Identifying Similar-Bicliques in Bipartite Graphs

Kai Yao
The University of Sydney
Sydney, Australia
kyao8420@uni.sydney.edu.au

Lijun Chang
The University of Sydney
Sydney, Australia
Lijun.Chang@sydney.edu.au

Jeffrey Xu Yu
The Chinese University of Hong Kong
Hong Kong, China
yu@se.cuhk.edu.hk

ABSTRACT

Bipartite graphs have been widely used to model the relationship between entities of different types, where vertices are partitioned into two disjoint sets/sides. Finding dense subgraphs in a bipartite graph is of great significance and encompasses many applications. However, none of the existing dense bipartite subgraph models consider similarity between vertices from the same side, and as a result, the identified results may include vertices that are not similar to each other. In this paper, we formulate the notion of similar-biclique which is a special kind of biclique where all vertices from a designated side are similar to each other, and aim to enumerate all similar-bicliques. The naive approach of first enumerating all maximal bicliques and then extracting all maximal similar-bicliques from them is inefficient, as enumerating maximal bicliques is time consuming. We propose a backtracking algorithm MSBE to directly enumerate maximal similar-bicliques, and power it by vertex reduction and optimization techniques. Furthermore, we design a novel index structure to speed up a time-critical operation of MSBE, as well as to speed up vertex reduction. Efficient index construction algorithms are also developed. Extensive experiments on 17 bipartite graphs as well as case studies are conducted to demonstrate the effectiveness and efficiency of our model and algorithms.

PVLDB Reference Format:
Kai Yao, Lijun Chang, and Jeffrey Xu Yu. Identifying Similar-Bicliques in Bipartite Graphs. PVLDB, 15(11): 3085 - 3097, 2022.
doi:10.14778/3551793.3551854

1 INTRODUCTION

Bipartite graphs have been widely used in real-world applications to model relationships between entities of different types, such as customer-product networks [49], author-paper networks [28] and user-event networks [14]. A bipartite graph is denoted by $G = (V_L, V_R, E)$, where the vertex set is partitioned into two disjoint subsets V_L and V_R which are referred to as the L-side and R-side vertices of the bipartite graph, respectively; each edge $e \in E$ can only connect vertices from different sides. Finding dense subgraphs in a bipartite graph is of great significance and encompasses many applications, such as community detection [2, 26], anomaly detection [17, 43], and group recommendation [36, 45].

One classic notion of dense bipartite subgraph is *biclique* [40], which requires every pair of vertices from different sides of the subgraph to be connected by an edge. For example, for the bipartite graph in Figure 1 which represents researchers publishing in conference venues, the subgraph in the shadowed area is a biclique. In the literature, many algorithms have been proposed to enumerate all maximal bicliques [2, 4, 15, 30, 33, 48, 54] and to identify a biclique of the maximum size [36]. However, the biclique model has a fundamental limitation: vertices in a biclique are not necessarily similar to each other, despite that they share a set of common neighbors (e.g., vertices on the other side of the biclique). Consider the six researchers in the biclique in Figure 1, all of which publish in database conferences. Besides, researchers r_1, r_2, r_3 also publish in machine learning (ML) conferences, while r_4, r_5, r_6 publish in high-performance computing (HPC) conferences. Thus, the two groups of researchers, $\{r_1, r_2, r_3\}$ and $\{r_4, r_5, r_6\}$, are likely from different backgrounds and communities, i.e., ML vs. HPC.

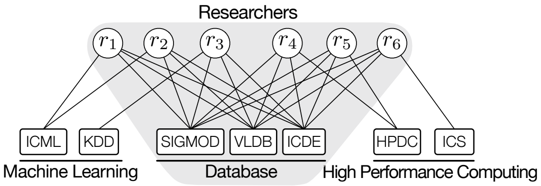


Figure 1: Example of researcher-venue bipartite graph

Motivated by this, we formulate the notion of *similar-biclique* by requiring all vertices from one side of the biclique to be similar to each other. Our empirical studies show that similar-bicliques can be detected much more efficiently than bicliques. Thus, identifying similar-bicliques is useful for the following applications.

- **Community detection.** Similar-biclique satisfies all the key requirements of community structure for bipartite graphs [26, 50, 55], and thus can be used to detect communities in interaction-type bipartite graphs such as user-rate-movie, customer-buy-product, and author-write-paper. Firstly, being a biclique, interactions between vertices from the two sides are intensive. Secondly, by enforcing the similarity constraint, users in a similar-biclique are similar to each other, e.g., having similar behaviours or interests.
- **Anomaly detection.** Similar-biclique can also be used for anomaly detection, which is a common task in e-commerce [5, 13, 36]. Here, the transactions of customers purchasing products form a customer-product bipartite graph. To improve the ranking of certain products, fraudsters may create fake accounts to purchase the products, i.e., click farming [10]. These fake accounts and the products they promote inevitably form a closely connected group, and meanwhile, these fake accounts will display a high level of synchronized behavior with each other [22]. Thus, suspicious groups (i.e., both the fraudulent accounts and the products they promote) can be captured by similar-bicliques.

Proceedings of the VLDB Endowment, Vol. 15, No. 11 ISSN 2150-8097.
doi:10.14778/3551793.3551854

Formally speaking, given a similarity threshold $0 < \varepsilon \leq 1$ and a size constraint $\tau \geq 0$, a vertex subset $C \subseteq V_L \cup V_R$ in a bipartite graph $G = (V_L, V_R, E)$ is a *similar-biclique* if (1) C is a biclique (i.e., $C_L \times C_R \subseteq E$), (2) all vertices of C_L are similar to each other (i.e., $\mathsf{sim}(u, v) \geq \varepsilon, \forall u, v \in C_L$), and (3) C satisfies the size constraint (i.e., $|C_L| \geq \tau$ and $|C_R| \geq \tau$). Here, C_L denotes $C \cap V_L$ and C_R denotes $C \cap V_R$; $\mathsf{sim}(u, v)$ measures the structural similarity between u and v, which is computed based on their neighbors $N(u)$ and $N(v)$ and will be formally defined in Section 2; the size constraint τ is introduced to avoid generating too small or too skewed results. Note that, we only apply the similarity constraint to one side of the vertices (either V_L or V_R as they are interchangeable), since in applications we are usually only interested in the similarity between "users". Nevertheless, the general technical ideas presented in this paper can also be applied to the variant of similar-biclique that imposes the similarity constraint on both sides of vertices.

We in this paper aim to enumerate all *maximal* similar-bicliques in a bipartite graph. We prove that this problem is #P-complete. As each (maximal) similar-biclique is contained in a maximal biclique, we could first enumerate all maximal bicliques, then extract maximal similar-bicliques from maximal bicliques, and finally eliminate all similar-bicliques that are either duplicates or not maximal. However, this approach is inefficient, as enumerating all maximal bicliques by the state-of-the-art algorithm PMBE [2] is already time consuming for large graphs. Thus, we propose the MSBE algorithm to directly enumerate maximal similar-bicliques, without first enumerating maximal bicliques.

MSBE follows the general backtracking framework of the Bron-Kerbosch algorithm [6] that enumerates all maximal *cliques* in a unipartite graph. Our observation is that once the set of L-side vertices C_L of a similar-biclique C is determined, its R-side vertices can be simply obtained as $C_R = \bigcap_{u \in C_L} N(u)$. Nevertheless, it is worth pointing out that we cannot ignore C_R during the enumeration process, since (1) the size of C_R will be used for pruning and (2) both C_L and C_R are needed for determining the maximality of the similar-biclique. MSBE iteratively builds up a partial solution $\langle C_L, C_R \rangle$, maintains a candidate set P_L that is used to grow C_L, and maintains an exclusive set Q_L that is used for checking the maximality of $\langle C_L, C_R \rangle$. In each recursion, a vertex $u \in P_L$ is added to C_L to grow the solution; after coming back from the recursion, u is moved from P_L to Q_L to avoid duplicates. To prune the search space, we propose the concept of *dominating*: $u \in P_L$ dominates $v \in P_L$ if $\mathsf{sim}(u, v) \geq \varepsilon$ and $N_{C_R}(u) \supseteq N_{C_R}(v)$, where $N_{C_R}(u) = N(u) \cap C_R$. We prove that if u dominates v, then we can prune the recursion of adding v to C_L when u is moved from P_L to Q_L. Furthermore, according to the definition, each vertex u in a similar-biclique C must have at least τ neighbors in C (i.e., $|N_C(u)| \geq \tau$), and each L-side vertex $u \in C_L$ must have at least $\tau - 1$ vertices that are similar to it; we call the vertices that are similar to u the *similar neighbors* of u, denoted $\Gamma(u)$. Thus, we propose to first prune all vertices that violate either of these two conditions, in a preprocess referred to as *vertex reduction*; our empirical studies show that a large portion of the input graph is pruned by vertex reduction.

We observe that a time-critical operation, in both vertex reduction and the recursion of backtracking, is computing $\Gamma(u)$ which would take $O(\sum_{v \in N(u)} |N(v)|)$ time, for a vertex $u \in V_L$. Note that, $\Gamma(u)$ is not stored in the graph representation, and it is also not affordable to store $\Gamma(u)$ (either in main memory or on disk) after it is computed as this would require a prohibitively large space. For example, it would take over 400GB on Bibsonomy, one of the graphs tested in our experiments. In view of this, we propose an offline-constructed index to speed up the computation of $\Gamma(u)$; note that, our index can be used to process maximal similar-biclique enumeration queries with different ε and τ values. This is based on the fact that for any similarity threshold ε, $\Gamma(u)$ is always a subset of $\Phi_u = \bigcup_{v \in N(u)} N(v)$, the 2-hop structural neighbors of u. Thus, we propose to first compute the similarity between u and every vertex of Φ_u in an offline process, and then compress them into a few segments which are stored in the index. Specifically, each segment is represented by $\mathsf{seg} = \langle \mathsf{v}_{\min}, \mathsf{v}_{\max}, \mathsf{s}_{\max}, c \rangle$ where $\mathsf{v}_{\min} \leq \mathsf{v}_{\max}$ are two vertices of Φ_u, $\mathsf{s}_{\max}$ is the largest similarity between u and vertices of Φ_u that are in the range $[\mathsf{v}_{\min}, \mathsf{v}_{\max}]$, and c is the number of vertices of Φ_u that are in $[\mathsf{v}_{\min}, \mathsf{v}_{\max}]$; here, the comparison between vertices is based on their ids. To obtain $\Gamma(u)$, we go through each segment seg of Φ_u that satisfies $\mathsf{seg.s}_{\max} \geq \varepsilon$, and compute the similarity between u and each $v \in [\mathsf{seg.v}_{\min}, \mathsf{seg.v}_{\max}]$; note that, segments with $\mathsf{s}_{\max} < \varepsilon$ are skipped. Furthermore, we also use the index to speed up vertex reduction by first pruning vertices based on upper bounds of $\Gamma(u)$, which can be efficiently obtained based on the index without computing similarities.

Our main contributions are summarized as follows:

- We formulate the concept of similar-biclique, which can be used to detect interesting dense subgraphs from a bipartite graph. To the best of our knowledge, this is the first work investigating structural similarity between vertices in dense bipartite subgraph mining.
- We develop a backtracking algorithm MSBE to enumerate all maximal similar-bicliques in a bipartite graph. Vertex reduction and optimization techniques are also proposed.
- We design a novel index structure to facilitate the computation of similar neighbors. We propose a two-phase algorithm for efficient vertex reduction based on the index.
- We propose effective and efficient index construction algorithms by investigating two different strategies.

Extensive empirical studies on 17 real bipartite graphs as well as case studies are conducted to demonstrate the efficiency of our algorithms and the effectiveness of our similar-biclique model. Our algorithm is up to 6 orders of magnitude faster than PMBE.

Organization. Related works are reviewed in below. Section 2 provides preliminaries including the definition of similar-biclique and the problem statement. Section 3 introduces a baseline algorithm and our MSBE algorithm. Section 4 presents our index structure, index-based algorithms and index construction algorithms. Section 5 reports experimental results. Section 6 concludes the paper.

Related Works. We categorize the related works as follows.

(1) Maximal biclique enumeration. The problem of enumerating all maximal bicliques has been widely studied. The existing studies can be classified into two categories, depending on whether the input graph is bipartite or not. When the input graph is bipartite, the existing algorithms [2, 42, 54] generally enumerate subsets of vertices from one side, and then the intersection of their neighbors form the other side of the biclique. Besides, frequent item-set

mining techniques have also been utilized to enumerate maximal bicliques [29, 30, 48, 53], as these two problems are highly related to each other. The state-of-the-art algorithm for maximal biclique enumeration over bipartite graphs is PMBE proposed in [2], which is compared in our experiments. There are also studies that aim at enumerating all maximal (non-induced) bicliques from a general graph, i.e., the input graph is not bipartite. For example, it is studied from a theoretical viewpoint in [15], consensus algorithms are proposed in [4], and a divide-and-conquer algorithm is proposed in [33]. However, these algorithms are generally slower than the algorithms that specifically handle bipartite graphs. Moreover, none of the existing studies on maximal biclique enumeration take into consideration the structural similarity between vertices.

(2) Maximal clique enumeration. The problem of enumerating all maximal cliques in a unipartite graph has also been extensively studied. The existing algorithms generally follow the backtracking framework of Bron and Kerbosch [6] with optimization techniques being proposed in [8, 9, 16, 37, 46]. However, these algorithms cannot handle bipartite graphs.

(3) Dense bipartite subgraph identification. Besides biclique, other models have also been proposed for dense bipartite subgraph identification, such as quasi-biclique [34], k-biplex [52], (α, β)-core [24], k-bitruss [57], and k-wing [43]. Quasi-biclique and k-biplex relax the biclique model by allowing each vertex in one side of the result to miss a certain number of neighbors from the other side. On the other hand, (α, β)-core requires each vertex from one side to be connected to a certain number of vertices from the other side, and k-bitruss and k-wing require each edges to be involved in a certain number of $(2, 2)$-bicliques. None of these models consider similarity between vertices, and our case study in Section 5.2 demonstrates that similar-biclique outperforms k-biplex and (α, β)-core in anomaly detection. Note that, it is also possible to integrate similarity constraint into these dense bipartite subgraph models, in a similar way to similar-bicliques. However, this would require thorough studies, from problem hardness analysis to algorithm design, for each model. For example, although (α, β)-core can be computed in polynomial time, computing *similar*-(α, β)-cores may require an exponential time as the number of maximal *similar*-(α, β)-cores could be exponential. Thus, we leave these to our future studies.

(4) Structural vertex similarity. Structural vertex similarity refers to similarity measures between pairs of vertices that are computed based on solely the topology of the graph [27]. They are usually categorized as *local-topology based* and *global-topology based.* For the former, the similarity between two vertices is computed based on their neighbors, i.e., locally. Examples include Jaccard similarity [20], cosine similarity [41], Dice's coefficient [12], hub depressed/promoted index [35] and Adamic–Adar index [3]. For the latter, the global structure information is utilized to derive the similarity between two vertices. Examples include Katz [23], SimRank [21], C-Rank [51], P-Rank [56], and MatchSim [32]. As global-topology-based measures usually need to access the entire graph to compute the similarity, they are computationally more expensive than the local-topology-based measures. Thus, we adopt local-topology-based measures in this paper.

2 PRELIMINARY

We consider an unweighted and undirected bipartite graph $G = (V_L, V_R, E)$, where V_L and V_R denote the two disjoint vertex sets (i.e., L-side vertices and R-side vertices) and $E \subseteq V_L \times V_R$ denotes the edge set. Without loss of generality, we assume that vertices of V_L take (integer) ids from $\{1, 2, \ldots, |V_L|\}$, and vertices of V_R take ids from $\{1 + |V_L|, 2 + |V_L|, \ldots, |V_R| + |V_L|\}$. For any vertex $v \in V_L$ (resp. $v \in V_R$), we say it is an L-side vertex (resp. R-side vertex). For any vertex subset $C \subseteq V_L \cup V_R$, we use C_L and C_R to denote the subsets of vertices of C that are from V_L and V_R, respectively, i.e., $C_L = C \cap V_L$ and $C_R = C \cap V_R$. We call the set of neighbors of u in G, denoted $N_G(u) = \{v \mid (u, v) \in E\}$, the *structural neighbors* of u. Denote $d_G(u) = |N_G(u)|$ the *structural degree* of u. Besides structural neighbor and structural degree, we will also define similar neighbor and similar degree based on structural similarity.

Definition 1 (Structural Similarity)**.** Given two vertices u and v in G, their structural similarity is defined as $\mathsf{sim}(u, v) = \frac{|N_G(u) \cap N_G(v)|}{|N_G(u) \cup N_G(v)|}$.

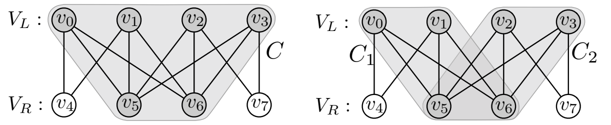
The structural similarity $\mathsf{sim}(u, v)$ is between 0 and 1. It measures the Jaccard similarity between the set of structural neighbors of u and that of v. Given a similarity threshold $\varepsilon > 0$, we say u and v are similar if $\mathsf{sim}(u, v) \geq \varepsilon$. The set of vertices that are similar to u is called the *similar neighbors* of u, denoted $\Gamma_{G,\varepsilon}(u)$, i.e., $\Gamma_{G,\varepsilon}(u) = \{v \in V_L \cup V_R \mid \mathsf{sim}(u, v) \geq \varepsilon\}$. Accordingly, denote $\delta_{G,\varepsilon}(u) = |\Gamma_{G,\varepsilon}(u)|$ the *similar degree* of u. Note that, as the structural similarity between vertices from different sides is always 0, similar neighbors only contain vertices from the same side. For presentation simplicity, we call structural similarity simply as *similarity*, and omit the subscripts G and/or ε from the notations.

Definition 2 (Similar-Biclique)**.** Given a bipartite graph $G = (V_L, V_R, E)$ and a similarity threshold $\varepsilon > 0$, a vertex subset $C \subseteq V_L \cup V_R$ is a similar-biclique if

- C is a *biclique*, i.e., $C_L \times C_R \subseteq E$, and
- all vertices from the L-side are similar to each other, i.e., $\mathsf{sim}(u, v) \geq \varepsilon, \forall u, v \in C_L$.

We also denote C as $\langle C_L, C_R \rangle$. A similar-biclique is *maximal* if it is not a subset of any larger similar-biclique.

Note that for presentation simplicity, the similarity constraint is assumed to be considered for the L-side vertices. To apply the similarity constraint for R-side vertices in applications, we can simply swap the roles of V_L and V_R in G.



(a) A maximal biclique (b) Two maximal similar-bicliques

Figure 2: Maximal biclique vs. maximal similar-biclique

Example 1. *Figure 2(a) shows a bipartite graph G with $V_L = \{v_0, \ldots, v_3\}$ and $V_R = \{v_4, \ldots, v_7\}$, in which the subgraph C in the gray area is a maximal biclique. However, v_1 and v_2 in the L-side of C are not similar to each other for $\varepsilon = 0.6$, since $\mathsf{sim}(v_1, v_2) = 0.5$. Subgraphs C_1 and C_2 as shown in Figure 2(b) are two maximal similar-bicliques, in each of which all vertices on the L-side are similar to each other.*

Problem 1 (Maximal Similar-Biclique Enumeration)**.** Given a bipartite graph $G = (V_L, V_R, E)$, a similarity threshold $\varepsilon > 0$ and a size constraint $\tau > 0$, we study the problem of enumerating all maximal similar-bicliques C in G that satisfy the size constraint τ (i.e., $|C_L| \geq \tau$ and $|C_R| \geq \tau$).

The size constraint τ is adopted here to avoid generating too small or too skewed similar-bicliques (i.e., with very few or no vertices in one side). For presentation simplicity, we assume the same size constraint τ for both sides. Note that, the techniques that we are going to present in this paper can be straightforwardly extended to handle different size constraints on the two sides.

Theorem 1. *The problem of enumerating all maximal similar-bicliques is #P-complete.*

The proof is omitted and can be found in the full version [1].

Remark about Strctural Similarity. In Definition 1, we use the Jaccard similarity to measure the structural similarity since it has been widely used and shown great success in many graph analysis tasks, such as structural graph clustering [7, 47], link prediction [31, 38], and local graph sparsification [44]. Nevertheless, other local-topology-based similarity measures as reviwed at the end of Introduction, such as cosine similarity: $\frac{|N_G(u) \cap N_G(v)|}{\sqrt{d_G(u) \times d_G(v)}}$, hub promoted index: $\frac{|N_G(u) \cap N_G(v)|}{\min\{d_G(u), d_G(v)\}}$, can be easily plugged into our model and algorithms. We will point out the changes that need to be made to adopt these measures, when presenting our algorithms.

3 OUR ALGORITHMS

In this section, we propose an MSBE algorithm to enumerate all maximal similar-bicliques. Before that, we first in Section 3.1 present a baseline algorithm based on the existing maximal biclique enumeration algorithms.

3.1 A Baseline Algorithm

It is easy to observe that maximal similar-bicliques must be contained in maximal bicliques, since every similar-biclique is also a biclique. Thus, a naive approach is first enumerating all maximal bicliques by invoking one of the existing maximal biclique enumeration algorithms, and then post-process the detected maximal bicliques to obtain maximal similar-bicliques. Specifically, for each maximal biclique, we extract maximal similar-bicliques by imposing the similarity constraint on L-side vertices, and then eliminate all similar-bicliques that are either duplicates or non-maximal. We omit the details, since our empirical study in Section 5 shows that enumerating all maximal bicliques by the state-of-the-art algorithm PMBE [2] is already time consuming for large graphs.

3.2 Our MSBE Algorithm

According to the definition of similar-biclique, if we build a similarity graph G_s for V_L where two vertices of V_L are connected by an edge if their similarity is at least ε, then for every (maximal) similar-biclique C, its L-side vertices C_L form a clique in G_s. Moreover, once the L-side vertices C_L of a maximal similar-biclique are determined, the R-side vertices C_R can be easily obtained as the set of common neighbors of C_L, i.e., $C_R = \bigcap_{u \in C_L} N(u)$. In view of this, we propose to adopt the general backtracking framework of the Bron-Kerbosch algorithm [6] to enumerate all maximal similar-bicliques. However, there are two issues. (1) The similarity graph G_s is not available in the input. (2) The set of L-side vertices C_L of a maximal similar-biclique C is not necessarily a *maximal* clique in G_s, though C_L is a clique in G_s. This is because, a too large C_L may result in a too small C_R, violating the size constraint τ on C_R.

Algorithm 1: MSBE$(G = (V_L, V_R, E), \varepsilon, \tau)$

```
1  for each $u \in V_L \cup V_R$ do $\mathsf{del}(u)$ = false;
2  VReduce$(G, \varepsilon, \tau, \mathsf{del}(\cdot))$;
3  for each $u \in V_L$ s.t. $\mathsf{del}(u)$ = false do
4      $C_L \leftarrow \{u\}; C_R \leftarrow \{v \in N(u) \mid \mathsf{del}(v) = \textbf{false}\}$;
5      $P_L \leftarrow \emptyset; Q_L \leftarrow \emptyset$;
6      Obtain $\Gamma(u)$;
7      for each $v \in \Gamma(u)$ do
8          if $v > u$ then $P_L \leftarrow P_L \cup \{v\}$ ;
9          else $Q_L \leftarrow Q_L \cup \{v\}$;
10     Enum$(C_L, C_R, P_L, Q_L)$;

   Procedure Enum$(C_L, C_R, P_L, Q_L)$
11 if $\nexists u \in P_L \cup Q_L$ s.t. $N(u) \supseteq C_R$ then
12     if $|C_L| \geq \tau$ and $|C_R| \geq \tau$ then output $\langle C_L, C_R \rangle$ ;
13 for each $u \in P_L$ do
14     $C'_L \leftarrow C_L \cup \{u\}; C'_R \leftarrow C_R \cap N(u)$;
15     Obtain $\Gamma(u)$;
16     $P'_L \leftarrow P_L \cap \Gamma(u); Q'_L \leftarrow Q_L \cap \Gamma(u)$;
17     if $|C'_L| + |P'_L| \geq \tau$ and $|C'_R| \geq \tau$ then Enum$(C'_L, C'_R, P'_L, Q'_L)$ ;
18     $P_L \leftarrow P_L \setminus \{u\}; Q_L \leftarrow Q_L \cup \{u\}$;
```

We propose techniques to address the above issues, and the pseudocode of our algorithm is shown in Algorithm 1, denoted MSBE. We first prune unpromising vertices by invoking VReduce (Lines 1–2), which will be introduced shortly in Algorithm 2; a vertex u is pruned if $\mathsf{del}(u)$ = **true**. For each remaining vertex $u \in V_L$ with $\mathsf{del}(u)$ = **false**, we enumerate all maximal similar-bicliques containing u (Lines 3–10). To do so, we iteratively grow a partial solution $\langle C_L, C_R \rangle$ where C_L is initialized as $\{u\}$ (Line 4). Besides C_L and C_R, we also maintain P_L and Q_L, in a similar fashion to the Bron-Kerbosch algorithm [6]. Specifically, P_L is a set of candidate vertices that is used to grow C_L, and Q_L is a set of previously considered candidate vertices that is used for checking the maximality of $\langle C_L, C_R \rangle$; note that, each vertex of $P_L \cup Q_L$ must be similar to all vertices of C_L. P_L and Q_L are initialized at Lines 5–9, after which we invoke the procedure Enum for enumeration (Line 10); to avoid duplicates, the similar neighbors of u with larger vertex id than u are put in P_L, and those with smaller vertex id are put in Q_L.

In Enum, if the current similar-biclique $\langle C_L, C_R \rangle$ is maximal, we report it (Lines 11–12); note that $\langle C_L, C_R \rangle$ is maximal if and only if there is no vertex $u \in P_L \cup Q_L$ such that $N(u) \supseteq C_R$. Next, Enum iteratively adds a vertex of P_L to C_L, updates the corresponding C_R, P_L and Q_L, and recursively invokes itself to enumerate more similar-bicliques (Lines 14–17). After processing $u \in P_L$, we remove u from P_L and add it to Q_L (Line 18).

Vertex Reduction. As a similar-biclique needs to have at least τ vertices on each side, each vertex u in a similar-biclique C must have at least τ structural neighbors in C (i.e., $|N_C(u)| \geq \tau$). Furthermore, as all L-side vertices in a similar-biclique C are similar to each other, each L-side vertex u must have at least $\tau - 1$ similar neighbors in C (i.e., $|\Gamma_C(u)| \geq \tau - 1$). As a result, we can exclude all vertices that violate either of these two conditions from being considered in the

Algorithm 2: VReduce$(G = (V_L, V_R, E), \varepsilon, \tau, \mathsf{del}(\cdot))$

```
1 for each $u \in V_L \cup V_R$ do $d(u) \leftarrow |N(u)|$;
2 for each $u \in V_L$ do Obtain $\Gamma(u)$ and set $\delta(u) \leftarrow |\Gamma(u)|$;
3 while $(\exists u \in V_L \cup V_R$ s.t. $\mathsf{del}(u) =$ false and $d(u) < \tau)$ or $(\exists u \in V_L$ s.t. $\mathsf{del}(u) =$ false and $\delta(u) < \tau - 1)$ do
4     for each $v \in N(u)$ do $d(v) \leftarrow d(v) - 1$;
5     if $u \in V_L$ then
6         Obtain $\Gamma(u)$;
7         for each $v \in \Gamma(u)$ do $\delta(v) \leftarrow \delta(v) - 1$;
8     $\mathsf{del}(u) =$ true;
```

Algorithm 3: SimNei$(G = (V_L, V_R, E), u, \varepsilon, \mathsf{del}(\cdot))$

```
  Output: $\Gamma(u)$
1 $\Gamma(u) \leftarrow \emptyset$;
2 for each $v \in V_L$ do $c(v) \leftarrow 0$;
3 for each $v \in N(u)$ do
4     for each $w \in N(v)$ and $w \neq u$ do $c(w) \leftarrow c(w) + 1$;
5 for each $v \in V_L$ s.t. $c(v) \neq 0$ and $\mathsf{del}(v) =$ false do
6     if $\frac{c(v)}{d(u)+d(v)-c(v)} \geq \varepsilon$ then $\Gamma(u) \leftarrow \Gamma(u) \cup \{v\}$;
7 return $\Gamma(u)$;
```

enumeration procedure Enum, i.e., mark them as deleted; we call this process as *vertex reduction.*

We propose an algorithm VReduce to conduct vertex reduction, whose pseudocode is shown in Algorithm 2. Firstly, we obtain the structural degree $d(u)$ for each vertex $u \in V_L \cup V_R$ (Line 1), and obtain the similar degree $\delta(u)$ for each L-side vertex $u \in V_L$ (Line 2). Then, as long as there is a non-deleted vertex u with $d(u) < \tau$ or a non-deleted L-side vertex u with $\delta(u) < \tau - 1$, we mark u as deleted (Line 8), decrease the structure degree of u's structural neighbors by 1 (Line 4), and decrease the similar degree of u's similar neighbors by 1 if u is an L-side vertex (Lines 5–7).

Compute Similar Neighbors $\Gamma(u)$**.** One fundamental operation in both Algorithm 1 and Algorithm 2 is computing $\Gamma(u)$ for an L-side vertex u; note that $\Gamma(u)$ is not stored with the graph G. A straightforward method to collect $\Gamma(u)$ is computing $\mathsf{sim}(u, v)$ for each vertex $v \in V_L$. The time complexity would be $O(|E|)$, as it needs to visit every edge of G. This is inefficient, by noting that Algorithm 1 and Algorithm 2 need to compute $\Gamma(u)$ for many vertices u and for multiple times.

We propose a more efficient algorithm in Algorithm 3, denoted SimNei. Instead of blindly computing $\mathsf{sim}(u, v)$ for each $v \in V_L$, we only compute $\mathsf{sim}(u, v)$ for those v with $\mathsf{sim}(u, v) > 0$. Our main idea is to first compute the number of common neighbors $c(v)$ between u and v for each 2-hop structural neighbor v of u (Lines 3–4). Then, $\mathsf{sim}(u, v)$ can be calculated as $\frac{c(v)}{d(u)+d(v)-c(v)}$ (Line 6).[1] Note that, in our implementation, to make the time complexity of SimNei to be independent of $|V_L|$ which may be large, we only initialize $c(\cdot)$ once at the beginning of the entire algorithm execution (e.g., in MSBE), and after using $c(\cdot)$ at Line 4–6 of Algorithm 3 we reset those updated $c(\cdot)$ to be 0. In addition, we also collect at Line 4 the set of vertices with non-zero $c(\cdot)$ into a set S, such that Line 5 as well as the resetting of $c(\cdot)$ can be conducted in $O(|S|)$ time. As a result, the time complexity of SimNei is $O(\sum_{v \in N(u)} d(v))$, which is lower than $O(|E|)$.

Optimizations for Enum**.** We further propose two optimization techniques to speed up the Enum procedure. Recall that, an instance of Enum is represented by (C_L, C_R, P_L, Q_L) where $C_R = \bigcap_{u \in C_L} N(u)$ and $|C_R| \geq \tau$,[2] and aims to enumerate all maximal similar-bicliques C^* satisfying $C_L \subset C^*_L \subseteq C_L \cup P_L$. Firstly, an enumeration instance can be terminated once we know that it will not generate any maximal similar-biclique. That is, by including any subset of vertices of P_L into C_L, the resulting similar-biclique is not maximal. This is formulated in the lemma below.

Lemma 1 (Early Termination)**.** *If there exists a vertex $u \in Q_L$ such that u is similar to all vertices of P_L and $N(u) \supseteq C_R$, then there is no maximal similar-biclique C^* with $C_L \subset C^*_L \subseteq C_L \cup P_L$ and thus we can terminate this enumeration instance.*

Proofs of Lemmas 1 and 2 can be found in the full version [1].

Secondly, we can reduce the number of instances generated at Line 17 of Algorithm 1, based on the concept of dominating set.

Definition 3 (Dominating Set)**.** Given an instance (C_L, C_R, P_L, Q_L) of Enum, for two distinct vertices $u, v \in P_L \cup Q_L$, we say that u dominates v if $\mathsf{sim}(u, v) \geq \varepsilon$ and $N_{C_R}(u) \supseteq N_{C_R}(v)$, where $N_{C_R}(u) = N(u) \cap C_R$. The dominating set of u, denoted $\mathsf{DomSet}(u)$, is the subset of vertices of P_L that are dominated by u, i.e., $\mathsf{DomSet}(u) = \{v \in P_L \mid \mathsf{sim}(u, v) \geq \varepsilon \wedge N_{C_R}(u) \supseteq N_{C_R}(v)\}$.

Note that, a vertex does not dominate itself.

Lemma 2. *Given an instance (C_L, C_R, P_L, Q_L) of* Enum *and a vertex $u^* \in P_L \cup Q_L$, any maximal similar-biclique C^* with $C_L \subset C^*_L \subseteq C_L \cup P_L$ must contain a vertex of $P_L \backslash \mathsf{DomSet}(u^*)$.*

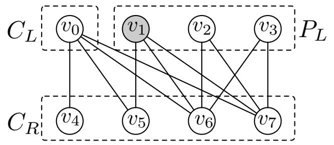


Figure 3: Example of domination

Example 2. *Consider the instance in Figure 3 where $Q_L = \emptyset$. For $\varepsilon = 0.6$, $v_1 \in P_L$ is similar to both v_2 and v_3. Moreover, we can see that $N_{C_R}(v_1) \supseteq N_{C_R}(v_2)$ and $N_{C_R}(v_1) \supseteq N_{C_R}(v_3)$. Thus,* $\mathsf{DomSet}(v_1) =$ *$\{v_2, v_3\}$, and we know that every maximal similar-biclique C^* with $C_L \subset C^*_L \subseteq C_L \cup P_L$ must contain v_1 since $P_L \backslash \mathsf{DomSet}(v_1) = \{v_1\}$.*

To apply Lemma 2, we revise Line 13 of Algorithm 1 as follows: we first choose a vertex u^* from $P_L \cup Q_L$ before the "for loop", and then replace "$u \in P_L$" with "$u \in P_L \backslash \mathsf{DomSet}(u^*)$" in the "for loop" statement. This means that we do not generate enumeration instances, at Line 17 of Algorithm 1, for vertices $u \in \mathsf{DomSet}(u^*)$. To maximize the benefit of Lemma 2, u^* is chosen as the one that minimizes $|P_L \backslash \mathsf{DomSet}(u^*)|$ among all vertices of $P_L \cup Q_L$.

Theorem 2. *The time complexity of Algorithm 1 is $O(|V_L| \cdot |E| \cdot 2^{|V_L|})$.*

The proof of Theorem 2 can be found in the full version [1].

[1] The formula should be changed to $\frac{c(v)}{\sqrt{d(u) \times d(v)}}$ for cosine similarity, and to $\frac{c(v)}{\min\{d(u), d(v)\}}$ for hub promoted index.

[2] To be more precise, we should exclude from C_R all vertices that are marked as deleted.

Discussions. MSBE is different from both maximal clique enumeration algorithms for unipartite graphs and maximal biclique enumeration algoirthms for bipartie graphs, as follows. Firstly, MSBE needs to compute the similar neighbors for L-side vertices, which are not required by any of the existing algorithms. Secondly, compared with maximal clique enumeration algorithms, MSBE needs to consider common structural neighbors C_R of C_L, in addition to common similar neighbors. Thirdly, compared with the state-of-the-art algorithm PMBE for maximal biclique enumeration, MSBE needs to maintain the set Q_L for checking maximality of similar-bicliques. Forthly, our optimization techniques for Enum are also different.

In MSBE, we need to obtain the similar neighbors $\Gamma(\cdot)$ of an L-side vertex multiple times, e.g., at Lines 6 and 15 of Algorithm 1 and Lines 2 and 6 of Algorithm 2. We can either invoke SimNei to compute $\Gamma(u)$ every time when it is needed, or store $\Gamma(u)$ in main memory after it is computed for the first time and then directly retrieve it for all subsequent requests. We use MSBE to denote the algorithm that uses the first strategy, and mat-MSBE the algorithm that uses the second strategy (here, mat stands for materialization).

4 SPEEDING UP SIMILAR NEIGHBOR COMPUTATION AND VERTEX REDUCTION

MSBE has the disadvantage of repeatedly computing the similar neighbors from scratch which is time consuming, while mat-MSBE may demand an extremely large main memory to store the similar neighbors. For example, it would take more than 400GB main memory for the graph bibsonomy used in our experiments even for a moderate $\varepsilon = 0.5$. In this section, we propose an offline-constructed index to speed up the computation of $\Gamma(u)$ as well as vertex reduction. We give an overview of the index structure in Section 4.1, present our index-based algorithms in Section 4.2, and discuss index construction and maintenance in Section 4.3.

4.1 Overview of Index Structure

Let Φ_u be the set of 2-hop structural neighbors of u, i.e., $\Phi_u = \bigcup_{v \in N(u)} N(v)$. Firstly, we have the following lemma.

Lemma 3. *For any similarity threshold $\varepsilon > 0$, the set of similar neighbors of u is a subset of Φ_u, i.e., $\Gamma_\varepsilon(u) \subseteq \Phi_u$.*

Proof. The correctness of the lemma directly follows from the fact that any vertex $v \notin \Phi_u \cup \{u\}$ has no common neighbor with u and thus $\mathsf{sim}(u, v) = 0$. □

Based on Lemma 3, one possible indexing strategy is pre-computing and storing $\mathsf{sim}(u, v)$ for each $u \in V_L \cup V_R$ and each $v \in \Phi_u$.[3] However, the space complexity of this strategy would be $O(|V_L|^2 + |V_R|^2)$, which is prohibitively high even for moderate-sized graphs since the space requirement is essentially the same as the case of mat-MSBE when ε is very small. For example, even for a moderate-sized graph with 10^6 vertices, the storage space would be over 2TB.

Instead of storing Φ_u and the structural similarities in their raw format, we summarize them into a few segments.

Definition 4 (Segment)**.** A segment, denoted seg, of Φ_u is a four-tuple $\langle \mathsf{v}_{\min}, \mathsf{v}_{\max}, \mathsf{s}_{\max}, c \rangle$, where $\mathsf{v}_{\min} \le \mathsf{v}_{\max}$ are two vertices of Φ_u, $\mathsf{s}_{\max} = \max_{v \in \Phi_u : \mathsf{v}_{\min} \le v \le \mathsf{v}_{\max}} \mathsf{sim}(u, v)$, and $c = |\{v \in \Phi_u \mid \mathsf{v}_{\min} \le v \le \mathsf{v}_{\max}\}|$. Here, vertex comparison is based on vertex id.

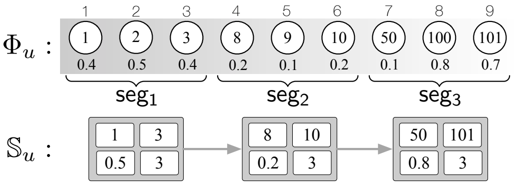


Figure 4: Overview of index structure

Algorithm 4: indexedSN($u, \varepsilon, G, \mathcal{I}$, del($\cdot$))

```
1 $\Gamma(u) \leftarrow \emptyset$;
2 for each seg $\in \mathbb{S}_u$ s.t. seg.$\mathsf{s}_{\max} \ge \varepsilon$ do
3     for each $v \in$ [seg.$\mathsf{v}_{\min}$, seg.$\mathsf{v}_{\max}$] do
4         if del($v$) = false and $v \ne u$ then
5             if ub($u, v$) $\ge \varepsilon$ and sim($u, v$) $\ge \varepsilon$ then
6                 $\Gamma(u) \leftarrow \Gamma(u) \cup \{v\}$;
7 return $\Gamma(u)$;
```

Given a segment seg $= \langle \mathsf{v}_{\min}, \mathsf{v}_{\max}, \mathsf{s}_{\max}, c \rangle$ of Φ_u, we use $V(\text{seg})$ to denote $\{v \in \Phi_u \mid \mathsf{v}_{\min} \le v \le \mathsf{v}_{\max}\}$. It is immediate from the definition that $c = |V(\text{seg})|$ and

- $\mathsf{v}_{\min}$ (resp. $\mathsf{v}_{\max}$) is the smallest (resp. largest) vertex id in $V(\text{seg})$;
- $\mathsf{s}_{\max}$ is the largest similarity between u and a vertex of $V(\text{seg})$, and thus $\mathsf{s}_{\max}$ provides an upper bound of $\mathsf{sim}(u, v)$ for all $v \in V(\text{seg})$.

Thus, we say that seg *covers* vertices $V(\text{seg})$. *A set of segments* $\mathbb{S}_u = \{\text{seg}_1, \dots, \text{seg}_k\}$ *covers* Φ_u *if* $\bigcup_{\text{seg} \in \mathbb{S}_u} V(\text{seg}) = \Phi_u$. In this paper, we only consider disjoint segments, i.e., $V(\text{seg}_i) \cap V(\text{seg}_j) = \emptyset$ for $i \ne j$. Our index structure, denoted $\mathcal{I}$, covers Φ_u by a set of segments, for all $u \in V_L \cup V_R$. That is, $\mathcal{I}$ consists of $\mathbb{S}_u$ such that $\mathbb{S}_u$ covers Φ_u, for all $u \in V_L \cup V_R$.

Example 3. *Figure 4 shows the 2-hop structural neighbors Φ_u of u, which are sorted in increasing order regarding vertex id. The decimal below each vertex is the similarity w.r.t. u. Φ_u is covered by three segments* seg$_1$, seg$_2$, seg$_3$. *Take* seg$_1$ *as an example, the two numbers in the first row (i.e.,* 1 *and* 3*) represent* $\mathsf{v}_{\min}$ *and* $\mathsf{v}_{\max}$*, and the two numbers in the second row (i.e.,* 0.5 *and* 3*) represent* $\mathsf{s}_{\max}$ *and* c.

4.2 Index-based Algorithms

In this subsection, we present index-based algorithms for similar neighbor computation and for vertex reduction.

Index-based Similar Neighbor Computation. The pseudocode of using the index $\mathcal{I}$ to efficiently obtain the similar neighbors $\Gamma(u)$ for a vertex u is shown in Algorithm 4, denoted indexedSN. We go through each segment seg $\in \mathbb{S}_u$ with seg.$\mathsf{s}_{\max} \ge \varepsilon$ (Line 2), and compute $\mathsf{sim}(u, v)$ for each $v \in$ [seg.$\mathsf{v}_{\min}$, seg.$\mathsf{v}_{\max}$] (Line 3); recall that (1) seg.$\mathsf{s}_{\max}$ upper bounds $\mathsf{sim}(u, v)$ for each $v \in V(\text{seg})$, and (2) $V(\text{seg})$ is not stored in the index structure $\mathcal{I}$. As computing $\mathsf{sim}(u, v)$ needs to intersect two sets $N(u)$ and $N(v)$ which is costly, we propose to first apply a filtering for the pair u and v based on an upper bound $\mathsf{ub}(u, v)$ of $\mathsf{sim}(u, v)$ (Line 5); if $\mathsf{ub}(u, v) < \varepsilon$, then we have $\mathsf{sim}(u, v) \le \mathsf{ub}(u, v) < \varepsilon$ and thus $v \notin \Gamma(u)$. For the similarity

[3] Note that, we also need to index Φ_u for $u \in V_R$, since in practice the similarity constraint can be put on either L-side or R-side vertices.

in Definition 1, it is easy to verify that $\mathsf{sim}(u, v) \le \frac{\min\{d(u),d(v)\}}{\max\{d(u),d(v)\}}$; we set this as $\mathsf{ub}(u, v)$, which can be calculated in constant time.[4] indexedSN is expected to run faster than SimNei (Algorithm 3) as the former can skip an entire segment if its $\mathsf{s}_{\max}$ is smaller than ε.

Index-based Two-Phase Vertex Reduction. Based on indexedSN, we can speed up VReduce (Algorithm 2) by invoking indexedSN to compute $\Gamma(u)$. However, this is still inefficient, as VReduce needs to compute $\Gamma(u)$ for all $u \in V_L$ (see Line 2 of Algorithm 2). We propose to utilize the index $\mathcal{I}$ to first obtain an upper bound of the similar degree for vertex reduction, as proved in the lemma below.

Lemma 4 (Upper Bound of Similar Degree)**.** *Let $\mathbb{S}_u$ be the set of segments that cover Φ_u. Then, the similar degree $\delta_\varepsilon(u)$ of u is upper bounded by $\sum_{\mathsf{seg} \in \mathbb{S}_u : \mathsf{seg}.\mathsf{s}_{\max} \ge \varepsilon} \mathsf{seg}.c$.*

Proof. This lemma directly follows from the fact that $\mathsf{sim}(u, v) < \varepsilon$ for all $v \in \bigcup_{\mathsf{seg} \in \mathbb{S}_u : \mathsf{seg}.\mathsf{s}_{\max} < \varepsilon} V(\mathsf{seg})$. □

Consider the part of the index in Figure 4 and suppose $\varepsilon = 0.4$. By scanning $\mathbb{S}_u$, we obtain an upper bound of u's similar degree as 6, i.e., $\mathsf{seg}_1.c + \mathsf{seg}_3.c = 6$; seg_2 is omitted since its $\mathsf{s}_{\max}$ is only 0.2.

Algorithm 5: indexedVR($G, \mathcal{I}, \varepsilon, \tau, \mathsf{del}(\cdot)$)

```
   /* Phase-I: vertex reduction based on structural degree and upper bound
      of similar degree                                                   */
1  for each $u \in V_L \cup V_R$ do $d(u) \leftarrow |N(u)|$;
2  for each $u \in V_L$ do $\bar{\delta}(u) \leftarrow \sum_{\mathsf{seg} \in \mathbb{S}_u : \mathsf{seg}.\mathsf{s}_{\max} \ge \varepsilon} \mathsf{seg}.c$;
3  while $(\exists u \in V_L \cup V_R$ s.t. $\mathsf{del}(u) =$ false and $d(u) < \tau)$ or $(\exists u \in V_L$ s.t.
     $\mathsf{del}(u) =$ false and $\bar{\delta}(u) < \tau - 1)$ do
4      for each $v \in N(u)$ do $d(v) \leftarrow d(v) - 1$;
5      $\mathsf{del}(u) \leftarrow$ true;
   /* Phase-II: vertex reduction based on structural degree and similar
      degree                                                              */
6  for each $u \in V_L \cup V_R$ do $\mathsf{del2}(u) \leftarrow \mathsf{del}(u)$;
7  for each $u \in V_L$ s.t. $\mathsf{del}(u) =$ false do
8      $(\delta_p(u), \mathsf{idx}(u)) \leftarrow \mathsf{progressiveSN}(u, \varepsilon, G, \mathcal{I}, \mathsf{del2}(\cdot), \tau - 1, 1)$;
9  while $(\exists u \in V_L \cup V_R$ s.t. $\mathsf{del}(u) =$ false and $d(u) < \tau)$ or $(\exists u \in V_L$ s.t.
     $\mathsf{del}(u) =$ false and $\delta_p(u) < \tau - 1)$ do
10     for each $v \in N(u)$ do $d(v) \leftarrow d(v) - 1$;
11     if $u \in V_L$ then
12         $\Gamma(u) \leftarrow \mathsf{indexedSN}(u, \varepsilon, G, \mathcal{I}, \mathsf{del}(\cdot))$;
13         for each $v \in \Gamma(u)$ do
14             $\delta_p(v) \leftarrow \delta_p(v) - 1$;
15             if $\delta_p(v) = \tau - 2$ and $d(v) \ge \tau$ then
16                 $(r, \mathsf{idx}(v)) \leftarrow$
                     $\mathsf{progressiveSN}(v, \varepsilon, G, \mathcal{I}, \mathsf{del2}(\cdot), 1, \mathsf{idx}(v))$;
17                 $\delta_p(v) \leftarrow \delta_p(v) + r$;
18     $\mathsf{del}(u) \leftarrow$ true;

   Procedure progressiveSN($u, \varepsilon, G, \mathcal{I}, \mathsf{del2}(\cdot), t, b$)
   /* Let $\mathbb{S}_u$ be $\{\mathsf{seg}_1, \mathsf{seg}_2, \ldots, \mathsf{seg}_{|\mathbb{S}_u|}\}$                    */
19 $r \leftarrow 0$;
20 for each $i \in \{b, b+1, \ldots, |\mathbb{S}_u|\}$ s.t. $\mathsf{seg}_i.\mathsf{s}_{\max} \ge \varepsilon$ do
21     for each $v \in [\mathsf{seg}_i.\mathsf{v}_{\min}, \mathsf{seg}_i.\mathsf{v}_{\max}]$ do
22         if $\mathsf{del2}(v) =$ false and $v \ne u$ then
23             if $\mathsf{ub}(u, v) \ge \varepsilon$ and $\mathsf{sim}(u, v) \ge \varepsilon$ then
24                 $r \leftarrow r + 1$;
25     if $r \ge t$ then return $(r, i + 1)$;
26 return $(r, |\mathbb{S}_u| + 1)$;
```

Furthermore, we also observe that the structural degree can be obtained efficiently. Thus, we propose a two-phase approach for vertex reduction, which first conducts vertex reduction by using structural degree and upper bound of similar degree in Phase-I, and then using structural degree and similar degree in Phase-II. The pseudocode of our two-phase vertex reduction is shown in Algorithm 5, denoted indexedVR. In Phase-I, we first obtain the structural degree $d(u)$ for each $u \in V_L \cup V_R$ (Line 1), and an upper bound $\bar{\delta}(u)$ of the similar degree for each vertex $u \in V_L$ (Line 2). Then, as long as there is a non-deleted vertex $u \in V_L \cup V_R$ satisfying $d(u) < \tau$ or a non-deleted vertex $u \in V_L$ satisfying $\bar{\delta}(u) < \tau - 1$ (Line 3), we mark u as deleted and update the structural degree of its structural neighbors (Lines 4–5); note that, we do not update $\bar{\delta}(\cdot)$ in Phase-I. In Phase-II, we first compute a *progressive* similar degree, denoted $\delta_p(\cdot)$, for each non-deleted L-side vertex, by invoking progressiveSN (Lines 7–8). Here, $\delta_p(u)$ is a lower bound of u's similar degree $\delta(u)$, and it records the number of similar neighbors that have been computed for u; our computation of $\delta_p(u)$ ensures that $\delta_p(u) \ge \tau - 1$ if and only if $\delta(u) \ge \tau - 1$. Then, as long as there is a non-deleted vertex $u \in V_L \cup V_R$ satisfying $d(u) < \tau$ or a non-deleted vertex $u \in V_L$ satisfying $\delta_p(u) < \tau - 1$, we mark u as deleted (Line 18) and update the structural degree of its structural neighbors (Line 10). Furthermore, if u is an L-side vertex, we also obtain the set $\Gamma(u)$ of similar neighbors of u (Line 12), and update the progressive similar degree $\delta_p(v)$ to satisfy the invariant that $\delta_p(v) \ge \tau - 1$ if and only if $\delta(v) \ge \tau - 1$ for each $v \in \Gamma(u)$ (Lines 13–17). Note that, in our implementation, we use a queue to maintain the vertices that satisfy the condition at Line 3 or Line 9; as a result, we do not need to loop through all non-deleted vertices to find the unpromising vertices.

In Algorithm 5, for an L-side vertex u, we compute $\delta_p(u)$ instead of $\delta(u)$. Our main motivation is that for an L-side vertex u satisfying $d(u) \ge \tau$, we only need to compute $\tau - 1$ of its similar neighbors to certify that it is a promising vertex. That is, we stop the computation of $\Gamma(u)$ once $\delta_p(u) \ge \tau - 1$; however, if some of the computed similar neighbors of u are later removed (i.e., marked as deleted), then we need to update $\delta_p(u)$ by computing more similar neighbors of u (Lines 15–17 of Algorithm 5). As a result, for vertices with high similar degrees in the remaining graph (i.e., obtained by removing all unpromising vertices), we only need to compute a small portion of their similar neighbors to prevent them from being removed and thus save unnecessary similar neighbor computations. The pseudocode of computing $\delta_p(u)$ is shown in Lines 19–26 of Algorithm 5, denoted progressiveSN. It is invoked only when $\delta_p(u) < \tau - 1$ and there are still unchecked segments of Φ_u. In progressiveSN, we check the segments of $\mathbb{S}_u$ one by one (Line 20–24), and stop once we have found enough similar neighbors for u (Line 25). We record the index of the first unchecked segment in $\mathsf{idx}(u)$ (Line 8).

indexedVR is better than VReduce (Algorithm 2), since (1) Phase-I of indexedVR is lightweight but very effective at pruning vertices as demonstrated by our empirical studies, and (2) indexedVR uses indexedSN and progressiveSN to compute the similar neighbors.

Overall Algorithm. Our index-based MSBE improves upon Algorithm 1 by replacing the invocation to VReduce at Line 2 with invoking indexedVR for vertex reduction, and invokes indexedSN

[4] The upper bound for cosine similarity is $\frac{\min\{d(u),d(v)\}}{\sqrt{d(u)\times d(v)}}$, while the upper bound for hub promoted index is 1 and thus not useful.

to compute $\Gamma(u)$ at Lines 6 and 15. Nevertheless, the time complexity of index-based MSBE remains $O(|V_L| \cdot |E| \cdot 2^{|V_L|})$ as proved in Theorem 2, by noting that the time complexity of indexedSN remains $O(|E|)$. Despite of having the same time complexity, our empirical studies in Section 5 show that the index-based approach can improve the efficiency of MSBE by several orders of magnitude.

4.3 Index Construction and Maintenance

In this subsection, we present two algorithms to construct the index based on the ideas of *largest gap* and *steady segment*, respectively. Note that, the indexes are constructed offline, and once constructed, they can be used to process maximal similar-biclique enumeration queries with different ε and τ values.

Largest Gap (LG) Index. Recall that, our index structure summarizes a subset of vertices of Φ_u and their similarities to a vertex u by four numbers seg $= \langle \mathsf{v}_{\min}, \mathsf{v}_{\max}, \mathsf{s}_{\max}, c\rangle$, where $\mathsf{s}_{\max}$ is an upper bound of the similarity between u and each $v \in \Phi_u$ such that $\mathsf{v}_{\min} \leq v \leq \mathsf{v}_{\max}$. To obtain the similar neighbors of u that are in the range $[\mathsf{v}_{\min}, \mathsf{v}_{\max}]$, we need to go through each vertex $v \in [\mathsf{v}_{\min}, \mathsf{v}_{\max}]$ and test its similarity with u (e.g., see Line 3 of Algorithm 4) even if $v \notin V(\text{seg})$. We call a vertex v that is in the range $[\mathsf{v}_{\min}, \mathsf{v}_{\max}]$ but not in $V(\text{seg})$ a *fake vertex*.

Intuitively, we should minimize the number of fake vertices when constructing the index. We call the index built by this strategy the largest gap (LG) index. We omit the details, since it is outperformed by our steady segment index as introduced next.

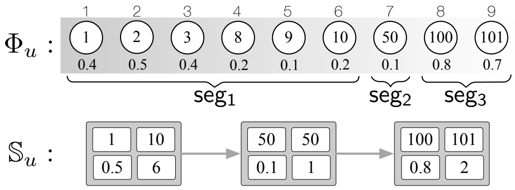


Figure 5: Example of consLG

Example 4. *Figure 5 shows the three segments constructed by the largest gap strategy for the same Φ_u as Example 3.*

Choosing the number of segments to cover Φ_u. It is easy to see that the more the number of segments, the fewer the number of fake vertices introduced by the segments. In the extreme case of covering Φ_u by $|\Phi_u|$ segments, there will be no fake vertices introduced. However, the space complexity would be too high to be practical, as discussed in Section 4.1. Thus, we set the number of segments for covering Φ_u as $\alpha \cdot \log |\Phi_u|$ where α is a user defined parameter, in viewing that a fixed number for different Φ_u will not work as $|\Phi_u|$ varies a lot across different vertices u.

Steady Segment (SS) Index. The LG index ignores the similarities (between u and different vertices) in a segment, and thus may result in a very wide range of similarity values for a segment. This is not good for indexedSN and progressiveSN, as they need to check all vertices covered by a segment seg even if there is only one vertex in seg whose similarity to u is no lower than ε. Motivated by this, we aim to construct *steady segments* such that all similarities in a segment are close to each other.

Definition 5 (Steady Segment)**.** Given a steady threshold $0 < \gamma < 1$, a segment seg $= \langle \mathsf{v}_{\min}, \mathsf{v}_{\max}, \mathsf{s}_{\max}, c\rangle$ of Φ_u is steady if $\max_{v \in V(\text{seg})} \mathsf{sim}(u, v) - \min_{v \in V(\text{seg})} \mathsf{sim}(u, v) \leq \gamma$.

The first term, $\max_{v \in V(\text{seg})} \mathsf{sim}(u, v)$, is exactly seg.$\mathsf{s}_{\max}$. For ease of presentation, we denote the second term, $\min_{v \in V(\text{seg})} \mathsf{sim}(u, v)$, by seg.$\mathsf{s}_{\min}$, the smallest similarity value. A segment seg is steady if seg.$\mathsf{s}_{\max}$ − seg.$\mathsf{s}_{\min} \leq \gamma$. The main advantage of a steady segment is that if seg is steady and satisfies seg.$\mathsf{s}_{\max} \geq \varepsilon$, then it is likely that many vertices of $V(\text{seg})$ have similarity values to u no lower than ε, and thus most of the computation will not be wasted.

Ideally, we would like to find the minimum number of steady segments to cover Φ_u. However, the number of required steady segments could be very large. For example, if the steady threshold γ is very close to 0 and all vertices of Φ_u have different similarity values to u, then the number of required steady segments to cover Φ_u is $|\Phi_u|$. Thus, we instead construct a fixed number of steady segments to cover as many vertices of Φ_u as possible, and then cover the remaining uncovered vertices of Φ_u by as few segments as possible by ignoring the difference between the similarity values.

Given γ and k, our problem is to find k steady segments to cover as many vertices of Φ_u as possible. We first construct, for each vertex $v \in \Phi_u$, a maximal steady segment seg$_v$ that starts at v (i.e. seg$_v$.$\mathsf{v}_{\min} = v$), and then select k segments $\mathbb{S}^*$ from $\{\text{seg}_v \mid v \in \Phi_u\}$ such that $\left|\bigcup_{v \in \mathbb{S}^*} V(\text{seg}_v)\right|$ is maximized. This is an instance of the maximum k-coverage problem which is NP-hard [39]. We select the k segments in a greedy manner. That is, the k segments are selected one-by-one. Let S be the starting vertices of the currently selected segments. Then, the next segment to be added to S is $\arg\max_{v \in \Phi_u} \left|\bigcup_{v' \in S \cup \{v\}} V(\text{seg}_{v'})\right|$. As this function is submodular, the greedy approach achieves an approximation ratio of $1 - \frac{1}{e}$ [18].

The pseudocode is shown in Algorithm 6, denoted consSS. For each vertex u, we first compute its 2-hop structural neighbors Φ_u and their similarities to u (Line 2), and sort Φ_u in increasing vertex id order (Line 3). Then, for each $v_i \in \Phi$, we compute the maximal steady segment seg$_{v_i}$ that starts at v_i, by iteratively trying to add the next vertex to the segment (Lines 7–12). Next, we iteratively add to $\mathbb{S}_u$ the segment of $\mathbb{C}$ that covers the largest number of uncovered vertices of Φ_u (Lines 14–22). Note that, after adding a segment into $\mathbb{S}_u$, we also need to update the remaining segments of $\mathbb{C}$ to be disjoint from the segments of $\mathbb{S}_u$ (Lines 18–22). During this process, for time efficiency consideration, we do not maintain seg.$\mathsf{s}_{\max}$; instead, we compute seg.$\mathsf{s}_{\max}$ for each segment seg $\in \mathbb{S}_u$ later (Line 23). Finally, we create the minimum number of segments to cover all vertices of Φ_u that are not covered by $\mathbb{S}_u$ (Lines 24–25).

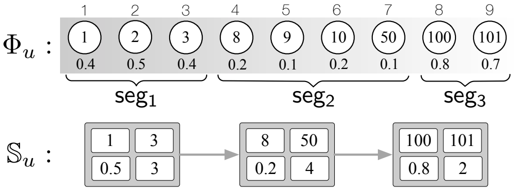


Figure 6: Example of consSS

Example 5. *Figure 6 shows the three steady segments constructed for the same Φ_u in Examples 3 and 4, where $\gamma = 0.1$.*

Algorithm 6: $\mathsf{consSS}(G = (V_L, V_R, E), \alpha, \gamma)$

```
1  for each $u \in V_L \cup V_R$ do
2      $\Phi_u \leftarrow \mathsf{SimNei}(G, u, \frac{1}{2|V_L|+2|V_R|})$;
3      Let $\{v_1, v_2, \ldots, v_{|\Phi_u|}\}$ be vertices of $\Phi_u$ in increasing vertex id order;
4      $\mathbb{C} \leftarrow \emptyset$;
5      for $i \leftarrow 1$ to $|\Phi_u|$ do
6          $\mathsf{s}_{\min} \leftarrow 1$;   $\mathsf{s}_{\max} \leftarrow 0$;
7          for $j \leftarrow i$ to $|\Phi_u|$ do
8              if $\mathsf{sim}(u, v_j) < \mathsf{s}_{\min}$ then $\mathsf{s}_{\min} \leftarrow \mathsf{sim}(u, v_j)$;
9              if $\mathsf{sim}(u, v_j) > \mathsf{s}_{\max}$ then $\mathsf{s}_{\max} \leftarrow \mathsf{sim}(u, v_j)$;
10             if $\mathsf{s}_{\max} - \mathsf{s}_{\min} > \gamma$ then
11                 $\mathsf{seg}_{v_i} \leftarrow \langle v_i, v_{j-1}, \mathsf{null}, j - i\rangle$;
12                 break;
13         $\mathbb{C} \leftarrow \mathbb{C} \cup \{\mathsf{seg}_{v_i}\}$;
14     $\mathbb{S}_u \leftarrow \emptyset$;   $k \leftarrow \min\{|\Phi_u|, \alpha \cdot \log |\Phi_u|\}$;
15     while $|\mathbb{S}_u| < k$ and $\mathbb{C} \neq \emptyset$ do
16         $\mathsf{seg}^* \leftarrow \arg\max_{\mathsf{seg} \in \mathbb{C}} \mathsf{seg}.c$;
17         $\mathbb{S}_u \leftarrow \mathbb{S}_u \cup \{\mathsf{seg}^*\}$;
18         for each $\mathsf{seg} \in \mathbb{C}$ do
19             if $\mathsf{seg}^*.\mathsf{v}_{\min} < \mathsf{seg}.\mathsf{v}_{\min} \leq \mathsf{seg}^*.\mathsf{v}_{\max}$ then
20                 Remove seg from $\mathbb{C}$
21             else if $\mathsf{seg}.\mathsf{v}_{\min} < \mathsf{seg}^*.\mathsf{v}_{\min} \leq \mathsf{seg}.\mathsf{v}_{\max}$ then
22                 Let $v$ be the vertex that immedidately precedes $\mathsf{seg}^*.\mathsf{v}_{\min}$
                     in $\Phi_u$, change $\mathsf{seg}.\mathsf{v}_{\max}$ to be $v$, and update $\mathsf{seg}.c$
                     accordingly in $\mathbb{C}$;
23     for each $\mathsf{seg} \in \mathbb{S}_u$ do Compute $\mathsf{seg}.\mathsf{s}_{\max}$;
24     for each maximal consecutive sequence of vertices $v_i, v_{i+1}, \ldots, v_j$ of $\Phi_u$
         that are not covered by $\mathbb{S}_u$ do
25         Add to $\mathbb{S}_u$ the segment that covers $\{v_i, \ldots, v_j\}$;
26 return $\mathcal{I} = \{\mathbb{S}_u \mid u \in V_L \cup V_R\}$;
```

Similarity Tree. Lines 5–12 of Algorithm 6, which constructs the initial maximal steady segments for each vertex, has a high time complexity of $O(|\Phi_u|^2)$, and may dominate the total running time of Algorithm 6. In view of this, we build a similarity tree data structure $\mathcal{T}_u$ for each Φ_u to speed up the process. $\mathcal{T}_u$ is similar to a range tree or segment tree [11]. Each tree node t of $\mathcal{T}_u$ represents a range of vertices of Φ_u — specifically, the vertices corresponding to the leaf nodes of the subtree rooted at t — and records two values $t.\mathsf{s}_{\min}$ and $t.\mathsf{s}_{\max}$ which are, respectively, the smallest similarity and the largest similarity among the vertices represented by t. An example similarity tree is shown in Figure 7. t_5 represents the third vertex and the forth vertex, while t_3 represents the last four vertices of Φ_u. Let $\{v_1, v_2, \ldots, v_{|\Phi_u|}\}$ be the vertices of Φ_u sorted in increasing id order. We first create one leaf node t for each vertex $v \in \Phi_u$ with $t.\mathsf{s}_{\min} = t.\mathsf{s}_{\max} = \mathsf{sim}(u, v)$. Then, we construct the tree layer-by-layer in a bottom-up manner. Let T_i be the list of tree nodes at the current layer. We go through T_i by accessing two tree nodes each time. For each pair of tree nodes t and t', we create a new tree node t_p as their parent and put t_p into list T_{i+1}; if there is only one node in the last step, we directly put it into T_{i+1}. Note that, $t_p.\mathsf{s}_{\min} = \min\{t.\mathsf{s}_{\min}, t'.\mathsf{s}_{\min}\}$ and similarly $t_p.\mathsf{s}_{\max}$. The construction finishes when a layer has only one node, which is the root of the similarity tree. It is easy to see that tree construction takes $O(|\Phi_u|)$ time, as the tree is a complete binary tree.

To compute the maximal steady segment of v, we first traverse the similarity tree upwards, starting from the leaf node that corresponds to v, and then go downwards. During the process, we maintain $\mathsf{s}_{\min}$ and $\mathsf{s}_{\max}$, which are initialized by $\mathsf{sim}(u, v)$. In the upward phase, if the current tree node t is a right child of its parent, then we directly go to its parent. Otherwise, let t' be the right-sibling of t. If $\max\{\mathsf{s}_{\max}, t'.\mathsf{s}_{\max}\} - \min\{\mathsf{s}_{\min}, t'.\mathsf{s}_{\min}\} \leq \gamma$ which means that we can include all vertices represented by t' into the segment, then we update $\mathsf{s}_{\min}$ and $\mathsf{s}_{\max}$ by $\min\{\mathsf{s}_{\min}, t'.\mathsf{s}_{\min}\}$ and $\max\{\mathsf{s}_{\max}, t'.\mathsf{s}_{\max}\}$, respectively, and go to its parent. Otherwise, we go to t' and move into the downward phase. In the downward phase, let t_l be the left child of the current node t. If $\max\{\mathsf{s}_{\max}, t_l.\mathsf{s}_{\max}\} - \min\{\mathsf{s}_{\min}, t_l.\mathsf{s}_{\min}\} \leq \gamma$, then we update $\mathsf{s}_{\min}$ and $\mathsf{s}_{\max}$ and go to t's right child. Otherwise, we go to t's left child. Finally, when we arrive at a leaf node, we can decide whether the corresponding vertex should be included into the segment or not. It is easy to see that this process takes $O(\log |\Phi_u|)$ time which is the height of the similarity tree. Thus, constructing the maximal steady segment for all vertices of Φ_u takes $O(|\Phi_u| \log |\Phi_u|)$ time.

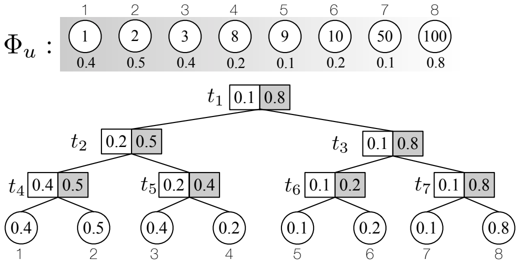


Figure 7: Similarity tree data structure

Example 6. *Suppose we are going to construct the maximal steady segment for the second vertex of Φ_u as shown in Figure 7 with $\gamma = 0.3$. Initially, $\mathsf{s}_{\min} = \mathsf{s}_{\max} = 0.5$ and t is the second leaf node. As t is a right child of its parent t_4, we directly go to t_4. Now, t_4 is a left child of its parent t_2, and t_4's right sibling is t_5. As $\max\{\mathsf{s}_{\max}, t_5.\mathsf{s}_{\max}\} - \min\{\mathsf{s}_{\min}, t_5.\mathsf{s}_{\min}\} = 0.5 - 0.2 = 0.3 \leq \gamma$, we include all vertices represented by t_5 (i.e., the third and forth vertices) into the segment, update $\mathsf{s}_{\min}$ to be $\max\{\mathsf{s}_{\min}, t_5.\mathsf{s}_{\min}\} = 0.2$ and $\mathsf{s}_{\max}$ to be 0.5, and then go to its parent t_2. t_2 is a left child of its parent and its right sibling is t_3. As $\max\{\mathsf{s}_{\max}, t_3.\mathsf{s}_{\max}\} - \min\{\mathsf{s}_{\min}, t_3.\mathsf{s}_{\min}\} = 0.8 - 0.1 > \gamma$, we go to t_3 and move into the downward phase. t_3's left child is t_6 and $\max\{\mathsf{s}_{\max}, t_6.\mathsf{s}_{\max}\} - \min\{\mathsf{s}_{\min}, t_6.\mathsf{s}_{\min}\} = 0.5 - 0.1 > \gamma$, we go to its left child t_6. Similarly, we go to t_6's left child, which is a leaf node corresponding to the fifth vertex of Φ_u. We find that the first vertex cannot be included into the segment. Thus, the maximal steady segment consists of three vertices, the second, third, and forth vertices.*

Analysis of consSS. For each vertex $u \in V_L \cup V_R$, Line 2 of Algorithm 6 takes $O(\sum_{v \in N(u)} d(v))$ time, Line 3 as well as Lines 5–12 take $O(|\Phi_u| \log |\Phi_u|)$ time; Lines 5–12 use the similarity tree data structure as discussed above. The while loop at Line 15 runs for at most $\alpha \cdot \log |\Phi_u|$ iterations, and each iteration takes $O(|\Phi_u|)$ time. Lines 23–25 take $O(|\Phi_u|)$ time. Thus, the total time complexity of consSS is $O\Big(\sum_{u \in V_L \cup V_R} \big(\alpha|\Phi_u| \log |\Phi_u| + \sum_{v \in N(u)} d(v)\big)\Big)$ time.

Index Maintenance. When the graph changes, e.g., edges are inserted or deleted, we can easily modify our index structure to reflect the changes. The main observation is that updates to the index are localized; suppose a new edge (u, v) is inserted, then only the $\mathbb{S}_w$ for $w \in \Phi_u \cup \Phi_v \cup \{u, v\}$ will change. We omit the details.

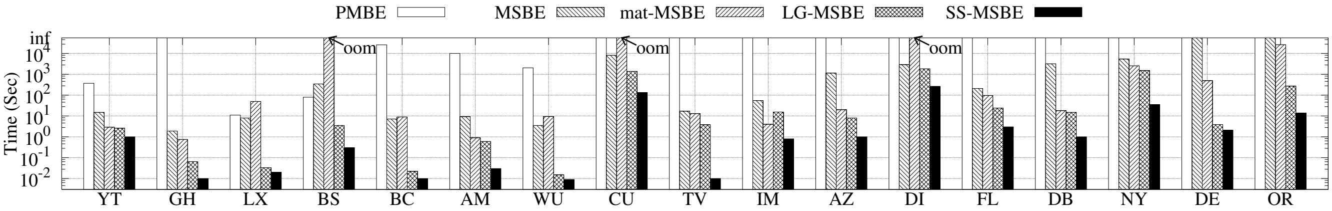


Figure 8: Running time on all graphs ($\varepsilon = 0.5$, $\tau = 3$)

Table 1: Statistics of graphs

Abbreviation	Graph	$\lvert V_L \rvert$	$\lvert V_R \rvert$	$\lvert E \rvert$	Type
YT	YouTube	94, 238	30, 087	293, 360	Membership
GH	GitHub	56, 519	120, 867	440, 237	Membership
LX	Linux	42, 045	337, 509	599, 858	Post
BS	Bibsonomy	767, 447	5, 794	801, 784	Assignment
BC	BookCross	105, 278	340, 523	1, 149, 739	Rating
AM	ActorMovie	127, 823	383, 640	1, 470, 404	Appearance
WU	WebUni	6, 202	200, 148	1, 948, 004	Appearance
CU	CiteULike	731, 769	153, 277	2, 338, 554	Assignment
TV	TVTropes	64, 415	87, 678	3, 232, 134	HasFeature
IM	IMDB	303, 617	896, 302	3, 782, 463	Appearance
AZ	Amazon	1, 879, 572	1, 162, 941	4, 955, 492	Rating
DI	Discogs	1, 754, 823	270, 771	5, 302, 276	Affiliation
FL	Flickr	395, 979	103, 631	8, 545, 307	Membership
DB	DBLP	1, 953, 085	5, 624, 219	12, 282, 059	Authorship
NY	NYTimes	299, 752	101, 636	69, 679, 427	Appearance
DE	Delicious	833, 081	33, 778, 221	101, 798, 957	Interaction
OR	Orkut	2, 783, 196	8, 730, 857	327, 037, 487	Affiliation

5 EXPERIMENTS

In this section, we evaluate the efficiency of our algorithms as well as the effectiveness of our similar-biclique model.

Algorithms. We compare the following algorithms.

- PMBE: the state-of-the-art algorithm proposed in [2] for enumerating all maximal bicliques.
- MSBE: our Algorithm 1 equipped with all the optimizations in Section 3.2.
- mat-MSBE: the materialized version of MSBE, as discussed at the end of Section 3.2.
- LG-MSBE and SS-MSBE: our index-based algorithms that use the largest gap and steady segment index, respectively.

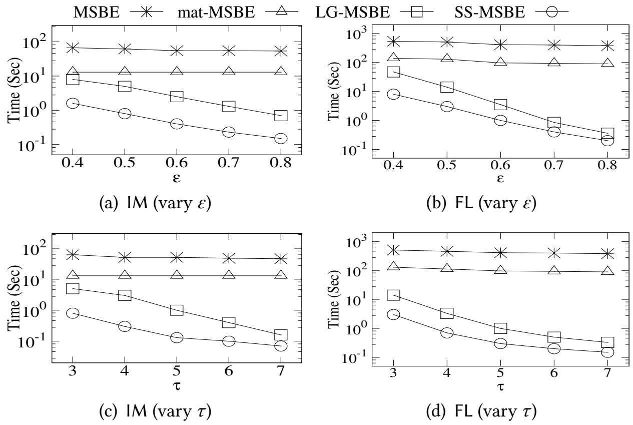
The source code of PMBE is obtained from the authors of [2]. All our algorithms are implemented in C++ and run in main memory. All experiments are conducted on a machine with an Intel(R) 3.2GHz CPU and 64GB main memory running Ubuntu 18.04.5. We set a timeout of 10 hours for running an algorithm on a graph.

Datasets. We evaluate the algorithms on 17 real bipartite graphs, all of which are publicly available on KONECT [5]. Statistics of the graphs are shown in Table 1, where the graphs are listed in increasing order regarding the number of edges.

Query Parameters. A maximal similar-biclique enumeration query consists of two parameters, ε and τ. ε is chosen from $\{0.4, 0.5, 0.6, 0.7, 0.8\}$, and is set as 0.5 by default. τ is chosen from $\{3, 4, 5, 6, 7\}$, and is set as 3 by default. In addition, we also have parameters α and γ in index construction; we set $\alpha = 1$ and $\gamma = 0.3$ by default.

5.1 Efficiency Evaluations

In this subsection, we evaluate the efficiency of the algorithms. Note that, we also implemented a version of MSBE without the optimizations of Enum proposed in Section 3.2; it is omitted from the experiments since it times out in almost all the testings.



(a) IM (vary ε)

(b) FL (vary ε)

(c) IM (vary τ)

(d) FL (vary τ)

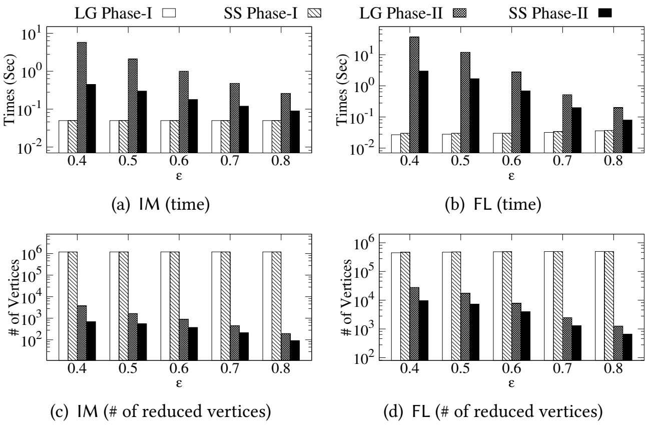
Figure 9: Running time by varying ε and τ

Running time on all graphs. The running time of the five algorithms on all graphs with default ε and τ is illustrated in Figure 8. We can see that mat-MSBE slightly improves upon MSBE when it is feasible to store the similar neighbors of all vertices in main memory. However, mat-MSBE runs out-of-memory on BS, CU, and DI, as marked by "oom" in Figure 8; for example, the memory consumption on BS would be over 400GB. Note that the memory consumption of mat-MSBE mainly depends on the structure, rather than the size, of the input graph, and thus mat-MSBE does not run out-of-memory on other larger graphs. Our two index-based algorithms, LG-MSBE and SS-MSBE, are the fastest and they outperform the other two algorithms that do not use index by up to 5 orders of magnitude. SS-MSBE is generally faster than LG-MSBE. Compared with the state-of-the-art maximal biclique enumeration algorithm PMBE, SS-MSBE is up to 6 orders of magnitude faster; note that, PMBE also uses τ to prune the search space. Thus, we exclude PMBE from our remaining evaluations.

Running time by varying ε and τ. The running time of our four algorithms on IM and FL by varying ε and τ are shown in Figure 9. We can see that the running time of LG-MSBE and SS-MSBE decreases when either ε or τ increases. This is because, more vertices will be pruned by indexedVR when either ε or τ increases, and thus the enumeration process of LG-MSBE and SS-MSBE run faster. Also, indexedVR runs faster when ε or τ increases, as can be seen from Figure 10. In contrast, the running time of MSBE and mat-MSBE is not so sensitive to ε or τ, as the dominating part of these two algorithms is computing similar neighbors for vertices.

Efficiency of indexedVR. In this experiment, we evaluate the efficiency of indexedVR for our two index structures. Recall that

[5] http://konect.cc/networks/



(a) IM (time)

(b) FL (time)

(c) IM (# of reduced vertices)

(d) FL (# of reduced vertices)

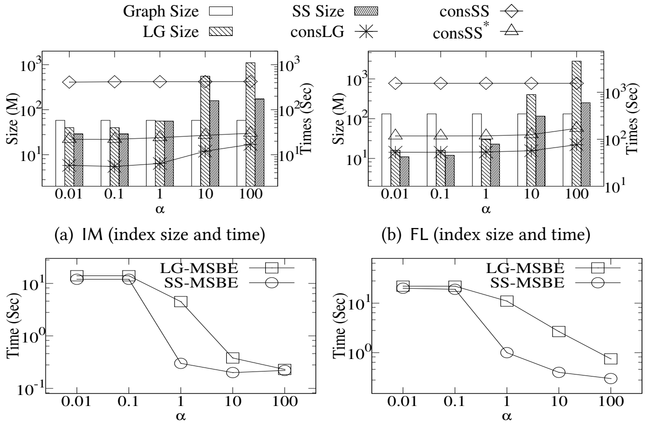
Figure 10: Efficiency of indexedVR ($\tau = 3$)

Table 2: Index size and construction time

Graph	Size	Largest gap index		Steady segment index		
		consLG (Sec)	Size	consSS (Sec)	consSS* (s)	Size
YT	4.6M	0.7	6.7M	21	1.8	8.3M
GH	7M	1.1	6.6M	90	4.9	6.2M
LX	9.2M	48	31M	9, 485	588	50M
BS	12.4M	506	75M	49, 685	6, 975	59M
BC	18M	16	27M	1, 200	130	30M
AM	24M	1.5	24M	81	7	20M
WU	30M	11	15M	656	40	16M
CU	36M	1, 130	73M	296,094	12,547	103M
TV	50M	13	4.4M	110	14	2.3M
IM	58M	10	56M	420	28	56M
AZ	76M	22	122M	2, 220	152	124M
DI	82M	549	132M	45,967	5,187	146M
FL	132M	106	30M	3, 102	235	23M
DB	188M	29	299M	1, 905	177	324M
NY	1.1G	2, 623	35M	20, 934	4, 397	14M
DE	1.5G	3, 071	2.3G	129, 304	13, 435	3.1G
OR	5G	21, 874	690M	246, 045	23, 872	459M

indexedVR (Algorithm 5) has two phases. Thus, we separately report the results of each phase. The running time on IM and FL are shown in Figures 10(a) and 10(b). We can see that the two index structures take almost the same time for the first phase, while the second phase of SS index-based indexedVR is much faster than LG index-based. This can partially be explained by the number of vertices that need to be pruned in the second phase, as reported in Figures 10(c) and 10(d). We remark that, for a fixed ε and τ, the total number of pruned vertices by different indexes are the same, and also the same as that pruned by the index-free approach VReduce. Thus, from the number of vertices that are pruned in Phase-II as shown in Figure 10, we can conclude that SS index prunes much more vertices than LG index in Phase-I. For example, for dataset FL and $\varepsilon = 0.4$, SS index prunes 467, 329 vertices in Phase-I and 9, 691 vertices in Phase-II, while LG index prunes 449, 195 vertices in Phase-I and 27, 825 vertices in Phase-II. As the second phase dominates the running time, SS index is superior.

Index size and construction time on all graphs. The size of the two indexes on all graphs are shown in the fourth column and last column of Table 2. As a comparison, we also report the graph size in the second column of Table 2. We can see that in most cases, the sizes of the two indexes are similar to each other and are at the same level as the graph size, and thus they are affordable to be stored in main memory.



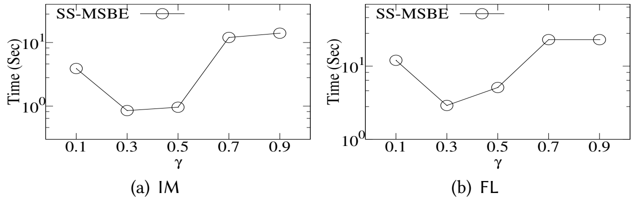
(a) IM (index size and time)

(b) FL (index size and time)

(c) IM (efficiency of MSBE)

(d) FL (efficiency of MSBE)

Figure 11: Index performance by varying α



(a) IM

(b) FL

Figure 12: Efficiency of SS-MSBE by varying γ

The running time of our index construction algorithms consLG, consSS, and consSS* are reported in the third, fifth, and sixth columns of Table 2, respectively. consLG runs the fastest due to its simplicity. Nevertheless, consSS*, which optimizes consSS by the similarity tree data structure, is only slightly slower than consLG.

Index performance by varying α. In this experiment, we evaluate the effect of α on the index size, index construction time and efficiency of MSBE. The results are shown in Figure 11. Recall that α controls the number of segments constructed for Φ_u. As expected, the index size and index construction time increase along with the increasing of α, as shown in Figures 11(a) and 11(b). When α is no larger than 1, the index size is at most at the same level as the graph size, but when α reaches 100, the index size can be much larger than the graph size. As shown in Figures 11(c) and 11(d), the running time of both LG-MSBE and SS-MSBE decreases when α increases. This is because the more the number of segments, the fewer the number of fake vertices. To strike a balance between index size and efficiency of MSBE, we recommend to set $\alpha \in [0.1, 10]$.

Efficiency of SS-MSBE by varying γ. In this experiment, we evaluate the performance of SS-MSBE for different γ values. Note that, the index size and index construction time of consSS are almost not affected by γ; thus we omit these results. This is because consSS selects a fixed number of steady segments (i.e., $\alpha \log |\Phi_u|$) to cover as many vertices of Φ_u as possible, and then it covers all remaining uncovered vertices of Φ_u by using the fewest number of disjoint segments. Thus, the total number of segments generated for Φ_u is at most $2\alpha \log |\Phi_u| + 1$, which is independent of γ. Figure 12 shows the running time of SS-MSBE by varying γ from 0.1 to 0.9. We can see that when γ is small (e.g., $\gamma < 0.3$), the performance of SS-MSBE is not good. The main reason is that when γ is small, a steady segment

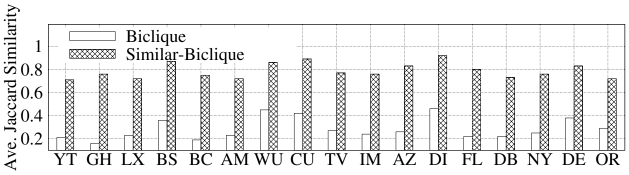
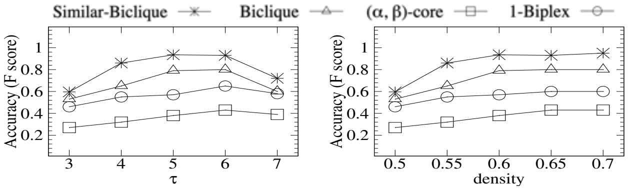


Figure 13: Average Jaccard similarity



(a) Varying τ (b) Varying injected subgraph density

Figure 14: Case study 1: anomaly detection

will cover fewer vertices due to the tighter constraint. As a result, more vertices need to be covered by the ordinary segments, which then results in introducing more fake vertices. Also, when γ is large, the performance of SS-MSBE becomes worse. This is because for large γ (e.g., $\gamma = 1$), a steady segment is no longer steady and degenerates to the ordinary segment. This motivates us to introduce steady segment. We recommend the value of γ to be in $[0.3, 0.5]$.

5.2 Effectiveness Evaluations

Average Jaccard similarity. We compare the average Jaccard similarity between L-side vertices in a maximal (similar-)biclique. Specifically, for each maximal (similar-)biclique C, we compute the average of the Jaccard similarity between all pairs of vertices from C_L, and then the average result of all maximal (similar-)bicliques is reported in Figure 13. We can see that vertices in a similar-biclique are much more similar to each other than in a biclique.

Case study 1: anomaly detection. We compare similar-biclique with other dense bipartite subgraph models, biclique, (α, β)-core [24] and k-biplex [52], on anomaly detection in e-commerce applications. As mentioned in the Introduction, to improve the ranking of certain products, e-business owners may employ a set of fraudulent users to purchase a set of designated products. The fraudsters will also purchase other honest products trying to look “normal”; this is called “camouflage” in the literature. We consider a camouflage attack in the same way as [19] on “Amazon Review Data” (Magazine Subscriptions) [6], which contains $65,546$ reviews on $2,316$ magazines by $53,617$ users, by injecting 100 fraudulent users and 100 fraudulent products with various edge densities. The amount of camouflage (i.e., edges linking to honest products) added per fraudulent user is equal to the amount of fraudulent edges for that user. We adopt F-score, $\frac{2\times\text{precision}\times\text{recall}}{\text{precision}+\text{recall}}$, to evaluate the accuracy of detecting suspicious users and products. We apply the size constraint τ to all the models, where $\alpha = \beta = \tau$ for the (α, β)-core model; for our similar-biclique model, ε is set as 0.2. The results by varying τ and varying the density of the injected subgraph are shown in Figure 14. We can see that similar-biclique always achieves the highest accuracy. This is due to the similarity constraint imposed on users by similar-biclique, which naturally captures the reality that fraudulent users usually display a high level of synchronized behavior with each other. In contrast, biclique, 1-biplex, and (α, β)-core all have a low precision and thus low F-score.

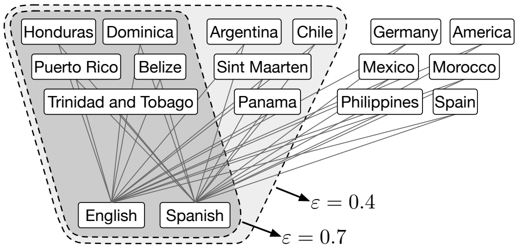


Figure 15: Case study 2: similar-bicliques in Unicode ($\tau = 2$)

Case study 2: interesting pattern detection on Unicode. We also conduct a case study on the Unicode dataset [25] to illustrate the hierarchical structure of similar-bicliques by varying the similarity threshold ε. Unicode captures the languages that are spoken in a country. The three similar-bicliques detected for $\varepsilon = 0.7$, 0.4, 0.01 are reported in Figure 15, where the entire result corresponds to $\varepsilon = 0.01$; the similarity constraint is imposed on the countries and $\tau = 2$. We have the following observations. Firstly, the five countries in the similar-biclique for $\varepsilon = 0.7$ are all located in the Caribbean Sea Area with English and Spanish being their main language (around 90% population speak English and Spanish). Secondly, more countries from Latin America, e.g. Argentina and Chile, are included in the similar-biclique for $\varepsilon = 0.4$, and the newly added four countries speak more diverse languages. For example, in Sint Maarten, besides English and Spanish, around 8% population speak Virgin Islands Creole English and 4% population speak Dutch [7]. Lastly, when ε is 0.01, similar-biclique degenerates to biclique, and more countries are included, e.g., America and Germany. This demonstrates that similar-biclique can detect interesting patterns.

6 CONCLUSION

In this paper, we formulated the notion of similar-biclique, and proposed algorithms as well as optimization techniques to enumerate all similar-bicliques in a bipartite graph. Besides, index structures are also designed to speed up the computation. Extensive empirical studies on real bipartite graphs demonstrated the effectiveness of our similar-biclique model and the efficiency of our algorithms. Case studies show that the similar-biclique model can be used to detect anomalies as well as interesting dense subgraph patterns. Our work initiates the study of integrating similarity constraint into dense bipartite subgraph mining, by taking the biclique model. For future studies, it will be interesting to integrate similarity constraint into other dense bipartite subgraph models, such as quasi-biclique, k-biplex, (α, β)-core, k-bitruss and k-wing. We believe that our proposed index structures will also be useful for these extensions.

ACKNOWLEDGMENTS

This work was supported by the Australian Research Council Fundings of FT180100256 and DP220103731, and the Research Grants Council of Hong Kong, China under No. 14203618, No. 14202919 and No. 14205520.

[6] https://nijianmo.github.io/amazon/index.html

[7] https://www.unicode.org/cldr/cldr-aux/charts/25/summary/root.html

REFERENCES

[1] [n.d.]. full version: https://lijunchang.github.io/pdf/2022-msbe-tr.pdf.

[2] Aman Abidi, Rui Zhou, Lu Chen, and Chengfei Liu. 2020. Pivot-based Maximal Biclique Enumeration.. In *IJCAI*. 3558–3564.

[3] Lada A Adamic and Eytan Adar. 2003. Friends and neighbors on the web. *Social networks* 25, 3 (2003), 211–230.

[4] Gabriela Alexe, Sorin Alexe, Yves Crama, Stephan Foldes, Peter L Hammer, and Bruno Simeone. 2004. Consensus algorithms for the generation of all maximal bicliques. *Discrete Applied Mathematics* 145, 1 (2004), 11–21.

[5] Mohammad Allahbakhsh, Aleksandar Ignjatovic, Boualem Benatallah, Seyed-Mehdi-Reza Beheshti, Elisa Bertino, and Norman Foo. 2013. Collusion detection in online rating systems. In *Asia-Pacific Web Conference*. Springer, 196–207.

[6] Coen Bron and Joep Kerbosch. 1973. Algorithm 457: finding all cliques of an undirected graph. *Commun. ACM* 16, 9 (1973), 575–577.

[7] Lijun Chang, Wei Li, Lu Qin, Wenjie Zhang, and Shiyu Yang. 2017. pSCAN: Fast and Exact Structural Graph Clustering. *IEEE Trans. Knowl. Data Eng.* 29, 2 (2017), 387–401.

[8] Lijun Chang, Jeffrey Xu Yu, and Lu Qin. 2013. Fast Maximal Cliques Enumeration in Sparse Graphs. *Algorithmica* 66, 1 (2013), 173–186.

[9] James Cheng, Linhong Zhu, Yiping Ke, and Shumo Chu. 2012. Fast algorithms for maximal clique enumeration with limited memory. In *Proc. of KDD'12*. 1240–1248.

[10] Vacha Dave, Saikat Guha, and Yin Zhang. 2013. Viceroi: Catching click-spam in search ad networks. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 765–776.

[11] Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. 2008. *Computational geometry: algorithms and applications, 3rd Edition*. Springer.

[12] Lee R Dice. 1945. Measures of the amount of ecologic association between species. *Ecology* 26, 3 (1945), 297–302.

[13] 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.

[14] Radia EL BACHA and Thi Thi Zin. 2018. Ranking of influential users based on user-tweet bipartite graph. In *2018 IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI)*. IEEE, 97–101.

[15] David Eppstein. 1994. Arboricity and bipartite subgraph listing algorithms. *Information processing letters* 51, 4 (1994), 207–211.

[16] David Eppstein, Maarten Löffler, and Darren Strash. 2010. Listing all maximal cliques in sparse graphs in near-optimal time. In *International Symposium on Algorithms and Computation*. Springer, 403–414.

[17] Siva Charan Reddy Gangireddy, Cheng Long, and Tanmoy Chakraborty. 2020. Unsupervised fake news detection: A graph-based approach. In *Proceedings of the 31st ACM conference on hypertext and social media*. 75–83.

[18] Dorit S Hochbaum. 1996. Approximating covering and packing problems: set cover, vertex cover, independent set, and related problems. In *Approximation algorithms for NP-hard problems*. 94–143.

[19] Bryan Hooi, Hyun Ah Song, Alex Beutel, Neil Shah, Kijung Shin, and Christos Faloutsos. 2016. Fraudar: Bounding graph fraud in the face of camouflage. In *Proc. of KDD'16*.

[20] Paul Jaccard. 1901. Distribution de la flore alpine dans le bassin des Dranses et dans quelques régions voisines. *Bull Soc Vaudoise Sci Nat* 37 (1901), 241–272.

[21] Glen Jeh and Jennifer Widom. 2002. Simrank: a measure of structural-context similarity. In *Proc. of KDD'02*. 538–543.

[22] Meng Jiang, Peng Cui, Alex Beutel, Christos Faloutsos, and Shiqiang Yang. 2014. Catchsync: catching synchronized behavior in large directed graphs. In *Proc. of KDD'14*. 941–950.

[23] Leo Katz. 1953. A new status index derived from sociometric analysis. *Psychometrika* 18, 1 (1953), 39–43.

[24] Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew Tomkins. 1999. Trawling the web for emerging cyber-communities. *Computer networks* 31, 11-16 (1999), 1481–1493.

[25] Jérôme Kunegis. 2013. Konect: the koblenz network collection. In *Proceedings of the 22nd international conference on world wide web*. 1343–1350.

[26] Sune Lehmann, Martin Schwartz, and Lars Kai Hansen. 2008. Biclique communities. *Physical review E* 78, 1 (2008), 016108.

[27] Elizabeth A Leicht, Petter Holme, and Mark EJ Newman. 2006. Vertex similarity in networks. *Physical Review E* 73, 2 (2006), 026120.

[28] Michael Ley. 2002. The DBLP computer science bibliography: Evolution, research issues, perspectives. In *International symposium on string processing and information retrieval*. Springer, 1–10.

[29] Jinyan Li, Haiquan Li, Donny Soh, and Limsoon Wong. 2005. A correspondence between maximal complete bipartite subgraphs and closed patterns. In *European Conference on Principles of Data Mining and Knowledge Discovery*. Springer, 146–156.

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

[31] David Liben-Nowell and Jon Kleinberg. 2007. The link-prediction problem for social networks. *Journal of the American society for information science and technology* 58, 7 (2007), 1019–1031.

[32] Zhenjiang Lin, Michael R Lyu, and Irwin King. 2012. MatchSim: a novel similarity measure based on maximum neighborhood matching. *Knowledge and information systems* 32, 1 (2012), 141–166.

[33] Guimei Liu, Kelvin Sim, and Jinyan Li. 2006. Efficient mining of large maximal bicliques. In *International Conference on Data Warehousing and Knowledge Discovery*. Springer, 437–448.

[34] Xiaowen Liu, Jinyan Li, and Lusheng Wang. 2008. Quasi-bicliques: Complexity and binding pairs. In *International Computing and Combinatorics Conference*. Springer, 255–264.

[35] Linyuan Lü and Tao Zhou. 2011. Link prediction in complex networks: A survey. *Physica A: statistical mechanics and its applications* 390, 6 (2011), 1150–1170.

[36] Bingqing Lyu, Lu Qin, Xuemin Lin, Ying Zhang, Zhengping Qian, and Jingren Zhou. 2020. Maximum biclique search at billion scale. *Proceedings of the VLDB Endowment* (2020).

[37] Kazuhisa Makino and Takeaki Uno. 2004. New algorithms for enumerating all maximal cliques. In *Scandinavian workshop on algorithm theory*. Springer, 260–272.

[38] Víctor Martínez, Fernando Berzal, and Juan-Carlos Cubero. 2016. A survey of link prediction in complex networks. *ACM computing surveys (CSUR)* 49, 4 (2016), 1–33.

[39] Nimrod Megiddo, Eitan Zemel, and S Louis Hakimi. 1983. The maximum coverage location problem. *SIAM Journal on Algebraic Discrete Methods* 4, 2 (1983), 253–261.

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

[41] Gerard Salton. 1989. Automatic text processing: The transformation, analysis, and retrieval of. *Reading: Addison-Wesley* 169 (1989).

[42] Michael J Sanderson, Amy C Driskell, Richard H Ree, Oliver Eulenstein, and Sasha Langley. 2003. Obtaining maximal concatenated phylogenetic data sets from large sequence databases. *Molecular biology and evolution* 20, 7 (2003), 1036–1042.

[43] Ahmet Erdem Sarıyüce and Ali Pinar. 2018. Peeling bipartite networks for dense subgraph discovery. In *Proc. of WSDM'18*. 504–512.

[44] Venu Satuluri, Srinivasan Parthasarathy, and Yiye Ruan. 2011. Local graph sparsification for scalable clustering. In *Proc. of SIGMOD'11*. 721–732.

[45] Xiaoyuan Su and Taghi M Khoshgoftaar. 2009. A survey of collaborative filtering techniques. *Advances in artificial intelligence* 2009 (2009).

[46] Etsuji Tomita, Akira Tanaka, and Haruhisa Takahashi. 2006. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical computer science* 363, 1 (2006), 28–42.

[47] Tom Tseng, Laxman Dhulipala, and Julian Shun. 2021. Parallel Index-Based Structural Graph Clustering and Its Approximation. In *Proc. of SIGMOD'21*. 1851–1864.

[48] Takeaki Uno, Masashi Kiyomi, Hiroki Arimura, et al. 2004. LCM ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. In *Fimi*, Vol. 126.

[49] Jun Wang, Arjen P De Vries, and Marcel JT Reinders. 2006. Unifying user-based and item-based collaborative filtering approaches by similarity fusion. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*. 501–508.

[50] Xiaodong Wang and Jing Liu. 2018. A comparative study of the measures for evaluating community structure in bipartite networks. *Information Sciences* 448 (2018), 249–262.

[51] Seok-Ho Yoon, Sang-Wook Kim, and Sunju Park. 2016. C-Rank: A link-based similarity measure for scientific literature databases. *Information sciences* 326 (2016), 25–40.

[52] Kaiqiang Yu, Cheng Long, P Deepak, and Tanmoy Chakraborty. 2021. On Efficient Large Maximal Biplex Discovery. *IEEE Transactions on Knowledge and Data Engineering* (2021).

[53] Mohammed J Zaki and Ching-Jui Hsiao. 2002. CHARM: An efficient algorithm for closed itemset mining. In *Proceedings of the 2002 SIAM international conference on data mining*. SIAM, 457–473.

[54] 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.

[55] Zhong-Yuan Zhang and Yong-Yeol Ahn. 2015. Community detection in bipartite networks using weighted symmetric binary matrix factorization. *International Journal of Modern Physics C* 26, 09 (2015), 1550096.

[56] Peixiang Zhao, Jiawei Han, and Yizhou Sun. 2009. P-rank: a comprehensive structural similarity measure over information networks. In *Proceedings of the 18th ACM conference on Information and knowledge management*. 553–562.

[57] Zhaonian Zou. 2016. Bitruss decomposition of bipartite graphs. In *International Conference on Database Systems for Advanced Applications*. Springer, 218–233.