Efficient Algorithms for Maximal k-Biplex Enumeration

ABSTRACT

Mining maximal subgraphs with cohesive structures from a bipartite graph has been widely studied. One important cohesive structure on bipartite graphs is k-biplex, where each vertex on one side disconnects at most k vertices on the other side. In this paper, we study the *maximal k-biplex enumeration* problem which enumerates all maximal k-biplexes. Existing methods suffer from efficiency and/or scalability issues and have the time of waiting for the next output exponential w.r.t. the size of the input bipartite graph (i.e., an exponential delay). In this paper, we adopt a reverse search framework called `bTraversal`, which corresponds to a depth-first search (DFS) procedure on an implicit solution graph on top of all maximal k-biplexes. We then develop a series of techniques for improving and implementing this framework including (1) carefully selecting an initial solution to start DFS, (2) pruning the vast majority of links from the solution graph of `bTraversal`, and (3) implementing abstract procedures of the framework. The resulting algorithm is called `iTraversal`, which has its underlying solution graph significantly sparser than (around 0.1% of) that of `bTraversal`. Besides, `iTraversal` provides a guarantee of polynomial delay. Our experimental results on real and synthetic graphs, where the largest one contains more than one billion edges, show that our algorithm is up to four orders of magnitude faster than existing algorithms.

CCS CONCEPTS

• **Mathematics of computing** → Graph algorithms.

KEYWORDS

Bipartite graph, maximal biplex, maximal subgraph enumeration

ACM Reference Format:

. 2018. Efficient Algorithms for Maximal k-Biplex Enumeration. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 24 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

In many applications, two types of entities are involved and interact with each other. Some examples include (1) social media where users comment on articles [47], (2) e-commerce services where customers post reviews on or purchase products [41], (3) collaboration networks where authors publish papers [26], etc. In these applications, the two types of entities and the interactions between them can be naturally modelled as a *bipartite graph* with the entities being vertices and the interactions being edges.

For a given bipartite graph, a *dense/cohesive* subgraph within it usually carries interesting information that can be used for solving practical problems such as fraud detection [17], online recommendation [18, 36] and community search [19, 44]. For example, in social networking applications, when a group of users are paid to promote a specific set of fake articles via retweets, the induced subgraph by these users and the articles would be dense. Identifying such dense subgraphs would help to detect the fake users and articles [17]. As a second example, in e-commerce services, after identifying a dense subgraph between customers and products, it would be natural to recommend products to those customers which disconnect the products within the subgraph [18, 36].

Quite a few definitions have been proposed for a dense bipartite graph, including biclique [48], (α, β)-core [28], k-bitruss [43], k-biplex [37, 46], δ-quasi-biclique [30], etc. These definitions impose different requirements on the connections and/or disconnections within a subgraph. For example, biclique requires that any vertex from one side *connects all* vertices from the other side, (α, β)-core requires that any vertex from one side *connects* at least *a certain* number of vertices from the other side, and k-biplex requires that any vertex from one side *disconnects at most k* vertices from the other side, where k is a small positive integer specified by users.

In this paper, we study the problem of enumerating *Maximal k-BiPlexes*, called *MBP*s, for the following considerations. First, k-biplex imposes a strict enough requirement on the connections within a subgraph yet allows some disconnections which are common in real applications (due to data quality issues such as incomplete data). Second, k-biplex satisfies the *hereditary property* [10], i.e., any subgraph of a k-biplex is still a k-biplex. This can be utilized to design efficient frameworks for enumerating MBPs (details will be discussed later). Third, other definitions have some shortcomings when used for the aforementioned applications. Specifically, (1) biclique may impose a too strict requirement on the connections, i.e., with even one single connection missed from a biclique, the subgraph is no longer a biclique. (2) (α, β)-core and k-bitruss can be computed efficiently, but they impose no constraints on the disconnections within a subgraph, i.e., a vertex may disconnect many vertices from the other side. (3) δ-quasi-biclique does not satisfy the hereditary property so enumerating the corresponding maximal structures is much harder than enumerating MBPs. We have conducted some case studies, which show that k-biplex works better for fraud detection on e-commerce platforms and captures more cohesive subgraphs than other definitions including biclique and (α, β)-core (details will be presented in Section 6.3).

There are several existing methods, which can be used or adapted for enumerating MBPs. The first one is called `iMB` [37, 46], which uses two prefix trees to organize the subsets of vertices of the two sides and searches MBPs with backtracking and various pruning strategies. Nevertheless, it suffers from two issues: (1) it is not scalable and cannot handle big graphs since most of its pruning

Woodstock '18, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

techniques rely highly on some size constraints imposed on the k-biplexes to enumerate; (2) the delay, which represents the amount of waiting time for the next MBP to return or for the termination of the algorithm, is exponential w.r.t. the number of vertices of the bipartite graph. The second baseline is based on *graph inflation*. Specifically, it first inflates a given bipartite graph into a general one by including an edge between every pair of two vertices at the same side and then enumerates all maximal $(k+1)$-plexes on the inflated general graphs (for which, the state-of-the-art is `FaPlexen` [49]). Here, a $(k+1)$-plex on a general graph represents a subgraph, where each vertex v disconnects at most $(k+1)$ vertices (including v) within the subgraph [4]. The correctness of this method is based on the fact that a k-biplex on the bipartite graph corresponds to a $(k+1)$-plex on the inflated general graph. Nevertheless, the graph inflation step would usually generate very dense graphs and enumerating $(k+1)$-plexes on a dense graph is rather time-consuming.

In this paper, we adopt a *reverse search* framework [10] which we call `bTraversal` for enumerating MBPs. `bTraversal` is originally designed for enumerating maximal subgraph structures that satisfy the hereditary property (each such structure is called a *solution*). The key insight is that given a solution H, it is possible to find another solution by *excluding* some existing vertices from H and then *including* some new ones to H. Specifically, `bTraversal` involves two steps. First, it finds one solution H_0 as the initial one, which can be any one among all solutions. Second, it finds solutions from H_0 via a procedure of excluding and including vertices from and to H_0 and then recursively performs the procedure from those found solutions until no new solutions are found. Suppose we take each solution as a vertex and create a directed edge from solution H to solution H' if `bTraversal` can find H' from H via the aforementioned procedure of excluding and including vertices. Then, we obtain a graph structure on top of all found solutions, which is called a *solution graph* [14]. We refer to the vertices and directed edges in the solution graph as nodes and links, respectively. Then, `bTraversal` corresponds to a *depth-first search* (DFS) procedure over the solution graph. According to [14], the solution graph constructed in this way is *strongly connected* and thus `bTraversal` is able to enumerate all solutions starting from any solution.

Nevertheless, `bTraversal` is still insufficient in the following aspects. First, `bTraversal` requires that any solution should be reachable from any other solution in the solution graph (i.e., the solution graph is strongly connected) so that it can find *all* solutions from *any* initial solution. To fulfill this requirement, it would find many solutions from one solution. Consequently, the underlying solution graph tends to be dense and the DFS procedure on the solution graph would be costly. Note that the time complexity of DFS is proportional to the number of links in the solution graph. Second, `bTraversal` is originally designed for general structures that satisfy the hereditary property, but not just for the k-biplexes. Consequently, it overlooks those unique characteristics of k-biplexes that would otherwise help to improve the algorithm. Third, `bTraversal` does not support enumerating MBPs with size at least a threshold (called large MBPs) - it has to enumerate all MBPs first and then filter out the MBPs violating the size constraint, which is inefficient.

We observe that the requirement of a strongly connected solution graph by `bTraversal` is stronger than necessary. In fact, it would be sufficient as long as all solutions are reachable from some *specific* solution since we can then start the DFS procedure from this solution to reach all solutions. Motivated by this observation, in this paper, we propose an improved framework called `iTraversal`, which begins DFS from a carefully selected solution. Specifically, it selects $H_0 = (L_0, \mathcal{R})$ as the initial solution, where $\mathcal{R}$ is the set containing all vertices from the right side of the bipartite graph and L_0 is any maximal set of vertices from the left side with $(L_0, \mathcal{R})$ being a k-biplex. With this designated initial solution, `iTraversal` makes it possible to significantly sparsify the solution graph that is defined by `bTraversal` while maintaining that all solutions are reachable from this initial solution. Specifically, we develop a series of three techniques for sparsifying the solution graph, namely (1) left-anchored traversal, (2) right-shrinking traversal, and (3) exclusion strategy. Technique (1) is based on a discovery that in one step of *including* vertices to generate new solutions from a certain solution, pruning all vertices on the right side from being included would still guarantee that all solutions are reachable from our initial solution. Technique (2) is based on another discovery that by retaining only those links from a solution $H = (L, R)$ to another solution $H' = (L', R')$ with $R' \subseteq R$ and removing all the other links, all solutions are still reachable from the initial solution. Technique (3) is a technique that prunes a vertex from being included to find new solutions from a solution if the vertex appears in an *exclusion set* that is maintained during the running of `iTraversal`. Based on our experimental results, the number of links in the solution graph of `iTraversal` sparsified with the three techniques is about 0.1% of that in the original solution graph of `bTraversal`.

In summary, our major contributions are summarized as follows.

- We propose a new framework `iTraversal` for enumerating MBPs. We further develop three techniques, namely left-anchored traversal, right-shrinking traversal, and exclusion strategy, for sparsifying the solution graph under `iTraversal`. We prove that the delay of finding the next solution with `iTraversal` is *polynomial* w.r.t. the number of vertices and improves that of the conventional `bTraversal` framework (Section 3). We remark that (1) the first two techniques are novel and work only with `iTraversal` but not with `bTraversal` and (2) the third technique was proposed for `bTraversal` but its correctness for `iTraversal`, which is not trivial to prove, is verified in this paper.
- We design an efficient algorithm for a key procedure that is involved in the `iTraversal` framework, which is to enumerate solutions within a graph that *almost* satisfies the definition of k-biplex (Section 4).
- We extend `iTraversal` to enumerate those MBPs with the size of at least a threshold (i.e., large MBPs) without enumerating all MBPs, which is not possible when using the conventional `bTraversal` framework (Section 5).
- We conduct extensive experiments on both real and synthetic datasets, which verify that (1) k-biplex works better in a fraud detection task than some other structures including biclique and (α, β)-core and captures cohesive subgraphs and (2) the proposed algorithms with new techniques are up to four orders of magnitude faster than existing algorithms including the one based on `bTaversal` (Section 6).

Among other sections, we define our problem in Section 2, review the related work in Section 7 and conclude the paper in Section 8.

2 PROBLEM DEFINITION

In this paper, we consider an undirected and unweighted bipartite graph $G = (\mathcal{L} \cup \mathcal{R}, \mathcal{E})$ with two disjoint vertex sets $\mathcal{L}$, $\mathcal{R}$ and an edge set $\mathcal{E}$. $\mathcal{L}$ and $\mathcal{R}$ are supposed to be on the left and right side, respectively. We denote by $V(G)$ the set of vertices in G, i.e., $V(G) = \mathcal{L} \cup \mathcal{R}$, and by $E(G)$ the set of edges in G, i.e., $E(G) = \mathcal{E}$. Given $L \subseteq \mathcal{L}$ and $R \subseteq \mathcal{R}$, the *induced (bipartite) subgraph* $G[L \cup R]$ of G consists of the set of vertices $L \cup R$ and the set of edges between L and R. Note that all subgraphs mentioned in this paper refer to induced subgraphs, and we use H or (L, R) as a shorthand of $H = G[L \cup R]$.

Given $v \in \mathcal{L}$ and $R \subseteq \mathcal{R}$, we define $\Gamma(v, R)$ (resp. $\overline{\Gamma}(v, R)$) to be the set of vertices that are in R and connect (resp. disconnect) v, i.e., $\Gamma(v, R) = \{u \mid (v, u) \in \mathcal{E} \text{ and } u \in R\}$ (resp. $\overline{\Gamma}(v, R) = \{u \mid (v, u) \notin \mathcal{E} \text{ and } u \in R\}$). Note that $\Gamma(v, R) \cup \overline{\Gamma}(v, R) = R$. In addition, we define $\delta(v, R) = |\Gamma(v, R)|$ and $\overline{\delta}(v, R) = |\overline{\Gamma}(v, R)|$. Given $u \in \mathcal{R}$ and $L \subseteq \mathcal{L}$, $\Gamma(u, L)$ (resp. $\overline{\Gamma}(u, L)$) and $\delta(u, L)$ (resp. $\overline{\delta}(u, L)$) are similarly defined. Next, we introduce the cohesive bipartite structure k-biplex that is exploited in this paper.

Definition 2.1 (k-biplex [37]). Let k be a positive integer. An induced subgraph $G[L \cup R]$ of a bipartite graph G is said to be a *k-biplex* if (1) $\overline{\delta}(v, R) \leq k, \forall v \in L$ and (2) $\overline{\delta}(u, L) \leq k, \forall u \in R$.

For a k-biplex, parameter k represents the number of missing edges that each vertex in $G[L \cup R]$ can tolerate. We note that it is possible to use different k's at different sides and the techniques developed in this paper can be easily adapted to this case. Moreover, the k-biplex structures satisfy the *hereditary property* [10], which we present in the following lemma.

Lemma 2.2 (Hereditary Property). *If $H = (L, R)$ is a k-biplex, any subgraph $H' = (L', R')$ of H with $L' \subseteq L, R' \subseteq R$ is a k-biplex.*

This can be easily verified by the fact that with some vertices excluded from a k-biplex, each remaining vertex has the number of disconnections non-increasing, i.e., still bounded by k.

As there might exist a large number of k-biplexes, one common practice is to return a compact representation of the set of all k-biplexes, namely the set of all *maximal k-biplexes*.

Definition 2.3 (Maximal k-biplex). A k-biplex $G[L \cup R]$ is said to be *maximal* if and only if there is no other k-biplex $G[L' \cup R']$ which is a superset of $G[L \cup R]$ (i.e., $L \subseteq L'$ and $R \subseteq R'$).

In this paper, we use MBP as a shorthand of maximal k-biplex when the context is clear. We are ready to formalize the problem exploited in this paper:

Problem 1 (Maximal k-biplex Enumeration [37]). *Given a bipartite graph $G = (\mathcal{L} \cup \mathcal{R}, \mathcal{E})$ and a positive integer k, the* Maximal k-biplex Enumeration Problem *aims to report all MBPs.*

We call each maximal induced subgraph of graph G that is a k-biplex as a *solution*. The maximal k-biplex enumeration problem is to enumerate all solutions. We use the terms "maximal k-biplex (MBP)" and "solution" interchangeably throughout this paper.

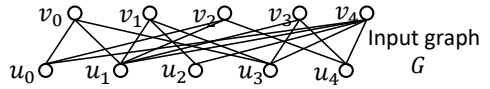


Figure 1: Input graph used throughout the paper.

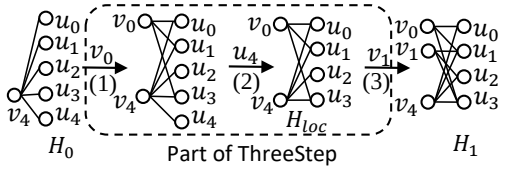


Figure 2: Illustration of the `ThreeStep` procedure.

3 THE ITRAVERSAL ALGORITHM

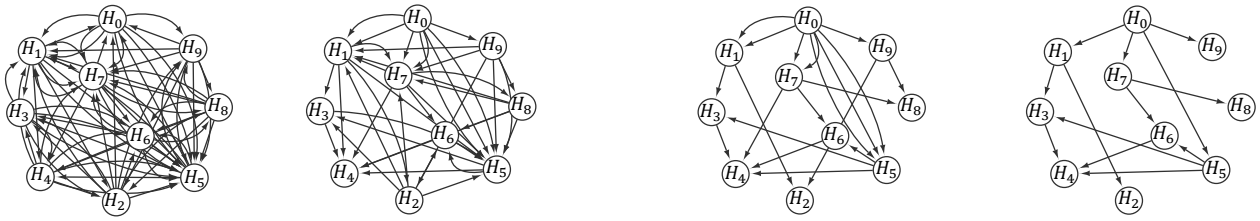
We adopt a *reverse search* framework which we call `bTraversal` [10] for enumerating MBPs. `bTraversal` is a framework for enumerating maximal subgraph structures that satisfy the hereditary property. In the sequel, we review `bTraversal` in Section 3.1. We observe that `bTraversal` imposes a requirement that is more demanding than necessary and we relax it to achieve an improved framework called `iTraversal` in Section 3.2. We then develop a series of three techniques, namely left-anchored traversal (Section 3.3), right-shrinking traversal (Section 3.4), and exclusion strategy (Section 3.5) for further boosting `iTraversal`'s performance. Finally, we present a summary of `iTraversal` and its running time and delay in Section 3.5.

3.1 The Basic Framework: `bTraversal`

The key insight of `bTraversal` is that given a solution H (which corresponds to a set of vertices), it is possible to find another solution by *excluding* some existing vertices from and *including* some new ones to H. The rationale is that (1) due to the hereditary property, H is a k-biplex so it will still be a k-biplex after some vertices are excluded; and (2) after some vertices are excluded from H, it becomes possible to include some new vertices to H while retaining the property and thus it finds another solution.

Specifically, `bTraversal` first finds one solution H_0 as the initial one, which can be any one among all solutions. This can be achieved easily, say, by iteratively including vertices to an initially empty set while retaining the k-biplex property until it is not possible to do so. It then finds solutions from H_0 via a procedure of excluding and including vertices from and to H_0 and recursively performs the procedure from those solutions found for finding more solutions until no new solutions are found. It uses the following three-step procedure for finding solutions from one solution H, which we call `ThreeStep`.

- **Step 1 (Almost-satisfying graph formation).** For each vertex $v \in V(G) \backslash V(H)$, it forms a new induced subgraph $G[V(H) \cup \{v\}]$ (or simply $G[H, v]$) by including v to H. We call each such graph $G[H, v]$ an *almost-satisfying graph* since it is not a k-biplex (since otherwise H is not maximal) and would be so if one vertex, i.e., v, is excluded.
- **Step 2 (Local solution enumeration).** For each almost-satisfying graph $G[H, v]$, it enumerates all (induced) subgraphs H_{loc} of $G[H, v]$ that (1) involve v, (2) are k-biplexes, and (3) are *maximal* w.r.t. $G[H, v]$ (which means that there exists no vertex $u \in V(H) \cup \{v\} \backslash V(H_{loc})$ such that $G[H_{loc}, u]$ is a k-biplex). Essentially, it solves the MBP enumeration problem with the input



(a) bTraversal ($\mathcal{G}$) (b) Left-anchored traversal ($\mathcal{G}_L$) (c) Right-shrinking traversal ($\mathcal{G}_R$) (d) iTraversal ($\mathcal{G}_E$)

Figure 3: Solution graphs underlying different algorithms (based on the input graph in Figure 1).

of $G[H, v]$, which should be much easier than the original one with the input G. We call such a subgraph H_{loc} a *local solution* since it is maximal locally w.r.t. $G[H, v]$ and may not be maximal w.r.t. G. We call the procedure of enumerating all local solutions within an almost-satisfying graph $G[H, v]$ `EnumAlmostSat`. In Section 4, we present an implementation of `EnumAlmostSat`.

- **Step 3 (Local solution extension).** For each local solution H_{loc}, it extends H_{loc} to a real solution H' (i.e., H' is maximal w.r.t. G), by iteratively including to H_{loc} vertices from outside H_{loc} until H_{loc} becomes maximal w.r.t. G. We remark that during this step, for each local solution H_{loc}, it is extended to only one real solution H', e.g., it includes vertices to H_{loc} by following a pre-set order on all vertices.

Example 3.1. Consider the `ThreeStep` for finding H_1 from H_0 in Figure 2 with $k = 1$. We (1) form an almost-satisfying graph $G[H_0, v_0]$ by including v_0 to H_0, (2) find a local solution H_{loc} by excluding u_4 from $G[H_0, v_0]$ and (3) extend H_{loc} to H_1 by including v_1 to H_{loc} while retaining the k-biplex property.

We summarize the `bTraversal` algorithm in Algorithm 1. A solution may be traversed from multiple solutions. To avoid duplication, a B-tree is used for storing those solutions that have been found, where the key of a solution is specified by the vertices of the solution (Line 1 and 7-8).

Algorithm 1: The algorithm: `bTraversal`.

```
Input: Bipartite graph G = (L ∪ R, E), integer k ≥ 1;
Output: All maximal k-biplexes;
1 Initialize H_0 as any maximal k-biplex, B-tree T = {H_0};
2 ThreeStep(G, H_0, T);
3 Procedure ThreeStep(G, H, T)
4     (Step 1) foreach v in V(G)\V(H) do
5         (Step 2) foreach H_loc in EnumAlmostSat(G[H, v]) do
6             (Step 3) Extend H_loc to be a maximal k-biplex H'
                with vertices from V(G)\V(H_loc);
7             if H' ∉ T then
8                 Insert H' to T;
9                 ThreeStep(G, H', T);
```

Suppose we take each solution as a node and create a directed edge from solution H to solution H' if `bTraversal` can find H' from H via the above three-step procedure. Then, we obtain a graph structure on top of all solutions. This graph structure is called a *solution graph* [14], which we denote by $\mathcal{G}$. We note that a solution graph is a multi-graph since from one solution H, `bTraversal` may find another solution H' by forming different almost-satisfying graphs. We refer to the vertices and directed edges in the solution graph as nodes and links, respectively, and reserve the former notions for those in the graph G. Then, `bTraversal` corresponds to a *depth-first search* (DFS) procedure over the solution graph $\mathcal{G}$. According to [14], the solution graph $\mathcal{G}$ is *strongly connected* and thus `bTraversal` is able to enumerate all solutions starting from any solution. To illustrate, consider the input graph in Figure 1 with $k = 1$. The corresponding solution graph is shown in Figure 3(a), which is strongly connected with 10 solutions and 76 links.

3.2 An Improved Framework: `iTraversal`

`bTraversal` makes a requirement that any solution is reachable from any other solution in the solution graph (i.e., the solution graph is strongly connected) so that it can find *all* solutions from *any* initial solution. To fulfill this requirement, it would find many solutions from one solution. To see this, consider the above three-step procedure `ThreeStep`, where $O(|V(G)|)$ almost-satisfying graphs are formed (Step 1), for each almost-satisfying graph, an exponential number of local solutions are enumerated (Step 2), and each local solution is extended to a real solution (Step 3). Consequently, the underlying solution graph would be dense and the DFS procedure on the solution graph would be costly. Note that the time complexity of DFS is proportional to the number of links in the solution graph.

We observe that the requirement of a strongly connected solution graph by `bTraversal` is stronger than necessary. In fact, it would be sufficient as long as all solutions are reachable from some *specific* solution since we then can start the DFS procedure from this specific solution and reach all solutions. Motivated by this, in this paper, we propose an improved traversal framework called `iTraversal`, which performs the DFS procedure from some *specific* but not *arbitrary* solution on a solution graph. With a designated initial solution, `iTraversal` makes it possible to significantly sparsify the solution graph that is defined by `bTraversal` while maintaining that all solutions are reachable from the initial solution. To illustrate, consider again the example in Figure 1. The solution graph that is defined by `bTraversal` is shown in Figure 3(a), which involves 76 links and is strongly connected. One solution graph that could be defined by `iTraversal` is shown in Figure 3(d) and the initial solution that could be chosen by `iTraversal` is $H_0 = (v_4, \{u_i\}_{i=0}^4)$. All solutions are reachable from H_0 in the sparsified solution graph with 13 links.

One immediate question is: *what is a good initial solution H_0 among all possible solutions?* We consider two desiderata: (1) H_0 can be computed easily and (2) the solution graph defined by `bTraversal` can be sparsified significantly (by dropping some links from the solution graph) while all solutions are still kept reachable from H_0.

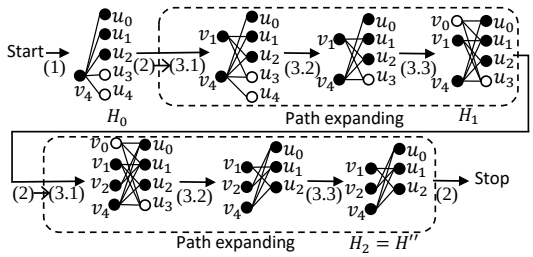


Figure 4: Illustration of the four-step procedure of finding a path involving only left-anchored links, i.e., $\mathcal{P}_L(H'')$ (black nodes denote the vertices in the target solution H'').

Our proposal is to use $H_0 = (L_0, \mathcal{R})$ as the initial solution, where L_0 is any maximal set of vertices from $\mathcal{L}$ with $(L_0, \mathcal{R})$ being a k-biplex [1]. To illustrate, consider the example in Figure 1 with $k = 1$. We obtain $H_0 = (L_0, \mathcal{R})$ where $L_0 = \{v_4\}$ and $\mathcal{R} = \{u_0, u_1, u_2, u_3, u_4\}$. Next, we explain how H_0 meets the two desiderata.

Consider the first desideratum. We can construct H_0 easily as follows. First, we initialize H_0 as $(\emptyset, \mathcal{R})$. Note that $(\emptyset, \mathcal{R})$ is a k-biplex since there are no vertices at the left side (i.e., $\emptyset$) and for each vertex at right side (i.e., $\mathcal{R}$), it disconnects from no vertices from the left side. Second, we extend H_0 by iteratively including vertices from $\mathcal{L}$ while maintaining that H_0 is a k-biplex until this is not possible. At the end, H_0 corresponds to a maximal k-biplex, i.e., a solution. This process would check for each vertex from $\mathcal{L}$ whether it can be included to H_0, which is efficient.

Consider the second desideratum. With $H_0 = (L_0, \mathcal{R})$ as the initial solution, we are able to identify a set of paths to traverse from H_0 to all solutions since H_0 includes $\mathcal{R}$ and can reach every solution by iteratively including vertices from the left side of the target solution and excluding vertices that are not in the target solution. Hence, we can drop a large amount of links that do not appear along any of these paths (details are in Section 3.3 and Section 3.4). In this way, the solution graph $\mathcal{G}$ could be sparsified significantly while retaining that all solutions are reachable from H_0, as shown in Figure 3(d).

3.3 `iTraversal`: Left-anchored Traversal

Let $H'' = (L'', R'')$ be any solution that is different from the initial solution $H_0 = (L_0, \mathcal{R})$ and $\mathcal{P} = \langle H_0, H_1, ..., H_n \rangle$ be a path from H_0 to $H_n = H''$ in $\mathcal{G}$. Consider the first link among the path $\mathcal{P}$, i.e., $\langle H_0, H_1 \rangle$. Recall that when finding H_1 from H_0 via the procedure `ThreeStep` in Section 3.1, it first *includes* a vertex $v \in V(G) \backslash V(H_0)$ for forming an almost-satisfying graph (Step 1). We observe that this vertex v is always from the left side since $V(G) \backslash V(H_0) = \mathcal{L} \backslash L_0$. We call such a link, which is formed by including a vertex from the left side for forming an almost-satisfying graph in the procedure `ThreeStep`, as a *left-anchored* link. We note that left-anchored links are defined based on solutions but not intermediate ones (e.g., local solutions) and each link is either a left-anchored link or a non-left-anchored one. To illustrate, consider the link $\langle H_0, H_1 \rangle$ from H_0 to H_1 in Figure 2. It is a left-anchored link since the almost-satisfying graph is formed by including vertex $v_0 \in \mathcal{L}$.

[1] An alternative proposal is $H_0 = (\mathcal{L}, R_0)$ that is defined symmetrically and all techniques proposed in this paper would still apply. These two proposals are symmetric and are evaluated empirically in experiments.

Based on the above discussion, we know that the first link of any path from H_0 to H'' is a left-anchored link. This triggers the following question: *can we always find a path from H_0 to H'', which involves left-anchored links only?* The answer is interestingly positive. In the following, we present a procedure, which defines for any solution H'' a path $\langle H_0, H_1, ..., H_n \rangle$ with $H_n = H''$ in $\mathcal{G}$, which traverses from H_0 to H'' and involves left-anchored links only. We denote this path by $\mathcal{P}_L(H'')$. Specifically, the procedure has four steps and maintains the following invariant (which can be proved by induction).

$$R'' \subseteq R_i, \text{ for } i = 0, 1, ..., n \tag{1}$$

- **Step 1: Path initialization.** Initialize i to be 0. Note that $R'' \subseteq R_i = \mathcal{R}$ (basis step for proving the invariant).
- **Step 2: Termination checking.** If $L'' \backslash L_i = \emptyset$, set $n = i$ and stop.
- **Step 3: Path expanding.** Find another solution H_{i+1} from H_i via a *left-anchored* link in $\mathcal{G}$ as follows:
  - **Step 3.1.** Pick a vertex v in $L'' \backslash L_i$ and form an almost-satisfying graph $G[H_i, v]$.
  - **Step 3.2.** Find a local solution $H'_{i+1} = (L'_{i+1}, R'_{i+1})$ by extending $((L'' \cap L_i) \cup \{v\}, R'')$ to be maximal within $G[H_i, v]$. Note that $((L'' \cap L_i) \cup \{v\}, R'')$ is: (1) a subgraph of $G[H_i, v]$ (since $(L'' \cap L_i) \subseteq L_i$ and $R'' \subseteq R_i$) and (2) a k-biplex (since it is subgraph of H'').
  - **Step 3.3.** Extend H'_{i+1} to be a MBP (within G), which we denote by $H_{i+1} = (L_{i+1}, R_{i+1})$. Note that $R'' \subseteq R'_{i+1} \subseteq R_{i+1}$ (induction step for proving the invariant).
- **Step 4: Repetition.** Increase i by 1 and go to Step 2.

Example 3.2. Given solution $H'' = (L'', R'')$, where $L'' = \{v_1, v_2, v_4\}$ and $R'' = \{u_0, u_1, u_2\}$, based on the input graph in Figure 1 with $k = 1$. We consider a path from $H_0 = (L_0, \mathcal{R})$, where $L_0 = \{v_4\}$, to H'' formed by the above procedure, as shown in Figure 4. For the first round, we (1) initialize the path, (2) check $L'' \backslash L_0 = \{v_1, v_2\}$, (3.1) pick vertex v_1 and form an almost-satisfying graph $G[H_0, v_1]$, (3.2) find a local solution $(\{v_1, v_4\}, \{u_0, u_1, u_2, u_3\})$ which includes $L'' \cap L_0 \cup \{v_1\} = \{v_1, v_4\}$ and R'' and (3.3) extend it to a solution $H_1 = (L_1, R_1)$ where $L_1 = \{v_0, v_1, v_4\}$ and $R_1 = \{u_0, u_1, u_2, u_3\}$. We repeat for the second round, (2) check $L'' \backslash L_1 = \{v_2\}$, (3.1) form an almost-satisfying graph $G[H_1, v_2]$, (3.2) find a local solution $(\{v_1, v_2, v_4\}, \{u_0, u_1, u_2\})$ which includes $L'' \cap L_1 \cup \{v_2\} = \{v_1, v_2, v_4\}$ and R'' and (3.3) extend it to a solution $H_2 = (L_2, R_2)$ where $L_2 = \{v_1, v_2, v_4\}$ and $R_2 = \{u_0, u_1, u_2\}$. Finally, we check $L_2 \backslash L'' = \emptyset$ and get $H_2 = H''$.

Lemma 3.3. *The procedure of finding the path $\mathcal{P}_L(H'')$ for a given solution H'' would always terminate and path $\mathcal{P}_L(H'')$ ends at H'', i.e., $H_n = H''$.*

Proof. We first prove that the procedure would always terminate. To this end, we define a similarity measurement between two MBPs. Given two MBPs $H = (L, R)$ and $H' = (L', R')$, we define the *similarity* between H and H', denoted by $S(H, H')$, as the number of vertices that are shared by H and H', i.e.,

$$S(H, H') = |V(H) \cap V(H')| = |L \cap L'| + |R \cap R'|. \tag{2}$$

We deduce that H_{i+1} shares at least one more vertex with H'' than H_i for $i = 0, 1, ..., n-1$. That is,

$$S(H_{i+1}, H'') \geq S(H_i, H'') + 1, \text{ for } i = 0, 1, ..., n-1. \tag{3}$$

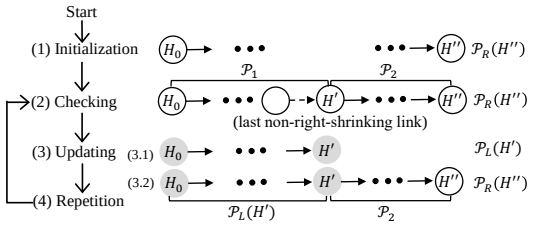


Figure 5: Illustration of the procedure for finding a path involving only right-shrinking links, i.e., $\mathcal{P}_R(H'')$ (the path with gray nodes refers to a new path after updating).

This is because (1) both R_{i+1} and R_i include R'' (based on the invariant of the procedure); and (2) L_{i+1} includes all vertices that are shared by L_i and L'' and at least one vertex v from $L''\backslash L_i$. Therefore, we further deduce that the procedure would always stop since the similarity to H'' increases by at least 1 after each round and it is bounded by $|H''|$.

We then prove that $H_n = H''$ by contradiction. Suppose $H_n \neq H''$. We deduce that H'' would not be an MBP since $H_n = (L_n, R_n)$ is a larger k-biplex containing H'', given (1) $H'' \subseteq H_n$ (since $L''\backslash L_n = \emptyset$ which means $L'' \subseteq L_n$ and $R'' \subseteq R_n$ based on the invariant), (2) $H'' \neq H_n$ based on the assumption, and (3) H_n is a k-biplex. This leads to a contradiction. □

In conclusion, we succeed in finding for any solution H'' a path that traverses from H_0 to H'' and involves left-anchored links only in $\mathcal{G}$. Therefore, we propose to *drop* all non-left-anchored links from $\mathcal{G}$. We denote the resulting solution graph by $\mathcal{G}_L$. For example, $\mathcal{G}_L$ based on the input graph in Figure 1 is shown in Figure 3(b), which involves 41 links and all solutions are reachable from H_0. It is clear that a DFS procedure from H_0 on $\mathcal{G}_L$, which we call the *left-anchored traversal*, would return all solutions. We present this result in the following lemma.

Lemma 3.4. *Given a bipartite graph $G(\mathcal{L} \cup \mathcal{R}, \mathcal{E})$ with an initial MBP $H_0 = (L_0, \mathcal{R})$, the left-anchored traversal enumerates all MBPs.*

Remarks. We remark that the four-step procedure for finding the path $\mathcal{P}_L(H'')$ is a conceptual one for verifying the correctness of the left-anchored traversal only. The implementation of left-anchored traversal will be discussed in Section 3.5. In addition, we note that the sparsified solution $\mathcal{G}_L$ is no longer strongly connected as $\mathcal{G}$ does. To see this, consider that a solution in the form of $(\mathcal{L}, R_0)$ (formed by extending $(\mathcal{L}, \emptyset)$ to be an MBP). There exist no links going from this solution in $\mathcal{G}_L$ since its left side is full. In our experiments, we show that $\mathcal{G}_L$ is significantly sparser than $\mathcal{G}$, e.g., $\mathcal{G}_L$ has about 20× fewer links than $\mathcal{G}$ on average.

3.4 `iTraversal`: Right-shrinking Traversal

Consider a link from $H = (L, R)$ to $H' = (L', R')$ in $\mathcal{G}_L$. We say it is a *right-shrinking link* iff $R' \subseteq R$. Note that we focus on the right side for this definition.

Consider the path $\mathcal{P}_L(H'') = \langle H_0, H_1, ..., H_n \rangle$ from H_0 to a solution H'', where $H'' = H_n$, as defined in Section 3.3. We have two observations: (1) the first link (which is from H_0 to H_1) is a right-shrinking link since $R_1 \subseteq R_0$ and (2) the last link (which is from H_{n-1} to H_n) is also a right-shrinking link since $R_n = R'' \subseteq R_{n-1}$ according to the invariant in Section 3.3. This triggers the following question: *can we reach H'' from H_0 by traversing right-shrinking links only in $\mathcal{G}_L$?* The answer is interestingly also positive.

In the following, we present a procedure, which defines a path in $\mathcal{G}_L$ for any solution H'', which traverses from H_0 to H'' and involves right-shrinking links only. We denote this path by $\mathcal{P}_R(H'')$. Specifically, the procedure has four steps as follows. A visual illustration of the following procedure is shown in Figure 5.

- **Step 1 (Path initialization).** Find the path $\mathcal{P}_L(H'')$ from H_0 to H'' via the four-step procedure in Section 3.3 and initialize $\mathcal{P}_R(H'')$ to be $\mathcal{P}_L(H'')$.
- **Step 2 (Termination checking).** Check if $\mathcal{P}_R(H'')$ involves only right-shrinking links; If so, stop; otherwise, let $\langle H, H' \rangle$ be the *last* non-right-shrinking link in $\mathcal{P}_R(H'')$, $\mathcal{P}_1$ be the portion from H_0 to H' in $\mathcal{P}_R(H'')$, and $\mathcal{P}_2$ be the portion from H' to H'' in $\mathcal{P}_R(H'')$. Note that $\mathcal{P}_2$ involves right-shrinking links only.
- **Step 3 (Path updating).**
  - **Step 3.1.** Find the path $\mathcal{P}_L(H')$ from H_0 to H' via the four-step procedure in Section 3.3.
  - **Step 3.2.** Update $\mathcal{P}_R(H'')$ by replacing $\mathcal{P}_1$ with $\mathcal{P}_L(H')$. Note that both $\mathcal{P}_1$ and $\mathcal{P}_L(H')$ start with H_0 and end at H'.
- **Step 4 (Repetition).** Go to Step 2 for another round.

Lemma 3.5. *The procedure of finding the path $\mathcal{P}_R(H'')$ for a given solution H'' would always terminate and the found $\mathcal{P}_R(H'')$ involves right-shrinking links only.*

Proof. We first prove that the procedure would always terminate with two steps. First, we show that H' at one round does not appear in $\mathcal{P}_2$ at a previous round by contradiction. Let $\langle H_{cur}, H'_{cur} \rangle$ be the last non-right-shrinking link at the current round and $\langle H_{pre}, H'_{pre} \rangle$ be that at the previous round. Suppose H'_{cur} appears in the path $\mathcal{P}_2$ at the previous round. There exist a path from H'_{cur} to H'_{pre} (since H'_{pre} is in the current $\mathcal{P}_2$ from H'_{cur} to H'') and another one from H'_{pre} to H'_{cur} (since H'_{cur} is in the previous $\mathcal{P}_2$ from H'_{pre} to H''), both involving right-shrinking links only. We then deduce that $R'_{pre} \subseteq R'_{cur}$ and $R'_{cur} \subseteq R'_{pre}$, which imply that $R'_{cur} = R'_{pre}$. In addition, we have $R'_{pre} \subseteq R_{cur}$ based on the invariant in Section 3.3 over the left-anchored sub-path $\langle H_{cur}, ..., H'_{pre} \rangle$. We deduce that $R'_{cur} = R'_{pre} \subseteq R_{cur}$, which leads to a contradiction to the fact that $\langle H_{cur}, H'_{cur} \rangle$ is non-right-shrinking link. Second, we deduce that $\mathcal{P}_2$ would involve at least one more new solution after each round, which further implies that the above procedure would always terminate since the number of unique solutions is bounded.

We then prove that the path $\mathcal{P}_R(H'')$ involves right-shrinking links only. This can be clearly verified by the Step 2 (Termination checking), i.e., the procedure only stops when $\mathcal{P}_R(H'')$ involves right-shrinking links only. □

In conclusion, we succeed in finding for any solution H'' a path that traverses from H_0 to H'' and involves right-shrinking links only in $\mathcal{G}_L$. Therefore, we propose to *drop* all non-right-shrinking links from $\mathcal{G}_L$. We denote the resulting solution graph by $\mathcal{G}_R$. For example, $\mathcal{G}_R$ based on the input graph in Figure 1 is shown in Figure 3(c), which involves 21 links and all solutions are reachable from H_0. It is clear that a DFS procedure from H_0 on $\mathcal{G}_R$, which we call the *right-shrinking traversal*, would return all solutions. We present this result in the following lemma.

Lemma 3.6. *Given a bipartite graph $G(\mathcal{L} \cup \mathcal{R}, \mathcal{E})$ with an initial MBP $H_0 = (L_0, \mathcal{R})$, the right-shrinking traversal lists all MBPs.*

Remarks. We remark that the four-step procedure for finding the path $\mathcal{P}_R(H'')$ is a conceptual one for verifying the correctness of the right-shrinking traversal only. The implementation of right-shrinking traversal will be discussed in Section 3.5. In addition, the right-shrinking traversal is on top of the left-anchored traversal. Besides leading to a sparser solution graph, it re-organizes the search space. To be specific, any solution $H'' = (L'', R'')$ reachable from $H = (L, R)$ in $\mathcal{G}_R$ must satisfy $R'' \subseteq R$. One benefit is that it would be natural to impose some size constraints on the MBPs to be enumerated. The traversal from solution $H = (L, R)$ can be pruned if R shrinks below the size threshold (details will be presented in Section 5).

3.5 iTraversal: Summary and Analysis

We present the `iTraversal` algorithm, which employs left-anchored traversal and right-shrinking traversal in Algorithm 2. `iTraversal` differs from `bTraversal` in the following aspects. First, it takes $(L_0, \mathcal{R})$ but not an arbitrary MBP as the initial solution, where L_0 is a maximal subset of $\mathcal{L}$ such that $(L_0, \mathcal{R})$ is a MBP (Line 1). Second, in Step 1 of forming almost-satisfying graphs, it *prunes* those vertices in $\mathcal{R}$ from consideration so that it would traverse along left-anchored links only (Line 5). This implements the left-anchored traversal. Third, in Step 2 of enumerating local solutions, it *prunes* those local solutions H_{loc} for which there exists a vertex $u \in \mathcal{R}$ such that u is not in H_{loc} and $H_{loc} \cup \{u\}$ is a k-biplex (Line 7). These local solutions can be pruned since they can be extended to solutions with the right side containing a vertex that is not contained by the right side of the current solution, i.e., the links from the current solution to these solutions are non-right-shrinking links. This implements the right-shrinking traversal.

Algorithm 2: The algorithm: `iTraversal`.

Input: Bipartite graph $G = (\mathcal{L} \cup \mathcal{R}, \mathcal{E})$, integer $k \geq 1$;
Output: All maximal k-biplexes;
1 **Initialize** $H_0 = (L_0, \mathcal{R})$, B-tree $\mathcal{T} = \{H_0\}$;
2 `iThreeStep`$(G, H_0, \mathcal{T})$;
3 **Procedure** `iThreeStep`$(G, H, \mathcal{T})$
4 **(Step 1) foreach** v *in* $V(G) \backslash V(H)$ **do**
5 **if** v *in* $\mathcal{R}$ **then Continue;** //Left-anchored traversal ;
6 **(Step 2) foreach** H_{loc} *in* `EnumAlmostSat`$(G[H, v])$ **do**
7 **if** *there exists* u *in* $\mathcal{R} \backslash V(H_{loc})$ *s.t.* $G[H_{loc}, \{u\}]$ *is k-biplex* **then Continue;** //Right-shrinking traversal;
8 **(Step 3)** Extend H_{loc} to be a maximal k-biplex H' with vertices from $V(G) \backslash V(H_{loc}) \backslash \mathcal{R}$;
9 **if** $H' \notin \mathcal{T}$ **then**
10 Insert H' to $\mathcal{T}$;
11 `iThreeStep`$(G, H', \mathcal{T})$;

Remark. For `bTraversal`, it can be further enhanced with a so-called *exclusion* strategy [4]. The idea is to maintain for each solution an *exclusion* set once the solution is traversed and then prune the links towards those solutions which involve a vertex in the exclusion set. Details are referred to the technical report [1]. We verify that this strategy is applicable to `iTraversal`, for which the correctness proof is included in the technical report [1] for the sake of space. In conclusion, `iTraversal` implements left-anchored traversal, right-shrinking traversal and the exclusion strategy. The solution graph underlying `iTraversal` is denoted by $\mathcal{G}_E$, which is even sparser than $\mathcal{G}_R$. For example, $\mathcal{G}_E$ based on the input graph in Figure 1 is shown in Figure 3(d), which involves 13 links and all solutions are reachable from H_0.

Total running time. Let α be the number of solutions. The time cost of `iTraversal` is dominated by that of calling the `iThreeStep` procedure α times, each when a solution is found for the first time. Consider the time cost of the `iThreeStep` procedure. Let β be the time complexity of the `EnumAlmostSat` procedure and γ be the number of local solutions returned by the `EnumAlmostSat` procedure. The `EnumAlmostSat` procedure is called $O(|\mathcal{L}|)$ times, costing $O(|\mathcal{L}| \cdot \beta)$ time. There are $O(|\mathcal{L}| \cdot \gamma)$ local solutions and for each, the cost is the sum of $O(|\mathcal{R}| \cdot |H_{max}|)$ (for Line 7), $(|\mathcal{L}| \cdot |H_{max}|)$ (for Line 8), and $O(\log \alpha \cdot |H_{max}|)$ (for Line 9-11), where $H_{max} = (L_{max}, R_{max})$ is the solution with the maximum size.

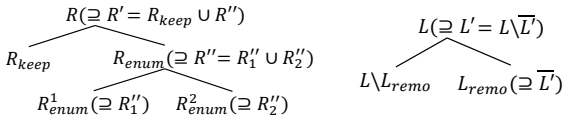
Therefore, the overall time complexity of `iTraversal` is $O(\alpha \cdot (|\mathcal{L}| \cdot \beta + |\mathcal{L}| \cdot \gamma \cdot (|V(G)| \cdot |H_{max}| + \log \alpha \cdot |H_{max}|)))$. Here, α is exponential w.r.t. the bipartite graph size. $\beta = O((|L_{max}| \cdot |R_{max}|)^{k+1})$ and $\gamma = O((|L_{max}| \cdot |R_{max}|)^{k})$ are polynomial with k as a constant and will be discussed in Section 4. In a simpler form, the time complexity is $O(\alpha \cdot |\mathcal{L}| \cdot (|L_{max}| \cdot |R_{max}|)^{k+1} \cdot |V_G| \cdot |H_{max}|)$.

Delay. The delay of an enumeration algorithm corresponds to the maximum of three parts, namely (1) the time spent after the algorithm starts and before the first solution is found, (2) the time spent between any two consecutive solutions are found, and (3) the time spent after the last solution is found and till the algorithm terminates. With a small trick [38], i.e., we print a solution before and after the recursive call (Line 11 of Algorithm 2) in an alternating manner during the sequence of recursive calls, the algorithm would output at least one solution every two successive recursive calls of the `iThreeStep` procedure. Therefore, the delay corresponds to the time complexity of the `iThreeStep` procedure, i.e., $O(|\mathcal{L}| \cdot \beta + |\mathcal{L}| \cdot \gamma \cdot (|V(G)| \cdot |H_{max}| + \log \alpha \cdot |H_{max}|)))$, which is polynomial with k as a constant (β and γ are polynomial, which will be discussed in Section 4). We remark that (1) `iTraversal` improves the delay of `bTraversal` based on the same implementation of `EnumAlmostSat`, which is $O(|V(G)| \cdot \beta + |V(G)| \cdot \gamma \cdot (|V(G)| \cdot |H_{max}| + \log \alpha \cdot |H_{max}|))$; and (2) `iMB` and the graph inflation based algorithm `FaPlexen` have their delay exponential w.r.t. the size of the bipartite graph [46, 49].

4 THE ENUMALMOSTSAT PROCEDURE

Let (L, R) be a solution and $(L \cup \{v\}, R)$ be an almost-satisfying graph, where $v \in \mathcal{L} \backslash L$. The `EnumAlmostSat` procedure is to enumerate all local solutions within $(L \cup \{v\}, R)$, which involve v. Essentially, the task is to explore a search space of $\{(L', R')\}$, where $L' \subseteq L$ and $R' \subseteq R$, and find those (L', R')'s such that $(L' \cup \{v\}, R')$ is a local solution. Note that this search space has a size of $O(2^{|L|+|R|})$, and hence simply enumerating each pair (L', R') would be costly.

In this paper, we develop a series of techniques for refining the enumerations on L and R so as to reduce and/or prune the search space (Section 4.1, 4.2, 4.4, and 4.3). We finally present an



(a) Partitions on L (b) Partitions on R

Figure 6: Illustration of partitions on L and R.

algorithm based on these refined enumerations and analyze its time complexity (Section 4.5).

4.1 Refined Enumeration on R: 1.0

We start with an observation presented in the following lemma.

Lemma 4.1. *Given an almost-satisfying graph $(L \cup \{v\}, R)$, each vertex $u \in R$ that connects v is involved in all local solutions within $(L \cup \{v\}, R)$.*

Proof. The complete proof is in the technical report [1]. □

Based on Lemma 4.1, we can partition R into two sets, namely one containing those vertices that connect v and the other containing the remaining vertices. We denote the former by R_{keep} and the latter by R_{enum}. An illustration is shown in Figure 6(a). Then, R_{keep} is involved in all local solutions within $(L \cup \{v\}, R)$ and hence the enumeration of the vertices in R_{keep} can be avoided when enumerating $R' \subseteq R$. Specifically, when enumerating the subsets of R, we enumerate $R'' \subseteq R_{enum}$ only and for each R'', we construct a R' as $R'' \cup R_{keep}$. In addition, we only need to enumerate those R'''s with $|R''| \leq k$ since otherwise v would disconnect more than k vertices in R' and $(L' \cup \{v\}, R')$ cannot be a k-biplex. With this, the number of subsets of R to enumerate is reduced from $O(2^{|R|})$ to $O(|R_{enum}|^k)$.

4.2 Refined Enumeration on R: 2.0

Based on Section 4.1, the search space is reduced from one containing all subsets of R to one containing all subsets R'' of R_{enum} with $|R''| \leq k$. In this section, we further prune some subsets R'' of R_{enum} by refining the enumeration on R_{enum}.

Specifically, we partition R_{enum} into two groups, namely $R^1_{enum} = \{u \in R_{enum} \mid \overline{\delta}(u, L) \leq k-1\}$ and $R^2_{enum} = R_{enum} \backslash R^1_{enum} = \{u \in R_{enum} \mid \overline{\delta}(u, L) = k\}$. Instead of enumerating $R'' \subseteq R_{enum}$ with $|R''| \leq k$ directly, we enumerate $R''_1 \subseteq R^1_{enum}$ and $R''_2 \subseteq R^2_{enum}$ with $|R''_1 \cup R''_2| \leq k$ and construct R'' as $R''_1 \cup R''_2$. An illustration is shown in Figure 6(a). We then have the following lemma for pruning some enumerations of R''_1 and R''_2.

Lemma 4.2. *Let $R' = R'' \cup R_{keep}$ and $R'' = R''_1 \cup R''_2$, where $R''_1 \subseteq R^1_{enum}$, $R''_2 \subseteq R^2_{enum}$, and $|R''_1 \cup R''_2| \leq k$. There does not exist a subset L' of L such that $(L' \cup \{v\}, R')$ is a local solution if (1) $|R''_1 \cup R''_2| < k$ and (2) $R^1_{enum} \backslash R''_1 \neq \emptyset$.*

Proof. The complete proof is in the technical report [1]. □

Based on Lemma 4.2, we can prune those enumerations of R''_1 and R''_2 with $|R''_1 \cup R''_2| < k$ and $R^1_{enum} \backslash R''_1 \neq \emptyset$, and thus it reduces the enumerations of subsets R'' of R_{enum}.

4.3 Refined Enumeration on L: 1.0

Given a subset R' of R, where $R' = R'' \cup R_{keep}$ and $R'' = R''_1 \cup R''_2$, we enumerate the subsets of L and for each subset L', construct $(L' \cup \{v\}, R')$ as a candidate of a local solution. A straightforward method is to enumerate all subsets of L. However, the search space would be $O(2^{|L|})$. In this section, we aim to refine the enumeration on L so that the search space is reduced.

We have an observation presented in the following lemma.

Lemma 4.3. *In the subgraph $(L' \cup \{v\}, R')$, where $L' = L$ and $R' = R_{keep} \cup R''_1 \cup R''_2$, all vertices except for those in R''_2 disconnect at most k vertices and vertices in R''_2 disconnect exactly $(k+1)$ vertices.*

Proof. This can be verified by the definitions of k-biplex, R_{keep}, R''_1 and R''_2. The detailed proof is in the technical report [1]. □

Based on Lemma 4.3, we can construct potential local solutions $(L' \cup \{v\}, R')$ by removing a *minimal* set $\overline{L'}$ of vertices from L so that all vertices in R''_2 would disconnect at most k remaining vertices, where $L' = L \backslash \overline{L'}$. Note that (1) after removing $\overline{L'}$ from L, those remaining vertices in $L \cup \{v\}$ (i.e., the vertices in $L' \cup \{v\}$) and vertices in $R_{keep} \cup R''_1$ would still disconnect at most k vertices due to the hereditary property and (2) $\overline{L'}$ needs to be *minimal* so that $(L' \cup \{v\}, R')$ would be a *maximal* k-biplex.

In addition, when a vertex v' that connects all vertices in R''_2 is removed, each vertex in R''_2 would still disconnect $(k+1)$ vertices. Therefore, we focus on the set L_{remo} of the vertices that disconnect at least one vertex in R''_2, i.e., $L_{remo} = \{v' \in L \mid \overline{\delta}(v', R''_2) > 0\}$, and enumerate subsets $\overline{L'}$ of L_{remo} to be removed. An illustration is shown in Figure 6(b). For each minimal set $\overline{L'}$ such that $(L' \cup \{v\}, R')$ is a local solution, where $L' = L \backslash \overline{L'}$, it involves no more than $|R''_2|$ vertices. This is because only vertices in R''_2 disconnect more than k vertices (exactly $(k+1)$ vertices) and to make one vertex in R''_2 disconnect at most k vertices, it is enough to remove one vertex from L_{remo}.

In conclusion, we enumerate $\overline{L'}$ from L_{remo} with $|\overline{L'}| \leq |R''_2| \leq k$ to be removed, and for each $\overline{L'}$, we construct $(L' \cup \{v\}, R')$ as a candidate of local solution, where $L' = L \backslash \overline{L'}$. In this way, the search space of enumerating L is reduced from $O(2^{|L|})$ to $O(|L_{remo}|^k)$.

4.4 Refined Enumeration on L: 2.0

When enumerating $\overline{L'}$ from L to be removed, we follow an increasing order of $|L|$ from 0 to $|R''_2|$. If for a subset $\overline{L'}$ and $L' = L \backslash \overline{L'}$, $(L' \cup \{v\}, R')$ is a local solution, we prune all supersets of $\overline{L'}$ from being enumerated to be removed since removing each of them would construct a k-biplex that is not maximal within $(L \cup \{v\}, R)$.

4.5 `EnumAlmostSat`: Summary and Analysis

We present the algorithm for the `EnumAlmostSat` procedure in Algorithm 3, which is based on the 2.0 versions for enumerating R and L. We use $Powerset(L_{remo})$ to denote the set of all subsets of L_{remo} (Line 4 and 7). Note that output solutions in line 6 can be non-globally optimal since an almost-satisfying graph rather than the whole graph is used as the context for checking the maximality. The correctness can be guaranteed by the lemmas and discussions in this section.

Time complexity. The time complexity of `EnumAlmostSat` is dominated by that of enumerating subsets of R and L (Line 2-8). There are $O(|R_{enum}|^k)$ enumerations of R'' (Line 2), and for each one, it incurs the following costs. First, it partitions L, which takes $O(|L| \cdot |R|)$

Algorithm 3: The algorithm: `EnumAlmostSat`.

Input: Almost-satisfying graph $(L \cup \{v\}, R)$;
Output: All local solutions including v within $(L \cup \{v\}, R)$;
1 Partition R into R_{keep}, R^1_{enum} and R^2_{enum} (Section 4.1 and 4.2);
2 **foreach** $R' = R_{keep} \cup R''_1 \cup R''_2$ *(as constructed in Section 4.2)* **do**
3 Construct L_{remo} as $\{v' \in L \mid \overline{\delta}(v', R''_2) > 0\}$;
4 **foreach** $\overline{L'} \in Powerset(L_{remo})$ *with* $|\overline{L'}| \leq |R''_2|$ *in an ascending order of* $|\overline{L'}|$ **do**
5 Construct L' as $L \backslash \overline{L'}$;
6 **if** $(L' \cup \{v\}, R')$ *is a local solution* **then**
7 Prune those subsets from $Powerset(L_{remo})$, which are supersets of $\overline{L'}$;
8 Print $(L' \cup \{v\}, R')$;

time (Line 3). Second, it enumerates subsets $\overline{L'}$ of L_{remo} with the size at most $|R''_2|$ on L_{remo} (Line 4). There are $O(|L_{remo}|^k)$ such $\overline{L'}$'s, and for each $\overline{L'}$, the cost is dominated by that of checking the maximality (Line 6), which is $O(|L| \cdot |R|)$. In summary, `EnumAlmostSat` would return at most $\gamma = O(|R_{enum}|^k \cdot |L_{remo}|^k)$ local solutions, and the time complexity of `EnumAlmostSat` is $\beta = O(|R_{enum}|^k \cdot |L_{remo}|^k \cdot |L| \cdot |R|)$, which is polynomial with k as a constant.

5 EXTENSIONS OF ITRAVERSAL

In some scenarios, one may want to impose some size constraints on one side or both sides of a MBP to be enumerated. For example, one is interested in only those MBPs with the size on either side to be at least a threshold θ. We show that such constraints can be conveniently incorporated to the `iTraversal` algorithm so that not all MBPs need to be enumerated, which achieves better efficiency. In contrast, for `bTraversal`, these constraints cannot be incorporated easily and all MBPs need to be enumerated and then a filtering step as post-processing is necessary, which is inefficient. For illustration, we consider the constraint that a MBP has the sizes on its both sides at least θ, and we call such a MBP a *large MBP*. However, the techniques can be easily customized for slightly different constraints such that a MBP has the size on its left or right side at least a threshold θ.

Right-side pruning. Recall that `iTraversal` adopts the right-shrinking traversal, i.e., it traverses along links from a solution $H = (L, R)$ to another $H' = (L', R')$ with $R' \subseteq R$. This would allow several opportunities for pruning. (1) Almost-satisfying graph pruning: When an almost-satisfying graph $G[H, v]$ is formed in Step 1 of `iThreeStep`, we prune it if $\delta(v, R) + k < \theta$. This is because any solution found based on this almost-satisfying graph would involve v at the left side and less than θ vertices at the right side (including at most $\delta(v, R)$ vertices that connect v and k vertices that disconnect v). (2) Local solution pruning: When enumerating the local solutions of an almost-satisfying graph $G[(L, R), v]$ in Step 2 of `iThreeStep`, the enumerations of the subsets R' of R with $|R'| < \theta$ can be pruned (correspondingly, the enumerations of the subsets L' of L can be saved) since all solutions formed from these local solutions would have a size on their right side being smaller than θ (due to the right-shrinking traversal). (3) Solution pruning: When a solution $H = (L, R)$ with $|R| < \theta$ is found, we skip the recursive call of `iThreeStep` from H since all solutions that will be found via this call would have the size of their right side at most $|R|$ and smaller than θ (due to the right-shrinking traversal).

Left-side pruning. Based on the exclusion strategy, we derive the following pruning technique. When a solution H is found, we check whether $|\mathcal{L}| - |\mathscr{E}(H)| < \theta$, where $\mathscr{E}(H)$ is the exclusion set maintained for H by the exclusion strategy. If so, we skip the recursive call `iThreeStep` from H so that all links from H are pruned. To verify, consider a solution $H' = (L', R')$ that can be reached from H via the call `iThreeStep` from H. There are two cases. Case 1: $L' \cap \mathscr{E}(H) \neq \emptyset$. The link from H to H' would be pruned by the exclusion strategy. Case 2: $L' \cap \mathscr{E}(H) = \emptyset$. We have $|L'| \leq |\mathcal{L}| - |\mathscr{E}(H)| < \theta$, and thus H' cannot be a large MBP.

Table 1: Real datasets

Name	Category	$\lvert\mathcal{L}\rvert$	$\lvert\mathcal{R}\rvert$	$\lvert\mathcal{E}\rvert$
Divorce	HumanSocial	9	50	225
Cfat	Miscellaneous	100	100	802
Crime	Social	551	829	1,476
Opsahl	Authorship	2,865	4,558	16,910
Marvel	Collaboration	19,428	6,486	96,662
Writer	Affiliation	89,356	46,213	144,340
Actors	Affiliation	392,400	127,823	1,470,404
IMDB	Communication	428,440	896,308	3,782,463
DBLP	Authorship	1,425,813	4,000,150	8,649,016
Google	Hyperlink	17,091,929	3,108,141	14,693,125

6 EXPERIMENTS

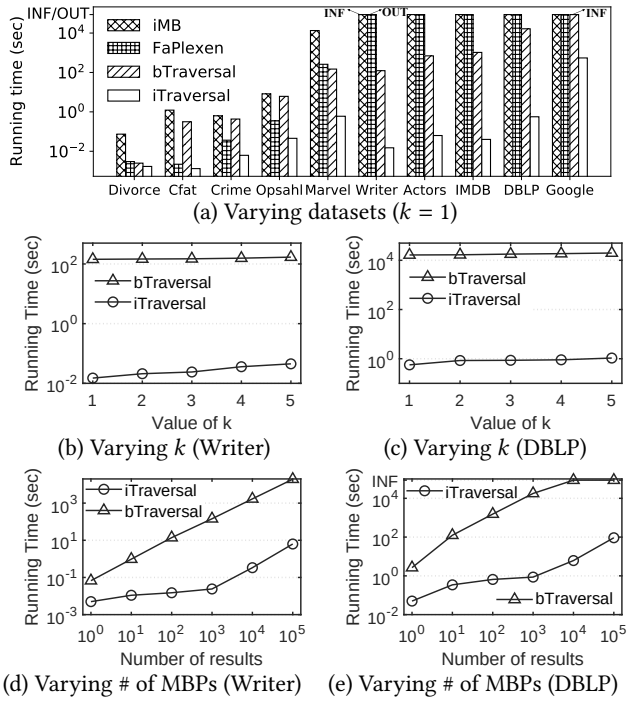
Datasets. We use both real and synthetic datasets. The real datasets are taken from various domains (http://konect.cc/) and summarized in Table 1. The synthetic datasets are Erdös-Réyni (ER) graphs, which are generated by first creating a certain number of vertices and then randomly creating a certain number of edges between pairs of vertices. We set the number of vertices and edge density as 100k and 10 for synthetic datasets, respectively, by default. Here, the edge density of a graph $G = (\mathcal{L} \cup \mathcal{R}, \mathcal{E})$ is defined as $|\mathcal{E}|/(|\mathcal{L}|+|\mathcal{R}|)$.

Algorithms. We compare our algorithm `iTraversal`, with three baselines, namely, `iMB` [46], `FaPlexen` [49] and `bTraversal`. `iMB` was proposed for enumerating maximal k-biplexes. `FaPlexen` is the state-of-the-art algorithm for enumerating large maximal k-plexes and we run it on the inflated graph of a bipartite graph for enumerating maximal k-biplexes. `bTraversal` follows the conventional framework of reverse search and implements the `EnumAlmostSat` procedure by first inflating the graph and then using an existing algorithm for enumerating local maximal k-plexes [4].

Settings. All algorithms were implemented in C++ and executed on a machine with a 2.66GHz processor and 32GB of memory, with CentOS installed. We set the running time limit (INF) as 24 hours and the memory budget (OUT) as 32GB.

6.1 Comparison among Algorithms

Results of running time (real datasets). The results of running time when returning the first 1,000 MBPs (by following the existing studies [4]) are shown in Figure 7(a) for varying datasets, (b) and (c) for varying k, and (d) and (e) for varying the number of returned MBPs (results on Writer and DBLP are shown only and those on other datasets show similar clues and are put in the technical report



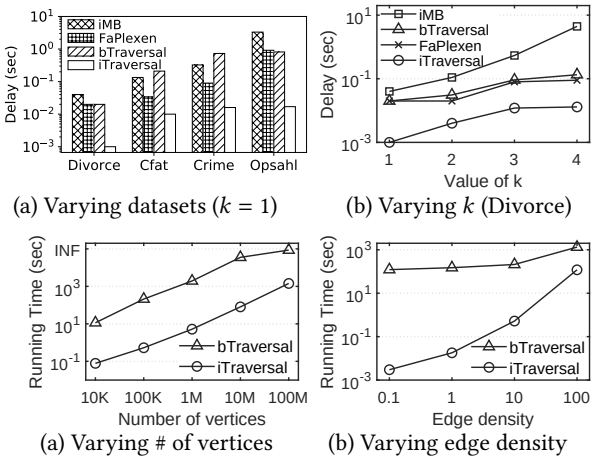
(a) Varying datasets ($k = 1$)
(b) Varying k (Writer)
(c) Varying k (DBLP)
(d) Varying # of MBPs (Writer)
(e) Varying # of MBPs (DBLP)

Figure 7: Results of running time (real datasets)

[1]). We have the following observations. First, `iTravesal` outperforms all other algorithms and can finish on all datasets. Second, `iMB` cannot finish within INF on large datasets due to its ineffective pruning techniques. Third, `FaPlexen` cannot finish within OUT on large datasets either, mainly due to its underlying graph inflation procedure. For example, for Marvel which has 96 thousand edges, it would generate more than 200 million edges after graph inflation. Therefore, we exclude `iMB` and `FaPlexen` for the remaining experiments of enumerating MBPs. Fourth, `bTraversal` is more scalable than `iMB` and `FaPlexen` but still cannot finish on the Google dataset within INF. Fifth, `iTraversal` and `bTraversal` have their running time increase as k and the number of MBPs grows, and `iTraversal` is up to four orders of magnitude faster than `bTraversal`. We note that `iTraversal` has its running time clearly rising as k increases when compared with `bTraversal`. The reason is that `iTraversal` adopts the proposed `L2.0+R2.0`, whose pruning power gets weaker as k increases, while `bTraversal` adopts a procedure based on graph inflation, which is not that sensitive to k.

Results of delay (real datasets). The results are shown in Figure 8. Note that we use small datasets for this experiment since some baseline algorithms `iMB` cannot find all solutions on larger datasets within INF and thus the delay cannot be measured. In general, the delay of all algorithms increases with k and `iTraversal` has the smallest delays. This is consistent with our theoretical analysis that the delay of `iTraversal` is polynomial while those of `iMB` and `FaPlexen` are exponential w.r.t. the size of the input bipartite graph.

Scalability test/Varying # of vertices (synthetic datasets). The results when returning the first 1,000 MBPs are shown in Figure 9(a). `iTraversal` achieves at least 100× speedup compared with `bTraversal` and can handle large datasets with 100 million vertices and 1 billion edges within INF while `bTraversal` cannot.

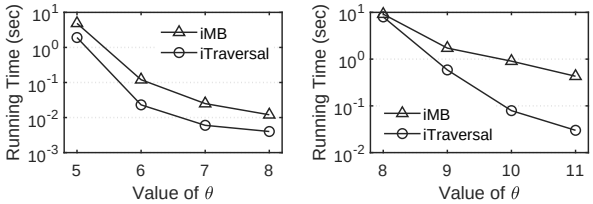


(a) Varying datasets ($k = 1$)
(b) Varying k (Divorce)
(a) Varying # of vertices
(b) Varying edge density

Figure 9: Results of running time (synthetic datasets)

Varying edge density (synthetic datasets). The results are shown in Figure 9(b). `iTraversal` outperforms `bTraversal` by around 1-5 orders of magnitude. The speedup decreases as the graph becomes denser. Possible reasons include: (1) the density gap between the original almost-satisfying graph and the inflated graph narrows (note that `bTraversal` involves a graph inflation step for implementing `EnumAlmostSat`) and (2) the number of solutions decreases in dense bipartite graphs.

Extension of `iTraversal` for enumerating large MBPs (real datasets). We compare the extension of `iTraversal` with `iMB`, for which some pruning techniques are developed for enumerating large MBPs [46]. We exclude `FaPlexen` and `bTraversal` for comparison because (1) `FaPlexen` cannot handle large datasets due to the graph inflation procedure and (2) `btraversal` cannot be extended with techniques for pruning small MBPs, but needs to enumerate all MBPs and then filter out those small ones. In addition, for both `iTraversal` and `iMB`, we perform a $(\theta - k)$-core decomposition [28] on the input bipartite graph first before enumerating large MBPs on them. This is due to the fact that each large MBP, i.e., a MBP with the size of both sides at least θ, corresponds to a $(\theta - k)$-core, which is clear. The results of running time on Writer and DBLP are shown in Figure 10 for varying θ. We observe that the running time decreases as θ grows. This is because both the size of $(\theta - k)$-core and the number of large MBPs decrease as θ grows. `iTraversal` is faster than `iMB` by up to four orders of magnitude.

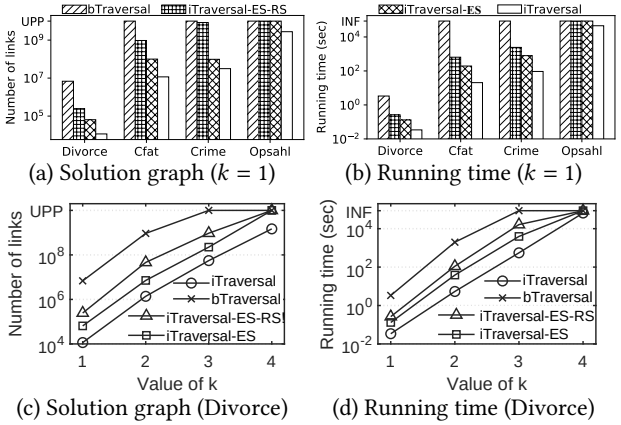


(a) Varying θ (k=1, Writer)
(b) Varying θ (k=1, DBLP)

Figure 10: Results of enumerating large MBPs

6.2 Performance Study of `iTraversal`

`bTraversal` vs `iTraversal` We compare three different versions of `iTraversal`, namely (1) `iTraversal`: the full version involving left-anchored traversal, right-shrinking traversal and exclusion strategy, (2) `iTraversal-ES`: the full version without the exclusion strategy, and (3) `iTraversal-ES-RS`: `iTraversal-ES` without the



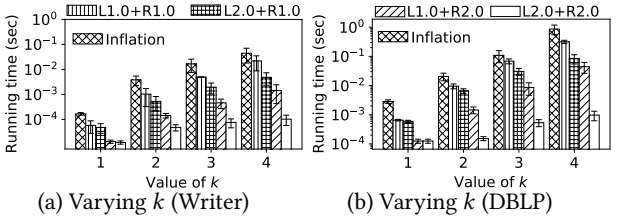
(a) Solution graph ($k = 1$) (b) Running time ($k = 1$)

(c) Solution graph (Divorce) (d) Running time (Divorce)

Figure 11: `bTraversal` vs `iTraversal`

right-shrinking traversal. We also consider the `bTraversal` for comparison. All these algorithms adopt the `L2.0+R2.0` algorithm for the `EnumAlmostSat` procedure for fair comparison. We measure the number of links of the solution graph and the running time. The results are shown in Figure 11. Note that we use those small real datasets for this experiment only since `bTraversal` cannot find all solutions within INF. We set the maximum allowed number of links (UPP) as 10^{10}. Figure 11(a) and (b) show that `iTraversal` would generate the sparsest solution graph and achieves up to 1000× speedup compared with `bTraversal`. These results also verify the effectiveness of left-anchored traversal, right-shrinking traversal, and exclusion strategy. Figure 11(c) and (d) show the results of varying k. The scale of solution graph and the running time of all algorithms grow exponentially w.r.t. k.

Performance study of `EnumAlmostSat`. We consider different combinations of enumeration methods on R and L as described in Section 4. The algorithm `L1.0+R1.0` corresponds to the combination of "refined enumeration on L: 1.0" and "refined enumeration on R: 1.0". Similarly, we have `L1.0+R2.0`, `L2.0+R1.0`, and `L2.0+R2.0`. In addition, we consider a baseline method `Inflation` which conducts a graph inflation procedure and then uses an existing procedure for enumerating local maximal k-plexes [4]. We execute these algorithms 1,000 times with random almost-satisfying graphs that are constructed based on Writer and DBLP. Specifically, we run `iTraversal` on a dataset (Writer or DBLP), collect the first 1,000 MBPs and then for each MBP $H = (L, R)$, we add to H a random vertex $v \in \mathcal{L} \backslash L$. We present the average running time in Figure 12. According to the results, the running time of all algorithms grows as k increases and `L2.0+R2.0` is the fastest, achieving up to 1000× speedup compared with `Inflation`.



(a) Varying k (Writer) (b) Varying k (DBLP)

Figure 12: Comparison among algorithms of `EnumAlmostSat`

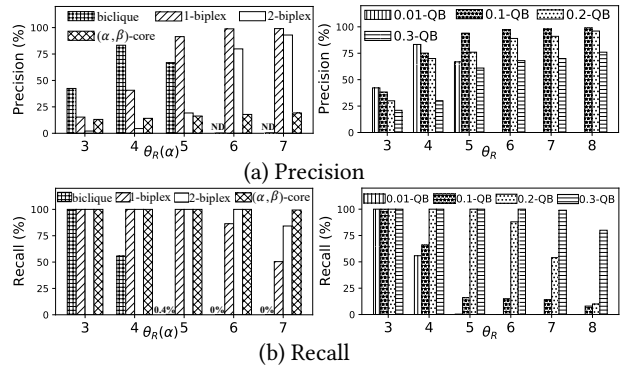
Left-anchored traversal vs Right-anchored traversal. Recall that the left-anchored traversal is based on an initial solution $H_0 = (L_0, \mathcal{R})$. A symmetric option is to use an intial solution $H_0' = (\mathcal{L}, R_0)$ as described in Section 3.2. We compare these two options by varying k and measuring the running of time of returning the first 1,000 MBPs. The results are put in the technical report [1] for sake of the space. We observe from the results that these two options have similar results. For left-anchored traversal, the scale of its solution graph depends on both sides, e.g., the number of almost-satisfying graphs is affected by $\mathcal{L}$ and the number of local MBPs within an almost-satisfying graphs is affected by both $\mathcal{L}$ and $\mathcal{R}$. There are no clear metrics to determine a dominating side.

6.3 Case Study: Fraud Detection

In this case study, we investigate four types of cohesive subgraphs, namely biclique, k-biplex, (α, β)-core, and δ-quasi-biclique (abbreviated as δ-QB), for a fraud detection task in face of camouflage attacks [21]. Specifically, we use the "Amazon Review Data" (the software category) [35], which contains 459,436 reviews on 21,663 softwares by 375,147 users. We then consider a scenario with the random camouflage attack [21], which injects a fraud block with 2K fake users, 2K fake products (i.e., softwares), 200K fake comments, and 200k camouflage comments to the data. The fake comments (resp. camouflage comments) are randomly generated between pairs of fake users and fake products (resp. real products) such that each fake user has an equal number of fake comments and camouflage ones. The rationale behind the random camouflage attack is that in reality, fake users could be coordinated to comment on a set of products and/or deliberately post comments on some real products so as not to be spotted out [21]. We then find four types of cohesive subgraphs, namely biclique, k-biplex (with $k = 1$ and $k = 2$), (α, β)-core, and δ-QB (with $\delta = 0.01, 0.1, 0.2$ and 0.3). For biclique, k-biplex and δ-QB, we explore different size constraints on the number of users (denoted by θ_L) and the number of products (denoted by θ_R). For (α, β)-core, we explore different settings of α and β with α=θ_R and β=θ_L. We classify all users and products that are involved in the found subgraphs as fake items and others as real ones. We then measure the precision, recall and F1 score.

The results are shown in Figure 13, where $\theta_L(\beta)$ is fixed at 4 and $\theta_R(\alpha)$ varies from 3 to 7. We have the following observations.

- **k-biplex.** It has both high precision and high recall with appropriate settings of θ_R (e.g., for 1-biplex, it has precision around 90% and recall around 100% when $\theta_R = 5$). This shows that in a scenario with random camouflage attacks, fake users and products can be effectively spotted out by finding k-biplexes. In this scenario, a group of fake users (e.g., those who are coordinated) would connect most of products in a pool (e.g., those that are to be promoted by paying the fake users) and deliberately disconnect few of the products, and this phenomenon is effectively captured by k-biplexes. In addition, we find that k-biplexes with a large k would involve many disconnections and become less cohesive. Hence, small k's are usually used for k-biplex in real applications, e.g., the setting $k = 1$ achieves the best result for this fraud detection application.
- **biclique.** It usually has very low recall as shown in Figure 13(b) (e.g., it has recall around 55% when $\theta_R = 4$ and close to 0 when $\theta_R \geq 5$), since biclique requires complete connections which is too strict. We remark that if no users are found, the precision and F1 score are undefined, marked as "ND" in Figure 13(a) and (c).



(a) Precision

(b) Recall

$\theta_R(\alpha)$	3	4	5	6	7
biclique	0.59	**0.67**	0.01	ND	ND
1-biplex	0.27	0.58	**0.96**	0.92	0.67

$\theta_R(\alpha)$	3	4	5	6	7
2-biplex	0.04	0.09	0.32	**0.89**	0.88
(α,β)-core	0.23	0.25	0.28	0.30	**0.32**

θ_R	3	4	5	6	7	8
0.01-QB	0.59	**0.67**	0.01	ND	ND	ND
0.2-QB	0.46	0.82	0.86	**0.88**	0.67	0.18

θ_R	3	4	5	6	7	8
0.1-QB	0.55	**0.70**	0.27	0.25	0.24	0.14
0.3-QB	0.34	0.46	0.75	0.80	**0.82**	0.79

(c) F1 score

Figure 13: Case Study 1: Fraud Detection

- (α, β)**-core.** It has high recall but very low precision, since it would usually find large but sparse subgraphs (e.g., it has precision below 25% in all settings shown in the figure and not beyond 30% even with larger α's). We note that all (α, β)-cores found are connected. In addition, we observe that its precision grows as both α and β become larger, but then the recall would drop. By exploring various settings of α and β, we find that when $\alpha = 30$ and $\beta = 12$, it has the best F1 score of 0.64, which is still worse than that of 1-biplex (i.e., 0.96).
- δ**-QB.** With larger δ's, it would allow more disconnections in a δ-QB, and thus the precision decreases and the recall increases. In addition, with larger θ_R's, it would find fewer δ-QBs that are large, and thus the precision increases and the recall decreases. By trying various settings, we found that when $\delta = 0.2$ and $\theta_R = 6$, it has the best F1 score of 0.88, which is worse than that of 1-biplex. We also note that a k-biplex with the sizes of both sides at least θ is a k/θ-QB, and a δ-QB with the sizes of both sides at most θ is a $\lceil\theta\delta\rceil$-biplex. With typical settings of k (e.g., below 5) and δ (e.g., a small real number below 1), k-biplexes are usually denser than δ-QBs since for the latter, there could be possibly be many disconnections when the sizes are large.

In summary, k-biplexes are superior over other structures for fraud detection, as confirmed by the F1 scores shown in Figure 13(c) (with the best of each method in bold).

7 RELATED WORK

Dense subgraphs of bipartite graphs. Extensive studies have been conducted on finding and/or enumerating dense subgraphs of bipartite graphs, including bicliques [2, 8, 25, 31, 34, 48], (α, β)-cores [28], k-biplexes [37, 46], k-bitrusses [43, 50] and quasi-bicliques [23, 24, 30, 45]. Existing studies of bicliques focus on maximal biclique enumeration [2, 25, 34, 48] and maximum biclique discovery [8, 31]. (α, β)-cores have been used in applications such as recommendation systems [15] and community search [19, 44]. Existing studies of k-biplexes focus on large maximal k-biplex enumeration [37, 46]. A k-bitruss [43, 50] is a bipartite graph where each edge is contained in at least k butterflies, where a butterfly corresponds to a complete 2×2 biclique [42]. There are two types of definition for a quasi-biclique $H = (L, R)$, namely (1) δ-quasi-biclique [30], where each vertex in L (resp. R) misses at most $\delta \cdot |R|$ (resp. $\delta \cdot |L|$) edges with $\delta \in [0, 1)$) and (2) γ-quasi-biclique [24], where at most $\gamma \cdot |L| \cdot |R|$ edges can be missed with $\gamma \in [0, 1)$. Existing studies of quasi-bicliques focus on finding the maximum quasi-biclique [23, 24, 30, 45]. There are some other studies which find subgraphs with a certain density and degree [29, 33]. In this paper, we focus on k-biplex for reasons as discussed in Section 1 (i.e., k-biplex imposes strict enough requirements on connections within a subgraph, tolerates some disconnections, and satisfies the hereditary property).

Maximal subgraphs enumeration. In general, there are two types of methodology, namely Bron-Kerbosch scheme [5] and reverse search [3, 10], for enumerating maximal subgraphs of desired properties. The original Bron-Kerbosch (BK) algorithm is proposed in [5] for enumerating maximal cliques. BK uses branch-and-bound and backtracking to filter out the branches that cannot yield desired subgraphs. Quite a few variants of BK have been proposed for solving various enumeration problems, including (bi)cliques [2, 16, 39, 48], k-plexes [13, 49], signed cliques [9, 27], motif cliques [22], temporal cliques [20], quasi-bicliques [37] and etc. However, the time complexity of BK and its variants is polynomial w.r.t. the worst-case number of desired subgraphs. In addition, these algorithms output desired subgraphs with exponential delay. Reverse search was first proposed as a general framework for enumerating subgraphs [3]. It defines a "successor" function to traverse from one solution to the others, which corresponds to a DFS on an implicit solution graph where the nodes represent desired subgraphs and the links capture the traversals from desired subgraphs to other desired subgraphs. Recently, this framework has been adopted to solve various enumeration problems, e.g., independent sets [11, 40], cliques [7, 12, 32], k-plexes [4], and general structures that satisfy the hereditary property [6, 10, 14]. The algorithms following this framework achieve an output-sensitive time complexity that is proportional to the number of desired subgraphs (within the input graph). Moreover, there is a polynomial time guarantee on the time cost per solution. In this paper, we adopt the reverse search framework and develop various techniques for improving and instantiating the framework for enumerating maximal k-biplexes. We note that our techniques of improving the framework for k-biplexes are new and different from those that were developed for other specific structures such as independent sets and cliques and the latter cannot be applied to k-biplexes.

8 CONCLUSION

In this paper, we study the *maximal k-biplex enumeration* problem. We develop an efficient reverse search-based algorithm with the polynomial delay guarantee. Extensive experiments on real and synthetic datasets demonstrate that our algorithm outperforms the existing methods in terms of total running time and delay significantly. In the future, we will investigate efficient parallel and distributed implementations. Another interesting research direction is to adapt the proposed reverse search-based algorithm to enumerate some other cohesive subgraphs over bipartite graphs.

REFERENCES

[1] 2021. Efficient Algorithms for Maximal k-biplex Enumeration (Technical report). https://anonymous.4open.science/r/SGMOD22-MBPE-F551.

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

[3] David Avis and Komei Fukuda. 1996. Reverse Search for Enumeration. *Discret. Appl. Math.* 65, 1-3 (1996), 21–46.

[4] Devora Berlowitz, Sara Cohen, and Benny Kimelfeld. 2015. Efficient Enumeration of Maximal k-Plexes. In *Proc. ACM SIGMOD Int. Conf. Manage. Data.* 431–444.

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

[6] Yixin Cao. 2020. Enumerating maximal induced subgraphs. *arXiv preprint arXiv:2004.09885* (2020).

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

[8] Lu Chen, Chengfei Liu, Rui Zhou, Jiajie Xu, and Jianxin Li. 2021. Efficient Exact Algorithms for Maximum Balanced Biclique Search in Bipartite Graphs. In *Proc. ACM SIGMOD Int. Conf. Manage. Data.* 248–260.

[9] Zi Chen, Long Yuan, Xuemin Lin, Lu Qin, and Jianye Yang. 2020. Efficient Maximal Balanced Clique Enumeration in Signed Networks. In *Proc. World Wide Web Conf., WWW*. 339–349.

[10] Sara Cohen, Benny Kimelfeld, and Yehoshua Sagiv. 2008. Generating all maximal induced subgraphs for hereditary and connected-hereditary graph properties. *J. Comput. Syst. Sci.* 74, 7 (2008), 1147–1159.

[11] Alessio Conte, Roberto Grossi, Andrea Marino, Takeaki Uno, and Luca Versari. 2017. Listing Maximal Independent Sets with Minimal Space and Bounded Delay. In *String Processing and Information Retrieval - 24th International Symposium, SPIRE 2017 (Lecture Notes in Computer Science, Vol. 10508)*. Springer, 144–160.

[12] Alessio Conte, Roberto Grossi, Andrea Marino, and Luca Versari. 2020. Sublinear-Space and Bounded-Delay Algorithms for Maximal Clique Enumeration in Graphs. *Algorithmica* 82, 6 (2020), 1547–1573.

[13] Alessio Conte, Tiziano De Matteis, Daniele De Sensi, Roberto Grossi, Andrea Marino, and Luca Versari. 2018. D2K: Scalable Community Detection in Massive Networks via Small-Diameter k-Plexes. In *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*. 1272–1281.

[14] Alessio Conte and Takeaki Uno. 2019. New polynomial delay bounds for maximal subgraph enumeration by proximity search. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*. 1179–1190.

[15] 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.

[16] David Eppstein, Maarten Löffler, and Darren Strash. 2013. Listing All Maximal Cliques in Large Sparse Real-World Graphs. *ACM J. Exp. Algorithmics* 18 (2013).

[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] Stephan Günnemann, Emmanuel Müller, Sebastian Raubach, and Thomas Seidl. 2011. Flexible Fault Tolerant Subspace Clustering for Data with Missing Values. In *11th IEEE International Conference on Data Mining, ICDM 2011*. 231–240.

[19] Yang Hao, Mengqi Zhang, Xiaoyang Wang, and Chen Chen. 2020. Cohesive Subgraph Detection in Large Bipartite Networks. In *SSDBM 2020: 32nd International Conference on Scientific and Statistical Database Management*. ACM, 22:1–22:4.

[20] Anne-Sophie Himmel, Hendrik Molter, Rolf Niedermeier, and Manuel Sorge. 2017. Adapting the Bron-Kerbosch algorithm for enumerating maximal cliques in temporal graphs. *Soc. Netw. Anal. Min.* 7, 1 (2017), 35:1–35:16.

[21] 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. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*. ACM, 895–904.

[22] Jiafeng Hu, Reynold Cheng, Kevin Chen-Chuan Chang, Aravind Sankar, Yixiang Fang, and Brian Y. H. Lam. 2019. Discovering Maximal Motif Cliques in Large Heterogeneous Information Networks. In *Proc. 36th IEEE Int. Conf. Data Eng., ICDE 2019*. IEEE, 746–757.

[23] Dmitry I. Ignatov. 2019. Preliminary Results on Mixed Integer Programming for Searching Maximum Quasi-Bicliques and Large Dense Biclusters. In *Supplementary Proceedings of ICFCA 2019 Conference and Workshops (CEUR Workshop Proceedings, Vol. 2378)*. CEUR-WS.org, 28–32.

[24] Dmitry I Ignatov, Polina Ivanova, and Albina Zamaletdinova. 2018. Mixed Integer Programming for Searching Maximum Quasi-Bicliques. In *International Conference on Network Analysis*. Springer, 19–35.

[25] Kyle Kloster, Blair D Sullivan, and Andrew van der Poel. 2019. Mining maximal induced bicliques using odd cycle transversals. In *Proceedings of the 2019 SIAM International Conference on Data Mining*. SIAM, 324–332.

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

[27] Rong-Hua Li, Qiangqiang Dai, Lu Qin, Guoren Wang, Xiaokui Xiao, Jeffrey Xu Yu, and Shaojie Qiao. 2021. Signed Clique Search in Signed Networks: Concepts and Algorithms. *IEEE Trans. Knowl. Data Eng.* 33, 2 (2021), 710–727.

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

[29] Hsiao-Fei Liu, Chung-Tsai Su, and An-Chiang Chu. 2013. Fast Quasi-biclique Mining with Giraph. In *IEEE International Congress on Big Data, BigData Congress 2013*. IEEE Computer Society, 347–354.

[30] Xiaowen Liu, Jinyan Li, and Lusheng Wang. 2008. Quasi-bicliques: Complexity and Binding Pairs. In *Computing and Combinatorics, 14th Annual International Conference, COCOON 2008 (Lecture Notes in Computer Science, Vol. 5092)*. Springer, 255–264.

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

[32] Kazuhisa Makino and Takeaki Uno. 2004. New Algorithms for Enumerating All Maximal Cliques. In *Algorithm Theory - SWAT 2004, 9th Scandinavian Workshop on Algorithm Theory (Lecture Notes in Computer Science, Vol. 3111)*. Springer, 260–272.

[33] Michael Mitzenmacher, Jakub Pachocki, Richard Peng, Charalampos E. Tsourakakis, and Shen Chen Xu. 2015. Scalable Large Near-Clique Detection in Large-Scale Networks via Sampling. In *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*. ACM, 815–824.

[34] Arko Provo Mukherjee and Srikanta Tirthapura. 2016. Enumerating maximal bicliques from a large graph using mapreduce. *IEEE Transactions on Services Computing* 10, 5 (2016), 771–784.

[35] Jianmo Ni. 2018. *Amazon Review Data.* https://nijianmo.github.io/amazon/index.html

[36] Ardian Kristanto Poernomo and Vivekanand Gopalkrishnan. 2009. Towards efficient mining of proportional fault-tolerant frequent itemsets. In *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*. 697–706.

[37] Kelvin Sim, Jinyan Li, Vivekanand Gopalkrishnan, and Guimei Liu. 2009. Mining maximal quasi-bicliques: Novel algorithm and applications in the stock market and protein networks. *Statistical Analysis and Data Mining* 2, 4 (2009), 255–273.

[38] U Takeaki. 2003. *Two general methods to reduce delay and change of enumeration algorithms.* Technical Report.

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

[40] Shuji Tsukiyama, Mikio Ide, Hiromu Ariyoshi, and Isao Shirakawa. 1977. A New Algorithm for Generating All the Maximal Independent Sets. *SIAM J. Comput.* 6, 3 (1977), 505–517.

[41] 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.

[42] Kai Wang, Xuemin Lin, Lu Qin, Wenjie Zhang, and Ying Zhang. 2019. Vertex Priority Based Butterfly Counting for Large-scale Bipartite Networks. *Proc. VLDB Endow.* 12, 10 (2019), 1139–1152.

[43] Kai Wang, Xuemin Lin, Lu Qin, Wenjie Zhang, and Ying Zhang. 2020. Efficient Bitruss Decomposition for Large-scale Bipartite Graphs. In *Proc. 36th IEEE Int. Conf. Data Eng., ICDE 2020*. IEEE, 661–672.

[44] Kai Wang, Wenjie Zhang, Xuemin Lin, Ying Zhang, Lu Qin, and Yuting Zhang. 2021. Efficient and Effective Community Search on Large-scale Bipartite Graphs. In *Proc. 36th IEEE Int. Conf. Data Eng., ICDE 2021*. IEEE, 85–96.

[45] Lusheng Wang. 2013. Near optimal solutions for maximum quasi-bicliques. *Journal of Combinatorial Optimization* 25, 3 (2013), 481–497.

[46] Kaiqiang Yu, Cheng Long, P Deepak, and Tanmoy Chakraborty. 2021. On Efficient Large Maximal Biplex Discovery. *IEEE Trans. Knowl. Data Eng.* (2021).

[47] Xichen Zhang and Ali A Ghorbani. 2020. An overview of online fake news: Characterization, detection, and discussion. *Information Processing & Management* 57, 2 (2020), 102025.

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

[49] Yi Zhou, Jingwei Xu, Zhenyu Guo, Mingyu Xiao, and Yan Jin. 2020. Enumerating Maximal k-Plexes with Worst-Case Time Guarantee. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020*. 2442–2449.

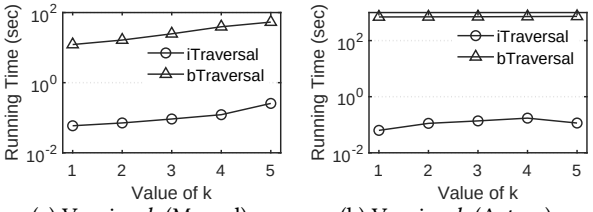
[50] Zhaonian Zou. 2016. Bitruss Decomposition of Bipartite Graphs. In *Database Systems for Advanced Applications - 21st International Conference, DASFAA 2016 (Lecture Notes in Computer Science, Vol. 9643)*. Springer, 218–233.

A ADDITIONAL EXPERIMENTAL RESULTS

We present the additional experimental results and performance study in this section.

A.1 Comparison among Algorithms

Additional results of running time (real datasets). The additional results of running time when returning the first 1,000 MBPs are shown in Figure 14. In general, they show the similar clues with those on Writer and DBLP, and `iTraversal` is up to four orders of mannitude faster than `bTraversal` on additional datasets.



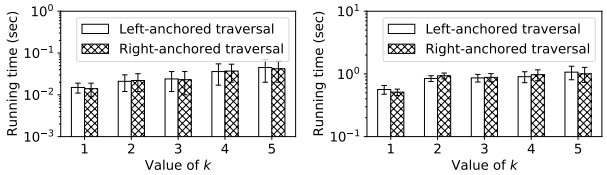
(a) Varying k (Marvel) (b) Varying k (Actors)

(c) Varying k (IMDB) (d) Varying k (Google)

Figure 14: Additional results of running time (real datasets)

A.2 Performance Study of `iTraversal`

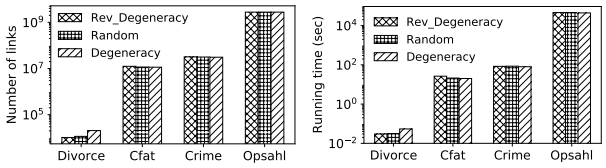
Left-anchored traversal vs Right-anchored traversal (Additional results). The results are shown in Figure 15. We compare these two options by varying k and measuring the running of time of returning the first 1,000 MBPs. We observe from the results that these two options have similar results.



(a) Varying k (Writer) (b) Varying k (DBLP)

Figure 15: Left-anchored vs right-anchored traversals

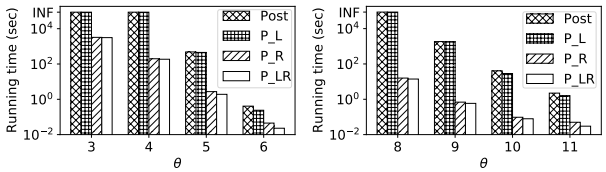
Performance study of orderings of vertices (Additional study). We note that `iTraversal` runs based on a pre-set ordering of vertices. For example, when forming almost-satisfying graphs or extending a local solution to be a maximal k-biplex, vertices are iteratively included based on an underlying ordering. Therefore, we study the effects of different orderings of vertices, namely, (1) `Random`: a random ordering of vertices, (2) `Degeneracy`: the degeneracy ordering that can be obtained by executing the existing algorithm for topological ordering, and (3) `rev-Degeneracy`: the reverse degeneracy ordering. We measure the number of links of the solution graph and the running time. The results are shown in Figure 16. We observe that these orderings have similar performances. Moreover, no heuristic metric motivates us to select a specific ordering of vertices.



(a) Solution graphs ($k = 1$). (b) Running time ($k = 1$).

Figure 16: Comparisons among different orderings.

Performance study of `iTraversal` for enumerating large MBPs (Additional study). We consider two types of pruning techniques proposed in Section 5, and compare four different versions of `iTraversal` for enumerating large MBPs, namely (1) `Post`: `iTraversal` without any pruning technique, (2) `P_L`: `iTraversal` with the left-side pruning, (3) `P_R`: `iTraversal` with the right-side pruning, and (4) `P_LR`: `iTraversal` with the left-side and right-side pruning. The results of running time are shown in Figure 17. We see that `P_L` and `P_R` are both faster than the basic version `Post`. `P_L` achieves small improvements compared with `P_R`. This is because the left-side pruning, derived based on the exclusion strategy, only works when the size of the set $\mathcal{L}\backslash\mathcal{E}(H)$ is less than θ. Moreover, `P_LR` combining all pruning techniques performs the best.



(a) Varying θ (k=1, Writer) (b) Varying θ (k=1, DBLP)

Figure 17: Comparison among pruning for listing large MBs.

A.3 Dataset Statistics

We analyze the properties of datasets used, according to the number of MBPs w.r.t. different values of k and θ. Since enumerating all MBPs within large datasets is time-consuming, we report the total number of MBPs (w.r.t. different k) only for four small datasets (i.e., Divorce, Cfat, Crime and Opsahl), and report the total number of large MBPs (w.r.t. different θ) for the remaining six large datasets. We show the results of the effect of varying k and θ on the number of large MBs in Figure 18 and Figure 19, respectively. In general, we have the number of (large) MBPs exponentially increase (resp. decrease) as k (resp. θ) grows.

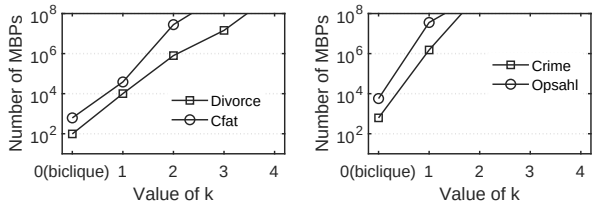


Figure 18: Results of # of MBPs (varying k)

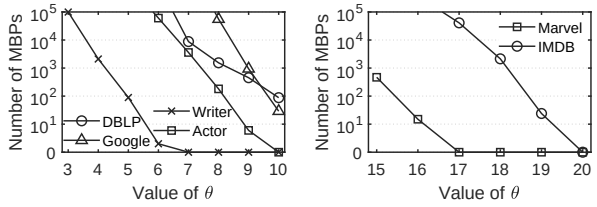


Figure 19: Results of # of large MBPs (varying θ)

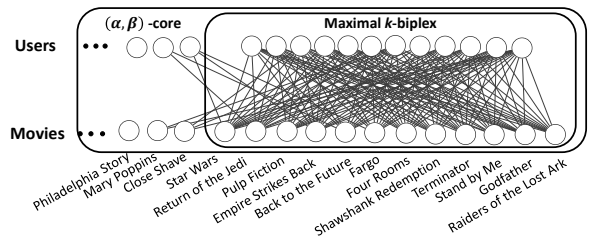


Figure 20: Case Study 2: Cohesiveness study

A.4 Additional Case Study: Cohesiveness Study

In this case study, we investigate the cohesiveness of subgraphs found by finding three types of subgraph, namely maximal biclique, maximal k-biplex (MBP), and (α, β)-core, on the MoiveLense dataset (http://konect.unikoblenz.de), which contains 20K ratings from 1.6K users on about 1K movies. For a subgraph $H = (L \cup R, E)$, we define its density as $|E|/(|L| \times |R|)$, which is a real number in $[0, 1]$.

- **k-biplex.** We first show a MBP with $k = 2$, which is returned by our algorithm and has 12 users, 12 movies, and 127 edges, enclosed by the inner rectangle in Figure 20. The cohesiveness of this MBP is reflected by (1) it has a density of 0.88 and (2) the 12 users have similar interests in movies and the 12 movies have similar topics, e.g., these users like action, adventure and science fiction movies such as "Star Wars" and "Terminator".
- **biclique.** The presented cohesive structure (found as an MBP) is not a biclique as $144 - 127 = 17$ edges are missed between the users and the movies. Hence, this cohesive structure would be missed when finding bicliques. In fact, the maximum biclique that can be found within the MBP includes only 18 vertices and 81 edges, which are 75% and 63.8% of those in the MBP, respectively.
- **(α, β)-core.** We show a (23,23)-core, which involves the presented cohesive structure and has 197 users, 167 movies and 7303 edges, enclosed by the outer rectangle in Figure 20. We remark that there does not exist a (24,24)-core as we have checked empirically. The cohesiveness of this structure is not as high as that of the MBP, which is reflected by (1) the density of this structure is 0.22 (while that of the MBP is 0.88) and (2) some users do not like quite a few movies in the subgraph (e.g., a user in this structure may dislike up to 198-23=175 movies in this structure).

In summary, we show a structure with high cohesiveness, which can be found by finding maximal k-biplexes, but not by maximal bicliques or (α, β)-cores.

B ADDITIONAL PROOFS

Lemma 4.1. Given an almost-satisfying graph $(L \cup \{v\}, R)$, each vertex $u \in R$ that connects v is involved in all local solutions within $(L \cup \{v\}, R)$.

Proof. Suppose $(L' \cup \{v\}, R')$ is a local solution and does not involve u, i.e., $u \notin R'$. Then, we consider another subgraph $(L' \cup \{v\}, R' \cup \{u\})$, which is larger than $(L' \cup \{v\}, R')$. We derive a contradiction by showing that $(L' \cup \{v\}, R' \cup \{u\})$ is a k-biplex since (1) each vertex $v' \in L'$ disconnects at most k vertices from $R' \cup \{u\}$ (note that $R' \cup \{u\}$ is subset of R and v' disconnects at most k vertices from R), (2) v disconnects at most k vertices from $R' \cup \{u\}$ (note that v disconnects at most k vertices from R' and v connects u), (3) each vertex $u' \in R'$ disconnects at most k vertices from $L' \cup \{v\}$ (note that $(L' \cup \{v\}, R')$ is a k-biplex) and (4) vertex u disconnects at most k vertices from $L' \cup \{v\}$ (note that u disconnects at most k vertices from L' and u connects v). □

Lemma 4.2. Let $R' = R'' \cup R_{keep}$ and $R'' = R''_1 \cup R''_2$, where $R''_1 \subseteq R^1_{enum}$, $R''_2 \subseteq R^2_{enum}$, and $|R''_1 \cup R''_2| \leq k$. There does not exist a subset L' of L such that $(L' \cup \{v\}, R')$ is a local solution if (1) $|R''_1 \cup R''_2| < k$ and (2) $R^1_{enum} \backslash R''_1 \neq \emptyset$.

Proof. This can be verified by contradiction. Suppose there exists a subset $L' \subseteq L$ such that $(L' \cup \{v\}, R')$ is a local solution. Then, another subgraph $(L' \cup \{v\}, R' \cup \{u\})$, where $u \in R^1_{enum} \backslash R''_1$, would be a k-biplex since (1) each vertex $v' \in L'$ disconnects at most k vertices from $R' \cup \{u\}$ (note that $R' \cup \{u\}$ is subset of R and v' disconnects at most k vertices from R), (2) v disconnects at most k vertices from $R' \cup \{u\}$ (note that v disconnects u and only those vertices in R'' and $|R''| < k$), (3) each vertex $u' \in R'$ disconnects at most k vertices from $L' \cup \{v\}$ (since $(L' \cup \{v\}, R')$ is a local solution) and (4) u disconnects at most k vertices from $L' \cup \{v\}$ (note that u disconnects at most $k - 1$ vertices from L since $u \in R^1_{enum}$ and u disconnects v since $u \in R_{enum}$). □

Lemma 4.3. In the subgraph $(L' \cup \{v\}, R')$, where $L' = L$ and $R' = R_{keep} \cup R''_1 \cup R''_2$, all vertices except for those in R''_2 disconnect at most k vertices and vertices in R''_2 disconnect exactly $(k + 1)$ vertices.

Proof. This can be verified as follows. (1) Each vertex $v' \in L$ disconnects at most k vertices in R' (this is because $\overline{\delta}(v', R) \leq k$ since (L, R) is a k-biplex, and $R' \subseteq R$). (2) Vertex v disconnects at most k vertices in R' (this is because $\overline{\delta}(v, R') = \overline{\delta}(v, R'' \cup R_{keep}) = \overline{\delta}(v, R'') \leq |R''| \leq k$. Note that v connects every vertex in R_{keep} by the definition of R_{keep}). (3) Each vertex $u \in R_{keep}$ disconnects at most k vertices in $L \cup \{v\}$ (this because u disconnects at most k vertices in L given that (L, R) is a k-biplex and u connects v by the definition of R_{keep}). (4) Each vertex u_1 in R''_1 disconnects at most k vertices in $L \cup \{v\}$ (this is because $\overline{\delta}(u_1, L) \leq k - 1$ by the

definition of R^1_{enum} and u_1 disconnects v). (5) Each vertex u_2 in R''_2 disconnects *exactly* $(k+1)$ vertices in $L \cup \{v\}$ (this is because $\overline{\delta}(u_2, L) = k$ by the definition of R^2_{enum} and u_2 disconnects v). □

C EXCLUSION STRATEGY

We summarize `iTraversal` with the exclusion strategy in Algorithm 4 and highlight the changes introduced by exclusion strategy. Basically, the exclusion strategy maintains an additional vertex set $Exc \subseteq \mathcal{L}$ which contains the vertices that must be excluded from a solution. As mentioned earlier, the improved three-step procedure `iThreeStep` is used to find solutions from one solution H. Among all the found solutions, the exclusion strategy further drops those solutions including vertices from Exc, i.e., we neither output these solutions (Line 11) nor recursively call `iThreeStep` from them (Line 12). This can be easily achieved by two steps, i.e., Line 6 and Line 10. The rationale behind is that those solutions can be outputted from another path from H_0. In addition, Exc is initialized as $\emptyset$ in line 1 and is expanded in line 13.

Algorithm 4: The algorithm: `iTraversal` (with exclusion strategy).

Input: Bipartite graph $G = (\mathcal{L} \cup \mathcal{R}, \mathcal{E})$, integer $k \geq 1$;
Output: All maximal k-biplexes;

1 **Initialize** $H_0 = (L_0, \mathcal{R})$, B-tree $\mathcal{T} = \{H_0\}$, $Exc = \emptyset$;
2 `iThreeStep`$(G, H_0, \mathcal{T}, Exc)$;
3 **Procedure** `iThreeStep`$(G, H, \mathcal{T}, Exc)$
4 (**Step 1**) **foreach** v *in* $V(G) \backslash V(H)$ **do**
5 **if** v *in* $\mathcal{R}$ **then Continue;** //Left-anchored traversal ;
6 **if** v *in* Exc **then Continue;** //Exclusion strategy ;
7 (**Step 2**) **foreach** H_{loc} *in* `EnumAlmostSat`$(G[H, v])$ **do**
8 **if** *there exists* u *in* $\mathcal{R} \backslash V(H_{loc})$ *s.t.* $G[H_{loc}, \{u\}]$ *is* k*-biplex* **then Continue;** //Right-shrinking traversal;
9 (**Step 3**) Extend H_{loc} to be a maximal k-biplex H' with vertices from $V(G) \backslash V(H_{loc}) \backslash \mathcal{R}$;
10 **if** $H' \notin \mathcal{T}$ *and* $V(H') \cap Exc = \emptyset$ **then**
11 Insert H' to $\mathcal{T}$;
12 `iThreeStep`$(G, H', \mathcal{T}, Exc)$;
13 $Exc \leftarrow Exc \cup \{v\}$;