Efficient and Effective Cohesive Subgraph Search in Bipartite Graphs

by

Aman Abidi

A thesis submitted in fulfilment of the requirements
for the degree of Doctor of Philosophy

at the

DEPARTMENT OF COMPUTING TECHNOLOGIES
SCHOOL OF SCIENCE, COMPUTING AND ENGINEERING TECHNOLOGIES
SWINBURNE UNIVERSITY OF TECHNOLOGY
MELBOURNE, AUSTRALIA

June, 2022

Declaration of Authorship

I, Aman Abidi, declare that this thesis titled, "Efficient and Effective Cohesive Subgraph Search in Bipartite Graphs" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have quoted from the work of others, the source is always given. Except for such quotations, this thesis is entirely my work.
- I have acknowledged all main sources of help.

Signed: 

Date: 16/06/2022

Abstract

Graph mining has gained much attention from researchers in the last few decades and is one of the novel approaches for mining large datasets. In particular, a bipartite graph is an interesting structure that can model the relationships between two different types of entities in real-world networks such as customer-product, word-document, and author-paper networks. Finding bipartite cohesive subgraphs is a fundamental problem that aims to find strongly connected subgraphs to understand such bipartite graphs better. Recently, many real-world applications require bipartite cohesive subgraphs that satisfy some constraints. Despite the power of traditional models in searching strongly connected patterns, using these models on bipartite graphs cannot fully explore the deeper insights as they are defined on general graphs. Therefore, in this thesis, we first study the fundamental problem of enumerating all bipartite cohesive subgraphs and then present two new bipartite cohesive subgraph models for distinct real-world application scenarios.

Firstly, the problem of enumerating all maximal bicliques in a bipartite graph is an important and well-known problem in data mining. Although research has been conducted on this problem, surprisingly, we find that pivot-based search space pruning, which is effective in clique enumeration, has not been exploited in the biclique scenario. Therefore, we explore the pivot pruning for biclique enumeration. We propose an algorithm for implementing the pivot pruning,

powered by an effective index structure Containment Directed Acyclic Graph ($CDAG$). Meanwhile, existing literature indicates contradictory findings on the order of vertex selection in biclique enumeration. As such, we re-examine the problem and suggest an offline ordering of vertices that expedite the pruning. We conduct an extensive performance study to demonstrate the efficiency and effectiveness of our proposed algorithm using real-world datasets from a wide range of domains.

Secondly, we observe that, for some applications, one is interested in finding bipartite cohesive subgraphs containing a specific vertex rather than finding all bipartite cohesive subgraphs. We study a new query-dependent bipartite cohesive subgraph search problem based on k-wing model, named as the personalized k-wing search problem. We introduce a k-wing equivalence relationship to summarize the edges of a bipartite graph G into groups. Therefore, all the edges of G are segregated into different groups, i.e., *k-wing equivalence class*, forming an efficient and wing number conserving index called *EquiWing*. Further, we propose a more compact version of *EquiWing*, *EquiWing-Comp*, which is achieved by integrating our proposed *k-butterfly loose* approach and discovering hierarchy properties. These indices are used to expedite the personalized k-wing search with a non-repetitive access to G, which leads to linear algorithms for searching the personalized k-wing. Moreover, we conduct a thorough study on the maintenance of the proposed indices for evolving bipartite graphs. We discover novel properties that help us localize the scope of the maintenance at a low cost. By exploiting the discoveries, we propose novel algorithms for maintaining the two indices, which substantially reduces the cost of maintenance. We perform extensive experimental studies in real, large-scale graphs to validate the efficiency and effectiveness of *EquiWing* and *EquiWing-Comp*.

Thirdly, a wide range of applications need less overlapping bicliques with

specific size constraints instead of enumerating all the maximal bicliques. We study a new biclique problem, called the top-k t-biclique coverage problem. A t-biclique is a biclique with a minimum size constraint t for one vertex set and the problem aims to find top-k t-bicliques maximizing the coverage on the other vertex set. The top-k t-biclique coverage problem has novel applications such as finding top-k courses while maximising student engagement. We prove that this problem is NP-hard. A straightforward way to address the problem first needs to enumerate and store all t-bicliques and then greedily select k promising t-bicliques, leading to an approximate guarantee on the coverage. However, it takes exponential space, which is impractical. We then apply a fast approximation scheme to solve this problem, which shaves the exponential space consumption by progressively updating top-k results. Observing that the fast approximation algorithm takes too much time on updating the results due to the coverage being computed from scratch for each update, an online index is devised to address the drawback. Due to the hardness of the problem, even the fast approximation algorithm cannot scale to a large dataset. To devise a scalable solution, we then propose a heuristic algorithm running in polynomial time. Thanks for the four carefully designed heuristic rules, the heuristic algorithm can find large coverage top-k t-bicliques extremely fast for large datasets. Apart from that, the heuristic result with large coverage can effectively prune unpromising enumerations in the fast greedy algorithm, which improves the efficiency of the fast approximation algorithm without compromising the approximation ratio. Extensive experiments are conducted on real datasets to justify the effectiveness and efficiency of the proposed algorithms.

Publications

- **Aman Abidi**, Rui Zhou, Lu Chen and Chengfei Liu. Pivot-based Maximal Biclique Enumeration. IJCAI-20, CORE rank A*. (Chapter 3, DOI : https://doi.org/10.24963/ijcai.2020/492)
- **Aman Abidi**, Lu Chen, Rui Zhou and Chengfei Liu. Searching Personalized k-wing in Bipartite Graphs. IEEE Transactions on Knowledge and Data Engineering, CORE rank A*. (Chapter 4, revision submitted)
- **Aman Abidi**, Lu Chen, Chengfei Liu and Rui Zhou. On Maximizing the Vertex Coverage for Top-k t-Bicliques in Bipartite Graphs. International Conference on Data Engineering 2022, CORE rank A*. (Chapter 5, DOI : 10.1109/ICDE53745.2022.00221)

Acknowledgments

I want to thank Almighty Allah (SWT) for completing my doctoral research and guiding me with wisdom, strength, and perseverance throughout my research journey.

I am incredibly grateful to my principal supervisor Prof. Chengfei Liu from the School of Software and Electrical Engineering, Department of Computer Science and Software Engineering at the Swinburne University of Technology. He is a great mentor who will always remain one of the best human beings that I have ever met in my life. He has guided me in my research, and I could not have completed my research without his support. He has always provided constructive criticism and constant nurturing whenever I stray from the right direction. He was always available whenever and wherever I needed his guidance. I am also grateful for his kind gesture of supporting me during the tough times of the pandemic. I am thankful that I had the chance to work under him.

I want to thank my associate supervisor Dr. Rui Zhou. He has helped me jump-start my PhD research and pointed me in the right direction to discover my research problems. I also would like to sincerely thank Dr. Lu Chen for his help and collaboration. He has been like a big brother to me who not only understood my limitations but constantly supported me in overcoming them. He has always been invested in my PhD research's ups and downs. He worked as the link between my supervisors and me and has guided me like one. I have

learned many things from them that were not confined to my research work but included many other life perspectives to become a strong person.

I would like to thank all members of my research group at Swinburne, Dr. Musfique Anwar, Dr. Mehdi Naseriparsa, Dr. Ahmed Al-Shammari, Dr. Afzal Azeem Chowdhary, Dr. Zafaryab Rasool, Xiaofan Li, Limeng Zhang and Yunfei Li. I want to mention Dr. Tarique Anwar and Dr. Quamar Niyaz, who motivated me to pursue research.

I am grateful to my parents, Irfan Abidi and Khushnuma Rizvi, for their unconditional support, prayers and love. They always stood by me at any cost whenever I was in challenging situations and encouraged me with their loving spirit. I am forever indebted for their sacrifices made for me, known and unknown to me, making me who I am today. I am also thankful to my younger brothers Yaman and Abbas for their encouragement. I love them from the bottom of my heart and could not have done it without them.

I want to thank my colleagues, friends and housemates, Dr. Afzal Azeem Chowdhary and Dr. Zafaryab Rasool, for being there through my ups and downs throughout my PhD journey and treating me like their younger brother. I want to mention Qambar Hasan, a dynamic personality who has taught me how to tackle real-world situations and help others with my knowledge. I would also like to thank my colleague and a wonderful friend Kaneez Fizza, who has always supported me during my research.

I am very appreciative to all my friends in Melbourne Mir Baqar, Askari Hasan, Md Aqib, Akshay Sharma, Dr. Amey Anupama, Ashok Meghwal, Dr. Krishna Prasad, Mirza Irfan Baig, Abhishek Pandey, Dr. Samira Moukannaa, Gokul Sidarth Thirunavukkarasu, Dr. Mst Farhana Diba, Dr. Suneeti Purohit, Shalmoly Mandol and Samra Irshad. To my friends back home for their prayers, love, and care. Taha Sheikh, Sameer Khan, Mohammad Shazan, Alya Rizvi,

Moiz Suhail, Adil Moid, Kausar Khan, Raza Abid, Zain Raza, Aun Murtuza, Saif Mehndi, Zafar Haider, Mohd Tahir, Tapan Upadhyay, Nazish Zaidi, Adil Ali, Rishabh Singh and Ayush Varshney.

I take this opportunity to sincerely acknowledge the Swinburne University of Technology for providing me with the Tuition Fees Scholarship (TFS) and the Australian Research Council (ARC), which supported my research and living expenses in Australia. My special thanks also go to the IT technical and faculty support office for their help in maintaining a good working environment in Swinburne. Last but not least, to the teachers of my Alma maters, Unity College and Aligarh Muslim University, who nurtured me and encouraged me to pursue research in the first place.

Contents

List of Figures

List of Tables

List of Algorithms

I want to dedicate this research to Almighty Allah, the Imam of my time Hazrat Mahdi (A.S.), my late grandmother (*Nani*) and my parents.

Chapter 1

Introduction

Graph mining has gained significant attention from researchers in the last few decades and is one of the novel approaches for mining large datasets [1]. The primary objective of graph mining is to discover the undisclosed and proficient patterns from these massive graphs. Among the numerous patterns and properties of graphs, cohesiveness is one property that has been studied a lot. Over the period of time, substantial success has been achieved in developing techniques for analysis of the cohesive groups in large graphs. Many cohesive subgraph models like k-cliques[2], k-truss[3], k-plex[4], k-core[5], etc. have been proposed. However, today's interlinked data from sources such as the web and online social networks has led to massive heterogeneous graphs. These heterogeneous graphs are represented by k-partite graphs. Among all the k-partite graphs, *bipartite graphs*, i.e., 2-partite graphs, have received significant attention because of its application to model the relationships between two different types of entities across various domains such as customer-product [6], gene co-expression [7], word-document [8] and author-paper [9] networks. Finding bipartite cohesive subgraphs has many applications, such as spam group detection in the web [10], generating probabilistic models for protein-protein interaction

network analysis [11], association rule mining from transactional databases [12] and sponsored advertisement [13]. Despite the power of traditional models in searching strongly connected patterns, using these models on bipartite graphs cannot fully explore the deeper insights as they are defined on general graphs [14]. Consequently, various bipartite cohesive subgraph models have been extensively studied, such as (α, β)-*core* [15], k-*bitruss* [16], k-*wing* [17], *biclique* [18] etc. Recently, many real-world applications require bipartite subgraphs that satisfy some constraints such as personalized, strongly connected and size constraint bipartite subgraphs. However, the conventional bipartite subgraph models cannot cater for such applications. Therefore, in this thesis, we first study the fundamental problem of enumerating all bipartite cohesive subgraphs, where we introduce the classical pruning techniques to improve the enumeration algorithm. Then we present two new bipartite cohesive subgraph models for distinct real-world application scenarios. In our research, we utilize the already discovered strongly connected bipartite structures, i.e., *biclique* and k-*wing*. One of the significant real world application scenarios for such strongly connected bipartite structures is *Anomaly detection* [19], [20]. Many online shopping platforms, such as Amazon and eBay, are prone to be attacked by fake reviews and comments to improve one's online sales or reduce competitors' sales. There is a high probability that such reviews and comments are caused via a large group of customers for a set of products to compliment/defame the rankings of their businesses selling their own or competitor's products. A biclique or k-wing can model such groups effectively in a bipartite graph. Consequently, during our research, we address the three interesting and challenging problems in the bipartite graphs: (1) Pivot-based maximal biclique enumeration; (2) Searching personalized k-wing in bipartite graphs; (3) On maximizing the vertex coverage for top-k t-bicliques in bipartite graphs.

Section 1.1 provides the background and motivations for investigating the above problems and explains the challenges these problems face. We summarize our contributions to address the challenges in our research problems in Section 1.2. Section 1.3 details the organization of the remaining thesis.

1.1 Motivations

In this section, we discuss the motivations for studying our research problems. We consider an input bipartite graph $G = (U, V, E)$ where U and V are the two disjoint vertex sets and $E \subseteq U \times V$ is the set of edges.

1.1.1 Pivot-based Maximal Biclique Enumeration

A biclique $B = (X \cup Y)$, $X \subseteq U$, $Y \subseteq V$ is a complete subgraph of a given bipartite graph G such that all the vertices of X are connected to all the vertices of Y. The maximal biclique is the one, which is not a proper subgraph of any other biclique in G. Consider an example in a *Youtube* network containing a relationship between users and groups. A biclique in this scenario consists of set of users and the set of groups which contain all the corresponding users. Exploring such biclique of users who share a common interest can be proved valuable for analyzing the behaviour and speculate the actions of a user in the network. Generally, identifying *maximal bicliques* proves to be useful, as they are not contained by any other bicliques. Therefore, we study the problem of *Maximal biclique enumeration* $\{MBE\}$ in a bipartite graph.

The domain of real-world applications of MBE includes community detection, bioinformatics, closed item sets, etc. Some typical applications include detecting overlapping communities in *Noordin Top Terrorist Network* for finding more dangerous elements [21], discovering large dense graphs in mas-

sive graphs for spam group detection [22], generating probabilistic model for protein-protein interaction network analysis [11], extracting gene-phenotype information from transactional databases [23], constructing the optimal phylogenetic tree [24] and gene expression [7]. From the domain of closed item sets, MBE is used in association rule mining from transactional databases [12]. Some of the other applications include reducing the role mining complexity in role-based access control system [25], learning context-free grammars [26] and providing a model with constraints for solving chemical process scheduling problems [27].

Currently there are few algorithms which address the exact problem of enumerating all maximal bicliques in a bipartite graph. The previous algorithms were unable to tackle some challenges associated with the biclique enumeration, which are:

1. *Inefficient pruning technique for the large search space*: the state-of-the-art algorithms iMBEA [28] and LCM-MBC [29] have deployed pruning techniques to reduce the search space to some extent. However, the resulting search space is still quite large and exacerbates the cost for enumeration.

2. *Unavailability of any index based structure*: surprisingly, there is no offline structure that has been proposed to improve the enumeration.

3. *Inefficient ordering of vertices*: ordering of vertices plays a crucial role in enumeration of search space [28], [30]. Although, the ordering proposed improved the overall enumeration time, yet an extra overhead of local ordering was introduced.

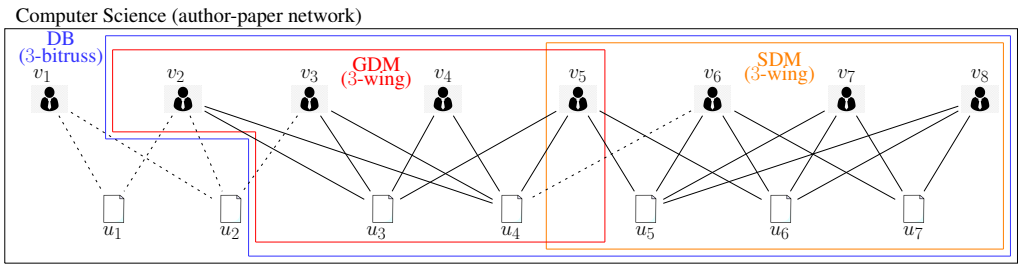


Figure 1.1: *k-bitruss*, *k-wing* and *personalized k-wing*

1.1.2 Searching Personalized k-wing in Bipartite Graphs

Finding bipartite cohesive subgraphs has rich applications such as word and document clustering [8], spam group detection in the web [31], and sponsored advertisement [13]. The objective of finding all the bipartite cohesive subgraphs is to aim at the entire bipartite graph and generally apply a global criterion to provide macroscopic information.

Existing models. To cater for the applications discussed above, various bipartite cohesive subgraph models have been studied such as (α, β)-*core* [15], *k-bitruss* [16], *k-wing* [17], etc. Butterfly (i.e., a 2×2 complete bipartite subgraph) is the basic building block in a bipartite graph and has drawn great attention recently [16], [17], [32]–[35]. It is analogous to a triangle (a smallest clique and basic building block of cohesive subgraphs) in a unipartite graph and serves as the smallest unit of cohesion [33], [34]. As such, *k-bitruss* and *k-wing* subgraphs which are based on butterfly ensure more cohesiveness and are preferred over (α, β)-*core* which is not based on butterfly. A *k-bitruss* is a bipartite subgraph which satisfies the minimum k-butterfly constraint defining that every edge in the bipartite subgraph participates in at least k butterflies and the bipartite subgraph is non-extensible. A *k-wing* strengthens a *k-bitruss* by adding a butterfly connectivity constraint defined as follows. Any two edges in a *k-wing* subgraph are either in a butterfly or can be reached by a set of edge-overlapping

butterflies. The rationale is that, a butterfly represents the strong and stable relationship among two pairs of vertices. If any two edges in a k-*bitruss* subgraph are also butterfly connected, this k-*bitruss* subgraph must be strongly connected and have a more cohesive structure among all involved vertices. This allows a vertex to participate in multiple bipartite cohesive subgraphs. To have a better understanding of the advantage of the butterfly connectivity constraint, we use the following example to explain.

Example 1.1. *Fig. 1.1 represents an author-paper network from the domain of Computer Science. Finding bipartite cohesive subgraphs using k-wing would allow us to quickly discover the authors who can potentially work together in groups. Fig. 1.1 contains a single 3-bitruss (blue) and two 3-wings (red and orange), consisting of edges represented with solid lines. The 3-bitruss can only discover a single bipartite cohesive subgraph (Database group (DB)), however, it is incapable to distinguish authors from two connected cohesive groups, e.g., v_4 and v_6 (not butterfly connected) are both considered within the broad DB research group. On the contrary, a k-wing model strengthens via butterfly connectivity leading to more strongly (butterfly) connected bipartite subgraphs, e.g., v_4 belongs to Graph Data Management group (GDM (3-wing)) and v_6 to Spatial Data Management group (SDM (3-wing)).*

Personalized bipartite cohesive subgraph. Distinct vertices in a bipartite graph may have distinct properties, which requires microscopic analysis, i.e., personalized. The motivation for personalization can be understood from the following example, Fig. 1.1 represents an author-paper network (DBLP), we demonstrate how personalized bipartite cohesive subgraph using k-*wing* model can help authors, for instance v_8 and v_5, discover their potential future collaborators in Fig. 1.1, i.e., within their respective 3-*wings*. In contrast, if we omit the personalization, the search for authors v_8 and v_5 would get the same re-

sult consisting of all the 3-*wings* in Fig. 1.1. The wide range of applications for a personalized bipartite cohesive subgraph search includes finding exclusive users for crowdfunding the user's project [36], author's collaborations and project fundings [37], [38], effective multilingual packaging [39], fake review detection [17], customized recommendation [40], providing stable evolution to the younger researchers [41], etc.

In this problem, we study *personalized k-wing search* based on the k-*wing* model. Given a bipartite graph $G(U, V, E)$, a query vertex $q \in U \cup V$ and an integer k, the personalized k-wing search returns all the k-*wings*. The problem of personalized k-wing search can be addressed by adapting the algorithm proposed for finding all the k-*wings* [17], which serves as our baseline. The idea is, we first compute the k-*bitrusses* using the state-of-the-art bitruss algorithm [16]. Then, among the k-*bitrusses*, we further explore within the k-*bitruss* containing q to form the personalized k-*wings* by grouping edges in the k-*bitruss* containing q together if the edges are butterfly connected. However, even if we ignore the cost for computing the k-*bitruss*, the baseline still has high time complexity, i.e., $\mathcal{O}(d_{max}^2|E|)$, where d_{max} is the maximum degree of G and $|E|$ is the number of edges in the personalized k-*wings*.

Since, in many real-world applications, such as online social networks [42], web graphs [43] and collaboration networks [44], bipartite graphs are evolving where vertices/edges will be inserted/deleted over time dynamically. Hence, we also examine *personalized k-wing search* in dynamic bipartite graphs. The insertion of an edge may affect the other edges by leading them to participate in a new k'-*wing* cohesive subgraph ($k' \neq k$). Unlike the case of k-*truss* [45] (the counterpart of k-*wing* for general graphs), where an edge could move to $(k+1)$-truss subgraph after insertion, the edge in a k-*wing* could move to *$(k+i)$-wing* subgraph after the insertion. Here, i could be greater than 1, which brings new

challenges for k-*wing* maintenance. A tight upper bound needs to be derived to estimate the $(k + i)$-*wing* subgraph that an edge could move to after insertion so that we can effectively identify the affected region with low cost. Similarly, we need to derive a tight lower bound for the case of deletion.

1.1.3 On Maximizing the Vertex Coverage for Top-k t-Bicliques

Enumeration of of all biclique based cohesive subgraph structures in a bipartite graph is a fundamental mining problem and has been well studied in the literature [29], [46]–[54] and [55]. However, in many real applications, finding all maximal bicliques could be less interesting. First, an application may have a specific size requirement on the two vertex sets of a biclique instead of an arbitrary sized maximal biclique. Second, an application may just need k promising results with required constraints, studied similarly for other subgraph structures [56]–[59]. Let us consider two real application scenarios below.

Creating k courses with t units for a department to maximize student engagement. With the advancement in the field of education, a large number of new units are being introduced across departments of an institution, which leads to demands of forming new courses. A course may consist of t units. Given a student-like-unit bipartite graph. The fact that a set of students who like all t-unit shows the potential of the t units together as a course. Clearly, there are many options for creating courses. Given the resource limitations of a department, finding top-k courses that most students like is more interesting and suitable for this application scenario. It helps departments update their curriculum for engaging more student.

Hiring collaborative research groups for a project. The unprecedented growth of research projects addressing the real world problems has gained immense significance. The successful completion of these projects is crucial due to the

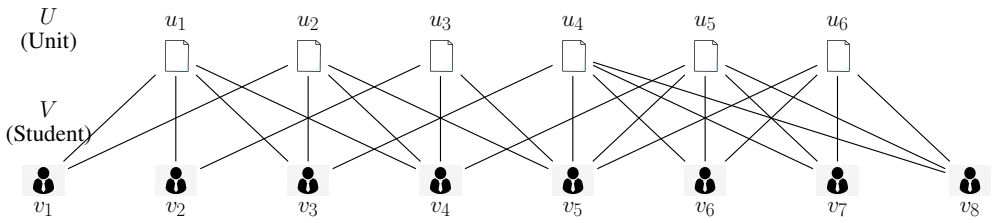


Figure 1.2: Bipartite graph representing student-like-unit network.

huge capital investment. Normally, a research group needs t researchers. The fact that t researchers expertise in common topics can reflect their success of a collaboration. Apart from that, a successful project also requires to engulf the range of research topics for such collaborative groups, so that all the objectives of the projects are addressed. Therefore, finding top-k such groups maximising topic coverage in a researcher-expertise-topic bipartite graph is expected for this application scenario.

As far as we know, none of existing variants of biclique models can handle the above requirements. In this problem, we propose the top-k t-biclique coverage problem to fill in this gap. We first allow users providing an integer t to specify a size constraint on one vertex set of a biclique, denoted as t-biclique. Then, top-k such t-bicliques maximising the vertex coverage on the other vertex set, serve as the optimum solution of the top-k t-biclique coverage problem. Below, we show how the proposed problem can be applied to the first application scenario mentioned above.

Example 1.2. *Figure 1.2 represents a student-like-unit bipartite graph. Suppose we need* 3 *courses with* 2 *units which can engage the maximum number of students. We put a size constraint* 2 *on the unit vertex set* (U)*, and find top-*3 2*-bicliques maximising the vertex coverage on the student vertex set* (V)*. The optimum results are the three* 2*-bicliques:* $\{(u_1, u_2), (v_1, v_3, v_4)\}$*,* $\{(u_1, u_3), (v_2, v_4)\}$*, and* $\{(u_4, u_5), (v_5, v_6, v_7, v_8)\}$*, covering* 8 *students. This ensures that the most students are engaged in these courses. In contrast, if we*

solve this problem using maximal biclique enumeration with size constraint 2 *for maximising the overall coverage, the corresponding result contains* $\{(u_1, u_2), (v_1, v_3, v_4)\}$, $\{(u_1, u_3), (v_2, v_4)\}$, *and* $\{(u_2, u_4), (v_3, v_5)\}$, *covering only* 5 *students. We can observe that the* 2*-biclique* $\{(u_4, u_5), (v_5, v_6, v_7, v_8)\}$ *is not included in the top-*k *maximal bicliques as it is not maximal and its corresponding maximal biclique* $\{(u_4, u_5, u_6), (v_5, v_6, v_7, v_8)\}$ *cannot satisfy the size constraint. Consequently, it is less optimum compared to the result of our proposed research problem.*

Due to NP-hardness of the problem, finding an exact solution for this problem is infeasible for large datasets. Therefore we consider approximation algorithms. A straightforward approximation algorithm is to first enumerate and store all t-bicliques, and then select k t-bicliques with the maximum coverage greedily, ensuring an approximation ratio w.r.t. coverage. However, there are still three challenges to be addressed.

1. *Exponential space complexity* - the greedy approach has to store all the enumerated t-bicliques, and then utilizes the algorithm for k-cover problem [60], which is impractical for large datasets as the number of t-bicliques requires exponential space.

2. *How to reduce unpromising* t*-biclique enumeration* - for any two t-bicliques with large individual coverage, they may collectively cover very few extra vertices. As such, it is very likely that only one of them will be in the top-k result. How to effectively avoid enumerating the other one is indeed challenging. On the other hand, some t-bicliques inherently have small coverage. How to avoid enumerating such small coverage t-bicliques is also challenging.

3. *Hardness of the problem* - although the greedy approaches quickly pro-

vide a guaranteed result for the problem. The worst case time complexity still remains exponential and could be impractical for large real datasets.

1.2 Contributions

1.2.1 Pivot-based Maximal Biclique Enumeration

We aim to address all the previous three challenges in maximal biclique enumeration. We propose an algorithm that can efficiently prune the search space using an index structure, without any overhead of extra computation during the enumeration. We propose Pivot based Maximal Biclique Enumeration (PMBE) algorithm for all maximal biclique enumeration. PMBE integrates *pivot pruning* technique with the LCM-MBC for reducing the search space. An efficient implementation of the pivot pruning is guaranteed by our proposed *containment directed acyclic graph* index structure *(CDAG)*. The semantics of ordering of vertices to enumerate maximal biclique is also studied, and a heuristic ordering is proposed for the heuristic pivot selection, which eventually optimizes the algorithm. Our major contributions are listed as below:

1. *Optimized pivot-based pruning algorithm for enumerating all maximal bicliques*: We present an optimized algorithm PMBE, which introduces the classical pivot technique for expediting the search space pruning, for the enumeration of all maximal bicliques.

2. *Efficient index structure for containment relationship*: We utilize an efficient offline structure called containment directed acyclic graph (CDAG), for systematizing pivot pruning.

3. *Rev-Topological order for heuristic pivot selection*: We study semantics and analyze the ordering of vertices in biclique generation, and suggest

an offline ordering *rev-topological* for efficient pivot pruning.

4. *Extensive performance studies across distinct real-world networks*: We conduct extensive experiments on 10 real-world datasets to evaluate the efficiency of our proposed algorithms.

1.2.2 Searching Personalized k-wing in Bipartite Graphs

To speed up the personalized k-wing search for real-world applications which may have an extensive number of queries, we first present a novel index, called *EquiWing-Graph*, based on k-wing equivalence adapted from truss equivalence. Using the proposed index, a personalized *k-wing* query can be processed in linear time w.r.t. the size of the result. The core idea of the index is the introduction of the *k-wing equivalent edge sets* of a bipartite graph. For a given bipartite graph G, two edges are *k-butterfly reachable*, if they participate in the same butterfly or are connected by a series of dense butterflies. We group the edges into different *equivalent edge sets* if edges are *k-butterfly reachable* and show that edges in the same group have *equivalent relationship*. *EquiWing-Graph* is a super graph based index which is composed of super nodes containing groups of edges having equivalent relationship. We further study the properties for merging super nodes and aligning the merged super nodes hierarchically, which immediately leads to a novel tree based structure, named as *EquiWing-Tree*. Compared to *EquiWing-Graph*, *EquiWing-Tree* takes less space and is faster for processing personalized k-wing queries thanks for the compact tree structure.

We summarize our contributions as follows:

1. *Effective bipartite cohesive subgraph model* - we propose the personalized *k-wing* model which ensures high cohesiveness.

2. *EquiWing-Graph* - we present the *k-wing equivalence* inspired by *truss*

equivalence [61] to construct the index. It summarizes the input bipartite graph into a super graph and is self-sufficient, which guarantees the personalized k-*wings* can be found in linear time.

3. *EquiWing-Tree* - we propose the k-*butterfly loose connectivity* to compress *EquiWing-Graph* and use the hierarchy property of the k-*wing* to expedite the query processing.

4. *Efficient maintenance of the indices* - we provide an efficient incremental algorithm to maintain *EquiWing-Graph* and *EquiWing-Tree* for dynamic bipartite graphs by avoiding unnecessary recomputation.

5. *Extensive experimental analysis* - we carry out extensive experimental analysis of both the indexing schemes and a baseline algorithm across 6 real-world datasets.

1.2.3 On Maximizing the Vertex Coverage for Top-k t-Bicliques

To address the challenges for this problem, we first adapt a fast approximation scheme. It progressively generates and maintains top-k t-bicliques while enumerating them, which naturally avoids storing all t-bicliques and shaves exponential space cost. To effectively increase the coverage of the maintained t-bicliques, a new generated t-biclique will replace a t-biclique in the maintained ones if and only if such replacement can bring a significant coverage improvement, which ensures the approximation guarantee. Noticing that checking such a replacement condition induces non-omissible computational cost, an online index is devised, which makes such cost non-dominating. Because of considering all the maximal t-bicliques, the fast greedy algorithm may not scale to large datasets. We therefore propose a heuristic algorithm, in which four carefully designed heuristic rules are proposed to find top-k t-bicliques with good

coverage for large datasets in polynomial time. Apart from using the heuristic algorithm standalone, we can also use its result as a lower bound for the fast greedy algorithm to further avoid enumerating non-promising t-bicliques by an upper bound based pruning as follows. For a vertex u, an upper bound of the coverage for any t-biclique containing u is derived by using our proposed r_t-core number, where the r_t-core number of u is the largest possible integer r such that there is a subgraph that 1) every vertex in the vertex set containing u has a degree no less than r, and 2) every vertex in the other vertex set has a degree no less than t. Then, based on the maintained top-k t-bicliques, vertices such that their upper bound coverage still cannot beat the maintained ones will be pruned. Equipped with the upper bound based pruning, the fast greedy algorithm can scale to reasonably large datasets.

We summarize our principal contributions as follows:

- *New problem* - we study and formulate a new problem of enumerating top-k t-bicliques, which maximizes the coverage of one vertex set of a bipartite graph.

- *Fast greedy approach with index* - we present an index-based fast greedy algorithm with an approximation ratio $1/4$ to enumerate top-k t-biclique with maximum coverage.

- *Novel polynomial heuristic approach* - we propose r_t-core number and based on that we present a novel heuristic approach that smartly generates t-bicliques with large coverage and selects top-k t-bicliques with maximum coverage among them.

- *Experimental analysis* - we carry out extensive experimental studies on all our proposed algorithms across various real datasets.

1.3 Organizations

The remaining part of the thesis is organized as follows:

1. Chapter 2 discusses the survey of the related work on modeling and searching cohesive subgraphs and bipartite subgraphs.

2. Chapter 3 studies the problem of enumerating all the maximal bicliques in the bipartite graph and provides a pivot-based algorithm with experimental evaluation.

3. Chapter 4 presents the problem of searching personalized k-wing in bipartite graphs, the corresponding algorithms and the experimental results.

4. Chapter 5 studies the problem of maximizing the vertex coverage for top-k t-bicliques, the corresponding algorithms and experimental evaluations.

5. Chapter 6 concludes and discusses the possible future work of this thesis.

Chapter 2

Literature Review

The problem of finding cohesive subgraphs is fundamental in graph mining. Finding cohesive subgraphs in a bipartite graph has received significant attention as it is more unique and challenging. Unfortunately, even the powerful traditional models for searching cohesive subgraphs cannot fully explore the more profound insights as they are defined on general graphs. Moreover, in the bipartite cohesive subgraph search, the studies are still at an early stage. This chapter conducts a detailed literature review on works related to our research problems. We first present different literature works on cohesive subgraph models in unipartite graphs in Section 2.1. Secondly, we introduce some of the bipartite subgraph models in Section 2.2. Lastly, we discuss related works on cohesive subgraphs search for unipartite and bipartite graphs in Section 2.3.

2.1 Cohesive subgraph models in unipartite graphs

This section presents some of the fundamental cohesive subgraph models studied in unipartite graphs for various application scenarios. Since bipartite graphs can be considered an extension of unipartite graphs, understanding such cohesive subgraph models provides an advantage in proposing bipartite cohesive sub-

graph models. Each cohesive subgraph model has predefined metrics or measurements to define cohesiveness. A subgraph that satisfies such cohesiveness metrics is considered a cohesive subgraph. A cohesive subgraph represents a community for a unipartite graph as all the vertices are of the same type, which is not the same for bipartite graphs. In particular, a community is a group of strongly connected vertices internally. We introduce some of the prevalent cohesive subgraph models in a unipartite graph as follows.

2.1.1 k-core

k-core was first introduced by Seidman in [4] as a connected subgraph with each vertex having at least k neighbors within the subgraph, and the subgraph is not extensible. The formal definition of a k-core in [4] and [62] is as follows.

Definition 2.1. *Given a unipartite graph G and an integer k, if $\exists$ H, a maximal connected subgraph of G with $\delta(H) \geq k$, where $\delta(H)$ is the minimum degree of a vertex in H. Then we say H is a k-core of G.*

A core number is assigned for each vertex v in graph G. For a vertex v; the core number is defined as $core(v) = k$, if v is contained by a k-core but not any $(k+1)$-core. The core number measures how sparse H, i.e., the higher the value of k more connected is the subgraph is. Core decomposition is the algorithm that discovers all the possible k-cores in G. The well-known core decomposition in-memory algorithm is proposed in [4], which requires $\mathcal{O}(m+n)$ time, where m is the number of edges and n is the number of vertices in G. The algorithm iteratively removes the vertex with the minimum degree in the current subgraph while efficiently updating the remaining vertices' degree and maintaining a non-increasing degree order w.r.t. the most recent degrees.

2.1.2 k-truss

k-truss model is another well known and extensively researched cohesive subgraph model in unipartite graphs. Contrary to k-core, which uses the degree of a vertex, a k-truss is defined based on triangles (Δ) formed by distinct vertices. A triangle in G represents a cycle of length 3. k-truss was first introduced by Cohen [3] and is defined as a subgraph where every edge takes part in at least (k-2) triangles ($k \geq 3$) in the subgraph. The formal definition of a k-truss is as follows.

Definition 2.2. *Given a unipartite graph G and an integer k, if $\exists\ H_k$, a maximal connected subgraph of G that satisfies $\forall e \in H_k\ :\ |\Delta(e)| \geq k$, where $|\Delta(e)|$ is the number of triangles in H_k in which an edge e participates. Then we say H_k is a k-truss of G.*

A truss number or trussness for an edge $e \in G$ is defined as the maximum value of k such that e is contained by a k-truss but not any (k+1)-truss. Similar to k-core, there are various truss decomposition algorithms for computing the trussness of all the edges in G. The well-known $\mathcal{O}(\sum_{v \in V(G)} deg(v)^2)$ truss decomposition is proposed by progressively removing all the edges with common neighbors no greater than (k-2) in the remaining graph, increment the value of k by one and repeat this process until there are no edges left in the remaining graph [3]. The time complexity of the truss decomposition was improved to $\mathcal{O}(m^{1.5})$ by adopting the bucket sort based technique for efficiently maintaining the non-decreasing order of the common neighbors [63]. A truss decomposition has also been studied for uncertain graphs [64], [65].

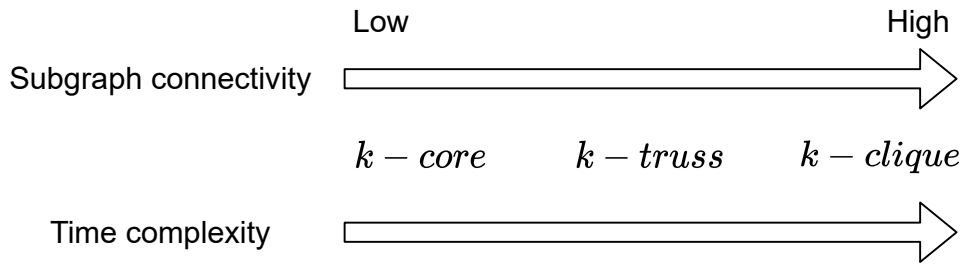


Figure 2.1: Cohesive subgraphs connectivity and time complexity comparison

2.1.3 Clique

Clique is the densest subgraph structure in graphs [66], [67]. It is defined as the subgraph in which all the vertices are connected. The formal definition of a clique is as follows.

Definition 2.3. *Given a unipartite graph G and an integer k, if $\exists$ C, a connected subgraph of G such that for each pair of vertex (u, v) in G forms an edge in C. Then we say C is a k-clique or clique when the context is obvious.*

A maximal clique is the one which any other clique cannot contain in G. The classical Bron–Kerbosch algorithm was proposed to enumerate all maximal cliques using the backtracking framework in [68]. [69] introduced the optimization of the enumeration algorithm by pivot pruning. Pivot pruning is based on the observation that any resulting maximal clique for a vertex must be a subset of its neighboring vertices. [70] proposed a greedy approach to select potential pivot, which improves the pivot pruning to reduce search space and expedites the maximal clique enumeration. [70] also provides the worst-case running time of BK-algorithm as $\mathcal{O}(3^{n/3})$, where n is the number of vertices in G.

2.1.4 Other models

Some of the other interesting, cohesive subgraph models in unipartite graphs are as follows. k-edge-connected subgraph [71]–[73] is a connected subgraph

that cannot be disconnected by removing less than k edges. k-edge-connected provide more cohesiveness than k-core within the polynomial time. However, this requires a trade-off with more computational cost.

k-plex model was introduced by [74] and is defined as a subgraph containing n vertices where each vertex is connected to at least n-k vertices. Optimized serial and parallel enumeration algorithms for all maximal k-plexes are studied in [75], [76].

A more generic model of k-(r, s)-nucleus, where r and s are small positive integers and $r < s$, was proposed in [77] inspired by the classical subgraph models k-core and k-truss. k-core and k-truss are the special cases subgraphs for a k-(r, s)-nucleus for $r = 1$, $s = 2$ and $r = 2$, $s= 3$ respectively.

Morris introduced the p-cohesion subgraph model in [78], where each vertex has at least a fraction p of all its neighbors within the subgraph. Unlike other subgraph models, p-cohesion ensures more inter-cohesiveness and outer-sparseness.

2.2 Biparatite cohesive subgraph models

In this section, we study some of the fundamental bipartite cohesive subgraph models. Bipartite graphs are considered an extension of unipartite graphs, representing two different types of entities and no same types of entities are connected. Unlike unipartite cohesive subgraph models, where the cohesiveness is measured using the interactions between the same type of vertices, it is not the same in bipartite cohesive subgraph models. Since any pair of vertices in a particular entity are not directly connected, a bipartite cohesive subgraph model defines connectivity based on the common interests of one type of entity in the other. Like unipartite subgraph models, each bipartite cohesive subgraph model

uses predefined metrics or measurements to define cohesiveness. We now introduce some of the fundamental cohesive subgraph models in a bipartite graph.

2.2.1 (α, β)-core

(α, β)-core was first introduced by [79] for good visualisation and analysis of bipartite graphs. A formal definition for (α, β)-core is as follows.

Definition 2.4. *Given a bipartite graph G and integers α, and β, if $\exists\ B(U' \cup V', E')$, a maximal induced subgraph of G with $|\Gamma(u)| \geq \alpha$ and $|\Gamma(v)| \geq \beta$, where $\Gamma(\cdot)$ denotes the neighbors of a vertex in B, $u \in U'$ and $v \in V'$. Then we say B is a (α, β)-core of G.*

[15], [80] and [81] extended the classical k-core decomposition algorithm for computing (α, β)-core. In particular, [81] proposed an efficient novel index (*BiCore-Index*) based approach to compute (α, β)-core in large bipartite graphs. *BiCore-Index* is proven to consume only $\mathcal{O}(m)$ linear space complexity, where m is the number of edges. Moreover, the algorithm's time complexity for *BiCore-Index* construction is given as $\mathcal{O}(\delta \cdot m)$, where δ is experimentally shown to be much smaller than $\sqrt{m}$ in practice. The notion of efficient *BiCore-Index* construction is to carefully consider the computation sharing between two vertex sets during the core decomposition.

2.2.2 k-wing

Butterfly (i.e., a 2×2 complete bipartite subgraph) is the basic building block in a bipartite graph and has drawn great attention recently [16], [17], [32]–[35]. It is analogous to a triangle (a smallest clique and basic building block of cohesive subgraphs) in a unipartite graph and serves as the smallest unit of cohesion [33], [34]. As such, k-*bitruss* and k-*wing* subgraphs which are based on butterfly

ensure more cohesiveness and are preferred over (α, β)-core which is not based on butterfly. A k-*bitruss* was first introduced in [82] as a bipartite subgraph which satisfies the minimum k-butterfly constraint defining that every edge in the bipartite subgraph participates in at least k butterflies and the bipartite subgraph is non-extensible. [82] proposed polynomial time algorithm $\mathcal{O}(|E|^2)$ for the k-bitruss decomposition algorithm and provides two significant properties. First, Uniqueness, i.e., the k-bitruss decomposition is unique. Second, Nestedness, i.e., a (k+1)-bitruss is contained within k-bitruss $\forall k \geq 0$.

k-wing was first proposed by Sariyüce et al. [17]. It strengthens a k-bitruss by adding a butterfly connectivity constraint defined as follows. Any two edges in a k-wing subgraph are either in a butterfly or can be reached by a set of edge-overlapping butterflies. The rationale is that a butterfly represents the strong and stable relationship among two pairs of vertices. If any two edges in a k-bitruss subgraph are also butterfly connected, this k-bitruss subgraph must be strongly connected and have a more cohesive structure among all involved vertices. This allows a vertex to participate in multiple bipartite cohesive subgraphs. The time complexity for k-wing decomposition proposed in [17] is $\mathcal{O}(|U||V||E|)$.

Counting the number of butterflies for each edge is the first step to compute a k-wing or k-bitruss. Given a bipartite graph G, the proposed algorithms [17], [82] for k-wing computation, first count's the number of butterflies on each edge in G. Afterwards, the edge with the lowest number of butterflies is iteratively removed from G until all the remaining edges appear in at least k butterflies. The demand for butterfly counting in large networks can be very high. Therefore a significant amount of study has been done for efficient butterfly counting. [83] proposed three different types of algorithms to count the number of butterflies (rectangles), i.e., In-Memory rectangle counting, I/O-Efficient rectangle counting and Parallel rectangle counting. The time complexity of In-Memory

rectangle counting is $\mathcal{O}(\sum(deg(v)^2))$. [84] presented exact and approximate algorithms for butterfly counting in bipartite graphs. The exact algorithm proposed in [84] has the complexity of $\mathcal{O}(min\{\sum(deg(v)^2), \sum(deg(u)^2)\})$. The difference in the time complexity [84] is due to selecting the cheaper option (vertex set) depending on the sum of vertex degrees in each vertex set. This selection of vertex sets yields noticeable speedups in performance over the [83]. Wang et al. [85] proposed a vertex priority based paradigm for computing butterflies with time complexity $\mathcal{O}(\sum_{(u,v)\in E} min\{deg(u), deg(v)\})$, instead of vertex sets as in [84]. [85] also proposed novel cache-aware strategies to further improve the efficiency of their algorithm.

2.2.3 Biclique

Biclique is the densest bipartite cohesive subgraph model. Biclique is a complete subgraph of a given bipartite graph such that all the vertices of one vertex set are connected to all the vertices of another vertex set. It is formally defined as follows.

Definition 2.5. *A biclique within G is a couple (set pair) (U', V') such that $U' \subseteq U, V' \subseteq V$ and $\forall u \in U', v \in V', (u, v) \in E$.*

A biclique is a *maximal* biclique if any other biclique cannot contain it. Many biclique problems have been studied, such as enumerating all maximal bicliques (NP-complete [86]), maximum balanced biclique (NP-complete [87]), maximum vertex biclique (polynomial time [87]) and maximum edge biclique (NP-complete [88]). The classical maximal biclique generation problem is composed of generating all the maximal bicliques induced and non-induced for a given graph G. This class of problem has received a lot of attraction due to two major reasons first, the time complexity for the solution of the problem and second, a wide range of application scenarios. Moreover, solutions for this problem

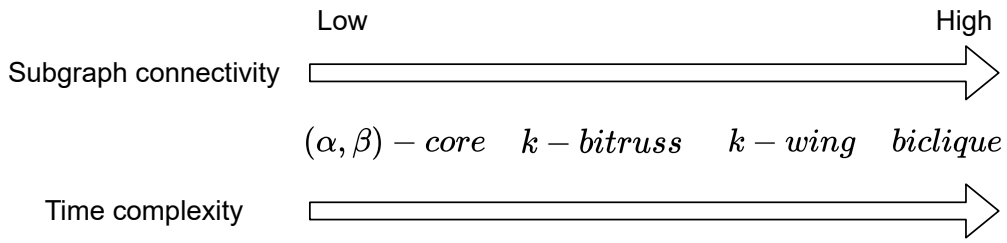


Figure 2.2: Bipartite cohesive subgraphs connectivity and time complexity comparison

can also be adapted for addressing other biclique problems. For the complexity part, it has been proved by Eppstein [86] in his proposed algorithm with the time complexity of $\mathcal{O}(a^3 \cdot 2^{2a} \cdot n)$, where n is the number of nodes in graph G, and a is the arboricity. The arboricity of the given graph is the minimum number of forests into which the edges of G can be partitioned. This shows that with respect to input size n, the problem cannot be solved in polynomial time, as the output size can be exponentially large in n. For more clarification, consider a complete graph with x vertices. As we know, exactly one maximal biclique will be induced by each proper partition of its vertex set in 2 subsets. Hence the number of maximal bicliques in a complete graph with x vertices is 2^{x-1}-1.

2.2.4 Other models

Some of the other interesting, bipartite cohesive subgraph models are as follows.

A relaxed version of biclique, quasi-biclique [89], was introduced to provide fault tolerance in a maximal biclique. Quasi-bicliques are used in modelling clusters in bipartite graphs for their resistance against noise and missing information, which are unavoidable in real-world datasets [90], [91].

[17] also introduced k-tip subgraph along with k-wing for identifying dense bipartite subgraphs. Similar to k-wing, k-tip was defined using the butterfly motif. However, a k-tip is defined for the vertices instead of edges.

Bi-triangle subgraph was proposed by [92], which is defined as a 6-cycle, or a cycle with three vertices from one vertex set and three vertices from another vertex set. Bi-triangle application mainly includes graph analyses such as transitivity coefficient, clustering coefficient, and truss computation in bipartite graphs [93].

The problem of finding diversified cohesive subgraphs in general graphs has also been extensively studied. Maximal cliques with less overlapping are utilized to address the problem of excessive size and redundancy in [94]–[96]. The problem of selecting top-k maximal cliques to maximize the vertex coverage is also studied in [97]. Diversified cliques and quasi-cliques search in temporal graphs have been studied in [58] and [98].

2.3 Searching cohesive subgraphs

Searching cohesive subgraphs determines all cohesive subgraphs for a user given query criteria. A query can be of any sort, such as a vertex, a set of vertices or a subgraph. In particular, subgraph cohesiveness is the common requirement for all such queries and is based on a cohesive subgraph structure. Searching cohesive subgraphs has been extensively studied for both unipartite and bipartite graphs under various application scenarios for real-world social networks. In the following, we introduce some of the cohesive subgraph models for searching.

2.3.1 Searching cohesive subgraphs.

In this set of work, we include the cohesive subgraph models from the field of unipartite graphs. The cohesive subgraph search has considered distinct subgraph structures such as k-*core* [38], [99], k-*truss* [45], [61], [100]–[102], k-*edge-connected* subgraph [71]–[73] and k-*clique* [96]. k-core is the least cohe-

sive subgraph but can be found most efficiently. k-edge-connected and k-truss subgraphs provide more cohesiveness than k-core. However, this requires a bit more computational cost. k-clique is the most cohesive subgraph as each vertex of a k-clique is connected to all the other (k-1) vertices but requires exponential running time. For each previously discussed cohesive subgraph models we now discuss their respective literature for finding such models.

k-core. Apart from simple undirected graphs, core decomposition has also been extensively studied in unipartite graphs containing vertices with attributes [103]–[106], weighted edges [107], and directed edges [108], [109]. A survey on core decomposition in networks is studied in [110]. There are numerous applications for a k-core model, e.g., Air traffic flow management [111], analyzing protein interaction networks [112], [113], large scale network analysis via graph visualization [114], [115], detecting abnormal patterns in graphs [116], [117], identifying influencers in a social network [118], finding attributed diverse communities [99], [119] and so on. Community search using k-core has a wide range of applications and has been comprehensively discussed in [120].

The massive growth in the input graph size mandates efficient algorithms to find k-core. Many techniques were studied, such as memory-efficient core decomposition [5], [121] and local estimation of core numbers [122]. A thorough comparison of core decomposition algorithms on a single PC across several large real-world networks is reported in [123]. k-core has also been studied for dynamic graphs, where algorithms are proposed to compute the new core number for the affected vertices efficiently and are known as core maintenance. The problem of core maintenance has been studied in [124]–[128]. k-core has been significantly adapted for modelling various community structures in real-world networks such as RB-k-core [129] (additional spatial constraint RB for k-core), (k, θ)-core [130] (k-core consists of vertices with probability at least θ),

(k, p)-core [131] (a k-core with at least p fraction of its neighboring vertices in the subgraph) and (k, h)-core [132] (all the vertices in a k-core are connected with at least h temporal edges).

k-truss. For the massive networks, several techniques are studied for truss decomposition, such as I/O-efficient truss decomposition [63], [133] and parallel algorithms [134]–[136]. Further, the problem of truss maintenance in dynamic graphs has been studied in [45], [61], [137]–[139]. Due to the cohesiveness and fast computation of k-truss, many k-truss community search models have been extensively studied to cater diverse range of application scenarios. Some of such communities are attribute-driven k-truss community [140], approximate closest community [141] (subgraphs containing a set of query nodes and a minimum diameter among such subgraphs), (k, r)-truss community in signed networks [142] (each edge has balanced and unbalanced support of k and r triangles respectively), probabilistic (k, γ)-truss community [64] (probability of each edge in a subgraph contained by $(k$-2$)$ triangles is at least γ). Community search using k-truss has also been discussed in detail in [120].

Clique. The problem of all maximal clique enumeration has been extensively studied in [143]–[147]. Enumerating all maximal cliques is time-consuming and might not always be interesting. Recently, some studies have focused on enumerating maximal cliques with fewer overlapping vertices [96], [97], [148]. Many algorithms have also studied the problem of clique enumeration for large networks, such as distributed algorithms [149]–[151] and external memory utilization [152], [153]. Moreover, maintaining maximal cliques in dynamic graphs has been studied [149], [154]–[158]. Clique has proven to be a significantly effective model across various application domains in searching cohesive communities. Some of the clique based communities are as follows. k-clique community is utilized to uncover overlapping community structures of complex net-

works in nature and society by Palla et al. [159]. A quasi clique model (γ-clique) [160] is a relaxed version of a clique that provide users to tune the edge density (using γ) for the resulting subgraph. [161] proposed a more customizable (α, γ)-online community to find γ-clique with each vertex having α neighbors in the subgraph and [162], [163] proposed (α, k)-clique for trust community mining in signed networks. Pattillo et al. [144] study several clique-based models and their respective properties.

2.3.2 Searching bipartite cohesive subgraphs.

In this set of work, the problem focuses on searching bipartite cohesive subgraphs. Many bipartite cohesive subgraph models have been studied, such as (α, β)-*core* [15], k-*bitruss* [16], k-*wing* [17], *biclique* [18] etc. We now present the detailed works related to previously discussed bipartite cohesive subgraph models as follows.

(α, β)-core. Recently, [164]–[166] and [40] proposed to search communities in bipartite graphs based on (α, β)-core structure. The computation of (α, β)-core for massive and dynamic bipartite networks is still in its early stage. Index-based approaches such as [81] and [166] have successfully computed (α, β)-cores for bipartite graphs with billion edges. They have also discussed the parallel and dynamic maintenance algorithms. [167] proposed a novel approaches for efficiently maintaining all the (α, β)-cores. Apart from the undirected bipartite graph (α, β)-core has been studied across other types of bipartite graphs as well, such as attributed graphs [168] and weighted graphs [40], [165]. (α, β)-core has a wide range of real-world applications scenarios, such as recommending good quality movies that the most active users like to other users [168], spam group detection on online shopping platforms [165], and solving other problems in bipartite graphs [166]. (α, β)-core also been utilized in modelling bipartite

cohesive communities. Unlike unipartite subgraphs, no users in a bipartite community are directly connected. Some of the (α, β)-core based communities are as follows. Cohesive subgraph search in weighted bipartite graphs ((k, ω)-core) [169], Pareto-optimal community [168], discovering personalized communities [165] and finding cohesive subgraphs with vertex engagement ((α, β)-core) and minimum tie strength (τ, number of butterflies) in bipartite graphs ($(\alpha, \beta)_\tau$-core) [164].

k-wing and k-bitruss. Bipartite cohesive subgraphs based on the butterfly subgraph have attracted much attention from researchers [16], [17], [35], [170]. Even though [33] and [34] have improved the butterfly counting, they are inefficient for searching bipartite cohesive subgraphs based on butterflies. Using the butterfly motif results in more connected bipartite subgraphs. As such, they are effective for applications that require denser bipartite subgraphs, such as word-document clustering [8], fraudulent group detection [17], [31], research collaboration networks [44] and viral marketing [13]. Many real-world applications, such as online social networks [42], web graphs [43] and collaboration networks [44], bipartite graphs are evolving where vertices/edges will be inserted/deleted over time dynamically. However, to the best of our knowledge, there has not been any work on finding k-wings in dynamic bipartite graphs. The insertion of an edge may affect the other edges by leading them to participate in a new k'-wing cohesive subgraph ($k' \neq k$). Unlike the case of k-truss [45] (the counterpart of k-wing for general graphs), where an edge could move to $(k+1)$-truss subgraph after insertion, the edge in a k-wing could move to $(k + i)$-wing subgraph after the insertion. Here, i could be greater than 1, which brings new challenges for k-wing maintenance.

Biclique. The biclique model can reveal the most cohesive bipartite subgraphs, and various biclique problems have been extensively studied [18], [29], [46],

[171], [172]. Tuza [173] proves bounds on the number of bicliques needed to cover the edges of a general graph. Orlin [174] proves for general bipartite graphs and Müller [175] for chordal bipartite graphs that computing the minimum cardinality of a biclique cover is NP-hard. Prisner [176] shows that bicliques are the key structure in certain classes of graphs and how generating all bicliques of such graphs seems an approach for recognition algorithms. Prisner [177] gives upper bounds on the number of bicliques in bipartite graphs and general graphs, exhibits examples of classes of graphs where the number of bicliques is exponential, and characterizes classes of graphs where the number of bicliques is polynomial with respect to the number of vertices in the graph. The complexity of deciding whether a graph contains a biclique of a certain size is first mentioned in [178] with the NP-completeness of the balanced complete bipartite subgraph problem. Bicliques have also been employed to study absolute bipartite retracts [179] and for characterizing chordal bipartite graphs [180], the problem being NP-hard. The biclique community till now has produced many algorithms for the same problem. Some of the noticeable works are the Modular Input Consensus Algorithm (MICA) [181], LCM-MBC [29], improved Maximal Biclique Enumeration (iMBEA) [28]. These works have provided the algorithm with the exponential run time cost. The dynamic maintenance for the biclique subgraph in a bipartite graph is discussed in [182].

Second, the domain of real-world applications of maximal biclique enumeration includes community detection, bioinformatics, closed item sets, etc. Some typical applications include detecting overlapping communities in *Noordin Top Terrorist Network* for finding more dangerous elements [21], discovering large dense graphs in massive graphs for spam group detection [22], generating probabilistic model for protein-protein interaction network analysis [11], extracting gene-phenotype information from transactional databases [23], constructing the

optimal phylogenetic tree [24] and gene expression [7]. From the domain of closed item sets, maximal biclique enumeration is used in association rule mining from transactional databases [12]. Some of the other applications include reducing the role mining complexity in role-based access control system [25], learning context-free grammars [26] and providing a model with constraints for solving chemical process scheduling problems [27].

We classify the literature tackling the problem of enumerating all maximal bicliques as follows.

1. **Exhaustive approach** - The naive approach to discovering all biclique is building all subsets of one vertex partition exhaustively. Further, we can retain the other partition by finding their common neighbors in other sets and checking for maximality. We need to store all the bicliques to verify their maximality in this approach. [183] presents an iterative algorithm to build subsets progressively. It starts from pairs of vertices and subsequently reaches a combination of larger sizes. Although the algorithm prunes the sizes of both partitions of a vertex in a biclique, ensuring the maximality of subgraphs requires enormous memory. The approach introduced in [184] uses set extension and expansion operations to build bicliques. A hash table is utilized that could determine the maximality of bicliques, and a queue to maintain bicliques prioritized by figure-of-merit values (e.g., p-values). The algorithm allows users to specify constraints on the figure-of-merit values that could filter out uninteresting bicliques. To conclude this category, it is right to say algorithms based on exhaustive search should implement effective pruning using some threshold on size or something which could reduce the enormous search space.

2. **Graph inflation to cliques** - This approach deals with converting a biclique problem to a clique problem. Since the clique problem has already

been well researched, we can easily solve this problem using heuristics and approximate approaches. We need to add edges to a bipartite graph to perform the conversion, i.e., inflation. This type of inflation is achieved by adding all possible edges between vertices of the same partition as described in [185]. After inflation, the resultant graph introduces another computational problem as the number of edges is enormously large. Consider G with $|U| = x$, $|V| = y$, and $|E| = m$. Inflating the bipartite graph results in a graph $B' = (Z', E')$, where $|Z'| = |X| + |Y| = x + y$ and $|E'| = |E| + \frac{x}{2} + \frac{y}{2} = m + \frac{x}{2} + \frac{y}{2}$. The edge density is defined as the ratio of the number of edges to the total number of possible edges [186], it is also known as the *"fill"*. Therefore, while *fill(G)* $=\frac{m}{x \times y}$, we get *fill(G')* $= \frac{m+\frac{x}{2}+\frac{y}{2}}{\frac{x+y}{2}}$. To illustrate how reasonable is the change, consider the bipartite graph G with $|X| = |Y| = k$ (for some $k \in N$), and $|E| = 0$ (i.e., $E =$, no edges). While the fill of G is zero, the asymptotic fill of G' is ≈ 50. (*fill(G)*$= \frac{x}{y} = 0$; *fill(G')* $\frac{0+(\frac{k}{2})+(\frac{k}{2})}{(\frac{k+k}{2})} = \lim_{k\to\infty} \frac{k^2-k}{2k^2-k} = 0.5$).

3. **Reduction to frequent itemset** - The reduction of the problem to frequent itemsets benefits, as cliques, by relying on a rich field of research. However, it contains other difficulties. The frequent closed itemsets approach comes from the field of data mining. It was observed in [187] that a transactional database could be represented by a bipartite graph, with a one-to-one correspondence between frequent closed itemsets and maximal bicliques. The transactional database corresponds to G, where U is the set of items (itemsets), V is the set of transactions (tids), and E is the set of pairs (item, transaction), i.e., an edge in the bipartite graph represents a transaction comprising the item. For example, a set of products offered by a supermarket. A transaction would be a subset of products (itemset) purchased by a customer. A subset of items is defined as a fre-

quent itemset if it occurs in at least one transaction. On the one hand, a frequent itemset and the set of transactions containing the frequent itemset form a biclique. On the other hand, the adjacency lists of a bipartite graph can be viewed as a transaction database by treating each vertex in one set as an item and each vertex in the other set as a transaction that contains a subset of items. A biclique can thus be mapped to a frequent itemset. A maximal biclique corresponds to a frequent closed itemset, where a frequent itemset I is said to be closed if the set of transactions containing I do not contain a superset of I. The support of a frequent itemset is the number of transactions in which the set occurs. Enumerating all maximal bicliques is equivalent to enumerating all frequent closed itemsets with support of at least 1. A correspondence between maximal bicliques of a general graph and frequent closed itemsets has been shown [188], suggesting that FPCLOSE and similar frequent itemset mining methods [189]–[192] help enumerate all maximal bicliques. Implementations of this approach require a post-processing step to obtain the transaction set for each frequent closed itemset. This is because the published methods output only the frequent itemsets (which correspond to half bicliques). Although this post-processing step is straightforward enough, it can be prohibitively time-consuming when the number of maximal bicliques is large. Moreover, known methods take the support level as an input parameter and find only frequent closed itemsets above the given support. In general, the lower the support, the longer the algorithms take. Support of 1 is the most difficult since all frequent closed itemsets must be found at this level. Therefore, these reductions to other domains may present practical and scalability problems [28] and may not be fully utilized to the fullest.

4. **Exploitation of specific graph** - Some special graphs like *Convex Bipartite* and *Chordal Bipartite* contain some properties that can be exploited to obtain better run time cost. A bipartite graph G is called convex (on V) if there exists an ordering of the vertices of V such that, for any vertex $u \in U$, vertices adjacent to v are consecutive in V [193]. A graph is called chordal bipartite if it is bipartite and each cycle of length at least 6 has a chord [194]. We can find the algorithms in the second category: limitation to convex bipartite graphs [181], [193], [194] and limitation to chordal bipartite graphs [194]. The real-world applications are limited for these graphs; hence there has been limited research done.

5. **Set enumeration tree** - recently, many algorithms have adapted the enumeration tree for a vertex set, systematically enumerating the power sets of the corresponding vertex set using a pre-imposed order. A smaller vertex set, say V, is selected for the set enumeration. Each node in the search space represents a possible candidate for a maximal biclique. The root represents an empty set. The *sub search space ($sub(X)$)* for a vertex set X is a subtree that is rooted at the vertex set X. Given a vertex set X, $cand(X)$ denotes the candidates for node X in the search space, containing only the vertex which comes later than the last vertex set in X. Some of the algorithms are FMBE [195], iMBEA [28], MINELMBC [196] and LCM-MBC [29].

Some of the other biclique problems which have been studied are top-k diverse bicliques and maximizing vertex coverage for top-k bicliques. Due to many cohesive subgraphs, selecting top-k representative subgraphs has been widely adopted. Such k results are usually considered as a diversified set, where each of the k results has fewer properties in common [119], [197], [198]. This set of problems focus on the various diversification definitions. Due to the similarity

of itemset and biclique, the maximal diverse frequent itemsets problem [199], [200] is discussed. An itemset is said to be diverse if its items belong to many different categories according to a given hierarchy of item categories. [201] studies the balancing of diversity for bipartite matching and defines it as the need to match a vertex with vertices in another vertex set across different bicliques. Further, the diverse pair matching is discussed in [202] and [203], which defines diversity as the sum of the differences in two vertex sets of a biclique.

The top-k coverage problem has been studied in bipartite graphs. This problem provides top-k bicliques, which maximize the bipartite graph's edge or vertex coverage. The theoretical significance of the problem has been well studied in [204]–[207]. Most of the works address the edge coverage, and the problem requires exponential time complexity in [208]. The problem of biclique partition is also studied along with the biclique cover problem, where the edges or vertices of k bicliques are exclusive to a biclique [209]. [210], [211] explore the constrained biclique cover problem in bipartite graphs.

Moreover, community search in a heterogeneous network has been studied in [212] and [213], which are based on the k-core and k-truss models, respectively.

Chapter 3

Pivot-based Maximal Biclique Enumeration

Enumerating maximal bicliques in a bipartite graph is an important problem in data mining, with innumerable real-world applications across different domains such as web community, bioinformatics, etc. Although substantial research has been conducted on this problem, surprisingly, we find that pivot-based search space pruning, which is effective in clique enumeration, has not been exploited in the biclique scenario. Therefore, we explore the pivot-based pruning for biclique enumeration. We propose an algorithm for implementing the pivot-based pruning, powered by an effective index structure Containment Directed Acyclic Graph (CDAG). Meanwhile, existing literature indicates contradictory findings on the order of vertex selection in biclique enumeration. As such, we re-examine the problem and suggest an offline ordering of vertices which expedites the pivot pruning. We conduct an extensive performance study using real-world datasets from a wide range of domains. The experimental results demonstrate that our algorithm is more scalable and outperforms all the existing algorithms across all datasets and can achieve a significant speedup against the previous algorithms.

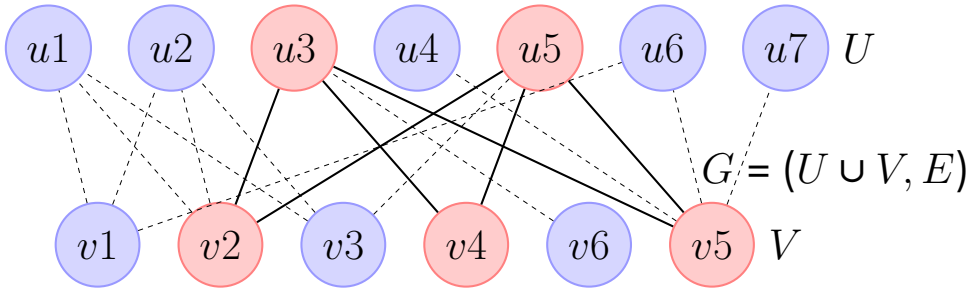


Figure 3.1: Maximal biclique $\{B = \{v2, v4, v5\}, \{u3, u5\}\}$ in a bipartite graph(G).

Chapter map. The chapter is organized as follows. Section 3.2 introduces the preliminaries and problem definition. Section 3.3 presents our methodology for dealing with the problem of enumerating all maximal bicliques. Section 3.4 presents the experimental analysis. Section 3.5 summarize this chapter.

3.1 Introduction

We study the problem of maximal biclique enumeration, we aim to address all the above three challenges. We propose an algorithm that can efficiently prune the search space using an index structure, without any overhead of extra computation during the enumeration. We propose **Pivot based Maximal Biclique Enumeration** (PMBE) algorithm for all maximal biclique enumeration. PMBE integrates *pivot pruning* technique with the LCM-MBC for reducing the search space. An efficient implementation of the pivot pruning is guaranteed by our proposed *containment directed acyclic graph* index structure *(CDAG)*. The semantics of ordering of vertices to enumerate maximal biclique is also studied, and a heuristic ordering is proposed for the heuristic pivot selection, which eventually optimizes the algorithm.

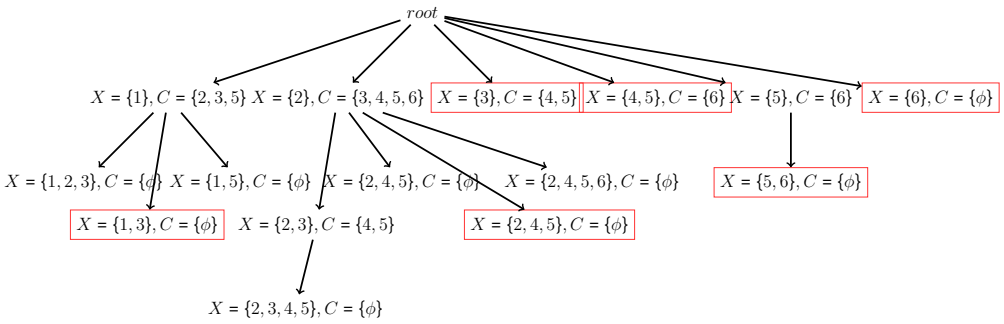


Figure 3.2: Enumeration tree for LCM-MBC algorithm, red boxes are the duplicate bicliques

3.2 Preliminaries and Problem Definition

Let $G = (U \cup V, E)$ be a bipartite graph, where U and V are two disjoint sets of vertices, and E is an edge set. A biclique is a complete subgraph of the given graph G, which is formalized as:

Definition 3.1. *Bipartite Complete Graph or Biclique: A biclique within G is a couple (set pair)* (X, Y) *such that* $X \subseteq U, Y \subseteq V$ *and* $\forall u \in X, v \in Y$, $(u, v) \in E$.

A biclique is a *maximal biclique* if it cannot be contained by any other biclique. Figure 3.1 shows the bipartite graph and a maximal biclique.

Problem Definition 1. *Given a bipartite graph* $G = (U \cup V, E)$ *with no multiple edges and no self-loops. We enumerate all the maximal bicliques in* G.

3.3 Methodology

This section explains how our approach works to tackle the challenges discussed previously. The *pivot pruning* approach has been first introduced by Bronn and Kerbosh [68] for finding cliques. It has proven to be a simple, yet effective technique in pruning cliques. Surprisingly, the technique has not been utilized, in the

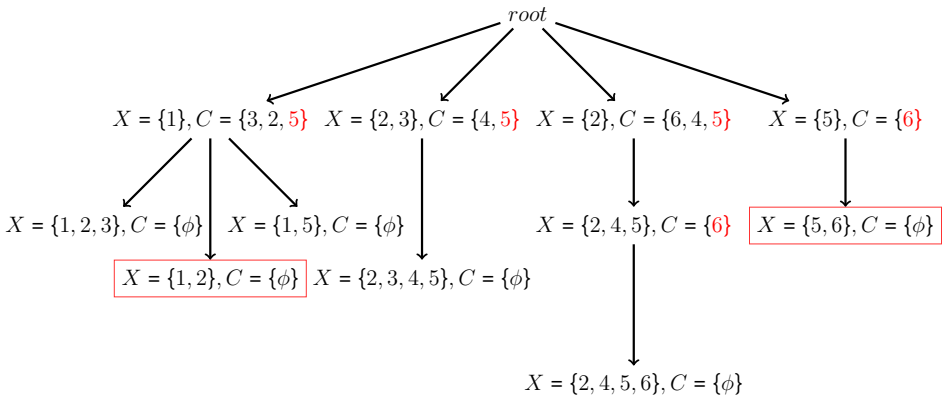


Figure 3.3: Enumeration tree for PMBE, Pivot is represented by red vertices. C shows the candidate at each level and red boxes are the duplicate bicliques.

field of bicliques, we now present a modified pivot pruning supported by ordering for an optimized enumeration of maximal bicliques. We observed that the *pivot pruning* effectiveness depends on the pivot selection. Therefore, an effective *rev-topological* order is proposed for the same. The proposed modification is achieved in the following steps. *Firstly*, the framework for our algorithm is discussed which uses two techniques namely *pivot pruning* and *rev-topological* ordering for optimization. Secondly, the *pivot pruning* is explained using the containment relationship (Definition 3.2). *Thirdly*, we propose the index CDAG, for the effective implementation of pivot pruning. *Fourthly*, we study the semantics of the ordering of vertices and suggest a *rev-topological* order for heuristic pivot selection. *Lastly*, the algorithm PMBE is discussed which enumerates all the maximal bicliques.

3.3.1 The Framework

The proposed framework of our approach can be observed in Algorithm 1. The framework uses the techniques of pruning and ordering to optimize the algorithm. We explain the two aspects of the framework, which are search space and enumeration. Some properties of enumeration are adapted from LCM-MBC

Algorithm 1: Enumerate all maximal bicliques

Input : $G = (U \cup V, E)$
Output: B is the maximal Biclique

```
1  Function EnumerateMaximalBiclique(U,V)
2      *Create a CDAG
3      *cand = Rev-Topological(V)
4      B = ∅
5      PMBE(B, U, cand)
6  Function PMBE(X,Y,cand)
7      if cand.isEmpty then
8          return
9      *Pivot Selection /* Select the vertex p from cand.
           */
10     foreach v ∈ cand do
11         cand ⟵ cand \ v
12         *Pivot pruning /* Skip the vertex (v) contained
               by the pivot p.                          */
13         if rangefinder(p,v) ≠ TRUE then
14             B = Γ(Γ(X ∪ {v}))
15             if B \ (X ∪ {v}) ⊆ cand then
16                 Output (B, Γ(X ∪ {v}))
17                 PMBE(B, Γ(X ∪ {v}), cand \ B)
```

[29].

Search Space. For a bipartite graph $G = (U \cup V, E)$, the search space is restricted to one of the vertex sets (U, V), as it can be used to determine the corresponding adjacency list. The search space of MBE is a set enumeration tree for a vertex set, which systematically enumerates the power-sets of the corresponding vertex set using a pre-imposed order. For the set enumeration, a smaller vertex set, say V, is selected. Each of the nodes in the search space represents a possible candidate for a maximal biclique. The root represents an empty set. The *sub search space (sub(X))* for a vertex set X is a subtree which is rooted at the vertex set X. Given a vertex set X, $cand(X)$ denotes the candidates for node X in the search space, containing only the vertex which comes later than the last vertex set in X.

Before introducing enumeration, we first define some notations to simplify the explanation. The common neighborhood for a vertex set X is denoted by $\Gamma(X)$. For a given vertex set $X \in P$ (P is the power-set of V), $\Gamma(X) = \{u | u \in U$ and $\forall v \in X, (u, v) \in E\}$.

The Enumeration. It is accomplished by using two functions as shown in Algorithm 1: (i) *EnumerateMaximalBiclique* and (ii) *PMBE*. EnumerateMaximalBiclique is responsible for performing the *rev-topological* order for an efficient pivot selection (line 3). *PMBE* which has been inspired by LCM-MBC for bipartite graphs, implements the pivot selection (line 10) and performs the *pivot pruning* (line 13) to enumerate all maximal bicliques. The framework includes creating the structure CDAG (line 2) for supporting *pivot pruning* and *rev-topological* ordering. Apart from the pruning, PMBE ensures that a biclique generated is maximal iff $\Gamma(\Gamma(X)) = X$ (line 15) [29]. The duplicate bicliques are pruned using line 16 [29]. The resulting optimized algorithm LCM-MBC enumerates the search space shown in Figure 3.2. LCM-MBC runs with an exponential complexity of $\mathcal{O}(|U||V|\beta)$, where β = number of maximal bicliques. However, it still suffers from the following two problems: firstly, the search space is still very large; secondly, there are numerous duplicate maximal bicliques generated (red boxes).

3.3.2 Pivot

The pivot technique for maximal clique enumeration is to prune search branches that cannot result in maximal clique as early as possible. The pivoting consists in the following: instead of iterating over each vertex and checking for maximal cliques, chose a pivot. The results will have to contain either the pivot or one of its non-neighbors, since if none of the non-neighbors of the pivot is included, then we can add the pivot itself to the result. Hence, only the pivot and its non-

neighbors need to be tested in the enumeration space. The properties of cliques and bicliques are distinct, hence selecting a pivot with the same notion will not be effective in the case of bicliques. Consequently, the pivot selection approach has to be modified for bicliques.

The motivation behind pivot pruning is to reduce the duplicate biclique enumeration. In our context, a pivot is a vertex selected among the candidates, which can be used to reduce or prune the enumeration space of our problem without losing any results. In order to define pivot formally, we first define the containment relationship as:

Definition 3.2. *Containment ($\subset$): For a given bipartite graph G=(U $\cup$ V, E), let $v_1, v_2 \in$ V. Then, v_1 is contained by v_2 iff $\Gamma(\{v_1\})$ is a subset of $\Gamma(\{v_2\})$.*

The containment relationship provides an elegant property which can be utilized as a criterion for pivot selection.

Property 3.1. *Given $v_1, v_2 \in cand(X)$, if $v_1 \subset v_2$, then sub(v_2) contains sub(v_1).*

Proof. Given $v_1 \subset v_2$, we have $\Gamma(\{v_1\}) \subseteq \Gamma(\{v_2\})$, which in turn implies $\Gamma(\Gamma(\{v_1\})) \subseteq \Gamma(\Gamma(\{v_2\}))$. Here $sub(v_1)$ and $sub(v_2)$ are the power sets of $\Gamma(\Gamma(\{v_1\}))$ and $\Gamma(\Gamma(\{v_2\}))$ respectively. Hence we have $sub(v_1)$ contained by $sub(v_2)$. □

Therefore, using Property 3.1, we define the pivot as:

Definition 3.3. *Pivot: Given a vertex set X, a vertex $p \in cand(X)$ is a pivot if $\exists\, v \in cand(X)$, such that $\Gamma(\{v\}) \subset \Gamma(\{p\})$.*

In the search space, at each level of recursion, i.e., vertex set X, we intend to prune away all the contained vertices for a corresponding selected pivot. Further, we propose the pivot pruning as follows:

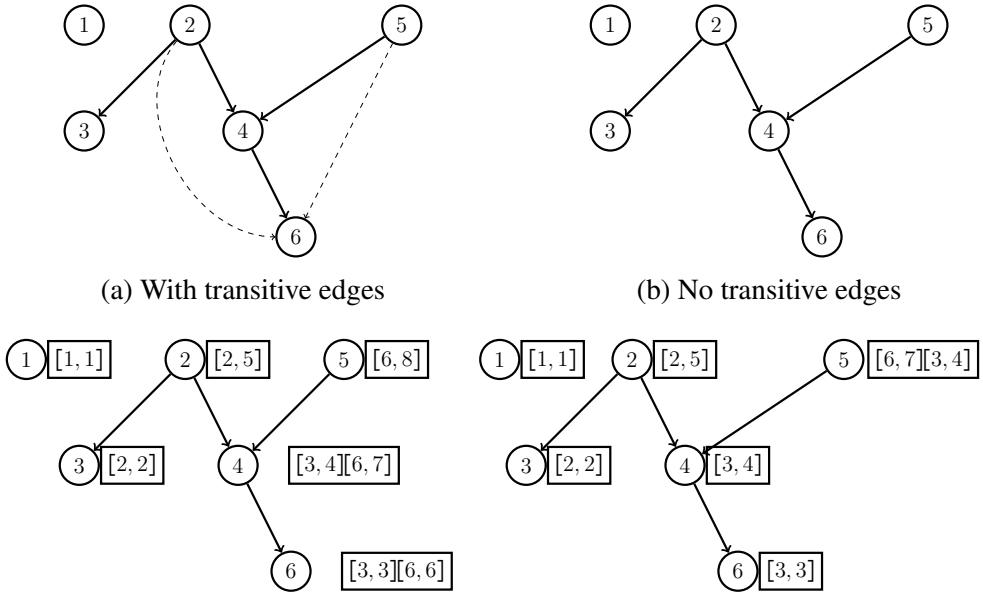
Proposition 3.1. *At each recursion level in search space, selecting one pivot* $p \in cand(X)$ *and pruning all vertices contained by p from cand(X) shall not lose any maximal biclique.*

According to Property 3.1, the pivot sub search space already contains sub search space of the pruned vertices. Therefore, removing the contained vertices would reduce the overhead of checking the branches that cannot result in maximal bicliques, which would not affect the results. The example of pivot pruning can be seen by comparing Figure 3.2 and 3.3, at level 1 vertices 4 and 6 are contained by pivot 5. The bicliques enumerated (Figure 3.2) from vertices 4 and 6 are redundant and an unnecessary extra cost is utilized for handling them. Once the pivot criterion is decided, we now apply the pivot pruning. The naive method includes selecting each neighbor of a vertex and finding it in the neighborhood of the pivot, is expensive and inefficient. Therefore, to address the problem of pivot pruning efficiently, we propose an index structure CDAG.

3.3.3 Containment Directed Acyclic Graph

The containment directed acyclic graph (CDAG) is an offline structure used to store the containment relationship (Definition 3.2). The challenge in implementing the pivot pruning is proportional to answering the query of whether vertex v_2 contains vertex v_1. CDAG addresses this problem and provides the acknowledgment for the query in almost constant time. It is composed of two components: directed acyclic graph (DAG) and range index (R). A DAG is created for a $cand$ where the directed edges represent the containment relationship. The vertex v_1 is said to be a *child* of v_2 if there exists a direct edge from v_2 to v_1. Although a DAG is sufficient for finding the containment relationship, we improve the structure by integrating the range index.

Definition 3.4. *The range index (*$R_v = [x, y]$*, where* $x, y \in Integer$*) for a*



(a) With transitive edges (b) No transitive edges

(c) Inefficient range index (d) Efficient range index

Figure 3.4: Containment Directed Acylic Graph(CDAG)

vertex $v \in cand$, is an interval assigned to the corresponding vertex of a DAG during the traversal. x and y denote the beginning and the termination of the traversal.

Figure 3.4(a) represents the DAG for the graph in Figure 3.1 and is sufficient for finding the containment relationship, however, we improve the structure by integrating the range index. For an efficient and non-redundant range indexing, the transitive edges from the DAG are removed, which results in Figure 3.4(b). The range index is assigned to each vertex in the DAG and can be utilized to find the containment relationship quickly. The range indexing is performed using a depth first search approach (DFS). We begin the traversal of the DAG with each root vertex and assign the range index to the vertices in a CDAG. For an efficient and non-redundant range indexing, the transitive edges from the DAG are removed. The vertices which are not contained by any other vertex are considered to be the root vertex, e.g., vertices $1, 2$ and 5 in Figure 3.4(c). However, if a vertex is contained by more than one vertex, it can have more

than one range index, e.g., vertices 4 and 6 in Figure 3.4(c). This unnecessary overhead of range indexes on vertices 4 and 6 can be reduced by adapting a technique from [214]. We create two types of range index for each vertex in a DAG, first, its own ($rangeindex$) and second, obtained from the common children ($child\_rangeindex$), which has been already visited. The fundamental approach of creating a CDAG is to first create a DAG and then implement the range indexing. For a given graph $cand$ a CDAG can be defined as:

Definition 3.5. *A CDAG, $C = (\bar{V}, \bar{E})$ is an acyclic graph, such that $\forall \bar{v} \in \bar{V}, \bar{v}$ is composed of $v \in cand$ and R_v, and $\forall (\bar{v}, \bar{u}) \in \bar{E}$, $\bar{u} \subset \bar{v}$.*

Figure 3.4(d) is the required CDAG for the $cand$. There are some common terminologies with respect to a CDAG. A **path** is defined as a sequence of edges from one vertex to the other in a CDAG. A vertex v of a CDAG is said to be **reachable** from another vertex u, if there exists a path starting from u and ending at v. Hence, reachability in a CDAG implies the containment relationship in our problem. Therefore, we conclude the following proposition.

Proposition 3.2. *v is said to be reachable from x in a CDAG, if range index of v lies within one of the range indexes of x.*

As shown in Algorithm 5, we provide the pseudo code for the operation rangefinder. To answer the query of containment relationship between vertices we contrive a function *rangefinder* (line 12) in Algorithm 1. We exploit the Proposition 3.2 to formulate *rangefinder*, i.e., comparing the range index of the vertices in a CDAG. The runtime complexity of the *rangefinder* is $\mathcal{O}(|\mathrm{R_{pivot}}||\mathrm{R_v}|)$, where $\mathrm{R_{pivot}}$ is the number of range indixes of pivot and $\mathrm{R_v}$ is the number of range indexes of vertex v. Figure 3.2 displays at each recursive level in an enumeration tree a pivot is selected and results in better pruning of the enumeration space. However, the pruning power of a pivot depends on the

Algorithm 2: rangefinder(p, v)

Input : Pivot vertex p, Vertex to be checked v
Output: True if v is contained by p

```
foreach r_p ∈ R_pivot do
    foreach r_v ∈ R_v do
        if r_v.begin ≥ r_p.begin and r_v.terminate ≤ r_p.terminate then
            return true
return false;
```

selection of the pivot. If a pivot selected does not contain any vertices, then *rangefinder* will be an overhead. Therefore, we select a pivot at each recursive level without any extra cost, such that it has the potential to contain more vertices. We now discuss the pivot selection technique.

3.3.4 Heuristic Pivot Selection Using *Rev-Topological* Ordering

The pivot selection requires an approach to be fast yet effective at the same time. We propose a heuristic pivot selection, where we select a vertex from $cand(X)$ in constant time. To ensure the effectiveness of the pivot, we study the ordering of $cand(X)$. Ordering of a vertex set manifests a critical role in the pivot selection. The algorithms like MICA and LCM-MBC have been using the lexicographical sort. The current fastest algorithm for our problem iMBEA, suggested sorting the vertices according to the non-decreasing order of number of common neighbors which incurs extra cost. The contradicting conclusion for the same problem on efficiency of discovering biclique is proposed in [30], suggesting the non-increasing order. From these contradicting techniques, we conclude that the better sorting technique being opted is closely related to the algorithm being implemented for the enumeration. We propose an offline ordering which saves the cost at each recursion level and also ensures the effec-

Algorithm 3: Rev-Topological(n)

```
Input  : n number of vertices in a CDAG, S is a stack, indegree
         contains the number of incoming edges for a vertex in a
         CDAG
Output: cand is the rev-topological order of cand
1  foreach node ∈ DAG do
2      if indegree[node]==0 then
3          S.push(node);
4  while S.isnotEmpty do
5      current=S.peek();
6      if current.children.isEmpty then
7          push all the unvisited children to S
8          if all children are visited then
9              S.pop();
10             cand.add(current);
11     else
12         S.pop();
13         cand.add(current);
14 return cand;
```

tiveness of pivot selection at the same time. The order follows the property that if $v_2 \subset v_1$ then v_2 should occur before v_1. With this specific ordering we can easily select the pivot from the $cand$, which ensures that all pruned vertices during the enumeration does not affect the results, as at the end, pivot enumerates their collective search space (Proposition 3.1). The CDAG created stores the reachability information and can be utilized to propose the required order which is *rev-topological* order. The *rev-topological* order is one where no vertex in a CDAG can occur before its reachable vertices. The *rev-topological* order is achieved by exploiting the already known algorithm for topological ordering. Algorithm 3 describes the Rev-Topological ordering.

Proposition 3.3. *The $cand(X)$ is ordered using rev-topological order offline using the CDAG to increase pivot pruning power.*

Following the selection of the ordering of vertices, we utilize the heuristic

approach to select a pivot among the $cand(X)$. Since, it is known that each vertex in $cand(X)$ can reach only the vertices before its occurrence, hence we select the last vertex in $cand(X)$. Therefore, given a list of vertices $cand(X)$ sorted in a *rev-topological* order, we heuristically select the last element of $cand(X)$ as the pivot. *Rev-topological* ordering visits all the root vertices in a CDAG and performs a DFS to explore the subtrees of each root. The implementation of *rev-topological* can be achieved by performing a reverse topological sort, i.e., a parent in a CDAG appears before the child. Using Figure 3.4(d) as a running example, only the vertices 1,2 and 5 are pushed in the stack initially. The *rev-topological* order obtained for the CDAG in Figure 3.4(d) is $\{1, 3, 6, 4, 2, 5\}$, vertex 5 is selected as a pivot. The runtime cost of the *rev-topological* ordering is of linear order. In case we have more than one choice for a node selection, a lexicographical selection is incorporated to provide the uniqueness. After discussing the techniques for addressing the challenges of pruning and ordering, we devise our algorithm PMBE.

3.3.5 PMBE

The algorithm incorporates the pivot pruning and *rev-topological* ordering of vertices, utilizing the CDAG for their effective implementation. The heuristic approach is then suggested for fast pivot selection.

Algorithm. Algorithm 1 is the pseudo code for our approach of enumerating all maximal bicliques. Given a bipartite graph $G = (U \cup V, E)$, the vertex set V is considered as the candidate initially. The algorithm is implemented in two steps: firstly, we create CDAG and utilize it to order the candidate set $cand$ offline (lines 2-3) and secondly, to enumerate all the maximal bicliques for $cand$ (line 5) using the function PMBE. The first step has been discussed in detail already, we further discuss the enumeration part. The inputs for enumeration are initially

vertex set $B = \emptyset$, U and $cand = V$.

Heuristic pivot selection. PMBE starts with the pivot selection, using Proposition 3.3, we heuristically select the last vertex from the sorted $cand$ (line 10).

Pruning using the pivot. After selecting the pivot, PMBE enumerates maximal bicliques for the $cand$ (line 11-20) by utilizing the optimized pivot pruning technique which uses the function $rangefinder$ (line 14). The $rangefinder$ returns *true* for a vertex *v*, if it is *reachable* from the *pivot*, hence the sub search space of *v* is pruned. After pruning, the maximal biclique is enumerated in line 15. Thus, eliminating a vertex *v* optimizes the algorithm by removing the overhead of biclique generation and duplicate verification.

Subsequently, the maximal biclique is reported in line 17. The recursive call (line 18) with the reduced $cand$ and increased B, is only proceeded if there are any candidates present (line 7). The effectiveness of PMBE and LCM-MBC can be compared by using Figures 3.2 and 3.3, which appreciates two aspects of the PMBE over LCM-MBC. First, the duplicate bicliques (red boxes) have been decreased, second, the efficient pivot selection is achieved using the *rev-topological* order.

Time Complexity. The effective runtime of PMBE is from line 7 to line 20 in Algorithm 1. The pivot selection line 10 is done using a heuristic approach in constant time. The complexity of rangefinder is $\mathcal{O}(|\mathrm{R_{pivot}}||\mathrm{R_v}|)$, which is almost constant. The function is recursively called only for the maximal bicliques and the total number of maximal bicliques is β (line 18). Each maximal biclique corresponds to a node in the search space with a cost of $\mathcal{O}(|V|d_{max})$ each, d_{max} is the maximum degree in V. Furthermore, the total running time complexity of the algorithm is same as of the enumeration of the search space, i.e., $\mathcal{O}(|V|d_{max}\beta)$.

Space Complexity. The space complexity depends on the implementation de-

tails. It is independent of the number of maximal bicliques as we do not save them in memory. The major contribution of space complexity are input bipartite graph which requires $\mathcal{O}(|E|)$ space and the enumeration in memory. Since we are using DFS the space for enumeration tree requires $\mathcal{O}(|V|d_{max})$. Each node in the enumeration tree is a biclique which is $\mathcal{O}(|U| + |V|)$. The CDAG created contains each vertex in $cand$, which requires $\mathcal{O}(|U|)$ space. Therefore, the total space complexity is $\mathcal{O}(|V|d_{max} + |E| + |V| + |U| + |V|) = \mathcal{O}(d_{max}|V| + |E|)$.

3.4 Experimental Results

The previous sections laid the conceptual foundations for the PMBE's potential to efficiently enumerate all maximal bicliques. In this section, we evaluate the efficiency of PMBE by comparing with the other state-of-the-art algorithms on real-world datasets across numerous domains.

Experimental Setup. All the experiments were conducted on Eclipse IDE, deployed on the platform 64x Intel(R)Core(TM) i5-6400T with CPU frequency 2.20GHz and 8 GB RAM, running Windows 10 Enterprise operating system. To perform a fair and comprehensive comparison between the algorithms we only clock the running time of each algorithm. The running time of each algorithm is averaged over 10, 7 or 5 runs for datasets that can be finished within 10 minutes, half an hour or one hour. The experiments were performed without using any kind of parallelism, i.e., single core was used. All the algorithms were implemented in Java.

Datasets. To establish the dominance of our algorithm, we have selected the datasets s.t. they cover diversified real-world application domains. Doing so will demonstrate the capabilities of the PMBE algorithm to efficiently enumerate all the maximal bicliques for a wide range of domains and graph characteristics.

Table 3.1: Real-world datasets

Datasets	$\|U\|$	$\|V\|$	$\|E\|$	Bicliques
Corporate Leadership (CL)	20	44	99	66
Unicode (U)	254	868	1255	460
UCforum (UC)	899	1421	33720	16261
Service (S)	6624	10106	50632	49538
Movielens u-t (User-Tag) (M_{u-t})	4009	20537	95580	41713
Movielens u-i (User-Movie) (M_{u-i})	4009	11610	95580	140266
MovieLens t-i(Tag-Movie) (M_{t-i})	16528	24129	95580	166380
Wikibooks (Books) (W)	2884	30997	201727	2365457

The datasets were obtained from KONECT repository [215]. Table 3.1 shows the list of datasets and their corresponding properties.

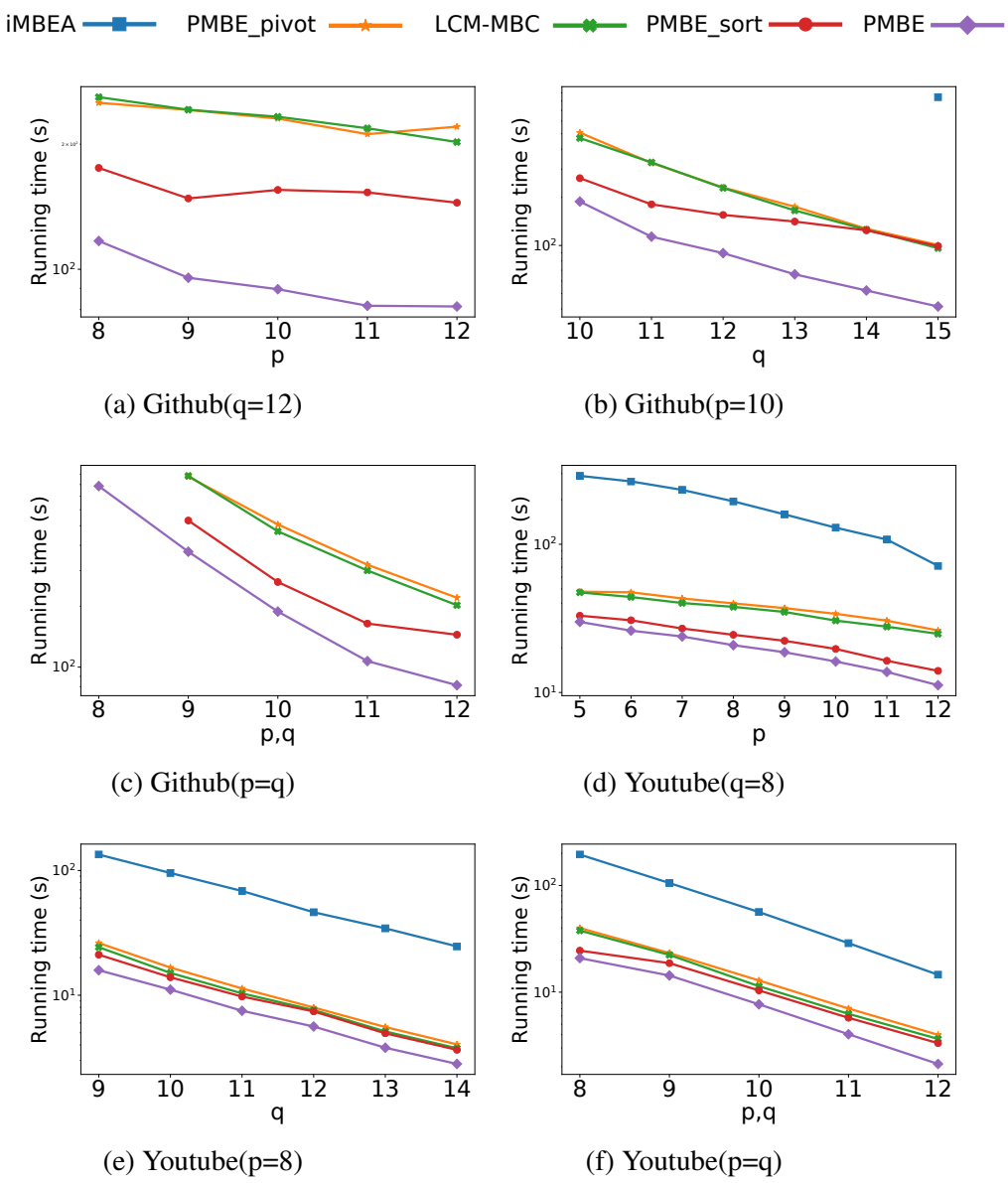
Algorithms. We now proceed to inspect the performances of the following five algorithms. Firstly, **iMBEA**, the algorithm is inspired from the classical BK algorithm [68] with an exponential time complexity of $\mathcal{O}(d_{max}|V|\beta)$. It employs non-decreasing sorting on $cand$ for efficient branching and pruning techniques to remove paths that cannot lead to maximal bicliques. Secondly, **LCM-MBC**, optimized version of LCM-MBC [29] for bipartite graphs. The optimized version does not require any unnecessary post-processing and duplicate biclique enumeration [28]. Thirdly, **PMBE**, pivot based algorithm which is proposed in this chapter. Lastly, to establish the effectiveness of pivot pruning and *rev-topological* ordering individually we use **PMBE_pivot** and **PMBE_rev-top**, where algorithm uses only pivot and ordering technique respectively. The runtime complexity of PMBE_pivot and PMBE_rev-top are of the same order of PMBE. Moreover, the index construction time which includes the creation of CDAG and performing *rev-topological* ordering is also shown in Table 3.2. We observe, improvement in the performance of PMBE with the increase in the size of the dataset (Table 3.1 and 3.2).

Results and Discussion. Table 3.2 illustrates the performance of all the al-

Table 3.2: Real-world datasets and their respective running time (sec.) for enumerating all maximal bicliques

Datasets	LCM-MBC	iMBEA	PMBE	PMBE_pivot	PMBE_rev-top	Index Construction
CL	0.002	0.003	0.003	0.004	**0.002**	0.005
U	0.028	0.03	0.028	0.031	**0.025**	0.025
UC	1.068	1.299	0.77	1.1	**0.76**	0.063
S	12.682	14.37	13.605	**12.675**	14.657	8.645
M_{u-t}	594.952	857.364	**438.126**	623.333	474.09	3.989
M_{u-i}	31.919	43.651	**29.016**	33.021	29.913	5.789
M_{t-i}	244.309	220.508	201.62	246.628	**193.478**	14.159
W	6.636	13.323	**4.265**	6.753	4.629	2.315

gorithms on real-world datasets. From these experiments, we conclude that PMBE has outperformed iMBEA and LCM-MBC in almost all the scenarios. However, for datasets Unicode and Service the PMBE_rev-top and PMBE_pivot outperform all the other algorithms. We observe, at least one of the versions of PMBE has outmatched iMBEA and LCM-MBC. The results of the experiments also imply the following conclusions: (i) we perceive that ordering plays a more crucial role than the pivot as the efficiency obtained from pivot pruning is less, compared to *rev-topological* order. (ii) Although the individual algorithms are less efficient, the combination of the two techniques enhances the algorithm PMBE further. We extend PMBE to find large maximal bicliques in large datasets, i.e., GitHub ($|U|$=56519, $|V|$=120867 and $|E|$=440237) and Youtube ($|U|$=94238, $|V|$=124325 and $|E|$=293360). Figure 3.5 displays the results of the algorithms which are all in our favor. The threshold for a biclique is the number of minimum vertices in the two sets of a maximal biclique. p and q are the thresholds for each of the biclique vertex sets [29]. Prior to this work none of previous algorithms have been tested across these large real-world datasets (iMBEA used bioinformatics datasets and LCM-MBC used maximum of 4904 vertices and 17404 edges). An essential finding from these experiments



(a) Github(q=12)

(b) Github(p=10)

(c) Github(p=q)

(d) Youtube(q=8)

(e) Youtube(p=8)

(f) Youtube(p=q)

Figure 3.5: Large real-world datasets with thresholds

is the superiority of the PMBE over previous algorithms across the entire domain of the datasets.

3.5 Summary

The problem of enumerating all maximal bicliques from a bipartite graph has been explored in this chapter. We propose the PMBE algorithm which incor-

porates the pivot pruning using the CDAG structure and the *rev-topological* ordering is proposed for heuristic pivot selection. The PMBE proposed contains advanced pruning and ordering techniques, which reduced the search space without introducing any local extra cost. We conducted extensive experiments on real datasets across various domains to demonstrate the superiority of PMBE over the previous algorithms.

Chapter 4

Searching Personalized k-wing in Bipartite Graphs

4.1 Introduction

There are extensive studies focusing on the application scenario that all the bipartite cohesive subgraphs need to be discovered in a bipartite graph. However, we observe that, for some applications, one is interested in finding bipartite cohesive subgraphs containing a specific vertex. In this chapter, we study a new query-dependent bipartite cohesive subgraph search problem based on k-wing model, named as personalized k-wing search problem. We study the k-wing equivalence relationship to summarize the edges of a bipartite graph G into groups. Therefore, all the edges of G are segregated into different groups, i.e. *k-wing equivalence class*, forming an efficient and wing number conserving index called *EquiWing-Graph*. Further, we propose a more compact index, *EquiWing-Tree*, which is achieved by using our proposed *k-butterfly loose* approach and discovering hierarchy properties. These indices are used to expedite the personalized k-wing search with a non-repetitive access to G, which leads to

linear algorithms for searching the personalized k-wing. Moreover, we conduct a thorough study on the maintenance of the proposed indices for evolving bipartite graphs. We discover novel properties that help us localize the scope of the maintenance at a low cost. By exploiting the discoveries, we propose novel algorithms for maintaining the two indices, which substantially reduces the cost of maintenance. We perform extensive experimental studies in real-world graphs to validate the efficiency and effectiveness of *EquiWing-Graph* and *EquiWing-Tree* compared to the baseline.

Chapter map. The remaining chapter is organized as follows. Section 4.2 discusses the preliminaries and problem definition. Section 4.3 introduces a baseline approach based on the bitruss decompostion. In Sections 4.4 and 4.5, we discuss the indexing schemes *EquiWing-Graph* and *EquiWing-Tree* for personalized k-wing search. Section 4.6 presents an efficient algorithm to maintain the proposed indices. Section 4.7 presents the experimental evaluation of all the algorithms. Section 4.8 summarizes the chapter.

4.2 Preliminaries and Problem Definition

In this section we formally introduce a *k-wing* by firstly defining a butterfly.

Definition 4.1. *Butterfly ($\bowtie$): Given a bipartite graph $G = (U, V, E)$ and four vertices $u, w \in U, v, x \in V$, a butterfly induced by u, v, w, x is a cycle of length of 4 consisting of edges (u, v), (w, v), (w, x) and $(u, x) \in E$.*

For a given bipartite graph in Fig. 4.1 edges (u_1, v_1), (u_1, v_2), (u_2, v_2) and (u_2, v_1) form a butterfly $\bowtie_{u_1v_1u_2v_2}$.

Butterfly support ($\bowtie(e, G)$). It is the number of butterflies in G containing an edge e and is denoted as $\bowtie(e, G)$. We replace $\bowtie(e, G)$ with $\bowtie(e)$ whenever the context is obvious.

Butterfly adjacency. Given two butterflies $\bowtie_1$ and $\bowtie_2$ in G, $\bowtie_1$ and $\bowtie_2$ are butterfly adjacent if $\bowtie_1 \cap \bowtie_2 \neq \emptyset$.

Butterfly connectivity ($\Leftrightarrow$). Given two butterflies $\bowtie_s$ and $\bowtie_t$ in G, $\bowtie_s$ and $\bowtie_t$ are butterfly connected if there exists a series of butterflies $\bowtie_1, \ldots, \bowtie_n$ in G, in which $n \geq 2$ such that $\bowtie_s = \bowtie_1$, $\bowtie_t = \bowtie_n$ and for $1 \leq i < n$, $\bowtie_i$ and $\bowtie_{i+1}$ are butterfly adjacent.

In Fig. 4.1, the edge (u_2, v_2) is contained in $\bowtie_{u_1v_1u_2v_2}$, $\bowtie_{u_2v_2u_4v_3}$ and $\bowtie_{u_2v_2u_3v_3}$ thus its *butterfly support* $\bowtie((u_2, v_2), G) = 3$. Also, $\bowtie_{u_1v_1u_2v_2}$ and $\bowtie_{u_2v_2u_3v_3}$ are said to be *butterfly connected* as $\bowtie_{u_1v_1u_2v_2} \cap \bowtie_{u_2v_2u_3v_3} = (u_2, v_2)$. $\bowtie_{u_1v_1u_2v_2}$ and $\bowtie_{u_3v_2u_4v_3}$ are *butterfly connected* through $\bowtie_{u_2v_2u_3v_3}$ in G.

Next, we define k-*wing bipartite cohesive subgraphs* [17].

Definition 4.2. *k-wing: A bipartite subgraph $H = (U, V, E) \subseteq G$ is a k-wing if it satisfies the following conditions:*

1. *Minimum butterfly constraint: $\forall\ e \in E(H)$, $\bowtie(e, H) \geq k$.*
2. *Butterfly Connectivity: $\forall\ e_1,\ e_2 \in E(H)$, $\exists\ \bowtie_1$ and $\bowtie_2 \in H$ such that $e_1 \in \bowtie_1$, $e_2 \in \bowtie_2$, then either $\bowtie_1 = \bowtie_2$ or $\bowtie_1$ and $\bowtie_2$ are butterfly connected.*
3. *Maximality: There is no $H' \subseteq G$ such that H' satisfies the above two conditions while $H \subset H'$.*

Further, we define the *wing number* for an edge as follows.

Definition 4.3. *Wing number ($\psi(e)$): For an edge $e \in E$, $\psi(e)$ is the maximum possible k such that there exists a k-wing in G containing e.*

Note, given $e \in E(G)$, $\psi(e) \leq \bowtie(e, G)$. For instance, in Fig. 4.1, the wing number of (u_2, v_2) is 2. Whereas its support in the graph is 3.

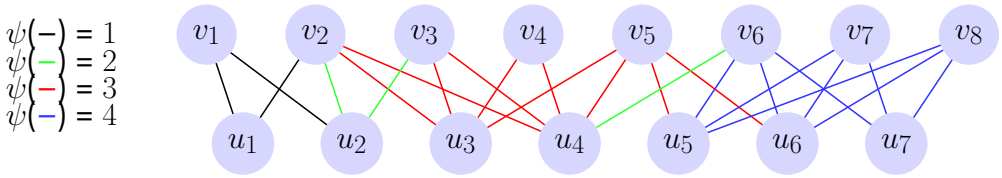


Figure 4.1: Wing number for the edges in G

Example 4.1. *Fig. 4.1 shows all the edges e in the bipartite graph G, with their respective wing number ψ. We observe a 4-wing in Fig. 4.1 (blue edges) and verify that every edge in a 4-wing is contained in at least 4 butterflies, any two edges in a 4-wing are reachable through adjacent butterflies, and 4-wing is maximal as well. We can also observe that the edges (u_4, v_5) and (u_5, v_5) are having the same wing number of 3. However, they belong to two different 3-wings since they are unreachable through adjacent butterflies,*

Further, to understand the personalized k-wing, we consider a scenario of a given user and determine the crowd that could exclusively provide funds for a user's specific project. Crowdfunding has emerged as one of the new opportunities for entrepreneurs to collect funds via a group of users, mainly on the social media platform (Twitter), who are interested in investing in their projects. [36] and [216] indicated that achieving the fundraising goal for a project is more correlated to the strength of connectivity among the users in a project, i.e., strongly connected users, rather than the number of users in a user-tweet network. Fig. 4.2 represents a user-tweet network. Each edge between user and tweet would

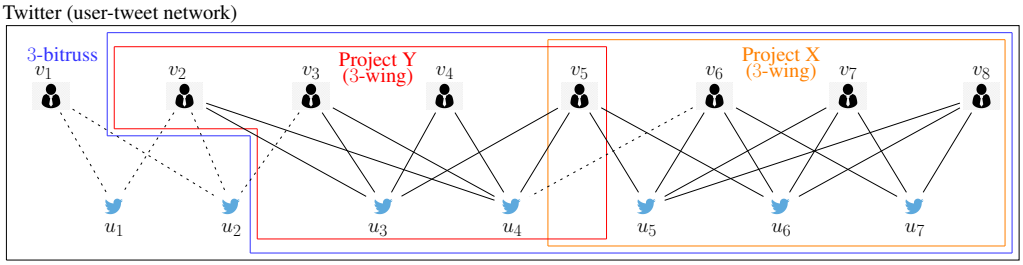


Figure 4.2: *Finding interesting projects (personalized k-wing) for crowdfunding in user-tweet networks*

represent a user's tweet containing a project's hashtag. There are two bipartite cohesive subgraphs, i.e., *Project X* and *Project Y*, in a user-tweet network. Therefore, we observe the user v_5 participates (tweets) in two different projects using 5 tweets. The k-*wing* model could successfully determine *Project X* and *Project Y* and their exclusive users who can participate in their respective crowdfunding. This is because it possesses sufficient strength of connectivity, i.e., butterfly connectivity. Unfortunately, the other bipartite cohesive subgraph models such as *(α, β)-core* and k-*bitruss* do not possess sufficient strength of connectivity and are likely to group the users with the uninteresting projects (tweets), i.e., v_1, v_2, v_4 with u_5, u_6, u_7.

We now propose the following personalized k-wing search problem.

Problem Definition. Given a bipartite graph $G = (U, V, E)$, a query vertex $q \in U \cup V$ and an integer $k \geq 1$, we return all k-*wings* containing q.

4.3 Baseline Approach

In this section, we describe the importance of *bitruss decomposition* and a baseline approach to address the problem.

Observing the wing number for an edge is the same as its bitruss number, the baseline solution applies the state-of-the-art bitruss decomposition algorithm for computing the wing number for every edge offline and then performs an online search for answering a personalized k-wing query.

Offline wing number computation. The bitruss decomposition algorithm from [16] is used to compute the wing number for all the edges. The time complexity of the bitruss decomposition is $\mathcal{O}(d_{max}^2|E|)$ for a given bipartite graph $G = (U, V, E)$ where d_{max} is the maximum degree. However, we omit a detailed

discussion of [16] for the sake of brevity. Fig. 4.1 shows $\psi(e)$, $\forall e \in E$.

Online search. For a query vertex q and an integer k, the *BaseLine* approach starts the search with every edge e incident on q satisfying $\psi(e) \geq k$. Such incident edges act as the initialized seed edges and a BFS is performed starting with these seed-edges. For each edge e in the seed, *BaseLine* includes e to the partial result and expands the seed as follows. Using the edge e to be included as the partial result, the BFS considers every e_0 as a new seed-edge if e_0 satisfies the constraints: (i) $\psi(e_0) \geq k$; (ii) e_0 forms a butterfly with e (all edges in this butterfly has wing number no less than k); (iii) e_0 is neither in the seed nor in the partial result. The expansion stops when no edge can be further included into the partial result, i.e., there is no seed-edge left. Due to the constraints applied during the search, edges satisfying the butterfly connectivity can be grouped together inherently. As such, the set of k-*wings* containing q can be derived by *BaseLine* as the query result. The time complexity of *BaseLine* is $\mathcal{O}(d_{max}^2|E(H)|)$, where d_{max} is the maximum degree and $|E(H)|$ is the number of edges in the k-*wings* containing q. The extra cost of $\mathcal{O}(d_{max}^2)$ compared to the general BFS is induced by checking the butterfly connectivity constraint.

Limitations of the BaseLine approach. For any edge (u, v) in a butterfly the algorithm needs to check all the edges incident on u and v and see if they satisfy all the constraints. This leads to the following two unnecessary operations. (i) *Overhead of accessing ineligible edges:* during the search, if the wing number of an edge is less than k, it would not be included in the k-*wing*. Therefore an extra unnecessary overhead is required to check the ineligible edges. (ii) *Redundant access of eligible edges:* if an edge e is added into the partial result or seed, it is accessed at least extra $3 \times k$ times in the BFS, which is an overhead. This is because for each eligible edge e ($\psi(e) \geq k$) of a butterfly, it will be accessed three times while doing the BFS from the other three eligible edges in

the same butterfly.

4.4 Super Graph Based Index

The limitations of *BaseLine* can be addressed by using one-time quick access only to eligible edges. If we can summarize edges that exist in the same k wing subgraphs together into super nodes and use super edges to link super nodes for reflecting the required butterfly connectivity between groups of edges, we only need to access the summarized connected super nodes for retrieving k-wing subgraphs. As such the drawbacks of *BaseLine* can be addressed.

The above ideas have been studied under the context of the truss model for unipartite graphs [61] . In this section, we study on how to adapt the techniques in [61] to our studied wing model for bipartite graphs, which serves as an advanced baseline called *EquiWing-Graph*.

4.4.1 k-wing Equivalent Edges

In this sub-section, we discuss the properties for summarizing edges together based on their wing number, i.e., k-wing equivalence. We first adapt several definitions to the context of the k-wing model.

Definition 4.4. *k-butterfly: A butterfly $\bowtie_{uvwx}$ in G is a k-butterfly, if min $\{\psi((u,v)), \psi((u,x)), \psi((w,x)), \psi((w,v))\} \geq k$.*

Definition 4.5. *k-butterfly connectivity ($\overset{k}{\Longleftrightarrow}$): Given two k-butterflies $\bowtie_x$ and $\bowtie_y$ in G, they are k-butterfly connected if there exists a sequence of $n \geq 2$ k-butterflies: $\bowtie_1, ..., \bowtie_n$ s.t. $\bowtie_x = \bowtie_1$, $\bowtie_y = \bowtie_n$ and for $1 \leq i < n, \bowtie_i \cap \bowtie_{i+1} \neq \emptyset$ and $\exists e \in \bowtie_i \cap \bowtie_{i+1}$, $\psi(e) = k$.*

Example 4.2. *Consider the bipartite graph G in Fig. 4.1, and butterflies $\bowtie_{u_5v_6u_6v_7}$*

and $\bowtie_{u_6v_7u_7v_8}$. *They are 4-butterfly connected as they share a common edge with the wing number* 4, *i.e.* $\bowtie_{u_5v_6u_6v_7} \cap \bowtie_{u_6v_7u_7v_8} = \{(u_6, v_7)\}$ *and* $\psi((u_6, v_7)) = 4$.

Now, we are ready to define the k-*wing reachability* for a pair of $e_1, e_2 \in E$.

Definition 4.6. k-*wing reachability: Given any two edges* $e_1, e_2 \in E$, *they are* k-*wing reachable, denoted by* $e_1 \overset{\bowtie^k}{=} e_2$, *if (1)* $\psi(e_1) = \psi(e_2) = k$, *and (2)* $\exists \bowtie_1$ *and* $\bowtie_2 \in H$ *such that* $e_1 \in \bowtie_1$, $e_2 \in \bowtie_2$, *then* $\bowtie_1 \overset{k}{\Longleftrightarrow} \bowtie_2$.

k**-wing equivalent edge set**. We can group all edges satisfying k-wing reachability together. Formally, given an edge $e \in E, \psi(e) = k$, the set of edges $\mathbb{C}_e = \{x | x \overset{\bowtie^k}{=} e, x \in E\}$ is the equivalence class of e. These edges have the equivalence relationship, i.e., for every edge, every pair of edges, and any triple edges in $\mathbb{C}_e$, the properties of reflexivity, symmetry, and transitivity should be satisfied correspondingly. For the reflexive property, consider an edge $e_1 \in \mathbb{C}_e$. By definition 4.2, there exists at least one subgraph $G' = (U', V', E') \subseteq G$ s.t. $e_1 \in E'$, and $\forall e \in E'$, $\psi(e) \geq k$. Since there exist at least one k-*butterfly* $\bowtie \subseteq G'$ s.t. $e_1 \in \bowtie$. That is, $e_1 \overset{\bowtie^k}{=} e_1$. For the symmetric property, consider a pair of two edges $e_1, e_2 \in \mathbb{C}_e$ such that $e_1 \overset{\bowtie^k}{=} e_2$. Either of the following cases holds: (1) e_1 and e_2 are in the same k-*butterfly*; (2) there exist two k-*butterflies* $\bowtie_1$ and $\bowtie_2$, such that $e_1 \in \bowtie_1$, $e_2 \in \bowtie_2$, and $\bowtie_1 \overset{k}{\Longleftrightarrow} \bowtie_2$. For the first case, $e_2 \overset{\bowtie^k}{=} e_1$ clearly. For the second case, since k-*butterfly connectivity* is symmetric, $\bowtie_2 \overset{k}{\Longleftrightarrow} \bowtie_1$, i.e., $e_2 \overset{\bowtie^k}{=} e_1$. For the transitive property, consider three edges e_1, e_2, $e_3 \in \mathbb{C}_e$, s.t. $e_1 \overset{\bowtie^k}{=} e_2$ and $e_2 \overset{\bowtie^k}{=} e_3$. One of the following cases holds: (1) there exist two k-*butterflies* $\bowtie_1$ and $\bowtie_2$, s.t. e_1, $e_2 \in \bowtie_1$ and e_2, $e_3 \in \bowtie_2$. If e_1 and e_3 are located in the same k-*butterfly*, then $e_1 \overset{\bowtie^k}{=} e_3$. Or else, $\bowtie_1 \cap \bowtie_2 = \{e_2\}$ and $\psi(e_2) = k$, so $\bowtie_1 \overset{k}{\Longleftrightarrow} \bowtie_2$. Hence, $e_1 \overset{\bowtie^k}{=} e_3$; (2) there exist s k-*butterflies* $\bowtie_{x_1}, \ldots, \bowtie_{x_s}$ in G, s.t. $e_1 \in \bowtie_{x_1}$, $e_2 \in \bowtie_{x_s}$, and all the edges joining these s consecutive k-*butterflies* are with the same *wing number*,

k. Meanwhile, there exist t k-*butterflies* $\bowtie_{y_1}, \ldots, \bowtie_{y_t}$ in G s.t. $e_2 \in \bowtie_{y_1}$, $e_3 \in \bowtie_{y_t}$ and all the edges joining these t k-*butterflies* are with the same *wing number*, k. If $\bowtie_{x_s} = \bowtie_{y_1}$, we know that $\bowtie_{x_1} \overset{k}{\Longleftrightarrow} \bowtie_{y_t}$ through a series of $(s + t - 1)$ adjacent k-*butterflies* $\bowtie_{x_1}, \ldots, \bowtie_{x_s}, \ldots, \bowtie_{y_t}$. Or else, we know that $\bowtie_{x_s} \cap \bowtie_{y_1} = e_2$ and $\psi(e_2) = k$, so $\bowtie_{x_1} \overset{k}{\Longleftrightarrow} \bowtie_{y_t}$ through a series of $(s + t)$ adjacent k-*butterflies* $\bowtie_{x_1}, \ldots, \bowtie_{x_s}, \bowtie_{y_1}, \ldots, \bowtie_{y_t}$. Hence, $e_1 \overset{\bowtie^k}{=} e_3$.

The above discussion focuse on a particular k, which can be extended to each k. That is, each equivalence class $\mathbb{C}_e$ is composed of edges with the same wing number, k, that are k-*butterfly connected*. Therefore an equivalence class $\mathbb{C}_e$ forms the basic unit for our personalized k-wing search.

Next we will show to how use the defined equivalent edge sets to build a super graph serving as an index for searching k-wing subgraphs.

4.4.2 EquiWing-Graph

We first define the *EquiWing-Graph* structure using the k-wing equivalent edge sets. Secondly, we devise the construction algorithm to develop the index. Thirdly, we propose an algorithm for the personalized k-wing search using *EquiWing-Graph*. Last but not least, we also discuss the time and space complexities of algorithms for index construction and k-wing search.

EquiWing-Graph. The idea for *EquiWing-Graph* is to exploit the k-*wing* relationships for any possible k wherein we form the equivalence class and summarize the given bipartite graph G to a super graph $EW\text{-}G\ (\chi, \Upsilon)$. To distinguish $EW\text{-}G$ from G, we use the term super node to represent a $\nu \in \chi$. Each of the super nodes $\nu \in \chi$ represents a separate equivalence class $\mathbb{C}_e$ where $e \in E$, and a super edge $(\mu, \nu) \in \Upsilon$, where $\mu, \nu \in \chi$ represents the two equivalence classes which are *butterfly connected*.

Index construction. Given a bipartite graph G, Algorithm 4 is the pseudo code

Algorithm 4: EquiWing-Graph Construction

Input : $G = (U, V, E)$
Output: Index : $EW\text{-}G\ (\chi, \Upsilon)$

1 **Function** `Index Construction (G=(U,V,E))`
2 $bitruss\_Decomposition(G)$;
3 $initialize\_all\_edges()$;
4 **for** $k \longleftarrow 1$ *to* k_{max} **do**
5 **while** $\exists e \in \mathcal{E}_k$ **do**
6 $e.visited = TRUE$;
7 Create a super node ν with $\nu.snID \leftarrow ++ snID$;
8 $\chi \leftarrow \chi \cup \{\nu\}$ or $\{\chi[k] \leftarrow \chi[k] \cup \{\nu\}\}$;
9 $Q.enqueue(e)$;
10 **while** $Q \neq \varnothing$ **do**
11 $x(u, v) \leftarrow Q.dequeue()$; $\nu \leftarrow \nu \cup \{x\}$;
12 **foreach** $id \in x.list$ **do**
13 Create an edge (μ, ν) where μ is an existing super node with $\mu.snID = id$.
14 $\Upsilon \leftarrow \Upsilon \cup \{(\mu, \nu)\}$
15 **foreach** e' *edges incident on* x **do**
16 **if** $\psi(e') \geq k$ **then**
17 $ButterFly(x, e', Q)$
18 **return** $EW\text{-}G\ (\chi, \Upsilon)$;
19 **Function** `ButterFly` (x, e', Q)
20 **foreach** e'' *edges incident on* x *and* e' **do**
21 **if** $\psi(x') \geq k$ *and* $\psi(e'') \geq k$ **then**
/* e', e'', x, x' form a k-butterfly. */
22 Process(e'', Q);/* Invoke Process, for x' and e' */
23 **Function** `Process` (e''', Q)
24 **if** $\psi(e''') = k$ **then**
25 **if** $e'''.visited = FALSE$ **then**
26 $e'''.visited = TRUE$; $Q.enqueue(e''')$;
27 **else**
28 **if** $snID \notin e'''.list$ **then**
29 $e'''.list \leftarrow e'''.list \cup snID$

to construct the *EquiWing-Graph* index. We start the algorithm by computing the wing number for $\forall e \in E$ (line 2), then we initialize the values to the attributes of those edges (*visited*: Boolean type, symbolizes whether the edge has been examined; *list*: a set of super nodes, all the super nodes which have already been explored and are connected via series of k-*butterflies* to the current super

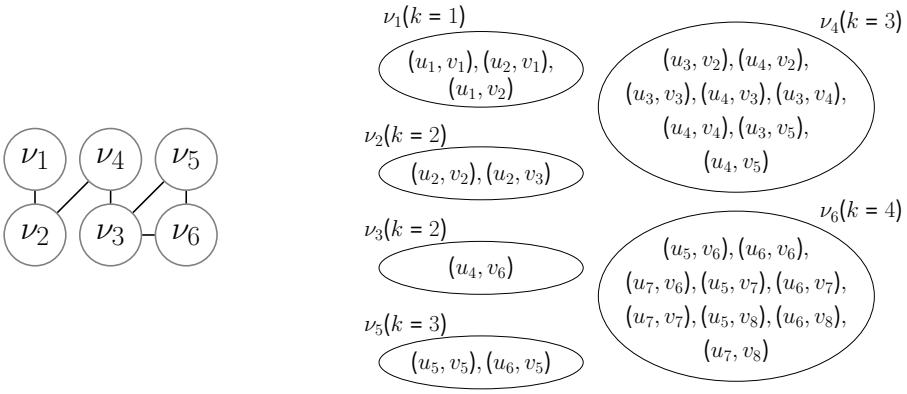


Figure 4.3: k-wing equivalence based index, EquiWing-Graph

node) and segregate the edges in distinct sets ($\mathcal{E}_k$) based on their wing number k (line 3). We then examine all the edges $e \in E$, using the sets $\mathcal{E}_k$, from $k = 1$ to $k = k_{max}$ (line 4). For the selected edge $e \in \mathcal{E}_k$, we create a new super node ν for its corresponding equivalence class (line 7-8). To explore all the edges within the same equivalence class $\mathbb{C}_e$, we perform BFS using e as a starting edge (line 10-17). During the exploration for the edge e, we also check for any previously explored $\mathbb{C}'_e$ in $e.list$, if so we form the super edges between the super nodes (line 12-14). Meanwhile, throughout the BFS, $\forall e : \psi(e) > k$, the edge e stores the current super node ($snID$) in $e.list$ (line 28-29).

Example 4.3. *The EquiWing-Graph of the bipartite graph in Fig. 4.1 is shown in Fig. 4.3. It contains 6 super nodes and each super node corresponds to a k-wing equivalence class in which an edge of G participates. We observe in Fig. 4.3 tabulated form, that all the edges from G are contained in EquiWing-Graph, within super nodes. For example, ν_4 represents 8 edges which are 3-butterfly connected edges in G, and form a 3-wing. Moreover, we also observe that EquiWing-Graph contains 6 super edges depicting the butterfly connectivity between super nodes.*

Time complexity. Algorithm 4 performs BFS to enumerate all the *butterflies* for each edge. Moreover, each edge needs to be checked for k-*butterfly connectivity*,

Algorithm 5: k-wing search using EquiWing-Graph

Input : $EW\text{-}G(\chi, \Upsilon)$, q and hash structure $H(q)$.
Output: $\mathbb{W}$: all k-wings containing q.

Function `EW-G_search` ($EW\text{-}G(\chi, \Upsilon)$,*q*,*H(q)*)
  $mark\_all\_nodes\_false()$; $l \longleftarrow 0$;
  `/* Perform BFS on EquiWing-Graph using seeds. */`
  foreach $\nu \in H(q)$ **do**
    if $\psi(\nu) \geq k$ *and* $\nu.visited = FALSE$ **then**
      $Q \longleftarrow \varnothing$; $Q.enqueue(\nu)$; $l \longleftarrow l + 1$;
      $\nu.visited = TRUE$; $\mathbb{W}_l \longleftarrow \varnothing$;
      while $Q \neq \varnothing$ **do**
        $\nu \longleftarrow Q.dequeue()$; $\mathbb{W}_l \longleftarrow \mathbb{W}_l \cup \{e | e \in \nu\}$;
        foreach $(\nu, \mu) \in \Upsilon$ **do**
          if $\psi(\mu) \geq k$ *and* $\mu.visited = FALSE$ **then**
            $\mu.visited = TRUE$; $Q.enqueue(\mu)$;
  return $\{\mathbb{W}_1, ..., \mathbb{W}_l\}$;

which takes $\mathcal{O}(d^2_{max})$. Therefore, the total time complexity of Algorithm 4 is $\mathcal{O}(d^2_{max}|E(G)|)$.

Space complexity. For an edge $e \in E$, $|e.list|$ is the number of super nodes which are butterfly connected to ν, where $e \in \nu$. Therefore, $e.list$ can take at most $\mathcal{O}(|E(G)|)$ space. This memory is free once e is processed. Hence, the space complexity of Algorithm 4 is $\mathcal{O}(|E(G)|)$.

Note that during the creation of the *EquiWing-Graph* all the k-*wings* are completely retained, and the $\Leftrightarrow$ between two k-*wings* is also maintained through the super edges. Therefore, we consider *EquiWing-Graph* as self-sufficient to identify all the k-*wings* as it contains all the critical information.

Query processing. Algorithm 5 represents the pseudo code for processing a personalized k-wing search query using *EquiWing-Graph*. To start our search we first find the super node which contains the query vertex q. This is achieved by maintaining a hash structure ($H(q)$) which has the vertex as its key and the values correspond to the list of super nodes ν containing the vertex q. This

structure can be built along with *EquiWing-Graph* construction. Algorithm 5 starts with a super node $\nu \in H(q)$ with $\psi(\nu) \geq k$, then explores all the neighbors μ using BFS in *EquiWing-Graph* s.t. $\forall \mu \in \Gamma(\nu), \psi(\mu) \geq k$. In the end, all the super nodes connected to ν via k-*butterflies* form a k-*wing* stored in $\mathbb{W}_l$. The exploration continues until all $\nu \in H(q)$ are visited. Algorithm 5 accesses only the edges in the super node ν where where $\psi(\nu) \geq k$ and each edge is accessed only once when reported as output in $\mathbb{W}_i$. Therefore, the time complexity of Algorithm 5 is $\mathcal{O}(|E(H)|)$, where $|E(H)|$ is the number of edges in the resulting k-*wings*.

Example 4.4. *Fig. 4.4 shows the k-wing search for the query vertex $q = v_5$ and $k = 3$. We first locate the super nodes $H(v_5)$, i.e., ν_4 and ν_5. Starting with ν_4, we firstly verify $\psi(\nu_4) \geq 3$. Since it satisfies the condition, we add all the edges of ν_4 to $\mathbb{W}_1$, then we explore the neighbors of ν_4. We find that none of $\mu \in \Gamma(\nu_4)$ has $\psi(\mu) \geq 3$, therefore, we report $\mathbb{W}_1$ as the first k-wing and move to ν_5. Similarly, we explore ν_5 and $\Gamma(\nu_5)$ and report $\mathbb{W}_2$ as the second k-wing. Hence, the query result is $\mathbb{W} = \{\mathbb{W}_1, \mathbb{W}_2\}$.*

4.5 Super Tree Based Index

The *EquiWing-Graph* is an efficient index, however, there are some limitations. Firstly, unnecessary segregation exists for super nodes which always occur together in a k-*wing*. Secondly, the unexplored hierarchical properties of k-*wing* can be used to reduce the index size. Therefore, to address the two drawbacks, we propose the notion of k-*butterfly loose connectivity* and exploit the hierarchical property among k-*wings* to form a more compact index with tree structure.

4.5.1 Properties for Building Super Trees

Observations. The notion of *k-butterfly connectivity* is used for summarizing a bipartite graph to form *EquiWing-Graph*, however, it is too strict which results in super nodes that can be merged together. For example, in Fig. 4.3, ν_2 and ν_3 will participate in the same *k-wing*, thus can be combined. As a result, we propose a relaxed version of *k-butterfly connectivity*, i.e., *k-butterfly loose connectivity*, as follows.

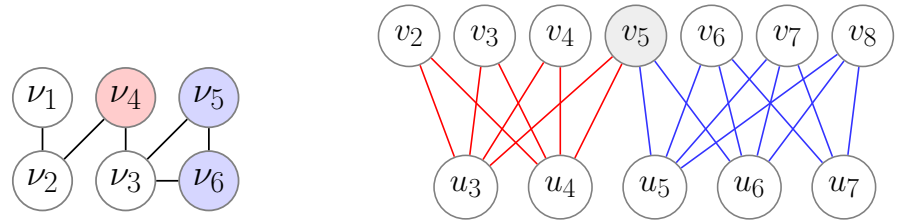
Definition 4.7. *k-butterfly loose connectivity ($\overset{\geq k}{\Longleftrightarrow}$): Given two super nodes ν_x and ν_y, such that $\psi(\nu_x) \leq \psi(\nu_y)$ and both $\psi(\nu_x$ and $\psi(\nu_y)$ are no less than k in EquiWing-Graph, ν_x and ν_y are k-butterfly loose connected if there exists a sequence of $n > 2$ connected super nodes $\nu_1, ..., \nu_n$ s.t. $\nu_x = \nu_1$, $\nu_y = \nu_n$ and $\forall \nu_i, \psi(\nu_i) \geq \psi(\nu_x)$.*

Next we show two theorems for reorganizing super nodes.

Theorem 4.1. *If any two super nodes $\nu, \mu \in \chi$ are k-butterfly loose connected and $\psi(\nu) = \psi(\mu)$. Then ν and μ can be combined together.*

Proof. Let there be two super nodes $\nu, \mu \in \chi$, with $\psi(\nu) = \psi(\mu) = k$. Using Definition 4.2, a *k-wing* for ν will always include all connected series of super nodes ν_i, with $\psi(\nu_i) \geq k$. Consequently, it will also include super nodes with $\psi(\nu_i) = k$, i.e., μ, connected via series of super nodes $\nu_1, \nu_2, ..., \nu_i$, where $\forall \psi(\nu_i) \geq k$. Using Definition 4.7, we deduce that ν and μ are k-butterfly loose connected. We can also observe that ν and μ will always occur together in the resulting *k-wing*. Therefore, we conclude that combining them will make the index compact without compromising its correctness. □

For example, Fig. 4.3, super node $\nu_2 \overset{\geq k}{\Longleftrightarrow} \nu_3$. ν_2 is connected to ν_3 via ν_4 and $\psi(\nu_4) \geq \psi(\nu_2) = \psi(\nu_3) = 3$. As such ν_2 and ν_3 can be combined.



(a) EquiWing-Graph

(b) 3-wings for v_5

Figure 4.4: 3-wing search for v_5 using EquiWing-Graph

Theorem 4.2. *If any two super nodes $\nu, \mu \in \chi$ are k-butterfly loose connected. Then ν and μ belong to the same connected k-wing subgraph.*

The correctness of Theorem 4.2 is clear by the definitions of *EquiWing-Graph* and *k-butterfly loose connectivity*.

Exploring the hierarchy property. The hierarchy property states that a *$(k+1)$-wing* is a subgraph of *k-wing* [170]. The super nodes in the index *EquiWing-Graph* can be segregated into different hierarchy levels based on the ψ. Fig. 4.5 represents the levels in the *EquiWing-Graph*. By incorporating the hierarchy relationship, the above two theorems and *EquiWing-Graph*, we are ready to propose a tree strictured index in the following subsection.

4.5.2 EquiWing-Tree

In this section, we propose a tree structure index, *EquiWing-Tree*.

Index construction. Logically, the index construction consists of the following main steps.

Step 1: Align super nodes to different hierarchical levels. We utilize the *EquiWing-Graph* construction Algorithm 4, and first classify the super nodes into different hierarchy levels by updating χ to $\chi[k]$.

Step 2: Merge super nodes at the same level. We exploit Theorem 4.1 to merge the super nodes with the same ψ value when necessary in *EquiWing-Graph*. Merging of two super nodes leads to super edges existing only between two

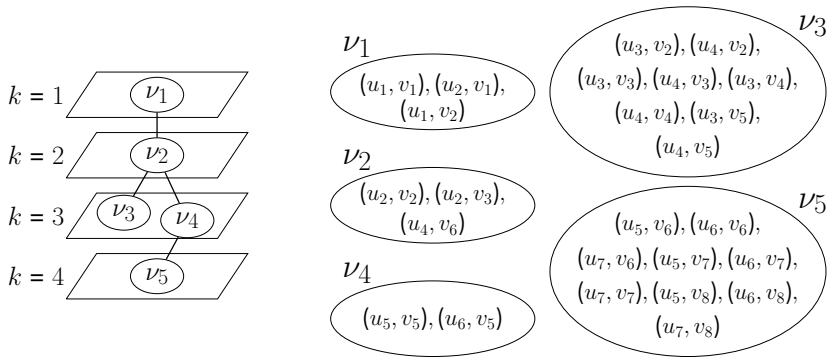


Figure 4.5: EquiWing-Tree

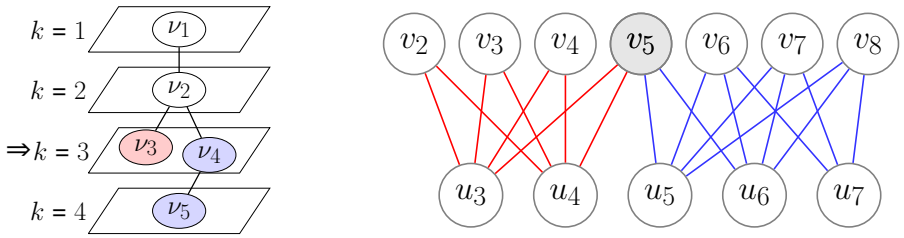
nodes in different hierarchy levels. Notice that after this step, the *EquiWing-Graph* can be treated as a *directed acyclic graph* by defining the direction of an edge as (ν, μ) if $\psi(\nu) < \psi(\mu)$ and vice versa.

Step 3: Linking super nodes belonging to the same k-wing. We create edges for two super node if 1) they belong to the same k-wing (Theorem 4.2) and 2) the created edge is not a transitive edge based on the edge direction defined above.

Step 4: Transitive reduction. We conduct a standard transitive reduction to the directed acyclic graph to reduce to edges much as possible.

The tree structured index. Now we discuss that after the index construction, the super graph becomes a tree structure. The first three steps ensure that super nodes with different wing numbers belonging to the same connected k-wing are reachable via directed edges, where edge directions capture k-wing hierarchical structure. By the definition of k-wing hierarchical structure, a super node ν should be linked from only one μ node such that $\psi(\mu) < \psi(\nu)$ if μ exists, which can be achieved by Step 3.

Time complexity. The time complexity for creating *EquiWing-Tree* is divided into two parts: (i) creating *EquiWing-Graph* that is $\mathcal{O}(d_{max}^2|E(G)|)$ and (ii) perform the discussed four steps dominated by the 4th step running in $\mathcal{O}(|\chi|^{2.3729})$ [217]. Therefore, the total construction time for *EquiWing-Tree* is $\mathcal{O}(d_{max}^2|E(G)|)$. The construction of *EquiWing-Tree* requires an extra space for combining super



(a) k-wing search (b) 3-wings for vertex v_5

Figure 4.6: The two 3-wings for the query vertex v_5

nodes, which is not the dominating part. Hence the space complexity of *EquiWing-Tree* construction is $\mathcal{O}(|E(G)|)$.

Example 4.5. *The EquiWing-Tree for the bipartite graph G is shown in Fig. 4.5. We observe that all the edges from EquiWing-Graph are contained in EquiWing-Tree. In Fig. 4.3, $\nu_2 \overset{\geq k}{\Longleftrightarrow} \nu_3$ and they have the same ψ value, hence, they are combined. In Fig. 4.5, ν_2 contains all the edges from ν_2 and ν_3 of EquiWing-Graph, which can be seen in the tabulated form in Fig. 4.5.*

Query processing. For the given query vertex q, the *EquiWing-Tree* index created can be used to process the personalized k-wing search query using the hash structure H which provides the list of super nodes $H(q)$ containing the vertex q. The index *EquiWing-Graph* (EW-G) can be replaced by *EquiWing-Tree* (EW-T) in Algorithm 5 to perform the k-wing search represented in Figure 4.6.

Complexity. Since we are only replacing index in Algorithm 5, the worst case time complexity remains $\mathcal{O}(|E(H)|)$. However, the time complexity of *EquiWing-Tree* reduces to $\mathcal{O}(|\chi|)$ from $\mathcal{O}(|\Upsilon| + |\chi|)$ in case of *EquiWing-Graph*, since *EquiWing-Tree* is a tree structure. Besides, it is crucial to note the number of super nodes in *EquiWing-Tree* is less than *EquiWing-Graph* (detailed in Section 4.7.1).

Example 4.6. *Fig. 4.6 shows the k-wing search for the query vertex $q = v_5$*

and k = 3. We perform the search using Algorithm 5. The result obtain from EquiWing-Tree (Fig. 4.6(b)) is the same as of EquiWing-Graph (Fig. 4.4(b)).

4.6 Dynamic Maintenance of Indices

In this section, we examine the k-wing cohesive subgraph search in a dynamic graph using the proposed indices. The dynamic change of a graph refers to the insertion/deletion of edges and vertices. We primarily focus on the edge insertion/deletion since vertex insertion/deletion can be regarded as a set of edge insertions/deletions incident to the vertex that is inserted/deleted.

Let us consider the insertion of an edge $e'(u', v')$ in G. This could result in forming new butterflies. Due to a newly formed butterfly $\bowtie_{u'v'uv}$, the butterfly support for (u, v), (u, v') and (u', v) in $\bowtie_{u'v'uv}$ increases. The increase in butterfly support may lead to the increase in the wing number for those edges. Moreover, the affected edges may not necessarily limit to the edges that are incident on u' and v'. The edge deletion has a similar effect. Calculating the wing number from scratch is costly. Therefore, to handle the wing number update efficiently, the key is to identify the scope of the affected edges in the graph precisely. Moreover, due to the nature of the bipartite graph, when forming/breaking a butterfly $\bowtie_{u'v'uv}$ because of inserting/deleting an edge (u', v'), the butterfly support for edge (u, v) in $\bowtie_{u'v'uv}$ increases/decreases by 1 while the butterfly support for (u, v'), (u', v) in $\bowtie_{u'v'uv}$ could increase/decrease more than 1, which brings new challenges and makes the well studied theoretical results for core and truss maintenance inapplicable to wing maintenance. We conduct a novel study on the wing maintenance problem.

The increase in wing number due to the insertion of an edge requires to update the indices *EquiWing-Graph* and *EquiWing-Tree* accordingly. We mainly

focus on discussing edge insertion as edge deletion has a similar effect on decreasing wing number. We first identify the affected edges in G (Section 4.6.1), which in-turn is used to recognize the affected super nodes in our proposed indices (Section 4.6.2) and then design a dynamic update algorithm (Section 4.6.3). The dynamic maintenance of *EquiWing-Tree* is achieved using a similar approach, which is discussed at the end of the section.

4.6.1 Identification of Affected Edges

Let $\psi(e)$ and $\psi'(e)$ be the wing number of $e \in E$ before and after inserting/deleting an edge e'. Since updating G from scratch is not a viable option, we confine the scope of the affected edges in G. We now present the following two evident observations regarding the edge insertion/deletion:

1. **Observation 1**: if e' is inserted into G with $\psi'(e') = a$, then $\forall e \in E$ having $\psi(e) \geq a$, $\psi'(e) = \psi(e)$

2. **Observation 2**: if e' is deleted from G with $\psi(e') = a$, then $\forall e \in E \setminus \{e'\}$ having $\psi(e) > a$, $\psi(e)$ remains unaffected.

The Observations 1 and 2 can be justified clearly as in both the cases e' does not participate in $(a + 1)$-*wing*.

According to Observations 1 and 2, if $\psi'(e')$ is known, the scope of the affected edges can be identified clearly. However, computing the precise $\psi'(e')$ is expensive, which may explore the entire graph and the wing number of other affected edges can be updated when computing the precise $\psi'(e')$. This contradicts to the intention, applying updates on the affected edges only, of using Observations 1 and 2.

To speed up the update computation, instead of computing the precise $\psi'(e')$, we propose a novel upper bound for $\psi'(e')$, denoted as $\overline{\psi'(e')}$ that can be calcu-

lated at a low cost. Using $\overline{\psi'(e')}$, we can identify a scope that is slightly larger than the scope identified by $\psi'(e')$, and then update the wing number for edges in the $\overline{\psi'(e')}$ identified scope only.

To derive the upper bound $\overline{\psi'(e')}$, we first introduce the definitions and lemmas below.

Definition 4.8. *k-level butterflies: For an edge $e(u,v)$ and $k \geq 1$, k-level butterflies containing e are defined as $\bowtie_e^k = \{\bowtie_{uvu'v'} : \min\{\psi(e'')|e'' \in \{u,u'\} \times \{v,v'\} \setminus \{(u,v)\}\} \geq k\}$. The number of butterflies in $\bowtie_e^k$ is denoted as $|\bowtie_e^k|$.*

Before showing the lemmas below, we introduce a new notation $\delta(\cdot)$. It denotes the difference of the butterfly support/wing number of an edge after and before the insertion of an edge, i.e., $\delta(\psi(e)) = \psi'(e) - \psi(e)$ and $\delta(\bowtie(e)) = \bowtie'(e) - \bowtie(e)$.

Lemma 4.1. *After the insertion of e', $\forall e \in E(G) \setminus \{e'\}$, $\psi'(e) - \psi(e) \leq \Delta$, where $\Delta = \max\{\delta(\bowtie(e)) : e \in I(e')\}$, where $I(e')$ denotes the set of edges incident to e'.*

Proof. An upper bound for $\delta(\bowtie(e))$ can be used to bound $\delta(\psi(e))$. The increase of butterfly support ($\delta(\bowtie(e))$) for an edge can be categorised into two: (i) *non-incident edges w.r.t. e'* : the edge $e(u,v)$ does not share any vertex with e', hence it can only form 1 new butterfly, which means $\delta(\bowtie(e)) = 1$; (ii) *incident edges w.r.t. e'* : for an edge e sharing a vertex in $e' = (u',v')$, the fourth vertex for the butterfly is not fixed, and is bounded by the common neighbors of the two non-incident vertices of e and e'. Hence, the maximum value of $\delta(\bowtie(e))$ can be given as $|\Gamma(u') \cap \Gamma(u)| - 1$ if the two edges incident on v' or $|\Gamma(v') \cap \Gamma(v)| - 1$ if the two edges incident on u'. Therefore $\Delta = \max\{\{\delta(\bowtie(e))|e \in I(e')\}, \{1\}\}$. □

Using Definition 4.8 and Lemma 4.1, we can have an upper bound for $\psi'(e')$ that is let $l = \max\{k : |\bowtie_{e'}^k| \geq k\}$, then $l + \Delta$ is the upper bound. This upper

bound is loose since it assumes there are l number of l-level butterflies before the insertion of e' and assumes that the insertion makes the wing number of every edge in the l-level butterflies increase by Δ, which is quite unlikely. A tighter upper bound for $\psi'(e')$ that we use is proposed using the lemma below.

Lemma 4.2. *If an edge $e'(u', v')$ is inserted into G, then $\psi'(e') \leq h$ holds, where $h = \max\{k : |\bowtie_{e'}^{k-\Delta}| \geq k\}$.*

Proof. We prove $\psi'(e') \leq h$ by contradiction. Let $\psi'(e') = a > h$, then there exists an a-wing cohesive subgraph where all the edges have wing number no less than a, and e' would be contained by at least a butterflies. Since the wing number of all the edges except e' can increase at most by Δ after insertion, we have $|\bowtie_{e'}^{a-\Delta}| \geq a$ before insertion. This also implies that $h \geq a$ as per the definition of h, which is a contradiction. □

With the aid of Lemma 4.2, the upper bound $\overline{\psi'(e')}$ is given below.

Corollary 4.1. $\overline{\psi'(e')} = \max\{k : |\bowtie_{e'}^{k-\Delta}| \geq k\}$ *is an upper bound of $\psi'(e')$.*

Using $\overline{\psi'(e')}$, the relaxed scope of the affected edges can be derived, i.e., $\forall e$ such that $\psi(e) \leq \overline{\psi'(e')}$.

Next we propose the theoretical findings that can help us further exclude the edges in the relaxed scope that do not need to perform update computations. The intuition is given as follows. According to Lemma 4.1, although the insertion of e' may increase the wing number of some edges by Δ, it may also increase the wing number of some edges at most by 1 as well. This motivates us to study the identification of edges such that they even cannot increase the wing number by 1 after inserting e' and we exclude these edges for the update computations. Note that, due to Observation 1 at the beginning of Section 4.6.1, only the edges with wing number less than upper bound ($\psi'(e')$) will be affected. Now we introduce

Lemma 4.3 below for identifying the range of the affected edges other than the newly inserted e':

Lemma 4.3. *If an edge $e'(u', v')$ is inserted into G, we first calculate the value of $\overline{\psi'(e')}$. Then for all the edges $e_0(u_0, v_0) \in E \cup \{e'\}$ with $\psi(e_0) = a < \overline{\psi'(e')}$, the wing number of e_0 may be updated as $\psi'(e_0) \geq a + 1$, if and only if:*

1. *A new butterfly with edges of $\{u', v'\} \times \{u_0, v_0\}$ is formed, and $\min\{\psi((u', v_0)), \psi((v', u_0)), \psi((u', v'))\} > a - \Delta$; or*
2. *For any $e_0(u_0, v_0) \neq e'(u', v')$, $\exists \bowtie_{u_0 v_0 u v}$ such that $\min\{\psi((v_0, u)), \psi((u, v)), \psi((v, u_0))\} > a - \Delta$ holds.*

Proof. Let there be an edge $e_0(u_0, v_0)$ with $\psi(e_0) = a$ and $\psi'(e_0) = a + 1$. The easiest case to satisfy is below. Before the insertion, e_0 only needs to involve $(a+1)$ number of $(a-\Delta+1)$-level butterflies and after the insertion, the butterfly support of all the edges in these $(a - \Delta + 1)$-level butterfly may increase by Δ, which makes them become $(a + 1)$-level butterflies and increases the wing number for edge e_0 from a to $a+1$. Then we prove that only the edges satisfying cases (1) and (2) can increase their wing number.

For case (1), due to the insertion of $e'(u', v')$, the butterfly support for (u', v_0), (v', u_0), and (u', v') could increase by Δ. As such, after the insertion $\min\{\psi((u', v_0)), \psi((v', u_0)), \psi((u', v'))\}$ can be increased to a value no less than $a + 1$ and it is also possible that there are at least $(a + 1)$ number of $(a + 1)$-level butterflies, which makes the value of $\psi'(e_0)$ at least $a + 1$.

For case (2), since $\min\{\psi((v_0, u)), \psi((u, v)), \psi((v, u_0))\}$ could be increased by Δ after the insertion, $\min\{\psi((v_0, u)), \psi((u, v)), \psi((v, u_0))\}$ can be increased to a value in $[a+1, \overline{\psi'(e')})$ and it is also possible that there would be at least i number of i-level butterflies ($a + 1 \leq i < \overline{\psi'(e')}$), which makes the value of $\psi'(e_0)$ at least $a + 1$.

Next consider an edge $e_0(u_0, v_0) \in E \cup \{e'\}$ with $\psi(e_0) = a$, if it does not satisfy case (1), there is no new formed butterfly containing e_0 to account for the increase in the wing number. Hence, it is impossible to have $\psi'(e_0) \geq a$. If it does not satisfy case (2), then the existing butterfly $\bowtie_{u_0v_0uv}$ is with either $\min\{\psi'((u_0, v)), \psi'((v, u)), \psi'((u, v_0))\} < a - \Delta + 1$ or $\min\{\psi'((u_0, v)), \psi'((v, u)), \psi'((u, v_0))\} \geq \overline{\psi'(e')}$. After the insertion we respectively have $\min\{\psi'((u_0, v)), \psi'((v, u)), \psi'((u, v_0))\} < a + 1$ or $\min\{\psi'((u_0, v)), \psi'((v, u)), \psi'((u, v_0))\} \geq \overline{\psi'(e')} + \Delta$. For the first situation, after the insertion, there would be no sufficient number of $(a + 1)$-level butterflies for e_0 for increasing its wing number. The second situation is out of the scope of this lemma and the wing number would not be affected due to the insertion. Thus if e_0 satisfies neither case (1) nor (2), it is impossible to have $\psi'(e_0) \geq a + 1$. □

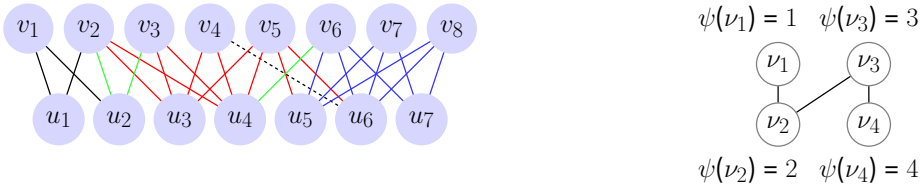
Based on Lemma 4.3 and the proposed upper bound, we refine the affected edge scope due to edge insertion/deletion as follows.

Refined scope of the affected edges. Using Corollary 4.1 and Lemma 4.3, we now summarize the edges whose wing number could be affected by the insertion/deletion of the edge e':

1. **Insertion case.** For $e \in E \cup \{e'\}$ with $\psi(e) < \overline{\psi'(e')}$, if e and e' form a new k-butterfly, or e is connected to e' via a series of adjacent k-butterflies, i.e., $e \overset{k}{\Longleftrightarrow} e'$, then e may have $\psi'(e) \geq \psi(e)$.

2. **Deletion case.** For $e \in E \setminus \{e'\}$ with $\psi(e) \leq \psi(e')$, if e and e' belong to a k-butterfly, or e is connected to e' via a series of adjacent k-butterflies, i.e., $e \overset{k}{\Longleftrightarrow} e'$ before the deletion, then e may have $\psi'(e) \leq \psi(e)$.

Note that, for the deletion case, using Lemma 1, we can say that for an affected edge e, $\delta(\psi(e)) \leq \Delta$ holds. Thus, we have a lower bound for the wing number of the affected edge e as $\psi'(e) \geq \psi(e) - \Delta$.



(a) Edge Insertion (b) Updated EW-G

Figure 4.7: Inserted new edge (v_4, u_6) and the updated EW-G

4.6.2 Identification of Affected Super Nodes

Since all the k-butterfly equivalent edges in the graph are grouped to form super nodes in *EquiWing-Graph*, the scope of the affected super nodes can be derived using the scope of the affected edges. We now propose the following theorem indicating the affected super nodes in *EquiWing-Graph*.

Theorem 4.3. *Given an inserted edge $e'(u', v')$ and a k-butterfly $\bowtie_{u'v'uv}$ where $k < \overline{\psi'(e')}$, then the following super nodes in EquiWing-Graph may be updated:* $\{\nu | \nu \in \chi, e \in \nu, \psi(e) = k\}$.

Theorem 4.3 can be easily proved using the scope of affected edges for the insertion. Since, only the *k-butterfly connected* edges are affected due to the insertion of e', which is the same as the edges in a super nodes. Hence, when a new edge e' is inserted to G, all the edges with a potential wing number increase, except e', are contained in the affected super nodes of *EquiWing-Graph*. Therefore, we avoid re-examining the original graph G to find the affected edges and hence save significant computational cost.

Example 4.7. *We insert a new edge $e'(u_6, v_4)$ in G as presented in Fig. 4.7(a), and $\overline{\psi'(u_6, v_4)} = 4$. We examine the butterflies $\bowtie_{u_6v_4u_4v_5}$, $\bowtie_{u_6v_4u_3v_5}$, and $\bowtie_{u_6v_4u_4v_6}$ as it includes e'. Since, $\psi(u_6, v_6) = 4 \not< 4$, it is not updated, and so as the edges k-butterfly connected with (u_6, v_6). However, the other edges $(u_3, v_4), (u_3, v_5)$, $(u_4, v_5), (u_6, v_5), (u_4, v_4)$, and (u_4, v_6), have the $\psi(e) < 4$. We recognize that*

ν_3, ν_4, and ν_5 (Fig. 4.3) include these edges, so they are considered to be affected. As a result, all the other edges within these super nodes including the inserted edge e', will be re-examined as their wing number may increase, based on Theorem 4.3.

Likewise, we propose the theorem for the deletion scenario indicating the affected super nodes of *EquiWing-Graph*:

Theorem 4.4. *Given an edge $e'(u', v')$ deleted from G, and $e' \in \nu$ where $\nu \in \chi$. Then the following super nodes in EquiWing-Graph may be updated: $\{\nu\} \cup \{\mu | \psi(\mu) < \psi(e'), \mu \in \chi, (\nu, \mu) \in \Upsilon\}$.*

The proof of Theorem 4.4 can be similarly accomplished using the scope of affected edges in deletion. Note, all the edges within the super node containing the edge e', would be affected and also ν will be the only affected super node with $k = \psi(e')$. Therefore, in Theorem 4.4, we update the super nodes with wing number $k < \psi(e')$ and explicitly add ν to the list of affected super nodes. According to Theorem 4.4, when a new edge e' is deleted from G, all the edges with potential wing number decrease are contained in the affected super nodes of *EquiWing-Graph*. Therefore, we save significant computational cost of exploring the whole graph by restricting the affected edges to a small region.

4.6.3 Dynamic Maintenance Algorithm

EquiWing-Graph update. After an edge insertion, let us denote the set of affected edges in G as E' and the set of affected super nodes in the *EquiWing-Graph* index as χ'. We can easily identify E' and χ' according the theoretical findings proposed in Sections 4.6.1 and 4.6.2. Using E' and χ', the index can be updated by Algorithm 6 as follows.

Algorithm 6: Dynamic maintenance of EW-G

Input : The affected edge set E', the affected super node set χ', and $EW\text{-}G$

Output: The updated EquiWing-Graph ($EW\text{-}G'$)

1 **Function** `Update(`χ'`,`E'`,`$EW\text{-}G$`)`
2 $EW\text{-}G' \longleftarrow EW\text{-}G$
3 **foreach** $\nu \in \chi'$ **do**
4 $EW\text{-}G' \longleftarrow EW\text{-}G' - \{\nu\}$
5 **foreach** $e(u, v) \in E'$ **do**
6 $U' \longleftarrow U' \cup u$, $V' \longleftarrow V' \cup v$
7 $E' \longleftarrow Update_K(G'(U', V'))$; `/* Update wing number of the affected edges E'. */`
8 $\delta_{EW\text{-}G} \longleftarrow Index\ Construction(G'(U', V'))$; `/* Call Algorithm 4. */`
9 $EW\text{-}G' \longleftarrow EW\text{-}G' \cup \delta_{EW\text{-}G}$
10 **foreach** $(\nu, \mu) \in \Upsilon_{EW\text{-}G'}$ **do**
11 **if** $\psi(\nu) = \psi(\mu)$ **then**
`/* merge the two super nodes` ν `and` μ `*/`
12 **return** $EW\text{-}G'(\chi, \Upsilon)$;

Super nodes update. We build a new *EquiWing-Graph* index $\delta_{EW\text{-}G}$ only for the E' induced subgraph of G using Algorithm 4, lines 7 to 8 in Algorithm 6. A super node in $\delta_{EW\text{-}G}$ can be merged into an existing unaffected super node in the index, which is done by lines 10 to 12.

Super edge update. We identify a subgraph $G' \subseteq G$, which is the subgraph induced by vertices contained in E'. The purpose for identifying G' is to preserve the butterfly connectivity between the affected edges and the unaffected edges, which is used to update the super edges in *EquiWing-Graph*. Thanks to G', we can safely remove χ' from the index (lines 3 to 4) since after the super node update, the newly computed super nodes in $\delta_{EW\text{-}G}$ can be linked to the index easily.

Complexity. The time complexity for Algorithm 6 is divided into three parts as follows. (i) It requires $\mathcal{O}(|\mathcal{X}'| + |E'|)$ to remove the affected super nodes from the current *EquiWing-Graph* index and finding affected subgraphs (lines 2-6).

(ii) $\mathcal{O}(d'^2_{max}|E'|)$, where d'_{max} is the maximum degree in affected subgraphs, is required to update the wing number of affected edges and compute the corresponding super nodes and edges ($\delta_{EW\text{-}G}$) in lines 7-8. (iii) $\mathcal{O}(|\Upsilon_{EW\text{-}G'}|)$, where $\Upsilon_{EW\text{-}G'}$ is the set of newly added super edges, is required to add $\delta_{EW\text{-}G}$ to the previous unaffected index $EW\text{-}G'$ and merge the super nodes which are k-butterfly connected (lines 9-11). Since part (ii) has the dominating time complexity, the time complexity of Algorithm 6 is $\mathcal{O}(d'^2_{max}|E'|)$. The space complexity of Algorithm 6 is $\mathcal{O}(|E|)$, as the index used in the algorithm contains all the edges.

Example 4.8. *For an inserted edge $e' = (u_6, v_4)$ in Fig. 4.7(a), the affected super nodes are ν_3, ν_4 and ν_5. Algorithm 6 computes $\delta_{EW\text{-}G}$ for the affected edges in the affected super nodes, resulting into two new super nodes ν_2 and ν_3. $\delta_{EW\text{-}G}$ is then appended to the existing index with unaffected super nodes ν_1 and ν_4. Fig. 4.7(b) displays the updated EquiWing-Graph.*

The dynamic update for the *EquiWing-Tree* is similar to that for *EquiWing-Graph*.

4.7 Experiments

In this section, we perform the experimental analysis for our proposed techniques on real-world datasets.

Experiment setup. All the experiments were conducted on Eclipse IDE, deployed on the platform 64x Intel(R)Core(TM) $i7$-$1065G7$ with CPU frequency 1.50GHz and 16 GB RAM, running Windows 10 Home operating system. All the algorithms were implemented in Java.

Algorithms. We have implemented the following three algorithms for our experiments:

Table 4.1: Dataset characteristics

Datasets	$\|E\|$	$\|V\|$	$\|U\|$	k_{max}	Graph Type
Marvel	96K	19K	6K	1761	Comic-Character
Producer	207K	138K	49K	219	Work-Producer
Location	226K	172K	53K	852	Entity-Place
Record_label	233K	168K	18K	497	Artist-Record
YouTube	293K	94K	30K	1316	User-Group
Stackoverflow	1.3M	545K	97K	1118	User-Post

- **BaseLine**: it is the implementation for k-wing search with wing number.
- **EquiWing-Graph** (EW-G): it exploits the index *EquiWing-Graph* to perform the subgraph search using Algorithm 5.
- **EquiWing-Tree** (EW-T): this approach utilizes the index *EquiWing-Tree* and performs the search.

Since the bitruss decomposition algorithm used for all the three algorithms is [16], the running time of all the algorithms is independent of the bitruss decomposition algorithm and it is replaceable if in future a better option is available. For our experiments, we consider that a query vertex could be in either U or V for all the algorithms.

Datasets. 6 real-world datasets, obtained from the KONECT repository [215], are used to evaluate our algorithms. Table 4.1 represents the characteristics of the datasets, including the number of edges, the numbers of vertices for U and V, and the maximum wing number (k_{max}) for the datasets.

4.7.1 Index Construction

We perform the experiments to evaluate the efficiency of the two indexing schemes: *EquiWing-Graph* (EW-G) and *EquiWing-Tree* (EW-T) in terms of index size, number of super nodes in the indices and index construction time in Table 4.2.

We observe in Table 4.2 that EW-T is more compact as compared to EW-G, i.e., there is a large number of super nodes in EW-G that can be combined,

Table 4.2: Index characteristics

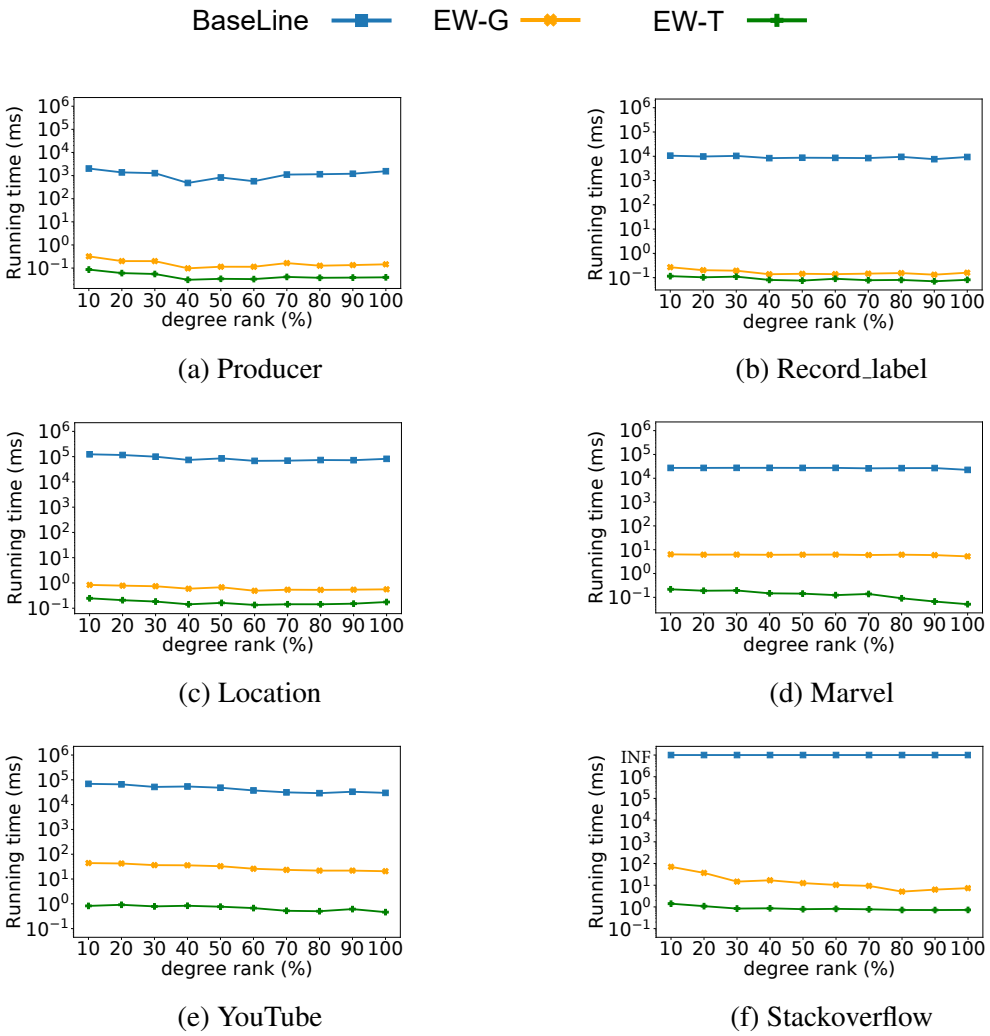
Graph	Graph Size (MB)	# Super Nodes		Index Space (MB)		Compre -ssion Ratio	Construction Time (sec.)	
		EW-G	EW-T	EW-G	EW-T		EW-G	EW-T
Marvel	1.681	11301	689	5.733	2.784	16.402	594.709	596.504
Producer	1.09	5323	2986	1.31	1.003	1.783	30.822	31.754
Location	2.7	5349	1213	2.962	2.004	4.41	1322.202	1322.713
Record_label	1.28	1965	392	1.496	0.931	5.013	590.53	590.597
YouTube	3.589	36795	811	24.977	3.317	45.37	3280.546	3299.05
Stackoverflow	10.892	93275	790	57.51	8.85	118.07	24286.213	24361.543

which reduces the number of the super nodes contained in EW-T to a great extent. Further, we use the *Compression Ratio* (C_R) (the ratio of the number of super nodes in EW-G and the number of compressed super nodes in EW-T) to measure the compactness provided by EW-T. The value of C_R increases as the size and k_{max} of the input bipartite graph increases and C_R is up to 118.07 for *Stackoverflow*. The size of EW-T is also smaller as compared to EW-G as shown in Table 4.2 up to 45 and 16 times for *YouTube* and $Marvel$ respectively. Later, this compactness of EW-T leads to the efficient query processing.

4.7.2 Query Processing

After the indices are constructed we can now utilize them to perform a k-wing search for a query vertex q. The experiment settings for evaluating query processing efficiency are inspired by [45], [61]. We consider two experimental settings below.

In the first set of experiments, to evaluate the diverse variety of the query processing time, vertices with different degrees across the datasets are selected randomly as query vertices. For each dataset, we sort all the vertices $v \in U \cup V$ in non-increasing order of their degrees and segregate them equally into ten buckets. The ordering is such that the first bucket contains the top 10% high-degree vertices and the last one contains the last 10% low-degree vertices in G. Once the ordering is obtained, we now perform a random selection of 100 query vertices from each bucket and the average query processing time for each bucket



(a) Producer (b) Record_label

(c) Location (d) Marvel

(e) YouTube (f) Stackoverflow

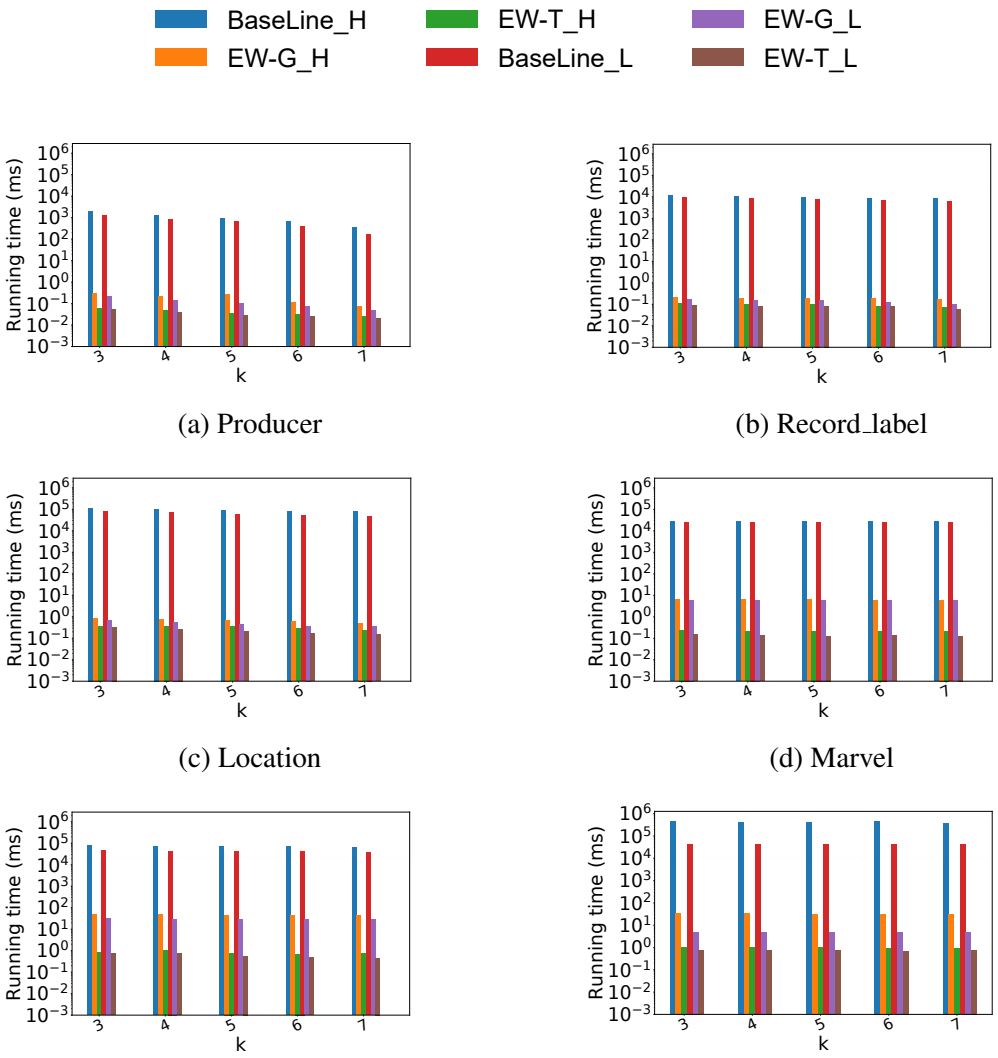
Figure 4.8: Varying degree percentile

is reported in Fig. 4.8. The corresponding k values for each dataset are k = 4 for $Producer$ and $Marvel$, k = 5 for $Record\_label$ and $Location$, k = 10 for $YouTube$ and k = 13 for $Stackoverflow$.

We report the following experimental observations in different real-world datasets. (1) *BaseLine* approach is the least efficient algorithm. It is very much slower than the index-based approaches EW-G and EW-T. In $Stackoverflow$, the queries take approximately 20 minutes for *BaseLine*, whereas the index-based approaches require less than a second. Hence, it is omitted in Fig. 4.8(f) for the $Stackoverflow$ dataset. This vast difference is due to the BFS explo-

ration of the bipartite graph and the costly butterfly-connectivity evaluations in the *BaseLine* approach. Therefore, processing a personalized k-wing query without an index becomes impractical in the real-world datasets. (2) In all the datasets, the k-wing search time drops firmly as the queries drawn from high-degree percentile buckets to low-degree buckets. (3) For the datasets with comparatively smaller value of k_{max} and C_R, i.e., $Producer$ and $Record\_label$, the maximum difference in the performances is observable but not much significant, and hence the running time of the two index-based approaches can be considered comparable on these datasets. However, as the value of either k_{max} or C_R of an input dataset increases, EW-G becomes much slower, EW-T is at least 10 times faster than EW-G across all the query vertices.

For the second set of experiments, we analyze the running time for the k-wing search across different real-world datasets by varying the parameter k. For each value of k in the experiments, we form two query sets: the first set contains 100 high-degree vertices (selected at random from the first 30% of the sorted vertices); the second set contains 100 low-degree vertices (selected at random from the remaining 70% vertices). We evaluate all the algorithms for each set of queries denoted with the suffix $\_H$ (High) and $\_L$ (Low). Henceforth, we have EW-G_H and EW-G_L for ***EquiWing-Graph*** index, EW-T_H and EW-T_L for ***EquiWing-Tree*** index, and ***BaseLine_H*** and ***BaseLine_L*** for ***BaseLine*** algorithm, for the corresponding query sets. The average running time for each method is reported in Fig. 4.9 while varying the parameter k. We observe that for most of the datasets, EW-T is the most efficient k-wing search method, which is at least one order of magnitude faster than EW-G for the high-degree queries, for maximum values of k. The better performance of EW-T can be explained using the reduced number of super nodes in comparison to EW-G, which expedites the query processing. The performance gap increases signif-



(a) Producer

(b) Record_label

(c) Location

(d) Marvel

(e) YouTube

(f) Stackoverflow

Figure 4.9: Varying k

icantly with the large value of k_{max} or C_R for the datasets, such as $Marvel$, $YouTube$ and $Stackoverflow$. These experimental evaluations yet again provide the superiority of *EquiWing-Tree* for k-wing search. Moreover, we can easily infer that an approach without an index is infeasible and impractical in large datasets. For $Stackoverflow$, the running time of *BaseLine* is at least 10^4 times slower than that of the algorithms using the indices.

Table 4.3: Dynamic maintenance (sec.) of EW-G and EW-T

Graph	Inserting an edge				Deleting an edge	
	With upper bound		No upper bound			
	EW-G	EW-T	EW-G	EW-T	EW-G	EW-T
Marvel	0.029	11.976	0.276	13.183	9.284	117.002
Producer	0.041	0.343	0.048	0.37	0.145	0.522
Location	0.275	0.731	0.302	0.754	5.769	6.299
Record_label	0.171	0.262	0.195	0.289	17.923	18.826
YouTube	0.087	25.753	0.694	26.360	59.327	452.502
Stackoverflow	368.083	639.862	723.043	991.458	1757.116	2949.929

4.7.3 Dynamic maintenance of EW-G and EW-T

In this set of experiments, we compare the performances of incremental update of the EW-G and EW-T when G is updated with new edge insertion and existing edge deletion. We randomly select 100 edges for insertion/deletion, and update the wing number for the edges in both the indices after each insertion/deletion. The average update time which includes the edge wing number update and the indices update time, is reported in Table 4.3. EW-G is updated using Algorithm 6, further we transform EW-G to produce EW-T. We can observe that the incremental update approaches (specifically for insertion) are several orders faster than constructing the EW-G and EW-T from scratch (Table 4.2) when G is updated. The update cost for EW-T is almost trivial for sparse and small datasets, e.g., $Producer$, as compared to dense or large datasets, e.g., $Marvel$ and $Stackoverflow$.

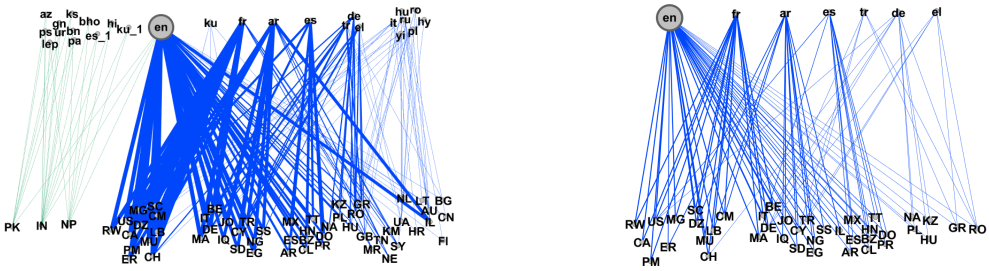
The results in Table 4.3 shows that the update time per edge insertion using bound ranges from 0.04% ($Record\_label$) to 2.68% ($Marvel$) of the index construction from scratch for EW-T. Thus, handling edge insertion becomes efficient. For the deletion case, the update time per edge deletion is more in comparison to insertion and ranges from 0.4% ($Location$) to 19.6% ($Marvel$) of the index construction from scratch for EW-T. We also exhibit the effectiveness of the upper bound for efficiently updating wing numbers of edges caused

by insertions. Since the upper bound effectively estimates the wing number of an inserted edge, the expensive cost for computing the precise wing number of the inserted edge can be avoided. From Table 4.3, we observe that the incremental insertion using the upper bound is always faster than that without using the upper bound in all the cases. In the case of small and sparse datasets, the effectiveness of the upper bound is moderate. However, in cases of dense or large datasets, the effectiveness of the upper bound becomes more significant. This demonstrates the superiority of our proposed upper bound.

4.7.4 Case Study on Unicode

We conduct a case study on the *Unicode* dataset [215] as it is small which makes visualization of *personalized k-wing* easier. It is composed of two sets of vertices, i.e., languages and countries. An edge between two vertices depicts the language spoken in the country. We perform the k-wing search for the English language ('en') as it is the most common language in the *Unicode* dataset, with k=6 and k=7. The results are shown in Fig. 4.10(a) and 4.10(b), respectively. We recognize that 'en' participates with two *6-wings* represented with green and blue subgraphs in Fig 4.10(a).

The green 6-*wing* contains languages like 'hi' (Hindi), 'pa' (Punjabi), 'ur' (Urdu), etc. and the countries India ('IN'), Pakistan ('PK') and Nepal ('NP'). This implies that although people in 'IN', 'PK' and 'NP' speak multiple languages including 'en', yet these languages are demographically restricted to the South-Asian region only. Therefore, using 'hi', 'pa', 'ur', etc. in multilingual packaging apart from the South-Asian region such as European, African or Central-Asian region would be less effective. The vice versa is also true for the languages 'fr' (French), 'es' (Spanish), 'de' (German), etc. from the blue k-*wing*. In contrast, the k-*bitruss* model groups the languages 'hi' and 'fr' to-



(a) Two 6-wings for q = en.

(b) One 7-wing for q = en.

Figure 4.10: k-wings in Unicode Language datasets

gether in a single subgraph, i.e., 6-*bitruss*. Such a bipartite cohesive subgraph connected by weak ties is unable to find the distinct suitable set of languages for the multilingual packaging for a region. Therefore we observe the superiority of k-*wing* using the butterfly-connectivity to segregate the distinct strongly connected subgraphs (green 6-*wing* including 'hi' and blue 6-*wing* including 'fr'). Now, to determine the languages which are more frequently used with 'en', we increase the value of k to 7 to obtain a denser subgraph. The resulting 7-*wing* subgraph is formed with the vertices (languages) 'fr', 'ar' (Arabic), 'es', 'tr' (Turkish), 'de' and 'el' (Greek) on one side and includes 40 countries on the other side in Fig. 4.10(b). The green subgraph in Fig. 4.10(a) is dissolved indicating that 'en' is more frequently spoken in more countries along with the other languages of the blue subgraph. We then present three distinct observations from Fig. 4.10(b) : (i) 'en' shows more affinity towards the European languages as it forms a denser k-*wing* with the 'fr', 'ar', 'es', 'tr', 'de' and 'el'. The two languages 'ar' and 'tr' are from Western Asia and others are from Europe. In real-world scenario, we notice that Nivea, a personal care brand, applies multilingual packaging in Germany using 'en', 'fr', 'es', 'de', 'el' and 'nl' (Dutch) languages [39]. Therefore, we could say that findings from our k-wing model are coherent with real-world applications. However, missing 'nl' in our results is due to the localization of the 'nl' in few countries (the Nether-

lands and Belgium) like Hindi, Urdu, etc. (ii) the languages 'es' and 'fr' show a similar level of cohesiveness for 'en' implying that they both should be used together with 'en'. An interesting finding reported in [218], where using 'fr' reduces the negative effect of 'es' caused in bilingual packaging with 'es' with 'en'. (iii) although french is used frequently preferred with *English* in many countries, surprisingly it is not a preferred combination in France itself (France is not included in Fig. 4.10(b)). This is also true as almost 87.2% speak French as their native language and the government of France excessively promotes its native language in commercial and workplace communications.

4.8 Summary

In this chapter, we study the problem of personalized k-wing search for large and dynamic bipartite graphs. We adapt the index *EquiWing-Graph* and propose *EquiWing-Tree* to tackle the problem efficiently, which leads to linear time search algorithms. We construct *EquiWing-Tree* by proposing the *k-butterfly loose connectivity* and exploiting the hierarchical property of k-wing, which further speeds up the query processing. We also discuss the efficient maintenance of the indices in dynamic bipartite graphs. The upper bound for the wing number during an edge insertion is also proposed. We conduct extensive experiments across diverse real-world datasets. From these experiments, we observe the superiority of our index based approaches *EquiWing-Graph* and *EquiWing-Tree* over the baseline approach. The experiments also display the better compression and performance of *EquiWing-Tree* over *EquiWing-Graph*. Moreover, a case study is presented to display the effectiveness of our personalized k-wing model.

Chapter 5

On Maximizing the Vertex Coverage for Top-k t-Bicliques

5.1 Introduction

Enumeration of all maximal bicliques in bipartite graphs is a well-studied fundamental problem. However, a wide range of applications need less overlapping bicliques with specific size constraints instead of all the maximal bicliques. In this chapter, we study a new biclique problem, called the top-k t-biclique coverage problem. A t-biclique is a biclique with a size constraint t for one vertex set and the problem aims to find k t-bicliques maximising the coverage on the other vertex set. The top-k t-biclique coverage problem has novel applications such as finding top-k courses while maximising student engagement. We prove that this problem is NP-hard. A straightforward way to address the problem first needs to enumerate and store all t-bicliques and then greedily select k promising t-bicliques, leading to an approximate guarantee on the coverage. However, it takes exponential space, which is impractical. We then apply a fast approximation scheme to solve this problem, which shaves the exponential space con-

sumption by progressively updating top-k results during the t-biclique enumeration. Observing that the fast approximation algorithm takes too much time on updating the results due to the coverage being computed from scratch for each update, an online index is devised to address the drawback. Due the hardness of the problem, even the fast approximation algorithm cannot scale to large dataset. To devise a scalable solution, we then propose a heuristic algorithm running in polynomial time. Thanks for four carefully designed heuristic rules, the heuristic algorithm can find large coverage top-k t-bicliques extremely fast for large datasets. Apart from that, the heuristic result with large coverage can effectively prune unpromising enumerations in the fast greedy algorithm, which improves the efficiency of the fast approximation algorithm without compromising the approximation ratio. Extensive experiments are conducted on real datasets to justify the effectiveness and efficiency of the proposed algorithms.

Chapter outline. We organize the remaining chapter as follows. Section 5.2 discusses the preliminaries, problem definition and hardness of the problem. Section 5.3 describes the baseline greedy approach. An index based fast greedy approach is introduced in Section 5.4. Section 5.5 presents the novel heuristic approach. Experimental evaluations of all the algorithms are discussed in Section 5.6.

5.2 Problem Formulation

In this section, we consider an undirected bipartite graph $G = (U \cup V, E)$. U and V are two disjoint sets of vertices in G. E is the edge set in G and $E \subseteq U \times V$. For each vertex $u \in U$, $n(u)$ denotes all the neighbours of u in G, i.e., $n(u) = \{v|(u, v) \in E\}$. For a vertex set $X \subseteq U$, $N(X)$ denotes all the common

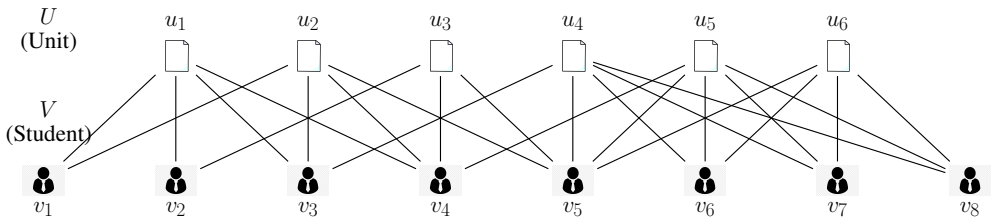


Figure 5.1: Bipartite graph representing student-like-unit network.

neighbors for the vertices in X, i.e., $N(X) = \bigcap_{u \in X} n(u)$. A *complete bipartite graph* or *biclique* B is then defined as a set pair $(X, N(X))$. In the rest of this chapter, we simplify the notation for the biclique $B = (X, N(X))$ to X when the context is obvious. Note, the role of U and V is interchangeable as per the requirement of an application.

t-biclique. For a user given threshold t for one of the vertex set (say X) in G, we now define a t-biclique below.

Definition 5.1. *t-biclique: A t-biclique $\hat{X}$ is defined as a set pair $(X, N(X))$ such that $|X| = t$.*

Example 5.1. *In Figure 5.1, there are a total of* 14 2*-bicliques such as* $\hat{X}_1$={$(u_5, u_6), (v_5, v_6, v_7, v_8)$}*,* 14 3*-bicliques such as* $\hat{X}_2$={$(u_2, u_3, u_5), (v_4, v_5)$}*,* 5 4*-bicliques such as* $\hat{X}_3$={$(u_1, u_2, u_3, u_5), (v_4)$} *and* 1 5*-biclique i.e.,* $\hat{X}_4$={$(u_2, u_3, u_4, u_5, u_6)$, (v_5)}.

t-biclique set coverage. For a given set of t-bicliques $\hat{\mathfrak{X}}$={$\hat{X}_1, \hat{X}_2, \ldots, \hat{X}_k$} in G, we denote the *coverage* of $\hat{\mathfrak{X}}$ on the vertex set V as $COV(\hat{\mathfrak{X}})$ and $COV(\hat{\mathfrak{X}})$= $\bigcup_{i=1}^{k} N(\hat{X}_i)$. The size of the t-biclique set coverage is denoted as $|COV(\hat{\mathfrak{X}})|$.

Example 5.2. *The corresponding* 2*-biclique set coverage in Figure 5.1 for the set* $\hat{\mathfrak{X}} = \{\hat{X}_5, \hat{X}_6, \hat{X}_7\}$*, where* $\hat{X}_5$={$(u_1, u_2), (v_1, v_3, v_4)$}*,* $\hat{X}_6$={$(u_1, u_3), (v_2, v_4)$}*, and* $\hat{X}_7$={$(u_4, u_5), (v_5, v_6, v_7, v_8)$}*, is* $COV(\hat{\mathfrak{X}}) = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$.

Problem description. In this chapter, we aim to find k t-bicliques $\hat{\mathfrak{X}}$ such that the corresponding coverage $|COV(\hat{\mathfrak{X}})|$ is maximized. Formally, the problem

studied is defined below.

Top-k t-biclique coverage problem. Given a bipartite graph G, a threshold t and an integer k, find a set of t-bicliques $\hat{\mathfrak{X}}$, such that $|\hat{\mathfrak{X}}| = k$ and $|COV(\hat{\mathfrak{X}})|$ is maximized.

Hardness of the problem. Finding top-k t-bicliques with maximized $|COV(\hat{\mathfrak{X}})|$ is NP-hard. We show when $k = 1$, this problem is equivalent to the maximum one-sided edge cardinality problem (MOFCP). MOFCP is proved to be NP-complete in [219]. The definition of MOFCP and reduction are shown as follows. Given a bipartite graph G and a positive integer z, the problem of MOFCP is to find a biclique $(X, N(X))$ such that $|X| = z$ and $|X \times N(X)|$ is maximized. For any given instance of MOFCP with z and G, it is trivial to observe, every solution for MOFCP is a solution for this problem with $k = 1$ and $t = z$ in G. On the other hand, for every solution $(X, N(X))$ of this problem with $k = 1$ and $t = z$ in G, after removing arbitrary $|X| - z$ vertices from X, leading X to X', $(X', N(X'))$ is a solution for MOFCP.

5.3 The Baseline Approach

In this section we provide a straightforward approach to address the problem of top-k t-biclique coverage. Given a bipartite graph $G = (U \cup V, E)$, a threshold t and an integer k, the approach includes finding all t-bicliques and selecting top-k among them to maximize the coverage.

t-biclique enumeration. It is easy to see that by enumerating every vertex set $X \in 2^U$ such that $|X| = t$ and $N(X) \neq \emptyset$, all the t-bicliques can be enumerated systematically. However, not all those t-bicliques can contribute to the top-k t-bicliques maximising the coverage. To make the baseline smarter, we propose a technique below to avoid enumerating non-promising t-bicliques and thereby

Algorithm 7: Baseline Algorithm

```
1  X̂ ← ∅; Ŝ ← ∅; GreedyCoverB (G, k, t);
2  Function GreedyCoverB (G(U, V), k, t)
3      G ← core(1, t);
4      t-BICLIQUE(U, ∅, t, ∅);
5      GreedyCoverBUtil(G, k);
6      return X̂;
7  Function t-BICLIQUE (U, X, t, N(X))
8      if |X| = t then
9          X̂ ← (X, N(X)); Ŝ ← Ŝ ∪ {X̂}; return ;
10     foreach u ∈ U do
11         U' ← ∅;
12         foreach u' ∈ U \ {u} do
13             if N(X ∪ {u}) ∩ n(u') ≠ ∅ then
14                 U' ← U' ∪ {u'};
15         if |U'| ≥ t − |X ∪ {u}| then
16             t-BICLIQUE(U', X ∪ {u}, t, N(X ∪ {u}) );
17         U ← U \ {u};
18 Function GreedyCoverBUtil (G, k)
19     while |X̂| < k OR Ŝ ≠ ∅ do
20         X̂_i ← argmax_{X̂_i ∈ Ŝ}{|V ∩ N(X_i)|};
21         V ← V \ N(X)_i; X̂ ← X̂ ∪ X̂_i; Ŝ ← {Ŝ} \ {X̂_i};
```

speeding up the enumeration process.

Applied pruning. We exploit the concept of (α, β)-*core subgraph* recently introduced in [15]. Using $(1, t)$-*core* provides an efficient (polynomial time) pruning of vertices based on their degree in the respective vertex set without losing any t-bicliques in the final result.

The enumeration algorithm. In Algorithm 7, t-BICLIQUE enumerates all t-bicliques and stores them in $\hat{\mathcal{S}}$. The function follows the idea of iterating for each vertex in the candidate set U and recursively adding the vertex to X forming a t-biclique (lines 10 to 19). Meanwhile, for each recursion, the candidate set U is pruned by removing vertices not in sharing neighbors with $X \cup \{u\}$ (lines 12 to 15). The condition in line 16 ensures that there are sufficient vertices to select from U, in order to accomplish a total of t vertices in X.

Maximising the coverage. Since the problem of top-k t-biclique coverage is proven to be NP-hard and the t-biclique enumeration is time consuming, we propose a straightforward solution by adapting the greedy algorithm in [60] for selecting t-bicliques maximising the coverage. The function GREEDYCOVERBUTIL iterates over the set of t-bicliques $\hat{\mathbb{S}}$ obtained from the function t-BICLIQUE (lines 21 to 24). In each iteration, a t-biclique $\hat{X}_i$ with the maximum coverage over the remaining vertex set V is selected in line 22, where the remaining vertex set V contains vertices of V which are not yet covered by the t-bicliques in $\hat{\mathbb{X}}$. Once $\hat{X}_i$ is selected, the uncovered vertices, $\hat{\mathbb{X}}$ and $\hat{\mathbb{S}}$ are updated in line 23.

Complexity and approximation analyses. Since the enumerated set of t-bicliques is the subset of 2^U and each recursive subproblem induced by t-BICLIQUE runs in $\mathcal{O}(|U|d_{max})$, where d_{max} is the maximum degree in U. Therefore the time complexity of Algorithm 7 is $\mathcal{O}(2^{|U|}|U|d_{max})$. Note that the time complexity for greedy selection (GREEDYCOVERBUTIL) of k t-bicliques is negligible in comparison to that of GREEDYCOVERB. The space required by Algorithm 7 is the same as the time complexity and is exponential as all the enumerated t-bicliques need to be stored in $\hat{\mathbb{S}}$. As shown in [60], a greedy algorithm that iteratively selects a set which covers the largest number of yet uncovered vertices constructively approximates max k-cover within a ratio of at least $(1 - \frac{1}{e}) \simeq 0.632$.

Example 5.3. *We show how Algorithm 7 find top-3 2-bicliques in Figure 5.2. it firstly enumerates and stores all 2-bicliques in $\hat{\mathbb{S}}$ by traversing the vertices using the vertex id order and then starts to select 3 2-bicliques as below. Initially all the vertices in V are uncovered and thus it includes the first biclique with the maximum coverage to $\hat{\mathbb{X}}$, i.e. $\hat{X}_1$=$\{(u_6, u_7), (v_7, v_8, v_9)\}$. The uncovered vertices are then reduced to $\{v_1, \ldots, v_6, v_{10}\}$. For the next 2-biclique covering the maximum remaining uncovered vertices, it has two choices, $\hat{X}_2$=$\{(u_2, u_5), (v_5, v_6)\}$ and $\hat{X}_3$=$\{(u_2, u_4), (v_2, v_6)\}$. It selects either of them (say $\hat{X}_2$). The uncov-*

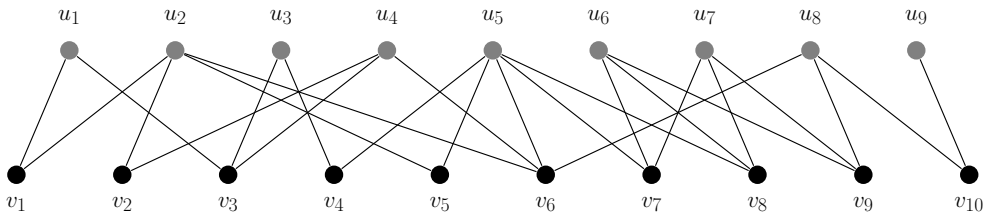


Figure 5.2: Bipartite graph G.

ered vertices are reduced to $\{v_1, \ldots, v_4, v_{10}\}$. *After that, every remaining* 2*-bicliques in* $\hat{\mathbb{S}}$ *covers* 1 *uncovered vertex. Therefore, it selects any of them (say* $\hat{X}_3$*). Hence, the resulting top-*3 2*-bicliques with maximum coverage are* $\hat{\mathfrak{X}}$=$\{\hat{X}_1, \hat{X}_2, \hat{X}_3\}$ *with* $COV(\hat{\mathfrak{X}})$=$\{v_2, v_5, v_6, v_7, v_8, v_9\}$ *and* $|COV(\hat{\mathfrak{X}})|$=6.

Limitations of baseline algorithm. The baseline algorithm independently enumerates all the t-bicliques and then maximizes the coverage with k t-bicliques. However, this leads to the following limitations: (i) Enumerating unpromising t-bicliques - having separate enumeration and coverage components makes Algorithm 7 incapable of taking advantage of the global coverage discovered before enumerating a t-biclique. We later show that combining the enumeration and coverage components prunes away a significant number of unpromising t-bicliques and also provides a guarantee for the corresponding result. (ii) Exponential space complexity - although the baseline algorithm provides a guarantee for the result, enumerating and storing all the t-bicliques which is exponential in number makes it inapplicable for larger datasets. Later in the experiments, we show that even for the smaller dense datasets, the baseline algorithm is inapplicable due to memory exhaustion. (iii) Unpromising vertices - the baseline algorithm uses $(1, t)$-core to prune out many vertices from U, however, there is still a significant number of vertices that result in unpromising t-bicliques. Since the baseline algorithm enumerates $2^{|U|}$ t-bicliques, reducing the vertices in U would significantly improve the performance.

5.4 Fast Greedy Approach with Index

In this section, we address the limitations of the baseline by presenting a fast greedy algorithm. The fast greedy algorithm avoids consuming exponential space for storing all t-bicliques while still providing an approximation guarantee. However, the fast greedy algorithm takes non-omissible extra computational cost for maximising the coverage apart from the cost of t-biclique enumeration. We then shave such extra cost by using an online index without compromising the provided approximation guarantee.

We first introduce the fast greedy algorithm and then present the online index.

5.4.1 Fast Greedy Algorithm

The intuition behind the fast greedy approach is to maintain at most k t-bicliques in the resulting set $\hat{\mathfrak{X}}$ throughout the entire enumeration. For each newly enumerated t-biclique $\hat{X}$, we consider $\hat{X}$ as a part of top-k result only if including $\hat{X}$

Algorithm 8: Fast Greedy Algorithm

1 **Function** GREEDYCOVERF (G, st, end, ind, t, k)
2 $G \leftarrow core(1, t)$;
3 t-BICLIQUE($U, \varnothing, t, \varnothing$);
/* Instead of storing $\hat{X}$, replace line 9 in Algorithm 7 with GREEDYCOVERFUTIL. */
4 **return** $\hat{\mathfrak{X}}$;
5 **Function** GREEDYCOVERFUTIL ($k, \hat{X}$)
6 **if** $|\hat{\mathfrak{X}}| < k$ **then**
7 $\hat{\mathfrak{X}} \leftarrow \hat{\mathfrak{X}} \cup \{\hat{X}\}$;
8 return;
9 $T \leftarrow |COV(\hat{\mathfrak{X}})| - |COV'(\hat{X}_{min}, \hat{\mathfrak{X}})| + |COV'(\hat{X}, \hat{\mathfrak{X}})|$;
10 **if** $T > (1 + \frac{1}{k}) \times |COV(\hat{\mathfrak{X}})|$ **then**
11 $\hat{\mathfrak{X}} \leftarrow \hat{\mathfrak{X}} \setminus \{\hat{X}_{min}\} \cup \{\hat{X}\}$;

to $\hat{\mathfrak{X}}$ (possibly removing $\hat{X}'$ from $\hat{\mathfrak{X}}$, where $\hat{X}'$ is a previously selected t-biclique to be replaced by $\hat{X}$) leads to a significant increase of $|COV(\hat{\mathfrak{X}})|$.

We first introduce the following definitions, which help us formally and quantitatively define the significant increase.

Definition 5.2. ***Private coverage contribution***. *Given a set of t-bicliques $\hat{\mathfrak{X}}$, and a t-biclique $\hat{X} \in \hat{\mathfrak{X}}$, the private coverage contribution of $\hat{X}$ w.r.t. $\hat{\mathfrak{X}}$ is defined as $COV'(\hat{X}, \hat{\mathfrak{X}}) = \{COV(\hat{X}) \setminus COV(\hat{\mathfrak{X}} \setminus \{\hat{X}\})\}$.*

Definition 5.3. ***Minimal t-biclique*** *($\hat{X}_{min}$). Given a set of t-bicliques $\hat{\mathfrak{X}}$, a minimal t-biclique ($\hat{X}_{min}$) of $\hat{\mathfrak{X}}$ is the one with the minimum private coverage contribution, i.e., $\hat{X}_{min} = argmin_{\hat{X}_j \in \hat{\mathfrak{X}}}\{|COV'(\hat{X}_j, \hat{\mathfrak{X}})|\}$.*

Significant coverage contribution. Based on the above two definitions, given a set of candidate top-k t-bicliques $\hat{\mathfrak{X}}$, a $\hat{X}_{min}$ in $\hat{\mathfrak{X}}$, and a newly generated t-biclique $\hat{X}$, we define that $\hat{X}$ has a significant coverage contribution, if by replacing $\hat{X}_{min}$ with $\hat{X}$, i.e., $\hat{\mathfrak{X}}' = \hat{\mathfrak{X}} \setminus \{\hat{X}_{min}\} \cup \{\hat{X}\}$, the following condition is satisfied.

$$|COV(\hat{\mathfrak{X}}')| > |COV(\hat{\mathfrak{X}})|(1 + \frac{1}{k}) \tag{5.1}$$

Using the definition of private coverage contribution, a less computational expensive condition can be defined below.

$$\begin{aligned} |COV(\hat{\mathfrak{X}})| - |COV'(\hat{X}_{min}, \hat{\mathfrak{X}})| + |COV'(\hat{X}, \hat{\mathfrak{X}}')| \\ > (1 + \frac{1}{k})COV(\hat{\mathfrak{X}}) \end{aligned} \tag{5.2}$$

We are ready to show the complete fast greedy algorithm.

Algorithm. Algorithm 8 only utilizes the set $\hat{\mathfrak{X}}$ for storing the promising k t-biclique candidates and discards the usage of $\hat{\mathbb{S}}$ as in Algorithm 7. Then the function t-BICLIQUE is called to enumerate all the t-bicliques without storing all enumerated t-bicliques. Instead, for each enumerated t-biclique $\hat{X}$ in Algorithm 7, the procedure GREEDYCOVERFUTIL is called to update $\hat{\mathfrak{X}}$. Finally, $\hat{\mathfrak{X}}$ is returned as the top-k t-bicliques.

The procedure GREEDYCOVERFUTIL is shown in lines 5 to 11 of Algorithm 8. When the number of t-bicliques is less than k, i.e. $|\hat{\mathfrak{X}}| < k$, $\hat{\mathfrak{X}}$ is updated by simply adding $\hat{X}$ (line 7). Otherwise, we check if the resulting coverage size T (line 9) of $\hat{X}$ is eligible to replace $\hat{X}_{min}$ (line 10) based on Equation 5.2. If the condition is satisfied $\hat{\mathfrak{X}}$ is updated to be $\hat{\mathfrak{X}} \setminus \{\hat{X}_{min}\} \cup \{\hat{X}\}$ (line 11).

Approximation analysis. Algorithm 8 follows the standard approximation scheme proposed in [220] used for solving the online maximum ρ-coverage (OMC) problem achieving 0.25-approximation. We shall show that Algorithm 8 has the same setup as the OMC problem and therefore also achieves 0.25-approximation.

The equivalence. Let V of G be the universe of elements in the OMC problem. The collection of subsets in the OMC problem consists of each progressively enumerated t-biclique. Finding top-k subsets without knowing the entire collection but covering the universe as much as possible is the same as the setup of Algorithm 8.

By [220], Algorithm 8 achieves 0.25-approximation if line 10 is replaced with Equation 5.1. Since Equation 5.2 has the same effect of Equation 5.1, Algorithm 8 also achieves 0.25-approximation.

Complexity analysis. Algorithm 8 only maintains top-k promising t-bicliques. Thus the space required is $\mathcal{O}(k \cdot |\hat{X}_{max}| + |G| + |t||V|)$, where $|\hat{X}_{max}|$ is the t-biclique with the maximum number of total vertices including both the vertex sets, $|G|$ is the space consumption for storing G, and $|t||V|$ is the maximum space consumption induced by recursion.

The extra dominating cost of Algorithm 8 compared to the cost of the t-biclique enumeration is line 9 for checking the condition in Equation 5.2. The cost for computing $COV(\hat{\mathfrak{X}})$ requires $\mathcal{O}(k \cdot |COV(\hat{X}_{max})|)$, which dominates the costs for computing $COV'(\hat{X}_{min}, \hat{\mathfrak{X}})$ and $COV'(\hat{X}, \hat{\mathfrak{X}})$. The number of t-

bicliques enumerated is of the order $\mathcal{O}(2^{|U|})$. Therefore, the time complexity for Algorithm 8 is $\mathcal{O}\left(2^{|U|}\ k\ |COV(\hat{X}_{max})| + 2^{|U|} d_{max}|U|\right)$.

5.4.2 Online Index For Fast Greedy Algorithm

As shown in the previous section, checking the condition in Equation 5.2 is time consuming, since it is checked from scratch. Therefore we now propose an online index called *Support Maintenance Index* (SM-INDEX) that efficiently (i) preserves the private coverage for each $\hat{X} \in \hat{\mathfrak{X}}$. (ii) computes the minimal t-biclique $\hat{X}_{min}$. (iii) maintains the coverage $COV(\hat{\mathfrak{X}})$ for the top-k t-bicliques. We first define the following terminology.

Definition 5.4. ***t-biclique vertex support*** *(VS): Given a set of t-bicliques $\hat{\mathfrak{X}}$ and $COV(\hat{\mathfrak{X}})$, for each vertex $v \in COV(\hat{\mathfrak{X}})$ the t-biclique vertex support of v is the set of t-bicliques in $\hat{\mathfrak{X}}$ containing v, i.e., $VS_v = \{\hat{X}_i | \hat{X}_i \in \hat{\mathfrak{X}} \wedge v \in COV(\hat{X}_i)\}$.*

SM-INDEX. It is composed of (i) *vertex support* structure and (ii) *private coverage support* structure.

Vertex support structure (VS_v). It is a hash structure. Each $v \in COV(\hat{\mathfrak{X}})$ is a key and the set of t-bicliques in $\hat{\mathfrak{X}}$ containing v is the corresponding value for v.

Private coverage support structure (PS). To efficiently compute $\hat{X}_{min}$ of $\hat{\mathfrak{X}}$, we maintain another hash structure PS. Each $\hat{X} \in \hat{\mathfrak{X}}$ is a key and $COV'(\hat{X}, \hat{\mathfrak{X}})$ is the corresponding value for $\hat{X}$.

Updating SM-INDEX. For a given set of t-bicliques $\hat{\mathfrak{X}}$, SM-INDEX must be updated every time $\hat{\mathfrak{X}}$ is updated with the newly enumerated t-biclique $\hat{X}$. Updating $\hat{\mathfrak{X}}$ includes firstly finding and removing $\hat{X}_{min}$ in $\hat{\mathfrak{X}}$, and then adding $\hat{X}$ to $\hat{\mathfrak{X}}$. The updated $\hat{\mathfrak{X}}$ containing $\hat{X}$ is denoted as $\hat{\mathfrak{X}}'$.

Algorithm 9: Index-based Fast Greedy Algorithm

```
1  Function IndGreedyCoverF(G, st, end, ind, t, k)
2      G ← core(1, t);
3      t-BICLIQUE(U, ∅, t, ∅);
       /* Replace GreedyCoverFUtil in Algorithm 8
          with IndGreedyCoverFUtil.                    */
4      return X̂;
5  Function IndGreedyCoverFUtil(k, X̂)
6      if |X̂| < k then
7          return INSERT(X̂);
8      DELETE(X̂_min); INSERT(X̂);
9      T ← |COV(X̂)| − |COV'(X̂_min, X̂)| + |COV'(X̂, X̂)|;
10     if T > (1 + 1/k) × |COV(X̂)| then
11         X̂ ← X̂ \ {X̂_min} ∪ {X̂};
12         X̂_min ← MIN(X̂);
13     else
14         DELETE(X̂);
15         INSERT(X̂_min);
16 Function Insert(X̂)
17     X̂ ← X̂ ∪ X̂; COV'(X̂, X̂) ← ∅
18     foreach v ∈ COV(X̂) do
19         VS_v ← VS_v ∪ {X̂};
20         if |VS_v| == 1 then
21             COV(X̂) ← COV(X̂) ∪ {v};
22             COV'(X̂, X̂) ← COV'(X̂, X̂) ∪ {v}
23         else if |VS_v| == 2 then
               /* COV'(X̂_j, X̂) ∈ VS_v \ {COV'(X̂_j, X̂)}   */
24             COV'(X̂_j, X̂) ← COV'(X̂_j, X̂) \ {v}
25 Function Delete(X̂)
26     X̂ ← X̂ \ {X̂};
27     foreach v ∈ COV(X̂) do
28         VS_v ← VS_v \ {X̂};
29         if VS_v == ∅ then
30             COV(X̂) ← COV(X̂) \ {v};
31         else if |VS_v| == 1 then
               /* X̂_j ∈ VS_v(X̂)                          */
32             COV'(X̂_j, X̂) ← COV'(X̂_j, X̂) ∪ {v}
```

Deletion of $\hat{X}_{min}$. For each $v \in COV(\hat{X}_{min})$, delete the entries of VS with v as the key. The updated value of VS_v causes two affections on PS.

- Case 1. Remove the entry from PS with key of $\hat{X}_{min}$.
- Case 2. If $|VS_v|$ becomes 1, v is now contained by only one t-biclique denoted as $\hat{X}'$. Then, the value in PS with key of $\hat{X}'$ should include v.

Insertion of $\hat{X}$. For each $v \in COV(\hat{X})$, update VS_v as $VS_v \cup \{\hat{X}\}$. After updating VS_v, there could arise following two cases triggering PS update.

- Case 1. If $|VS_v|$ is 1, i.e., v is exclusively contained only by $\hat{X}$ in $\hat{\mathcal{X}}'$. Then, the value in PS with key of $\hat{X}$ should include v.
- Case 2. if $|VS_v|$ increases from 1 to 2, i.e., v is contained by $\hat{X}$ as well as another t-biclique in $\hat{\mathcal{X}}'$. For each t-bicliques $\hat{X}'$ in VS_v, the value in PS with key of $\hat{X}'$ should exclude v.

Using SM-INDEX. We now discuss how to use the progressively updated index to speed up the computation of line 9 in Algorithm 8. Let VS and PS denote the index before the update while VS' and PS' denote the index after the update. $|COV(\hat{\mathcal{X}})|$ is simply $|VS|$. $\hat{X}_{min}$ and $|COV'(\hat{X}_{min}, \hat{\mathcal{X}})|$ can be derived by traversing PS, where the traversing cost can be absorbed when maintaining PS. For $|COV'(\hat{X}, \hat{\mathcal{X}}')|$, access PS' with key of $\hat{X}$. Based on the above discussion, line 9 runs in $\mathcal{O}(1)$ using the progressively updated index.

For the self-completeness purpose, the pseudo code for the fast greedy algorithm using SM-INDEX is shown in Algorithm 9.

Maintenance cost. The maintenance cost includes deletion and insertion costs for each recursive sub-problem in Algorithm 8. For a deletion, the dominating part is traversing vertices in $COV(\hat{X}_{min})$ to update PS and VS. For an insertion, the dominating part is traversing vertices in $COV(\hat{X})$ to update PS and VS. Since both $|COV(\hat{X}_{min})|$ and $|COV(\hat{X})|$ can be bounded by $|COV(\hat{X}_{max})|$, the maintenance cost for each update is $\mathcal{O}(|COV(\hat{X}_{max})|)$.

Compared to Algorithm 8 with no SM-INDEX, Algorithm 9 runs in the same complexity as t-biclique enumeration. The extra cost induced by considering coverage in each recursive sub-problem becomes non-dominating.

Space cost of the index. The space complexity of SM-INDEX is induced by storing VS and PS. VS is $\mathcal{O}(k|\hat{X}_{max}|)$, where $|\hat{X}_{max}|$ is the t-biclique with the maximum number of total vertices including both the vertex sets X and $N(X)$. Similar space cost could be concluded for PS, and therefore the overall space complexity of SM-INDEX is $\mathcal{O}(k|\hat{X}_{max}|)$.

Approximation ratio. The approximation ratio for Algorithm 9 using SM-INDEX is still $1/4 = 0.25$. This is because the index does not affect the greedy rule used.

Example 5.4. *Algorithm 9 finds top-*3 2*-bicliques with the maximum coverage for Figure 5.2 as follows. The first* 3 2*-bicliques enumerated by Algorithm 9 are* $\hat{X}_1 = \{(u_1, u_2), (v_1)\}$, $\hat{X}_2 = \{(u_1, u_3), (v_3)\}$ *and* $\hat{X}_3 = \{(u_1, u_4), (v_3)\}$*, and are treated as the starting result set* $\hat{\mathfrak{X}}$ *with coverage* $COV(\hat{\mathfrak{X}}) = \{v_1, v_3\}$*. The corresponding minimal* t*-biclique* $\hat{X}_{min}$ *is* $\hat{X}_3$ *with* $COV'(\hat{X}_3, \hat{\mathfrak{X}})=\varnothing$ *(*$\hat{X}_2$ *can also be selected). The next enumerated* 2*-biclique is* $\hat{X}_4=\{(u_2, u_4), (v_2, v_6)\}$*, which is eligible to update* $\hat{\mathfrak{X}}$ *and replace* $\hat{X}_{min}$ *from* $\hat{\mathfrak{X}}$*. Consequently,* $\hat{\mathfrak{X}}=\{\hat{X}_1, \hat{X}_2, \hat{X}_4\}$*,* $COV(\hat{\mathfrak{X}})=\{v_1, v_2, v_3, v_6\}$ *and* $\hat{X}_{min}=\hat{X}_2$ *with* $COV'(\hat{X}_2, \hat{\mathfrak{X}})=\{v_3\}$*. The next eligible* 2*-biclique to update* $\hat{\mathfrak{X}}$ *is* $\hat{X}_5=\{(u_6, u_7), (v_7, v_8, v_9)\}$*. The resulting updated results are* $\hat{\mathfrak{X}}=\{\hat{X}_1, \hat{X}_4, \hat{X}_5\}$ *and* $COV(\hat{\mathfrak{X}})=\{v_1, v_2, v_6, v_7, v_8, v_9\}$*, with* $\hat{X}_{min}=\hat{X}_1$ *whose* $COV'(\hat{X}_1, \hat{\mathfrak{X}})=\{v_1\}$*. To this end, no newly enumerated* 2*-biclique is able to replace* $\hat{X}_{min}$ *and hence the of top-*3 2*-biclique set is* $\hat{\mathfrak{X}}=\{\hat{X}_1, \hat{X}_4, \hat{X}_5\}$ *with* $COV(\hat{\mathfrak{X}})=\{v_1, v_2, v_6, v_7, v_8, v_9\}$ *and* $|COV(\hat{\mathfrak{X}})|=6$*. We observe that maximum coverage is the same as that determined by the baseline approach (Example 5.3) without storing all* 2*-bicliques. Algorithm 9 could produce different* $\hat{\mathfrak{X}}$*, however, the coverage remains* 6*.*

5.5 Heuristic Approach

The top-k t-biclique coverage problem is indeed tough. Even using a state-of-the-art approximation scheme, the adapted fast greedy algorithm still runs in exponential time. As a result, they cannot scale to even medium-sized datasets.

In this section, we propose a carefully designed heuristic algorithm. Instead of considering all the t-bicliques, we only consider promising t-bicliques based on our proposed heuristic rules. To fully reveal the potential of the promising t-bicliques, we group them into multiple sets of top-k results, where our proposed techniques dynamically decide the number of multiple sets according to the data distribution instead of fixing it artificially. Eventually, a top-k t-biclique set with the best coverage among the multiple sets serves as the heuristic result.

Apart from using the heuristic algorithm standalone, we then further use the heuristic result as the initial top-k t-bicliques in Algorithm 9. Since this carefully derived top-k t-bicliques have much better coverage, Algorithm 9 can be further sped up because a higher initial coverage can provide stronger pruning effectiveness. Thanks to this heuristic, it is possible to get approximate results with a guarantee on reasonably large datasets.

5.5.1 Heuristic Framework

In this section, we discuss the major components of the heuristic algorithm in Algorithm 10.

Heuristics for generating large coverage t-bicliques. Algorithm 10 first aims to generate t-bicliques progressively with the largest coverage to the least coverage. In order to do so, vertices in U are sorted according to our proposed r_t-core number in non-increasing order (line 1). Then, for each vertex $u \in U$ in order, a t-biclique containing u, denoted by $\hat{X}_u$, is heuristically generated (line 5).

Algorithm 10: HEURISTIC($G = (U \cup V, E)$,t,k)

```
1  order vertices in U using Heuristic 1 ;
2  X̂ ← {{∅}};
3  foreach u ∈ U do
4      if u satisfies Heuristic 2 then
5          X̂_u ← heuristically generate t-biclique containing u;
6          if X̂_u qualifies for Heuristic 3 then
7              X̂ ←create a new empty set of top-k t-biclique;
8              X̂ ← X̂ ∪ {X̂};
9          UPDATE(X̂_u, X̂); /* Heuristic 4                    */
10     else
11         break;
12 return argmax_{X̂∈X̂} {|COV(X̂)|};
```

Heuristics for generating top-k t-bicliques with large coverage. For generated $\hat{X}_u$, we propose grouping heuristics to decide whether a new top-k t-biclique set should be created to improve the effectiveness (lines 6 to 9). Then $\hat{X}_u$ will be used to update existing top-k t-biclique sets if it can improve the coverage of them (line 10).

Note that Algorithm 10 aims for showing ideas concisely. Details will be introduced in the next two subsections.

5.5.2 Heuristics for generating large coverage t-bicliques

A heuristic for generating t-bicliques progressively from the possible largest to the least individual coverage is as follows. We first order the vertices by their degree in non-descending order and then generate t-bicliques by the ordering. However, a vertex u with a high degree may have 2-hop neighbours all with low degrees, making the coverage of any t-bicliques containing u very small.

To address the above drawback, we propose a novel measurement for each vertex $u \in U$, denoted by r_t-core number $r_t(u)$. We formally introduce the definitions below.

Algorithm 11: r_t-core decomposition

```
1  Function r_t-CORE_SORT (t, G)
2      int x = |U| − 1; int md_u, pos_u[0, ..., x], L[0, ..., x];
3      int deg_u[0, ..., x]; Stack res ← ∅;
4      INIT(L[ ], md_u, deg_u[ ], deg_v[ ], G);
5      int P[0, ..., md_u]; P ← CALPOS(md_u, P);
6      foreach i = 0; i < |L|; i + + do
7          pos_u[i] ← P[deg_u[i]]; L[pos_u[i]] ← i;
8          P[deg_u[i]] ← P[deg_u[i]] + 1;
9      P[i] ← P[i − 1], ∀ i = md_u to 1;/* Reset P                    */
       /* r_t core decomposition                                      */
10     foreach i = 0; i < |L|; i + + do
11         u ← L[i]; res.push(u);
12         foreach v' ∈ n(u) do
13             if deg_v[v'] > t then
14                 deg_v[v'] ← deg_v[v'] − 1;
15                 if deg_v[v'] < t then
16                     foreach w ∈ n(v') do
17                         if d_w > d_u AND w ≠ u then
18                             SWAP(w, d_w, pos_u, deg_u, L, P);
19                             deg_u[w] ← deg_u[w] − 1;
20                             P[deg_u[w]] ← P[deg_u[w]] + 1;
21     return res;
22 Function CALPOS (md, B)
23     temp, start ← 1;
24     foreach i = 0; i ≤ md; i + + do
25         temp ← B[i]; B[i] ← start;
26         start ← start + temp;
27     return B;
28 Function SWAP (a, d_a, pos, deg, A, B)
29     p_a ← pos[a]; p_a' ← B[d_a]; a' ← A[p_a'];
30     if a ≠ a' then
31         pos[a] ← p_a'; A[p_a] ← a';
32         pos[a'] ← p_a; A[p_a'] ← a;
```

Definition 5.5. ***r_t-core.*** *Given a bipartite graph $G(U \cup V, E)$ and a threshold integer t, r_t-core is the maximal connected subgraph of G in which all vertices have degree at least r and t in the vertex sets U and V respectively.*

Definition 5.6. ***r_t-core number*** *($r_t(\cdot)$). Given a bipartite graph $G(U \cup V, E)$, for a vertex $u \in U$ and a threshold integer t, the r_t-core number of a vertex u*

$(r_t(u))$ *is the largest value of* r_t *s.t.* $u \in r_t$*-core.*

From the above definitions, we can see that: 1) a vertex u with a large degree may have a lower $r_t(u)$ and 2) a vertex u with large $r_t(u)$ has higher chance to form a t-biclique with large coverage.

We are ready to propose two heuristics below.

Heuristic 1. For any two vertices, $u_1, u_2 \in U$, if $r_t(u_1) \geq r_t(u_2)$, the coverage of a t-biclique containing u_1 may be more promising than that of a t-biclique containing u_2.

Using HEURISTIC 1, we can generate an order for U such that for any two vertices u_i and u_j in U, u_i appears before u_j if $r_t(u_i) \geq r_t(u_j)$ and $i > j$. We name this order as r_t-core order, severing for line 1 in Algorithm 10.

Heuristic 2. Given an r_t-core order of U, $u \in U$, a vertex u' is considered having higher potential to form a large coverage t-biclique together with u, if u' satisfies the following constraints: 1) u' appears after u, 2) u' is a 2-hop neighbour of u, 3) u' has the highest $r_t(u')$ among all the vertices satisfying 1) and 2), and 4) $|N(X_u) \cap N(\{u'\})|$ is maximized, where X_u is a biclique containing u.

Based on HEURISTIC 2, we are ready to propose a heuristic algorithm for generating a t-biclique with large coverage.

Large coverage t-biclique generation. For each vertex in an r_t-core order of U, we may generate a t-biclique with a large coverage by greedily applying HEURISTIC 2 $(t-1)$ times. This serves as the algorithm for line 5 in Algorithm 10. Note that this greedy algorithm may lead to a biclique that is not a t-biclique. In such a case, Algorithm 10 omits the remaining lines for this loop but tries the next vertex in U.

So far, we have discussed how to use r_t-core for heuristically generating t-bicliques with large coverage. Before further showing heuristics for generating

top-k t-bicliques, readers may be more interested in the cost of using r_t-core. We show an $\mathcal{O}(|E|)$ algorithm for deriving the r_t-core number of every vertex in U while generating an r_t-core order.

Algorithm for r_t-core. We propose to enumerate a non-increasing r_t-core number order of the vertices in U by using Algorithm 11. This algorithm uses similar framework for k-core decomposition [62]. The algorithm firstly initializes some auxiliary structures. L contains the set of vertices in U, which are later sorted by non-descending order of their degrees. The corresponding positions of vertices in L are stored in pos_u and updated accordingly. P stores the position of the first vertex with a specific degree in L and md_u is the maximum degree of a vertex in U. res is a stack where our output result of non-increasing r_t-core number is stored. During the initialization, Algorithm 11 requires $\mathcal{O}(|E|)$ for initializing md_u, L, deg_v and deg_u using INIT function, where $|E|$ is the number of edges in G. $\mathcal{O}(|L|)$ is required for updating the global starting positions in P using CALPOS and sorting the vertices in L in non-decreasing order by updating the positions of vertices in pos_u (lines 6 to 9). Each of the sorted vertex position in pos_u for L requires $\mathcal{O}|L|$ and $O(|md_u|)$ for resetting values of P (line 10). The algorithm now starts determining the r_t-core number by removing vertices from L. It iterates through all the vertices in U by the loop lines 11-24 in the order determined by L. The r_t-core number of current vertex u is the current degree of that vertex which is pushed in res. For each eligible neighbor v' of u i.e. $v' \in N(\{u\})$ s.t. $deg_v[v'] \geq t$, the algorithm reduces its degree by 1 and checks if v' can participate in an r_t-core (lines 13-23). After reducing the degree of v', if it cannot participate in the r_t-core ($deg_v[v'] < t$), the algorithm then reduces the degrees of its eligible neighbors (deg_u), updates its position in L (pos_u), and updates P as well using SWAP. An eligible neighbor w in $N(\{v'\})$ is the vertex which comes after u in the current L. For each w, SWAP requires constant time

Table 5.1: Difference between degree and r_t core number.

Vertex (U)	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9
Degree	2	4	2	3	5	3	3	3	1
r_2-core	2	2	2	2	2	3	3	2	1

and the cumulative iteration during the loops in lines 11, 13 and 17, is effectively traversing the overall edges in G. Therefore the time complexity of these steps is given as $O(|E|)$. The overall time complexity of Algorithm 11 is given as $O(|E| + |L|) = O(|E|)$.

We provide an example to illustrate our heuristic of t-biclique generation using r_t-core. As a comparison, we also show results of using the degree-based heuristic.

Example 5.5. *Table 5.1 shows the degree and r_t-core number for each vertex for Figure 5.2. Given $t = 2$, the resulting order of vertices based on non-increasing order of $d(\cdot)$ and $r_t(\cdot)$ are $(u_5, u_2, u_4, u_6, u_7, u_8, u_1, u_3, u_9)$ and $(u_6, u_7, u_1, u_2, u_3, u_4, u_5, u_8, u_9)$ respectively. We observe that forming a 2-biclique with top-2 highest degree vertices, i.e., u_5 and u_2, results in the coverage of 2 vertices. In contrast forming a 2-biclique using top-2 highest r_t-core number, i.e., u_6 and u_7, results in the coverage of 3 vertices.*

5.5.3 Heuristics for generating top-k t-bicliques

In this section, we focus on grouping the progressively generated t-bicliques into sets of top-k bicliques. Therefore, the chance of generating a set of t-bicliques with the maximum coverage is improved as there are multiple sets with large coverage for selection.

We observe that a t-biclique with the maximum coverage is important for generating a set of t-bicliques with good coverage. Apart from that, by Algorithm 10, t-bicliques are evaluated in a streaming flavour. Algorithm 10 cannot

guess a t-biclique with the maximum coverage. In contrast, Algorithm 10 may encounter several t-bicliques with the maximum coverage in turn up to the time. Based on this fact, we propose the heuristic below, which decides when a new top-k t-biclique set shall be created, i.e., the condition in line 6 in Algorithm 10.

Heuristic 3. Let $\hat{X}_{max}$ be a t-biclique with the maximum coverage before generating a t-biclique $\hat{X}$, if $|COV(\hat{X})| > |COV(\hat{X}_{max})|$, create a new top-$k$ t-biclique set.

By HEURISTICS 1, 2 and 3, the trade-off between effectiveness and efficiency of Algorithm 10 is well balanced. This is because of the following reasons. First, HEURISTIC 1 ensures large coverage t-bicliques are generated at early iterations of Algorithm 10, which makes the number of top-k sets well controlled, i.e., the cost for updating those sets is controlled. Second, by HEURISTICS 2 and 3, every generated t-biclique with a large coverage is considered, which promises the effectiveness.

Next, we discuss the heuristic for updating the top-k t-biclique sets created up to the time. It is trivial to see that if a created top-k t-biclique set $\hat{\mathfrak{X}}$ has $|\hat{\mathfrak{X}}| < k$, a new generated t-biclique $\hat{X}$ shall be included with no extra conditions. We focus on discussing how to update $\hat{\mathfrak{X}}$ with $|\hat{\mathfrak{X}}| = k$ when a new promising t-biclique is generated.

The heuristic that we will propose for updating $\hat{\mathfrak{X}}$ is inspired by the greedy rule in Algorithm 8. Refresh that $\hat{X}_{min} \in \hat{\mathfrak{X}}$ is the t-biclique in $\hat{\mathfrak{X}}$ with the minimum private coverage contribution w.r.t. $\hat{\mathfrak{X}}$.

Heuristic 4. Given $\hat{\mathfrak{X}}$, $\hat{X}_{min} \in \hat{\mathfrak{X}}$, and a newly generated t-biclique $\hat{X}$, let $\hat{\mathfrak{X}}'$ be $\hat{\mathfrak{X}} \setminus \{\hat{X}_{min}\} \cup \{\hat{X}\}$, if $COV'(\hat{X}, \hat{\mathfrak{X}}') > COV'(\hat{X}_{min}, \hat{\mathfrak{X}})$, replace $\hat{X}_{min}$ with $\hat{X}$ for $\hat{\mathfrak{X}}$.

HEURISTIC 4 captures every opportunity to enlarge COV' $(\hat{X}_{min}, \hat{\mathfrak{X}})$ for

Algorithm 12: UPDATE($\hat{X}$,$\hat{\mathbb{X}}$)

```
1  foreach X̂ ∈ 𝕏̂ do
2      if |X̂| < k then
3          X̂ ← X̂ ∪ {X̂};
4          INSERT(X̂);
5      else if |X̂| == k then
6          get X̂_min, COV'(X̂_min, X̂) from SM-INDEX;
7          DELETE(X̂_min);
8          INSERT(X̂);
9          get COV'(X̂, X̂) from SM-INDEX;
10         if |COV'(X̂, X̂)| > |COV'(X̂_min, X̂)| then
11             X̂ ← X̂ \ {X̂_min} ∪ {X̂};
12         else
13             DELETE(X̂);
14             INSERT(X̂_min);
```

improving the coverage of $\hat{\mathfrak{X}}$, which is very different from the greedy rule in Algorithm 8 that seeks for sufficient enlargement. This is because Algorithm 8 has an exponential number of t-bicliques for caviling. In contrast, the proposed heuristic algorithm only has up to $|U|$ t-bicliques to select. Algorithm 10 indeed shall cherish every opportunity for improving the coverage stated in HEURISTIC 4.

We then show a procedure for systematically updating every top-k t-biclique set using the discussed techniques.

Updating top-k t-bicliques sets. Algorithm 12 shows the major steps for updating top-k t-bicliques sets $\hat{\mathbb{X}}$, which is for line 10 in Algorithm 10.

As discussed above, HEURISTIC 4 uses private coverage contributions. As such, SM-INDEX, discussed in Section 5.4.2, is also created and maintained for each $\hat{\mathfrak{X}} \in \hat{\mathbb{X}}$ for speeding-up Algorithm 12. For $\hat{\mathfrak{X}} \in \hat{\mathbb{X}}$ with $|\hat{\mathfrak{X}}| < k$ (lines 2 to 4), Algorithm 12 simply includes $\hat{X}$ into $\hat{\mathfrak{X}}$ and updates SM-INDEX for $\hat{\mathfrak{X}}$. For $\hat{\mathfrak{X}} \in \hat{\mathbb{X}}$ with $|\hat{\mathfrak{X}}| = k$, Algorithm 12 fully takes advantage of SM-INDEX. It operates SM-INDEX to efficiently get the values of $|COV'(\hat{X}, \hat{\mathfrak{X}})|$ and $|COV'(\hat{X}_{min}, \hat{\mathfrak{X}})|$

(lines 6 to 9), which is utilized for achieving HEURISTIC 4 (lines 10 to 11). If $\hat{X}$ is tested against the heuristic rule, Algorithm 12 recovers SM-INDEX (lines 12 to 15).

Early termination. Applying HEURISTICS 1 and 4, Algorithm 10 may stop early without trying every vertex in U. This is because the reasons below. First, by the definition of r_t-core, $r_t(u)$ serves as an upper bound for the coverage of any t-biclique containing u. Then by HEURISTIC 1, the upper bound of vertices in the order is non-increasing. Therefore, by HEURISTIC 4, when an upper bound of a u is small, i.e., $r_t(u)$ is no greater than $min\{|COV'(\hat{X}_{min}, \hat{\mathfrak{X}})||\hat{\mathfrak{X}} \in \hat{\mathbb{X}}\}$, Algorithm 10 can stop (lines 4 and 12).

Time complexity of Algorithm 12. The time complexity of Algorithm 12 is $\mathcal{O}(|\hat{\mathbb{X}}|k|\hat{X}_{max}|)$. By HEURISTICS 1 and 2, $\hat{\mathbb{X}}$ is bounded by $\mathcal{O}(|U|)$ and is much less than $|U|$ as discussed. Therefore, we use $\mathcal{O}(|\hat{\mathbb{X}}|)$. As discussed in Section 5.4.2, the cost of each SM-INDEX update is bounded by $\mathcal{O}(k|\hat{X}_{max}|)$. For each loop, Algorithm 12 performs constant times of update.

Wrap up for the heuristic algorithm. The time complexity of Algorithm 10 including all components is $\mathcal{O}\left(|U|\ |\hat{\mathbb{X}}|k|\hat{X}_{max}|\right)$, which is clear based on the above discussion. The space complexity is $\mathcal{O}(|\hat{\mathbb{X}}|k|\hat{X}_{max}|)$, which is dominated by $|\hat{\mathbb{X}}|$ instances of SM-INDEX. In fact, Algorithm 10 runs much faster and takes much less space.

Example 5.6. *We display how Algorithm 10 finds results for Figure 5.2 (k = 3 and t = 2). Unlike the previous greedy approaches, it uses the proposed r_t-core ordering U = $(u_6, u_7, u_1, \ldots, u_5, u_8, u_9)$ for biclique generation. The very first 2-biclique generated for vertex u_6 is $\hat{X}_1$=$\{(u_6, u_7), (v_7, v_8, v_9)\}$ and is added to $\hat{\mathfrak{X}}$ with coverage of $COV(\hat{\mathfrak{X}})$=$\{v_7, v_8, v_9\}$. The next two generated 2-bicliques for u_7 and u_1 are also added to the top-3 results and the value $\hat{\mathfrak{X}}$=$\{\hat{X}_1, \hat{X}_2, \hat{X}_3\}$ where $\hat{X}_2$=$\{(u_7, u_5), (v_7, v_8)\}$ and $\hat{X}_3$=$\{(u_1, u_2), (v_1)\}$ with $COV(\hat{\mathfrak{X}})$=$\{v_1, v_7, v_8, v_9\}$*

and $\hat{X}_{min}$=$\hat{X}_2$ *with* $COV'(\hat{X}_2, \hat{\mathfrak{X}})$=∅. *For the first* 3 *results, we can observe the heuristic approach provides better coverage than the fast greedy approach (Example 5.4). The fourth* 2*-biclique is generated for the vertex* u_2 *is* $\hat{X}_4$ = $\{(u_2, u_4), (v_2, v_6)\}$ *which is also eligible to update* $\hat{\mathfrak{X}}$. *Consequently,* $\hat{\mathfrak{X}}$ = $\{\hat{X}_1, \hat{X}_3, \hat{X}_4\}$, $COV(\hat{\mathfrak{X}})$ = $\{v_1, v_2, v_6, v_7, v_8, v_9\}$, *and* $\hat{X}_{min}$ = $\hat{X}_3$ *with* $COV'(\hat{X}_3, \hat{\mathfrak{X}})$ = $\{v_1\}$. *After* $\hat{X}_4$, *no newly generated* 2*-biclique is eligible to update* $\hat{\mathfrak{X}}$ *and hence* $\hat{\mathfrak{X}}$ *is reported as the result of top-*3 2*-bicliques with* $COV(\hat{\mathfrak{X}})$ = $\{v_1, v_2, v_6, v_7, v_8, v_9\}$ *and* $|COV(\hat{\mathfrak{X}})|$ = 6. *We also observe the early termination due to* u_9 *with* $r_t(u_9)$ = 1 *since it is not sufficient to update the current top-*3 *results. The maximum coverage determined by the heuristic approach is the same as the fast greedy approach (Example 5.4), assuring a good coverage without enumerating all* 2*-bicliques.*

5.5.4 An Improved Fast Greedy Algorithm

The proposed heuristic approach provides various pruning techniques for further improving the running time of the fast greedy algorithm without affecting the guarantee. After applying those optimizations to Algorithm 3, Algorithm 3 can scale to a reasonably large dataset.

r_t-core based pruning. Given a vertex $u \in U$ with calculated $r_t(u)$, the maximum coverage for any biclique including u, denoted by X_u, is bounded by $r_t(u)$. As such, given any top-k t-biclique set $\hat{\mathfrak{X}}$, $r_t(u)$ serves as an upper bound for $|COV'(X_u, \hat{\mathfrak{X}}')|$, where $\hat{\mathfrak{X}}'$ is $\hat{\mathfrak{X}}$ by replacing $\hat{X}_{min} \in \hat{\mathfrak{X}}$ with X_u. We are ready to propose the pruning rule below.

Proposition 5.1. *Given a top-*k t*-biclique set* $\hat{\mathfrak{X}}$ *with* $\hat{X}_{min}$, *any vertex* u *with* $r_t(u)$ *such that*

$$|COV(\hat{\mathfrak{X}})| - |COV'(\hat{X}_{min}, \hat{\mathfrak{X}})| + r_t(u) \leq (1 + \frac{1}{k})COV(\hat{\mathfrak{X}}) \tag{5.3}$$

can be pruned.

The correctness of the above pruning is clear. This is because any t-biclique containing u has a private contribution no greater than $r_t(u)$.

The time complexity required for the pruning can be omitted. This is because SM-INDEX preserves $|COV'(\hat{X}_{min}, \hat{\mathcal{X}})|$ and $|COV(X)|$, and r_t-core number has been computed.

The above upper bound is derived by r_t-core number of the input bipartite graph. In fact, the idea can be applied to the local subgraph that each recursive sub problem in Algorithm 3 works. A tighter upper bound can be derived using the local r_t-core number, which further improves the pruning effectiveness. Note that computing the local r_t-core number will not affect the time complexity of Algorithm 9.

It is also trivial to see that applying r_t-core based pruning shall not affect the approximation guarantee of Algorithm 9. This is because the r_t-core based pruning only prunes vertices that cannot satisfy the greedy rule in Algorithm 9.

We would like to highlight that the larger the coverage of a top-k t-biclique set $\hat{\mathcal{X}}$ is, the better the pruning effectiveness is. Our proposed heuristic algorithm can derive such a large coverage top-k t-biclique set, and therefore can be used to prune lots of vertices.

Example 5.7. *We first prune the vertices in Figure 5.2 ($k = 3$ and $t = 2$) using the result of Algorithm 10. Remember that the generated top-k 2-bicliques are $\hat{\mathcal{X}} = \{\hat{X}_1, \hat{X}_3, \hat{X}_4\}$, with $|COV(\mathcal{X})| = 6$, $|COV'(\hat{X}_{min}, \mathcal{X})| = 1$. Therefore, based on Equation (3), any $u \in \{u_6, u_7, u_1, u_2, u_3, u_4, u_5, u_8, u_9\}$ with $r_t(u) \leq 3$ can be pruned. As such, there is no vertex left for Algorithm 9 and the result is still $|COV(\hat{\mathcal{X}})| = 6$ but is now certified to have the approximation guarantee provided by Algorithm 9. From this example, we can see that the large coverage top-k result provided by Algorithm 10 and the proposed r_t-core based upper*

Table 5.2: Dataset characteristics.

Datasets	$\|E\|$	$\|U\|$	$\|V\|$	Network type
Marvel	97K	7K	20K	Characters-Comics
Producer	207K	49K	139K	Producers-Works
CiteSeer	513K	105K	181K	Authors-Publications
Wiki	3.7M	183K	1.8M	Categories-Articles
Flickr	8.5M	395K	104K	Users-Groups

bound can effectively prune unpromising vertices.

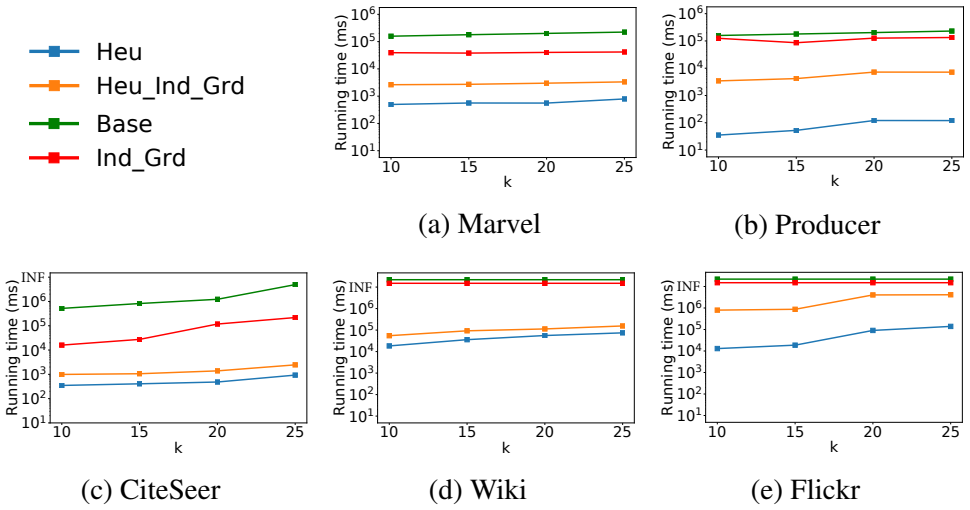
5.6 Experiments

In this section, we perform the experimental evaluations for our proposed techniques on real datasets.

Experiment setup. All the experiments were conducted on Eclipse IDE, deployed on the platform 64x Intel(R)Core(TM) i5-6400T with CPU frequency 2.20GHz and 8 GB RAM, running Windows 10 Enterprise operating system. All the algorithms were implemented in Java.

Algorithms. We have implemented the following four algorithms for comparison in our experiments:

- BASE: it is the algorithm implementing Algorithm 7.
- IND_GRD: it is an implementation of the fast greedy algorithm with the index as shown in Algorithm 9.
- HEU: it is an implementation of the heuristic approach as shown in Algorithm 10.
- HEU_IND_GRD: this algorithm represents an improved fast greedy algorithm, where it firstly prunes vertices using the heuristic approach HEU (Algorithm 10) and then enumerates top-k t-bicliques using IND_GRD and the optimizations proposed in Section 5.5.4.



(a) Marvel (b) Producer

(c) CiteSeer (d) Wiki (e) Flickr

Figure 5.3: Efficiency with varying k.

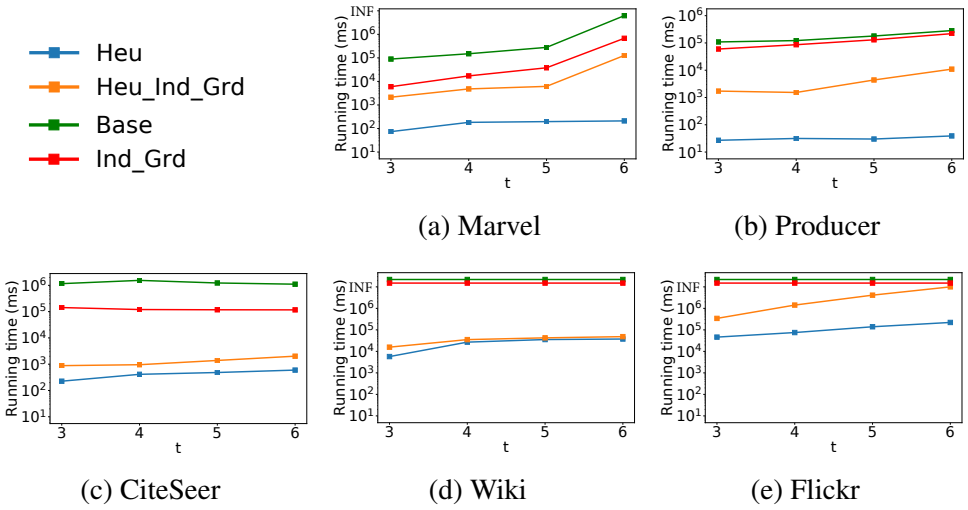
For our experiments, we consider the threshold for the smaller vertex set U and maximize the coverage in the larger vertex set V for all the algorithms.

Datasets. Five real datasets, obtained from the KONECT repository [215], are used to evaluate our algorithms. Table 5.2 shows the characteristics of the datasets.

Parameter settings. We evaluate our experiments by utilizing different settings of the query parameters: t (size constraint used for a t-biclique) and k (top-k values for the corresponding results). For all the datasets, the value of k is selected from 10, 15, 20 and 25, and the value of t can be 3, 4, 5 and 6.

5.6.1 Efficiency

Varying k. We vary the value of k and keep the value of t to be fixed for each dataset. The corresponding running time curves for the four algorithms on each dataset with t=5 are shown in Figure 5.3. We observe the following from these experiments. (i) As expected, BASE is the least efficient algorithm among all the algorithms, and for the datasets *Wiki* and *Flickr*, the algorithm runs out of memory due to storing all t-bicliques. IND_GRD is the second slowest algorithm



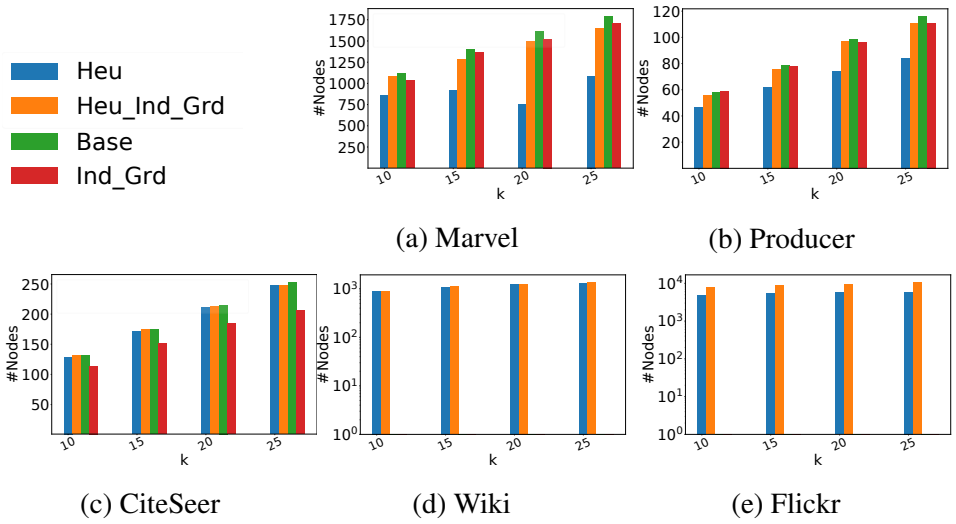
(a) Marvel (b) Producer

(c) CiteSeer (d) Wiki (e) Flickr

Figure 5.4: Efficiency with varying t.

which only outperforms BASE across all the datasets. HEU_IND_GRD is at least ten folds faster than IND_GRD due to prunings induced by heuristic lower bound and further optimizations. As expected, HEU is the fastest algorithm among all and executes efficiently even for the larger datasets as it does not include any exhaustive enumerations. (ii) Although the size of *Producer* is larger than *Marvel*, we observe the time difference between BASE and IND_GRD is significantly higher in *Marvel* as it is much denser. (iii) For large datasets *Wiki* and *Flickr*, BASE runs out of memory and IND_GRD cannot finish within the cut-off time (3 hours). HEU_IND_GRD is slightly slower than HEU since HEU_IND_GRD is an extension of HEU and HEU can find good results on the two datasets.

Varying t. In this set of experiments, we keep the value of k to be constant, i.e., k=15, for each dataset and vary the value of t. The corresponding running time curves for the four algorithms on each dataset are shown in Figure 5.4. We observe the following from these experiments. (i) As observed in previous set of experiments, BASE is yet again the least efficient and HEU is the fastest algorithm among all. Moreover, it runs out of memory for t=6 for *Marvel* dataset as it is dense and produces a large number of t-bicliques. (ii) For all the algorithms,



(a) Marvel (b) Producer

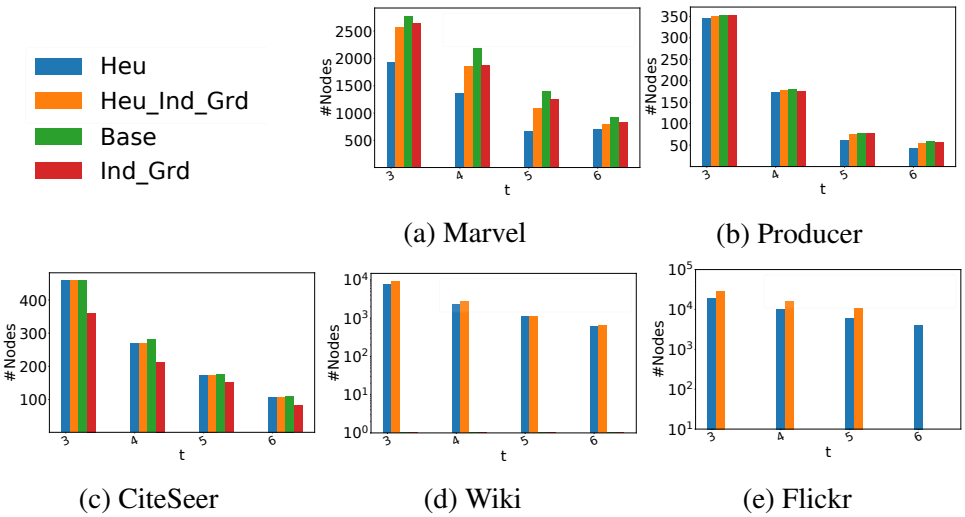
(c) CiteSeer (d) Wiki (e) Flickr

Figure 5.5: Effectiveness with varying k.

the running time steadily increases with the increase in t. For the small datasets *Marvel* and *Producer*, varying t does not affect the performance of HEU greatly. (iii) For *CiteSeer* and *Wiki*, the running times of HEU and HEU_IND_GRD are comparable, indicating the high extent of pruning power provided by our proposed heuristics and consequently providing similar coverage as observed later in Figure 5.6. For the largest dataset *Flickr*, BASE and IND_GRD cannot finish within the cut-off time. In contrast HEU_IND_GRD is able to provide results for all the values of t apart from 6. (iv) Although the number of pruned vertices using the lower bound provided by HEU is significant across the datasets *Marvel* (96%-57%), *Producer* (99%-75%), *CiteSeer* (99%-95%), *Wiki* (99%-98%) and *Flickr* (99%-98%), the running time of HEU_IND_GRD increases due to its high enumeration cost growing exponentially w.r.t. t.

5.6.2 Effectiveness

We perform analyses for effectiveness utilizing the same tuning of t and k for all the datasets. The difference lies in the resulting graphs, wherein instead of the running time we report the number of vertices covered by each algorithm.



(a) Marvel (b) Producer

(c) CiteSeer (d) Wiki (e) Flickr

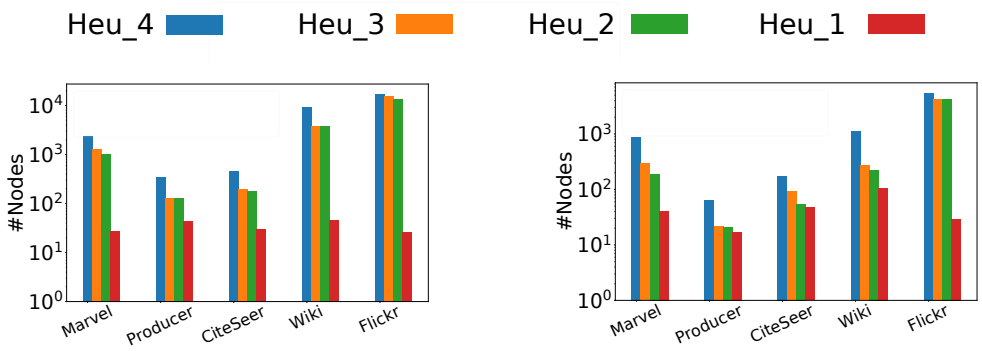
Figure 5.6: Effectiveness with varying t.

Varying k. In the first set of experiments, we vary k and keep t to be fixed (same values used previously so as to relate the efficiency and effectiveness of an algorithm) for each dataset. The corresponding coverage curves for all the algorithms on each dataset are shown in Figure 5.5. For all the algorithms, the coverage increases with the increase in k, which is obvious. We provide the following further observations. (i) BASE provides the maximum coverage across all the datasets which is as expected and it is also evident from its approximation ratio. However, generating all t-bicliques using BASE is impractical for reasonably large datasets such as *Wiki* and *Flickr* where the algorithm runs out of memory. (ii) The coverage of HEU_IND_GRD and IND_GRD is close to that of BASE even though their approximation ratio are different. (iii) The coverage of HEU becomes more comparable to that of HEU_IND_GRD as the size of dataset increases. The coverage of HEU is at least 60% of that of HEU_IND_GRD across all the datasets, which shows HEU can provide large coverage for large datasets quickly.

Varying t. For each dataset, we vary t, fix k (same values used previously) and report the coverage. The results for all datasets are shown in Figure 5.6. We provide the following observations. (i) For all the algorithms the cover-

age decreases with the increase in t since the individual coverage of the bicliques is reduced. (ii) BASE provides the largest coverage across all the datasets but runs out of memory for dense and large datasets. (iii) The coverage of HEU_IND_GRD and IND_GRD are comparable to that of BASE across all the datasets. (iv) For the dense datasets, the difference between HEU and HEU_IND_GRD is significant since each vertex with a large r_t-core number can participate in multiple t-bicliques and is found using enumeration which is currently not included by the HEU approach. However, for large datasets, the difference of HEU and HEU_IND_GRD is reduced. (v) In Figure 5.6, for small datasets such as *Marvel*, the coverage of HEU could cover as low as 50% of that of HEU_IND_GRD. For *CiteSeer* dataset, HEU and HEU_IND_GRD provide same coverage which is at least 95% of BASE. In contrast, for larger datasets like *Wiki* and *Flickr*, the coverage provided by HEU is at least 80% and 60% of that of HEU_IND_GRD respectively. Note that for $t = 6$, only HEU provides coverage within the cut-off time. Since the real datasets are most likely large and sparse, the trade-off between efficiency and effectiveness provided by HEU can be considered acceptable compared to other algorithms that either run out of memory or take several hours to execute.

Heuristics analyses. We now compare the effectiveness of the four heuristics proposed in Section 5.5. We utilize the incremental evaluation approach resulting in four heuristic algorithms: (i) HEU_1 orders the input vertices using *Heuristic 1* and finds k t-bicliques by grouping high degree vertices in U. (ii) HEU_2 orders the input vertices using *Heuristic 1* and generates each t-biclique using *Heuristic 2*. (iii) HEU_3 maintains several sets of k t-bicliques as proposed *Heuristic 3*, where each set is generated using HEU_2. (iv) HEU_4 is the same as HEU using all four heuristics. k is set to be 15 for all the datasets and the coverage of each algorithm is reported in Figures 5.7 (a) and (b) for



(a) t = 3 (b) t = 5

Figure 5.7: Effectiveness comparison for Heuristics 1,2,3 and 4.

t of 3 and 5 respectively. We observe that HEU_4 outperforms all the other algorithms. This is because only HEU_4 uses *Heuristic 4* to update the obtained k results, further enlarging the coverage. HEU_3 has no worse coverage than HEU_2 since the resulting set of t-bicliques in HEU_2 is also presented in HEU_3.The coverage of HEU_1 is the least across all the datasets.

5.7 Summary

In this chapter, we study the problem of top-k t-biclique coverage in bipartite graphs. We adapt the fast greedy algorithm and propose a novel heuristic approach with four heuristic rules based on the proposed r_t-core number. The heuristic approach can serve as a reasonable standalone algorithm for addressing the problem when datasets are large. Moreover, it is extended to serve as a lower bound for pruning vertices leading to unpromising results and thereby improve the performance of the fast greedy algorithm significantly while preserving the approximation guarantee. We conduct extensive experiments across real datasets to justify the efficiency and effectiveness of our proposed algorithms.

Chapter 6

Conclusion and Future Work

In this chapter, we briefly summarize the major contributions in this thesis in Section 6.1 and discuss some interesting future research directions that can be further explored in Section 6.2.

6.1 Conclusion

We study how to search bipartite cohesive subgraphs and propose efficient algorithms finding the respective bipartite cohesive subgraphs. In particular, we study and explore three significant bipartite cohesive subgraph models, i.e., biclique, personalized k-wing and top-k t-biclique, with different application scenarios, summarized as follows.

Firstly, we examine the problem of enumerating all maximal bicliques from a bipartite graph in Chapter 3. We propose the PMBE algorithm, which incorporates the pivot pruning using the CDAG structure. We also present the rev-topological ordering for heuristic pivot selection. The PMBE algorithm incorporates advanced pruning and ordering techniques, reducing the search space without introducing extra local costs. To demonstrate the efficiency of PMBE over the previous algorithms, we then conduct extensive experiments on real-

world datasets across various domains.

Secondly, we study the problem of personalized k-wing search for dynamic bipartite graphs in Chapter 4. This chapter presents two indices, *EquiWing* and *EquiWing-Comp*, further utilized for linear time search algorithms to tackle the problem efficiently. We construct *EquiWing-Comp* by proposing the *k-butterfly loose connectivity* and exploiting the hierarchical property of the k-wing, which further speeds up the query processing. For both the indices, we also study their efficient maintenance in dynamic bipartite graphs. We perform extensive experiments across large real-world datasets to demonstrate the superiority of our index-based approaches *EquiWing* and *EquiWing-Comp* over the baseline approach. These experiments also display the better performance and compression of *EquiWing-Comp* over *EquiWing*. Moreover, we present a case study to indicate the effectiveness of our personalized k-wing model.

Lastly, we study the problem of top-k t-biclique coverage in bipartite graphs in Chapter 5. We start by adapting the fast greedy algorithm and then propose a novel heuristic approach with four heuristic rules based on the proposed r_t-core number. We observe that the heuristic approach serves as a reasonable standalone algorithm for addressing the problem in large datasets. Moreover, we extend the heuristic approach to serve as a lower bound for pruning vertices leading to unpromising results and thereby improving the performance of the fast greedy algorithm significantly while preserving the approximation guarantee. To justify the efficiency and effectiveness of our proposed algorithms, we conduct extensive experiments across real-world datasets.

6.2 Future Work

The exploration of searching bipartite cohesive subgraphs is still far from an end. Bipartite graphs are rapidly being adopted across more new domains, lead-

ing to new applications with unique challenges. We now propose several possible directions for future work.

Extending the proposed models to other graphs. In this thesis, we explore the structural cohesiveness of a bipartite subgraph. However, many real-world bipartite networks contain attributes (location, timestamp or keyword) on the nodes. It would be interesting to incorporate such attributes and structural cohesiveness to determine new attributed bipartite cohesive subgraphs. The resulting attributed bipartite cohesive subgraphs could represent a group of entities with some semantic information, such as a group of closely working researchers within a specific location or are active in the same period of time. Several cohesive subgraph models have been proposed for attributed graphs (k, d)-MCCs [102], (k,r)-core [221], and r-clique [222]. Unfortunately, these works are mainly focused on unipartite graphs. Therefore, extending the personalized k-wing and top-k t-biclique models to the attributed bipartite graphs would be interesting.

Efficient I/O algorithms for Big Data. Real-world bipartite graphs are growing at an exponential pace. The number of edges and vertices could be in billions or even trillions, i.e., Big Data. Even though pruning techniques have reduced the search space for maximal biclique enumeration, the problem still requires exponential time complexity. Therefore, the existing algorithms might fail to process such large bipartite graphs reasonably. Several I/O efficient algorithms have been proposed for finding cohesive subgraph models such as cliques [152], k-ECCs [223], k-core [121] and k-truss [65]. Recently few I/O efficient algorithms [40], [166] have been proposed in bipartite graphs for (α, β)-core model. However, it would be interesting to see similar approaches for the more cohesive bipartite subgraphs like k-wing and biclique.

Bibliography

[1] T. Washio and H. Motoda, "State of the art of graph-based data mining," *SIGKDD Explorations*, vol. 5, no. 1, pp. 59–68,

[2] R. J. Mokken, "Cliques, clubs and clans," *Quality & Quantity*, vol. 13, no. 2, pp. 161–173, 1979.

[3] J. Cohen, "Trusses: Cohesive subgraphs for social network analysis," *National Security Agency Technical Report*, vol. 16, pp. 3–1, 2008.

[4] S. B. Seidman, "Network structure and minimum degree," *Social networks*, vol. 5, no. 3, pp. 269–287, 1983.

[5] J. Cheng, Y. Ke, S. Chu, and M. T. Özsu, "Efficient core decomposition in massive networks," in *2011 IEEE 27th International Conference on Data Engineering*, IEEE, 2011, pp. 51–62.

[6] J. Wang, A. P. De Vries, and M. J. Reinders, "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*, 2006, pp. 501–508.

[7] M. Kaytoue, S. O. Kuznetsov, A. Napoli, and S. Duplessis, "Mining gene expression data with pattern structures in formal concept analysis," *Information Sciences*, vol. 181, no. 10, pp. 1989–2001, 2011.

[8] I. S. Dhillon, "Co-clustering documents and words using bipartite spectral graph partitioning," in *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '01, San Francisco, California: ACM, 2001, pp. 269–274, ISBN: 1-58113-391-X.

[9] M. E. Newman, "The structure of scientific collaboration networks," *Proceedings of the national academy of sciences*, vol. 98, no. 2, pp. 404–409, 2001.

[10] D. Gibson, R. Kumar, and A. Tomkins, "Discovering large dense subgraphs in massive graphs," in *Proceedings of the 31st international conference on Very large data bases*, Citeseer, 2005, pp. 721–732.

[11] R. Schweiger, M. Linial, and N. Linial, "Generative probabilistic models for protein–protein interaction networks—the biclique perspective," *Bioinformatics*, vol. 27, no. 13, pp. i142–i148, 2011.

[12] M. K. Vanahalli and N. Patil, "An efficient parallel row enumerated algorithm for mining frequent colossal closed itemsets from high dimensional datasets," *Information Sciences*, vol. 496, pp. 343–362, 2019.

[13] D. C. Fain and J. O. Pedersen, "Sponsored search: A brief history," *Bulletin of the american Society for Information Science and technology*, vol. 32, no. 2, pp. 12–13, 2006.

[14] B. Liu, "Efficient core computation in bipartite and multilayer graphs.," Ph.D. dissertation, 2020.

[15] D. Ding, H. Li, Z. Huang, and N. Mamoulis, "Efficient fault-tolerant group recommendation using alpha-beta-core," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, 2017, pp. 2047–2050.

[16] K. Wang, X. Lin, L. Qin, W. Zhang, and Y. Zhang, "Efficient bitruss decomposition for large-scale bipartite graphs," in *ICDE*, IEEE, 2020, pp. 661–672.

[17] A. E. Sariyüce and A. Pinar, "Peeling bipartite networks for dense subgraph discovery," in *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, WSDM 2018, Marina Del Rey, CA, USA, February 5-9, 2018*, 2018, pp. 504–512.

[18] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins, "Trawling the web for emerging cyber-communities," *Computer networks*, vol. 31, no. 11-16, pp. 1481–1493, 1999.

[19] M. Allahbakhsh, A. Ignjatovic, B. Benatallah, S.-M.-R. Beheshti, E. Bertino, and N. Foo, "Collusion detection in online rating systems," in *Asia-Pacific Web Conference*, Springer, 2013, pp. 196–207.

[20] A. Beutel, W. Xu, V. Guruswami, C. Palow, and C. Faloutsos, "Copycatch: Stopping group attacks by spotting lockstep behavior in social networks," in *Proceedings of the 22nd international conference on World Wide Web*, 2013, pp. 119–130.

[21] T. Alzahrani and K. J. Horadam, "Finding maximal bicliques in bipartite networks using node similarity," *Applied Network Science*, vol. 4, no. 1, 21:1–21:25, 2019.

[22] D. Gibson, R. Kumar, and A. Tomkins, "Discovering large dense subgraphs in massive graphs," in *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005*, 2005, pp. 721–732.

[23] Y. Xiang, P. R. Payne, and K. Huang, "Transactional database transformation and its application in prioritizing human disease genes," *IEEE/ACM*

transactions on computational biology and bioinformatics, vol. 9, no. 1, pp. 294–304, 2011.

[24] A. C. Driskell, C. Ané, J. G. Burleigh, M. M. McMahon, B. C. O'Meara, and M. J. Sanderson, "Prospects for building the tree of life from large sequence databases," *Science*, vol. 306, no. 5699, pp. 1172–1174, 2004.

[25] L. Wu, L. Dong, Y. Wang, *et al.*, "Uniform-scale assessment of role minimization in bipartite networks and its application to access control," *Physica A: Statistical Mechanics and its Applications*, vol. 507, pp. 381–397, 2018.

[26] R. Yoshinaka, "Towards dual approaches for learning context-free grammars based on syntactic concept lattices," in *Developments in Language Theory - 15th International Conference, DLT 2011, Milan, Italy, July 19-22, 2011. Proceedings*, 2011, pp. 429–440.

[27] S. Mouret, I. E. Grossmann, and P. Pestiaux, "Time representations and mathematical models for process scheduling problems," *Computers & Chemical Engineering*, vol. 35, no. 6, pp. 1038–1063, 2011.

[28] Y. Zhang, C. A. Phillips, G. L. Rogers, E. J. Baker, E. J. Chesler, and M. A. Langston, "On finding bicliques in bipartite graphs: A novel algorithm and its application to the integration of diverse biological data types," *BMC bioinformatics*, vol. 15, no. 1, p. 110, 2014.

[29] J. Li, G. Liu, H. Li, and L. Wong, "Maximal biclique subgraphs and closed pattern pairs of the adjacency matrix: A one-to-one correspondence and mining algorithms," *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 12, pp. 1625–1637, 2007.

[30] P. Damaschke, "Enumerating maximal bicliques in bipartite graphs with favorable degree sequences," *Information Processing Letters*, vol. 114, no. 6, pp. 317–321, 2014.

[31] D. Gibson, R. Kumar, and A. Tomkins, "Discovering large dense subgraphs in massive graphs," in *Proceedings of the 31st International Conference on Very Large Data Bases*, ser. VLDB '05, Trondheim, Norway: VLDB Endowment, 2005, pp. 721–732, ISBN: 1-59593-154-6.

[32] S. Aksoy, T. G. Kolda, and A. Pinar, "Measuring and modeling bipartite graphs with community structure," *J. Complex Networks 2017*, vol. 5, no. 4, pp. 581–603,

[33] S.-V. Sanei-Mehri, A. E. Sariyuce, and S. Tirthapura, "Butterfly counting in bipartite networks," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '18, London, United Kingdom: ACM, 2018, pp. 2150–2159, ISBN: 978-1-4503-5552-0.

[34] K. Wang, X. Lin, L. Qin, W. Zhang, and Y. Zhang, "Vertex priority based butterfly counting for large-scale bipartite networks," *PVLDB*, vol. 12, no. 10, pp. 1139–1152, 2019.

[35] K. Wang, X. Lin, L. Qin, W. Zhang, and Y. Zhang, "Towards efficient solutions of bitruss decomposition for large-scale bipartite graphs," *The VLDB Journal*, pp. 1–24, 2021.

[36] C.-T. Lu, S. Xie, X. Kong, and P. S. Yu, "Inferring the impacts of social media on crowdfunding," in *WSDM 2014*, pp. 573–582.

[37] V. Arnaboldi, R. I. M. Dunbar, A. Passarella, and M. Conti, "Analysis of co-authorship ego networks," in *Advances in Network Science*, A. Wierzbicki, U. Brandes, F. Schweitzer, and D. Pedreschi, Eds., Cham:

Springer International Publishing, 2016, pp. 82–96, ISBN: 978-3-319-28361-6.

[38] M. Sozio and A. Gionis, "The community-search problem and how to plan a successful cocktail party," in *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '10, Washington, DC, USA: ACM, 2010, pp. 939–948, ISBN: 978-1-4503-0055-1.

[39] V. Huettl-Maack and J. Schwenk, "Effects of multilingual product packaging on product attitude, perceived quality, and taste perceptions," in *Advances in Advertising Research (Vol. VI) 2016*, pp. 351–363.

[40] K. Wang, W. Zhang, X. Lin, Y. Zhang, L. Qin, and Y. Zhang, "Efficient and effective community search on large-scale bipartite graphs," in *37th IEEE International Conference on Data Engineering, ICDE 2021*, IEEE, 2021, pp. 85–96.

[41] B. d. P. F. e Fonseca, R. B. Sampaio, M. V. de Araújo Fonseca, and F. Zicker, "Co-authorship network analysis in health research: Method and potential use," *Health Research Policy and Systems 2016*, vol. 14, no. 1, pp. 1–10,

[42] R. Kumar, J. Novak, and A. Tomkins, "Structure and evolution of online social networks," in *Link mining: models, algorithms, and applications*, Springer, 2010, pp. 337–357.

[43] R. V. Oliveira, B. Zhang, and L. Zhang, "Observing the evolution of internet as topology," in *SIGCOMM*, 2007, pp. 313–324.

[44] A. Abbasi, L. Hossain, and L. Leydesdorff, "Betweenness centrality as a driver of preferential attachment in the evolution of research collab-

oration networks," *Journal of Informetrics*, vol. 6, no. 3, pp. 403–412, 2012.

[45] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu, "Querying k-truss community in large and dynamic graphs," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, ACM, 2014, pp. 1311–1322.

[46] A. Abidi, R. Zhou, L. Chen, and C. Liu, "Pivot-based maximal biclique enumeration," in *IJCAI*, C. Bessiere, Ed., Jul. 2020, pp. 3558–3564.

[47] A. Albano and A. P. do Lago, "A convexity upper bound for the number of maximal bicliques of a bipartite graph," *Discret. Appl. Math.*, vol. 165, pp. 12–24, 2014.

[48] T. Alzahrani and K. J. Horadam, "Finding maximal bicliques in bipartite networks using node similarity," *Appl. Netw. Sci.*, vol. 4, no. 1, 21:1–21:25, 2019.

[49] P. Damaschke, "Enumerating maximal bicliques in bipartite graphs with favorable degree sequences," *Information Processing Letters*, vol. 114, no. 6, pp. 317–321, 2014.

[50] A. Das and S. Tirthapura, "Incremental maintenance of maximal bicliques in a dynamic bipartite graph," *IEEE Trans. Multi Scale Comput. Syst.*, vol. 4, no. 3, pp. 231–242, 2018. DOI: 10.1109/TMSCS.2018.2802920. [Online]. Available: https://doi.org/10.1109/TMSCS.2018.2802920.

[51] D. Hermelin and G. Manoussakis, "Efficient enumeration of maximal induced bicliques," *Discret. Appl. Math.*, vol. 303, pp. 253–261, 2021.

[52] A. P. Mukherjee and S. Tirthapura, "Enumerating maximal bicliques from a large graph using mapreduce," *IEEE Trans. Serv. Comput.*, vol. 10, no. 5, pp. 771–784, 2017.

[53] L. Chen, C. Liu, R. Zhou, J. Xu, and J. Li, "Efficient exact algorithms for maximum balanced biclique search in bipartite graphs," in *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, ACM, 2021, pp. 248–260.

[54] ——, "Efficient maximal biclique enumeration for large sparse bipartite graphs," *Proceedings of the VLDB Endowment*, vol. 15, no. 8, 2022. DOI: 10.14778/3529337.3529341.

[55] R. Wang, M. Liao, and C. Qin, "An efficient algorithm for enumerating maximal bicliques from a dynamically growing graph," in *The International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery*, Springer, 2019, pp. 329–337.

[56] B. Ding, J. X. Yu, S. Wang, L. Qin, X. Zhang, and X. Lin, "Finding top-k min-cost connected trees in databases," in *IEEE 23rd International Conference on Data Engineering*, IEEE, 2007, pp. 836–845.

[57] M. Gupta, J. Gao, X. Yan, H. Cam, and J. Han, "Top-k interesting subgraph discovery in information networks," in *IEEE 30th International Conference on Data Engineering*, IEEE, 2014, pp. 820–831.

[58] Z. Yang, A. W. Fu, and R. Liu, "Diversified top-k subgraph querying in a large graph," in *SIGMOD '16: International Conference on Management of Data, San Francisco, CA, USA, June 26 - July 01, 2016*, ACM, 2016, pp. 1167–1182. DOI: 10.1145/2882903.2915216. [Online]. Available: https://doi.org/10.1145/2882903.2915216.

[59] E. Galbrun, A. Gionis, and N. Tatti, "Top-k overlapping densest subgraphs," *Data Mining and Knowledge Discovery*, vol. 30, no. 5, pp. 1134–1165, 2016.

[60] U. Feige, "A threshold of ln n for approximating set cover," *Journal of the ACM (JACM)*, vol. 45, no. 4, pp. 634–652, 1998.

[61] E. Akbas and P. Zhao, "Truss-based community search: A truss-equivalence based indexing approach," *Proceedings of the VLDB Endowment*, vol. 10, no. 11, pp. 1298–1309, 2017.

[62] V. Batagelj and M. Zaversnik, "An o(m) algorithm for cores decomposition of networks," *CoRR*, vol. cs.DS/0310049, 2003.

[63] J. Wang and J. Cheng, "Truss decomposition in massive networks," *Proc. VLDB Endow.*, vol. 5, no. 9, pp. 812–823, May 2012, ISSN: 2150-8097.

[64] X. Huang, W. Lu, and L. V. Lakshmanan, "Truss decomposition of probabilistic graphs: Semantics and algorithms," in *Proceedings of the 2016 International Conference on Management of Data*, 2016, pp. 77–90.

[65] Z. Zou and R. Zhu, "Truss decomposition of uncertain graphs," *Knowledge and Information Systems*, vol. 50, no. 1, pp. 197–230, 2017.

[66] R. D. Luce and A. D. Perry, "A method of matrix analysis of group structure," *Psychometrika*, vol. 14, no. 2, pp. 95–116, 1949.

[67] J. Cheng, Y. Ke, A. W.-C. Fu, J. X. Yu, and L. Zhu, "Finding maximal cliques in massive networks," *ACM Transactions on Database Systems (TODS)*, vol. 36, no. 4, pp. 1–34, 2011.

[68] C. Bron and J. Kerbosch, "Algorithm 457: Finding all cliques of an undirected graph," *Communications of the ACM*, vol. 16, no. 9, pp. 575–577, 1973.

[69] E. A. Akkoyunlu, "The enumeration of maximal cliques of large graphs," *SIAM Journal on Computing*, vol. 2, no. 1, pp. 1–6, 1973.

[70] E. Tomita, A. Tanaka, and H. Takahashi, "The worst-case time complexity for generating all maximal cliques and computational experiments," *Theoretical Computer Science*, vol. 363, no. 1, pp. 28–42, 2006.

[71] R. Zhou, C. Liu, J. X. Yu, W. Liang, B. Chen, and J. Li, "Finding maximal k-edge-connected subgraphs from a large graph," in *EDBT*, 2012, pp. 480–491.

[72] R. S. Burt *et al.*, *Brokerage and closure: An introduction to social capital*. Oxford university press, 2005.

[73] L. Chang, J. X. Yu, L. Qin, X. Lin, C. Liu, and W. Liang, "Efficiently computing k-edge connected components via graph decomposition," in *SIGMOD*, 2013, pp. 205–216.

[74] S. B. Seidman and B. L. Foster, "A graph-theoretic generalization of the clique concept," *Journal of Mathematical sociology*, vol. 6, no. 1, pp. 139–154, 1978.

[75] D. Berlowitz, S. Cohen, and B. Kimelfeld, "Efficient enumeration of maximal k-plexes," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, 2015, pp. 431–444.

[76] B. Wu and X. Pei, "A parallel algorithm for enumerating all the maximal k-plexes," in *Pacific-Asia conference on knowledge discovery and data mining*, Springer, 2007, pp. 476–483.

[77] A. E. Sariyuce, C. Seshadhri, A. Pinar, and U. V. Catalyurek, "Finding the hierarchy of dense subgraphs using nucleus decompositions," in *Proceedings of the 24th International Conference on World Wide Web*, 2015, pp. 927–937.

[78] S. Morris, "Contagion," *The Review of Economic Studies*, vol. 67, no. 1, pp. 57–78, 2000.

[79] A. Ahmed, V. Batagelj, X. Fu, S.-H. Hong, D. Merrick, and A. Mrvar, "Visualisation and analysis of the internet movie database," in *2007 6th International Asia-Pacific Symposium on Visualization*, IEEE, 2007, pp. 17–24.

[80] M. Cerinšek and V. Batagelj, "Generalized two-mode cores," *Social Networks*, vol. 42, pp. 80–87, 2015.

[81] B. Liu, L. Yuan, X. Lin, L. Qin, W. Zhang, and J. Zhou, "Efficient (α, β)-core computation: An index-based approach," in *The World Wide Web Conference*, 2019, pp. 1130–1141.

[82] R. Zhou, C. Liu, J. X. Yu, W. Liang, and Y. Zhang, "Efficient truss maintenance in evolving networks," *arXiv preprint arXiv:1402.2807*, 2014.

[83] J. Wang, A. W.-C. Fu, and J. Cheng, "Rectangle counting in large bipartite graphs," in *2014 IEEE International Congress on Big Data*, IEEE, 2014, pp. 17–24.

[84] S.-V. Sanei-Mehri, A. E. Sariyuce, and S. Tirthapura, "Butterfly counting in bipartite networks," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 2150–2159.

[85] K. Wang, X. Lin, L. Qin, W. Zhang, and Y. Zhang, "Vertex priority based butterfly counting for large-scale bipartite networks.," *PVLDB*, 2019.

[86] D. Eppstein, "Arboricity and bipartite subgraph listing algorithms," *Information processing letters*, vol. 51, no. 4, pp. 207–211, 1994.

[87] M. R. Garey, "Computers and intractability: A guide to the theory of np-completeness," *Revista Da Escola De Enfermagem Da USP*, vol. 44, no. 2, p. 340, 1979.

[88] S. Lonardi, W. Szpankowski, and Q. Yang, "Finding biclusters by random projections," *Theoretical Computer Science*, vol. 368, no. 3, pp. 217–230, 2006.

[89] K. Sim, J. Li, V. Gopalkrishnan, and G. Liu, "Mining maximal quasi-bicliques to co-cluster stocks and financial ratios for value investment," in *Sixth International Conference on Data Mining (ICDM'06)*, IEEE, 2006, pp. 1059–1063.

[90] H.-F. Liu, C.-T. Su, and A.-C. Chu, "Fast quasi-biclique mining with giraph," in *2013 IEEE International Congress on Big Data*, IEEE, 2013, pp. 347–354.

[91] J. Li, K. Sim, G. Liu, and L. Wong, "Maximal quasi-bicliques with balanced noise tolerance: Concepts and co-clustering applications," in *Proceedings of the SIAM International Conference on Data Mining, SDM 2008, April 24-26, 2008, Atlanta, Georgia, USA*, SIAM, 2008, pp. 72–83.

[92] T. Schank and D. Wagner, "Approximating clustering coefficient and transitivity.," *Journal of Graph Algorithms and Applications*, vol. 9, no. 2, pp. 265–275, 2005.

[93] Y. Yang, Y. Fang, M. E. Orlowska, W. Zhang, and X. Lin, "Efficient bi-triangle counting for large bipartite networks," *Proceedings of the VLDB Endowment*, vol. 14, no. 6, pp. 984–996, 2021.

[94] J. Wang, J. Cheng, and A. W. Fu, "Redundancy-aware maximal cliques," in *KDD '19: The 19th ACM SIGKDD Conference on Knowledge Dis-*

covery and Data Mining, Chicago, IL, USA, August 11-14, 2013, ACM, 2013, pp. 122–130. DOI: 10.1145/2487575.2487689. [Online]. Available: https://doi.org/10.1145/2487575.2487689.

[95] X. Li, R. Zhou, Y. Dai, *et al.*, "Mining maximal clique summary with effective sampling," in *IEEE International Conference on Data Mining*, IEEE, 2019, pp. 1198–1203.

[96] X. Li, R. Zhou, L. Chen, *et al.*, "Finding a summary for all maximal cliques," in *37th IEEE International Conference on Data Engineering, ICDE 2021*, IEEE, 2021, pp. 1344–1355.

[97] L. Yuan, L. Qin, X. Lin, L. Chang, and W. Zhang, "Diversified top-k clique search," *The VLDB Journal*, vol. 25, no. 2, pp. 171–196, 2016.

[98] L. Lin, P. Yuan, R.-H. Li, J. Wang, L. Liu, and H. Jin, "Mining stable quasi-cliques on temporal networks," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2021.

[99] A. A. Chowdhary, C. Liu, L. Chen, R. Zhou, and Y. Yang, "Finding attribute diversified communities in complex networks," in *International Conference on Database Systems for Advanced Applications*, Springer, 2020, pp. 19–35.

[100] L. Chen, C. Liu, R. Zhou, J. Xu, J. X. Yu, and J. Li, "Finding effective geo-social group for impromptu activities with diverse demands," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 698–708.

[101] L. Chen, C. Liu, K. Liao, J. Li, and R. Zhou, "Contextual community search over large social networks," in *35th IEEE International Conference on Data Engineering, ICDE 2019*, IEEE, 2019, pp. 88–99.

[102] L. Chen, C. Liu, R. Zhou, J. Li, X. Yang, and B. Wang, "Maximum co-located community search in large scale social networks," *Proceedings of the VLDB Endowment*, vol. 11, no. 10, pp. 1233–1246, 2018.

[103] Y. Fang, R. Cheng, S. Luo, and J. Hu, "Effective community search for large attributed graphs," *Proceedings of the VLDB Endowment*, vol. 9, no. 12, pp. 1233–1244, 2016.

[104] Y. Fang, R. Cheng, Y. Chen, S. Luo, and J. Hu, "Effective and efficient attributed community search," *The VLDB Journal*, vol. 26, no. 6, pp. 803–828, 2017.

[105] Y. Fang, R. Cheng, X. Li, S. Luo, and J. Hu, "Effective community search over large spatial graphs," *Proceedings of the VLDB Endowment*, vol. 10, no. 6, pp. 709–720, 2017.

[106] Y. Fang, Z. Wang, R. Cheng, *et al.*, "On spatial-aware community search," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 4, pp. 783–798, 2018.

[107] C. Giatsidis, D. M. Thilikos, and M. Vazirgiannis, "Evaluating cooperation in communities with the k-core structure," in *2011 International conference on advances in social networks analysis and mining*, IEEE, 2011, pp. 87–93.

[108] C. Giatsidis, D. M. Thilikos, and M. Vazirgiannis, "D-cores: Measuring collaboration of directed graphs based on degeneracy," *Knowledge and information systems*, vol. 35, no. 2, pp. 311–343, 2013.

[109] Y. Fang, Z. Wang, R. Cheng, H. Wang, and J. Hu, "Effective and efficient community search over large directed graphs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 11, pp. 2093–2107, 2018.

[110] F. D. Malliaros, C. Giatsidis, A. N. Papadopoulos, and M. Vazirgiannis, "The core decomposition of networks: Theory, algorithms and applications," *The VLDB Journal*, vol. 29, no. 1, pp. 61–92, 2020.

[111] L. Yuan, L. Qin, X. Lin, L. Chang, and W. Zhang, "Effective and efficient dynamic graph coloring," *Proceedings of the VLDB Endowment*, vol. 11, no. 3, pp. 338–351, 2017.

[112] G. D. Bader and C. W. Hogue, "An automated method for finding molecular complexes in large protein interaction networks," *BMC bioinformatics*, vol. 4, no. 1, pp. 1–27, 2003.

[113] S. Wuchty and E. Almaas, "Peeling the yeast protein network," *Proteomics*, vol. 5, no. 2, pp. 444–449, 2005.

[114] J. I. Alvarez-Hamelin, L. Dall'Asta, A. Barrat, and A. Vespignani, "K-core decomposition: A tool for the visualization of large scale networks," *arXiv preprint cs/0504107*, 2005.

[115] Y. Zhang and S. Parthasarathy, "Extracting analyzing and visualizing triangle k-core motifs within networks," in *2012 IEEE 28th international conference on data engineering*, IEEE, 2012, pp. 1049–1060.

[116] K. Shin, T. Eliassi-Rad, and C. Faloutsos, "Corescope: Graph mining using k-core analysis—patterns, anomalies and algorithms," in *2016 IEEE 16th international conference on data mining (ICDM)*, IEEE, 2016, pp. 469–478.

[117] K. Shin, T. Eliassi-Rad, and C. Faloutsos, "Patterns and anomalies in k-cores of real-world graphs with applications," *Knowledge and Information Systems*, vol. 54, no. 3, pp. 677–710, 2018.

[118] M. Kitsak, L. K. Gallos, S. Havlin, *et al.*, "Identification of influential spreaders in complex networks," *Nature physics*, vol. 6, no. 11, pp. 888–893, 2010.

[119] A. A. Chowdhary, C. Liu, L. Chen, R. Zhou, and Y. Yang, "Finding attribute diversified community over large attributed networks," *World Wide Web*, pp. 1–39, 2021.

[120] Y. Fang, X. Huang, L. Qin, *et al.*, "A survey of community search over big graphs," *The VLDB Journal*, vol. 29, no. 1, pp. 353–392, 2020.

[121] D. Wen, L. Qin, Y. Zhang, X. Lin, and J. X. Yu, "I/o efficient core graph decomposition: Application to degeneracy ordering," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 1, pp. 75–90, 2018.

[122] M. P. O'Brien and B. D. Sullivan, "Locally estimating core numbers," in *2014 IEEE International Conference on Data Mining*, IEEE, 2014, pp. 460–469.

[123] W. Khaouid, M. Barsky, V. Srinivasan, and A. Thomo, "K-core decomposition of large networks on a single pc," *Proceedings of the VLDB Endowment*, vol. 9, no. 1, pp. 13–23, 2015.

[124] Y. Zhang, J. Yu, Y. Zhang, and L. Qin, "A fast order-based approach for core maintenance," in *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, Los Alamitos, CA, USA: IEEE Computer Society, Apr. 2017, pp. 337–348.

[125] N. Wang, D. Yu, H. Jin, C. Qian, X. Xie, and Q.-S. Hua, "Parallel algorithm for core maintenance in dynamic graphs," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, IEEE, 2017, pp. 2366–2371.

[126] A. E. Sariyüce, B. Gedik, G. Jacques-Silva, K.-L. Wu, and Ü. V. Çatalyürek, "Streaming algorithms for k-core decomposition," *Proceedings of the VLDB Endowment*, vol. 6, no. 6, pp. 433–444, 2013.

[127] R.-H. Li, J. X. Yu, and R. Mao, "Efficient core maintenance in large dynamic graphs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 10, pp. 2453–2465, 2013.

[128] H. Aksu, M. Canim, Y.-C. Chang, I. Korpeoglu, and Ö. Ulusoy, "Distributed k-core view materialization and maintenance for large dynamic graphs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 10, pp. 2439–2452, 2014.

[129] K. Wang, X. Cao, X. Lin, W. Zhang, and L. Qin, "Efficient computing of radius-bounded k-cores," in *2018 IEEE 34th international conference on data engineering (ICDE)*, IEEE, 2018, pp. 233–244.

[130] C. Pabst, R. Higgins, J. R. Goicoechea, *et al.*, "Disruption of the orion molecular core 1 by wind from the massive star θ 1 orionis c," *Nature*, vol. 565, no. 7741, pp. 618–621, 2019.

[131] C. Zhang, F. Zhang, W. Zhang, *et al.*, "Exploring finer granularity within the cores: Efficient (k, p)-core computation," in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, IEEE, 2020, pp. 181–192.

[132] H. Wu, J. Cheng, Y. Lu, *et al.*, "Core decomposition in large temporal graphs," in *2015 IEEE International Conference on Big Data (Big Data)*, IEEE, 2015, pp. 649–658.

[133] F. Zhao and A. K. Tung, "Large scale cohesive subgraphs discovery for social network visual analysis," *Proceedings of the VLDB Endowment*, vol. 6, no. 2, pp. 85–96, 2012.

[134] Y. Shao, L. Chen, and B. Cui, "Efficient cohesive subgraphs detection in parallel," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, 2014, pp. 613–624.

[135] J. Cohen, "Graph twiddling in a mapreduce world," *Computing in Science & Engineering*, vol. 11, no. 4, pp. 29–41, 2009.

[136] L. Quick, P. Wilkinson, and D. Hardcastle, "Using pregel-like large scale graph processing frameworks for social network analysis," in *2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, IEEE, 2012, pp. 457–463.

[137] Y. Zhang and J. X. Yu, "Unboundedness and efficiency of truss maintenance in evolving graphs," in *Proceedings of the 2019 International Conference on Management of Data*, 2019, pp. 1024–1041.

[138] V. R. Jakkula and G. Karypis, "Streaming and batch algorithms for truss decomposition," in *2019 First International Conference on Graph Computing (GC)*, IEEE, 2019, pp. 51–59.

[139] S. Ebadian and X. Huang, "Fast algorithm for k-truss discovery on public-private graphs," *arXiv preprint arXiv:1906.00140*, 2019.

[140] X. Huang and L. V. Lakshmanan, "Attribute-driven community search," *Proceedings of the VLDB Endowment*, vol. 10, no. 9, pp. 949–960, 2017.

[141] X. Huang, L. V. Lakshmanan, J. X. Yu, and H. Cheng, "Approximate closest community search in networks," *arXiv preprint arXiv:1505.05956*, 2015.

[142] Y. Wu, R. Sun, C. Chen, X. Wang, and Q. Zhu, "Maximum signed (k, r)-truss identification in signed networks," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020, pp. 3337–3340.

[143] L. Chang, J. X. Yu, and L. Qin, "Fast maximal cliques enumeration in sparse graphs," *Algorithmica*, vol. 66, no. 1, pp. 173–186, 2013.

[144] J. Pattillo, N. Youssef, and S. Butenko, "On clique relaxation models in network analysis," *European Journal of Operational Research*, vol. 226, no. 1, pp. 9–18, 2013.

[145] R. D. Alba, "A graph-theoretic definition of a sociometric clique," *Journal of Mathematical Sociology*, vol. 3, no. 1, pp. 113–126, 1973.

[146] D. Eppstein and D. Strash, "Listing all maximal cliques in large sparse real-world graphs," in *International Symposium on Experimental Algorithms*, Springer, 2011, pp. 364–375.

[147] D. Eppstein, M. Löffler, and D. Strash, "Listing all maximal cliques in sparse graphs in near-optimal time," in *International Symposium on Algorithms and Computation*, Springer, 2010, pp. 403–414.

[148] J. Wang, J. Cheng, and A. W.-C. Fu, "Redundancy-aware maximal cliques," in *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '13, Chicago, Illinois, USA: ACM, 2013, pp. 122–130, ISBN: 978-1-4503-2174-7.

[149] A. Das, S.-V. Sanei-Mehri, and S. Tirthapura, "Shared-memory parallel maximal clique enumeration from static and dynamic graphs," *ACM Transactions on Parallel Computing (TOPC)*, vol. 7, no. 1, pp. 1–28, 2020.

[150] Z. Wang, Q. Chen, B. Hou, *et al.*, "Parallelizing maximal clique and k-plex enumeration over graph data," *Journal of Parallel and Distributed Computing*, vol. 106, pp. 79–91, 2017.

[151] Y. Xu, J. Cheng, and A. W.-C. Fu, "Distributed maximal clique computation and management," *IEEE Transactions on Services Computing*, vol. 9, no. 1, pp. 110–122, 2015.

[152] J. Cheng, L. Zhu, Y. Ke, and S. Chu, "Fast algorithms for maximal clique enumeration with limited memory," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2012, pp. 1240–1248.

[153] J. Cheng, Y. Ke, A. W.-C. Fu, J. X. Yu, and L. Zhu, "Finding maximal cliques in massive networks by h*-graph," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, 2010, pp. 447–458.

[154] D. Duan, Y. Li, R. Li, and Z. Lu, "Incremental k-clique clustering in dynamic social networks," *Artificial Intelligence Review*, vol. 38, no. 2, pp. 129–147, 2012.

[155] A. Das, M. Svendsen, and S. Tirthapura, "Incremental maintenance of maximal cliques in a dynamic graph," *VLDB J.*, vol. 28, no. 3, pp. 351–375, 2019.

[156] T. J. Ottosen and J. Vomlel, "Honour thy neighbour: Clique maintenance in dynamic graphs," *on Probabilistic Graphical Models*, p. 201, 2010.

[157] V. Stix, "Finding all maximal cliques in dynamic graphs," *Computational Optimization and applications*, vol. 27, no. 2, pp. 173–186, 2004.

[158] S. Sun, Y. Wang, W. Liao, and W. Wang, "Mining maximal cliques on dynamic graphs efficiently by local strategies," in *33rd IEEE International Conference on Data Engineering, ICDE 2017*, IEEE Computer Society, 2017, pp. 115–118.

[159] G. Palla, I. Derényi, I. Farkas, and T. Vicsek, "Uncovering the overlapping community structure of complex networks in nature and society," *nature*, vol. 435, no. 7043, pp. 814–818, 2005.

[160] J. Abello, M. G. C. Resende, and S. Sudarsky, "Massive quasi-clique detection," in *LATIN 2002: Theoretical Informatics, 5th Latin American Symposium, Cancun, Mexico, April 3-6, 2002, Proceedings*, 2002, pp. 598–612.

[161] W. Cui, Y. Xiao, H. Wang, Y. Lu, and W. Wang, "Online search of overlapping communities," in *Proceedings of the 2013 ACM SIGMOD international conference on Management of data*, 2013, pp. 277–288.

[162] R.-H. Li, Q. Dai, L. Qin, *et al.*, "Efficient signed clique search in signed networks," in *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, IEEE, 2018, pp. 245–256.

[163] R. Li, Q. Dai, L. Qin, *et al.*, "Signed clique search in signed networks: Concepts and algorithms," *IEEE Transactions on Knowledge and Data Engineering*, 2019.

[164] Y. He, K. Wang, W. Zhang, X. Lin, and Y. Zhang, "Exploring cohesive subgraphs with vertex engagement and tie strength in bipartite graphs," *Inf. Sci.*, vol. 572, pp. 277–296, 2021.

[165] K. Wang, W. Zhang, Y. Zhang, L. Qin, and Y. Zhang, "Discovering significant communities on bipartite graphs: An index-based approach," *IEEE Trans. Knowl. Data Eng.*, 2021.

[166] B. Liu, L. Yuan, X. Lin, L. Qin, W. Zhang, and J. Zhou, "Efficient (α, β)-core computation in bipartite graphs," *The VLDB Journal*, pp. 1–25, 2020.

[167] W. Bai, Y. Chen, D. Wu, Z. Huang, Y. Zhou, and C. Xu, "Generalized core maintenance of dynamic bipartite graphs," *Data Mining and Knowledge Discovery*, pp. 1–31, 2021.

[168] Y. Zhang, K. Wang, W. Zhang, X. Lin, and Y. Zhang, "Pareto-optimal community search on large bipartite graphs," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2021, pp. 2647–2656.

[169] X. Liu and X. Wang, "Cohesive subgraph identification in weighted bipartite graphs," *Applied Sciences*, vol. 11, no. 19, p. 9051, 2021.

[170] Z. Zou, "Bitruss decomposition of bipartite graphs," in *International Conference on Database Systems for Advanced Applications*, Springer, 2016, pp. 218–233.

[171] L. Chen, C. Liu, R. Zhou, J. Xu, and J. Li, "Efficient exact algorithms for maximum balanced biclique search in bipartite graphs," in *Proceedings of the 2021 International Conference on Management of Data*, 2021, pp. 248–260.

[172] B. Lyu, L. Qin, X. Lin, Y. Zhang, Z. Qian, and J. Zhou, "Maximum biclique search at billion scale," *Proceedings of the VLDB Endowment*, 2020.

[173] Z. Tuza, "Covering of graphs by complete bipartite subgraphs; complexity of 0–1 matrices," *Combinatorica*, vol. 4, no. 1, pp. 111–116, 1984.

[174] J. Orlin, *Contentment in graph theory: Covering graphs with cliques*, 1977.

[175] H. Müller, "On edge perfectness and classes of bipartite graphs," *Discrete Mathematics*, vol. 149, no. 1-3, pp. 159–187, 1996.

[176] E. Prisner, "Bicliques in graphs ii: Recognizing k-path graphs and underlying graphs of line digraphs," in *International Workshop on Graph-Theoretic Concepts in Computer Science*, Springer, 1997, pp. 273–287.

[177] E. Prisner, "Bicliques in graphs i: Bounds on their number," *Combinatorica*, vol. 20, no. 1, pp. 109–117, 2000.

[178] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979, ISBN: 0-7167-1044-7.

[179] H.-J. Bandelt, M. Farber, and P. Hell, "Absolute reflexive retracts and absolute bipartite retracts," *Discrete Applied Mathematics*, vol. 44, no. 1-3, pp. 9–20, 1993.

[180] M. C. Golumbic and C. F. Goss, "Perfect elimination and chordal bipartite graphs," *Journal of Graph Theory*, vol. 2, no. 2, pp. 155–163, 1978.

[181] G. Alexe, S. Alexe, Y. Crama, S. Foldes, P. L. Hammer, and B. Simeone, "Consensus algorithms for the generation of all maximal bicliques," *Discrete Applied Mathematics*, vol. 145, no. 1, pp. 11–21, 2004.

[182] A. Das and S. Tirthapura, "Incremental maintenance of maximal bicliques in a dynamic bipartite graph," *IEEE Trans. Multi-Scale Computing Systems*, vol. 4, no. 3, pp. 231–242, 2018.

[183] A. C. Driskell, M. J. Sanderson, O. Eulenstein, R. H. Ree, and S. Langley, "Obtaining Maximal Concatenated Phylogenetic Data Sets from Large Sequence Databases," *Molecular Biology and Evolution*, vol. 20, no. 7, pp. 1036–1042, Jul. 2003, ISSN: 0737-4038.

[184] R. A. Mushlin, A. Kershenbaum, S. T. Gallagher, and T. R. Rebbeck, "A graph-theoretical approach for pattern discovery in epidemiological research," *IBM systems journal*, vol. 46, no. 1, pp. 135–149, 2007.

[185] K. Makino and T. Uno, "New algorithms for enumerating all maximal cliques," in *Scandinavian Workshop on Algorithm Theory*, Springer, 2004, pp. 260–272.

[186] P. Erdős and A. Rényi, "On the existence of a factor of degree one of a connected random graph," *Acta Mathematica Hungarica*, vol. 17, no. 3-4, pp. 359–368, 1966.

[187] M. J. Zaki and M. Ogihara, "Theoretical foundations of association rules," in *3rd ACM SIGMOD workshop on research issues in data mining and knowledge discovery*, 1998, pp. 71–78.

[188] J. Li, H. Li, D. Soh, and L. Wong, "A correspondence between maximal complete bipartite subgraphs and closed patterns," in *European Conference on Principles of Data Mining and Knowledge Discovery*, Springer, 2005, pp. 146–156.

[189] M. J. Zaki and C.-J. Hsiao, "Charm: An efficient algorithm for closed itemset mining," in *Proceedings of the 2002 SIAM international conference on data mining*, SIAM, 2002, pp. 457–473.

[190] J. Wang, J. Han, and J. Pei, "Closet+: Searching for the best strategies for mining frequent closed itemsets," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2003, pp. 236–245.

[191] G. Grahne and J. Zhu, "Efficiently using prefix-trees in mining frequent itemsets.," in *FIMI*, vol. 90, 2003.

[192] T. Uno, M. Kiyomi, and H. Arimura, "Lcm ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets," in *Fimi*, vol. 126, 2004.

[193] D. Nussbaum, S. Pu, J.-R. Sack, T. Uno, and H. Zarrabi-Zadeh, "Finding maximum edge bicliques in convex bipartite graphs," *Algorithmica*, vol. 64, no. 2, pp. 311–325, 2012.

[194] T. Kloks and D. Kratsch, "Computing a perfect edge without vertex elimination ordering of a chordal bipartite graph," *Information Processing Letters*, vol. 55, no. 1, pp. 11–16, 1995.

[195] A. Das and S. Tirthapura, "Shared-memory parallel maximal biclique enumeration," in *26th IEEE International Conference on High Performance Computing, Data, and Analytics, HiPC 2019, Hyderabad, India, December 17-20, 2019*, IEEE, 2019, pp. 34–43.

[196] G. Liu, K. Sim, and J. Li, "Efficient mining of large maximal bicliques," in *International Conference on Data Warehousing and Knowledge Discovery*, Springer, 2006, pp. 437–448.

[197] L. Chen, C. Liu, R. Zhou, J. Xu, J. X. Yu, and J. Li, "Finding effective geo-social group for impromptu activities with diverse demands," in *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, ACM, 2020, pp. 698–708.

[198] C. Liu, L. Chen, R. Zhou, and A. A. Chowdhary, "Attribute diversified community search," in *Software Foundations for Data Interoperability and Large Scale Graph Data Analytics*, Springer, 2020, pp. 3–17.

[199] D. Wu, D. Luo, C. S. Jensen, and J. Z. Huang, "Efficiently mining maximal diverse frequent itemsets," in *Database Systems for Advanced Applications - 24th International Conference, DASFAA 2019, Chiang Mai,*

Thailand, April 22-25, 2019, Proceedings, Part II, ser. Lecture Notes in Computer Science, vol. 11447, Springer, 2019, pp. 191–207.

[200] A. Verma, S. Dawar, R. Kumar, S. B. Navathe, and V. Goyal, "High-utility and diverse itemset mining," *Appl. Intell.*, vol. 51, no. 7, pp. 4649–4663, 2021.

[201] F. Ahmed, J. P. Dickerson, and M. Fuge, "Diverse weighted bipartite b-matching," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, International Joint Conferences on Artificial Intelligence Organization, 2017, pp. 35–41.

[202] F. V. Fomin, P. A. Golovach, L. Jaffke, G. Philip, and D. Sagunov, "Diverse pairs of matchings," LIPIcs, vol. 181, 26:1–26:12, 2020.

[203] F. V. Fomin, P. A. Golovach, F. Panolan, G. Philip, and S. Saurabh, "Diverse collections in matroids and graphs," LIPIcs, vol. 187, 31:1–31:14, 2021.

[204] P. Chalermsook, S. Heydrich, E. Holm, and A. Karrenbauer, "Nearly tight approximability results for minimum biclique cover and partition," in *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, ser. Lecture Notes in Computer Science, vol. 8737, Springer, 2014, pp. 235–246.

[205] H. U. Simon, "On approximate solutions for combinatorial optimization problems," *SIAM J. Discret. Math.*, vol. 3, no. 2, pp. 294–310, 1990.

[206] H. Gruber and M. Holzer, "Inapproximability of nondeterministic state and transition complexity assuming p≠np," in *Developments in Language Theory, 11th International Conference, DLT 2007, Turku, Finland, July 3-6, 2007, Proceedings*, ser. Lecture Notes in Computer Science, vol. 4588, Springer, 2007, pp. 205–216.

[207] S. Jukna and A. S. Kulikov, "On covering graphs by complete bipartite subgraphs," *Discret. Math.*, vol. 309, no. 10, pp. 3399–3403, 2009.

[208] J. Orlin, in *Indagationes Mathematicae (Proceedings)*, Elsevier, vol. 80, 1977, pp. 406–424.

[209] H. Fleischner, E. Mujuni, D. Paulusma, and S. Szeider, "Covering graphs with few complete bipartite subgraphs," *Theor. Comput. Sci.*, vol. 410, no. 21-23, pp. 2045–2053, 2009.

[210] J. Chen and I. A. Kanj, "Constrained minimum vertex cover in bipartite graphs: Complexity and parameterized algorithms," *J. Comput. Syst. Sci.*, vol. 67, no. 4, pp. 833–847, 2003.

[211] A. Epasto and E. Upfal, "Efficient approximation for restricted biclique cover problems," *Algorithms*, vol. 11, no. 6, p. 84, 2018.

[212] Y. Fang, Y. Yang, W. Zhang, X. Lin, and X. Cao, "Effective and efficient community search over large heterogeneous information networks," *Proceedings of the VLDB Endowment*, vol. 13, no. 6, pp. 854–867, 2020.

[213] Y. Yang, Y. Fang, X. Lin, and W. Zhang, "Effective and efficient truss computation over large heterogeneous information networks," ICDE, 2020.

[214] R. Agrawal, A. Borgida, and H. V. Jagadish, "Efficient management of transitive relationships in large data and knowledge bases," in *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data, Portland, Oregon, USA, May 31 - June 2, 1989.*, 1989, pp. 253–262.

[215] J. Kunegis, "Konect: The koblenz network collection," in *Proceedings of the 22nd International Conference on World Wide Web*, ACM, 2013, pp. 1343–1350.

[216] V. Rakesh, J. Choo, and C. K. Reddy, "Project recommendation using heterogeneous traits in crowdfunding," in *ICWSM 2015*, vol. 9.

[217] A. V. Aho, M. R. Garey, and J. D. Ullman, "The transitive reduction of a directed graph," *SIAM Journal on Computing*, vol. 1, no. 2, pp. 131–137, 1972.

[218] M. Gopinath, M. Glassman, and P. Nyer, "How culture of targeting impacts the evaluation of products with multilingual packaging," *Psychology & Marketing 2013*, vol. 30, no. 6, pp. 490–500,

[219] M. Dawande, P. Keskinocak, J. M. Swaminathan, and S. Tayur, "On bipartite and multipartite clique problems," *Journal of Algorithms*, vol. 41, no. 2, pp. 388–403, 2001.

[220] G. Ausiello, N. Boria, A. Giannakos, G. Lucarelli, and V. T. Paschos, "Online maximum k-coverage," *Discrete Applied Mathematics*, vol. 160, no. 13-14, pp. 1901–1913, 2012.

[221] F. Zhang, Y. Zhang, L. Qin, W. Zhang, and X. Lin, "When engagement meets similarity: Efficient (k, r)-core computation on social networks," *arXiv preprint arXiv:1611.03254*, 2016.

[222] M. Kargar and A. An, "Keyword search in graphs: Finding r-cliques," *Proceedings of the VLDB Endowment*, vol. 4, no. 10, pp. 681–692, 2011.

[223] L. Yuan, L. Qin, X. Lin, L. Chang, and W. Zhang, "I/o efficient ecc graph decomposition via graph reduction," *The VLDB Journal*, vol. 26, no. 2, pp. 275–300, 2017.