Efficient Algorithms for Maximum k-Biplex Search on Bipartite Graphs

ABSTRACT

Enumerating maximal k-biplexes (MBPs) of a bipartite graph has been used for applications such as fraud detection. Nevertheless, there usually exists an exponential number of MBPs, which brings up two issues when enumerating MBPs, namely the effectiveness issue (many MBPs are of low values) and the efficiency issue (enumerating all MBPs is not affordable on large graphs). Existing proposals of tackling this problem impose constraints on the number of vertices of each MBP to be enumerated, yet they are still not sufficient (e.g., they require to specify the constraints, which is often not user-friendly, and cannot control the number of MBPs to be enumerated directly). Therefore, in this paper, we propose to find K MBPs with the most edges called MaxBPs, where K is a positive integral user parameter. The new proposal well avoids the drawbacks of existing proposals (i.e., the number of MBPs to be enumerated is directly controlled and the MBPs to be enumerated tend to have high values since they have more edges than the majority of MBPs). We formally prove the NP-hardness of the problem. While it is natural to adapt existing algorithms of MBP enumeration to the new problem, they all have the worst-case time complexity of $O^*(2^n)$, where O^* suppresses the polynomials and n is the number of vertices in the graph. We then design two *branch-and-bound* algorithms, among which, the better one called `FastBB` improves the worst-case time complexity to $O^*(\gamma_k^n)$, where γ_k is a real number that relies on k and is *strictly* smaller than 2. For example, for $k = 1$, γ_k is equal to 1.754. We further introduce three techniques for boosting the performance of the branch-and-bound algorithms, among which, the best one called `PBIE` can further improve the time complexity to $O^*(\gamma_k^{d^3})$ for large sparse graphs, where d is the maximum degree of the graph (note that $d << n$ for sparse graphs). We conduct extensive experiments on both real and synthetic datasets, and the results show that our algorithm is up to four orders of magnitude faster than all baselines and finding MaxBPs works better than finding all MBPs for a fraud detection application.

CCS CONCEPTS

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

KEYWORDS

bipartite graph; maximum biplex; maximum subgraph search

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

© 2018 Association for Computing Machinery.

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00

https://doi.org/XXXXXXX.XXXXXXX

ACM Reference Format:

. 2018. Efficient Algorithms for Maximum k-Biplex Search on Bipartite Graphs. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX).* ACM, New York, NY, USA, 17 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Bipartite graph is an important type of data structure where vertices are divided into two disjoint sets at two sides and each edge connects a vertex at one side and another at the other side. Bipartite graphs have been widely used for modeling the interactions between two types of entities in many real applications, with the entities being vertices and the interactions being edges. Some examples include commenting interactions between users and articles in social media [47], purchasing interactions between customers and products in E-commerce [33], visiting interactions between users and websites in web applications [4], etc.

A bipartite subgraph is called a k-biplex if each vertex at one side *disconnects* at most k vertices at the other side, where k is often a small positive integer [30, 44, 45]. k-biplex has been used to capture *dense/cohesive* subgraphs in a given bipartite graph for solving practical problems such as anomaly detection [12, 43], online recommendation [14, 28] and community search [15, 36]. For example, in e-commerce platforms, some agents are paid to promote the ranks of certain products by coordinating a group of fake users to post fake comments. The subgraphs induced by these fake users and the products they promote would be dense and likely to be k-biplexes [12].

Motivations. There are a few studies on the problem of enumerating *maximal k-biplexes* (MBPs) [30, 44, 45]. Nevertheless, there usually exist an exponential number of MBPs, which brings two issues of enumerating all MBPs. First, not all MBPs carry essential information (e.g., those MBPs with few vertices are often deemed not interesting [30]). Second, the process of enumerating all MBPs is costly (e.g., according to existing studies [44], when large graphs are used, enumerating all MBPs is not affordable). To mitigate these issues, existing studies [45] impose some constraints on the number of vertices at each of the two sides of a MBP to be enumerated, e.g., they enumerate only MBPs with *at least* a certain number vertices at each side. While this strategy makes it possible to control the number of MBPs to be enumerated, it achieves the goal only in an indirect way and introduces an additional issue of requiring to set proper thresholds of the number of vertices. In cases where users have no prior knowledge about the thresholds, they would find it not user-friendly. They can try different thresholds, but then the enumeration processes would run multiple times and bring up the time costs.

Motivated by these issues, in this paper, we propose to study the problem of finding K MBPs with the most edges, where K is an integral parameter. We call each of these MBPs a *maximum k-biplex* (MaxBP). Compared with existing studies on k-biplexes, the problem of finding MaxBPs enjoys three advantages. First, each

MaxBP to be found has more edges than those that are not returned, and thus the found MBP is of more significance. In our experiments, we verify this via a case study, which shows that a method based on MaxBPs provides F1 score up to 0.99 for a fraud detection task. Second, it provides a *direct* control on the number of MBPs to be found without the need of making multiple trials of enumeration. Third, compared with alternative proposals, e.g., finding the *first* K MBPs (as existing studies [45] do) or finding K MBPs with the most vertices, our solution to the problem of finding MaxBPs would return MBPs that are more balanced, which are deemed to be superior over imbalanced structures [23].

Baseline Methods. Given the fact that MaxBPs are the MBPs with the most edges, we can adapt the existing algorithms of enumerating MBPs, namely `iMB` [44] and `iTraversal` [45], to find the MaxBPs and yield the following two baseline methods. The first one called `iMBadp` adapts `iMB` [44] (a branch-and-bound algorithm) by incorporating additional pruning techniques which prune those branches that cannot hold any MBP with more edges than the K MaxBPs found so far. The second baseline called `iTradp` simply runs `iTraversal` [45] (a reverse search based algorithm) and returns K MaxBPs since `iTraversal` cannot be equipped with pruning techniques easily. Nevertheless, `iMBadp` has its efficiency highly rely on the pruning techniques and `iTradp` needs to explore all MBPs, which is time-consuming. Both of them have the worst-case time complexity of $O^*(2^n)$, where n is the number of vertices in the given bipartite graph and O^* suppresses polynomials. Furthermore, we can adapt those existing algorithms of enumerating maximal k-plexes, which are counterparts of MBPs on general graphs. Here, a k-plex is a subgraph with each vertex disconnecting at most k vertices in the subgraph. Therefore, the third baseline method called `FPadp` adapts `FaPlexen` [51] (the state-of-the-art algorithm for enumerating maximal k-plexes) with some additional pruning techniques that are tailored for MBPs. Still, `FPadp` is inferior to the new algorithms we will introduce in this paper both theoretically and empirically.

New Methods. We first introduce a *branch-and-bound* algorithm called `BasicBB`, which is based on a conventional and widely-used branching strategy that we call *Bron-Kerbosch (BK) branching* [5]. The BK branching recursively partitions the search space (i.e., the set of all possible MBPs) to multiple sub-spaces via *branching*. `BasicBB` has the worst-case time complexity $O(n \cdot d \cdot 2^n)$ (i.e., $O^*(2^n)$), where d is the maximum degree of the graph. This time complexity is the same as those of the baseline methods. We then introduce a new branching strategy called *Symmetric-BK (Sym-BK) branching*, which is symmetric to the BK branching but better suits our problem of finding MaxBPs. We further present our method for determining an ordering of vertices, which is critical for Sym-BK branching. We finally introduce a new branch-and-bound algorithm called `FastBB`, which is based on the Sym-BK branching. `FastBB` has its worst-case time complexity $O(n \cdot d \cdot \gamma_k^n)$ (i.e., $O^*(\gamma_k^n)$), where γ_k is *strictly* smaller than 2 and depends on the setting of k. For example, when $k = 1$, γ_k is 1.754. This is a remarkable theoretical improvement over the prior solutions given that many existing algorithms of enumerating subgraphs are based on BK branching and have the worst-case time complexity of $O^*(2^n)$ [30, 39, 44].

In addition, we adapt two existing techniques for boosting the efficiency and scalability of the branch-and-bound (`BB`) algorithms including `BasicBB` and `FastBB`. They share the idea of constructing *multiple* problem instances of finding MaxBPs each on a smaller subgraph. Specifically, the first technique, called *progressive bounding* (`PB`), is adapted from an existing study of finding the biclique with the most edges [23]. The second technique, called *inclusion-exclusion* (`IE`), is adapted from the *decomposition* technique, which has been widely used for enumerating and finding subgraph structures [6, 8, 40]. `PB` improves the practical performance of a `BB` algorithm only while `IE` improves both the theoretical time complexity (for certain sparse graphs) and the practical performance. We then present `PBIE`, which combines `PB` and `IE` naturally. `PBIE` enjoys the benefits of both `PB` and `IE`. We note that all these techniques are orthogonal to the `BB` algorithms, i.e., any of these techniques, namely `PB`, `IE`, and `PBIE`, can be used to boost the efficiency and/or scalability of `BasicBB` and `FastBB`. To the best of our knowledge, this is first time that `PB` and `IE` are combined naturally.

Contributions. Our major contributions are summarized below.

- This is the first study on the problem of finding MaxBPs. We formally prove the NP-hardness of the problem.
- We propose an efficient branch-and-bound algorithm, called `FastBB`, which is based on a novel Sym-BK branching strategy. In particular, `FastBB` achieves the state-of-the-art worst-case time complexity $O(n \cdot d \cdot \gamma_k^n)$ with $\gamma_k < 2$.
- We further introduce a combined framework, called `PBIE`, to further boost the performance of `FastBB`. `PBIE` combines two adapted frameworks, namely the progressive bounding framework `PB` and the inclusion-exclusion based framework `IE`. When `PBIE` is used with `FastBB`, the worst-time time complexity becomes $O(d^4 \cdot \gamma_k^{d^3})$. Note that this is better than that of `FastBB` on certain graphs (e.g., those sparse graphs with $d << n$).
- We conduct extensive experiments using both real and synthetic datasets, and the results show that (1) the proposed algorithms are up to four orders of magnitude faster than all baselines and (2) finding MaxBPs work better in a fraud detection task than enumerating MBPs.

Roadmap. The rest of this paper is organized below. Section 2 defines the problem and shows its NP-hardness. Section 3 presents the branch-and-bound algorithms `BasicBB` and `FastBB`. Section 4 presents the frameworks `PB`, `IE` and `PBIE`. We conduct extensive experiments in Section 5. Section 6 reviews the related work and Section 7 concludes the paper.

2 PROBLEM DEFINITION

Let $G = (L \cup R, E)$ be an undirected and unweighted bipartite graph, where L and R are two disjoint vertex sets and E is an edge set. For the graph G, we use $V(G)$, $L(G)$, $R(G)$, and $E(G)$ to denote its set of vertices, left side, right side and set of edges, respectively, i.e., $V(G) = L \cup R$, $L(G) = L$, $R(G) = R$, and $E(G) = E$. Given $X \subseteq L$ and $Y \subseteq R$, we use $G[X \cup Y]$ to denote the induced (bipartite) graph of G, i.e., $G[X \cup Y]$ includes the set of vertices $X \cup Y$ and the set of edges between X and Y. All subgraphs considered in this paper are induced subgraphs. We use H or (X, Y) as a shorthand of $H = G[X \cup Y]$.

Given $v \in L$, we use $\Gamma(v, R)$ (resp. $\overline{\Gamma}(v, R)$) to denote the set of neighbours (resp. non-neighbours) of v in R, i.e., $\Gamma(v, R) = \{u \mid (v, u) \in E \text{ and } u \in R\}$ (resp. $\overline{\Gamma}(v, R) = \{u \mid (v, u) \notin E \text{ and } u \in R\}$). We define $\delta(v, R) = |\Gamma(v, R)|$ and $\overline{\delta}(v, R) = |\overline{\Gamma}(v, R)|$. We use d to denote the maximum degree of vertex in G. We have symmetric definitions for each vertex $u \in R$.

Next, we review the definition of k-biplex [44].

Definition 1 (k-biplex [44]). *Given a graph $G = (L, R, E)$, a positive integer k, $X \subseteq L$ and $Y \subseteq R$, an induced subgraph $G[X \cup Y]$ is said to be a k-biplex if $\overline{\delta}(v, Y) \leq k$, $\forall v \in X$ and $\overline{\delta}(u, X) \leq k$, $\forall u \in Y$.*

A k-biplex H is said to be maximal if there is no other k-biplex H' containing H, i.e., $V(H) \subseteq V(H')$. Large real graphs usually involve numerous maximal k-biplexes and most of them highly overlap. In this paper, we aim to find K maximal k-biplexes with the most edges, where K is a positive integral user-parameter. In addition, we consider two size constraints θ_L and θ_R on each maximal k-biplex H to be found, namely $|L(H)| \geq \theta_L$ and $|R(H)| \geq \theta_R$. These constraints would help to filter out some skewed maximal k-biplexes, i.e., the number of vertices at one side is extremely larger than that at the other side. To guarantee that all found maximal k-biplexes are connected, we further require $\theta_L \geq 2k + 1$ and $\theta_R \geq 2k + 1$ based on the following lemma.

Lemma 1. *A k-biplex H is connected if $|L(H)| \geq 2k + 1$ and $|R(H)| \geq 2k + 1$.*

Proof. This can be proved by contradiction. Suppose H is not connected and is partitioned into two connected components, namely H_1 and H_2. We derive the contradiction by showing that H is not a k-biplex: for a vertex v_1 in $L(H_1)$, it disconnects more than k vertices, i.e., $\overline{\delta}(v_1, R(H)) \geq |R(H_2)| \geq k + 1$. Specifically, we derive $\overline{\delta}(v_1, R(H)) \geq |R(H_2)|$ since v_1 from H_1 disconnects all vertices in H_2 based on the assumption, and we derive $|R(H_2)| \geq k + 1$ by (1) for a vertex v_2 in L_2, $|R(H_2)| \geq \delta(v_2, R(H))$ since all neighbours of v_2 within H reside in $R(H_2)$ based on the assumption and (2) $\delta(v_2, R(H)) \geq R(H) - k \geq 2k + 1 - k \geq k + 1$ since v_2 disconnects at most k vertices (H is a k-biplex) and $R(H) \geq 2k + 1$. □

We formalize the problem studied in this paper as follows.

Problem 1 (Maximum k-biplex Search). *Given a bipartite graph $G = (L \cup R, E)$, four positive integers $K > 0$, $k > 0$, $\theta_L \geq 2k + 1$ and $\theta_R \geq 2k + 1$, the maximum k-biplex search problem aims to find K maximal k-biplexes such that each found maximal k-biplex H satisfies that $|L(H)| \geq \theta_L$, $|R(H)| \geq \theta_R$ and $|E(H)|$ is larger than $|E(H')|$ for any other maximal k-biplex H' that is not returned.*

In this paper, we use MBP and MaxBP as a shorthand of a maximal k-biplex and one of the K maximal k-biplexes with the most edges, respectively.

NP-hardness. The maximum k-biplex search problem is NP-hard, which we present in the following lemma.

Lemma 2. *The maximum k-biplex search problem is NP-hard.*

Proof. We prove by showing a polynomial reduction from a well-known NP-complete problem, namely *maximum clique search*, to the maximum k-biplex search problem. Note that we consider a maximum k-biplex search problem with $\theta_L = \theta_R = 0$ and $K = 1$ in the proof. We define their decision problems as follows.

- CLIQUE: given a general graph $G = (V, E)$ and a positive integer α, does G contain a clique with at least α vertices?
- BIPLEX: given a bipartite graph $\mathcal{G} = (\mathcal{L} \cup \mathcal{R}, \mathcal{E})$ and two positive integers k and α', does $\mathcal{G}$ contain a k-biplex with at least α' edges?

For simplicity, we show a polynomial reduction from CLIQUE to BIPLEX with $k = 1$. We can prove general cases similarly.

Let $G = (V, E)$ and α be the inputs of an instance of CLIQUE. W.l.o.g., we assume that $\alpha = \frac{1}{2}|V|$ is a positive integer. This can be achieved with some inflation tricks. Specifically, (1) if the original input α is smaller than $\frac{1}{2}|V|$ (which can be a fractional number), we add to G a new vertex v and $|V|$ edges between v and other vertices. Obviously, every clique would include v, increasing α by 1 but $\frac{1}{2}|V|$ by 0.5. (2) If $\alpha > \frac{1}{2}|V|$, we add to G a new vertex v with no edges. Clearly, every clique would not include v, increasing $\frac{1}{2}|V|$ by 0.5 only. By repeating above two steps, we can make $\alpha = \frac{1}{2}|V|$ finally.

Now, we construct an instance of BIPLEX (with $k = 1$) with $\mathcal{G} = (\mathcal{L} \cup \mathcal{R}, \mathcal{E})$ and α'. To be specific,

$$\mathcal{L} = V \text{ and } \mathcal{R} = E \cup W \cup U,\ |W| = \frac{1}{2}\alpha(\alpha - 5),\ |U| = 2\alpha$$

where $W \cup U$ is a set of new elements and $|V| = 2\alpha$. Assume $V = \{v_1, ..., v_{2\alpha}\}$ and $W = \{w_1, ..., w_{2\alpha}\}$. We have

$$\begin{aligned}\mathcal{E} &= \mathcal{E}(\mathcal{G}[V \cup E]) \cup \mathcal{E}(\mathcal{G}[V \cup W]) \cup \mathcal{E}(\mathcal{G}[V \cup U]) \\ &= \{(v, e) \mid v \in V, e \in E, v \notin e\} \cup \{(v, w) \mid v \in V, w \in W\} \\ &\quad \cup \{(v_i, u_j) \mid v_i \in V, u_j \in U, i \neq j\}, \\ \alpha' &= \frac{1}{2}\alpha^3 + \frac{3}{2}\alpha^2 - \alpha.\end{aligned}$$

To guarantee $|W| \geq 0$, we assume $\alpha \geq 5$ with the inflation tricks. The above construction can be finished in polynomial time.

We then show that *G has a clique with at least α vertices iff $\mathcal{G}$ has a 1-biplex with at least α' edges*. We consider the following two cases.

Case 1: G has a clique $G[C]$ with α vertices, i.e., $C \subseteq V$ and $|C| = \alpha$. Consider $X = V \backslash C$ and $Y = E(G[C]) \cup W \cup U$. We prove this case by showing that $\mathcal{G}[X \cup Y]$ is a 1-biplex with α' edges. To be specific, for each vertex $v \in X$, we know: (1) vertex v connects all vertices from $E(G[C])$ since v is not an endpoint of any edge in $E(G[C])$, thereby yielding $|X| \times |E(G[C])|$ edges in total; (2) vertex v connects all vertices from W based on the construction, yielding $|X| \times |W|$ edges in total; and (3) vertex v disconnects exactly one vertex in U based on the construction, yielding $|X| \times (|U| - 1)$ edges in total. For each vertex $u \in Y$, we can similarly verify that vertex u disconnects no more than one vertex from X. In summary, we have a 1-biplex $\mathcal{G}[X \cup Y]$ with

$$|\mathcal{E}(\mathcal{G}[X \cup Y])| = |X| \times |E(G[C])| + |X| \times |W| + |X| \times (|U| - 1),$$

where $|X| = \alpha$, $|E(G[C])| = \frac{1}{2}\alpha(\alpha - 1)$, $|W| = \frac{1}{2}\alpha(\alpha - 5)$ and $|U| = 2\alpha$. Therefore, $|\mathcal{E}(\mathcal{G}[X \cup Y])|$ is exactly α'.

Case 2: G does not have a clique with at least α vertices. If no 1-biplex is found in $\mathcal{G}$, the proof is finished. Otherwise, let $\mathcal{G}[X \cup Y]$ be an arbitrary 1-biplex in $\mathcal{G}$ such that $X \subseteq \mathcal{L}$ and $Y \subseteq \mathcal{R}$. We finish the proof by showing that $|\mathcal{E}(\mathcal{G}[X \cup Y])| < \alpha'$. To estimate $|\mathcal{E}(\mathcal{G}[X \cup$

$Y])|$, we divide Y into two disjoint parts $Y_0 \cup Y_1$, i.e., vertices in Y_0 connect all vertices from X and vertices in Y_1 disconnect exactly one vertex from X. For Y_0, we have: (1) it includes all vertices from $E(G[V\backslash X]) \subseteq E$ since every edge in $E(G[V\backslash X])$ has no endpoint in X; (2) it includes all vertices from U based on our construction; and (3) it includes $|V\backslash X|$ vertices from W (with $X = \{v_1, ..., v_{|X|}\}$, vertices in $\{w_{|X|+1}, ..., w_{|V|}\}$ would connect all vertices from X based on our construction). For Y_1, it includes at most $|X|$ vertices since otherwise there exists at least a vertex in X that disconnects at least 2 vertices based on the pigeonhole principle. This contradicts to the definition of 1-biplex $\mathcal{G}[X \cup Y]$. In summary, we have

$$\begin{aligned}|\mathcal{E}(\mathcal{G}[X \cup Y])| &\le |X| \times |Y_0| + (|X| - 1) \times |Y_1| \\ &= |X|\times(|E(G[V\backslash X])|+|U|+|V\backslash X|) + (|X|-1)\times|X| \qquad (1)\end{aligned}$$

Let $x = |X|$. We have $0 \le x \le 2\alpha$ and consider two cases.
Case 2.1: $\alpha < x \le 2\alpha$. Let $y = x - \alpha$ $(0 < y \le \alpha)$. We have $|X| = \alpha + y$ and $|V\backslash X| = \alpha - y$. Moreover, we can obtain $|E(G[V\backslash X])| \le \frac{1}{2}|V\backslash X|(|V\backslash X| - 1) = \frac{1}{2}(\alpha - y)(\alpha - y - 1)$ where the equality holds iff $G[V\backslash X]$ is a clique. According to equation (1), we have $|\mathcal{E}(\mathcal{G}[X \cup Y])| \le (\alpha + y)[\frac{1}{2}(\alpha - y)(\alpha - y - 1) + \frac{1}{2}\alpha^2 - \frac{5}{2}\alpha + 2\alpha - (\alpha + y)] + (\alpha + y - 1)(\alpha + y)$ which reduces to

$$|\mathcal{E}(\mathcal{G}[X \cup Y])| - \alpha' \le \frac{1}{2}y[y^2 - (\alpha - 1)y - \alpha - 2].$$

It is easy to verify that $y^2 - (\alpha-1)y - \alpha - 2$ is negative for $0 \le y \le \alpha$. Therefore, we have $|\mathcal{E}(\mathcal{G}[X \cup Y])| < \alpha'$.
Case 2.2: $0 \le x \le \alpha$. Let $y = \alpha - x$ $(0 \le y \le \alpha)$. We have $|X| = \alpha - y$ and $|V\backslash X| = \alpha + y$. Since $|V\backslash X| = 2\alpha - x \ge \alpha$ and G does not have a clique with at least α vertices, it is easy to verify that $|E(G[V\backslash X])| < \frac{1}{2}|V\backslash X|(|V\backslash X| - 1) - y$ since otherwise there exists a clique with at least α vertices in G (note that the right term can be regarded as a process of iteratively removing y edges from a clique with $|V\backslash X|$ vertices and after removing an edge, the maximum clique in the remaining graph has its size decrease by at most 1, which yields a clique with $|V\backslash X| - y = \alpha + y - y = \alpha$ vertices). According to equation (1), we have $|\mathcal{E}(\mathcal{G}[X \cup Y])| < (\alpha - y)[\frac{1}{2}(\alpha + y)(\alpha + y - 1) - y + \frac{1}{2}\alpha^2 - \frac{5}{2}\alpha + 2\alpha - (\alpha - y)] + (\alpha - y - 1)(\alpha - y)$ which reduces to

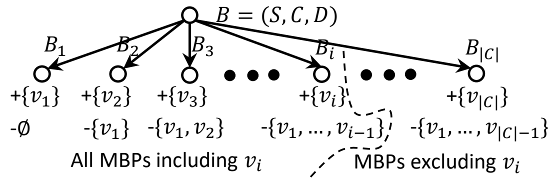
$$|\mathcal{E}(\mathcal{G}[X \cup Y])| - \alpha' < \frac{1}{2}y[-y^2 - (\alpha - 3)y - \alpha + 2].$$

It is easy to verify that $y[-y^2 - (\alpha - 3)y - \alpha + 2] \le 0$ for $0 \le y \le \alpha$ and $\alpha \ge 5$. We thus have $|\mathcal{E}(\mathcal{G}[X \cup Y])| < \alpha'$. □

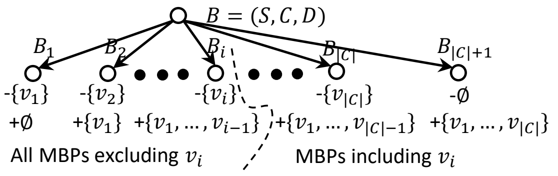
Remarks. In the following sections (Section 3 and Section 4), we focus on the setting of $K = 1$ (i.e., the problem becomes to find a MBP with the maximum number of edges) when presenting the algorithms for ease of presentation. We note that these algorithms can be naturally extended for general settings of K with minimal efforts (i.e., we maintain K MaxBPs instead of 1 MaxBP found so far throughout the algorithm for pruning) and are tested in Section 5.

3 BRANCH-AND-BOUND ALGORITHMS

We first introduce a *branch-and-bound* algorithm called `BasicBB`, which is based on a conventional and widely-used branching strategy that we call *Bron-Kerbosch (BK) branching* [5], in Section 3.1. `BasicBB` has the worst-case time complexity $O(|V| \cdot d \cdot 2^{|V|})$. We



(a) BK Branching



(b) Sym-BK Branching

Figure 1: Illustration of two branching strategies. Each node denotes a branch (S, C, D). The notation "+" means to include a vertex by adding it S and "-" means to exclude a vertex by adding it to D.

then introduce a new branching strategy called *Symmetric-BK (Sym-BK) branching*, which is symmetric to the BK branching but better suits our problem of finding MaxBP, in Section 3.2. We further present our method for determining an ordering of vertices, which is critical for Sym-BK branching, in Section 3.3. We finally introduce a new branch-and-bound algorithm called `FastBB`, which is based on Sym-BK branching, and analyze its time complexity in Section 3.4. `FastBB` has its worst-case time complexity $O(|V| \cdot d \cdot \gamma_k^{|V|})$, where γ_k is strictly smaller than 2 and depends on the setting k. For example, when $k = 1$, γ_k is 1.754.

3.1 A BK Branching based Branch-and-Bound Algorithm: `BasicBB`

Our first attempt is to adapt the seminal *Bron-Kerbosch* (BK) algorithm [5]. It recursively partitions the search space (i.e., the set of all possible MBPs) to multiple sub-spaces via *branching*. Specifically, each sub-space is represented by a triplet of three sets as explained below.

- **Partial set** S. Set of the vertices that *must* be included in every MBP within the space.
- **Candidate set** C. Set of the vertices that *can* be included in S in order to form a MBP within the space.
- **Exclusion set** D. Set of vertices that *must not* be included in any MBP within the space.

We further denote by S_L, C_L, and D_L the left side of S, C, and D, respectively, and define S_R, C_R, and D_R similarly for the right side.

The recursive process of the BK algorithm starts from the full search space with $S = \emptyset$, $C = V$, and $D = \emptyset$. Consider the branching step at a current branch $B = (S, C, D)$. Let $\langle v_1, v_2, ..., v_{|C|}\rangle$ be a sequence of the vertices in C. The branching step would partition the space to $|C|$ sub-spaces (and correspondingly $|C|$ branches), where the i^{th} branch, denoted by $B_i = (S_i, C_i, D_i)$, *includes* S and v_i and *excludes* $v_1, v_2, ..., v_{i-1}$. Formally, we have

$$S_i = S \cup \{v_i\};\ D_i = D \cup \{v_1, v_2, ..., v_{i-1}\};\ C_i = C - \{v_1, v_2, ..., v_i\} \quad (2)$$

For illustration, consider Figure 1(a).

We call the above branching strategy *BK branching*. BK branching essentially corresponds to a recursive *binary* branching process. It first splits the current branch into two branches, one including v_1 (this is the branch B_1) and the other excluding v_1. Then, it further splits the latter into two branches, one including v_2 (this is the branch B_2) and the other excluding v_2. It continues the process until the last branch, which excludes $v_1, v_2, ..., v_{|C|-1}$ and includes $v_{|C|}$ (this corresponds to the branch $B_{|C|}$), is formed. In particular, the branches $B_1, B_2, ..., B_i$ cover all MBPs including v_i and branches $B_{i+1}, ..., B_{|C|}$ cover those excluding v_i, as indicated by the dashed line in Figure 1(a).

We note that BK branching relies on an ordering of vertices in the candidate set C, i.e., $\langle v_1, v_2, ..., v_{|C|}\rangle$, for producing branches. In this paper, we follow the existing studies [1, 48] and use the *non-decreasing vertex ordering* (where vertices are ranked in an ascending order of their degrees in $S\cup C$, i.e., $\delta(v_i, S\cup C) \le \delta(v_j, S\cup C)$ for any $i < j$) since this would help with effective pruning as shown empirically. We note that normally the ordering does not affect the theoretical time complexity of the algorithm based on BK branching.

During the recursive search process, some pruning techniques can be applied. Let $B = (S, C, D)$ be a branch. First, branch B can be pruned if S is not a k-biplex since (1) each partial set in the search space corresponding to this branch is a *superset* of S and (2) based on the hereditary property of k-biplex, any superset of a non-k-biplex is not a k-biplex. Second, branch B can be pruned if an upper bound of the left side (resp. the right side) of a k-biplex in the space is smaller than θ_L (resp. θ_R) based on the problem definition. Third, branch B can be pruned if an upper bound of the number of edges in a k-biplex in the space is smaller than the largest one of a k-biplex known so far. Forth, branch B can be pruned if there exists a vertex in D such that including this vertex to each k-biplex in the space would still result in a k-biplex. We will elaborate on these pruning rules in detail in Section 3.4. The recursive process of the BK algorithm terminates at a branch $B = (S, C, D)$ if $G[S \cup C]$ is a k-biplex since $G[S \cup C]$ would be the MaxBP within the space of the branch.

We call this BK algorithm, which is a *branch-and-bound* algorithm based on BK branching and pruning techniques, BasicBB, and present its pseudo-code in Algorithm 1. Similar to many existing algorithms that are based on BK branching, the worst-case time complexity of BasicBB is $O(|V| \cdot d \cdot 2^{|V|})$ (i.e., $O^*(2^{|V|})$) [30, 44], though its practical performance can be boosted by the the pruning techniques.

Algorithm 1: The branch-and-bound algorithm based on BK branching: BasicBB

```
Input: A graph $G(L \cup R, E)$, $k$, $\theta_L$ and $\theta_R$
Output: The maximal $k$-biplex $H^*$ with the most edges
1 $H^* \leftarrow G[\emptyset]$; // Global variable
2 BasicBB-Rec($\emptyset, L \cup R, \emptyset$); return $H^*$;
3 Procedure BasicBB-Rec($S, C, D$)
      /* Termination */
4     if $G[S \cup C]$ is $k$-biplex then
5         $H^* \leftarrow G[S \cup C]$ if $|E(G[S \cup C])| > |E(H^*)|$; return;
      /* Pruning */
6     if any of pruning conditions is satisfied (details in Section 3.4)
        then return ;
      /* BK Branching */
7     Create $|C|$ branches $B_i = (S_i, C_i, D_i)$ based on Equation (2);
8     for each branch $B_i$ do
9         Basic-Rec($S_i, C_i, D_i$);
```

3.2 A New Branching Strategy: Sym-BK Branching

We observe that there exists a branching strategy, which is natural and *symmetric* to BK branching. Specifically, consider the branching step at a current branch $B = (S, C, D)$. Let $\langle v_1, v_2, ..., v_{|C|}\rangle$ be a sequence of the vertices in C. The branching step would partition the space to $(|C| + 1)$ sub-spaces (and correspondingly $(|C| + 1)$ branches), where the i^{th} branch, denoted by $B_i = (S_i, C_i, D_i)$, *includes* S and $v_1, v_2, ..., v_{i-1}$ and *excludes* v_i. Here, v_0 and $v_{|C|+1}$ both correspond to null. Formally, we have

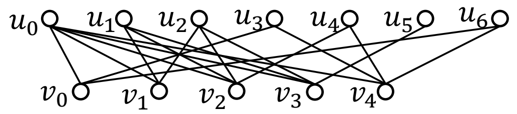
$$S_i = S \cup \{v_1, v_2, ..., v_{i-1}\};\ D_i = D \cup \{v_i\};\ C_i = C - \{v_1, v_2, ..., v_i\} \quad (3)$$

For illustration, consider Figure 1(b).

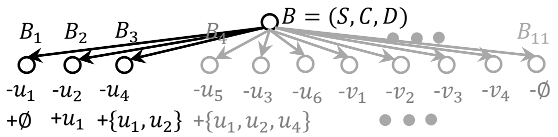
We call the above branching strategy *symmetric-BK (Sym-BK) branching*. Sym-BK branching corresponds to another recursive *binary* branching process, which is symmetric to that of the BK branching. Specifically, it first splits the current branch into two branches, one excluding v_1 (this is the branch B_1) and the other including v_1. Then, it further splits the latter into two branches, one excluding v_2 (this is the branch B_2) and the other including v_2. It continues the process until the last branch, which includes $v_1, v_2, ..., v_{|C|}$ (this corresponds to the branch $B_{|C|+1}$), is formed. In particular, the branches $B_1, B_2, ..., B_i$ cover all MBPs *excluding* v_i and branches $B_{i+1}, ..., B_{|C|+1}$ cover those *including* v_i, as indicated by the dashed line in Figure 1(b).

Sym-BK branching vs. BK branching. They are symmetric to each other and both of them are natural branching strategies. They differ in that among two branches formed by a binary branching based on a vertex, BK branching recursively partitions the branch *excluding* the vertex while Sym-BK recursively partitions one *including* the vertex. Sym-BK branching produces one more branch than BK branching does at each branching step (i.e., $|C| + 1$ branches vs. $|C|$ branches), but this difference of one extra branch is negligible given that there can be many branches produced at the branching step. Compared with BK branching, Sym-BK branching has the following advantages when adopted for our problem of finding the MaxBP.

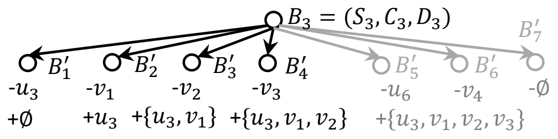
First, at each branching step, it would produce branches with *bigger* partial sets S_i (note that the i^{th} branch by Sym-BK branching involves $|S| + (i - 1)$ vertices in the partial set while that by BK branching involves $|S| + 1$ vertices). Consequently, the produced branch would have a larger chance to be pruned due to the hereditary property of k-biplex (if a set of vertices is not a k-biplex, then none of its supersets is, but not vice versa). Second, the partial set of the j^{th} branch, i.e., S_j, is always a superset of that of the i^{th} branch, i.e., S_i, for any $j > i$. Consequently, if S_i is not a k-biplex (which



(a) Input graph used throughout the paper


$B = (S, C, D)$
B_1
B_2
B_3
B_4
B_{11}
-u_1 -u_2 -u_4 -u_5 -u_3 -u_6 -v_1 -v_2 -v_3 -v_4 -$\emptyset$
+$\emptyset$ +u_1 +$\{u_1, u_2\}$ +$\{u_1, u_2, u_4\}$

(b) Case 1 ($S = \{u_0, v_0\}$, $C = \{u_1, u_2, u_4, u_5, u_3, u_6, v_1, v_2, v_3, v_4\}$ and $D = \emptyset$)


$B_3 = (S_3, C_3, D_3)$
B'_1
B'_2
B'_3
B'_4
B'_5
B'_6
B'_7
-u_3 -v_1 -v_2 -v_3 -u_6 -v_4 -$\emptyset$
+$\emptyset$ +u_3 +$\{u_3, v_1\}$ +$\{u_3, v_1, v_2\}$ +$\{u_3, v_1, v_2, v_3\}$

(c) Case 2 ($S_3 = \{u_0, v_0, u_1, u_2\}$, $C_3 = \{u_3, v_1, v_2, v_3, u_6, v_4\}$ and $D_3 = \{u_4, u_5\}$)

Figure 2: Illustration of two cases of Sym-BK branching ($k = 2$).

means the branch B_i can be pruned), then all branches following B_i can be pruned (since their partial sets are supersets of S_i and thus they are not k-biplexes either based on the hereditary property).

To illustrate, consider the example in Figure 2. One branching step of Sym-BK branching is shown in Figure 2(b). The fourth branch has the partial set of $S_4 = \{u_0, v_0, u_1, u_2, u_4\}$, which is not a k-biplex. All the following branches have their partial sets as supersets of S_4, and thus they can be pruned immediately (as indicated by the shaded color in the figure).

In fact, with Sym-BK branching and a carefully-designed ordering of vertices (details will be introduced in Section 3.3), our new branch-and-bound algorithm `FastBB` would have the worst-case time complexity of $O(|V| \cdot d \cdot \gamma_k^{|V|})$ with $\gamma_k < 2$, which is strictly smaller than that of the `BasicBB` algorithm based on BK branching (details will be introduced in Section 3.4).

Remarks. In [51], the authors design a branching strategy, which performs one of two branching operations at a branch depending on the situation: (1) generating two branches based on a *single* vertex in C (i.e., one including and the other not including this vertex) and (2) generating at most $(|C| + 1)$ branches based on all vertices in C with pruning. We call this branching strategy *hybrid branching*. Our Sym-BK branching is superior over hybrid branching for our problem in two aspects. First, Sym-BK branching has a more simplified form. Second, the branch-and-bound algorithm based on Sym-BK branching has lower worst-case time complexity *theoretically* (details are in Section 3.4) and runs faster *empirically* (details are in Section 5).

3.3 Sym-BK Branching: Ordering of Vertices

The Sym-BK branching relies on an ordering of the vertices in C. Recall that in the branches $B_1, B_2, ..., B_{|C|+1}$ generated by Sym-BK branching, the partial set of a branch B_j is always a superset of that of a preceding branch B_i ($i < j$). Therefore, our idea is to figure out a small subset C' of vertices in C such that including them *collectively* to the partial set S would violate the k-biplex definition. We then put these vertices *before* other vertices in the ordering. In this way, the branch with the partial set of $S \cup C'$ and all the following branches can be pruned directly. We elaborate on this idea in detail next.

We notice that $G[S \cup C]$ is not a k-biplex since otherwise the recursion would terminate at this branch. It means that there exists at least a vertex in $S \cup C$, which has more than k disconnections within $G[S \cup C]$. Without loss of generality, we assume that the vertex is from the left side and denote it by $\hat{v} \in S_L \cup C_L$. Consider the set of vertices that disconnect $\hat{v}$ in C_R, i.e., $\overline{\Gamma}(\hat{v}, C_R)$. We know that including $\overline{\Gamma}(\hat{v}, C_R)$ to S collectively would violate the k-biplex definition. Specifically, if $\hat{v}$ is already in S_L, i.e., $\hat{v} \in S_L$, we can include at most $k - \overline{\delta}(\hat{v}, S_R)$ vertices from $\overline{\Gamma}(\hat{v}, C_R)$ to S without violating the k-biplex definition; and if $\hat{v}$ is not yet in S_L, i.e., $\hat{v} \in C_L$, we can include $\hat{v}$ together with at most $k - \overline{\delta}(\hat{v}, S_R)$ vertices from $\overline{\Gamma}(\hat{v}, C_R)$ to S without violating the k-biplex definition. For the simplicity of notations, we define

$$a = k - \overline{\delta}(\hat{v}, S_R); \; b = \overline{\delta}(\hat{v}, C_R). \tag{4}$$

Intuitively, a means the greatest possible number of disconnections that $\hat{v}$ can have when including more vertices from C to S for forming MBPs. Note that we have $0 \le a \le k$ (since S is a k-biplex and thus $\overline{\delta}(\hat{v}, S_R) \le k$) and $a < b$ (since $b - a = \overline{\delta}(\hat{v}, S_R \cup C_R) - k > 0$). Based on $\hat{v}$, we define an ordering of the vertices in C.

Case 1: $\hat{v} \in S_L$. In this case, we define the ordering as follows.

$$\langle u_1, u_2, ..., u_b, u_{b+1}, ..., u_{|C|} \rangle, \tag{5}$$

where $u_1, u_2, ..., u_b$ are vertices from $\overline{\Gamma}(\hat{v}, C_R)$ in any order and $u_{b+1}, u_{b+2}, ..., u_{|C|}$ are vertices from $C_R - \overline{\Gamma}(\hat{v}, C_R)$ in any order. Based on this ordering, the branch B_{a+2} would have the partial set $S \cup \{u_1, u_2, ..., u_{a+1}\}$, which is not a k-biplex (since $\hat{v}$ would have more than k disconnections). Therefore, branches $B_{a+2}, B_{a+3}, ..., B_{|C|+1}$ can be pruned and only the first $(a + 1)$ branches, namely $B_1, B_2, ..., B_{a+1}$, would be kept. To illustrate, we consider a branch with $S = \{u_0, v_0\}$, $D = \emptyset$ and $C = \{u_1, u_2, u_3, u_4, u_5, u_6, v_1, v_2, v_3, v_4\}$ for finding a MaxBP with $k = 2$ from the input graph in Figure 2(a). Based on v_0 in S that disconnects 3 vertices, i.e., $\{u_1, u_2, u_4\}$, we define the ordering $\langle u_1, u_2, u_4, u_5, u_3, u_6, v_1, v_2, v_3, v_4 \rangle$ for Sym-BK branching as shown in Figure 2(b). The branches $B_4, ..., B_{11}$ can be pruned since B_4 has the partial set $\{u_0, u_1, u_2, u_4, v_0\}$ not a k-biplex (v_0 has more than $k = 2$ disconnections).

Case 2: $\hat{v} \in C_L$. In this case, we define the ordering as follows.

$$\langle \hat{v}, u_1, u_2, ..., u_b, u_{b+2}, ..., u_{|C|} \rangle, \tag{6}$$

where $u_1, u_2, ..., u_b$ are vertices from $\overline{\Gamma}(\hat{v}, C_R)$ in any order and $u_{b+2}, u_{b+2}, ..., u_{|C|}$ are vertices from $C_R - \overline{\Gamma}(\hat{v}, C_R) - \{\hat{v}\}$ in any order. Based on this ordering, the branch B_{a+3} would have the partial set as $S \cup \{\hat{v}, u_1, u_2, ..., u_{a+1}\}$, which is not a k-biplex (since $\hat{v}$ would have more than k disconnections). Therefore, branches $B_{a+3}, B_{a+4}, ..., B_{|C|+1}$ can be pruned and only the first $(a + 2)$ branches, namely $B_1, B_2, ..., B_{a+2}$, would be kept. To illustrate, we consider another branch B_3 with $S_3 = \{u_0, u_1, u_2, v_0\}$, $C_3 = \{u_3, u_6, v_1, v_2, v_3, v_4\}$ and $D_3 = \{u_4, u_5\}$ in Figure 2(c). Based on u_3 in C that disconnects 3 vertices, i.e., $\{v_1, v_2, v_3\}$, we define the

ordering $\langle u_3, v_1, v_2, v_3, u_6, v_4\rangle$ for Sym-BK branching as shown in Figure 2(c). The branches B_5', B_6' and B_7' can be pruned since B_5' has the partial set as $\{u_0, u_1, u_2, u_3, v_0,$
$v_1, v_2, v_3\}$ not a k-biplex (u_3 has more than $k = 2$ disconnections).

We note that there could be multiple vertices, which have more than k disconnections among $S \cup R$, and for each of them, we can define an ordering as above. We call these vertices *candidate pivots* and the vertex that we pick for defining an ordering the *pivot*. An immediate question is: *which one should we select as the pivot among the candidate pivots?* To answer this question, we quantify the benefits of specifying the ordering based on a specific candidate pivot $\hat{v}$. There are two benefits (for simplicity, we discuss the case of $\hat{v} \in S_L$ only, and the other case is similar and thus omitted). Benefit 1: $(|C| - a)$ branches, namely B_{a+2}, B_{a+3}, ..., $B_{|C|+1}$, are pruned for $\hat{v}$. Therefore, the smaller a is, the larger the Benefit 1 is. Benefit 2: For Branch B_{a+1}, we have $|C_{a+1}| \leq |C| - b$ since (1) C_{a+1} is updated to be $C - \{u_1, u_2, ..., u_{a+1}\}$ (please refer to Equation (5)) and (2) the vertices $u_{a+2}, u_{a+3}, ..., u_b$ can be further excluded from C_{a+1} since including each of these vertices to S_{a+1} would violate the k-biplex definition. Therefore, the larger b is, the larger the Benefit 2 is. In summary, for a vertex with a *smaller* a and/or a *larger* b, the overall benefits would be more significant. Therefore, we select the candidate pivot $\hat{v}$ with the largest $(b - a) = \overline{\delta}(\hat{v}, S_R \cup C_R) - k$ as the pivot. Equivalently, it would select the candidate pivot with the most disconnections within $S \cup C$. Furthermore, to achieve a better worst-case time complexity (details will be introduced in Section 3.4), we first select the pivot among the pivot candidates in S if possible; otherwise, we select one among those in C.

To illustrate, we consider again the example in Figure 2. For a branch B in Figure 2(b), v_0 would be selected as the pivot since v_0 in S has the number of disconnections more than k and the greatest among other vertices in S. For another branch B_3 in Figure 2(c), u_3 would be selected as the pivot since (1) every vertex in S disconnects less than k vertices and (2) u_3 has the number of disconnections more than k and the greatest among other vertices in C.

3.4 A Sym-BK Branching based Branch-and-Bound Algorithm: FastBB

Based on the Sym-BK branching strategy and the aforementioned pruning techniques (details will be presented in this section), we design a branch-and-bound algorithm, called FastBB. The pseudo-code of FastBB is presented in Algorithm 2. FastBB differs from BasicBB only in the branching step (i.e., Lines 7 - 10 of Algorithm 2). Next, we elaborate on the pruning conditions (Line 6 in Algorithm 2) in detail.

Pruning conditions. Let $B = (S, C, D)$ be a branch. We first define $\tau_L = k + \min_{u \in S_R} \delta(u, S_L \cup C_L)$ and $\tau_R = k + \min_{v \in S_L} \delta(v, S_R \cup C_R)$, which can be verified to be the upper bound of the number of vertices at the left side and that at the right side of a MBP covered by the branch B, respectively. We can prune the branch B if any of the following four conditions is satisfied.

(1) S is not a k-biplex.
(2) $\tau_L < \theta_L$ or $\tau_R < \theta_R$.
(3) $|E(G[S \cup C])| \leq |E(H^*)|$ or $\tau_L \times \tau_R \leq |E(H^*)|$, where H^* is the MaxBP found so far.
(4) There exists a vertex $v \in D_L$ such that $\overline{\delta}(v, S_R \cup C_R) \leq k$ and $\{w \in S_R \cup C_R \mid \overline{\delta}(w, S_L \cup C_L) \geq k\} \subseteq \Gamma(v, R)$ or symmetrically there exists such a vertex $u \in D_R$.

Condition (1) holds because of the hereditary property of k-biplex, Condition (2) is based on the size constraints of the two sides of MaxBP to be found, Condition (3) is based on the objective of the problem (i.e., to maximize the number of edges in a MBP), and Condition (4) holds because all k-biplexes covered by this branch (if any) would not be maximal (since an additional vertex v or u can be included in each of them without violating the k-biplex definition).

Algorithm 2: The branch-and-bound algorithm based on Sym-BK branching: FastBB

```
Input: A graph $G(L \cup R, E)$, $k$, $\theta_L$ and $\theta_R$
Output: The maximal $k$-biplex $H^*$ with the most edges
1  $H^* \leftarrow G[\emptyset]$; // Global variable
2  FastBB-Rec($\emptyset, L \cup R, \emptyset$); return $H^*$;
3  Procedure FastBB-Rec($S, C, D$)
       /* Termination */
4      if $G[S \cup C]$ is a $k$-biplex then
5          $H^* \leftarrow G[S \cup C]$ if $|E(G[S \cup C])| > |E(H^*)|$ and return
       /* Pruning */
6      if any of pruning conditions is satisfied then return ;
       /* Sym-BK Branching */
7      Select a pivot vertex $\hat{v}$ and determine an ordering based on $\hat{v}$ (Section 3.3);
8      if $\hat{v} \in S$ then Create $a + 1$ branches $\{B_1, B_2, ..., B_{a+1}\}$ (Equation (3) and (5)) ;
9      else if $\hat{v} \in C$ then
10         Create $a + 2$ branches $\{B_1, B_2, ..., B_{a+2}\}$ (Equation (3) and (6));
11     for each branch $B_i \in \{B_1, B_2, ..., B_i, ...\}$ do
12         FastBB-Rec($S_i, C_i, D_i$);
```

Worst-case time complexity. The worst-case time complexity of FastBB is strictly better than than of BasicBB, which we show in the following theorem.

Theorem 1. *Given a bipartite graph G,* FastBB *finds the MaxBP in time $O(|V| \cdot d \cdot \gamma_k^{|V|})$ where γ_k is the largest positive real root of $x^{k+4} - 2x^{k+3} + x^2 - x + 1 = 0$. For example, when $k = 1$, 2 and 3, $\gamma_k = 1.754$, 1.888 and 1.947, respectively.*

Proof. We give a sketch of the proof and put the details in the technical report [3]. We recursively maintain two arrays to record the degree of each vertex v within $G[S]$ or $G[S \cup C]$, i.e., $\delta(v, S)$ or $\delta(v, S \cup C)$. Then, the recursion of FastBB-Rec runs in polynomial time $O(|V| \cdot d)$. Specifically, the time cost is dominated by the part of checking pruning condition (4) in line 6. This part has two steps, namely finding all those vertices with at least k disconnections from $S \cup C$ in $O(|S \cup C|)$ time and checking the pruning condition (4) for each vertex in D in $O(|D| \cdot d)$ time, where $|S \cup C|$ and $|D|$ are both bounded by $O(|V|)$.

Next, we analyze the number of recursions. Let $T(n)$ be the largest number of recursions where $n = |C|$. We have two cases.

Case 1 ($\hat{v} \in S$). We remove i vertices from C in B_i ($1 \le i \le a$) and b vertices from C in B_{a+1} in the worst-case. Hence, we have

$$T_1(n) \le \sum_{i=1}^{a} T_1(n-i) + T_1(n-b). \tag{7}$$

As discussed earlier, we have $a \le k$ and $a < b$. It is easy to verify that we reach the maximum of $T_1(n)$ when $a = k$ and $b = k+1$. We thus have $T_1(n) \le \sum_{i=1}^{k} T_1(n-i) + T_1(n-k-1)$. By solving this linear recurrence, the worst-case running time is $O(|v| \cdot d \cdot \gamma_k^{|V|})$ where γ_k is the largest positive real root of $x^{k+2} - 2x^{k+1} + 1 = 0$. For example, $\gamma_k = 1.618, 1.839$ and 1.928 when $k = 1, 2$ and 3, respectively.

Case 2 ($\hat{v} \in C$). We remove i vertices from C in B_i ($1 \le i \le a+1$) and $b+1$ vertices from C in B_{a+2} in the worst-case. Hence, we have

$$T_2(n) \le \sum_{i=1}^{a+1} T_2(n-i) + T_2(n-b-1). \tag{8}$$

Assume $\hat{v} \in C_L$, we consider two scenarios, i.e., $\overline{\delta}(\hat{v}, S_R \cup C_R) \ge k+2$ and $\overline{\delta}(\hat{v}, S_R \cup C_R) = k+1$.

- For Scenario 1, we can imply $a \le k$ and $b > a+1$. When $a = k$ and $b = k+2$, $T_2(n)$ reaches the maximum. Hence, the worst-case running time is $O(|V| \cdot d \cdot \gamma_k^{|V|})$ where γ_k is the largest positive real root of $x^{k+4} - 2x^{k+3} + x^2 - x + 1 = 0$. For example, when $k = 1$, 2 and 3, we have $\gamma_k = 1.754, 1.888$ and 1.947, respectively.
- For Scenario 2, we can imply $a \le k$ and $b > a$. When $a = k$ and $b = k+1$, $T_2(n)$ reaches the maximum. Thus the worst-case running time is $O(|V| \cdot d \cdot \gamma_k^{|V|})$ where γ_k is the largest positive real root of $x^{k+3} - 2x^{k+2} + 1 = 0$. For example, when $k = 1$, 2 and 3, we have $\gamma_k = 1.839, 1.928$ and 1.966, respectively.

The idea of remaining proof is to show that the analysis of Scenario 2 can be further improved based on our pivot selection strategy. Consequently, Scenario 2 would has the worst-case time complexity smaller than Scenario 1, and thus the worst-case time complexity of FastBB would be bounded by Scenario 1. □

4 EFFICIENCY AND SCALABILITY BOOSTING TECHNIQUES

In this section, we further introduce three techniques for boosting the efficiency and scalability of the branch-and-bound (BB) algorithms introduced in Section 3, namely *progressive bounding* (PB) in Section 4.1, *inclusion-exclusion* (IE) in Section 4.2, and PBIE, which combines PB and IE, in Section 4.3.

4.1 Progressive Bounding Framework: PB

PB is adapted from an existing study of finding the biclique with the maximum number of edges [23]. The major idea of PB is to run a BB algorithm multiple times, and for each time, it imposes appropriate constraints on the MBP to be found, including lower and upper bounds of the number of vertices on both the left and right sides of the MBP. Then, it returns the MBP with the most edges found at different times. With the constraints captured by the lower and upper bounds, there are two benefits, namely (1) the BB algorithm can be run on a reduced graph instead of the original one and (2) the efficiency of the BB algorithm on the reduced graph can be further boosted with additional pruning techniques based on the upper bounds. Note that only the pruning techniques based on the lower bounds θ_L and θ_R are used in the BB algorithms.

Let H^* be the MaxBP with the maximum $E(H^*)$. We have the following prior knowledge about the number of vertices at the left and right sides of H^*, i.e., $|L(H^*)|$ and $|R(H^*)|$.

$$\theta_L \le |L(H^*)| \le \delta_{max}^R + k;\ \theta_R \le |R(H^*)| \le \delta_{max}^L + k. \tag{9}$$

where $\delta_{max}^R = \max_{u \in R} \delta(u, L)$ and $\delta_{max}^L = \max_{v \in L} \delta(v, R)$. The lower bounds θ_L and θ_R are inherited from the problem definition. The upper bounds of $\delta_{max}^R + k$ and $\delta_{max}^L + k$ can be verified easily and the proofs for them are thus omitted. We denote by LB_L^i, UB_L^i, LB_R^i, and UB_R^i the lower bound and upper bound of the number of vertices at the left and right sides, respectively, which the PB would use to capture the constraints at the i^{th} time. Then, PB would set these bounds *progressively* as follows.

$$LB_L^i = \max\{LB_L^{i-1}/2, \theta_L\};\ UB_L^i = LB_L^{i-1}; \tag{10}$$

$$LB_R^i = \max\{|E(H_{i-1}^*)|/UB_L^i, \theta_R\};\ UB_R^i = \delta_{max}^L + k; \tag{11}$$

where $LB_L^0 = \delta_{max}^R + k$ and H_{i-1}^* is the MBP found at the $(i-1)^{th}$ time and H_0^* can be set as $G[\emptyset]$. Essentially, it (1) splits the range of possible values of $|L(H^*)|$, namely $[\theta_L, \delta_{max}^R + k]$, into intervals with lengths decreasing *logarithmically*, (2) uses the boundaries of the intervals as lower and upper bounds of $|L(H^*)|$, and (3) then uses the upper bound of $|L(H^*)|$ and the MBP with the most edges found so far to further tighten the lower bound of $|R(H^*)|$. Note that it would generate $O(\log(\delta_{max}^R + k))$ sets of constraints.

It would then run the BB algorithm for each set of constrains captured by $LB_L^i, UB_L^i, LB_R^i, UB_R^i$ in the order of $i = 1, 2, \ldots$. At the i^{th} time, it utilizes the lower and upper bounds as follows. First, it reduces the graph by computing the $(|LB_R^i - k|, |LB_L^i - k|)$-core of G since according to [23], any MBP with at least $|LB_L^i|$ vertices at the left side and $|LB_R^i|$ vertices at the right side must reside in the $(|LB_R^i - k|, |LB_L^i - k|)$-core of G. Second, when it runs a BB algorithm, it prunes a branch $B = (S, C, D)$ if $|S_L| > |UB_L^i|$ or $|S_R| > UB_R^i$.

In summary, PB runs a BB algorithm multiple times, each time on a reduced graph. Hence, PB would boost the practical performance of a BB algorithm.

4.2 Inclusion-Exclusion based Framework: IE

IE is adapted from the *decomposition* technique, which has been widely used for enumerating and finding subgraph structures [6, 8, 40]. The major idea of IE is to partition the graph into multiple ones (which may overlap) and run a BB algorithm on each of the subgraphs. Finally, it returns among the found MBPs the one with the most edges. Specifically, it partitions the graph G to $|L|$ subgraphs, namely $G_i = (L_i, R_i, E_i)$ for $1 \le i \le |L|$, as follows.

$$L_i = \Gamma_2(v_i, L) - \{v_1, \ldots, v_{i-1}\}; \tag{12}$$

$$R_i = \bigcup_{v \in L_i} \Gamma(v, R); \tag{13}$$

$$E_i = \{(v, u) | v \in L_i, u \in R_i, (v, u) \in E\} \tag{14}$$

where $L = \{v_1, v_2, \ldots, v_{|L|}\}$ and $\Gamma_2(v_i, L)$ denotes the set of 2-hop neighbors of v_i in L. We note that the number of vertices in G_i, i.e., $|L_i| + |R_i|$, is bounded by d^3, where d is maximum degree of the graph G. We verify that the MaxBP H^* must reside in one of the subgraphs formed as above (for which the proof could be found in the technical report [3]). Furthermore, the MBP found in G_i would *include* v_i and *excludes* $v_1, v_2, ..., v_{i-1}$.

Furthermore, it prunes the following vertices from a subgraph G_i.

- $v \in L_i$ with $\delta(v, R_i) < \theta_R - k$ or $|\Gamma(v, R_i) \cap \Gamma(v_i, R)| < \theta_R - 2k$;
- $u \in R_i$ with $\delta(u, L_i) < \theta_L - k$.

The correctness of pruning the vertices as above can be verified by contradiction based on the size constraints based on θ_L and θ_R (the detailed proof can be found in the technical report [3]).

Finally, it runs the BB algorithm on each graph G_i with some vertices pruned by starting from the branch $B_i = (S_i, C_i, D_i)$ with $S_i = \{v_i\}$, $D_i = \{v_1, v_2, ..., v_{i-1}\}$, and $C_i = V(G_i) - \{v_1, v_2, ..., v_i\}$. It then returns the MBP with the most edges among all MBPs found.

With the IE framework, the time complexity of a BB algorithm can be improved in certain cases. For example, it improves the time complexity of FastBB from $O(|V| \cdot \gamma_k^{|V|})$ to $O(|L| \cdot d^3 \cdot \gamma_k^{d^3})$ for certain sparse graphs (e.g., those with $d^3 < |V|$).

4.3 Combining PB and IE: PBIE

We observe that the PB and IE frameworks can be naturally combined for our problem of finding the MaxBP. Specifically, we can first use PB to construct multiple reduced graphs G_i with corresponding constraints of lower and upper bounds of the number of vertices at the left and right sides of a graph. Then, when for each reduced graph G_i, we further use IE to construct $|L(G_i)|$ subgraphs $G_i^{v_j}$ for $v_j \in L(G_i)$ with some vertices pruned if possible. Finally, we invoke a BB algorithm (e.g., FastBB) on each subgraph $G_i^{v_j}$ with the constraints and return the MBP with the most edges among all found MBPs. The pseudo-code of PBIE is presented in Algorithm 3.

Algorithm 3: Combine PB and IE: PBIE (for FastBB)

```
   Input: A graph $G(L, R, E)$, $k$, $\theta_L$ and $\theta_R$
   Output: The maximal $k$-biplex $H^*$ with the most edges
1  $H_0^* \leftarrow \emptyset$; $LB_L^0 \leftarrow \delta_{max}^R + k$; $i \leftarrow 1$;
2  while true do
3      Set $LB_L^i$, $UB_L^i$, $LB_R^i$, $UB_R^i$ according to Equations (10) and (11);
4      if $UB_L^i \le \theta_L$ then
5          return $H_{i-1}^*$;
6      Compute a reduced graph $G_i$ as the
         $(|LB_R^i - k|, |LB_L^i - k|)$-core of $G$;
7      $H_i^* \leftarrow H_{i-1}^*$;
8      for $v_j \in L(G_i)$ do
9          Construct a subgraph $G_i^{v_j}$ based on Equations (12 - 14)
             with some vertices further pruned;
10         $H_i^* \leftarrow$ invoke a FastBB algorithm with $G_i^{v_j}$, $k$, the
             lower/upper bounds, and $H_i^*$;
11     $i \leftarrow i + 1$;
12 return $H_{i-1}^*$;
```

Time complexity. The time cost is dominated by part of invoking FastBB (line 3-11). There are at most $O(\log(\delta_{max}^R + k))$ iterations (line 3-11). For each iteration, it constructs at most $O(|L|)$ subgraphs. Hence, FastBB is invoked by at most $O(\log(\delta_{max}^R + k) \cdot |L|)$ times.

Table 1: Real datasets

Dataset	Category	$\|L\|$	$\|R\|$	$\|E\|$
Divorce	Feature	9	50	225
Cities	Feature	46	55	1342
Cfat	Biology	200	200	1537
Opsahl	Authorship	2,865	4,558	16,910
Writer	Authorship	89,356	46,213	144,340
YouTube	Affiliation	94,238	30,087	293,360
Location	Feature	172,091	53,407	293,697
Actors	Affiliation	127,823	383,640	1,470,404
IMDB	Affiliation	303,617	896,302	3,782,463
DBLP	Authorship	1,425,813	4,000,150	8,649,016
Amazon	Rating	6,703,391	957,764	12,980,837
Google	Hyperlink	17,091,929	3,108,141	14,693,125

Besides, the number of vertices in $G_{i+1}^{v_j}$ is bounded by $O(d^3)$. Therefore, the time complexity of PBIE (when used for boosting FastBB) is $O(\log(\delta_{max}^R + k) \cdot |L| \cdot d^4 \cdot \gamma_k^{d^3})$ where γ_k is a real number strictly smaller than 2 (refer to Theorem 1 for details). We remark that the large graphs in real applications are usually sparse and have d far smaller than the total number of vertices.

5 EXPERIMENTS

Datasets. We use both real and synthetic datasets in our experiments. The real datasets are summarized in Table 1 (http://konect.cc/). The Erdös-Réyni (ER) synthetic datasets are generated by first creating a certain number of vertices and then randomly adding a certain number of edges between pairs of vertices. We define the edge density of a bipartite graph $G(L \cup R, E)$ as $|E|/(|L| + |R|)$. We set the number of vertices and edge density as 100k and 10 for synthetic datasets, respectively, by default.

Algorithms. We compare our algorithm PBIE+FastBB with three baselines, namely iMBadp [44], FPadp [51] and iTradp [45]. Specifically, PBIE+FastBB adopts the combined framework PBIE and employs the improved algorithm FastBB within the framework. iMBadp and iTradp correspond to the adaptions of existing algorithms designed for enumerating MBPs, namely iMB [44] and iTraversal [45]. Specifically, iMBadp adopts the BK branching for each side of a bipartite graph and is equipped with the pruning techniques developed in this paper. iTradp follows a *reverse search* [7] method. FPadp corresponds to a branch-and-bound algorithm with the hybrid branching strategy [51] and the pruning techniques developed in this paper.

Settings. All algorithms were written in C++ and run on a machine with a 2.66GHz CPU and 32GB main memory running CentOS. We set the time limit (INF) as 24 hours and use 4 representative datasets as default ones, i.e., Writer, Location, DBLP and Google, which cover various graph scales. We set both θ_L and θ_R as $2k + 1$ and both K and k as 1 by default. Our code, data and additional experimental results are available at https://anonymous.4open.science/r/PVLDB22-MaxBP-D58E.

5.1 Comparison among algorithms

All datasets. We compare all algorithms on various datasets and show the running time in Figure 3. We have the following observations. First, PBIE+FastBB outperforms all other algorithms on

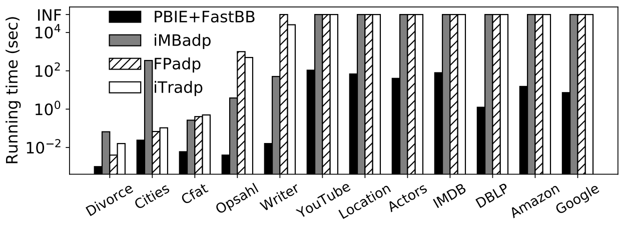
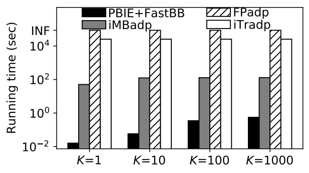
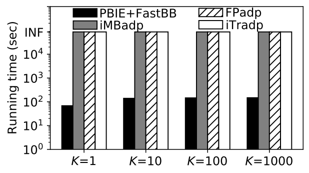
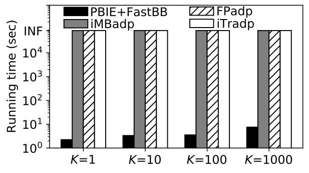


Figure 3: Comparison on all real datasets ($k = 1$)


(a) Varying K (Writer)


(b) Varying K (Location)


(c) Varying K (DBLP)

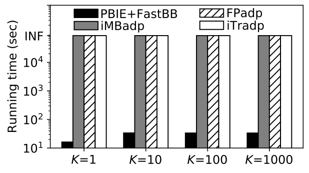
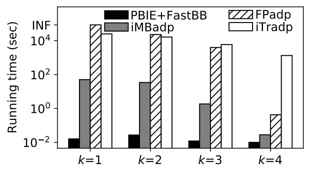
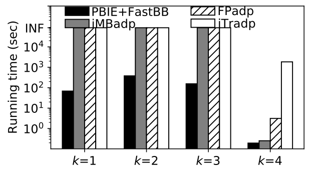
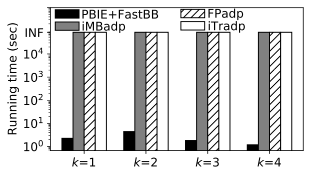

(d) Varying K (Google)

Figure 4: Comparison by varying K ($k = 1$)


(a) Varying k (Writer)


(b) Varying k (Location)


(c) Varying k (DBLP)

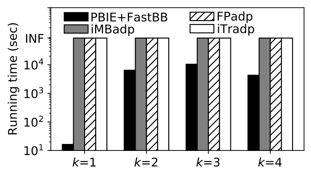
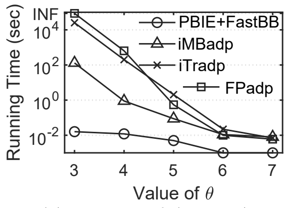
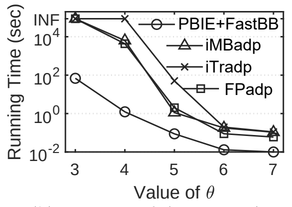
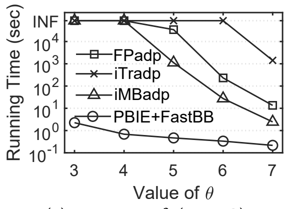

(d) Varying k (Google)

Figure 5: Comparison by varying k


(a) Varying θ (Writer)


(b) Varying θ (Location)


(c) Varying θ (DBLP)

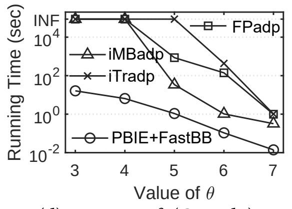

(d) Varying θ (Google)

Figure 6: Comparison by varying $\theta = \theta_L = \theta_R$ ($k = 1$)

all datasets. This is consistent with our theoretical analysis that the worst-case running time of PBIE+FastBB is smaller than that of other algorithms. Second, PBIE+FastBB can handle all datasets within INF while others cannot finish on large datasets, e.g., Amazon and Google, which demonstrates its scalability. This is mainly because the framework PBIE would quickly locate the MaxBP at several smaller subgraphs while dramatically pruning many unfruitful vertices so as to reduce the search space.

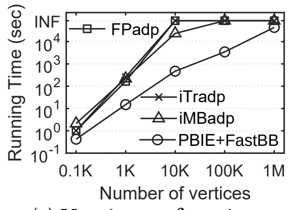
Varying K. The results of finding K MaxBPs are shown in Figure 4. PBIE+FastBB outperforms other algorithms by around 2-5 orders of magnitude. Besides, it has the running time clearly rise as K increases compared to other algorithms. Possible reasons include (1) FastBB would explore more MBPs to find the K MaxBPs as K grows and (2) PBIE becomes less effective as K grows. In addition, iTradp has the running time almost not changed with K since it needs to explore almost all MBPs without any powerful pruning.

Varying k. The results are shown in Figure 5. PBIE+FastBB significantly outperforms other algorithms by up to five orders of magnitude. In addition, it has the running time first increase and then decrease as k grows. Possible reasons include: (1) the number of k-biplexes increases exponentially with k, which causes the increase of the running time; (2) the thresholds θ_L and θ_R (i.e., $2k + 1$ by default) increase with k and correspondingly the pruning techniques that are based on θ_L's and θ_R become more effective.

Varying θ_L and θ_R thresholds. The results are shown in Figure 6. PBIE+FastBB outperforms all other algorithms by achieving up to around 1000× speedup. Besides, the running time of all algorithms decreases as θ_L and θ_R grow. The reason is two-fold: (1) the search space (e.g., the number of large k-biplexes with the size of each side at least θ_L and θ_R) decreases as θ_L and θ_R grow and (2) the pruning rules are more effective for larger θ_L's and θ_R's.

Varying # of vertices (synthetic datasets). The results are shown in Figure 7(a). PBIE+FastBB outperforms other algorithms by achieving at least 10× speedup and can handle the largest datasets with 1 million vertices and 10 million edges within INF while others cannot. Besides, the speedup increases as the graph become larger. This is mainly because PBIE would prune more unfruitful vertices when locating the MaxBP in several smaller subgraphs. In addition, the results are well aligned with the theoretical results, i.e., the worst-case time complexity of PBIE+FastBB is exponential wrt d^3 while that of others is exponential wrt $|V|$. Hence, PBIE+FastBB has larger speed-ups as the graph scale becomes larger (where d remains almost the same due to the fixed edge density, i.e., 10).

Varying edge density (synthetic datasets). The results are shown in Figure 7(b). PBIE+FastBB achieves at least 10× speedup compared with other algorithms. In addition, the speedup decreases as the graph becomes denser. The reason is that the maximum degree of the bipartite graph, i.e., d, increases as the graph becomes denser.


(a) Varying # of vertices

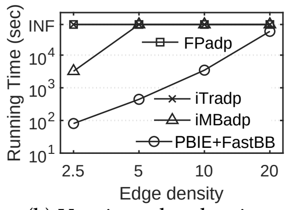
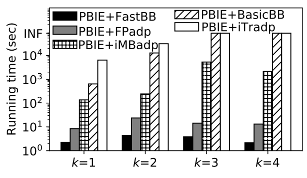
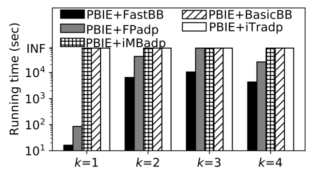
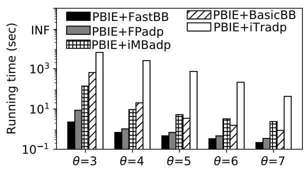

(b) Varying edge density

Figure 7: Comparison on synthetic datasets (k = 1)


(a) Varying k (DBLP)


(b) Varying k (Google)


(c) Varying θ (DBLP, k=1)

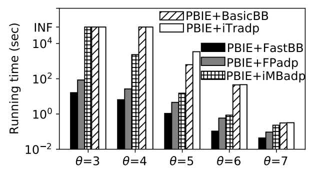
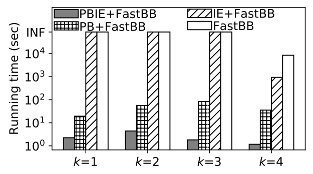
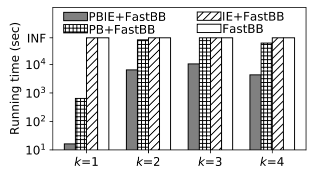
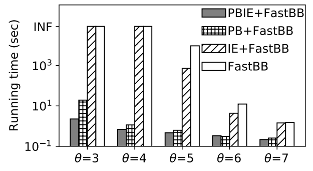

(d) Varying θ (Google, k=1)

Figure 8: Comparison among various enumeration schemes


(a) Varying k (DBLP)


(b) Varying k (Google)


(c) Varying θ (DBLP, k=1)

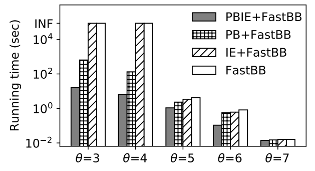

(d) Varying θ (Google, k=1)

Figure 9: Comparison among frameworks

5.2 Performance study

Comparison among various enumeration schemes. We study the effect of various enumeration schemes, namely FastBB, BasicBB, FPadp, iMBadp, and iTradp. We note that all of them are run with the framework PBIE for fair comparison. In particular, BasicBB adopts the classic BK branching and uses the non-decreasing vertex ordering. The results are shown in Figure 8(a) and (b) for varying k, and (c) and (d) for varying $\theta = \theta_L = \theta_R$. First, FastBB outperforms other algorithms, which demonstrates the superiority of proposed Sym-BK branching scheme. Besides, the achieved speedup decreases with θ since the search space (e.g., the number of large MBPs with the size of each side at least θ_L and θ_R) decreases as θ_L and θ_R grow. Second, the algorithms following the BK branching perform better than iTradp that follows a reverse search method. The reason is that the later one cannot be enhanced with effective pruning rules as branch-and-bound algorithms.

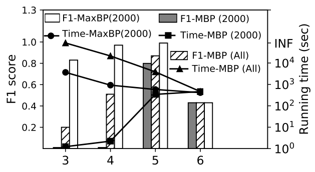
Comparison among frameworks. We study the effect of different frameworks and compare four different versions, namely (1) FastBB: without any framework, (2) IE+FastBB: with adapted framework IE, (3) PB+FastBB: with adapted progressive bounding framework PB, and (4) PBIE+FastBB: with proposed combined framework. We note that all frameworks adopt FastBB for fair comparison. The results are shown in Figure 9(a) and (b) for varying k, and (c) and (d) for varying $\theta = \theta_L = \theta_R$. First, both IE+FastBB and PB+FastBB outperform FastBB, which demonstrates the efficiency and scalability of adapted frameworks. Moreover, PBIE+FastBB performs the best and can handle all datasets and settings within INF. Second, PB+FastBB performs better than IE+FastBB. This is because PB can quickly locate the MaxBP at a much smaller subgraph. Third, the speedup decreases as θ grows. As discussed earlier, the search space gets smaller with θ and thus the frameworks would have less effects on the running time.

5.3 Case study: Fraud Detection

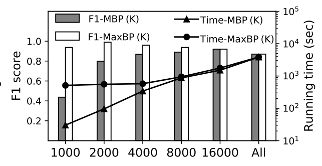
We investigate two cohesive models, namely maximal k-biplex and maximum k-biplex, for a fraud detection task [16] on the Software dataset [27]. The dataset contains 459,436 reviews on 21,663 softwares by 375,147 users. We consider the random camouflage attack scenario [16] where a fraud block with 1K fake users ("K" means a thousand), 1K fake softwares, 50K fake comments, and 50K camouflage comments, is injected to the dataset. Specifically, we randomly generate the fake comments (resp. camouflage comments) between pairs of fake users and fake products (resp. real products). We note that this attack can be easily conducted in reality to help fake users evade the detection, e.g., fake users are coordinated to deliberately post comments on some real products [16]. We then find MaxBPs and MBPs from the bipartite graph, and classify all users and products involved in the found subgraphs as fake items and others as real ones. We notice that the biclique model performs worse than k-biplex on this case study. We explore various settings and find that the achieved best F1-score of maximal biclique (at $\theta_R = 4$) and maximum biclique (at $\theta_R = 4$) are 0.48 and 0.54, respectively. In particular, when $\theta_R \geq 5$, there are few large bicliques and thus either maximal biclique or maximum biclique has the recall close to 0. This is because the complete connection requirement of biclique is too strict, which is consistent with the observation in [45].

We measure the running time and F1 score, and show the results in Figure 10 where θ_L is fixed at 4.

- **Varying θ_R.** We find the 2000 MaxBPs (denoted by "MaxBP (2000)"), first-2000 MBPs (denoted by "MBP (2000)"), i.e., the first 2000 MBPs yielded by the algorithm, and all MBPs (denoted by "MBP (All)") with θ_R varying from 3 to 6, and show the results in Figure 10(a). We have the following observations. (1) MaxBP (2000) achieves the best F1 score (0.99) when $\theta_R = 5$ among all methods. (2) Both MBP (2000) and MBP (All) achieve their best F1 scores, 0.80 and 0.87, respectively, when $\theta_R = 5$. (3) Under the setting with the best F1 scores, i.e., $\theta_R = 5$, MaxBP (2000) and MBP (2000) run comparably fast and both of them run faster than MBP (All). For (1) and (2), the reason could be that the fraud blocks tend to reside in large MBPs with more edges, and for (3),



(a) Varying θ_R



(b) Varying K ($\theta_R = 5$)

Figure 10: Case Study: Fraud Detection

the reason could be that MaxBP (2000) and MBP (2000) need to explore a set of *some* but not all MBPs.

- **Varying** K. We find the K MaxBPs (denoted by "MaxBP (K)") and the first-K MBPs (denoted by "MBP (K)") with $\theta_R = 5$ since it give the best F1 scores, and show the results in Figure 10(b). We have the following observations. (1) MaxBP (K) has the F1 score higher than MBP (K) on all settings, and achieves the best F1 score at $K = 2000$. (2) MaxBP (K) provides a better trade-off between F1 score and running time than MBP (K), e.g., MaxBP (K) provides a F1 score 0.94 with the running time 512 seconds at $K = 1000$ while MBP (K) provides a similar F1 score 0.91 with the running time 1512 seconds at $K = 16000$.

In summary, MaxBP outperforms MBP for fraud detection in terms of F1 score and the running time, as shown in Figure 10.

6 RELATED WORK

This is the first work on the maximum k-biplex search. Below we review some related work, including other cohesive bipartite structures and maximum cohesive bipartite subgraph search.

Cohesive bipartite structures. Recently, many studies have been conducted on finding cohesive subgraphs of bipartite graphs, including bicliques [1, 6, 19, 23, 26, 42, 48], (α, β)-cores [20], quasi-bicliques [17, 18, 22, 37], k-biplexes [30, 44, 45], k-bitrusses [35, 52], etc. Biclique is a bipartite graph where any vertex at one side connects all vertices at the other side. Recent works on biclique focus on enumerating maximal bicliques [1, 19, 48]. Given a bipartite graph, (α, β)-core is the maximal subgraph where any vertex at one side connects a certain number of vertices (i.e., α or β) at the other side. It has many applications, including recommendation systems [10] and community search [15, 36]. Existing works of k-biplexes focus on enumerating large maximal k-biplex [30, 44] and none of them study the maximum k-biplex search problem as this paper does. A k-bitruss [35, 52] is a bipartite graph where each edge is contained in at least k butterflies, where a butterfly corresponds to a complete 2×2 biclique [34]. In the literature, there are two types of quasi-biclique, i.e., (1) δ-quasi-biclique [22] is a bipartite graph $G(L, R, E)$ 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 [18, 41] is a bipartite graph $G(L, R, E)$ that can miss at most $\gamma \cdot |L| \cdot |R|$ edges with $\gamma \in [0, 1)$. Existing works of quasi-bicliques focus on finding subgraphs with a certain density and degree [21, 25]. In this paper, we focus on k-biplex since it (1) imposes strict enough requirements on connections within a subgraph and tolerates some disconnections and (2) satisfies the hereditary property, which facilitates efficient solutions. In [45], a case study of fraud detection on e-commerce platforms is conducted, which shows that k-biplex works better than some other cohesive subgraph structures including biclique, (α, β)-core, and δ-quasi-biclique for the application.

Maximum biclique search. The maximum biclique search problem has attracted much attention in recent years [6, 9, 13, 23, 24, 29, 31, 38, 46, 49, 50]. In general, there are three lines of works, namely maximum edge biclique search (MEBS) [9, 23, 29, 31] which finds a biclique H^* such that $E(H^*)$ is maximized, maximum vertex biclique search (MVBS) [13] which finds a biclique H^* such that $V(H^*)$ is maximized and maximum balanced biclique search (MBBS) [6, 24, 38, 46, 49, 50] which finds a biclique H^* such that $V(H^*)$ is maximized and $L(H^*) = R(H^*)$. First, the MEBS problem is NP-hard, for which many techniques have been proposed. Authors in [9, 31] adopt the integer linear programming techniques to find a MBE, which is not scalable for large bipartite graphs. A recent study [23] proposes a progressive bounding framework to deal with large bipartite graphs. Besides, a Monte Carlo algorithm is proposed in [29], which finds a MEB with a fixed probability. Second, the MVBS problem can be solved in polynomial time by finding a maximum matching [13]. Third, the MBBS problem is NP-hard, for which both exact methods and approximate methods have been developed. To be specific, exact methods proposed in [6, 24, 50] are branch-and-bound algorithms which use the widely-used Bron-Kerbosch (BK) branching [5]. Besides, approximate methods include [38, 46] which introduce a local search framework to find an approximate MBB, [2, 32] which adopt approximate algorithms for independent set problems by converting MBBS into a maximum independent set problem, and [49] which proposes a heuristic algorithm with tabu search and graph reduction. In summary, there are two types of solutions, namely exact methods and approximate methods. For exact algorithms, most of them follow the branch-and-bound framework and use the BK branching strategy. However, based on our experimental results, BK branching strategy performs worse than our proposed Sym-BK branching strategy for the problem of finding MaxBP. For approximate methods, they cannot be adapted to find the MaxBP exactly.

Maximum quasi-biclique search. The maximum quasi-biclique search problem aims to find a γ-quasi-biclique or δ-quasi-biclique H^* that $V(H^*)$ is maximized, which is a NP-hard problem [17, 22]. Authors in [17, 18] use mixed integer programming to find a maximum γ-quasi-biclique exactly, which cannot handle large datasets. Authors in [22, 37] propose greedy algorithms to find an approximate maximum δ-quasi-biclique, which cannot be adapted to find the MaxBP exactly.

7 CONCLUSION

In this paper, we study the *maximum k-biplex search* problem, which is find K maximal k-biplexes with the most edges. We propose two branch-and-bound algorithms, among which the better one `FastBB` is based on a novel Sym-BK branching strategy and achieves better worst-case time complexity than adaptions of existing algorithms. We further develop frameworks to boost the efficiency and scalability of the branch-and-bound algorithms including `FastBB`. Extensive experiments are conducted on real and synthetic datasets to demonstrate the efficiency of our algorithms and the effectiveness of proposed techniques. In the future, we plan to develop efficient parallel algorithms for the maximum k-biplex search problem and explore the possibility of adapting our algorithms to find other types of maximum cohesive subgraphs in bipartite graphs.

REFERENCES

[1] 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.

[2] Ahmad A. Al-Yamani, S. Ramsundar, and Dhiraj K. Pradhan. 2007. A Defect Tolerance Scheme for Nanotechnology Circuits. *IEEE Trans. Circuits Syst. I Regul. Pap.* 54-I, 11 (2007), 2402–2409.

[3] Anonymous Authors. 2022. Efficient Algorithms for Maximum k-biplex Search on Bipartite Graphs (Technical report). https://anonymous.4open.science/r/SIGMOD23-MaxBP-C5D0/SIGMOD_MaxBP_report.pdf.

[4] Alex Beutel, Wanhong Xu, Venkatesan Guruswami, Christopher Palow, and Christos Faloutsos. 2013. Copycatch: stopping group attacks by spotting lockstep behavior in social networks. In *Proceedings of the 22nd international conference on World Wide Web*. 119–130.

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

[6] 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.

[7] 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.

[8] 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.

[9] Milind Dawande, Pinar Keskinocak, Jayashankar M Swaminathan, and Sridhar Tayur. 2001. On bipartite and multipartite clique problems. *Journal of Algorithms* 41, 2 (2001), 388–403.

[10] 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.

[11] Fedor V Fomin and Petteri Kaski. 2013. Exact exponential algorithms. *Commun. ACM* 56, 3 (2013), 80–88.

[12] 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.

[13] Michael R Garey and David S Johnson. 1979. *Computers and intractability*. Vol. 174. freeman San Francisco.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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

[21] 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.

[22] 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.

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

[24] Ciaran McCreesh and Patrick Prosser. 2014. An Exact Branch and Bound Algorithm with Symmetry Breaking for the Maximum Balanced Induced Biclique Problem. In *Integration of AI and OR Techniques in Constraint Programming International Conference (Lecture Notes in Computer Science, Vol. 8451)*, Helmut Simonis (Ed.). Springer, 226–234.

[25] 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.

[26] 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.

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

[28] 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.

[29] Eran Shaham, Honghai Yu, and Xiao-Li Li. 2016. On finding the maximum edge biclique in a bipartite graph: a subspace clustering approach. In *Proceedings of the 2016 SIAM International Conference on Data Mining*. SIAM, 315–323.

[30] 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.

[31] Melih Sözdinler and Can Özturan. 2018. Finding Maximum Edge Biclique in Bipartite Networks by Integer Programming. In *2018 IEEE International Conference on Computational Science and Engineering (CSE)*. IEEE, 132–137.

[32] Mehdi Baradaran Tahoori. 2006. Application-independent defect tolerance of reconfigurable nanoarchitectures. *ACM J. Emerg. Technol. Comput. Syst.* 2, 3 (2006), 197–218.

[33] 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.

[34] 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.

[35] 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.

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

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

[38] Yiyuan Wang, Shaowei Cai, and Minghao Yin. 2018. New heuristic approaches for maximum balanced biclique problem. *Information Sciences* 432 (2018), 362–375.

[39] Zhuo Wang, Qun Chen, Boyi Hou, Bo Suo, Zhanhuai Li, Wei Pan, and Zachary G Ives. 2017. Parallelizing maximal clique and k-plex enumeration over graph data. *J. Parallel and Distrib. Comput.* 106 (2017), 79–91.

[40] Zhengren Wang, Yi Zhou, Mingyun Xiao, and Bakhadyr Khoussainov. 2022. Listing Maximal k-Plexes in Large Real-World Graphs. *arXiv preprint arXiv:2202.08737* (2022).

[41] Changhui Yan, J Gordon Burleigh, and Oliver Eulenstein. 2005. Identifying optimal incomplete phylogenetic data sets from sequence databases. *Molecular phylogenetics and evolution* 35, 3 (2005), 528–535.

[42] Jianye Yang, Yun Peng, and Wenjie Zhang. 2022. (p,q)-biclique counting and enumeration for large sparse bipartite graphs. *Proceedings of the VLDB Endowment* (2022).

[43] Kaiqiang Yu and Cheng Long. 2021. Graph Mining Meets Fake News Detection. In *Data Science for Fake News*. Springer, 169–189.

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

[45] Kaiqiang Yu, Cheng Long, Shengxin Liu, and Da Yan. 2022. Efficient Algorithms for Maximal k-Biplex Enumeration. In *Proc. ACM SIGMOD Int. Conf. Manage. Data*.

[46] Bo Yuan, Bin Li, Huanhuan Chen, and Xin Yao. 2015. A New Evolutionary Algorithm with Structure Mutation for the Maximum Balanced Biclique Problem. *IEEE Trans. Cybern.* 45, 5 (2015), 1040–1053.

[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.

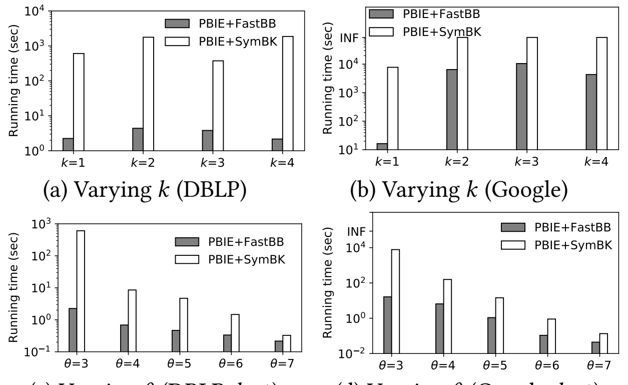
[49] Yi Zhou and Jin-Kao Hao. 2017. Combining tabu search and graph reduction to solve the maximum balanced biclique problem. *CoRR* abs/1705.07339 (2017).

[50] Yi Zhou, André Rossi, and Jin-Kao Hao. 2018. Towards effective exact methods for the Maximum Balanced Biclique Problem in bipartite graphs. *European Journal of Operational Research* 269, 3 (2018), 834–843.

[51] 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*. 2442–2449.

[52] 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) Varying k (DBLP) (b) Varying k (Google)

(c) Varying θ (DBLP, k=1) (d) Varying θ (Google, k=1)

Figure 11: Comparison among various orderings

A ADDITIONAL EXPERIMENTAL RESULTS

Comparison among various orderings of vertices for Sym-BK branching. We consider the Sym-BK branching and study the effect of various orderings of the vertices in C.

B WORST-CASE TIME COMPLEXITY ANALYSIS

The worst-case time complexity of `FastBB` is strictly better than than of `BasicBB` based on Theorem 1 (in Section 3.4). Below we provide a detailed analysis.

Theorem 1. *Given a bipartite graph G,* `FastBB` *finds the MaxBP in time $O(|V| \cdot d \cdot \gamma_k^{|V|})$ where γ_k is the largest positive real root of $x^{k+4} - 2x^{k+3} + x^2 - x + 1 = 0$. For example, when $k = 1$, 2 and 3, $\gamma_k = 1.754$, 1.888 and 1.947, respectively.*

Proof. We recursively maintain two arrays to record the degree of each vertex v within $G[S]$ or $G[S \cup C]$, i.e., $\delta(v, S)$ or $\delta(v, S \cup C)$. Then, the recursion of `FastBB-Rec` runs in polynomial time $O(|V| \cdot d)$. Specifically, the time cost is dominated by the part of checking pruning condition (4) in line 6 of Algorithm 2. This part has two steps, namely finding all those vertices with at least k disconnections from $S \cup C$ in $O(|S \cup C|)$ time and checking the pruning condition (4) for each vertex in D in $O(|D| \cdot d)$ time, where $|S \cup C|$ and $|D|$ are both bounded by $O(|V|)$.

Next, we analyze the number of recursions. Let $T(n)$ be the largest number of recursions where $n = |C|$. We have two cases.
Case 1 ($\hat{v} \in S$). We remove i vertices from C in B_i ($1 \le i \le a$) and b vertices from C in B_{a+1} in the worst-case. Hence, we have

$$T_1(n) \le \sum_{i=1}^{a} T_1(n-i) + T_1(n-b). \tag{15}$$

As discussed earlier, we have $a \le k$ and $a < b$. It is easy to verify that we reach the maximum of $T_1(n)$ when $a = k$ and $b = k + 1$. We thus have $T_1(n) \le \sum_{i=1}^{k} T_1(n-i) + T_1(n-k-1)$. By solving this linear recurrence, the worst-case running time is $O(|v| \cdot d \cdot \gamma_k^{|V|})$ where γ_k is the largest positive real root of $x^{k+2} - 2x^{k+1} + 1 = 0$. For example, $\gamma_k = 1.618$, 1.839 and 1.928 when $k = 1, 2$ and 3, respectively.

Case 2 ($\hat{v} \in C$). We remove i vertices from C in B_i ($1 \le i \le a+1$) and $b+1$ vertices from C in B_{a+2} in the worst-case. Hence, we have

$$T_2(n) \le \sum_{i=1}^{a+1} T_2(n-i) + T_2(n-b-1). \tag{16}$$

Assume $\hat{v} \in C_L$, we consider two scenarios, i.e., $\overline{\delta}(\hat{v}, S_R \cup C_R) \ge k+2$ and $\overline{\delta}(\hat{v}, S_R \cup C_R) = k + 1$.

- For Scenario 1, we can imply $a \le k$ and $b > a + 1$. When $a = k$ and $b = k+2$, $T_2(n)$ reaches the maximum. Hence, the worst-case running time is $O(|V| \cdot d \cdot \gamma_k^{|V|})$ where γ_k is the largest positive real root of $x^{k+4} - 2x^{k+3} + x^2 - x + 1 = 0$. For example, when $k = 1$, 2 and 3, we have $\gamma_k = 1.754$, 1.888 and 1.947, respectively.
- For Scenario 2, we can imply $a \le k$ and $b > a$. When $a = k$ and $b = k + 1$, $T_2(n)$ reaches the maximum. Thus the worst-case running time is $O(|V| \cdot d \cdot \gamma_k^{|V|})$ where γ_k is the largest positive real root of $x^{k+3} - 2x^{k+2} + 1 = 0$. For example, when $k = 1$, 2 and 3, we have $\gamma_k = 1.839$, 1.928 and 1.966, respectively.

The idea of remaining proof is to show that the analysis of Scenario 2 can be further improved based on our pivot selection strategy. Consequently, Scenario 2 would has the worst-case time complexity smaller than Scenario 1, and thus the worst-case time complexity of `FastBB` would be bounded by Scenario 1.

We first show two pruning rules designed based on our pivot selection strategy, which would help to improve Scenario 2. Intuitively, these pruning rules are built upon branch B_1 (formed based on Equation (6)) and try to remove from C_1 to S_1 more vertices so as to reduce the number of potential branches to be created when branching at B_1. The rationale behind is that (1) after excluding pivot $\hat{v}$ at B_1, vertices in C that disconnect to $\hat{v}$, i.e., $\overline{\Gamma}(\hat{v}, C_R)$, would disconnect one vertex fewer at the other side of $S \cup C - \{\hat{v}\}$ and (2) these vertices are thus more likely to be appeared in the MaxBP to be found in B_1. To be specific, we have following two pruning rules (wlog, assume $\hat{v} \in C_L$).
Pruning 1. Remove from C_1 to S_1 all vertices in $\overline{\Gamma}(\hat{v}, C_R)$, if the following two conditions are satisfied.

(i) Pivot $\hat{v}$ disconnects exactly $k + 1$ vertices at $S_R \cup C_R$ (i.e., $\overline{\delta}(\hat{v}, S_R \cup C_R) = k + 1$), and
(ii) For every vertex u in C_R that disconnects to $\hat{v}$, vertex u disconnects no more than k vertices from $S_L \cup C_L$ (i.e., $\forall u \in \overline{\Gamma}(\hat{v}, C_R)$, $\overline{\delta}(u, S_L \cup C_L) \le k$).

Proof of Pruning 1. This can be proved by contradiction. Assume that the MaxBP H to be found in B_1 does not include a vertex $w \in \overline{\Gamma}(\hat{v}, C_R)$. We derive the contradiction by showing that $G[V(H) \cup \{\hat{v}\}]$ is a k-biplex which has more edges than H (note that the MaxBP found in this branch cannot include $\hat{v}$ since $\hat{v}$ has been excluded). Specifically, (1) for a vertex $u \in L(H)$, we have $\overline{\delta}(u, R(H)) \le k$ since H is a k-biplex; (2) For $\hat{v}$, we have $\overline{\delta}(\hat{v}, R(H)) \le \overline{\delta}(\hat{v}, S_R \cup C_R \backslash \{w\}) \le \overline{\delta}(\hat{v}, S_R \cup C_R) - 1 \le (k+1) - 1 = k$ since $R(H) \subseteq S_R \cup C_R \backslash \{w\}$ (note that H is a subgraph of $G[C \cup S]$ and H does not include w by assumption), vertex w disconnects vertex $\hat{v}$ and $\overline{\delta}(\hat{v}, S_R \cup C_R)$ is exactly $k+1$ by pruning condition (i); (3) For a vertex $u \in S_R \cap R(H)$, we have $\overline{\delta}(u, L(H) \cup \{\hat{v}\}) \le \overline{\delta}(u, S_L \cup C_L) \le k$ since $L(H) \cup \{\hat{v}\} \subseteq S_L \cup C_L$ and every vertex in S_R disconnects no more than k vertices from $S_L \cup C_L$ based on the pivot selection; (4) For a vertex $u \in C_R \cap R(H)$, if u connects $\hat{v}$, we have

$\overline{\delta}(u, L(H) \cup \{\hat{v}\}) \le \overline{\delta}(u, L(H)) \le k$ since H is a k-biplex. If u disconnects $\hat{v}$, we have $\overline{\delta}(u, L(H) \cup \{\hat{v}\}) \le \overline{\delta}(u, S_L \cup C_L) \le k$ based on $L(H) \cup \{\hat{v}\} \subseteq S_L \cup C_L$ and pruning condition (ii).

Example 1. *To illustrate, we consider again the example in Figure 2. We Consider a branch B'_3 with $S'_3 = \{v_0, u_0, u_1, u_2\}$, $C'_3 = \{u_3, u_6, v_1, v_2, v_3, v_4\}$ and $D'_3 = \{u_4, u_5\}$ shown in Figure 2(c).We then select u_3 as the pivot based on our pivot selection strategy. Consider the branch to be created by excluding u_3 based on Equation 6, we observe that pruning 1 is triggered since (1) pivot u_3 disconnects 3 vertices from the other side of $S'_3 \cup C'_3$, i.e., $\{v_1, v_2, v_3\}$ and each vertex in $\{v_1, v_2, v_3\}$ disconnects 2 vertices from the other side of $S'_3 \cup C'_3$, i.e., $\{u_3, u_6\}$. Therefore, $\{v_1, v_2, v_3\}$ can be directly included to S'_3*

Pruning 2. Remove from $\overline{\Gamma}(\hat{v}, C_R)$ to S_1 those vertices each of which u satisfies the following two conditions.

(a) u disconnects no more than $k+1$ vertices from $S_L \cup C_L$ (i.e., $\overline{\delta}(v, S_L \cup C_L) \le k+1$), and

(b) For every vertex v in $C_L \backslash \{\hat{v}\}$ that disconnects to u, vertex v disconnects no more than k vertices from $S_R \cup C_R$ (i.e., $\forall v \in \overline{\Gamma}(u, C_L \backslash \{\hat{v}\}), \overline{\delta}(v, S_R \cup C_R) \le k$).

We define by $\mathcal{I}$ the set of those vertices such that $\mathcal{I} \subseteq \overline{\Gamma}(\hat{v}, C_R)$.

Proof of Pruning 2. We prove it by contradiction. Assume that the MaxBP H to be found in B_1 does not include a vertex $w \in I$. We derive the contradiction by showing that $G[V(H) \cup \{w\}]$ is a k-biplex. Specifically, (1) for a vertex $u \in R(H)$, we have $\overline{\delta}(u, L(H)) \le k$ since H is a k-biplex; (2) For vertex w, we have $\overline{\delta}(w, L(H)) \le \overline{\delta}(w, S_L \cup C_L \backslash \{\hat{v}\}) \le \overline{\delta}(w, S_L \cup C_L) - 1 \le (k+1) - 1 = k$ since $L(H) \subseteq S_L \cup C_L \backslash \{\hat{v}\}$ (note that H is a subgraph of $G[S \cup C]$ and $\hat{v}$ has been excluded), vertex w disconnects vertex $\hat{v}$ and $\overline{\delta}(w, S_L \cup C_L) \le k+1$ based on the pruning condition (a); (3) For a vertex $u \in S_L \cap L(H)$, we have $\overline{\delta}(u, R(H) \cup \{w\}) \le \overline{\delta}(u, S_R \cup C_R) \le k$ since $R(H) \cup \{w\} \subseteq S_R \cup C_R$ and every vertex in S_L disconnects no more than k vertices from $S_R \cup C_R$ based on the pivot selection. (4) For a vertex $u \in C_L \cap L(H)$, if u connects w, we have $\overline{\delta}(u, R(H) \cup \{w\}) = \overline{\delta}(u, R(H)) \le k$ since H is a k-biplex. If u disconnects w, we have $\overline{\delta}(u, R(H) \cup \{w\}) \le \overline{\delta}(u, S_R \cup C_R) \le k$ since $R(H) \cup \{w\} \subseteq S_R \cup C_R$ and pruning condition (b).

We then improve the analysis of Scenario 2 by using measure-and-conquer techniques [11]. Specifically, we propose a novel function $T_3(\rho)$ for bounding the maximum number of recursions, which assigns different weights to the vertices in C:

$$T_3(\rho),\ \rho = n_{\le k} \times w_1 + n_{k+1} \times w_2,\ 0 < w_1 < w_2 \le 1, \tag{17}$$

where $n_{\le k}$ (resp. n_{k+1}) is the number of vertices with no more than k (resp. exactly $k+1$) disconnections in C. We remark that the previous function $T_2(n)$ assigns the equal weight, i.e., 1, to all vertices in C. The new proposed function is motivated by following observations/intuitions, namely (1) $G[S \cup C]$ in Scenario 2 is a $(k+1)$-biplex and would become a k-biplex if no vertex in $S \cup C$ disconnects $k+1$ vertices at the other side, and (2) those vertices with $k+1$ disconnections have higher effects on the analysis while vertices with no more than k disconnections is of less importance.

We denote a vertex with exactly $k+1$ disconnections by a *boundary* vertex. Wlog, assume that $\hat{v} \in C_L$ and there are x boundary vertices in $\overline{\Gamma}(\hat{v}, C_R)$. We derive $x \ge 1$ in the worst-case (since otherwise Pruning 1 is triggered). Wlog, assume the first $k+1-x$ (resp. the last x) vertices in $\overline{\Gamma}(\hat{v}, C_R)$ are non-boundary vertices (resp. boundary vertices). Consider the worst case where $a = k$, $b = k+1$, and neither pruning 1 nor pruning 2 would be triggered, we analyze $k+2$ branches by $T_3(\rho)$ below.

- For B_1, we exclude only a boundary vertex $\hat{v}$ in the worst case. After branching, those x boundary vertices in $\overline{\Gamma}(\hat{v}, C_R)$ will disconnect k vertices from $S_L \cup C_L \backslash \{\hat{v}\}$ (since they all disconnect $\hat{v}$). Therefore, the number of recursions of B_1 is
$$T_3(\rho - (w_2 - w_1)x - w_2). \tag{18}$$
- For the remaining branches, i.e., $\{B_2, B_3, ..., B_{k+2}\}$, recall that B_i ($i \ge 2$) includes vertices $\{\hat{v}, u_1, ..., u_{i-2}\}$ and excludes vertex u_{i-1}. For each branch B_i, we have (1) a boundary vertex $\hat{v}$ is removed, (2) except $\hat{v}$, there exists at least a boundary vertex disconnected to u_{i-1} (since otherwise Pruning 2 would be triggered) and it would disconnect to k vertices after excluding u_{i-1}, and (3) for each branch B_i in $\{B_2, ..., B_{k+2-x}\}$, it removes $i-1$ vertices, i.e., $\{u_1, ..., u_{i-1}\}$, from C to S, each of which has the weight w_1. For each branch B_i in $\{B_{k+3-x}, ..., B_{k+2}\}$, it removes $k+2-x$ vertices with weight w_1 and $i-(k+2-x)$ vertices with weight w_2 from C to S. In summary, for each branch B_i in $\{B_2, ..., B_{k+2-x}\}$, we have
$$T_3(\rho - w_2 - (w_2 - w_1) - w_1) = T_3(\rho - w_1 i - (2w_2 - w_1)). \tag{19}$$
For each branch B_i in $\{B_{k+3-x}, ..., B_{k+2}\}$, we have
$$\begin{aligned} &T_3(\rho - w_2 - (w_2 - w_1) - w_1(k+2-x) - w_2(i - (k+2-x))) \\ &\quad = T_3(\rho - w_1(k+1-x) - w_2 i - (2w_2 - w_1)). \end{aligned} \tag{20}$$

Therefore, for the number of recursions in Scenario 2, we have

$$\begin{aligned} T_3(\rho) \le{}& T_3(\rho - (w_2 - w_1)x - w_2) + \sum_{i=1}^{k+1-x} T_3(\rho - w_1 i - (2w_2 - w_1)) \\ &+ \sum_{i=1}^{x} T_3(\rho - w_1(k+1-x) - w_2 i - (2w_2 - w_1)). \end{aligned} \tag{21}$$

It is easy to verify that we reach the maximum of $T_3(\rho)$ when $x = 1$. In the following analysis, we set $w_1 = 0.36$ and $w_2 = 1$ since (1) the best choice of w_1 and w_2 is unknown and (2) finding the best choice is challenging and far too enough (for example, if we can find a setting of w_1 and w_2 to ensure that the second scenario performs better than the first scenario, the worst-case running time will be bounded by the first scenario). Hence, the worst-case running time is $O(\gamma_k^{\rho})$ where $\rho \le |V|$ and γ_k is the largest positive real root of $x^{0.36k+3.64} - x^{0.36k+2.64} - x^{0.36k+2} - x^{0.36k+1.64} + x^{0.36k+1.28} + x^{0.36k+1} + x^{1.36} + 1 = 0$. For example, when $k = 1$, 2 and 3, we have $\gamma_k = 1.753$, 1.866 and 1.945. Hence, Scenario 2 performs better than Scenario 1. □

C ADDITIONAL PROOFS IN SECTION 4

We prove the correctness of claims mentioned in Section 4.2.

Claim 1. *Given $|L|$ subgraphs, namely $G_i = (L_i, R_i, E_i)$ for $1 \le i \le |L|$ formed based on Equation (12)-(14), the MaxBP H^* must reside in one of the subgraphs.*

Proof. Assume that v_i is the vertex with the smallest index in H^*. We show that H^* would be found in G_i formed as above, which can be proved by contradiction. Suppose that such a H^* is not in G_i, which means $L(H^*) - L_i \neq \emptyset$ or $R(H^*) - R_i \neq \emptyset$. We thus consider following two cases.

- **Case 1** $L(H^*) - L_i \neq \emptyset$. Assume that a vertex $v \in L(H^*) - L_i$. We derive the contradiction by showing that H^* is not a k-biplex since vertex v disconnects more than k vertices, i.e., $\overline{\delta}(v, R(H^*)) \geq \delta(v_i, R(H^*)) = |R(H^*)| - \overline{\delta}(v_i, R(H^*)) \geq \theta_R - k \geq (2k+1) - k = k+1$ (note that (1) $\overline{\delta}(v, R(H^*)) \geq \delta(v_i, R(H^*))$ since v is not a 2-hop neighbor of v_i by assumption and thus disconnects v_i's neighbors in $R(H^*)$, (2) $|R(H^*)| \geq \theta_R \geq 2k+1$ by the definition of problem and (3) $\overline{\delta}(v_i, R(H^*)) \leq k$ since H^* is a k-biplex and includes vertex v_i).
- **Case 2** $R(H^*) - R_i \neq \emptyset$. Assume that a vertex $u \in R(H^*) - R_i$. We derive the contradiction by showing that H^* is not a k-biplex since u disconnects more than k vertices, i.e., $\overline{\delta}(u, L(H^*)) = |L(H^*)| \geq \theta_L \geq 2k+1$ (note that $\overline{\delta}(u, L(H^*)) = |L(H^*)|$ since (1) $L(H^*) \subseteq L_i$ otherwise we can finish the proof based on Case 1 and (2) u disconnects all vertices in $L(H^*)$ based on the definition of R_i).

Based on the above two cases, we finish the proof. □

Claim 2. *Given subgraph G_i ($1 \leq i \leq |L|$) formed based on Equation (12)-(14), the following vertices from a subgraph can be pruned from G_i without missing the MaxBP to be found in G_i.*

- $v \in L_i$ with $\delta(v, R_i) < \theta_R - k$ or $|\Gamma(v, R_i) \cap \Gamma(v_i, R)| < \theta_R - 2k$;
- $u \in R_i$ with $\delta(u, L_i) < \theta_L - k$.

Proof. The idea is to show that these vertices cannot be appeared in the MaxBP H to be found in G_i. This can be proved by contradiction. In general, there are three cases.

- **Case 1**. Assume that H includes a vertex $v \in L_i$ such that $\delta(v, R_i) < \theta_R - k$. We derive the contradiction by showing that H is not a k-biplex since vertex v disconnects more than k vertices from $R(H)$, i.e., $\overline{\delta}(v, R(H)) = |R(H)| - \delta(v, R(H)) > \theta_R - (\theta_R - k) = k$.
- **Case 2**. Assume that H includes a vertex $v \in L_i$ such that $|\Gamma(v, R_i) \cap \Gamma(v_i, R)| < \theta_R - 2k$. We derive the contradiction by showing that H is not a k-biplex since vertex v disconnects more than k vertices from $R(H)$, i.e., $\overline{\delta}(v, R(H)) > \delta(v_i, R(H)) - (\theta_R - 2k) \geq (|R(H)| - k) - (\theta_R - 2k) \geq (\theta_R - k) - (\theta_R - 2k) = k$ (note that (1) $\overline{\delta}(v, R(H)) > \delta(v_i, R(H)) - (\theta_R - 2k)$ since v has less than $\theta_R - 2k$ common neighbors with v_i in R and thus connects at most $\theta_R - 2k$ vertices at $\Gamma(v_i, R(H))$ and (2) $\delta(v_i, R(H)) \geq |R(H)| - k$ since H is a k-biplex).
- **Case 3**. Assume that H includes a vertex $u \in R_i$ such that $\delta(u, L_i) < \theta_L - k$. We derive the contradiction by showing that H is not a k-biplex since vertex u disconnects more than k vertices from $L(H)$, i.e., $\overline{\delta}(u, L(H)) = |L(H)| - \delta(u, L(H)) > \theta_L - (\theta_L - k) = k$.

Combining the above three cases, we thus finish the proof. □