informatics

MDPI

Article

Keyword-Constrained Community Search over Bipartite Networks

Junyi Jiang [1,2]*, Xiaoxu Song [3] and Guofeng Jin [1]

[1] School of Business Administration, Liaoning Technical University, Fuxin 123000, China; jinguofeng@126.com
[2] School of Marxism, Liaoning Economy Vocational and Technical College, Shenyang 110000, China
[3] School of Software, Shenyang University of Technology, Shenyang 110000, China; xiaoxu.song84@gmail.com
* Correspondence: jiangjunyi2019@yeah.net

Abstract

Bipartite networks are widely used to model interactions between two types of entities in many real-world applications, such as recommendation systems, social networks, and academic networks. Community search aims to identify cohesive groups containing query entities, while keyword-constrained community search further incorporates semantic requirements. However, keyword-constrained community search over large-scale unweighted bipartite graphs remains insufficiently explored. In this paper, we investigate the keyword-constrained bi-community search problem over unweighted bipartite graphs and propose an efficient graph analytics framework. Specifically, we develop two search algorithms based on global and local search paradigms as baseline solutions, and further propose a Hybrid Index (HI) that integrates keyword inverted lists with structural core-based labels to accelerate query processing. Extensive experiments on four real-world bipartite graph datasets demonstrate that the proposed HI-based approach achieves significant improvements in query efficiency compared with the baseline algorithms.

Keywords: bipartite graph; community search; keyword constraint; (α, β)-core

1. Introduction

With the rapid development of social networks and intelligent information systems, massive heterogeneous data are continuously generated from complex interactions among different types of entities. For example, users and items naturally construct interaction networks in recommendation systems [1,2], while researchers and publications form complex relationships in academic networks [3]. Many of these applications can be modeled as bipartite graphs, where two types of entities are represented as two disjoint vertex sets and their interactions are represented as edges [4,5]. Such bipartite representations provide an effective way to analyze heterogeneous information networks and discover meaningful groups among different types of entities.

Community search (CS) aims to identify cohesive subgraphs containing query vertices under predefined structural constraints. Unlike traditional community detection, which requires partitioning the entire network, community search focuses on retrieving query-specific communities and is more suitable for interactive analysis scenarios. According to whether semantic constraints are considered, community search over bipartite graphs can be categorized into two paradigms: keyword-free community search and keyword-constrained community search (KCS). The former has been extensively studied in recent

Check for updates

Received: 9 August 2026
Revised: 26 October 2026
Accepted: 30 October 2026
Published: 5 November 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

years, and various cohesive community models have been proposed, including the (α, β)-core [2,6–9], bitruss [10–12], and biclique [5]. However, structural cohesiveness alone is insufficient for many real-world applications, where users often require communities with specific semantic meanings. Therefore, KCS further incorporates keyword constraints into community search to identify communities satisfying both structural cohesiveness and semantic relevance.

Such keyword-constrained communities are valuable in many real-world scenarios involving heterogeneous information networks. For example, on e-commerce platforms such as Amazon and Alibaba [13,14], users and items naturally form bipartite networks, and KCS can support semantic-aware group recommendation by retrieving communities satisfying both interaction cohesiveness and user preferences [2]. In academic networks, KCS can support expert finding by identifying researchers with shared interests and related research results [15]. In social networks such as Facebook and Twitter, users and pages form bipartite graphs where edges represent user-page interactions. KCS can assist in discovering abnormal interaction patterns, such as groups of users associated with suspicious pages [16,17].

Existing studies on keyword-constrained community search mainly focus on unipartite graphs, where each vertex is associated with a set of keywords. A variety of methods have been proposed based on different keyword constraints [18–23], including keyword relevance, keyword coverage, keyword-constrained community retrieval, and hierarchical indexing. Unlike unipartite graphs, keywords in bipartite graphs are typically associated with only one vertex layer, while the other vertex layer contains no keyword information. This asymmetric keyword distribution poses new challenges for keyword-constrained community search over bipartite graphs. There are only a few studies related to our work on bipartite graphs. Li et al. [24] proposed the significant community search problem over weighted attributed bipartite graphs based on the (α, β)-core model, aiming to retrieve keyword-aware communities with the maximum minimum attribute weight. Zhang et al. [25] further extended this problem by replacing the (α, β)-core model with the k-bitruss model, aiming to report communities with stronger structural cohesiveness.

Despite its potential in analyzing heterogeneous information networks, existing studies on KCS over bipartite graphs are mainly limited to weighted attributed graphs, and KCS over unweighted bipartite graphs remains unexplored. On the one hand, weighted attributes are not always available in real-world applications. For example, in academic networks, research topics are naturally represented as keywords associated with authors or papers, whereas meaningful numerical weights for these topics are often unavailable or difficult to define. On the other hand, even when attribute weights are available, optimizing them is not always the primary objective. For example, in fraud detection, investigators aim to identify all suspicious communities associated with a queried fraud pattern, since overlooking any suspicious transaction or account may result in missed fraud cases.

Motivated by the above observations, we investigate KCS over unweighted bipartite graphs. Existing community search algorithms generally follow two search paradigms: global search and local search. The former first identifies the entire search space satisfying the structural constraints and then extracts the target community containing the query vertex, whereas the latter incrementally expands from the query vertex to directly locate the target community. Based on these two paradigms, we first develop two baseline algorithms for KCS over bipartite graphs. However, as the sizes of bipartite graphs continue to grow, both baseline algorithms suffer from high online query costs due to repeated graph traversal and keyword filtering. To support efficient online query processing, we design a hybrid index that combines keyword inverted lists with bi-core labels, enabling efficient candidate selection and reducing unnecessary graph exploration during query processing.

Contributions: In this paper, we propose an efficient graph analytics framework for keyword-constrained community search over unweighted bipartite graphs. The proposed framework enables the discovery of semantically meaningful and structurally cohesive communities by jointly considering structural relationships and keyword information. Specifically, we design two baseline algorithms based on global and local search paradigms and further propose a Hybrid Index (HI) that integrates keyword inverted lists with bi-core labels to efficiently filter candidate vertices and reduce unnecessary graph exploration during query processing. Extensive experiments on four real-world bipartite graph datasets demonstrate that our proposed approach can effectively identify keyword-aware communities and achieve significant improvements in query efficiency.

The remainder of this paper is organized as follows. Section 2 reviews related studies on community search and keyword-constrained graph analysis. Section 3 introduces the problem definition and preliminary concepts. Section 4 presents the basic solutions including global and local search algorithms. Section 5 presents the proposed Hybrid Index (HI) and HI-based query processing algorithms. Section 6 reports the experimental results and analysis. Section 7 concludes this paper.

2. Related Work

Community Search on Unipartite Graphs: Community search on unipartite graphs has been extensively studied in recent years. Existing studies mainly differ in the structural cohesiveness models adopted, including k-core [26–29], k-truss [20,30–32], and k-clique [33,34]. To improve the semantic relevance of discovered communities, many studies incorporate attribute constraints into community search models on unipartite attributed graphs [18–21,35,36]. Existing methods define attribute cohesiveness in different ways. For example, some approaches measure cohesiveness based on attribute coverage or the number of shared attributes, which rely on exact attribute matching [18,19]. Other methods utilize attribute distances to capture semantic similarity, such as minimizing the maximum attribute distance or computing the average distance among vertices [20,35]. Query-aware methods further consider the attribute similarity between vertices and the query vertex [21]. Wang et al. [23] further proposed a cohesiveness-aware hierarchical index to accelerate attributed community search under different cohesiveness metrics.

Community Search on Bipartite Graphs: Compared with unipartite graphs, community search on bipartite graphs has received relatively less attention. Existing studies mainly extend structural cohesiveness models, such as (α, β)-core [2,6–9], bitruss [10–12], biclique [5,37], and other models [38,39], to identify dense communities in bipartite graphs. For example, Ding et al. [2] proposed GraphRec, a community search algorithm based on the (α, β)-core model. Luo et al. [7] introduced the concept of bi-core number and studied the maintenance of bi-core structures under dynamic updates. Wang et al. [12] extended cohesive subgraph search to bipartite graphs by introducing the bitruss model, where a k-bitruss is defined as a maximal subgraph in which each edge is contained in at least k butterflies. More recently, Zhang et al. [39] proposed an (α, β)-dense subgraph model for bipartite graphs and developed efficient algorithms for computing the corresponding density decomposition. Although these studies have achieved significant progress in structural community search, they mainly focus on structural cohesiveness and do not consider semantic constraints such as keywords. Only a few studies further incorporate attributes or keywords into bipartite community search. For example, Li et al. [24] proposed the significant community search problem over weighted attributed bipartite graphs based on the (α, β)-core model, aiming to retrieve keyword-aware communities with the maximum minimum attribute weight. Zhang et al. [25] further extended this problem by replacing the

(α, β)-core model with the k-bitruss model, aiming to identify communities with stronger structural cohesiveness.

3. Preliminaries

In this section, we first formulate the Keyword-constrained Bi-Community Search (KBCS) problem. We then propose two naive solutions and present an overview of our proposed approach. Table 1 summarizes the frequently used notations used throughout this paper.

Table 1. Frequently used notations.

Notation	Description
$G = (U, V, E, K)$	A keyword-annotated bipartite graph
$N(u, G)$	Neighbors of vertex u in graph G
$deg_G(u)$	Degree of vertex u in G, i.e., $deg_G(u) = \|N(u, G)\|$
$k(v)$	Keyword set associated with vertex v
Q_K	Query keyword set
$O_{\alpha=1}(u)$	the α-offset of vertex u when $\alpha = 1$
$O_{\beta=1}(u)$	the β-offset of vertex u when $\beta = 1$

We consider a bipartite graph $G = (U, V, E, K)$, where U and V represent the vertex sets in the upper and lower layers, respectively, with $U \cap V = \emptyset$, and $E \subseteq U \times V$ denotes the edge set of G. Each vertex $v \in V$ is associated with a set of keywords $k(v)$, where $k(v) \subseteq K$ and $K = \bigcup_{v \in V} k(v)$ is the global keyword set. Let $N(u, G)$ denote the set of neighbors of a vertex u in G. We represent G by adjacency lists and use $N(w)$ as shorthand for $N(w, G)$ when the graph is clear from the context. The degree of vertex u is denoted as $deg_G(u) = |N(u, G)|$. For a subgraph $H \subseteq G$, we use $U(H)$ and $V(H)$ to denote the upper and lower vertex sets of H, respectively.

Definition 1. (α, β)-core (bi-core) [2]. *The (α, β)-core of a bipartite graph G is the maximal subgraph C of G with vertex sets $U' \subseteq U$ and $V' \subseteq V$, where $\forall u \in U', deg_C(u) \geq \alpha$ and $\forall v \in V', deg_C(v) \geq \beta$.*

Given two bi-cores (α_i, β_i)-core and (α_j, β_j)-core, if $\alpha_i \geq \alpha_j$ and $\beta_i \geq \beta_j$, then the (α_i, β_i)-core is a subgraph of the (α_j, β_j)-core, i.e., (α_i, β_i)-core $\subseteq (\alpha_j, \beta_j)$-core. For fixed (α, β), the core is unique. Bi-cores are nested under componentwise increases in the parameters, while cores with incomparable parameter pairs need not contain one another. However, similar to the classic k-core, an (α, β)-core is not necessarily connected. Therefore, we can further partition the (α, β)-core into (α, β)-communities based on connectivity.

Definition 2. (α, β)-community (Bi-community) [40]. *Given a subgraph H of the (α, β)-core, H is an (α, β)-community if it is connected and no other connected subgraph of the (α, β)-core contains it as a subgraph.*

Definition 3. Keyword-constrained $(\boldsymbol{\alpha, \beta})$-community. *Given a bipartite graph G, a query vertex q, degree constraints α and β, and a query keyword set $Q_K \subseteq K$, a subgraph H is a keyword-constrained (α, β)-community if*

1. *H is connected and contains q;*
2. *$\forall u \in U(H), deg_H(u) \geq \alpha$ and $\forall v \in V(H), deg_H(v) \geq \beta$;*
3. *$\forall v \in V(H)$, $k(v) \cap Q_K \neq \emptyset$;*
4. *H is maximal: no other subgraph satisfies (1)–(3) and contains H.*

Statement 1: Given a bipartite graph $G = (U, V, E, K)$, a query vertex q, parameters α and β, and a query keyword set Q_K, the keyword-constrained bi-community search problem (KBCS) is to identify the keyword-constrained (α, β)-community containing q.

Example 1. *Figure 1 illustrates an example of KBCS. Given a query vertex $q = u_1$ and a query keyword set $Q_K = \{a, b, c\}$, we aim to find the keyword-constrained $(1, 3)$-community containing u_1. The returned community consists of vertices $\{u_1, u_2, u_3, u_4, u_5, v_1, v_2, v_4\}$. The degrees of all upper-layer vertices are no less than $\alpha = 1$, while the degrees of all lower-layer vertices are no less than $\beta = 3$. Moreover, the keyword sets of v_1, v_2, and v_4 overlap with Q_K, satisfying the keyword constraint. Therefore, this subgraph is a valid keyword-constrained $(1, 3)$-community.*

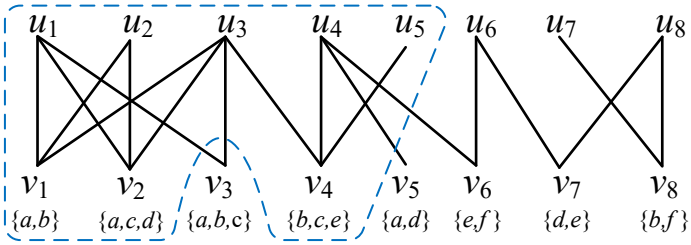


Figure 1. An example of KBCS. The dashed lines indicate the $(1, 3)$-community satisfying $\{a, b, c\}$.

4. Basic Solutions

Two classic ideas are widely used in community search: *global search* and *local search*. Global search first computes a subgraph satisfying the required constraints over the entire graph and then extracts the target community containing the query vertex. In contrast, local search starts from the query vertex and explores only its local neighborhood to identify the target community.

Inspired by these two ideas, we propose two naive solutions for the KBCS.

4.1. Global Algorithm

The global algorithm first prunes the graph based on the keyword constraint and (α, β)-core, and then extracts the connected component containing the query vertex.

Description: Algorithm 1 illustrates the global algorithm for KBCS, which consists of three phases. First, keyword filtering is performed by removing every vertex $v \in V$ satisfying $k(v) \cap Q_K = \emptyset$, together with its incident edges (lines 1–3). Second, the algorithm iteratively removes upper vertices with degree smaller than α and lower vertices with degree smaller than β until the remaining graph forms an (α, β)-core (lines 4–5). Finally, the algorithm performs a breadth-first traversal from q to obtain the keyword-constrained (α, β)-community (lines 6–12).

Complexity analysis: The keyword filtering, (α, β)-core pruning, and breadth-first traversal phases each process every vertex and edge at most once. Therefore, the overall time complexity of the global algorithm is $O(|U| + |V| + |E|)$.

4.2. Local Algorithm

In contrast to the global algorithm, which processes the entire graph before extracting the target community, the local algorithm explores only the neighborhood of the query vertex and dynamically maintains the candidate community during expansion.

Description: Algorithm 2 presents the local algorithm for KBCS. It starts from the query vertex and incrementally expands a candidate set (line 1). During the expansion, a vertex is explored only if it satisfies the corresponding degree constraint and, for lower vertices, the keyword constraint; otherwise, it is recursively removed by DELETECAND, which further eliminates vertices violating the (α, β) constraints (lins 2–7). Finally, if q has been removed,

the algorithm returns $\emptyset$; otherwise, a BFS in the remaining candidate-induced subgraph returns the connected component containing q (lins 8–11).

Algorithm 1: Global Algorithm for KBCS

Input: Bipartite graph $G = (U, V, E, K)$, query vertex q, parameters α, β, keyword set Q_K.

Output: Keyword-constrained (α, β)-community H containing q.

```
1  foreach v ∈ V do
2      if k(v) ∩ Q_K = ∅ then
3          remove v and its incident edges from G;
4  while ∃w ∈ U ∧ deg_G(w) < α or ∃w ∈ V ∧ deg_G(w) < β do
5      remove w and its incident edges from G;
6  F ← ∅ ; H ← ∅ ;
7  F.push(q) ; H ← H ∪ q;
8  while F ≠ ∅ do
9      w ← F.front() ; F.pop();
10     foreach w' ∈ N(w) ∧ w' is unvisited do
11         F.push(w') ; H ← H ∪ w';
12 return H;
```

Algorithm 2: Local Algorithm for KBCS

Input: Bipartite graph $G = (U, V, E, K)$, query vertex q, parameters α, β, keyword set Q_K.

Output: Keyword-constrained (α, β)-community H containing q.

```
1  Cand ← {q};
2  foreach w ∈ Cand do
3      if w ∈ U ∧ deg_G(w) ≥ α or w ∈ V ∧ deg_G(w) ≥ β ∧ k(w) ∩ Q_K ≠ ∅ then
4          foreach w' ∈ N(w) ∧ w' is unvisited do
5              add w' to Cand;
6      else
7          DeleteCand(Cand, w);
8  if q ∉ Cand then
9      H ← ∅;
10 else
11     H ← the connected component containing q in Cand;
12 return H;
13 Function DeleteCand(Cand, w)
14     remove w from Cand;
15     foreach w' ∈ N(w) do
16         deg_G(w') ← deg_G(w') − 1;
17         if w' ∈ Cand ∧ w' ∈ U ∧ deg_G(w') < α or
            w' ∈ Cand ∧ w' ∈ V ∧ deg_G(w') < β then
18             DeleteCand(Cand, w');
```

Complexity analysis: Each vertex is inserted into and removed from the candidate set at most once, and each edge is processed at most a constant number of times during expansion and recursive pruning. Therefore, the overall time complexity is $O(|V(H)| + |E(H)|)$, where H is the explored subgraph. In the worst case, it becomes $O(|V| + |E|)$.

4.3. Discussion of Basic Solutions

The above two algorithms provide straightforward solutions to KBCS by following the global-search and local-search paradigms, respectively. However, both algorithms suffer from considerable online overhead when processing large bipartite graphs.

The global algorithm performs keyword filtering and (α, β)-core decomposition over the entire graph before identifying the target community. Consequently, a large number of irrelevant vertices and edges may still be processed even though they are unrelated to the query. In contrast, the local algorithm restricts the search to the neighborhood of the query vertex, but it still repeatedly verifies keyword and structural constraints during expansion and recursive pruning, resulting in redundant computations. The key observation is that both algorithms spend substantial online cost on determining the valid search space under keyword and structural constraints. Therefore, an effective mechanism that narrows the search space before graph traversal can significantly reduce the online query cost. Motivated by this observation, we propose a hybrid index that combines keyword information with bi-core labels.

5. Hybrid Index

As discussed in Section 4, both the global and local algorithms suffer from substantial online computational overhead due to repeated keyword filtering and structural constraint verification during graph traversal. The key bottleneck lies in the fact that these algorithms dynamically determine the valid search space at query time, which becomes increasingly expensive as the graph size grows. To address this issue, we design a Hybrid Index (HI) that leverages offline preprocessing to encode both keyword information and structural properties, enabling efficient pruning of irrelevant vertices before graph traversal.

5.1. The Structure of HI

The design of HI is motivated by two key observations. First, keyword information is static and does not change with query parameters. Therefore, a keyword inverted index can be pre-built to efficiently retrieve vertices satisfying keyword constraints during query processing. Second, bi-cores exhibit a hierarchical nesting property, where a (α, β)-core with larger parameters is contained in those with smaller parameters. Based on this property, we construct bi-core offset labels for each vertex to record the maximum (α, β) parameters that the vertex can satisfy. Although keyword constraints may break the original bi-core structure, the bi-core offset labels can still be used to quickly determine whether a vertex satisfies the structural constraints of a query and enable effective pruning.

Based on these observations, we design the Hybrid Index (HI), which consists of two core components: the Keyword Inverted Index and the bi-core offset labels. The keyword inverted index is used to quickly identify candidate vertices satisfying keyword constraints, while the bi-core offset labels filter vertices that do not satisfy structural constraints. By combining these two types of index information, HI can effectively reduce the candidate search space before applying the global or local search algorithms, thereby reducing the online query cost.

5.1.1. Keyword Inverted Index

The keyword inverted index is designed to efficiently retrieve vertices satisfying keyword constraints before graph traversal. It maps each keyword to a set of vertices

containing that keyword. Formally, for a keyword k, we maintain an inverted list $I(k)$ storing all vertices v such that $k \in k(v)$. Since keywords only appear on one side of the bipartite graph (i.e., V), the index is constructed only over this partition. Given a query keyword set QK, the candidate vertices satisfying the keyword constraints can be quickly obtained by accessing the corresponding inverted lists.

To further optimize query performance, we apply the following indexing strategy: Degree-based ordering: Vertices in each inverted list are sorted by their degree in descending order. This strategy is based on the observation that a vertex with degree smaller than the required β value cannot belong to any (α, β)-core, since it cannot satisfy the minimum degree constraint on the V-side. Therefore, vertices with larger degrees are more likely to satisfy the structural constraints and remain in the final community.

Lemma 1. *For a vertex $v \in I(k)$, where $k \in Q_K$, if $\deg(v) < \beta$, then v cannot belong to any keyword-constrained (α, β)-community.*

Proof. It is straightforward from the degree constraint of (α, β)-core that every vertex in V must satisfy $\deg(v) \geq \beta$. Therefore, v cannot belong to any keyword-constrained (α, β)-community. □

Example 2. *According to the bipartite graph shown in Figure 1, the keyword sets of vertices in V are used to construct the inverted index. For example, in Table 2, v_1, v_2, v_3, and v_5 contain keyword a, so the corresponding inverted list is $I(a) = \{v_1, v_2, v_3, v_5\}$. Similarly, we can obtain $I(b) = \{v_1, v_3, v_4, v_8\}$ and $I(c) = \{v_2, v_3, v_4\}$. Given a query keyword set $Q_K = \{a, b, c\}$, the corresponding inverted lists are accessed to retrieve the candidate vertices. By combining these lists, we obtain $I(a) \cup I(b) \cup I(c) = \{v_1, v_2, v_3, v_4, v_5, v_8\}$.*

Table 2. An example of a keyword inverted list.

Keyword	Inverted List	Keyword	Inverted List
a	$\{v_1, v_2, v_3, v_5\}$	b	$\{v_1, v_4, v_3, v_8\}$
c	$\{v_2, v_4, v_3\}$	d	$\{v_2, v_5, v_7\}$
e	$\{v_4, v_6, v_7\}$	f	$\{v_6, v_8\}$

5.1.2. Bi-Core Offset Labels

We adopt the bi-core offset labeling scheme [17] and integrate it with keyword inverted lists. Unlike the traditional k-core label in unipartite graphs, which can be represented by a single value, a bi-core requires two parameters (α, β) to characterize its structural constraint. Therefore, multiple offset values are required to describe the structural participation of a vertex. For a vertex $w \in U \cup V$ and a given α, the α-offset $O_{\alpha=i}(w)$ is defined as the maximum β value such that w can be contained in an (α, β)-core. Similarly, the β-offset $O_{\beta=j}(w)$ represents the maximum α value such that w can be contained in an (α, β)-core.

Example 3. *Consider the bipartite graph shown in Figure 2. We fix $\alpha = 1$ and compute the corresponding α-offset values according to different $(1, \beta)$-cores. The vertices u_1-u_5, v_1, v_2, and v_4 are contained in the $(1,3)$-core, indicating that their maximum β value under $\alpha = 1$ is 3, i.e., $O_{\alpha=1}(w) = 3$. The vertices u_6-u_8, v_3, and v_6-v_8 are not contained in the $(1,3)$-core but remain in the $(1,2)$-core. Therefore, their α-offset values are $O_{\alpha=1}(w) = 2$. The vertex v_5 only exists in the $(1,1)$-core, and thus $O_{\alpha=1}(v_5) = 1$. This example demonstrates that the α-offset records the maximum β value that a vertex can satisfy when α is fixed.*

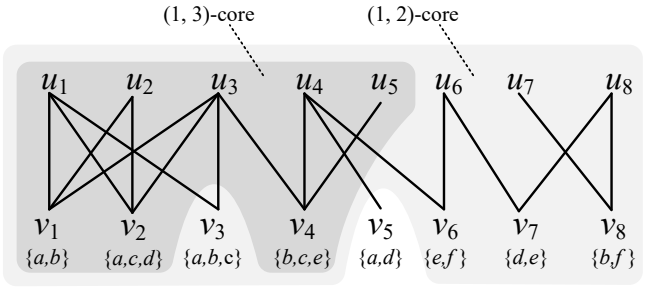


Figure 2. An example of bi-core offset labels. The shaded areas indicate the $(1,3)$-core and $(1,2)$-core, respectively.

For a vertex w, the number of α-offset values is bounded by the maximum possible α value, while the number of β-offset values is bounded by the maximum possible β value. In large-scale graphs, these values may become large, resulting in additional storage overhead if all possible offsets are maintained. To reduce the label size, we store only selected offset values needed for query processing. Let δ denote the largest value such that a (δ,δ)-core exists. We obtain δ as the maximum core number from a κ-core decomposition of the underlying undirected graph [27], since a κ-core of this graph is a (κ,κ)-core. A nonempty (α,β)-core satisfies $\min(\alpha,\beta) \leq \delta$, although either parameter individually may exceed δ. Our label representation is designed for subsequent query processing: when $\alpha \leq \delta$, we use the α-offset to check whether it is at least β; when $\alpha > \delta$, we use the β-offset to check whether it is at least α. Thus, δ determines the selection and use of the stored offsets, rather than bounding all feasible parameter values. Accordingly, the label of a vertex consists of multiple offset values:

$$L(w) = \{O_{\alpha=1}(w), O_{\alpha=2}(w), ..., O_{\alpha=\delta}(w), O_{\beta=1}(w), O_{\beta=2}(w), ..., O_{\beta=\delta}(w)\}.$$

Under this label representation, each vertex stores at most 2δ offset values. This compact representation enables efficient structural pruning during query processing.

Lemma 2. *Given a pair of parameters (α_i,β_i), if $\beta_i > O_{\alpha=\alpha_i}(w)$, then vertex w cannot be contained in any (α_i,β_i)-core. Similarly, if $\alpha_i > O_{\beta=\beta_i}(w)$, then vertex w cannot be contained in any (α_i,β_i)-core.*

Proof. According to the definition of α-offset, $O_{\alpha=\alpha_i}(w)$ represents the maximum β value such that vertex w can be contained in an (α_i,β)-core. Let $C_{a,b}$ denote the (a,b)-core and $W(C_{a,b}) = U(C_{a,b}) \cup V(C_{a,b})$. Suppose $\beta_i > O_{\alpha=\alpha_i}(w)$ and $w \in W(C_{\alpha_i,\beta_i})$. Then, $O_{\alpha=\alpha_i}(w) = \max(\{b \geq 1 : w \in W(C_{\alpha_i,b})\} \cup \{0\}) \geq \beta_i > O_{\alpha=\alpha_i}(w)$, a contradiction. Similarly, if $\alpha_i > O_{\beta=\beta_i}(w)$ and $w \in W(C_{\alpha_i,\beta_i})$, then $O_{\beta=\beta_i}(w) = \max(\{a \geq 1 : w \in W(C_{a,\beta_i})\} \cup \{0\}) \geq \alpha_i > O_{\beta=\beta_i}(w)$, again a contradiction. Here, a and b range over positive integers. □

Complexity analysis: The space overhead of HI consists of the keyword inverted index and the bi-core offset labels. The inverted index stores each vertex in V according to its associated keywords, resulting in $O(k^-|V|)$ space, where k^- is the average number of keywords per vertex. Each vertex stores at most $2\delta - 1$ offset values in the bi-core offset labels, leading to $O(\delta|U| + \delta|V|)$ space. Therefore, the overall space complexity of HI is $O(\delta|U| + (\delta + k^-)|V|)$.

5.2. Construction of HI

The HI is constructed offline and consists of two components: the keyword inverted index and the bi-core offset labels. The keyword inverted index is built by scanning the

keyword information of vertices in V, while the bi-core offset labels are generated based on the hierarchical structure of bi-cores.

Description: Algorithm 3 illustrates the construction process of HI. First, we construct the keyword inverted index by traversing all vertices in V, and compute δ in $O(m)$ time using CoreDecomposition(G), which returns the maximum core number [27]. For each vertex v, every keyword $k \in k(v)$ is inserted into the corresponding inverted list $I(k)$. After construction, vertices in each inverted list are sorted according to their degrees in descending order, which supports efficient candidate filtering during query processing. Then, we construct the bi-core offset labels. For each fixed α, we iteratively increase β and perform a peeling process on the original graph, repeatedly removing violating vertices at each threshold until no violation remains. When a vertex violates the degree constraint of the current (α, β)-core, it is removed from the graph and its corresponding offset value is recorded. Specifically, when a vertex w is removed under the fixed α and current β, we set $O_{\alpha=\alpha_i}(w) = \beta - 1$ (line 10). After obtaining all α-offset values, the same procedure is performed by exchanging α and β to compute the β-offset values (line 13). Finally, all offset values are stored as the bi-core offset labels of each vertex.

Algorithm 3: Construction of Hybrid Index

Input: Bipartite graph $G = (U, V, E, K)$
Output: Hybrid Index HI

```
1  Initialize I(·) and O(·) ; δ ← CoreDecomposition(G);
2  foreach vertex v ∈ V do
3      foreach keyword k ∈ k(v) do
4          add v into I(k);
5  for α from 1 to δ do
6      G⁻ = (U⁻, V⁻, E⁻, K⁻) ← G;
7      β ← 1;
8      while G⁻ ≠ ∅ do
9          while there exists w ∈ U⁻ with |N(w, G⁻)| < α or w ∈ V⁻ with
             |N(w, G⁻)| < β do
10             O_α(w) ← β − 1;
11             remove w and its incident edges from G⁻;
12         β ← β + 1;
13 run Lines 5-12 by swapping α with β;
```

Complexity analysis: The construction of the keyword inverted index requires scanning each keyword of vertices in V, resulting in $O(k^-|V|)$ time, where k^- denotes the average number of keywords per vertex. The algorithm performs $O(\delta)$ fixed-parameter peeling runs, each taking $O(|E|)$ time, resulting in a total time complexity of $O(\delta|E|)$. Therefore, the overall construction complexity of HI is $O(\delta|E| + k^-|V|)$.

5.3. HI-Based KBCS

The HI-based KBCS framework integrates the keyword inverted index and bi-core offset labels to reduce the search space before search. The inverted index retrieves keyword-related vertices, while the offset labels prune vertices that violate the (α, β) structural constraints. The global or local algorithm is then applied on the remaining candidate graph to obtain the final keyword-constrained (α, β)-community.

Description: Algorithm 4 illustrates the query process of HI-based KBCS. Given a query vertex q, parameters (α, β), and keyword set Q_K, the algorithm first accesses the

inverted lists of query keywords. For each retrieved vertex, its bi-core offset label is checked to determine whether it satisfies the structural constraint. Vertices that violate the constraint are directly pruned, while the remaining vertices and their valid neighbors are added into the candidate set *Cand* (Lines 1–8). Finally, the global or local search algorithm is performed on the candidate subgraph G_{Cand} to obtain the keyword-constrained (α, β)-community containing q (Lines 9–12).

Algorithm 4: HI-based KBCS Framework

Input: Bipartite graph $G = (U, V, E, K)$, query vertex q, parameters α, β, keyword set Q_K, Hybrid Index HI
Output: Keyword-constrained (α, β)-community H containing q

```
1  Cand ← ∅;
2  foreach k ∈ Q_K do
3      foreach v ∈ I(k) do
4          if α ≤ δ ∧ O_α(v) ≥ β or α > δ ∧ O_β(v) ≥ α then
5              add v into Cand;
6              foreach u ∈ N(v) do
7                  if α ≤ δ ∧ O_α(u) ≥ β or α > δ ∧ O_β(u) ≥ α then
8                      add u into Cand;
9  if global search then
10     H ← GlobalAlgorithm(G_Cand, q, α, β, Q_K);
11 else
12     H ← LocalAlgorithm(G_Cand, q, α, β, Q_K);
13 return H;
```

Complexity analysis: For a query keyword set Q_K, retrieving vertices from the inverted index requires $O(\sum_{k \in Q_K} |I(k)|)$ time. Checking the offset label of each vertex takes constant time, and the neighbor expansion for retained vertices requires $O(\sum_{v \in Cand} |N(v)|)$ time. Therefore, the filtering phase of HI has a total complexity of $O(\sum_{k \in Q_K} |I(k)| + \sum_{v \in Cand} |N(v)|)$. After filtering, the subsequent global or local search is performed on the reduced candidate graph G_{Cand}, which avoids traversal over the original graph.

6. Experiments

In this section, we evaluate the performance of the proposed methods on real-world bipartite graph datasets. We aim to study the efficiency and effectiveness of the proposed HI-based algorithms, as well as the impact of the Hybrid Index on query processing.

6.1. Experimental Setup

Datasets: We conduct extensive experiments on four real-world bipartite graphs. Among them, DBLP (http://dblp.uni-trier.de/xml/ (accessed on 3 July 2026)) is an academic network; ML (https://grouplens.org/datasets/movielens (accessed on 18 June 2026)) and AN (https://www.kaggle.com/datasets/CooperUnion/anime-recommendations-database (accessed on 3 July 2026)) are movie networks; and EP (http://konect.cc/networks/ (accessed on 13 July 2026)) is a product rating network. In ML and AN, edges represent user–item rating interactions, so vertex degrees count distinct rating partners. Table 3 presents the statistics of each bipartite graph, where $|U|$ and $|V|$ denote the numbers of vertices on the two sides of the graph, respectively, $|E|$ denotes the number of edges, and

δ denotes the largest value satisfying (δ, δ)-core $\neq \emptyset$. For DBLP, publication years, venues, and research fields are used as keywords (2230 distinct keywords; 3.62 per paper); ML uses movie genres as keywords (19; 1.68 per vertex); AN uses anime genres as keywords (57; 3.29); and each EP product is assigned a rating bucket and a category label (36; 2 per vertex).

Table 3. Statistics of used bipartite graphs.

Graphs	Category	$\lvert U \rvert$	$\lvert V \rvert$	$\lvert E \rvert$	δ
dblp (DBLP)	Authorship	315,950	168,511	542,405	6
MovieLens (ML)	Movie	6041	3953	1,000,209	221
Anime (AN)	Movie	73,515	11,197	7,813,727	351
Epinions (EP)	Product	120,492	755,760	13,668,320	151

Algorithms: We evaluate the query efficiency of the following algorithms, as well as the construction time and space consumption of the HI.

- ACS: the attributed community search method for bipartite graphs [24], included as an additional baseline.
- MCR: a baseline extended from MCR-Tree [8]. It first retrieves the structural (α, β)-community containing q, then removes keyword-incompatible vertices, applies cascading degree peeling, and extracts the remaining component containing q.
- BCN: a baseline extended from the Bi-core Number framework [7], followed by the same keyword filtering, degree peeling, and query-component extraction as MCR.
- Global: the baseline global search algorithm described in Algorithm 1.
- Local: the baseline local search algorithm described in Algorithm 2.
- HI + Global: the global search algorithm accelerated by the HI, as described in Algorithm 4.
- HI + Local: the local search algorithm accelerated by the HI, as described in Algorithm 4.

Experimental settings: We implement all the above algorithms. All algorithms are implemented in C++, and the experiments are conducted on a Windows Server equipped with a 2.10 GHz 16-core CPU and 256 GB memory. For each dataset, we randomly select ten query vertices and randomly choose query keywords from the keyword set associated with each selected vertex to generate query instances. Subsequently, we vary the target parameter while keeping other parameters fixed to investigate the impact of different parameter settings on algorithm performance. Specifically, we study the effects of α, β, the number of query keywords $|Q_K|$, and the dataset size $|E|$ on query efficiency. For ML and AN, Table 3 reports nonempty (δ, δ)-cores with $\delta = 221$ and 351, respectively; by core nesting, nonempty structural cores therefore exist in the full graphs for the tested thresholds 20–100. These thresholds impose minimum within-community rating counts on users and items. The common setting $\alpha = \beta = 100$ provides a fixed structural requirement when varying $|Q_K|$ or $|E|$, while the separate α and β experiments examine asymmetric requirements. Unless otherwise specified, all experiments are repeated five times, and the average running time is reported.

6.2. Query Performance

Under different parameter settings, we evaluate the performance of the algorithms by comparing their results across different datasets.

Exp-1: Varying α. Figures 3a–d evaluate the query performance by varying the value of α. For the sparse DBLP dataset, we fix $\beta = 5$ and test $\alpha = 1, 2, 3, 4, 5$; for the other datasets, we set $\beta = 100$ and vary $\alpha = 20, 40, 60, 80, 100$. The number of query keywords is

set to 3. As shown in the figures, HI-based algorithms achieve lower query times than their corresponding Global and Local baselines across all datasets, and also outperform ACS, MCR, and BCN on ML, AN, and EP. For the DBLP dataset, the query time of HI+Global and HI+Local generally decreases as α increases, since a larger core constraint reduces the candidate search space and avoids unnecessary graph traversal; however, BCN is faster at $\alpha = 3, 4, 5$. For the ML dataset, all algorithms show relatively stable performance, while the HI-based algorithms still achieve consistent improvements. For the EP dataset, the Local algorithm performs worse than the Global algorithm due to the large number of vertices traversed during local expansion. In contrast, the HI-based algorithms effectively filter irrelevant vertices through the index and achieve substantial performance gains. Overall, the results demonstrate that the proposed HI can consistently improve query efficiency under different α settings and across various datasets.

Exp-2: Varying β. Figures 4a–d evaluate the query performance by varying the value of β. For the sparse DBLP dataset, we fix $\alpha = 5$ and test $\beta = 1, 2, 3, 4, 5$; for the other datasets, we set $\alpha = 100$ and vary $\beta = 20, 40, 60, 80, 100$. The number of query keywords is set to 3. As shown in the figures, the performance trends with respect to β are similar to those observed in the α experiments. The query time of HI-based algorithms remains consistently lower than that of the corresponding Global and Local baselines, as well as ACS, MCR, and BCN on ML, AN, and EP. For the DBLP dataset, increasing β gradually reduces the query time of HI+Global and HI+Local, as a stricter core constraint narrows the search space, although BCN is faster at $\beta = 4, 5$. For the ML dataset, all algorithms show relatively stable performance, while HI-based algorithms still achieve noticeable improvements. For the EP dataset, the Local algorithm requires more query time than the Global algorithm due to extensive local expansion, whereas the HI-based algorithms effectively reduce the traversal cost through keyword and core-based filtering. Overall, the results further verify that the HI can consistently accelerate KBCS queries with different β values.

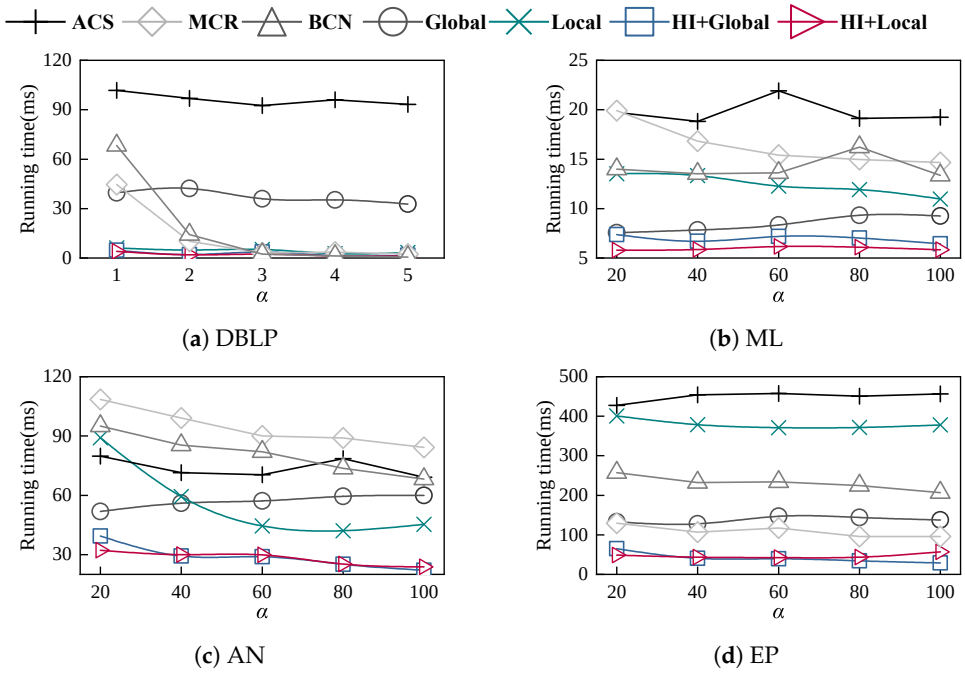


Figure 3. Effect of α on running time for KBCS.

Exp-3: Varying $|Q_K|$. Figures 5a–d evaluate the query performance by varying the value of $|Q_K|$. For all datasets, we vary $|Q_K|$ from 1 to 5. The values of α and β are fixed to 100 for all datasets except DBLP, where $\alpha = \beta = 5$. As shown in the figures, HI-based

algorithms outperform ACS, MCR, and BCN on ML, AN, and EP across the tested keyword-set sizes, while BCN is fastest in four of the five DBLP settings. In general, the query time of Local and HI+Local increases as $|Q_K|$ grows due to the additional keyword verification cost. The Global algorithm shows different trends across datasets, where the query time increases on DBLP and AN but remains relatively stable on ML and EP. This is because Global needs to remove vertices that do not satisfy the keyword constraints before core decomposition. With more query keywords, fewer vertices need to be pruned, which can reduce the filtering cost and offset the additional keyword checking overhead.

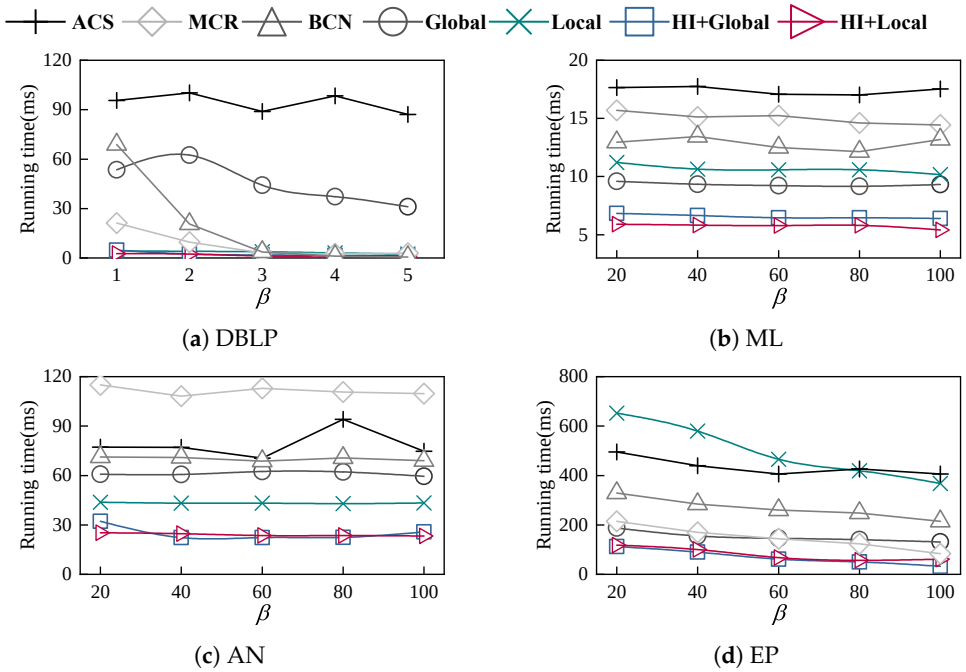


Figure 4. Effect of β on running time for KBCS.

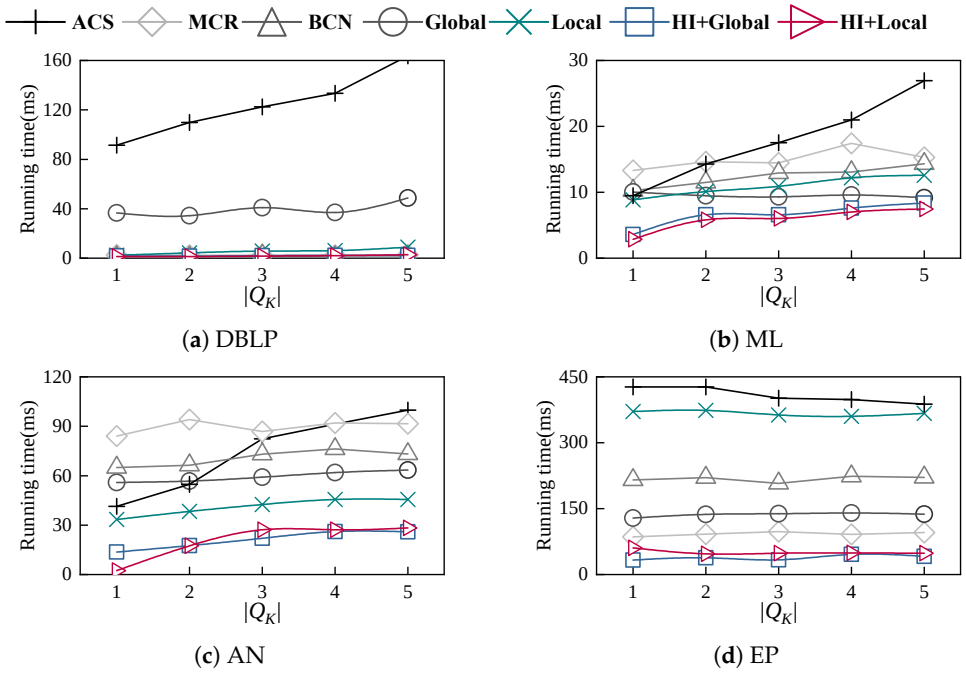
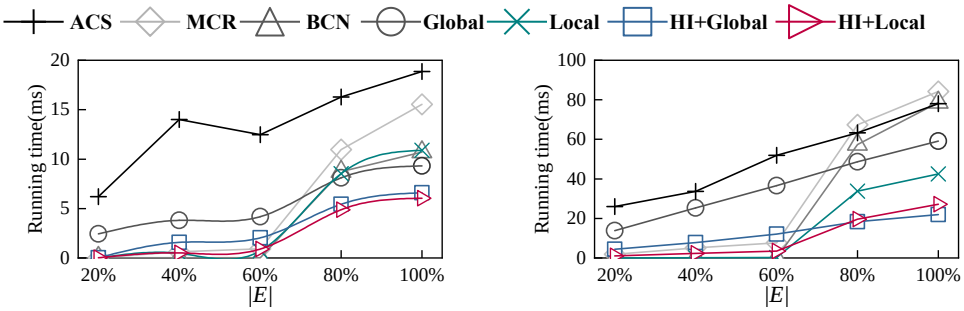


Figure 5. Effect of $|Q_K|$ on running time for KBCS.

Exp-4: Varying $|E|$. Figure 6a,b evaluate the query performance by varying the value of $|E|$. Specifically, we select the ML and AN datasets and construct subgraphs containing 20%,

40%, 60%, 80%, and 100% of the original edges. The values of α and β are fixed to 100, and the number of query keywords is set to 3. As shown in the figures, query times generally increase with graph size, since larger graphs introduce more vertices and edges that need to be processed. The increase is more significant for the baseline algorithms, especially Global and Local, which require extensive graph traversal and pruning operations on large graphs. At 80% and 100% of the edges, the HI-based algorithms also outperform ACS, MCR, and BCN, whereas BCN is faster on the smaller subgraphs with 20%–60% of the edges. These results demonstrate that the HI maintains good scalability as the graph size increases.



(**a**) ML (**b**) AN

Figure 6. Effect of $|E|$ on running time for KBCS.

6.3. HI Evaluation

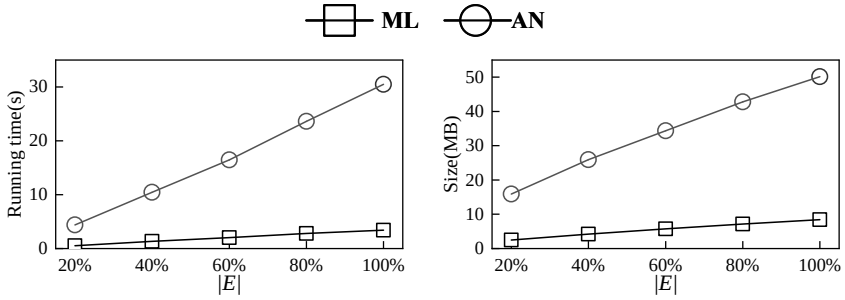
We evaluate the HI by measuring its construction time and space consumption on different datasets. Furthermore, we investigate the impact of dataset size on the construction efficiency and space overhead of the HI.

Exp-5: HI Construction and Space Overhead. Table 4 presents the construction time and space overhead of the HI on different datasets. As shown in the table, the construction time of HI varies across different datasets. Specifically, the HI can be constructed within 0.67 s, 3.4 s, 30.48 s, and 57.67 s on DBLP, ML, AN, and EP, respectively. Meanwhile, the corresponding space overheads are only 5.4 MB, 8.43 MB, 50.13 MB, and 492.197 MB. The construction time and space overhead increase on larger datasets, which is mainly because more vertices and keyword information need to be processed and stored during index construction. Overall, the HI requires limited construction cost and space overhead on different datasets.

Table 4. HI construction and space overhead.

Metrics	DBLP	ML	AN	EP
Construction (Second)	0.67	3.4	30.48	57.67
Space (MB)	5.4	8.43	50.13	492.197

Exp-6: Index Scalability with Varying $|E|$. To investigate the impact of graph size on the HI, we construct subgraphs with different numbers of edges by varying $|E|$ from 20% to 100% of the original datasets. The construction time and space overhead of HI on ML and AN are shown in Figure 7a,b. As the graph size increases, both the construction time and space overhead grow almost linearly. For the ML dataset, the construction time increases from 0.547 s to 3.4 s, while the space overhead increases from 2.49 MB to 8.43 MB. Similarly, for the AN dataset, the construction time increases from 4.39 s to 30.48 s, and the space overhead increases from 15.912 MB to 50.13 MB. This is because larger graphs contain more vertices and keyword information that need to be processed and stored during index construction. The results show that the construction cost and space overhead of HI increase smoothly with graph size.



(**a**) Construction time of index (**b**) Size of index

Figure 7. Effect of $|E|$ on index construction time and index size.

Exp-7: Effectiveness Comparison. We compare KBCS with ACS [24] on ML and AN ($\alpha = \beta = 100$, three query keywords). Community size is the total number of vertices, and feasible-vertex coverage is the fraction of lower-layer vertices retrieved from the maximal community containing the query vertex and satisfying the given structural and keyword constraints. Results are averaged over queries where both methods return nonempty communities. Table 5 shows that KBCS achieves 100% coverage, while ACS achieves 29.87% on ML and 26.39% on AN. ACS's additional attribute and weight optimization excludes vertices satisfying KBCS constraints, making KBCS more suitable for complete retrieval.

Table 5. Community size and feasible-vertex coverage.

	ML		AN	
Method	**Size**	**Coverage**	**Size**	**Coverage**
ACS	522.30	29.87%	1386.43	26.39%
KBCS	2166.10	100%	8212.00	100%

6.4. Case Study

This case study is conducted on an application–sensor bipartite network of interest (BNOI). The dataset (https://github.com/15514783351/RemoteSensing (accessed on 14 July 2026)) contains the relationships between sensors and application scenarios in the remote sensing domain. In this bipartite network, one type of vertex represents application scenarios, while the other type represents sensors. An edge indicates that a sensor has been used for a specific application scenario in the related literature. In addition, each sensor vertex is associated with a set of technical keywords describing its characteristics. We set the keyword query condition $QK = \{\text{SAR}\}$, which aims to retrieve sensors containing the Synthetic Aperture Radar (SAR) technology label and their associated applications. Meanwhile, the structural constraints are set as $\alpha = 2$ and $\beta = 4$, where α requires each application vertex to be connected to at least two sensor vertices, and β requires each sensor vertex to be connected to at least four application vertices.

The partial search results are shown in Figure 8. The discovered community contains four sensor vertices, seven application vertices, and 22 edges. This case study demonstrates that the proposed approach can effectively retrieve query-satisfying communities from complex sensor–application networks by integrating keyword constraints and (α, β)-core structural constraints. By specifying technical keywords, domain experts can intuitively analyze the relationships between sensors and application scenarios, providing support for exploring the associations between sensor technologies and application domains.

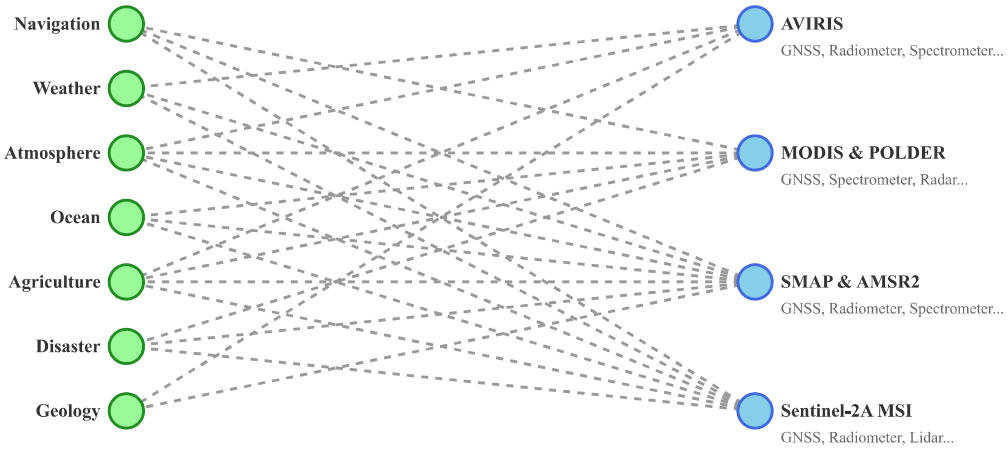


Figure 8. The displayed partial results, where ellipses indicate omitted keywords.

7. Conclusions

In this paper, we studied the keyword-constrained bi-community search (KBCS) problem over unweighted bipartite graphs, aiming to identify communities satisfying both structural and keyword constraints. To improve the efficiency of large-scale community search, we developed two algorithms based on global and local search paradigms and proposed a Hybrid Index (HI) that integrates keyword inverted lists with bi-core offset labels to reduce unnecessary exploration during query processing. Extensive experiments on four real-world bipartite graph datasets demonstrate that the proposed HI-based approach achieves significant improvements in query efficiency compared with the baseline algorithms. The proposed framework provides an effective graph analytics approach for retrieving cohesive and semantically meaningful entity groups in heterogeneous information networks, with potential applications in academic networks, recommendation systems, and social networks. In future work, we plan to investigate more flexible keyword matching strategies and extend the framework to other cohesive subgraph models.

Author Contributions: Conceptualization, J.J. and G.J.; methodology, J.J.; software, J.J.; validation, J.J. and X.S.; writing—original draft preparation, J.J.; writing—review and editing, X.S. and G.J.; supervision, G.J. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Key Project of Liaoning Provincial Social Science Planning Fund (Grant No. L24AKS004), "Research on the Formation and Governance of Cyberspace Cultural Ecology".

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The datasets used in this study are publicly available from their corresponding sources. The detailed descriptions, statistics, and access information of the datasets are provided in Section 6.1 (Experimental Setup). Our implementation is available at https://github.com/Juanjuan542/KBCS (accessed on 29 October 2026).

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Ntoutsi, E.; Stefanidis, K.; Nørvåg, K.; Kriegel, H.P. Fast group recommendations by applying user clustering. In *Proceedings of the International Conference on Conceptual Modeling*; Springer: Cham, Switzerland, 2012; pp. 126–140.
2. Ding, D.; Li, H.; Huang, Z.; Mamoulis, N. Efficient fault-tolerant group recommendation using alpha-beta-core. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*; ACM: New York, NY, USA, 2017; pp. 2047–2050.
3. Ley, M. The DBLP computer science bibliography: Evolution, research issues, perspectives. In *Proceedings of the International Symposium on String Processing and Information Retrieval*; Springer: Cham, Switzerland, 2002; pp. 1–10.

4. Beutel, A.; Xu, W.; Guruswami, V.; Palow, C.; Faloutsos, C. Copycatch: Stopping group attacks by spotting lockstep behavior in social networks. In *Proceedings of the 22nd international Conference on World Wide Web*; ACM: New York, NY, USA, 2013; pp. 119–130.
5. Zhang, Y.; Phillips, C.A.; Rogers, G.L.; Baker, E.J.; Chesler, E.J.; Langston, M.A. On finding bicliques in bipartite graphs: A novel algorithm and its application to the integration of diverse biological data types. *BMC Bioinform.* **2014**, *15*, 110.
6. Wang, K.; Zhang, W.; Lin, X.; Zhang, Y.; Li, S. Discovering hierarchy of bipartite graphs with cohesive subgraphs. In *Proceedings of the 2022 IEEE 38th International Conference on Data Engineering (ICDE)*; IEEE: New York, NY, USA, 2022; pp. 2291–2305.
7. Luo, W.; Yang, Q.; Fang, Y.; Zhou, X. Efficient Core Maintenance in Large Bipartite Graphs. *Proc. ACM Manag. Data* **2023**, *1*, 1–26.
8. Luo, C.; Zhu, Y.; Liu, Q.; Gao, Y.; Chen, L.; Xu, J. MCR-Tree: An Efficient Index for Multi-dimensional Core Search. *Proc. ACM Manag. Data* **2024**, *2*, 1–25.
9. Zong, C.; Meng, L.; Yang, X.; Wang, B.; Qiu, T.; Zhu, R. Zero-Redundancy Search for Bi-Components in Bipartite Graphs. *Proc. ACM Manag. Data* **2026**, *4*, 1–26.
10. Zou, Z. Bitruss decomposition of bipartite graphs. In *Proceedings of the International Conference on Database Systems for Advanced Applications*; Springer: Cham, Switzerland, 2016; pp. 218–233.
11. Sarıyüce, A.E.; Pinar, A. Peeling bipartite networks for dense subgraph discovery. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*; ACM: New York, NY, USA, 2018; pp. 504–512.
12. Wang, K.; Lin, X.; Qin, L.; Zhang, W.; Zhang, Y. Efficient bitruss decomposition for large-scale bipartite graphs. In *Proceedings of the 2020 IEEE 36th International Conference on Data Engineering (ICDE)*; IEEE: New York, NY, USA, 2020; pp. 661–672.
13. Linden, G.; Smith, B.; York, J. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet Comput.* **2003**, *7*, 76–80.
14. Wang, J.; Huang, P.; Zhao, H.; Zhang, Z.; Zhao, B.; Lee, D.L. Billion-scale commodity embedding for e-commerce recommendation in alibaba. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*; ACM: New York, NY, USA, 2018; pp. 839–848.
15. Xu, X.; Liu, J.; Wang, Y.; Ke, X. Academic Expert Finding via $(k, \mathcal{P})$-Core based Embedding over Heterogeneous Graphs. In *Proceedings of the 2022 IEEE 38th International Conference on Data Engineering (ICDE)*; IEEE: New York, NY, USA, 2022; pp. 338–351.
16. Lyu, B.; Qin, L.; Lin, X.; Zhang, Y.; Qian, Z.; Zhou, J. Maximum biclique search at billion scale. In *Proceedings of the VLDB Endowment*; ACM: New York, NY, USA, 2020.
17. Liu, B.; Yuan, L.; Lin, X.; Qin, L.; Zhang, W.; Zhou, J. Efficient (α, β)-core computation: An index-based approach. In *Proceedings of the The World Wide Web Conference*; ACM: New York, NY, USA, 2019; pp. 1130–1141.
18. Fang, Y.; Cheng, R.; Luo, S.; Hu, J. Effective community search for large attributed graphs. *Proc. VLDB Endow.* **2016**, *9*, 1233–1244.
19. Huang, X.; Lakshmanan, L.V.S. Attribute-driven community search. *Proc. VLDB Endow.* **2017**, *10*, 949–960.
20. Liu, Q.; Zhu, Y.; Zhao, M.; Huang, X.; Xu, J.; Gao, Y. VAC: Vertex-centric attributed community search. In *Proceedings of the 2020 IEEE 36th International Conference on Data Engineering (ICDE)*; IEEE: New York, NY, USA, 2020; pp. 937–948.
21. Wang, Y.; Ye, S.; Xu, X.; Geng, Y.; Zhao, Z.; Ke, X.; Wu, T. Scalable community search with accuracy guarantee on attributed graphs. In *Proceedings of the 2024 IEEE 40th International Conference on Data Engineering (ICDE)*; IEEE: New York, NY, USA, 2024.
22. Ye, J.; Zhu, Y.; Chen, L. Top-r keyword-based community search in attributed graphs. In *Proceedings of the 2023 IEEE 39th International Conference on Data Engineering (ICDE)*; IEEE: New York, NY, USA, 2023; pp. 1652–1664.
23. Wang, Y.; Peng, Z.; Ke, X.; Xu, X.; Wu, T.; Gao, Y. Cohesiveness-aware hierarchical compressed index for community search on attributed graphs. *Proc. ACM Manag. Data* **2025**, *3*, 1–27.
24. Li, D.; Liang, X.; Hu, R.; Zeng, L.; Wang, X. (α, β)-AWCS: (α, β)-Attributed Weighted Community Search on Bipartite Graphs. In *Proceedings of the 2022 International Joint Conference on Neural Networks (IJCNN)*; IEEE: New York, NY, USA, 2022; pp. 1–8.
25. Zhang, Z.; Li, H.; Huang, K.; Jiang, Y. k-bitruss-Attributed Weighted Community Search on Attributed Weighted Bipartite Graphs. In *Proceedings of the 2023 IEEE International Conference on Big Data (BigData)*; IEEE: New York, NY, USA, 2023; pp. 562–567.
26. Barbieri, N.; Bonchi, F.; Galimberti, E.; Gullo, F. Efficient and effective community search. *Data Min. Knowl. Discov.* **2015**, *29*, 1406–1433.
27. Batagelj, V.; Zaversnik, M. An O(m) algorithm for cores decomposition of networks. *arXiv* **2003**, arXiv:cs/0310049.
28. Bonchi, F.; Khan, A.; Severini, L. Distance-generalized core decomposition. In *Proceedings of the 2019 International Conference on Management of Data*; ACM: New York, NY, USA, 2019; pp. 1006–1023.
29. Cui, W.; Xiao, Y.; Wang, H.; Wang, W. Local search of communities in large graphs. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*; ACM: New York, NY, USA, 2014; pp. 991–1002.
30. Akbas, E.; Zhao, P. Truss-based community search: A truss-equivalence based indexing approach. *Proc. VLDB Endow.* **2017**, *10*, 1298–1309.
31. Huang, X.; Cheng, H.; Qin, L.; Tian, W.; Yu, J.X. Querying k-truss community in large and dynamic graphs. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*; ACM: New York, NY, USA, 2014; pp. 1311–1322.

32. Huang, X.; Lakshmanan, L.V.; Yu, J.X.; Cheng, H. Approximate closest community search in networks. *arXiv* **2015**, arXiv:1505.05956.
33. Cui, W.; Xiao, Y.; Wang, H.; Lu, Y.; Wang, W. Online search of overlapping communities. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*; ACM: New York, NY, USA, 2013; pp. 277–288.
34. Tsourakakis, C.E.; Bonchi, F.; Gionis, A.; Gullo, F.; Tsiarli, M.A. Denser than the densest subgraph: Extracting optimal quasi-cliques with quality guarantees. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*; ACM: New York, NY, USA, 2013; pp. 104–112.
35. Wang, Y.; Gu, C.; Xu, X.; Zeng, X.; Ke, X.; Wu, T. Efficient and effective (k, p)-core-based community search over attributed heterogeneous information networks. *Inf. Sci.* **2024**, *661*, 120076.
36. Zhang, Z.; Huang, X.; Xu, J.; Choi, B.; Shang, Z. Keyword-centric community search. In *Proceedings of the 2019 IEEE 35th International Conference on Data Engineering (ICDE)*; IEEE: New York, NY, USA, 2019; pp. 422–433.
37. Yuan, L.; Qin, L.; Zhang, W.; et al.. Index-based densest clique percolation community search in networks. *IEEE Trans. Knowl. Data Eng.* **2017**, *30*, 922–935.
38. Wang, K.; Lin, X.; Qin, L.; Zhang, W.; Zhang, Y. Vertex Priority Based Butterfly Counting for Large-scale Bipartite Networks. *Proc. VLDB Endow.* **2019**, *12*, 1139–1152.
39. Zhang, Y.; Li, R.H.; Zhang, Q.; Qin, H.; Qin, L.; Wang, G. Density Decomposition of Bipartite Graphs. *Proc. ACM Manag. Data* **2025**, *3*, 1–25.
40. Wang, K.; Zhang, W.; Lin, X.; Zhang, Y.; Qin, L.; Zhang, Y. Efficient and effective community search on large-scale bipartite graphs. In *Proceedings of the 2021 IEEE 37th International Conference on Data Engineering (ICDE)*; IEEE: New York, NY, USA, 2021; pp. 85–96.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.