the paper.

Table 1: Symbols and meanings

Symbol	Meaning
G(U,V,E)	An attributed bipartite graph with vertex set U and V, and edge set E
$W_U(u)$	The keyword set of vertex u in U(G)
$W_V(v)$	The keyword set of vertex v in V(G)
$deg(u, G)$	The degree of vertex u in U(G)
$deg(v, G)$	The degree of vertex v in V(G)
$G[S'_u, S'_v]$	The largest connected subgraph of G s.t. $q \in G[S'_u, S'_v]$, and $\forall u \in G[S'_u, S'_v], S'_u \subseteq W_U(u), \forall v \in G[S'_u, S'_v], S'_v \subseteq W_V(v)$
$G_{(\alpha,\beta)}[S'_u, S'_v]$	The largest connected subgraph of G s.t. $q \in G_{(\alpha,\beta)}[S'_u, S'_v]$, and $\forall u \in G_{(\alpha,\beta)}[S'_u, S'_v], deg(u, G) \geq \alpha, S'_u \subseteq W_U(u), \forall v \in G_{(\alpha,\beta)}[S'_u, S'_v], deg(v, G) \geq \beta, S'_v \subseteq W_V(v)$

Definition 1 ((α, β)-Core). Given a bipartite graph G and two positive integers α and β, a subgraph $C_{\alpha,\beta}$ is an (α, β)-core of G if $deg(u, C_{\alpha,\beta}) \geq \alpha$ for each $u \in U(C_{\alpha,\beta})$ and $deg(v, C_{\alpha,\beta}) \geq \beta$ for each $v \in V(C_{\alpha,\beta})$.

Example 3.1. In Fig.2(a), $\{A, C, D, E, G, H, I\}$ is a (2,2)-core. The (1,1)-core has vertices $\{A, B, C, D, E, F, G, H, I, J, K\}$, and is composed of two (1,1)-core components: $\{A, B, C, D, E, F, G, H, I\}$ and $\{J, K\}$. Each (α, β)-core in Fig.2(a) is listed in Fig.2(b).

Definition 2 ((α, β)-Connected Component). Given a bipartite graph G and its (α, β)-core, $C_{\alpha,\beta}$, a subgraph $G_{\alpha,\beta}$ is an (α, β)-connected component if $(1) G_{\alpha,\beta} \subseteq C_{\alpha,\beta}$ and $G_{\alpha,\beta}$ is connected; $(2) G_{\alpha,\beta}$ is maximal.

Definition 3 ((α, β)-Community). Given a vertex q, we call the (α, β)-connected component containing q the (α, β)-community, denoted as $G_{\alpha,\beta}(q)$.

Definition 4 (Attributed (α, β)-Community). Given an attributed bipartite graph G, two positive integers α and β, a query vertex q and a keyword set $S \subseteq W(q)$ (i.e., $q \in U(G)$), a subgraph g is an attributed (α, β)-community of G if it satisfies the following constraints:

(1) **Connectivity Constraint.** g is a connected subgraph which contains q.
(2) **Structure Cohesiveness Constraint.** $\forall u \in U(g), deg(u, g) \geq \alpha$ and $\forall v \in V(g), deg(v, g) \geq \beta$.
(3) **Keyword Cohesiveness Constraint.** The size of $(|L_U(g)| + |L_V(g)|)$ is maximal, where $L_U(g) = \cap_{u \in U(g)}(W_U(u) \cap S)$ represents the set of keywords shared in S by all vertices of $U(g)$ and $L_V(g) = \cap_{v \in V(g)}(W_V(v))$ represents the set of keywords shared by all vertices of $V(g)$.
(4) **Maximality Constraint.** There exists no other $g' \supset g$ satisfying above constraints with $L_U(g') = L_U(g)$ and $L_V(g') = L_V(g)$.
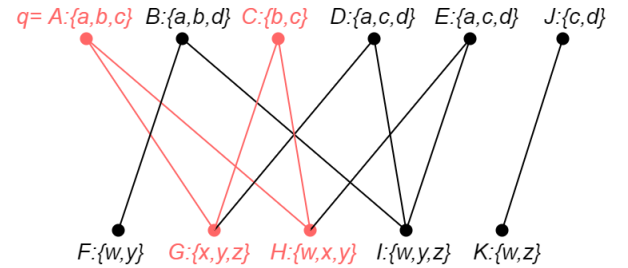

Figure 3: Illustrating an attributed $(2, 2)$-community g

Example 3.2. Considering the bipartite graph G in Fig.2(a), let q=A, α=2, β=2. If $S=\{a, b, c\}$, we can find an attributed (2,2)-community g as Fig.3 illustrates (in red corlor), whose shared keyword set $L_U(g) = \{b, c\}, L_V(g) = \{x, y\}$.

Problem Statement. Given an attributed bipartite graph G, parameters α and β, a query vertex q and a keyword set $S \subseteq W(q)$, the *attributed (α, β)-community search* problem aims to find the attributed (α, β)-communities in G. For ease of representation, we regard q as a vertex in $U(G)$ in this paper. Since the final result must contains q, we regard S as S_U, the maximum keyword set which is possible to be shared by all vertices in $U(G)$.

4 BASIC SOLUTION

We use $G[S_u, S_v]$ to denote the largest connected subgraph of G, where each vertex in $U(G[S_u, S_v])(V(G[S_u, S_v]))$ contains $S_u(S_v)$ and $q \in G[S_u, S_v]$. We use $G_{\alpha,\beta}[S_u, S_v]$ to denote the largest connected subgraph of $G[S_u, S_v]$, in which every vertex in $U(G_{\alpha,\beta}[S_u, S_v])$ has degree being at least α and every vertex in $V(G_{\alpha,\beta}[S_u, S_v])$ has

degree being at least β. We call $\{S_u, S_v\}$ a qualified keyword set for the query vertex q on the graph G, if $G_{\alpha,\beta}[S_u, S_v]$ exists.

Given a query vertex q, a straightforward method to find the attributed (α, β)-communities in G performs three steps. First, for one layer of the bipartite graph which contains q, here we consider it as $U(G)$ and consider S as S_U, all nonempty subsets of S_U, S_{U1}, S_{U2}, ..., $S_{U(2^l-1)}$ ($l = |S_U|$), are enumerated, and for each $v \in V(G)$, we put all different keywords in $W_V(v)$ into S_V and enumerate all nonempty subsets of S_V (i.e., S_{V1}, S_{V2}, ..., $S_{V(2^k-1)}$ ($k = |S_V|$)). Then for each set $\{S_{Ui}, S_{Vj}\}$($1 \leq i \leq 2^l - 1, 1 \leq j \leq 2^k - 1$), we verify the existence of $G_{(\alpha,\beta)}[S_{Ui}, S_{Vj}]$ and compute it when it exists. Finally, we output the subgraphs having the most shared keywords among all $G_{(\alpha,\beta)}[S_{Ui}, S_{Vj}]$.
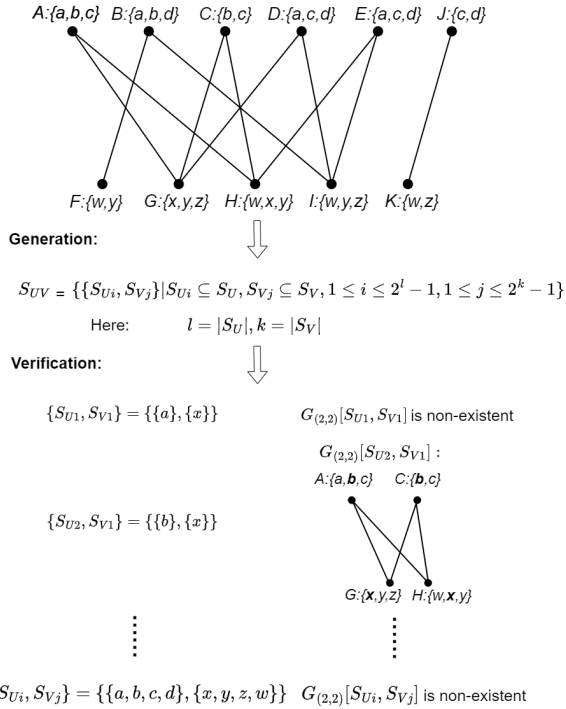

Figure 4: Generation and verification of candidate keyword sets

We can summarize the straightforward method into a two-step framework, generation and verification of candidate keyword sets. Considering the bipartite graph G in Fig.2(a), let q=A, α=2, β=2, S=$\{a, b, c\}$, Fig.4 shows how we find attributed $(2, 2)$-communities through the two-step framework, and the the computational complexity for the proposed framework is the same as that for the *Basic* algorithm mentioned below.

Here we first give the procedure to verify the existence of $G_{\alpha,\beta}$ (q, G') in a given subgraph G' of G for each given candidate keyword set.

THEOREM 4.1. *Given a bipartite graph G, It takes $O(d_{umax} \cdot (n_u + n_v \cdot d_{vmax}))$ to compute $G_{\alpha,\beta}(q, G')$.*

PROOF. There are n_u nodes in $U(G')$, n_v nodes in $V(G')$ and we denote the largest degree of these nodes in $U(G')$($V(G')$) as

---

Algorithm 1: Compute $G_{\alpha,\beta}(q, G')$

> **for** $u \in U(G')$ **do**
>> **if** $deg(u, G') < \alpha$ **then**
>>> remove u and its incident edges from G';
>
> **while** $q \in G'$ **do**
>> $x \leftarrow min_{v \in V(G')} deg(v, G')$;
>> **if** $x \geq \beta$ **then**
>>> return G';
>> **else**
>>> **for** $v \in V(G')$ **do**
>>>> **if** $deg(v, G') < \beta$ **then**
>>>>> **for** $p \in N(v)$ **do**
>>>>>> remove (p, v);
>>>>>> **if** $deg(p, G') < \alpha$ **then**
>>>>>>> remove p and its incident edges from G';
>>>>> remove v;
>
> return $G_{\alpha,\beta}(q, G')$;

d_{umax}(d_{vmax}). Removing all $u \in U(G)$ with degree less than α cost $O(n_u \cdot d_{umax})$, and the while loop in line 4-15 cost $O(n_v \cdot d_{vmax} \cdot d_{umax})$. □

Algorithm 2: Basic

> Initialize ψ using S, φ using $V(G)$;
> **while** *true* **do**
>> $max \leftarrow 0$, $m \leftarrow 0$, $\phi_m \leftarrow \varnothing$;
>> **for** $\psi' \in \psi$ **do**
>>> **for** $\varphi' \in \varphi$ **do**
>>>> find $G[\psi', \varphi']$ from G;
>>>> Compute $G_{(\alpha,\beta)}[\psi', \varphi']$ from $G[\psi', \varphi']$;
>>>> **if** $G_{(\alpha,\beta)}[\psi', \varphi']$ *exists* **then**
>>>>> $m \leftarrow (|\psi'| + |\varphi'|)$;
>>>>> **if** $max \leq m$ **then**
>>>>>> $max \leftarrow m$;
>>>>>> $\phi_m.add(\psi' + \varphi')$;
>
> **if** $\phi_m \neq \varnothing$ **then**
>> output the communities of keyword sets in ϕ_m;

Based on the straightforward method, we present Algorithm2, a baseline query algorithm called *Basic*. The input of *basic* is a bipartite graph G, a query vertex q, two positive integers α and β, and a set S. It first initializes a set, ψ, of candidate keyword sets with each being a nonempty subset of S(i.e., $S_1, S_2, S_3, ..., S_{U(2^l-1)}$ ($l = |S|$)) (line 1). After that, for each vertex in $V(G)$, we enumerate all nonempty subsets of $W_V(v)$, put them into φ and ensure that each element in φ appears only once. In the while loop (lines 2–12), it

first set $m = 0$, indicating the size of current keyword sets, $max = 0$, indicating the maximal size of all keyword sets and an empty set ϕ_m (line 3) for collecting all the qualified keyword sets. Then for each $\psi' \in \psi$ and for each $\varphi' \in \varphi$, it finds $G_{\alpha,\beta}[\psi', \varphi']$ from $G_{\alpha,\beta}$ by considering the keyword and degree constraints (line 4-7). If $G_{\alpha,\beta}[\psi', \varphi']$ exists, the sum of numbers of elements in ψ' and φ' is recorded by m. Then we compare m with max. If $max \leq m$, it then assign m to max and put the set of current keywords in ψ' and φ' into ϕ_m (line 10-12). After checking all the candidate keyword sets in ψ and φ, if there are at least one qualified keyword sets in ϕ_m, it output the communities of keyword sets in ϕ_m (line 13-14).

THEOREM 4.2. *Given a bipartite graph G, Basic computes $G_{\alpha,\beta}[S_u, S_v]$ in $O(n_v \cdot 2^{|S_v|_{max}} \log(n_v \cdot 2^{|S_v|_{max}}) + 2^{|S|} \cdot 2^{|S_v|_{max}} \cdot O(compute\ G_{\alpha,\beta}(q, G')))$.*

PROOF. We use $|S_v|_{max}$ to represent the $W_V(v)$ of largest size among all $v \in V(G)$, Initializing ψ and φ can be completed in $O(2^{|S|} + n_v \cdot 2^{|S_v|_{max}} \log(n_v \cdot 2^{|S_v|_{max}}))$ and the while loop in line 2-12 costs $O(2^{|S|} \cdot 2^{|S_v|_{max}} \cdot O(compute\ G_{\alpha,\beta}(q, G')))$. □

One major drawback of the straightforward method is that we need to compute $(2^l - 1) \times (2^k - 1)$ subsets of attributes and verify the existence of corresponding subgraphs (i.e.,$G_{(\alpha,\beta)}[S_{Ui}, S_{Vj}]$). For large values of l and k, the computation overhead makes this method impractical. To alleviate this problem, we study methods to simplify the generation and verification of candidate keyword sets, and propose two improved algorithms.

5 IMPROVED ATTRIBUTED (α, β)-COMMUNITY SEARCH ALGORITHM

In this section, we shrink the range of possible candidate keyword sets and develop two more efficient algorithms: the incremental algorithm (Inc) verify the candidate sets from smaller to larger ones while the decremental algorithm (Dec) examine larger candidate sets to smaller ones.

5.1 The Incremental Algorithm

Attributed bipartite graphs have the anti-monotonicity property regarding the attributed (α, β)-community search, which is shown in the following lemma:

LEMMA 5.1. *Given a graph G, a vertex $q \in G$, set S_u and S_v of keywords, if there exists a subgraph $G_{\alpha,\beta}[S_u, S_v]$, then there exists a subgraph $G_{\alpha,\beta}[S_u', S_v'] \supseteq G_{\alpha,\beta}[S_u, S_v]$ for any subset $S_u' \subseteq S_u, S_v' \subseteq S_v$.*

PROOF. Based on the definition of $G_{\alpha,\beta}[S_u, S_v]$, each vertex in $U(G_{\alpha,\beta}[S_u, S_v])$ contains S_u and each vertex in $V(G_{\alpha,\beta}[S_u, S_v])$ contains S_v. Consider two new keyword sets $S_u' \subseteq S_u, S_v' \subseteq S_v$, we can easily conclude that each vertex in $U(G_{\alpha,\beta}[S_u, S_v])$ contains S_u' and each vertex in $V(G_{\alpha,\beta}[S_u, S_v])$ contains S_v' as well. Also, note that $q \in G_{\alpha,\beta}[S_u, S_v]$. These two properties imply that there exists one subgraph of G, namely $G_{\alpha,\beta}[S_u, S_v]$, with each vertex in $U(G)$ has degree being at least α and each vertex in $V(G)$ has degree being at least β, such that it contains q and every vertex in

its upper(lower) layer contains $S_u'(S_u')$. It follows that there exists such a subgraph with maximal size (i.e.,$G_{\alpha,\beta}[S_u', S_v']$). □

LEMMA 5.2. *Given two groups of keyword sets $\{S_{u1}, S_{v1}\}$ and $\{S_{u2}, S_{v2}\}$, if $G_{\alpha,\beta}[S_{u1}, S_{v1}]$ and $G_{\alpha,\beta}[S_{u2}, S_{v2}]$ exist, we have $G_{\alpha,\beta}[S_{u1 \cup u2}, S_{v1 \cup v2}] \subseteq G_{\alpha,\beta}[S_{u1}, S_{v1}] \cap G_{\alpha,\beta}[S_{u2}, S_{v2}]$.*

PROOF. Based on Lemma 1, since $\{S_{u1}, S_{v1}\} \subseteq \{S_{u1 \cup u2}, S_{v1 \cup v2}\}$ and $G_{\alpha,\beta}[S_{u1}, S_{v1}]$ exsits, we have $G_{\alpha,\beta}[S_{u1 \cup u2}, S_{v1 \cup v2}] \subseteq G_{\alpha,\beta}[S_{u1}, S_{v1}]$. For the same reason, we have $G_{\alpha,\beta}[S_{u1 \cup u2}, S_{v1 \cup v2}] \subseteq G_{\alpha,\beta}[S_{u2}, S_{v2}]$. It directly follows the lemma. □

This lemma implies, if $\{S_u', S_v'\}$ is generated from $\{S_{u1}, S_{v1}\}$ and $\{S_{u2}, S_{v2}\}$, we can find $G_{\alpha,\beta}[S_u', S_v']$ from $G_{\alpha,\beta}[S_{u1}, S_{v1}] \cap G_{\alpha,\beta}[S_{u2}, S_{v2}]$ directly. Since every vertex in $G_{\alpha,\beta}[S_{u1}, S_{v1}] \cap G_{\alpha,\beta}[S_{u2}, S_{v2}]$ contains both $\{S_{u1}, S_{v1}\}$ and $\{S_{u2}, S_{v2}\}$, we do not need to consider the keyword constraint again when finding $G_{\alpha,\beta}[S_u', S_v']$.

In addition, considering the degree constraint of $G_{\alpha,\beta}[S_u', S_v']$, there is a key observation that, if S_u', S_v' is a qualified keyword set, then there are at least β vextices in $U(G_{\alpha,\beta}[S_u', S_v'])$ containing set S_u' and α vertices in $N(q)$ containing set S_v'. This observation implies, we can generate all the candidate keyword sets directly by using the query vertex q and q' neighbors, without touching other vertices.

Based on above lemmas and observation, we introduce the algorithm Inc. Compared with $Basic$, it shrinks the initial candidate keyword sets and can always verify the existence of $G_{(\alpha,\beta)}[\psi', \varphi']$ within a subgraph of G instead of the entire graph G , and thus the subgraph for such verification shrinks when the candidate set ψ', φ' expands. Therefore, a large sum of redundant computation is reduced during the verification process.

Algorithm 3 presents Inc. First it initializes a set, $\psi\{\psi_1, \psi_2, ..., \psi_i\}$, of candidate keyword sets with each being a keyword of S. Then for each $v \in N(q)$, it puts each keyword in $W_V(v)$ into S_V and initializes a set, $\varphi\{\varphi_1, \varphi_2, ..., \varphi_j\}$, of candidate keyword sets with each being a keyword of S_V (line 1). For each candidate keyword set $\psi_i(\varphi_j)$ in $\psi(\varphi)$, it traverse G and put nodes containing $\psi_i(\varphi_j)$ into $P_i(Q_j)$ (line 2). Considering the key observation that, if $\psi_i(\varphi_j)$ is a qualified keyword set, then there are at least β nodes in $U(G)$ containing ψ_i and α nodes in $V(G)$ containing φ_j, so it removes P_i and ψ_i if $|P_i| < \beta$, and removes Q_j and φ_j if $|Q_j| < \alpha$ as well (line 3). Then, we set $l = 0$, indicating the sizes of current keyword sets, and initialize a set ϕ of $< c, G_{\alpha,\beta}[c] >$ pairs. In a $< c, G_{\alpha,\beta}[c] >$ pair, c contains a set, ψ', of keywords from ψ and a set, φ', of keywords from φ, and $G_{\alpha,\beta}[c]$ is an (α,β)-community of G where each vertex in $U(G_{\alpha,\beta}[c])$ contains ψ' and each vertex in $V(G_{\alpha,\beta}[c])$ contains φ' (line 4). $\forall \psi' \in \psi$ and $\forall \varphi' \in \varphi$, we verify the existence of $G_{(\alpha,\beta)}[\psi', \varphi']$ and put the qualified $< c, G_{\alpha,\beta}[c] >$ pairs into ϕ_l (line 5-10). In the while loop (lines 11–18), for every two $< c, G_{\alpha,\beta}[c] >$ pairs, denoted as $< c_1, G_{\alpha,\beta}[c_2] >$ and $< c_2, G_{\alpha,\beta}[c_2] >$ in ϕ_l, we find $G_{(\alpha,\beta)}[c_1 \cup c_2]$ from $G[c_1 \cup c_2]$, the shared subgraph of $G_{\alpha,\beta}[c_2]$ and $G_{\alpha,\beta}[c_2]$ (line 12-15). If $G_{(\alpha,\beta)}[c_1 \cup c_2]$ exists, we put the pair of $c_1 \cup c_2$ and $G_{(\alpha,\beta)}[c_1 \cup c_2]$ into the set ϕ_{l+1} (line 16-17). When ϕ_l is empty, we stop the loop. Next, we look for the qualified keyword sets c, which contain the most keywords, from ϕ_0 to ϕ_{l-1}. Finally, we output the communities of keyword sets c.

Algorithm 3: Inc

Initialize ψ using S, φ using $N(q)$;

generate $P\{P_1, P_2, ..., P_i\}$ and $Q\{Q_1, Q_2, ..., Q_j\}$ by ψ and φ;

update ψ, φ, P, Q;

$c \leftarrow \varnothing, \phi_l \leftarrow \varnothing, l \leftarrow 0$;

for $\psi_i \in \psi$ **do**

    for $\varphi_j \in \varphi$ **do**

        Compute $G_{(\alpha,\beta)}[\psi_i, \varphi_j]$ from the subgraph induced

          on P_i and Q_j;

        if $G_{(\alpha,\beta)}[\psi_i, \varphi_j]$ *exists* **then**

            $c \leftarrow \{\psi_i, \varphi_j\}$;

            $\phi_l.add(<c, G_{\alpha,\beta}[c]>)$;

while $\phi_l \neq \varnothing$ **do**

    for $<c_1, G_{\alpha,\beta}[c_1]> \in \phi_l$ **do**

        for $<c_2, G_{\alpha,\beta}[c_2]> \in \phi_l$ **do**

            $G[c_1 \cup c_2] \leftarrow G_{\alpha,\beta}[c_1] \cap G_{\alpha,\beta}[c_2]$;

            Compute $G_{(\alpha,\beta)}[c_1 \cup c_2]$ from $G[c_1 \cup c_2]$;

            if $G_{\alpha,\beta}[c_1 \cup c_2]$ *exists* **then**

                $\phi_{l+1}.add(<c_1 \cup c_2, G_{\alpha,\beta}[c_1 \cup c_2]>)$;

                $l \leftarrow l + 1$;

find c when $|c|$ is maximum from ϕ_0 to ϕ_{l-1};

output $G_{\alpha,\beta}[c]$;

THEOREM 5.3. *Given a bipartite graph G, Inc computes $G_{\alpha,\beta}[c]$ in* $O((|S| + |S_V| - 1) \cdot |S| \cdot |S_V|(|S| \cdot |S_V| + O(Compute\ G_{\alpha,\beta}(q, G))))$.

PROOF. In Algorithm 3, we use d to denote the degree of q and $|S_v|_{max}$ to represent the $W_V(v)$ of largest size among all $v \in N(q)$, lines 1 can be completed in $O(|S| + d \cdot |S_v|_{max} \log(d \cdot |S_v|_{max}))$ time. Line 2-3 can be completed in $O(n_u \cdot |S| + n_v \cdot d \cdot |S_v|_{max} \log(d \cdot 2^{|S_v|_{max}}))$ time. Line 5-10 can be completed in $O(|S| \cdot |S_V| \cdot O(Compute\ G_{\alpha,\beta}(q, G)))$ time. In while loop, each time it takes $O(|S| \cdot |S_V|(|S| \cdot |S_V| + O(Compute\ G_{\alpha,\beta}(q, G))))$ time to find qualified communities and put them into a new set ϕ_{l+1}, in the worst case, it runs $(|S| + |S_V| - 1)$ times. □

Example 5.4. Considering G in Fig.2(a), let $q=A$, $\alpha=2$, $\beta=2$ and $S=\{a, b, c\}$, Fig.5(a) shows a (2,2)-core of G. By Algorithm 3, we first find set of keyword sets $\psi\{\{a\}, \{b\}, \{c\}\}$, $\varphi\{\{w\}, \{x\}, \{y\}, \{z\}\}$ and then verify that $G_{2,2}[\{b\}, \{x\}]$, $G_{2,2}[\{b\}, \{y\}]$, $G_{2,2}[\{c\}, \{x\}]$ and $G_{2,2}[\{c\}, \{y\}]$ exists as Fig.5(b) and Fig.5(c) show. In the first while loop, we choose 2 qualified keyword sets from $\{\{b, x\}, \{b, y\}, \{c, x\}, \{c, y\}\}$ and get their union set (e.t.$\{bc, xy\}$ *from* $\{b, x\}$ *and* $\{c, y\}$). By Lemma 2, we only need to verify the new candidate keyword set under nodes in $G_{2,2}[\{b\}, \{x\}]$ and $G_{2,2}[\{c\}, \{y\}]$. Fig.5(d) shows the final attributed community $G_{2,2}[\{b, c\}, \{x, y\}]$.
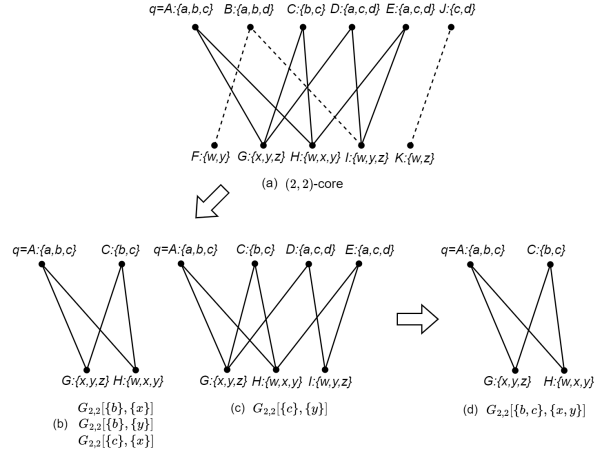

Figure 5: An example of finding $(2,2)$-community by *Inc* method

5.2 The Decremental Algorithm

The decremental algorithm, denoted by *Dec*, differs from the incremental algorithm on both the generation and verification of candidate keyword sets.

5.2.1 Generation of candidate keyword sets.

LEMMA 5.5. *Given a vertex set V of $q's$ neighbors, a qualified keyword set S_u and a set S_V containing all nonempty subsets of $W_V(v)$. For each $S_v \in S_V$, if less than α vertices in V containing S_v, we have $G_{\alpha,\beta}[S_u, S_v]$ doesn't exist.*

PROOF. Assume that $\{S_u, S_v\}$ is a qualified keyword set, then there are at least β vertices in $U(G_{\alpha,\beta}[S_u, S_v])$ containing S_u and α vertices of $q's$ neighbors containing S_v. This contradicts the condition that less than α vertices in V contains S_v, so lemma 3 is proved. □

We generate the candidate keyword sets, ψ, of $U(G)$ by enumerating all nonempty subsets of S_U. For each vertex $v \in N(q)$, we enumerate all nonempty subsets of $W_V(v)$ and put them into a new set φ, the elements of which are different from each other. Then we update the candidate keyword sets by removing those contained by less than α of q' neighbors.

Example 5.6. Consider a query vertex $Q(\alpha = 3)$ with 5 neighbors in Fig.6(a), where the selected keywords of each vertex are listed in the curly braces. For each neighbor of Q, all nonempty subsets of its keyword sets are generated, as shown in Fig.6(b). We can easily filter out the subset which occurs equal to or more than three times and form the set $\varphi\{\{x\}, \{y\}, \{z\}, \{x, y\}\}$.
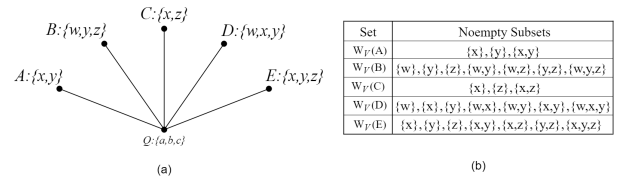

Figure 6: An example of candidate generation in *Dec* method

5.2.2 Verification of candidate keyword sets. As candidates can be obtained using S and q' neighbors directly, we can verify them in a decremental manner (larger candidate keyword sets first and smaller candidate keyword sets later). During the verification process, once finding the attribute (α, β)-communities for candidate keyword sets of the same size, *Dec* does not need to verify smaller candidate keyword sets. Therefore, compared with the incremental algorithm, *Dec* can save the cost of verifying smaller candidate keywords, thus it may be faster practically.

Algorithm 4: Dec

Initialize ψ using S, φ using $N(q)$;
create $P_1, P_2, ..., P_i$ and $Q_1, Q_2, ..., Q_j$ by ψ and φ;
update ψ, φ, P, Q;
$c \leftarrow \varnothing, S \leftarrow \varnothing, ans \leftarrow \varnothing, max \leftarrow 0$;
for $\psi_i \in \psi$ **do**
    for $\varphi_j \in \varphi$ **do**
        $c \leftarrow \{\psi_i, \varphi_j\}$;
        $S.add(c)$;

sort $S\{S_1, S_2, ..., S_{i \times j}\}$ in descending order;
for $S_k \in S$ **do**
    if $|S_k| < max$ **then**
        break;
    else
        compute $G_{(\alpha, \beta)}[S_k]$ from the subgraph induced on
          P_i and Q_j;
        if $G_{(\alpha, \beta)}[S_k]$ *exists* **then**
          $ans.add(G_{(\alpha, \beta)}[S_k])$;
          $max \leftarrow (|S_k|)$;

return ans

Based on the above discussions, we design *Dec* as shown in Algorithm 4. We first generate candidate keyword sets ψ and φ respectively using S and q' neighbors, P_i denote the set of nodes containing ψ_i and Q_j denote the set of nodes containing φ_j (line 1-2). Next, we update ψ, φ, P, Q through removing the vertex sets and the corresponding keyword sets that dissatisfy structure cohesiveness constraint (line 3). Then, we set $max = 0$, indicating the maximal size of all candidate keyword sets, and initialize set S and c, where S contains c and c denotes a set consisting of a keyword set, ψ', from ψ and a keyword set, φ', from φ (line 4). $\forall \psi' \in \varphi$ and $\forall \varphi' \in \varphi$, we generate $(|\psi| \times |\varphi|)$ c and put them into S (line 5-8). For each subset of S, we sort it in descending order according to the number of elements in it (line 9). After that, while $S_k \in S$ and $|S_k| > max$, we verify the existence of $G_{(\alpha, \beta)}[S_k]$ in order. If $G_{(\alpha, \beta)}[S_k]$ exists, we put it into the set *ans* and replace max by $|S_k|$. For the rest set in S, when we find a set with less than max elements, we stop the verification and output the desired (α, β)−communities in *ans*.

THEOREM 5.7. *Given a bipartite graph G, Dec computes $G_{\alpha, \beta}[S_k]$ in $O((2^{|S|} \cdot d \cdot 2^{|S_v|max}) \cdot O(compute\ G_{\alpha, \beta}(q, G)) + n_v \cdot d \cdot 2^{|S_v|max} \log(d \cdot 2^{|S_v|max}))$.*

PROOF. In Algorithm 4, we use d to represent the degree of q, $|S_v|_{max}$ to represent the $W_V(v)$ of largest size among all $v \in N(q)$, we can initialize ψ and φ in $O(2^{|S|} + d \cdot 2^{|S_v|max} \log(d \cdot 2^{|S_v|max}))$ time. Line 2-3 can be completed in $O(n_u \cdot 2^{|S|} + n_v \cdot d \cdot 2^{|S_v|max} \log(d \cdot 2^{|S_v|max}))$ time. In line 5-8, set c can be generated in $O(2^{|S|} \cdot d \cdot 2^{|S_v|max})$ time. Then it takes $O(2^{|S|} \cdot d \cdot 2^{|S_v|max} \log(2^{|S|} \cdot d \cdot 2^{|S_v|max}))$ sorting S in descending order of the number of elements in S. In the worst case, it costs $O((2^{|S|} \cdot d \cdot 2^{|S_v|max}) \cdot O(compute\ G_{\alpha, \beta}(q, G)))$ to find all qualified $G_{(\alpha, \beta)}[S_k]$ in line 10-18. However, it will be much faster in practice. □

6 EXPERIMENTS

This section presents our experimental results. We evaluate the efficiency of the techniques for retrieving attributed (α, β)-communities.

6.1 Experiments setting

Algorithms. We implement and compare following algorithms: 1) a baseline algorithm *Basic* we propose in Section 4, 2) an improved algorithm $Basic^+$ based on Basic, 3) the improved attributed (α, β)-community search algorithm *Inc*, 4) the improved attributed (α, β)-community search algorithm *Dec* in Section 5.

Datasets. We evaluate the algorithms on eight real graphs which are *Enwikibooks*, *Movie*, *IMDB*, *Actor*, *Discogs*, *Idwiki*, *Plwiki* and *Nlwiki*. All the datasets we use can be found in KONECT (http://konect.cc/networks). Note that, for the datasets without attributes, we respectively generate two different kinds of keyword sets for the vertices in the different layer of the bipartite graphs. In each experiment we randomly select 8-13 keywords (average 10) for each vertex. The summary of datasets is shown in Table 1. U and V are vertex layers, $|E|$ is the number of edges, and $\hat{d}$ is the average degree of vertices.

Table 2: Datasets used in our experiments

ID	Dataset	$|U|$	$|V|$	$|E|$	$\hat{d}$
D0	Enwikibooks(Wikibooks edits)	79,268	249,725	766,272	4.66
D1	Movie(Actor movies)	127,823	383,640	1,470,404	5.75
D2	IMDB(komarix-imdb)	685,568	186,414	2,715,604	6.23
D3	Actor(actor2)	303,617	896,302	3,782,463	6.30
D4	Discogs(Discogs)	1,754,823	270,771	5,302,276	5.24
D5	Idwiki(edit-idwiki)	125,481	2,183,494	6,126,592	5.31
D6	Plwiki(edit-plwiki)	207,781	2,664,432	21,219,204	14.78
D7	Nlwiki(edit-nlwiki)	220,847	3,800,349	22,142,951	11.01

The algorithms are implemented in C++ and the experiments are run on a machine having two tetradeca-core Intel Xeon E5-2680 v4 processor, and 251GB of memory, with Ubuntu installed. We set the maximum running time for each test to be 3 days. If a test does not stop in the time limit, we denote the corresponding processing time as INF. The code is open-sourced in https://github.com/892681347/AttributeBigraph.

6.2 Evaluation of retrieving attributed (α, β)-community

Here we evaluate the performance of the algorithms (*Basic*, $Basic^+$, *Inc* and *Dec*) for querying attributed (α, β)-communities. We set

the default values of α and β to 3, and the input keyword set S is set to be the full set of keywords contained in the query vertex. For each dataset, we randomly select 300 query vertices with core numbers greater than or equal to the core number we set. The value of each data is the average result of those 300 queries. For each dataset, we also randomly select 20%, 40%, 60% and 80% of its vertices and obtain four subgraphs induced by these vertex sets, 20%, 40%, 60% and 80% of its keywords and obtain four keyword sets.

The running time of *Basic* is more than 3 days for all experiments, while the *Basic*$^+$ is unpredictable for large graphs (Idwiki, Plwiki and Nlwiki), so we record them as INF, and the effect of *Basic* and *Basic*$^+$ algorithm will not be described separately in the corresponding experiments.

Evaluating the effect of query parameters α and β. We vary α and β to assess the performance of these algorithms. In Fig.7(a)-7(h), β is fixed and the experimental parameter α gradually increases from 2 to 6. We can observe that as α keeps increasing, the running time of *Basic*$^+$, *Inc* and *Dec* algorithms decreases. This is because only a few number of vertices and edges are removed from the original graph when the query parameter α is small. When α is large, the resulting (α, β)-communities are much smaller than the original graph. Thus the size of subgraph directly impacts on the running time of *Basic*$^+$, *Inc* and *Dec* algorithms. Obviously, *Dec* algorithm takes less time than *Basic*$^+$ and *Inc* algorithms in any case. In Fig.8(a)-8(h), we fix α and vary β to compare the query efficiency. In the experiment, we gradually increase the experimental parameter β from 2 to 6 and the experimental results are similar to those when α increases. With the increase of β, the running time of *Basic*$^+$, *Inc* and *Dec* algorithms decreases. This is also because higher β returns a subgraph with less vertices from the original graph, while *Basic*$^+$ and *Inc* algorithms are easier to be affected by the number of vertices.
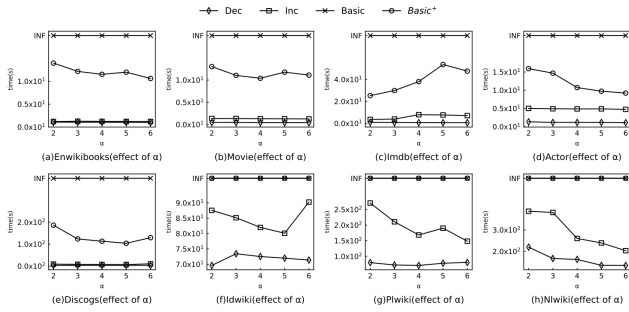

Figure 7: Effect of α

Evaluating the scalability w.r.t. keyword. In this experiment, we evaluate scalability over the fraction of keywords for each vertex. We vary the number of keywords by randomly sampling them from 20% to 100%. As shown in Fig.9(a)-9(h), when varying the number of keywords, the running time of *Basic*$^+$, *Inc* and *Dec* algorithms stably increases. This is because when the number of keywords increase, the number of subgraphs derived from the keywords and the vertices and edges in each subgraph will increase accordingly. The running time of *Basic*$^+$ and *Inc* algorithms increase faster than
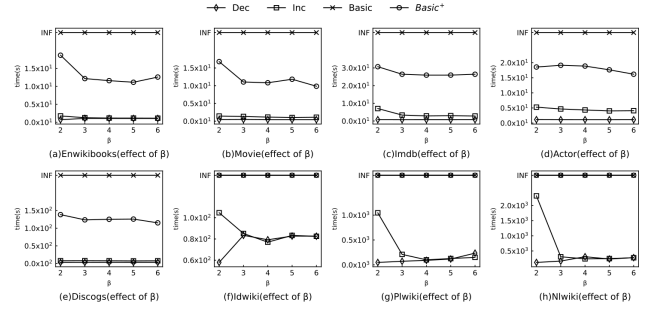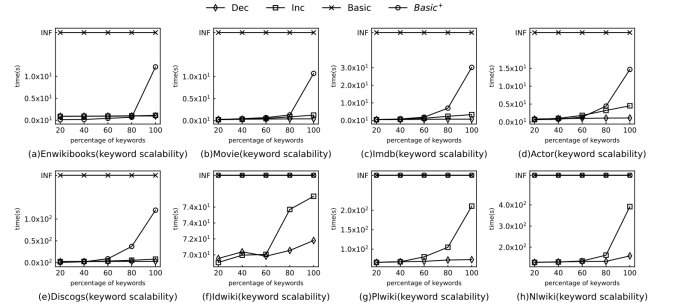

Figure 8: Effect of β
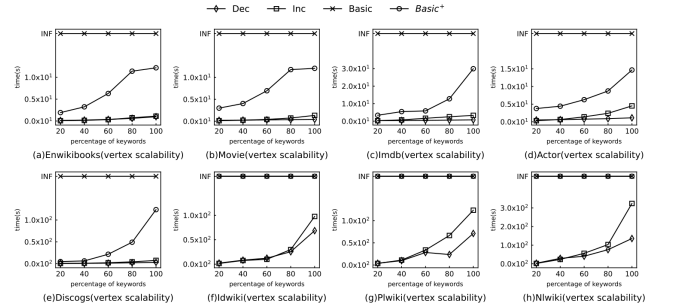

Figure 9: Scalability w.r.t. keyword

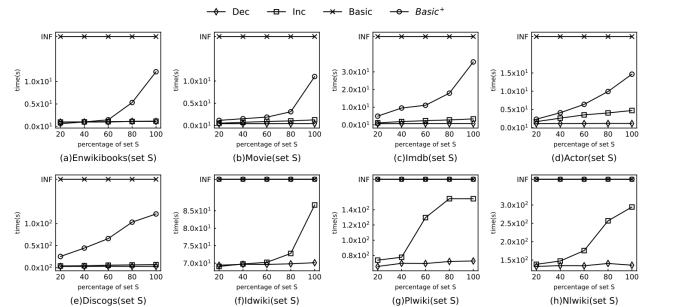**Figure 10: Scalability w.r.t. vertex**

Figure 11: Effect of set S

that of *Dec* algorithm as more keywords are involved, which indicates that *Dec* performs the better and has a good scalability in practice.

Evaluating the scalability w.r.t. vertex. In this experiment, we evaluate the scalability over different fraction of vertices. To test the scalability, we vary the number of vertices and edges by randomly sampling them respectively from 20% to 100% and keeping the induced subgraphs as the input graphs. All the keywords of vertices are considered. Fig.10(a)-10(h) show that, as the number of vertices increasing from 20% to 100%, the running time for $Basic^+$, Inc and Dec algorithms stably increases, and the running time of $Basic^+$ and Inc increases faster than that of Dec. For example, on Imdb, When the number of nodes increases from 20% to 100%, the running time of Dec increase from 0.30s to 0.75s, while that of $Basic^+$ increase from 3.38s to 29.93s and that of Inc increase from 0.28s to 3.32s. We see that Dec has better performance than Inc for most cases, but the opposite may occur in some cases with few vertices. This is because Inc algorithm is easier to be affected by the number of vertices than Dec.

Evaluating the effect of S. In this experiment, we evaluate the effect of the experimental parameter S on the efficiency of the algorithms. For each query vertex, we randomly sampling 20%, 40%, 60%, 80% and 100% keywords of it to form the query keyword set S. As shown in Fig.11(a)-11(h), We can see that with the increase of $|S|$, the running time of $Basic^+$ and Inc increase rapidly, while that of Dec algorithm increases slowly or almost unchanges. For example, on Actor, the running time of Dec increase form 1.08s to 1.13s, while that of $Basic^+$ increase form 2.32s to 14.68s and that of Inc increase form 1.65s to 4.70s. The result shows that Dec performs better than $Basic$ and Inc.

Case study. We conduct queries on the real dateset Southern women (small) from the KONECT (http://konect.cc/networks/), where each vertex in U represents a woman, each vertex in V represents a social activity and each edge indicates the woman participates in the social activity.
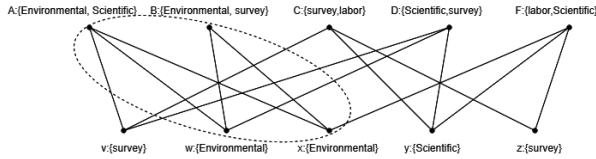

Figure 12: A real person-activity network

We use A as a query vertex, α and β are both set to 2, and S contains the keyword "environmental", the query result is shown in the circled part containing women $\{A, B\}$ and activities $\{w, x\}$ as Fig.11 shows. From the result, we can see the returned people A and B are active participants in environmental activities, and the social activities w and x are all environmental activities with multiple participants from U. In this case, if there is an environmental social activity that needs to recruit team members, then A and B can be given priority because they not only have a preference for environmental social activities but also have experience of cooperation among team members. If we search an (2,2)-community without considering keywords, the result will return the whole women and activities in Fig.12, which includes those who do not often participate in environmental activities. Obviously, the returned candidates

cannot be valid team members expected by an environmental activity. This is because we only consider the structure cohesiveness constraint but ignore the keyword cohesiveness constraint.

7 CONCLUSION

In this paper, we study the attributed (α, β)-community search problem. To solve this problem efficiently, we follow a two-step framework which first generates candidate keyword sets, and then verifies the existence of attributed (α, β)-community according to each candidate keyword set. Then we develop a basic and two improved query algorithms to retrieve the (α, β)-community through verifying the candidate keyword sets in a different order.We conduct extensive experiments on real-world graphs, and the results demonstrate the effectiveness of the attributed (α, β)-community model and the proposed techniques.

REFERENCES

[1] Alessandro Acquisti and Ralph Gross. 2006. Imagined communities: Awareness, information sharing, and privacy on the Facebook. In *International workshop on privacy enhancing technologies*. Springer, 36–58.

[2] Esra Akbas and Peixiang Zhao. 2017. Truss-based community search: a truss-equivalence based indexing approach. *Proceedings of the VLDB Endowment* 10, 11 (2017), 1298–1309.

[3] Nicola Barbieri, Francesco Bonchi, Edoardo Galimberti, and Francesco Gullo. 2015. Efficient and effective community search. *Data mining and knowledge discovery* 29, 5 (2015), 1406–1433.

[4] Kamal Berahmand, Sogol Haghani, Mehrdad Rostami, and Yuefeng Li. 2020. A new Attributed Graph Clustering by using Label Propagation in Complex Networks. *Journal of King Saud University - Computer and Information Sciences* (2020).

[5] Kamal Berahmand, Mehrnoush Mohammadi, Azadeh Faroughi, and Rojiar Pir Mohammadiani. 2022. A novel method of spectral clustering in attributed networks by constructing parameter-free affinity matrix. *Cluster computing* 25-2 (2022).

[6] Kamal Berahmand, Elahe Nasiri, Rojiar Pir mohammadiani, and Yuefeng Li. 2021. Spectral clustering on protein-protein interaction networks via constructing affinity matrix using attributed graph embedding. *Computers in Biology and Medicine* 138 (2021), 104933. https://doi.org/10.1016/j.compbiomed.2021.104933

[7] 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 WWW*. 119–130.

[8] Shuang Cai, Shiwei Ma, Dongzhou Gu, and Chang Wang. 2022. Human-Object Interaction Detection Based on Star Graph. *Int. J. Pattern Recognit. Artif. Intell.* 36, 9 (2022), 2250033:1–2250033:18. https://doi.org/10.1142/S0218001422500331

[9] Yankai Chen, Yixiang Fang, Reynold Cheng, Yun Li, Xiaojun Chen, and Jie Zhang. 2018. Exploring communities in large profiled graphs. *IEEE Transactions on Knowledge and Data Engineering* 31, 8 (2018), 1624–1629.

[10] Yankai Chen, Jie Zhang, Yixiang Fang, Xin Cao, and Irwin King. 2021. Efficient community search over large directed graphs: An augmented index-based approach. In *Proceedings of IJCAI*. 3544–3550.

[11] Zi Chen, Long Yuan, Xuemin Lin, Lu Qin, and Jianye Yang. 2020. Efficient maximal balanced clique enumeration in signed networks. In *Proceedings of The Web Conference 2020*. 339–349.

[12] Jonathan Cohen. 2008. Trusses: Cohesive subgraphs for social network analysis. *National security agency technical report* 16, 3.1 (2008).

[13] Wanyun Cui, Yanghua Xiao, Haixun Wang, and Wei Wang. 2014. Local search of communities in large graphs. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. 991–1002.

[14] 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*. 2047–2050.

[15] Yixiang Fang, Reynold Cheng, Yankai Chen, Siqiang Luo, and Jiafeng Hu. 2017. Effective and efficient attributed community search. *The VLDB journal* 26, 6 (2017), 803–828.

[16] Yixiang Fang, Reynold Cheng, Xiaodong Li, Siqiang Luo, and Jiafeng Hu. 2017. Effective community search over large spatial graphs. *Proceedings of the VLDB Endowment* 10, 6 (2017), 709–720.

[17] Yixiang Fang, Reynold Cheng, Siqiang Luo, and Jiafeng Hu. 2016. Effective Community Search for Large Attributed Graphs. *Proc. VLDB Endow.* 9, 12 (2016), 1233–1244.

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

[19] Yixiang Fang, Zhongran Wang, Reynold Cheng, Hongzhi Wang, and Jiafeng Hu. 2019. Effective and Efficient Community Search Over Large Directed Graphs. *IEEE Transactions on Knowledge and Data Engineering* 31 (2019), 2093–2107.

[20] Yixiang Fang, Kaiqiang Yu, Reynold Cheng, Laks VS Lakshmanan, and Xuemin Lin. 2019. Efficient algorithms for densest subgraph discovery. *arXiv preprint arXiv:1906.00341* (2019).

[21] Kongzhang Hao, Long Yuan, and Wenjie Zhang. 2021. Distributed Hop-Constrained s-t Simple Path Enumeration at Billion Scale. *Proc. VLDB Endow.* 15, 2 (2021), 169–182.

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

[23] Xin Huang, Hong Cheng, Lu Qin, Wentao Tian, and Jeffrey Xu Yu. 2014. Querying k-truss community in large and dynamic graphs. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data.* 1311–1322.

[24] Xin Huang, Laks VS Lakshmanan, and Jianliang Xu. 2017. Community search over big graphs: Models, algorithms, and opportunities. In *Proceedings of ICDE.* IEEE, 1451–1454.

[25] Xin Huang, Laks VS Lakshmanan, Jeffrey Xu Yu, and Hong Cheng. 2015. Approximate closest community search in networks. *arXiv preprint arXiv:1505.05956* (2015).

[26] Xin Huang and Laks V. S. Lakshmanan. 2017. Attribute-Driven Community Search. *Proc. VLDB Endow.* 10, 9 (2017), 949–960.

[27] Cheng Ji, Fan Wu, Zongwei Zhu, Li-Pin Chang, Huanghe Liu, and Wenjie Zhai. 2021. Memory-efficient deep learning inference with incremental weight loading and data layout reorganization on edge systems. *J. Syst. Archit.* 118 (2021), 102183. https://doi.org/10.1016/j.sysarc.2021.102183

[28] Mehdi Kaytoue, Sergei O. Kuznetsov, Amedeo Napoli, and Sébastien Duplessis. 2011. Mining gene expression data with pattern structures in formal concept analysis. *Inf. Sci.* 181, 10 (2011), 1989–2001.

[29] Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew Tomkins. 1999. Trawling the Web for Emerging Cyber-Communities. *Comput. Networks* 31, 11-16 (1999), 1481–1493.

[30] Michael Ley. 2002. The DBLP Computer Science Bibliography: Evolution, Research Issues, Perspectives. In *Proceedings of SPIRE (Lecture Notes in Computer Science)*, Vol. 2476. 1–10.

[31] Boge Liu, Long Yuan, Xuemin Lin, Lu Qin, Wenjie Zhang, and Jingren Zhou. 2019. Efficient (a,β)-core Computation: an Index-based Approach. In *Proceedings of WWW.* 1130–1141.

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

[33] 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.

[34] Sara C. Madeira and Arlindo L. Oliveira. 2004. Biclustering Algorithms for Biological Data Analysis: A Survey. *IEEE ACM Trans. Comput. Biol. Bioinform.* 1, 1 (2004), 24–45. https://doi.org/10.1109/TCBB.2004.2

[35] Wei Qi, Zhenzhen Huang, Dongqing Zhu, and Jiaxu Yu. 2022. Graph Neural Network Social Recommendation Algorithm Integrating Static and Dynamic Features. *Int. J. Pattern Recognit. Artif. Intell.* 36, 9 (2022), 2259019:1–2259019:18. https://doi.org/10.1142/S0218001422590194

[36] Rui Qiao, Ke Feng, Heng He, and Xiaolei Zhong. 2021. Graph Pattern Matching: Capturing Bisimilar Subgraph. *Int. J. Pattern Recognit. Artif. Intell.* 35, 3 (2021), 2150011:1–2150011:18. https://doi.org/10.1142/S0218001421500117

[37] Ahmet Erdem Sarıyüce 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.* 504–512.

[38] Stephen B Seidman. 1983. Network structure and minimum degree. *Social networks* 5, 3 (1983), 269–287.

[39] 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.* 939–948.

[40] 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 *Proceedings of SIGIR.* 501–508.

[41] Kai Wang, Xin Cao, Xuemin Lin, Wenjie Zhang, and Lu Qin. 2018. Efficient computing of radius-bounded k-cores. In *2018 IEEE 34th international conference on data engineering (ICDE).* IEEE, 233–244.

[42] Kai Wang, Xuemin Lin, Lu Qin, Wenjie Zhang, and Ying Zhang. 2019. Vertex Priority Based Butterfly Counting for Large-scale Bipartite Networks. *PVLDB* (2019).

[43] Kai Wang, Xuemin Lin, Lu Qin, Wenjie Zhang, and Ying Zhang. 2020. Efficient Bitruss Decomposition for Large-scale Bipartite Graphs. In *Proceedings of ICDE.* 661–672.

[44] 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 *2021 IEEE 37th International Conference on Data Engineering (ICDE).* IEEE, 85–96.

[45] Xudong Wu, Long Yuan, Xuemin Lin, Shiyu Yang, and Wenjie Zhang. 2019. Towards efficient k-tripeak decomposition on large graphs. In *International Conference on Database Systems for Advanced Applications.* Springer, 604–621.

[46] Yanping Wu, Jun Zhao, Renjie Sun, Chen Chen, and Xiaoyang Wang. 2021. Efficient Personalized Influential Community Search in Large Networks. *Data Science and Engineering* 6, 3 (2021), 310–322.

[47] Yang Yang, Zhao-Yang Fu, De-Chuan Zhan, Zhi-Bin Liu, and Yuan Jiang. 2021. Semi-Supervised Multi-Modal Multi-Instance Multi-Label Deep Network with Optimal Transport. *IEEE Trans. Knowl. Data Eng.* 33, 2 (2021), 696–709.

[48] Yang Yang, Jia-Qi Yang, Ran Bao, De-Chuan Zhan, Hengshu Zhu, Xiao-Ru Gao, Hui Xiong, and Jian Yang. 2021. Corporate Relative Valuation using Heterogeneous Multi-Modal Graph Neural Network. *IEEE Transactions on Knowledge and Data Engineering* (2021).

[49] Yang Yang, De-Chuan Zhan, Yi-Feng Wu, Zhi-Bin Liu, Hui Xiong, and Yuan Jiang. 2021. Semi-Supervised Multi-Modal Clustering and Classification with Incomplete Modalities. *IEEE Trans. Knowl. Data Eng.* 33, 2 (2021), 682–695.

[50] Yang Yang, De-Chuan Zhan, Yi-Feng Wu, and Yuan Jiang. 2018. Multi-network user identification via graph-aware embedding. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining.* Springer, 209–221.

[51] Long Yuan, Lu Qin, Xuemin Lin, Lijun Chang, and Wenjie Zhang. 2016. Diversified top-k clique search. *The VLDB Journal* 25, 2 (2016), 171–196.

[52] Long Yuan, Lu Qin, Xuemin Lin, Lijun Chang, and Wenjie Zhang. 2017. Effective and efficient dynamic graph coloring. *Proceedings of the VLDB Endowment* 11, 3 (2017), 338–351.

[53] Long Yuan, Lu Qin, Wenjie Zhang, Lijun Chang, and Jianye Yang. 2017. Index-based densest clique percolation community search in networks. *IEEE Transactions on Knowledge and Data Engineering* 30, 5 (2017), 922–935.

[54] Junhua Zhang, Long Yuan, Wentao Li, Lu Qin, and Ying Zhang. 2021. Efficient Label-Constrained Shortest Path Queries on Road Networks: A Tree Decomposition Approach. *Proc. VLDB Endow.* 15, 3 (2021), 686–698.

[55] 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), 1–18.

[56] 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.

[57] Zongwei Zhu, Jiawei Geng, Mingliang Zhou, and Bin Fang. 2022. Module Against Power Consumption Attacks for Trustworthiness of Vehicular AI Chips in Wide Temperature Range. *Int. J. Pattern Recognit. Artif. Intell.* 36, 3 (2022), 2250012:1–2250012:19. https://doi.org/10.1142/S0218001422500124

[58] Zhaonian Zou. 2016. Bitruss decomposition of bipartite graphs. In *International conference on database systems for advanced applications.* Springer, 218–233.