Efficient Maximal Biclique Enumeration for Large Sparse Bipartite Graphs

Lu Chen*
Swinburne University of Technology
Melbourne, Australia
luchen@swin.edu.cn

Chengfei Liu
Swinburne University of Technology
Melbourne, Australia
cliu@swin.edu.cn

Rui Zhou
Swinburne University of Technology
Melbourne, Australia
rzhou@swin.edu.cn

Jiajie Xu*
Soochow University
Suzhou, China
xujj@suda.edu.cn

Jianxin Li
Deakin University
Melbourne, Australia
jianxin.li@deakin.edu.au

ABSTRACT

Maximal bicliques are effective to reveal meaningful information hidden in bipartite graphs. Maximal biclique enumeration (MBE) is challenging since the number of the maximal bicliques grows exponentially w.r.t. the number of vertices in a bipartite graph in the worst case. However, a large bipartite graph is usually very sparse, which is against the worst case and may lead to fast MBE algorithms. The uncharted opportunity is taking advantage of the sparsity to substantially improve the MBE efficiency for large sparse bipartite graphs. We observe that for a large sparse bipartite graph, a vertex v may converge to a few vertices in the same vertex set as v via its neighbours, which reveals that the enumeration scope for a vertex could be very small. Based on this observation, we propose novel concepts: unilateral coreness for individual vertices, unilateral order for each vertex set and unilateral convergence (ς) for a large sparse bipartite graph. ς could be a few thousand for a large sparse bipartite graph with hundreds of million edges. Using the unilateral order, every vertex with τ unilateral coreness only needs to check at most 2^τ combinations so that all maximal bicliques can be enumerated and τ is bounded by ς, which leads to a novel MBE algorithm running in $O^*(2^\varsigma)$. We then propose a batch-pivots technique to eliminate all enumerations resulting in non-maximal bicliques, which guarantees that every maximal biclique is reported in $O(\varsigma e)$-delay, where e is the number of edges. We devise novel data structures that allow storing subgraphs at omissible space for further speeding up MBE. Extensive experiments are conducted on synthetic and real large datasets to justify that our proposed algorithm is faster and more scalable than the existing algorithms.

PVLDB Reference Format:
Lu Chen, Chengfei Liu, Rui Zhou, Jiajie Xu, and Jianxin Li. Efficient Maximal Biclique Enumeration for Large Sparse Bipartite Graphs. PVLDB, 15(8): 1559-1571, 2022.
doi:10.14778/3529337.3529341

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/S1mpleCod/cohesive_subgraph_bipartite.

*The corresponding authors.

Proceedings of the VLDB Endowment, Vol. 15, No. 8 ISSN 2150-8097.
doi:10.14778/3529337.3529341

1 INTRODUCTION

Biclique and its applications. A bipartite graph $G=(L,R,E)$ consists of two disjoint vertex sets L and R and the edges $E \subseteq L \times R$. Given a pair of vertex sets $(A \subseteq L, B \subseteq R)$, if for every $(u,v) \in A \times B$, $(u,v) \in E$, (A,B) is a biclique. If a biclique (A,B) cannot be further enlarged, (A,B) is a maximal biclique.

Finding bicliques in a bipartite graph is critical for bipartite graph data analytics and has a myriad of applications such as text mining [31, 35, 45], biological data analysis [4, 23, 38, 44], web community discovery [3, 20, 22, 29, 30, 32], and so on.

In this paper, we focus on devising efficient algorithms for maximal biclique enumeration (MBE). MBE has been recognised as the foundation of various biclique problems and studied in [2, 12, 17, 47]. We further justify the importance of solving the MBE problem efficiently as follows.

Importance of efficient MBE algorithm. An efficient maximal biclique enumeration algorithm can improve time complexities for various biclique problems and provide insightful hints for users to search for desired bicliques.

Improving time complexities for various biclique problems. For real applications, additional constraints, such as minimum number of vertices or edges, can be applied for discovering desired bicliques, which makes the biclique search problems inside the $W[i]$-hierarchy [27] with $i \geq 1$, i.e., they are not in the fixed-parameter tractable (FPT) class. This means, for those problems, even when specific parameters are provided, it is unlikely to have algorithms that run better than exponential time w.r.t. the size of a bipartite graph in the worst case. As such, efficient MBE is critical to a myriad of such biclique problems that are not in FPT. For instance, considering the size-constrained maximum edge biclique problem (MEB), the state-of-the-art algorithm [30] still has the same time complexity as its base which is inherently maximal clique enumeration although elegant pruning techniques were proposed. Improving the practical performance and time complexity of MBE are indeed important and can benefit variants of non-FPT biclique problems.

Providing searching hints. In real applications, users face challenges to find the desired bicliques by specifying precise size or edge constraints based on both their desiderata and the characteristics of datasets. Users may try different constraints, which not only is user-unfriendly but also causes extra burden to a search engine. For a large sparse bipartite graph, if there is an algorithm running MBE fast to provide biclique distribution information for different users, the above drawbacks can be mitigated. Let us treat Figure 1(a) as a comment-contain-stem bipartite graph that is typical for the text-mining application scenario, say, vertices 1 to 4 are comments,

vertices 5 to 9 are stems, and an edge means that a comment contains a stem. Suppose that a user wants to cluster comments by the biclique semantic, i.e., comments sharing at least k common stems are considered as a cluster. Using MBE, a search engine can provide useful hints such as: when k=1, all comments are in the same cluster; when k=2, comments 2 and 3 are in a more refined cluster, and so on. As such, the user can determine k with fewer attempts.

State-of-the-art. Recent works [2, 12] are built based on the MBE algorithm iMBEA [47], dedicated for bipartite graphs.

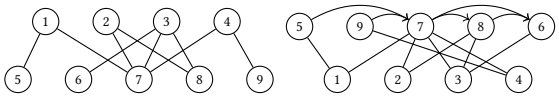
Enumeration schema [47]. iMBEA takes the advantage that all the maximal bicliques in a bipartite graph G=(L,R,E) can be enumerated by visiting the sets in either 2^L or 2^R (the powerset for L or R) and proposes to enumerate maximal bicliques from the vertex set with the minimum number of vertices recursively. In this paper, we *always assume* that R is the set of vertices that an MBE algorithm works on. The rationale that visiting 2^R leads to bicliques is as follows. Given a vertex set $B \in 2^R$ and let A be the common neighbours for the vertices in B, if $A \neq \emptyset$, (A,B) is a biclique. iMBEA uses a fixed order of R to enumerate 2^R with no repeating. iMBEA runs in $O(|R|d_{max}(R)2^{|R|})$ or $O(d^2_{max}(R)|R|^2\mathcal{B})$, where $d_{max}(R)$ denotes the maximum degree of vertices in R and $\mathcal{B}$ denotes the actual number of maximal bicliques contained in G.

Recent speeding-up [2, 12]. In [2], they discover that given two vertices $u,u' \in R$, if the neighbours of u' are a subset of the neighbours of u, then the maximal biclique containing u' can always be expanded by adding u. u is called a *pivot*. They propose a pivot-based algorithm that outperforms iMBEA in practice but still has the same time complexity as iMBEA. On the other hand, in [12], they show that given an order of R, the input bipartite graph can be partitioned into a set of subgraphs. Therefore, they divide the MBE problem for G into $|R|$ MBE problems, which achieves practical speeding-up.

In this paper, we focus on devising an efficient algorithm for enumerating all the maximal bicliques for a real large sparse bipartite graph with parameterised complexity much better than the existing algorithms, which enables solving the MBE problem in a reasonable time even with a single machine when a dataset is large.

Uncharted opportunity for MBE. We observe that real large bipartite graphs are sparse, i.e., the bipartite edge density $\frac{|E|}{|L|\times|R|}$ is small. Based on the statistics from [21], when the number of edges is at the million scale, the edge density is usually smaller than 10^{-5} and when the number of edges is at a hundred million scale, the edge density is smaller than 10^{-6}. The theory indicating that the sparsity could lead to an efficient MBE algorithm is below.

The parameterised approach. One promising way to solve the MBE problem efficiently for a large sparse bipartite graph is transforming the MBE problem to a set of parameterised problems. A parameterised problem is fixed-parameter tractable if the problem with input size n and a parameter p can be solved in $f(p)n^{O(1)}$, where f is allowed to grow exponentially w.r.t. p but is independent of n [14]. As such, if we can discover an effective and efficient transformation such that 1) the parameter for every transformed parameterised problem is no greater than p, 2) p is considerably smaller than $|L|$ or $|R|$, 3) the total number of parameterised problems is $|L|$ or $|R|$ and 4) the transformation runs in polynomial time w.r.t. the size of the bipartite graph, then the MBE problem can be solved substantially fast. Since all the existing algorithms for MBE do not bear the parameterised approach in mind when devising the algorithms, after adapting them to parameterised approaches, p can only be bounded by $|R|$. As such, although there could exist p in a large sparse bipartite graph with p significantly less than $|R|$, the existing algorithms still run in $O^*(2^{|R|})$. Novel techniques are sought after.



(a) Sparse bipartite graph **(b) The proposed order**

Figure 1: Examples

Our approach. We propose a novel MBE algorithm ooMBEA running in $O^*(2^{\varsigma(R)})$ by transforming the MBE problem on G into up to $|R|$ parameterised problems, where $\varsigma(R)$ is our proposed parameter named as *unilateral convergence*. $\varsigma(R)$ is small. For datasets such as *LiveJorunal* containing more than 112 million edges, $\varsigma(R)$ is only 7616. Major technical highlights of ooMBEA are as follows.

Effective and efficient transformation. We propose novel concepts: the unilateral coreness for individual vertices, the unilateral order for each vertex set and the unilateral convergence ($\varsigma(R)$) for a large sparse bipartite graph. We reveal that, by enforcing the unilateral order on the vertices in R, for every vertex $v \in R$ with the unilateral coreness of $\tau(v)$, ooMBEA only needs to check at most $2^{\tau(v)}$ subsets of R recursively and $\tau(v)$ is bounded by $\varsigma(R)$. We prove that ooMBEA only needs to check at most $|R|2^{\varsigma(R)}$ subsets of R to enumerate all the maximal bicliques, which is clearly much smaller than the existing bound $2^{|R|}$. We also show that the transformation runs in $O(d_{max}(L)|E(G)|)$ and much faster in practice since the degree distribution for a large sparse bipartite graph is skewed.

More effective pruning. Although the transformation reduces the upper bound of the number of subsets of R that we have to check, there still exist subsets that lead to non-maximal bicliques. To further reduce the subsets to be checked, we propose the batch-pivots technique. The main ideas are as follows. For each recursive subproblem expanding new subsets from a checked subset B, we further explore the subgraph $H \subseteq G$ induced by the vertices $A \subseteq L$ that are common neighbours of vertices in B and every vertex $v \in R \setminus B$ such that the intersection of the neighbours of v with A is not an empty set. We choose a batch of pivots consisting of *every* vertex $v \in R(H)$ whose neighbours are not a subset of the neighbours of any other v' in $R(H)$. We prove that, every vertex in the batch of pivots must lead to a maximal biclique and any other vertex of $R(H)$ cannot lead to a maximal biclique and therefore shall not be used to generate new subsets. This means, every recursive subproblem to be solved after the batch-pivots optimisation must report a maximal biclique, which shaves the necessity of maximality checking and guarantees that ooMBEA reports a maximal biclique in $O(\varsigma(R)|E(G)|)$-delay. In contrast, the best existing polynomial delay algorithm for MBE reports a maximal biclique in $O(d^2_{max}(R)|R|^2)$-delay. We also devise an efficient algorithm to compute the batch-pivots by caching and exploring local subgraphs.

Effective space consumption. Storing local subgraphs in a straightforward way makes the space complexity as high as $O(\varsigma(R)|E(G)|)$.

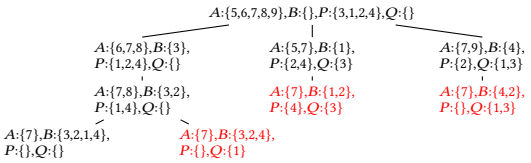


Figure 2: Recursion for iMBEA with pivoting

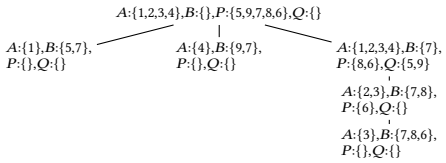


Figure 3: Recursion for our proposed algorithm

To get rid of the space consumption caused by storing local subgraphs, we use the input vertex sets L, R and the adjacency list among recursive subproblems while adjusting the positions of vertices according to each recursive subproblem. When backtracking to the current recursive subproblem, the positions vertices in R, L and adjacency list are restored to ensure the correctness. We study the necessary information needed for restoring. With our effort, the space consumption of our proposed algorithm is the same as the space complexity of without storing local subgraphs, i.e., $O(\varsigma(R)|L|+|G|)$.

Our principal contributions are summarised as follows.

(1) We explore how to use sparsity of a real large bipartite graph for devising efficient MBE algorithms. **Section 4**
   (a) We propose novel concepts: the unilateral coreness, order and convergence to reveal why the MBE problem can be solved fast for large sparse bipartite graphs.
   (b) We propose a novel MBE algorithm that runs in $O^*(2^{\varsigma(R)})$, where $\varsigma(R)$ is only a few thousand for bipartite graphs with hundreds of million edges.
(2) We propose effective batch-pivots techniques. **Section 5**
(3) The novel data structure shaves space overheads. **Section 6**
(4) Extensive experiments are conducted to evaluate the proposed techniques on real large datasets. **Section 8**

2 PROBLEM FORMULATION

Bipartite graph. Let $G=(L,R,E)$ denote a bipartite graph. L and R are the two sets of disjoint vertices in G. E is the set of edges in G and $E\subseteq L\times R$. For instance, Figure 1(a) shows a bipartite graph with $L=\{5,6,7,8,9\}$ and $R=\{1,2,3,4\}$.

W.l.o.g., we use $H\subseteq G$ to denote a subgraph of G and $N(v,H)$ to denote the neighbours of v in H. Given a set of vertex $X\subseteq L$ or $X\subseteq R$, we use $\Gamma(X,H)$ to denote the common neighbours of vertices in X, i.e., $\Gamma(X,H)=\cap_{v\in X}N(v,H)$ and $\Upsilon(X,H)$ to denote the union of the neighbours of vertices in X, i.e., $\Upsilon(X,H)=\cup_{v\in X}N(v,H)$. We use $N_2(v,H)$ to denote 2-hop neighbours of v in H, i.e., $N_2(v,H)=\Upsilon(N(v,H),H)\setminus\{v\}$. When the context H is clear, we use $\Gamma(X)$, $\Upsilon(X)$, and $N_2(v)$ for brevity.

Maximal biclique. A pair of vertex sets (A,B) is a biclique if $\forall(u\in A,v\in B)\in A\times B$, $(u,v)\in E$. If (A,B) cannot be further enlarged, (A,B) is a *maximal biclique*. For instance, in Figure 1(a), $(\{6,7,8\},\{3\})$ and $(\{7,8\},\{2,3\})$ are two of the maximal bicliques.

Problem statement. Given a bipartite graph G, report all the maximal bicliques in G.

3 STATE-OF-THE-ART

In this section, we revisit recent popular MBE algorithms.

Data structures for MBE. To efficiently enumerate maximal bicliques, the existing works such as [2, 12, 47] design algorithms that mainly work on four sets, $A\subseteq L$, and $B,P,Q\subseteq R$. The sets A, B are for storing candidate bicliques. P is the set of vertices for expanding B, Q is the set of vertices that have been expanded previously, and B,P,Q are disjoint. We use Algorithm 1 to demonstrate the algorithm incorporating the optimisations in the existing works.

Generating powerset leading to bicliques [47]. Let us first skip all the prunings and maximality checking in Algorithm 1. Initially, sets A,B,Q are empty sets and $P=R$. Algorithm 1 recursively generates $B\in 2^R$ by moving a vertex $v\in P$ to B and maintains A as $\Gamma(B)$, which leads to a biclique (A,B) (line 4). Then, for P, every vertex that cannot form a biclique with B are excluded by line 15. During the backtracking, the operation moving v from P to Q ensures that a set of 2^R is generated only once during the enumeration (line 18).

Maximality checking [47]. Although $B\in 2^R$ is generated with no repeating, the biclique which B leads to could be non-maximal, i.e., (A,B) could be non-maximal. In [47], the vertex set Q is introduced for checking the maximality. Initially Q is an empty set. For a recursive subproblem generating B and the corresponding A, Q is maintained as the set of vertices that 1) previously were in P (line 18) and 2) $\forall v\in Q$, $N(v)\cap A\neq\emptyset$ (lines 6 to 9). As such, the maximality checking becomes that if $\exists v\in Q$, $N(v)\cap A=A$, (A,B) is not a maximal biclique (line 7).

Embedded prunings. In the existing works, the neighbour containment based pruning is proposed and implemented differently. We first review the neighbour containment based pruning.

Neighbour containment based pruning (NCP) [2, 47]. Given a subgraph $H\subseteq G$, and two vertices v and v', if $N(v',H)\subseteq N(v,H)$, then all the bicliques in H containing v' without v can be enlarged by adding v. This idea can be used to reduce recursive subproblems that do not lead to maximal bicliques. In Algorithm 1, for every recursive subproblem, H is $P\cup A$ induced subgraph of G.

Sort based NCP. In [47], for a recursive subproblem with A,B,P,Q (or H), $\forall v\in P$, they are sorted by $|N(v)\cap A|$ (or $|N(v,H)|$) in a non-increasing order. Then, the algorithm branches at the vertex with the maximum $|N(v)\cap A|$. The rationale is that the larger $|N(v)\cap A|$ is, the higher chance of any other vertex v' with $N(v')\cap A\subseteq N(v)\cap A$ and therefore can be pruned.

Pivot based NCP. In [2], for a recursive subproblem, a vertex $v^*\in P$ such that v^* can maximise the pruning is selected as the pivot[1], i.e., v^* is $\arg\max_{v\in P}\{|S||S\subseteq P\wedge\forall v'\in S, N(v')\cap A\subseteq N(v)\cap A\}$. Due to the high computational cost of the pivot selection, an index is built, which is a partial order of the vertices in R according to the global neighbour containment relationships in G. For v and v', if $N(v')\subseteq N(v)$, a directed edge from v to v' is created. As such, given a vertex v, its pruning power can be estimated by traversing the partial order

[1] v^* serves as a pivot for P since it partitions P into two set of vertices, i.e., the vertices can be pruned and other vertices

Table 1: Recent MBE algorithms on bipartite graphs

Algorithm	Signature optimisations	Time complexity
iMBEA [47]	Maximality checking, sort based NCP	$O(\|L\|\|R\|2^{\|R\|})$
FMBEA [12]	Divide-and-conquer	$O(d^4_{max}2^{\|R\|})$
PMBEA [2]	Pivot based NCP, index for FMBEA pivot selection	$O(\|L\|\|R\|2^{\|R\|})$
ooMBEA	Unilateral order, batch-pivots, optimised data structure	$O(\varsigma(R)\|R\|\|E\|2^{\varsigma(R)})$

from the vertex via forwarding edges. The optimal pivot is selected heuristically based on the index. In Algorithm 1, lines 2 and 3 show how a selected pivot prunes branches.

In fact, using global containment relationships could make the pruning suboptimal. For instance, in Figure 2, for the recursive subproblem with A:{7,8}, B:{3,2}, P:{1,4}, Q:{}, the neighbours of 1 cannot contain those of 4 based on Figure 1(a), which makes the recursion A:{7}, B:{3,2,4}, P:{}, Q:{1} be explored but fruitless.

Aggressive expansion [47]. A special case of NCP based pruning is studied. Given $H \subseteq G$, and a vertex v used to expand a biclique, any vertex v' such that $N(v',H)=N(v,H)$ can be directly move to B together with v, which can reduce the enumeration depth. In Algorithm 1, B^+ and line 13 implement this idea. For instance, in Figure 2, the recursive subproblem with A:{7,8}, B:{3,2}, P:{1,4}, Q:{} aggressively includes {1,4} to B, leading to the recursive subproblem with A:{7}, B:{3,2,1,4}, P:{}, Q:{}.

Time complexity. The time complexity of Algorithm 1 has been interpreted quite differently by the exiting works [2, 12, 47] shown in Table 1. The most used interpretation is $O(|L||R|\mathcal{B})$, where $\mathcal{B}$ is claimed to be the total number of maximal bicliques in G. In fact, it should be $O(|L||R|2^{|R|})$ [47]. Alternatively, Algorithm 1 runs in $O(d^4_{max}\mathcal{B})$, since it reports each biclique in $O(d^4_{max})$-delay [47].

Opportunities. After revisiting the state-of-the-art techniques, we find the opportunities below. First, the time complexities of the existing algorithms are dominated by $2^{|R|}$ and $|R|$ is quite large for a real dataset. Although the existing works have made great efforts to reduce generating fruitless sets, those efforts cannot reduce the time complexity. Can we improve the time complexity for MBE? Second, NCP is implemented with heuristics that may compromise the pruning effectiveness. It would be great to explore the full potential of NCP. At last, the data structure for efficiently enumerating maximal biclique has not been studied carefully, which can practically speed up the algorithm further.

In this paper, for large sparse bipartite graphs, we explore the above opportunities and propose novel techniques summarised in Table 1, leading to an MBE algorithm (ooMBEA) with better time complexity, faster practical performance and better scalability.

4 ORDER OPTIMISED MAXIMAL BICLIQUE ENUMERATION

In most of the MBE algorithms, having a fixed total search order is important since it ensures that the powerset for R or L can be enumerated without repetition. However, the power of total search order has not been fully explored for the MBE problem. We reveal that applying a carefully studied order leads to an MBE algorithm with much more promising time complexity.

Algorithm 1: iMBEA(A,B,P,Q)

```
1  Procedure iMBEA(A,B,P,Q)
       /* Pivot-based pruning                                   */
2      p ← select a pivot from P;
3      foreach v∈{p}∪{v|v∈P,N(v)⊈N(p)} do
4          A',P',Q' ← ∅, B ← B∪{v}, A' ← A∩N(v);
           /* Maximality checking                               */
5          isMaximal ← true;
6          foreach q∈Q do
7              if A'⊆N(q) then isMaximal ← false, break ;
8              else
9                  if A'∩N(q)≠∅ then Q' ← Q'∪{q} ;
10         if isMaximal then
11             B⁺ ← ∅;
12             foreach v'∈P\{v} do
13                 if N(v')∩A'==A' then B⁺ ← B⁺∪{v'} ;
14                 else
15                     if N(v')∩A'≠∅ then P' ← P'∪{v'} ;
16             report (A',B∪B⁺);
17             if P'≠∅ then iMBEA(A', B∪B⁺,P',Q') ;
18         P ← P\{v}, Q ← Q∪{v};
```

4.1 Ordering vertices for bipartite graphs

Although considering ordering optimisations for improving the performance of maximal clique enumeration has been well studied, the study on that for improving the maximal biclique enumeration problem is still at a preliminary stage.

State-of-the-art. An order tailored for the maximum balanced biclique (MBB) problem, known as the bidegeneracy order, has been proposed in [9]. This order is derived by the peeling sequence of L and R based on their 1-hop and 2-hop neighbours, i.e., progressively removing the vertex in $L \cup R$ with the minimum number of 1-hop and 2-hop neighbours up to the time. The bidegeneracy order is more suitable for finding an MBB but not for MBE because of the reasons below. For the MBB problem, due to the balanced constraint, when generating a biclique, popular algorithms such as [9, 49] prefer to expand the two sets (A,B) in turn to avoid wasting time on computing imbalanced bicliques, which essentially checks up to $\sum_{i=1}^{|R|}\binom{|R|}{i}\cdot\binom{|L|}{i}$ sets, supposing $|R| \leq |L|$ in the worst case. In contrast, for the MBE problem, it is highly desired that the vertices belonging to a maximal biclique (A,B) can be included at the same time to reduce the number of recursions as much as possible. Therefore, for the MBE problem, it is more efficient to just enumerate either up to $2^{|L|}$ or $2^{|R|}$ sets as discussed. From the discussion, we can see that the bidegeneracy order is for $L \cup R$. In contrast, the order needed for MBE is for either L or R only.

Why ordering matters for MBE. Inspired by [9, 12], given an ordered vertex set, MBE on G is equivalent to MBE on a set of subgraphs. For MBE, the subgraphs are defined as follows.

Search scope of ordered vertices. Given a permutation of $R=(v_1,\ldots,v_{|R|})$, and $v_i \in R$, the search scope for v_i is the subgraph induced by $R^+_{v_i} \cap N_2(v_i)$ and $N(v_i,G)$, where $R^+_{v_i}$ are vertices $v_{i'}$ in R with $i' \geq i$.

Following the permutation of R, after enumerating maximal bicliques in the subgraph derived from every vertex in R, all maximal

bicliques can be derived. For instance, following the order indicated in Figure 1(b), ({5,7},{1}), ({9,7}, {4}), ({7,8,6}, {1,2,3,4}), ({8,6}, {2,3,4}) and ({6},{3}) induced subgraphs contain all maximal bicliques.

From the above discussion we can see that, for the MBE problem, a desired order should be 1) it can nicely bound $|R^+_{v_i} \cap N_2(v_i)|$ for all $1 \le i \le |R|$ tightly, 2) it can be computed with low cost , and 3) the upper bound of $|R^+_{v_i} \cap N_2(v_i)|$ is guaranteed small when a bipartite graph is sparse. Such an order can greatly reduce the time complexity since MBE is dominated by enumerating $2^{R^+_{v_i} \cap N_2(v_i)}$ for each v_i. However, for the MBE problem, such an order has not been studied yet.

4.2 The proposed order and search framework

We observe that when a bipartite graph is sparse, a vertex $v \in R$ may *converge* to a small number of vertices in R via the neighbours of v. This motivates us to give the definitions below.

Definition 1. *k **unilateral core**. Given a vertex $v \in R$, v is in a k unilateral core if there is a subgraph $H \subseteq G$ containing v such that $\forall v' \in R(H), |N_2(v', H)| \ge k$. The maximum k for v is called the unilateral coreness of v, denoted by $\tau(v, G)$. The maximum τ for all the vertices in R is defined as the unilateral convergence for R, denoted by $\varsigma(R)$. The same can be defined on every vertex in L and the vertex set L.*

For the MBE problem, since we only enumerate 2^R or 2^L to visit all the maximal bicliques, an order for R or L is defined as follows.

Definition 2. ***Unilateral order**. A permutation of $R = (v_1, ..., v_{|R|})$ is a unilateral order if $\forall 1 \le i \le |R|$, v_i has the minimum two-hop neighbours in H^+, where H^+ is the subgraph of G induced by $R^+_{v_i} = \{v_j \mid i \le j \le |R|\}$ and $N(v_i, G)$. The same order can be defined on L similarly.*

For example, in Figure 1(b), a unilateral order of for the vertex set {5,6,7,8,9} is shown, which is (5,9,7,8,6).

Another view of the unilateral convergence. The unilateral convergence and the unilateral order for G are the same as the degeneracy and degeneracy order for an *auxiliary graph* G' based on G. G' is constructed as follows. $V(G')$ consists of every vertex in $R(G)$. There is a single edge between two vertices in G' if the two vertices in G are 2-hop reachable. By the construction and Definition 1, the degeneracy of G' equals to $\varsigma(R)$ of G clearly, and it is the same for the two orders.

Order optimised MBE search framework (ooMBEA). We are ready to introduce ooMBEA, shown in Algorithm 2. It first determines a total search order that can speed up the enumeration (lines 1 to 3). Then following the total search order, Algorithm 2 solves at most $|P|$ number of MBE problems on subgraphs induced by vertices in A, P' and Q' (lines 4 to 10).

The search framework is general. Any order could be applied. In this paper, we propose to use the unilateral order for either L or R, which leads to following questions. First, why the unilateral order is used? Second, what is the overhead of using unilateral order? Third, for L and R, which vertex set should be chosen for enumerating maximal bicliques?

We answer the above questions in the following sub-sections.

4.3 The advantages of using unilateral order

Using the unilateral order, the largest size of the subproblems solved by Algorithm 2 can be nicely bounded. Therefore, it would improve the time complexity for enumerating maximal bicliques.

Algorithm 2: ooMBEA((L,R,E))

```
1  P ← either L or R estimated to be faster, Q ← ∅;
2  Order vertices in P by some criteria;
3  prune P by ordered based containment based pruning;
4  foreach v ∈ P in order do
5      A ← N(v,G), B ← {v}, P' ← N_2(v) ∩ P, Q' ← ∅;
6      compute B^+ as Algorithm 1,report (A, B ∪ B^+);
7      foreach v ∈ Q do
8          if N(v) ∩ A ≠ ∅ then Q' ← Q' ∪ {v};
9      iMBEA(A, B ∪ B^+, P', Q') ;
10     P ← P \ {v}, Q ← Q ∪ {v};
```

To simplify the discussion, we assume using R to enumerate all the maximal bicliques and $\varsigma(R) \le \varsigma(L)$.

Lemma 1. *Using the unilateral order, Algorithm 2 runs in $O(|R| 2^{\varsigma(R)})$.*

Proof sketch. iMBEA in Algorithm 2 runs in $O^*(2^{|P'|})$ by line 9. We only need to show that the maximum $|P'|$ in Algorithm 2 is $\varsigma(R)$. Since R is in the unilateral order, for a vertex v, $|N_2(v, H^+)|$ is at most $\tau(v,G)$ by the definition of the unilateral order, which ensures that $|P'|$ is bounded by $max\{\tau(v,G) | v \in R\}$, i.e., $\varsigma(R)$. Since Algorithm 2 runs at most $|R|$ instances of MBE problem using Algorithm 1, Algorithm 2 runs in $O(|R| 2^{\varsigma(R)})$. □

Discussion. We show that if $\varsigma(R)$ of G is small, the density of G is small. By the definition of $\varsigma(R)$, G with $\varsigma(R)$ contains $(\varsigma(R)+1)|L|$ edges in the extreme case when every $u \in L$ induces $\varsigma(R)+1$ distinct edges. As such, the density of G is no greater than $\frac{(\varsigma(R)+1)|L|}{|R||L|}$ ($\frac{(\varsigma(R)+1)}{|R|}$ after simplification). This implies that when $\varsigma(R)$ is small, the density of G is small. The lower and upper bounds for $\varsigma(R)$ are $d_{max}(L)-1$ and $|R|-1$ respectively. The lower bound is derived as follows. If there is $u \in L$ with degree of $d_{max}(L)$, all the neighbours of u in R have the unilateral corness $d_{max}(L)-1$, which serves as a lower bound for $\varsigma(R)$. The upper bound is immediate. In the experimental studies, we show that $\varsigma(R)$ is small in real datasets and $\varsigma(R)$ often matches its lower bound.

Based the proposed Lemma 1, we propose a guideline to determine which vertex set should be enumerated for Algorithm 2.

Guideline 1. *If $|R| 2^{\varsigma(R)} \le |L| 2^{\varsigma(L)}$, enumerate maximal bicliques on R; otherwise, enumerate on L.*

4.4 Algorithm for the order

We have shown that using the unilateral order can reduce the time complexity for MBE. Two questions arise. First, how to compute the unilateral order? Second, what is the cost of computing the unilateral order?

Order computation. We discuss Algorithm 3 based on two properties for $G=(L,R,E)$ below.

Property 1. *By progressively deleting any $v \in R$ with $|N_2(v,G)| < k$ and edges incident to v, remaining vertices in R are in k unilateral cores.*

Property 2. *After deleting $v \in R$, for any $v' \in N_2(v,G)$, $|N_2(v',G)|$ reduces by 1, which does not affect all the other vertices in R.*

The correctness of the above two prosperities is clear based the auxiliary graph discussed previously.

Algorithm 3: ucOrder((L,R,E))

```
1  ∀ v∈R, compute |N2(v)|;
2  sort vertices in R in increasing order by their 2-hop neighbours;
3  O←∅, τmax ← 0;
4  foreach v∈R in order do
5      O← append v to O;
6      if |N2(v)|>τmax then τmax←|N2(v)| ;
7      τ(v)← τmax;
8      foreach v'∈N2(v) such that v'∉O do
9          |N2(v')| ← |N2(v')|−1 ;
10         reorder R by the updated |N2(v')|;
11 return {O,τ};
```

Based on Properties 1 and 2, Algorithm 3 is easy to understand due to its similarity with the classic core decomposition algorithm. Algorithm 3 progressively deletes v with the minimum number of two-hop neighbours in the remaining subgraphs, which is ensured by the reordering (line 10). Meanwhile, the unilateral coreness of each vertex is recorded by the iteratively updated τ_{max}. We discuss Algorithm 3 for self-completeness purpose. Our key contribution here is discovering Properties 1 and 2 so that Algorithm 3 is applicable to our problem. Algorithm 3 can be applied to L as well.

Order computational cost. Algorithm 3 runs in $O(|E|d_{max}(L))$. Line 1 takes $O(|E|d_{max}(L))$ to compute $|N_2(v)|$ for every vertex in $v\in R$. Line 2 takes $O(|R|)$ time using the bucket sort technique. The nested loop takes $O(|E|d_{max}(L))$ since it visits two-hop neighbours for each vertex in R. Due to Property 2, reordering (line 10) can be done in $O(1)$ using the linear heap data structure [5]. Algorithm 3 runs much faster for a large sparse bipartite graph, since the degree distribution is skewed, i.e., high degree vertices are not many.

In summary, let R be the vertex set chosen to be enumerated by Guideline 1, Algorithm 2 runs in $O^*(2^{\varsigma(R)})$. We will show that $\varsigma(R)$ for a large sparse bipartite graph is significantly smaller than $|R|$ in practice in the experimental studies.

5 PRUNINGS BY A BATCH OF PIVOTS

In the previous section, we propose novel techniques that reduce the upper bound of the number of subsets of R to be checked for enumerating all the maximal bicliques, which leads to an MBE algorithm with a promising time complexity. There is still a room for improvement. For the subsets leading to non-maximal bicliques, if we can avoid generating and checking them, the MBE algorithm can run much faster in practice. We propose a novel batch-pivots technique. The immediate benefit of the batch-pivots technique is to eliminate all recursive subproblems generating subsets that result in non-maximal bicliques.

Before discussing the batch-pivots technique, we conduct a systematic study on the single pivot technique proposed in [2]. Realising that the single pivot technique could be adapted for MBE from the maximal clique enumeration (MCE) [36] for a unified graph, we first revisit the benefits that the single optimal pivot strategy can bring for MCE. Then we analyse that the single optimal pivot strategy cannot reduce the time complexity of MBE, making it less promising for MBE. In contrast, the batch-pivots technique can further enhance Algorithm 2 with a better polynomial-delay.

5.1 Single optimal pivot strategy

Single optimal pivot strategy for MCE. We first revisit the MCE problem and algorithm, and then discuss why the single optimal pivot strategy reduces the time complexity of MCE.

The MCE problem [25, 26, 36]. Given a graph $G=(V,E)$, a clique in G is a subgraph in G that every two different vertices in the subgraph are adjacent. A clique is called a maximal clique if the clique cannot be enlarged. The MCE problem reports all maximal cliques in G.

The MCE algorithm [36]. We revisit the algorithm first. Algorithm 4 uses three disjoint sets R, P and X to non-repetitively generate all sets in $2^{V(G)}$ that are cliques in G, and reports maximal cliques. The properties that R,P,X hold are as follows. R stores a clique during the enumeration. P stores the vertices to expand R. X stores the vertices that are previously in R while connecting every vertex in R. Initially, R and X are $\emptyset$ and P contains all the vertices in G. In each recursive subproblem, Algorithm 4 moves a vertex in P to R and then maintains the property of P and X by set operations (line 4). By maintaining the properties for R,P,X, if both P and X become $\emptyset$, R is reported as a maximal clique.

The above explanation ignores the pivoting technique applied in line 2. Without the pivoting technique Algorithm 4 runs in $O^*(2^n)$, where $n=|V(G)|$. This is because it has a recurrence of $T(n)=T(n-1)+T(n-2)+...+T(1)$ in the worst case, which is equivalent to $T(n)=T(n-1)+T(n-1)$ bounded by 2^n.

The pivoting technique. The pivoting technique is based on a property of the clique problem as follows. Given a vertex u, and its neighbours $N(u)$, for every clique in $N(u)$ induced subgraph of G, the clique can be enlarged by including u.

The pivoting technique leads to two reduction effects. First, it can reduce the number of branches, i.e., breadth reduction. Second, it can reduce the maximum size of subproblems that cannot be pruned, i.e., subproblem size reduction.

In [36], the trick that selects a pivot $u\in P\cup X$ such that u maximises $|N(u)\cap P|$ is proposed. By doing so, its breadth reduction is $|N(u)\cap P|$, i.e., only $c=|P|-|N(u)\cap P|$ branches left. It also ensures that for each branch that cannot be reduced, the subproblem size reduction is at least c as well, i.e., for line 4, $|N(v)\cap P|\le|N(u)\cap P|$ always holds. As such, the recurrence of Algorithm 4 is $T(n)=cT(n-c)$ in the worst case, which is bounded by $c^{\frac{n}{c}}$. When $c=3$, $c^{\frac{n}{c}}$ is the maximum for any $c\in\mathbf{Z}^+$. Therefore, $3^{\frac{n}{3}}$ is an upper bound for $T(n)=cT(n-c)$ and Algorithm 4 runs in $O^*(3^{\frac{n}{3}})$.

Single optimal pivot strategy for MBE. If we ignore the computational cost and mimic the same pivot selection trick as the MCE problem, can we make Algorithm 1 run in $O^*(3^{\frac{|R|}{3}})$? The answer is no, unfortunately. We provide a detailed analysis as follows.

We first introduce several definitions and symbols. The introduced definitions are based on the vertex sets A,B,P,Q for a recursive subproblem in Algorithm 1.

Definition 3. ***Local neighbour containment*** *($\subset_A$). Given $v,v'\in P\cup Q$, $v'\subset_A v$, if $N(v')\cap A\subset N(v)\cap A$. When $N(v')\cap A=N(v)\cap A$, if $v\ge v'$ (id of v is no less that of v'), then $v'\subset_A v$, and vice versa.*

Definition 4. ***Local containment of a vertex.*** *Let the set of vertices contained by $v\in P\cup Q$ be $\{v'|v'\in P,v'\neq v,v'\subset_A v\}$, denoted by $C_A(v)$. The local containment of v is $|C_A(v)|$.*

Notice that the single optimal pivot is p with $\arg\max_{v\in P}\{|C_A(v)|\}$.

Algorithm 4: MCE(R,P,X)

```
1 if P∪X==∅ then report R as a maximal clique ;
2 u ← arg max_{v∈P∪X} {|N(v)∩P|};
3 foreach v ∈ P\N(u) do
4     MCE(R∪{v}, P∩N(v), X∩N(v) );
5     P← P \ {v}, X←X∪{v};
```

Definition 5. ***Local two-hop neighbours***. *The local two hop neighbours of $v \in P$ is $N_2(v) \cap P$.*

$|N_2(v) \cap P|$ indicates the subproblem size after selecting v in the maximal biclique problem.

For a recursive subproblem of Algorithm 1 with p as the optimal pivot, the number of branches (line 3) is reduced to $|P| - |C_A(p)|$, which is the breadth-reduction effect of p. For each vertex v (line 3) that Algorithm 1 branches, the subproblem size $|P'|$ is $|N_2(v) \cap P|$ derived via lines 12 to 15, i.e., the subproblem reduction for each v is determined by the corresponding local two-hop neighbours of v, which may not always be the same as $|C_A(p)|$. Furthermore, the subproblem size reduction effect of the optimal pivot p (i.e., $|N_2(p) \cap P|$) is not always no less than $|N_2(v) \cap P|$, therefore cannot serve as an upper bound for any v looped by lines 12 to 15. In other words, the optimal pivot p cannot provide a *unified bounding effect* for both breadth and subproblem size reductions simultaneously. Consequently, Algorithm 1 cannot have a recurrence of $T(n) = cT(n-c)$ in the worst case, where $n = |R|$, and cannot run in $O^*(3^{\frac{n}{3}})$ by mimicking the pivoting technique for the maximal clique problem.

The above analysis shows that the complexity of MBE is difficult to reduce, which motivates us to seek new techniques for improving the polynomial-delay for MBE.

5.2 Selecting a batch of pivots

Along with the analysis for the single optimal pivot strategy, we first observe that, apart from finding a single optimal pivot, the local neighbour containment relationship has a nice property that enables us to select a batch of pivots. We then study the criteria for the optimal batch of pivots for MBE. Furthermore, we explore the important features of the *optimal* batch of pivots and prove that: 1) every branch that cannot be pruned by the optimal batch of pivots leads to a maximal biclique and 2) every vertex that leads to a maximal biclique is preserved in the optimal batch of pivots. As such, after adapting the batch-pivots technique to both Algorithms 1 and 2, Algorithm 2 runs much faster and also has a better polynomial-delay for reporting each maximal biclique.

We first introduce the optimal batch of pivots for MBE. The context of the discussions is a recursive subproblem that works on A, B, P and Q.

Definition 6. ***Optimal batch of pivots***. *An optimal batch of pivots denoted by S^* for a recursive subproblem shall be: 1) $\cup_{v \in S^*} C_A(v) = P$ and 2) there is no S' satisfying condition 1) while $|S'| < |S^*|$.*

We now show the nice property of Definition 3 that makes an optimal batch of pivots selection easy.

Property 3. ***Transitivity***. *Given $u, u', u'' \in P \cup Q$, if $u \subset_A u'$ and $u' \subset_A u''$, then $u \subset_A u''$.*

Property 3 clearly holds since the local neighbour containment relationship is defined based on set containment relationship. We are ready to propose the lemma below.

Lemma 2. *The optimal batch of pivots S^* for a recursive subproblem consists of every vertex $v \in P \cup Q$ such that there is no other vertex $v' \in P \cup Q$, $v \subset_A v'$.*

Proof sketch. We prove the lemma by contradiction. Assume that there are v and v' in S^* such that $v \subset_A v'$ exists. By Property 3, if we remove v, $|S^*|$ can be reduced by 1 while still satisfying the first condition in Definition 6, which contradicts the fact that S^* satisfies all the conditions in Definition 6. □

The optimal batch of pivots S^* can be easily embedded into Algorithm 1, i.e., we replace line 2 by deriving S^* and for line 3 we change the loop to $v \in S^* \cap P$. By using the optimal batch of pivots, the branch reduction effect can be maximised.

Bonus for the batch-pivots technique. We show the bonus for the batch-pivots technique in the following lemma.

Lemma 3. *Given a recursive subproblem with S^*, moving each $v \in S^* \cap P$ to B leads to a maximal biclique.*

Proof sketch. This lemma can be proved by contradiction. Suppose there is a vertex $v \in S^* \cap P$ leading to A', B', P', Q' that cannot generate a maximal biclique. This means that there is $v' \in P \cup Q$ that $v \subset_A v'$, which is against to the optimality of S^*. □

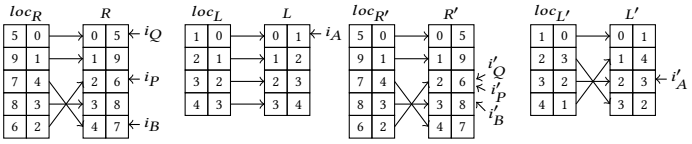
Lemma 3 reveals two facts. First, before each recursive call, if we always select S^* for reducing the branches, we do not need to check the maximality. Second, a better polynomial delay MBE algorithm could be devised if the cost of each recursive subproblem is nicely controlled.

Order-preserved batch-pivots technique. Applying the batch-pivots technique to Algorithm 2 directly cannot preserve that Algorithm 2 runs in $O^*(2^{\varsigma(R)})$. As such, we show a slightly modified batch of pivots to guarantee the time complexity.

Order-preserved batch of pivots. First, we modify Definition 3 as $v \subset v'$, if 1) v appears after v' in the unilateral order and 2) $N(v) \subset N(v')$. Then, the order-preserved pivot batch consists of any vertex v in the ordered R such that there exists no v' satisfying $v \subset v'$. Last but not least, the relative precedence of any two vertices in the order-preserved batch of pivots is the same as that of the two vertices in the unilateral order.

Where to apply. The order-preserved batch-pivots technique is used by line 3 in Algorithm 2 only, which is to ensure that line 6 in Algorithm 2 must report a maximal biclique and Algorithm 2 runs in $O^*(2^{\varsigma(R)})$. For every recursive subproblem solved by Algorithm 1 (iMBEA initialised by line 9 in Algorithm 2), the optimal batch-pivots technique is used.

Correctness. Algorithm 2 reports a maximal biclique for each v in the order-preserved batch of pivots. Since any two vertices in the pruned P still preserve their relative precedence in P before pruning and Algorithm 2 processes each vertex in the order-preserved batch of pivots (the pruned P) following the order, by the definition of the order-preserved batch of pivots, any v such that its neighbour vertex set is a subset of the neighbour vertex set of any other vertex processed before v shall not be in the order-preserved batch of pivots, which ensures that line 6 in Algorithm 2 reports a maximal biclique for each v in the pruned P.



(a) Before moving 8 to B **(b) After moving 8 to B**

Figure 4: Logical partitions and adjustment

For instance, for the bipartite graph shown in Figure 1(a), by Guideline 1, Algorithm 2 searches maximal bicliques following the order shown in Figure 1(b). After applying the order based batch-pivots optimisation, vertices 8 and 6 will be pruned. As such, in the search space equipping with our proposed batch-pivots technique, every recursive subproblem reports a maximal biclique.

Optimal batch-pivots computation. Given the fact that the overhead of the batch-pivots technique could be high if the algorithm for computing the batch of pivots is devised straightforwardly, i.e., performing $O(|R|^2)$ neighbour-intersections, where each neighbour-intersection runs in $O(d_{max}(R))$, we devise two techniques for speeding up the batch-pivots computation. We first propose to store the local subgraph H for each recursive subproblem to avoid computing the batch of pivots using the global graph information. Then, for each $v \in R(H)$, we only need to perform a depth-first search with depth of 2 to visit the two-hop neighbours of v and count the number of visits for each v' in the two-hop neighbours of v to determine the batch of pivots, i.e., any v' is not selected as a pivot if its degree equals to the number of its common neighbours with v. Doing that for every vertex in $R(H)$ if necessary, the batch of pivots can be derived in $O(\varsigma(R)|E(H)|)$. In practice, the batch-pivots computation is much faster since 1) the depth of depth-first search is only 2, 2) for most of the subgraphs, $|R(H)|$ is much less than ς, and 3) for any v' that has been excluded as a pivot, we do not need to perform the depth-first search for v'. The extra steps for the order-preserved batch of pivots are omitted for brevity.

6 REDUCING THE SPACE COMPLEXITY

In the previous section, we propose the novel batch-pivots technique. As discussed, to speed up the batch-pivots computation, a local subgraph for each recursive subproblem is stored. If we store the adjacency list for each subgraph, the extra space consumption would be as high as $O(\varsigma(R)|E(G)|)$. In this section, we devise techniques to shave such extra space consumption.

6.1 Optimised data structure

The tricks. The main idea to reduce the space cost is to avoid creating adjacency list and intermediate vertex sets as much as possible. To achieve that, we propose to use global vertex sets L and R, but adjust the locations of the vertices in both L and R to achieve the vertex sets A,B,P,Q that are created in each recursive subproblem in Algorithms 1 and 2. The same idea is applied to the global adjacency list for storing a local subgraph H. However, to ensure the correctness of the algorithm, we have to create intermediate sets so that, when backtracking to a recursive subproblem working on A,B,P,Q and H, the modified L, R and adjacency list can be recovered to the current A,B,P,Q and H. The question is what is the information needed to guarantee the correctness, which will be answered shortly after explicitly defining the data structures.

Vertex sets augmented with location index. For the two vertex sets L and R, location indices loc_L and loc_R are built. Given a vertex $u \in L$, $loc_L(u)$ returns the location of u in L, similar for each $v \in R$.

In Figure 4(a), loc_R shows the location index for R in Figure 1(a). The actual vertices of R are stored in the table R. The arrows between the two tables show the location mapping. For instance, $loc_R(7)=4$ means that vertex 7 is located at the fifth location in R.

Logical partitions for R. For the vertex set R, we partition R into four vertex sets: B,P,Q, and other vertices. Three location indicators i_B, i_P, i_Q store the starting position for B,P,Q respectively. Throughout the algorithm, the following properties are ensured. Vertices from the location i_B to $|R|$ form B. Vertices from the location i_P to i_B-1 form P. Vertices from the location i_Q to i_P-1 form Q. Other vertices are stored from the location 0 to i_Q-1. For instance, in Figure 4(a), using i_Q, i_P and i_B, we can find vertices in Q, P and B. Vertices 5, 9 form Q, vertices 6, 8 form P, and vertex 7 forms B.

Logical partitions for L. We partition L into the vertex set containing common neighbours for all $v \in B$ (i.e. A), and the other vertices. The location indicator i_A stores the starting position storing vertices of A. Throughout the algorithm, A consists of the vertices from the location i_A to $|L|$. For instance, in Figure 4(b), the table L' shows how to use i_A, and A consists of vertices 3, 2.

Logical partitions for adjacency list of R. For the adjacency list of v, denoted by adj_v, we partition it into the vertices contained in A and other vertices. A location indicator i_v stores starting position storing the vertices contained in A. Throughout the algorithm, vertices from the location i_v to $|adj_v|$ are neighbours of u contained in A.

Similarly, for $u \in L$, i.e., adj_u is logically partitioned into neighbours of u contained in $P \cup Q$ and other vertices.

Atomic operations for R. We now discuss the atomic operations frequently used in Algorithm 1 using the above data structure.

Moving v from P to B. Switch v with the vertex v' at the location of i_B-1 if necessary. Update the location information for v and v', and decrease i_B by 1.

Moving v from P to Q. Switch v with the vertex v' at the location of i_P if necessary, adjust the location information for v and v', and increase i_P by 1.

Removing v from Q. Similar to the above one, replace i_P with i_Q.

As shown in Figures 4(a) and 4(b), after moving vertex 8 from P to B, we need to remove vertices 5 and 9 from Q, which makes i_Q in Figure 4(a) increased by 2 to i'_Q shown in Figure 4(b).

Adjusting L and adjacency list are a simplified version of adjusting R. All the atomic operations run in constant time thanks to the location indices.

Besides space saving, the data structure has two advantages below. First, it provides the constant time is-in-a-set operation. Since the location information for R is maintained, and vertices in B, P and Q are in ranges of i_B to $|R|$, i_P to i_B-1 and i_Q to i_P-1, checking if a vertex is in P, Q, or B can be done in constant time via two location comparisons. Second it provides the linear time two-set intersection. Typical two-set intersection operations such as line 15 in Algorithm 1 can be performed in linear time because of the constant time is-in-a-set operation.

Table 2: Datasets and statistics

Dataset	Type	$\|L\|$	$\|R\|$	$\|E\|$	Density	$d_{max}(L)$	$d_{max}(R)$	$d_{2max}(L)$	$d_{2max}(R)$	$\varsigma(L)$	$\varsigma(R)$	#MB
MovieLens	tag–item	16k	7,601	71k	5.7×10^{-4}	641	308	5,817	3,217	640	1,639	140k
BookCrossing	rating	105k	340k	1.14m	3.2×10^{-5}	13k	2,502	5,935	151k	2,501	13.6k	54m
IMDB	association	303k	896k	3.78m	1.4×10^{-5}	1,334	1,590	15,233	15,415	1,589	1,333	5.16m
DBLP	authorship	1.95m	5.62m	12m	1.2×10^{-6}	1,386	287	2,119	7,519	286	1,385	4.89m
LiveJournal	membership	3.20m	7.49m	112m	4.7×10^{-6}	300	1.05m	2.6m	1.05m	1.05m	7,616	6.82b

6.2 Recovery during backtracking

The discussed operations are sufficient to adjust R, L and the adjacency list according to Algorithms 1 and 2. However, for a recursive subproblem, when the algorithm backtracks to this recursive subproblem, it has to recover the logical partitions for L, R and some of the adjacency lists to ensure the correctness. This is because other recursive subproblems rooted from this recursive subproblem have changed the logical partitions. In this section, we study how to recover the logical partitions efficiently.

We first define the logical equivalence for two permutations and then define the adjacency lists which we need to adjust.

Logical equivalence for R. Given two permutations R and R' and location indicators i_B,i_P,i_Q, R and R' have the same logical partitions, if the vertices in R locating from i_B to $|R|$ are the same as the vertices in R' locating from i_B to $|R|$ (the order does not matter) and the same applies for ranges of i_P to i_B-1 and i_Q to i_P-1.

The equivalences for permutations of L and permutations of an adjacency list can be defined similarly.

Adjacency lists to be recovered. Since a recursive subproblem essentially works on an A,P,Q induced subgraph of G, we only need to recover logical partitions for vertices in those sets.

It is trivial to see that, by recovering the logical equivalences for L, R, and the adjacency lists for the vertices discussed above, Algorithm 1 works correctly using the data structure that we devised.

We use Figure 4 to show how the recursive subproblem indicated by Figure 4(a) is recovered when backtracking from the recursive subproblem indicated by Figure 4(b). Using i_Q,i_P,i_B and vertices P={6,8}, we align {6,8} to the locations of i_B-1 to i_P to recover B for R. Since both of them are still within the locations of i_B-1 to i_P, no adjustments are needed. Then, vertices located from i_Q to i_P-1 for R' in Figure 4 (b) are the same as the vertices from i_Q to i_P-1 for R in Figure 4 (a). For L, the permeation of L' in Figure 4(b) is logically the same as L in Figure 4(a).

As a summary, given a recursive subproblem, storing the integers i_P,i_Q,i_B, i_A, a set of integers consisting of i_v for $v\in P\cup Q\cup A$, and the vertices in P is sufficient to recover the logical partitions when backtracking to the recursive subproblem.

Recovery time cost. Given a recursive subproblem with A,B,P,Q, the recovery runs in $O(|P|+|Q|+|A|)$. For the recovery of logical partition of R, we may perform $O(|P|+|Q|)$ atomic operations. Thanks for the location index, each atomic operation takes constant time. The recoveries of logical partition of L and each adjacency list also take constant time since they are the same as recovering B.

7 WRAP-UP AND DISCUSSION

In this section, we show the time and space complexities for Algorithm 2 equipped with our proposed techniques.

Lemma 4. *Algorithm 2 finds runs in* $O(|R|\varsigma(R)|E(G)|2^{\varsigma(R)})$.

Proof sketch. Algorithm 2 is dominated by solving $|R|$ number of MBE problems. For each of the MBE problem, there are up to $2^{\varsigma(R)}$ recursive subproblems. For each recursive subproblem working on H, the time complexity is $O(\varsigma(R)|E(H)|)$, dominated by computing the optimal batch of pivots and $|E(H)|$ is up to $|E(G)|$. □

Correctness. Algorithm 2 systemically checks every set in 2^R to derive all maximal bicliques. It only prunes sets that can form neither bicliques nor maximal bicliques, where the correctness of each pruning has been proven by [47] and our proof for Lemma 3.

Lemma 5. *In terms of polynomial delay, Algorithm 2 runs in* $O(\varsigma(R)|E(G)|\mathcal{B})$.

Proof sketch. To report the first maximal biclique, the dominating computation is the unilateral order running in $O(|E(G)|\ d_{max}(R))$. After that, by Lemma 3, each maximal clique can be reported in $O(\varsigma(R)|E(G)|)$ which is the complexity for a recursive subproblem. Let $\mathcal{B}$ denote the number of maximal bicliques in G, Algorithm 2 runs in $O(\varsigma(R)|E(G)|\mathcal{B})$. □

Lemma 6. *The space complexity of Algorithm 2 is* $O(\varsigma(R)|L|+|G|)$.

Proof sketch. Using the discussed data structure, each recursive subproblem needs $O(|P|+|Q|+|A|)$ space that cannot be freed until backtracking to the recursive subproblem, which can be bounded by $\varsigma(R)+|L|$ using our proposed order. Due to the depth-first nature of the algorithm and our proposed order, the depth is bounded by ς as well. Therefore, storing the information takes $O(\varsigma(R)|L|)$ space. Besides, we maintain order adjustable adjacency list and vertex sets for L and R, which takes $O(|G|)$ space. □

8 EXPERIMENTAL STUDIES

In this section, extensive experiments are conducted to evaluate the effectiveness and efficiency of the proposed techniques.

Implemented algorithms. We first introduce the implemented and evaluated algorithms, apart from our proposed Algorithm 2 denoted by ooMBEA.

iMBEA. We implement iMBEA [47]. We do not use the code [1] (denoted by iMBEA′) since it runs in $O(|R|d^2_{max}(R)2^n)$. We optimise the sorting technique in [47] using bucket sort technique, which makes each sort run in $O(|P|)$. Assuming $|R|\leq|L|$, iMBEA runs on R, which is claimed to be better.

PMBEA. We implement PMBEA [2]. For each recursion, it uses a pre-built index to select a $v\in P$, which serves as a pivot for NCP.

FMBEA. We implement FMBEA [12]. Using Algorithm 2 as a framework, FMBEA has configurations as follows. Assuming $|R|\leq|L|$, it uses R to enumerate maximal bicliques. The order which [12] uses is that vertices in R are in non-increasing order based on degree. FMBEA is a sequential algorithm and uses one single thread as other evaluated algorithms.

Table 3: Overall running time (sec. by default)

Data	iMBEA′	iMBEA	PMBEA	FMBEA	ooMBEA
MovieLens	129.67	17.98	17.21	16.45	4.89
BookCrossing	-	16,773	15,095	11,657	2,872
IMDB	-	38,797	35,693	1,531	67.72
DBLP	-	-	-	1,946	16.23
LiveJournal	48h($<20m$)	48h($<20m$)	48h($<22m$)	48h($<37m$)	48h (6.82b)

Table 4: Space consumption

Data	Edge list	iMBEA	PMBEA	FMBEA	ooMBEA
BookCrossing	14.3MB	91.2MB	87.4MB	98.1MB	31MB
IMDB	35.6MB	320MB	296MB	334MB	81MB
DBLP	183MB	1.23G	1.17G	1.28G	394MB
LiveJournal	1.47G	8.9G	9.1G	9.9G	3.1G

Other algorithms. We also implement variants of our proposed and the above algorithms to evaluate different techniques proposed in this paper. We give the explanations for those variants in the corresponding experiments.

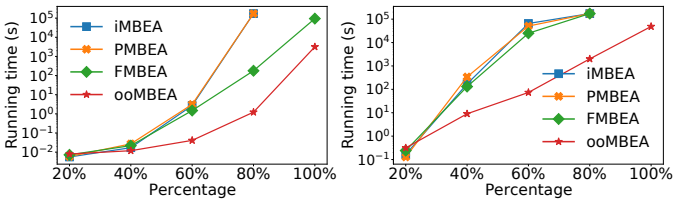
Datasets. We use five real datasets from KONECT [21] for the experimental studies. As shown in Table 2, those five datasets cover five different relationships. Except for *MovieLens*, all the datasets contain not less than 1 million edges. The largest dataset *LiveJournal* contains over 112 million edges. Table 2 also shows that even for the small scale dataset *MovieLens*, the bipartite graph is sparse, i.e., the density of *MovieLens* is 5.7×10^{-4}. For larger datasets, the density is orders of magnitude smaller. For instance, for *LiveJournal*, the density is merely 4.7×10^{-6}.

Measures. We measure the running time of each algorithm. The reported running time is the total CPU time (in seconds abbreviated as *s*), excluding the I/O cost of loading graph and indices from disk to main memory, and a timeout of 48 hours is set, denoted as '-'. All algorithms are implemented in C++. Experiments are conducted on a PC with CPU of Intel i7-12700KF, memory of 32GB DDR5 5600HZ, and Windows 11 (build 22000.493), no less than 20 times if the running time is less than 24 hours (5 times otherwise). The average results are reported.

8.1 Against to Baselines

Before showing results, we would like to indicate that for all the datasets that at least one of the state-of-the-art algorithms can report the number of maximal bicliques, ooMBEA reports the same number of maximal bicliques, which verifies the correctness.

Running time. We report the running times for ooMBEA and the state-of-the-art algorithms in Table 3. Every algorithm adopts the corresponding best configuration. ooMBEA outperforms all the other algorithms. For *DBLP*, iMBEA and PMBEA cannot finish within 48 hours. For the large dataset *LiveJournal*, all the algorithms cannot finish within 48 hours and we report the number of maximal bicliques at the cut-off time. ooMBEA can report over 6 billion maximal bicliques, significantly more than other algorithms. The reasons that ooMBEA performs well on *LiveJournal* are: 1) ooMBEA works on R with $\varsigma(R)$ of 7616, 2) it works on subgraphs tightly bounded by the proposed unilateral order and 3) it has much less polynomial-delay than all the other algorithms. Since iMBEA′ is much slower, we omit it in the following experimental studies.



(a) DBLP **(b) LiveJournal**

Figure 5: Scalability

Space consumption. The space consumptions for iMBEA, PMBEA, FMBEA and ooMBEA are shown in Table 4. ooMBEA needs much less space compared with the state-of-the-art algorithms. The major reasons are as follows. To ensure the claimed time complexity without changing the existing algorithms dramatically, a hash technique is needed [11]. Graphs are stored as adjacency hash sets, and during the recursion, the vertex sets are not stored as hash sets (dynamically creating a hash set is practically expensive). As such, for iMBEA, PMBEA and FMBEA, the space consumptions are high. For ooMBEA, although extra spaces are used to index R and L, those spaces are bounded by much smaller constant factors, which makes ooMBEA consume much less space. This experiment justifies that our proposed techniques make ooMBEA space-effective.

Scalability. For a dataset, we randomly select a fixed percentage of vertices for L and R, and compute their induced subgraph. For each fixed percentage other than 100% in Figure 5, 20 different sets of L and R are randomly selected for generating 20 subgraphs which algorithms work on. The average results for *DBLP* and *LiveJournal* are reported since they are large, and the other datasets have similar results. Due to the hardness of MBE, as the number of vertices in both sides of the bipartite graph increases, the running time of every algorithm increases dramatically. However, the running times for ooMBEA grow significantly slower compared to that for state-of-the-art algorithms, which justifies that ooMBEA can scale to large datasets compared to state-of-the-art algorithms.

Varying density. We show the running times of FMBEA and ooMBEA when varying the density. We fix the number of vertices in bipartite graphs as (5000,5000) while randomly generating 10 bipartite graphs for each bipartite density indicated in Figure 6(a), and we report the average running time of each algorithm for each bipartite density. ooMBEA runs faster than FMBEA when the density is below 0.2. This is because when a bipartite graph is sparse, $\varsigma(R)$ is small, and ooMBEA takes such an opportunity for speeding up MBE. When the density is denser than 0.2, the overheads of the optimisations in ooMBEA cannot bring sufficient speed-up, which makes ooMBEA slower than FMBEA.

8.2 Breaking down evaluations

Effect of vertex set selection. We conduct a set of experiments by switching the set of vertices each algorithm works on, denoted by iMBEAS, PMBEAS, FMBEAS and ooMBEAS. The results are shown in Table 5. Datasets that all algorithms cannot finish within 48 hours are not displayed. Results annotated by * mean that they are derived with algorithms applying Guideline 1.

Table 5: Vary the vertex set for enumeration (sec.)

Data	iMBEAS	PMBEAS	FMBEAS	ooMBEAS
MovieLens	15.52 *	14.83 *	16.32 *	8.89
IMDB	-	139,182	118,098	113
DBLP	-	-	-	50.68

Table 6: Variants of algorithms

Variants	Configuration
oVariant1	ooMBEA with no order optimisation
oVariant2	ooMBEA with no local optimisation
oVariant3	ooMBEA with no space optimisation

For *MovieLens*, iMBEAS, PMBEAS, and FMBEAS work on the vertex set following Guideline 1 in this experiment. They run faster compared to iMBEA, PMBEA, and FMBEA. This is because ς reveals a tighter bound for the search depth when enumerating the power set for L or R and smaller ς indicates less recursion depth. ooMBEAS runs slower than ooMBEA shown in Table 3 as expected. For *IMDB* and *DBLP*, iMBEAS, PMBEAS, FMBEAS work on the vertex set that both their paper and this paper consider to be suboptimal (against Guideline 1). For *IMDB*, all the algorithms run a bit slower than their default version shown in Table 3. This is because for *IMDB*, the difference of $\varsigma(R)$ and $\varsigma(L)$ is trivial, say, 1589 and 1333. For *DBLP*, the difference of $\varsigma(L)$ and $\varsigma(R)$ is more obvious than *IMDB* but much less significant than *BookCrossing*. Only ooMBEAS can finish within 48 hours for *DBLP* but almost forty times slower than ooMBEA. The above results justify that the vertex set selection makes great sense for large real datasets.

Effect of the order. We study how the order affects performance in practice. We use a variant of ooMBEA that removes the order optimisation, denoted by oVariant1. The running time ratios of oVariant1 and ooMBEA for all the datasets are demonstrated in Table 7 column $\frac{\text{oVariant1}}{\text{ooMBEA}}$. Without the unilateral order, oVariant1 runs slower than ooMBEA. As the size of the dataset increases, the ratio increases. For *LiveJournal* dataset, oVariant1 runs over five times slower than ooMBEA. The major reason that the unilateral order improves the performance is that the order ensures the sizes of the subproblems as small as possible, and the largest size is bounded. From the experiment, we can see that the unilateral order plays a critical role for speeding up.

Effect of storing local subgraphs. We study how storing local subgraphs affects the performance. We modify ooMBEA to oVariant2 by removing local graphs stored in each recursion. In Table 7, column $\frac{\text{oVariant2}}{\text{ooMBEA}}$ shows the running time ratios of oVariant2 and ooMBEA for all the datasets. oVariant2 runs 2.11 times slower than ooMBEA for *MovieLens*. As the datasets become large, the running time ratio increases. For *LiveJournal*, oVariant2 is more than 3 times slower than ooMBEA. The reasons that make oVariant2 slow are as follows. For each recursion, due to the lack of local subgraph information, the neighbours of a vertex in the input graph have to be used. This greatly increases the computational cost per recursion.

Effect of space optimisation. How the space optimisation affects performance is studied. A variant of ooMBEA denoted by oVariant3 is implemented to compare with ooMBEA. oVariant3 does not have the adjustable L and R partitioned into A,B,P,Q. It does not have the adjustable adjacency list either for storing local subgraphs for recursions. Instead, it creates local A, B, P, Q and local adjacency list for each recursion. Note that oVariant3 does not have the overhead for recovering the logical partition, which is necessary for ooMBEA. The running time ratios of oVariant3 and ooMBEA for all the datasets are demonstrated in Table 7 column $\frac{\text{oVariant3}}{\text{ooMBEA}}$. Surprisingly, oVariant3 is slower than ooMBEA even with no overheads of logical partition recovery. This is because, without the proposed space optimization techniques, oVariant3 has to apply memory during every recursion. This overhead is non-trivial. Besides, without the space optimisation techniques, the hash technique is necessary to speed up two-set intersection operation. The hash technique is slower than the location check based technique included in the space optimisation techniques. The results of this experiment justify the advantages of using space optimisation techniques.

Table 7: Variants and running time ratio

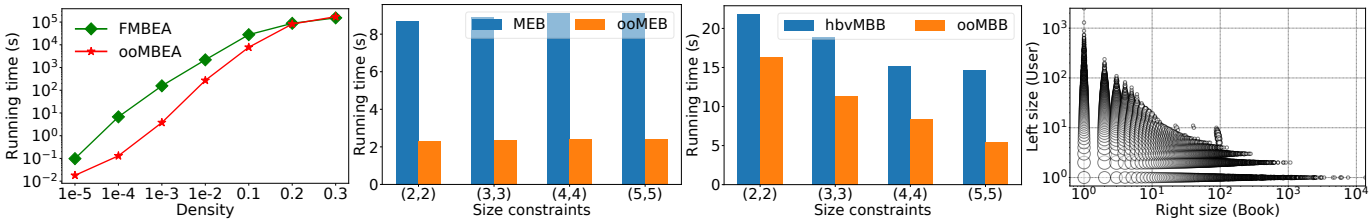
Data	$\frac{oVariant1}{ooMBEA}$	$\frac{oVariant2}{ooMBEA}$	$\frac{oVariant3}{ooMBEA}$
MovieLens	2.51	2.11	1.33
BookCrossing	3.49	2.75	1.47
IMDB	3.55	3.13	1.89
DBLP	4.64	3.21	1.79
LiveJournal	5.366	3.29	1.91

8.3 Case Studies

We conduct three case studies to show that ooMBEA can speed up two popular biclique problems in practice and provides insightful searching hints on *BookCrossing*.

(a,b) MEB search. The (a,b) size constrained MEB problem finds a biclique (A,B) such that $|A|\geq a$, $|B|\geq b$, and $|A\times B|$ is maximised. The state-of-the-art algorithm [30] (MEB) calls iMBEA as a subroutine on subgraphs consisting of vertices that cannot be pruned. Since MEB is dominated by iMBEA, MEB runs in $O^*(2^{|R|})$. We replace iMBEA in MEB to ooMBEA, and denote the new algorithm as ooMEB. The running times for the two algorithms for different size constraints are shown in Figure 6(b). ooMEB runs consistently around four times faster than MEB while still reporting an optimal result. This is mainly because ooMEB has the same pruning techniques as MEB but uses a better enumeration as the subroutine.

(a,b) MBB search. The (a,b) MBB problem finds a biclique (A,B) such that $|A|\geq a$, $|B|\geq b$, $|A|=|B|$ and $|A\times B|$ is maximised. The state-of-the-art algorithm [9], denoted by hbvMBB, uses a very different enumeration compared to MEB due to the balancing constraint, which allows that the enumeration always ends up to a p-time solvable subgraph and makes hbvMBB run in $O^*(1.3803^{\delta})$, where δ is the bidegeneracy of a bipartite graph. We preserve all optimisations in [9] but replace the enumeration and ordering parts with our enumeration and ordering, and the new algorithm is called ooMBB. Notice that since ooMBB does not expand (A,B) in-turn, ooMBB cannot ensure that the enumeration always ends up to p-time solvable subgraphs and therefore runs in $O^*(2^{\varsigma(R)})$. We would like to highlight that δ is derived by the peeling based on 1-hop and 2-hop neighbours, whereas $\varsigma(R)$ is derived by the peeling based on just 2-hop neighbours. Therefore, $\varsigma(R)$ is constant times smaller than δ. Since $1.3803^{\delta}\approx 1.9^{\delta/2}$, the time complexity of ooMBB is comparable with that of hbvMBB. The running times for hbvMBB and ooMBB when varying size constraints are shown in Figure 6(c). ooMBB is faster than hbvMBB while still reporting an optimal result. This



(a) Varying density **(b) MEB search** **(c) MBB search** **(d) Biclique dist.**

Figure 6: Effectiveness evaluations

is because ooMBB inherently avoids generating non-maximal biclique and explores up to $|R|$ subgraphs. In contrast, hbvMBB needs more enumerations to prune non-maximal bicliques, and it may also explore up to $(|L|+|R|)$ subgraphs.

Biclique searching hints. Using ooMBEA, the size distribution of all maximal bicliques in *BookCrossing* can be retrieved and drawn in 2900 seconds. The size distribution is shown in Figure 6(d). A cycle at (x,y) means that there are bicliques with the size (x,y), i.e., y users rate x books in common. The area of the cycle indicates the number of (x,y) bicliques. The larger an area is, the more (x,y) bicliques there are. If a user wants to find a large balanced biclique, Figure 6(d) can provide information that the largest balanced biclique has a size slightly larger than $(10,10)$. As such, the user can set the parameter as $(10,10)$. If a user wants to detect a large set of users (y) rating a larger set of common books $(x,\ x>y)$ for the fraudulent rating detection purpose [30], Figure 6(d) can also provide a useful hint. Figure 6(d) indicates that y cannot be set larger than 10. Otherwise x is too small. The user may set $(10,7)$ for finding $(10,7)$ maximum edge bicliques, which leverages chances for finding bicliques with large vertices in the user set and larger vertices in the book set. Given the fact that *BookCrossing* contains over 1 million edges, the user could try numerous large parameters before finding maximum edge bicliques if there is no statistical information given by Figure 6(d).

9 RELATED WORK

Maximal biclique enumeration. Early maximal biclique enumeration algorithms first build powerset for R, and then combine the sets of the powerset if they lead to a larger biclique [37, 46], which finds all the maximal biclique in a breath-first search manner. Most of the relative recent works solve MBE in depth-first search manner [2, 12, 24, 47]. Apart from the above algorithms, [11] proposes an output-sensitive MBE algorithm, taking exponential space and assuming the degree of the jth highest degree is $1/j^s$ of the highest degree, where $1\leq s\leq 2$, which makes the algorithm run in $O(d_{max}(R)|R|\mathcal{B})$. [17] reduces MBE to maximal clique enumeration. The vertex induced maximal biclique enumeration for unipartite graphs is studied in [18]. This problem is reduced to independent set problem and therefore can be solved as the same time complexity of the maximal independent set enumeration problem, which cannot be applied to bipartite graphs.

Maximum edge biclique. The problem of finding maximum edge biclique (MEB) is NP-hard. In [13], ILP formulations of MEB are proposed, which can find an MEB. An algorithm finding MEB with high probability is proposed [35]. Besides, the MEB problems for special cases of bipartite graphs are studied in [33], [34] and [6]. Recently, a novel exact MEB algorithm [30] is proposed, which can deal with large bipartite graphs. These techniques cannot be used for speeding up MBE since they focus on pruning the search space that does not lead to maximum edge bicliques.

Maximum balanced biclique. The maximum balanced biclique (MBB) problem is NP-hard [16]. Recently, novel exact MBB algorithms [9] are proposed. Subgraphs in which an MBB can be sought in polynomial-time are studied for speeding up the search. However, this technique cannot be applied to MBE since the number of maximal bicliques in those subgraphs could still be exponential w.r.t. the number of vertices of the subgraphs. In [9], a search order is proposed which aims to bound the number of vertices on both of the vertex sets. However, for the MBE problem, the time complexity is exponential w.r.t. just one of the two vertex sets. Our proposed order is more suitable for the MBE problem and has much fewer optimisation overheads.

Other cohesive subgraphs. Bipartite cohesive subgraph models, such as (α,β)-core [28, 41], bi-truss [39, 40], and bi-triangle [42], have drawn great attention recently for different applications. Cohesive subgraph models such as k-core [43, 48], k-truss [8, 10, 19], and densest subgraphs [7, 15], have been studied extensively.

10 CONCLUSION

In this paper, we study the problem of maximal biclique enumeration on real large sparse bipartite graphs. We propose a novel search order called the unilateral order tailored for the maximal biclique enumeration problem, which nicely captures the sparsity of a large sparse bipartite graph. Enumerating maximal biclique using the unilateral order runs in $O^*(2^\varsigma)$, where ς is much smaller than $min\{|L|,|R|\}$ for a real large sparse bipartite graph. We propose the novel batch-pivots technique, which further enhances the proposed algorithm with better polynomial-delay. A novel data structure is devised to reduce the space complexity. Extensive experiments are conducted on large real datasets up to hundreds of million edges to justify the practical performance of our proposed techniques.

ACKNOWLEDGMENTS

This work is jointly supported by the ARC Discovery Projects under Grant No. DP170104747, DP200103700 and DP220102191, the ARC Linkage Project under Grant No. LP180100750, and the National Natural Science Foundation of China under Grant No. 61872258.

REFERENCES

[1] 2017. https://github.com/ddervs/MBEA Online; accessed 12 December 2021.

[2] Aman Abidi, Rui Zhou, Lu Chen, and Chengfei Liu. 2020. Pivot-based Maximal Biclique Enumeration.. In *IJCAI*. 3558–3564. https://doi.org/10.24963/ijcai.2020/492

[3] Andrei Broder, Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, and Janet Wiener. 2011. Graph structure in the web. In *The Structure and Dynamics of Networks*. 183–194. https://doi.org/10.1016/S1389-1286(00)00083-9

[4] Dongbo Bu, Yi Zhao, Lun Cai, Hong Xue, Xiaopeng Zhu, Hongchao Lu, Jingfen Zhang, Shiwei Sun, Lunjiang Ling, Nan Zhang, et al. 2003. Topological structure analysis of the protein–protein interaction network in budding yeast. *Nucleic acids research* 31, 9 (2003), 2443–2450. https://doi.org/10.1093/nar/gkg340

[5] Lijun Chang and Lu Qin. 2018. *Cohesive subgraph computation over large sparse graphs: algorithms, data structures, and programming techniques*. Springer.

[6] Hao Chen and Tian Liu. 2017. Maximum Edge Bicliques in Tree Convex Bipartite Graphs. In *Frontiers in Algorithmics*, Mingyu Xiao and Frances Rosamond (Eds.). Springer International Publishing, Cham, 47–55.

[7] Lu Chen, Chengfei Liu, Kewen Liao, Jianxin Li, and Rui Zhou. 2019. Contextual Community Search Over Large Social Networks. In *ICDE*. 88–99. https://doi.org/10.1109/ICDE.2019.00017

[8] Lu Chen, Chengfei Liu, Rui Zhou, Jianxin Li, Xiaochun Yang, and Bin Wang. 2018. Maximum Co-located Community Search in Large Scale Social Networks. *PVLDB* 11, 10 (2018), 1233–1246. https://doi.org/10.14778/3231751.3231755

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

[10] Lu Chen, Chengfei Liu, Rui Zhou, Jiajie Xu, Jeffrey Xu Yu, and Jianxin Li. 2020. Finding Effective Geo-social Group for Impromptu Activities with Diverse Demands. In *KDD*, Rajesh Gupta, Yan Liu, Jiliang Tang, and B. Aditya Prakash (Eds.). 698–708. https://doi.org/10.1145/3394486.3403114

[11] Peter Damaschke. 2014. Enumerating maximal bicliques in bipartite graphs with favorable degree sequences. *Inform. Process. Lett.* 114, 6 (2014), 317–321.

[12] Apurba Das and Srikanta Tirthapura. 2019. Shared-Memory Parallel Maximal Biclique Enumeration. In *HiPC*. 34–43.

[13] Milind Dawande, Pinar Keskinocak, Jayashankar M Swaminathan, and Sridhar Tayur. 2001. On Bipartite and Multipartite Clique Problems. *J. Algorithms* 41, 2 (Nov. 2001), 388–403.

[14] Rod G Downey and Michael R Fellows. 1995. Fixed-parameter tractability and completeness I: Basic results. *SIAM Journal on computing* 24, 4 (1995), 873–921. https://doi.org/10.1137/S0097539792228228

[15] Yixiang Fang, Kaiqiang Yu, Reynold Cheng, Laks V. S. Lakshmanan, and Xuemin Lin. 2019. Efficient Algorithms for Densest Subgraph Discovery. *PVLDB* 12, 11 (2019), 1719–1732. https://doi.org/10.14778/3342263.3342645

[16] Michael R Garey and David S Johnson. 1990. *Computers and Intractability; A Guide to the Theory of NP-Completeness*.

[17] Alain Gély, Lhouari Nourine, and Bachir Sadi. 2009. Enumeration aspects of maximal cliques and bicliques. *Discrete applied mathematics* 157, 7 (2009), 1447–1459. https://doi.org/10.1016/j.dam.2008.10.010

[18] Danny Hermelin and George Manoussakis. 2021. Efficient enumeration of maximal induced bicliques. *Discrete Applied Mathematics* 303 (2021), 253–261. https://doi.org/10.1016/j.dam.2020.04.034

[19] Xin Huang, Hong Cheng, Lu Qin, Wentao Tian, and Jeffrey Xu Yu. 2014. Querying k-truss community in large and dynamic graphs. In *SIGMOD*. 1311–1322. https://doi.org/10.1145/2588555.2610495

[20] Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew Tomkins. 1999. Trawling the web for emerging cyber-communities. *Computer networks* 31, 11-16 (1999), 1481–1493. https://doi.org/10.1016/S1389-1286(99)00040-7

[21] Jérôme Kunegis. 2013. KONECT – The Koblenz Network Collection. In *WWW*. 1343–1350.

[22] Sune Lehmann, Martin Schwartz, and Lars Kai Hansen. 2008. Biclique communities. *Phys. Rev. E* 78 (Jul 2008), 016108. Issue 1. https://doi.org/10.1103/PhysRevE.78.016108

[23] Haiquan Li, Jinyan Li, and Limsoon Wong. 2006. Discovering motif pairs at interaction sites from protein sequences on a proteome-wide scale. *Bioinformatics* 22, 8 (2006), 989–996. https://doi.org/10.1093/bioinformatics/btl020

[24] Jinyan Li, Guimei Liu, Haiquan Li, and Limsoon Wong. 2007. Maximal biclique subgraphs and closed pattern pairs of the adjacency matrix: A one-to-one correspondence and mining algorithms. *TKDE* 19, 12 (2007), 1625–1637. https://doi.org/10.1109/TKDE.2007.190660

[25] Xiaofan Li, Rui Zhou, Lu Chen, Yong Zhang, Chengfei Liu, Qiang He, and Yun Yang. 2021. Finding a Summary for All Maximal Cliques. In *ICDE*. 1344–1355. https://doi.org/10.1109/ICDE51399.2021.00120

[26] Xiaofan Li, Rui Zhou, Yujun Dai, Lu Chen, Chengfei Liu, Qiang He, and Yun Yang. 2019. Mining Maximal Clique Summary with Effective Sampling. In *ICDM*. 1198–1203. https://doi.org/10.1109/ICDM.2019.00147

[27] Bingkai Lin. 2014. The parameterized complexity of k-biclique. In *SODA*. SIAM, 605–615. https://doi.org/10.1145/3212622

[28] Boge Liu, Long Yuan, Xuemin Lin, Lu Qin, Wenjie Zhang, and Jingren Zhou. 2019. Efficient (α, β)-core computation: An index-based approach. In *WWW*. 1130–1141. https://doi.org/10.1145/3308558.3313522

[29] Guimei Liu, Kelvin Sim, and Jinyan Li. 2006. Efficient mining of large maximal bicliques. In *International Conference on Data Warehousing and Knowledge Discovery*. Springer, 437–448. https://doi.org/10.1007/11823728˙42

[30] Bingqing Lyu, Lu Qin, Xuemin Lin, Ying Zhang, Shengping Qian, and Jingren Zhou. 2020. Maximum Biclique Search at Billion Scale. *PVLDB* 13, 9 (2020), 1359–1372. https://doi.org/10.14778/3397230.3397234

[31] Azam Sheikh Muhammad, Peter Damaschke, and Olof Mogren. 2016. Summarizing online user reviews using bicliques. In *SOFSEM*. Springer, 569–579. https://doi.org/10.1007/978-3-662-49192-8˙46

[32] Tsuyoshi Murata. 2004. Discovery of user communities from web audience measurement data. In *WI*. IEEE, 673–676. https://doi.org/10.1109/WI.2004.10154

[33] Doron Nussbaum, Shuye Pu, Jörg-Rüdiger Sack, Takeaki Uno, and Hamid Zarrabi-Zadeh. 2012. Finding Maximum Edge Bicliques in Convex Bipartite Graphs. *Algorithmica* 64, 2 (2012), 311–325. https://doi.org/10.1007/s00453-010-9486-x

[34] Arti Pandey, Gopika Sharma, and Nivedit Jain. 2020. Maximum Weighted Edge Biclique Problem on Bipartite Graphs. In *Algorithms and Discrete Applied Mathematics*. Springer, 116–128. https://doi.org/10.1007/978-3-030-39219-2˙10

[35] Eran Shaham, Honghai Yu, and Xiao-Li Li. 2016. On finding the maximum edge biclique in a bipartite graph: a subspace clustering approach. In *SDM*. SIAM, 315–323. https://doi.org/10.1137/1.9781611974348.36

[36] Etsuji Tomita, Akira Tanaka, and Haruhisa Takahashi. 2006. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical computer science* 363, 1 (2006), 28–42. https://doi.org/10.1016/j.tcs.2006.06.015

[37] Takeaki Uno, Masashi Kiyomi, and Hiroki Arimura. 2004. LCM ver. 2: Efficient Mining Algorithms for Frequent/Closed/Maximal Itemsets. In *Fimi*, Vol. 126.

[38] Oliver Voggenreiter, Stefan Bleuler, and Wilhelm Gruissem. 2012. Exact biclustering algorithm for the analysis of large gene expression data sets. In *BMC bioinformatics*, Vol. 13. 1–2. https://doi.org/10.1186/1471-2105-13-S18-A10

[39] Kai Wang, Xuemin Lin, Lu Qin, Wenjie Zhang, and Ying Zhang. 2019. Vertex Priority Based Butterfly Counting for Large-scale Bipartite Networks. *PVLDB* (2019). https://doi.org/10.14778/3339490.3339497

[40] Kai Wang, Xuemin Lin, Lu Qin, Wenjie Zhang, and Ying Zhang. 2020. Efficient bitruss decomposition for large-scale bipartite graphs. In *ICDE*. IEEE, 661–672. https://doi.org/10.1109/ICDE48307.2020.00063

[41] 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 *ICDE*. IEEE, 85–96. https://doi.org/10.1109/ICDE51399.2021.00015

[42] Yixing Yang, Yixiang Fang, Maria E Orlowska, Wenjie Zhang, and Xuemin Lin. 2021. Efficient bi-triangle counting for large bipartite networks. *PVLDB* 14, 6 (2021), 984–996. https://doi.org/10.14778/3447689.3447702

[43] Kai Yao and Lijun Chang. 2021. Efficient Size-Bounded Community Search over Large Networks. *PVLDB* 14, 8 (2021), 1441–1453. http://www.vldb.org/pvldb/vol14/p1441-yao.pdf

[44] Sungroh Yoon, Luca Benini, and Giovanni De Micheli. 2007. Co-clustering: a versatile tool for data analysis in biomedical informatics. *IEEE Trans. Inf. Technol. Biomed* 11, 4 (2007), 493–494. https://doi.org/10.1109/TITB.2007.897575

[45] Ryo Yoshinaka. 2011. Towards dual approaches for learning context-free grammars based on syntactic concept lattices. In *International Conference on Developments in Language Theory*. Springer, 429–440.

[46] Mohammed J Zaki and Ching-Jui Hsiao. 2002. CHARM: An efficient algorithm for closed itemset mining. In *SDM*. SIAM, 457–473. https://doi.org/10.1137/1.9781611972726.27

[47] Yun Zhang, Charles A Phillips, Gary L Rogers, Erich J Baker, Elissa J Chesler, and Michael A Langston. 2014. On finding bicliques in bipartite graphs: a novel algorithm and its application to the integration of diverse biological data types. *BMC bioinformatics* 15, 1 (2014), 1–18. https://doi.org/10.1186/1471-2105-15-110

[48] Yikai Zhang, Jeffrey Xu Yu, Ying Zhang, and Lu Qin. 2017. A Fast Order-Based Approach for Core Maintenance. In *ICDE*. 337–348. https://doi.org/10.1109/ICDE.2017.93

[49] 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. https://doi.org/10.1016/j.ejor.2018.03.010