Combinatorial Generation
Algorithms for Subgraphs

By
Zareen Alamgir

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
AT
NATIONAL UNIVERSITY OF COMPUTER
AND EMERGING SCIENCES
FAISAL TOWN, LAHORE
APRIL 2013

Supervised By: Dr. Syed Husnine

© Copyright by Zareen Alamgir, 2013
# Table of Contents

Table of Contents .............................................................. ii

List of Tables ........................................................................... vi

List of Figures ........................................................................... vii

Abstract ...................................................................................... ix

Publications ................................................................................ xi

Acknowledgements ..................................................................... xii

1 Introduction .............................................................................. 1

1.1 Pseudo-Biclique Generation ............................................... 2

1.2 c-Isolated Biclique Generation .............................................. 3

1.3 Revolving Door Path Generation .......................................... 3

1.4 Thesis Organization ............................................................ 4

2 Literature Review ..................................................................... 5

2.1 Combinatorial Algorithms .................................................. 5

2.2 Ordering in Combinatorial Algorithms ................................. 6

2.2.1 Lexicographic Order ..................................................... 6
2.2.2 Minimal Change Order ........................................... 7
2.3 Analysis of Generation Algorithms ............................. 8
2.4 Contemporary Results ............................................. 9
  2.4.1 Cliques ....................................................... 9
  2.4.2 Bipartite Cliques - Bicliques .............................. 10
  2.4.3 Pseudo-Cliques and Pseudo-Bicliques ..................... 11
  2.4.4 Isolated Cliques and Isolated Bicliques .................. 12
  2.4.5 Spanning Trees .............................................. 14
  2.4.6 Paths in Cliques ........................................... 14
  2.4.7 Conclusion .................................................. 16

3 Generation Algorithms for Pseudo-Bicliques .................. 17
  3.1 Introduction .................................................. 17
  3.2 Preliminaries ................................................ 18
    3.2.1 Weighted Graph ......................................... 19
  3.3 Generation of Weighted Pseudo-Bicliques .................... 19
    3.3.1 Generation Algorithm .................................. 22
    3.3.2 Maximal Pseudo-Bicliques .............................. 24
    3.3.3 Improvements for Efficient Computation ................. 24
  3.4 Multilevel Pseudo-Bicliques ................................ 26
    3.4.1 Generation Algorithm for Multilevel Pseudo-Bicliques .. 27
  3.5 Applications: Co-clustering and Social Networks .......... 28
  3.6 Computational Experiments .................................. 30
    3.6.1 Random Graphs ......................................... 31
    3.6.2 Stock Market: Stocks and Financial Ratios ............ 32
    3.6.3 Social Network for Movie Ratings by Viewers ........... 37
3.7 Conclusion ................................................................. 41

4 Generation Algorithm for c-Isolated Bicliques .......... 42
  4.1 Introduction ......................................................... 42
  4.2 Preliminaries ....................................................... 43
  4.3 Generation of c-Isolated Bicliques ......................... 44
    4.3.1 Trimming Stage .............................................. 46
    4.3.2 Generation Stage .......................................... 48
    4.3.3 Screening Stage .......................................... 49
    4.3.4 Time Complexity .......................................... 54
  4.4 Computational Experiments ................................. 64
    4.4.1 Scale Free Random Graphs .............................. 65
    4.4.2 Citation Network ........................................ 66
    4.4.3 Movie Network ........................................... 69
  4.5 Conclusion ....................................................... 72

5 Generation Algorithms for Combinatorial Path ........ 74
  5.1 Introduction ...................................................... 74
  5.2 Preliminaries .................................................... 75
  5.3 Revolving Door Algorithms for Spanning Paths ........ 77
    5.3.1 Naive Idea: List Spanning Path Algorithm .......... 77
    5.3.2 Efficient Block Spanning Path Algorithm ........... 83
  5.4 Revolving Door Algorithm for Path Set ................ 85
    5.4.1 Naive Scheme ............................................ 85
    5.4.2 Algorithm ................................................ 87
  5.5 Note on Generating All Paths .............................. 89
  5.6 Revolving Door Algorithm for Generating All Paths .... 91
List of Tables

2.1 Johnson and Trotter permutations for $n = 4$ ........................................... 15
3.1 Experimental results for different edge densities ................................. 31
3.2 Experimental results for different density thresholds ......................... 32
3.3 Experimental results for different number of nodes ......................... 32
3.4 Experimental results for movie rating network ................................. 39
4.1 Experimental results for different values of $c$ in a scale-free graph ...... 66
4.2 Example of a 5-isolated biclique in a citation network .................... 70
4.3 Experimental results for different values of $c$ in a citation network ...... 70
4.4 Experimental results for different values of $c$ in a movie network ...... 71
5.1 All path sets for $K_n$ ........................................................................ 88
5.2 All $K_3$ paths ............................................................................. 95
5.3 All $K_4$ paths ............................................................................. 100
5.4 All $K_5$ paths ............................................................................. 102
List of Figures

2.1 All spanning trees of an undirected graph ....................... 14
3.1 An illustrative example of a weighted pseudo-biclique .......... 22
3.2 A multilevel pseudo-biclique .................................. 27
3.3 Results of the experiments on random graphs .................. 33
3.4 Results of the experiments on medium-sized firms .............. 36
3.5 Results of the experiments on large-sized firms ................ 38
3.6 Results of the experiment on movie rating network ............. 40
4.1 c-Isolated biclique ............................................. 44
4.2 A bipartite graph $G$ and its bipartite complement graph $\bar{G}$ . . 49
4.3 The time required to generate all c-isolated bicliques in a scale-free graph 67
4.4 Distribution of c-isolated bicliques in a scale-free graph ........ 67
4.5 Conversion of a directed graph to a bipartite graph $G(V_1 \cup V_2, E)$ . 68
4.6 The time required to generate all c-isolated bicliques in a citation network 71
4.7 Distribution of c-isolated bicliques in a citation network .......... 72
4.8 Distribution of c-isolated bicliques in a movie network ........... 73
4.9 The time required to generate all c-isolated bicliques in a movie network 73
5.1 A complete graph on 5 vertices .................................. 75
5.2 A revolving door ................................................. 76
5.3 Generate $L_4$ from $L_3$ ........................................ 81
5.4 Block structure for $n = 4$ ....................................... 83
5.5 Case 1 for generating all paths ............................. 95
5.6 Case 3 for generating all paths ............................. 96
5.7 All $K_3$ paths ................................................. 99
Abstract

The field of computer science has greatly benefitted from the advances in combinatorial algorithms in the last few decades. This is because the advent of high speed computers has made it possible to generate lists of combinatorial objects in a practical amount of time. In areas such as genome science and data mining the problems are often vaguely defined, and researchers have to look for meaningful information in huge datasets.

In this thesis, efficient generation algorithms for subgraphs such as bicliques and paths in cliques are developed. Cliques and bicliques are used to model various real-world problems encountered in bio-informatics, data mining and networks. We consider two variations of bicliques: pseudo-bicliques and c-isolated bicliques. Pseudo-bicliques relax the rigid connectivity requirement of bicliques to cater for missing and noisy data. On the other hand, the c-isolated bicliques enforce a restriction on the external connectivity of the vertices in a biclique to model cohesive communities. This thesis presents an algorithm based on reverse search to list all pseudo-bicliques in a graph $G$. The algorithm takes linear time on average to generate each pseudo-biclique. On the other hand, our generation algorithm for c-isolated bicliques exploit underlying properties of an isolated biclique to trim the input graph. Furthermore, the algorithm deploys the vertex cover enumeration algorithm based on fixed point tractability (FTP) and lists all isolated bicliques in linear time, in the case where $c$ is constant. The performance of the proposed algorithms is evaluated on random graphs.
and real-world problems. The results are quite promising and confirm our theoretical findings.

In this research work, we also explore another combinatorial object called a clique. A constant amortized time algorithm is proposed to generate all spanning paths and all paths in a clique in minimal change order (an ordering in which successive elements differ in a small way).
Publications


Other Publications


Acknowledgements

I would like to thank Syed Husnine, my advisor, for his unlimited cooperation, help and encouragement. Without his valuable suggestions and constant support this research would not have been possible. I would also like to thank my co-advisor, Dr Fakhar Lodhi. He has a unique way of inspiring students through clarity of concept and fervor. He gave me confidence and encouragement to pursue my PhD program.

I am grateful to Dr. Arshad Hussain and Shafiq-ur-Rehman, my great teachers and mentors. I have learnt a lot from them during my stay at the National University. I am proud to be one of their students.

My friends at the National University played a big role in making my studies more enjoyable. I am thankful to Saira for sharing my thoughts and giving me useful suggestions and Sobia for all the good times and the tea parties we had together.

I owe much gratitude to my family for their support, and most of all for their patience with which they endure me during the years of PhD research. I would like to dedicate this thesis to my husband and parents, who provided a lot of encouragement, love and support.
Chapter 1

Introduction

Generation of combinatorial objects is of fundamental interest in computer science. The solution to various problems encountered in computing, networks, circuit analysis, statistics, chemistry and operations research begins by listing all the possibilities that can occur. Most computer scientists have come across an instance in which they unravel the mystery of the given problem by generating all the choices and searching for a desired property. In the last few decades, a tremendous amount of research has been carried out in the area of combinatorial generation. In fact, there is an explosion of interesting results. This renewed interest is mainly due to the increasing speed of computers and the emergence of parallel computing.

With high-speed computers, the construction of long lists of combinatorial objects has become a non-troublesome task (Ruskey 2003). Now, it is possible to solve practical problems that require sampling of a combinatorial object at random or an exhaustive search through all objects in a class (Savage 1997). For instance, various problems on electrical and cellular networks can be framed as optimization problems of the spanning trees. Some of these problems have complicated objective functions, and it is impossible to solve them efficiently. However, it is much easier to solve these problems by generating all the spanning trees. Apart from this, in the fields of data mining, social networks and genome science, enumeration has proven to be a very
successful tool. In these areas, problems are not precisely described and, therefore, researchers have to look for meaningful structures in huge data sets. The generation schemes of graph objects such as paths, trees, and cliques are deployed in pattern mining while generation of bicliques is extensively used in mining of frequent itemsets (Prisner 1997; Matsunaga1 et al. 2009; Uno and Arimura 2008).

Combinatorial generation is an interesting area that has obvious connections with various practical domains, but like other sciences, it also has some purely theoretical and philosophical aspects. A glimpse on the listing of combinatorial objects can stimulate the human imagination and help us to gain an important perspective on the problem under consideration.

This thesis studies methodologies for listing the combinatorial objects: pseudo-bicliques, c-isolated bicliques and paths in cliques. The generated lists can provide an insight into many real-life problems. Next, we briefly describe each of these objects and highlight the need for their generation algorithms.

1.1 Pseudo-Biclique Generation

Bicliques are used to model various real-world problems such as co-clustering and protein interaction discovery (Li et al. 2007; Ding et al. 2006). However, the rigid connectivity requirements of bicliques make it unsuitable for dealing with incorrect or missing data. Researchers have found that pseudo-biclique is a better model to represent real-life situations as it can deal with missing information (Li et al. 2008). Different models of pseudo-biclique were proposed in the last decade or so. In this thesis, we introduce a density based model of pseudo-bicliques. This model defines a pseudo-biclique as a bipartite subgraph in which the ratio of the number of its edges to the number of edges in a biclique of the same size is not less than a given threshold value.

The enumeration of density based pseudo-bicliques is a non-trivial task, as simple
and straightforward enumeration techniques like back-tracking and branch-and-bound involve a NP-complete problem (Uno 2010). In addition to this, the density based pseudo-bicliques are anti-monotone, that is, every subset of a pseudo-biclique is not a pseudo-biclique. This thesis extends the definition of pseudo-bicliques to weighted and multilevel pseudo-bicliques and develops an algorithm to generate all pseudo-bicliques. The performance of this algorithm is examined on random graphs and real-world problems: stock-financial ratios and social network for movie ratings. The results are promising and shows that average linear time is incurred to generate each pseudo-biclique.

1.2 c-Isolated Biclique Generation

Bicliques are often used to model web communities; a community on the web is defined as a set of sites that has more links to members of the community than to non-members. However, bicliques do not provide any control over external links of the nodes. Thus, communities represented by bicliques may or may not be cohesive and strong. To deal with this, we introduce an additional requirement of isolation on bicliques. The concept of c-isolation is a new one and has recently been introduced to cliques (Ito and Iwama 2009; Huffner et al. 2009).

An efficient algorithm for generating isolated bicliques in a given graph is highly desirable. In this work, we tackle this problem and propose an algorithm that takes linear time. We carry out computational experiments on scale free random graphs, and real world data sets: citation networks, and movie networks.

1.3 Revolving Door Path Generation

Finding all paths in a graph is a vital aspect in solving scheduling problems, computing accessibility and traffic flows. In some cases, we can map paths and circuits to
some specific class of permutation. This is helpful, since the generation of all permutations is one of the most studied areas in combinatorial generation. Recently, Knuth has compiled comprehensive material on permutation generation in his new volume on combinatorial generation (Knuth 2005).

We can significantly speed up combinatorial generation by listing objects in minimal change order; an order in which successive elements differ slightly. Furthermore, algorithms based on minimal change ordering give new insights into the structure of combinatorial objects as they usually involve elegant recursive constructions. The minimal change order in which two consecutive objects differ in exactly two positions is called revolving door order (Nijenhuis and Wilf 1978; Knuth 2005). Recently, much work has been done to list various combinatorial objects in minimal change and revolving door order: spanning trees (Knuth 2005), rooted and free trees (Korsh and Lafollette 2006), and lucas strings (Baril and Vajnovszki 2006).

In this thesis, we extend the work on minimal change order and propose an algorithm for generating spanning paths, all path sets and all paths in a clique $K_n$, where $n$ is the number of vertices.

1.4 Thesis Organization

The rest of the document is organized as follows:

Chapter 2 provides a brief outline of the background literature on the subject. We present generation algorithms for pseudo-bicliques in chapter 3. The details of our c-isolated bicliques generation algorithm is given in chapter 4. Chapter 5 gives revolving door algorithms for listing all spanning paths and all paths in a clique. We present conclusions and future directions in chapter 6.
Chapter 2

Literature Review

Combinatorial generation has a long and distinguished history. In fact, the exhaustive listing of combinatorial objects was one of the first nontrivial problems to be tackled by computer (Knuth 2005). This chapter briefly reviews the relevant literature and current state of knowledge pertaining to the proposed research. It also describes the basic terminologies that are helpful in analyzing and understanding the combinatorial algorithms.

2.1 Combinatorial Algorithms

Combinatorial algorithms investigate combinatorial objects such as permutations, combinations, set partitions, numerical partitions, binary trees and graphs. They find solutions to the fundamental questions regarding existence, enumeration and structure of the combinatorial object. Informally, combinatorial algorithms are classified into three categories: search, generation, and enumeration (Kreher and Stinson 1999). The search algorithm, as the name specifies searches for a specific object in a combinatorial class. The generation algorithm produces a list of all the combinatorial structures of a particular type in a certain order, such as lexicographic or minimal change order. However, in enumeration, we compute the number of different objects
of a certain type. Every generation algorithm is essentially an enumeration algorithm, since each object is counted as it is generated.

2.2 Ordering in Combinatorial Algorithms

Combinatorial generation algorithms produce the list of objects in some specific order. There are two types of orderings that are mostly used: lexicographic order, and minimal change order.

2.2.1 Lexicographic Order

The lexicographic order is a natural way of listing the combinatorial object. Formally, it can be defined as an increasing numeric order; we say \( a_1, a_2, \ldots, a_n < b_1, b_2, \ldots, b_m \), if

1. for some \( k \), \( a_k < b_k \) and \( a_i = b_i \) for \( i = 1, 2, \ldots, k - 1 \), or
2. \( n < m \) and \( a_i = b_i \) for \( i = 1, 2, \ldots, n \).

where \( < \) denote the ordering of the symbols of the alphabets.

For permutations, lexicographic order is simply the alphabetic order for lists of symbols. For example, the permutations of \( \{a, b, c\} \) in lexicographic order are \( abc, acb, bac, bca, cab \), and \( cba \). In the lexicographic ordering of subsets, two subsets are ordered by their smallest elements. For example, the subsets of \( \{a, b, c\} \) in lexicographic order are \( \{\}, \{a\}, \{a, b\}, \{a, b, c\}, \{a, c\}, \{b\}, \{b, c\}, \{c\} \).

The lexicographic order is sometimes called dictionary order. It is easy to comprehend, and proves quite handy in different situations. Many fastest known combinatorial algorithms list the objects in lexicographical order; they exploit the underlying lexicographic structure of the object to achieve efficiency.
2.2.2 Minimal Change Order

The minimal change order, is an ordering in which consecutive objects differ slightly in some pre-specified pattern. Lists of combinatorial objects in minimal change order are often called gray codes. Savage has conducted a comprehensive survey on the combinatorial gray codes (Savage 1997). A classical example of gray codes is binary reflected codes. In this method, we generate all binary numbers of particular bit-length such that consecutive numbers differ by only one bit. The list $L(i)$ of binary reflected codes, where $i$ is 1, 2, 3 bit numbers is as follows:

\[
L(1) = 0
eq 1
\]

\[
L(2) = 00
= 01
= 11
= 10
\]

\[
L(3) = 000
= 001
= 011
= 010
= 110
= 111
= 101
= 100
\]
Gray codes have various applications and are deployed in areas such as image processing (Amalraj et al. 1990), data compression (Richards 1986), circuit testing (Robinson and Cohn 1981), and combinatorial games and puzzles (Gardner 1971).

Revolving Door Order

This is a minimal change ordering in which two consecutive objects have distance two. That is, they differ in exactly two positions (Nijenhuis and Wilf 1978). The terminology is based on the revolving door that connects two rooms and is used to swap two persons between the rooms.

2.3 Analysis of Generation Algorithms

The lists produced by a generation algorithm can be huge, even for a combinatorial class of moderate size. Therefore, it is highly desirable that such algorithms are extremely efficient. Usually, there are an exponential number of objects in a class. Thus, we compute the time complexity of generation algorithms in terms of the time required to generate each output. In these algorithms, we are only concerned with the number of data structure changes that occur as the objects are being generated. In some cases, we even ignore the time to output each individual element. This is called “Dont count the output principle” (Ruskey 2003).

The complexity of generation algorithms is usually classified as: loop-less, constant amortized time (CAT) or polynomial. An algorithm is called loop-less if the worst case time delay between generating consecutive objects is constant. The term CAT has been coined by Frank Ruskey to stand for constant amortized time per element (Ruskey 2003). In a CAT algorithm, the amount of computation, after a short preprocessing phase, is proportional to the number of objects that are listed. Finally, a generation algorithm is considered polynomial-delay, if the time needed to generate the successive objects is polynomial.
In some cases, the number of objects in a combinatorial class is not exponential. For example, if we introduce the restriction of isolation on bicliques, then the number of bicliques is significantly reduce. In these cases, we bound the overall running time of the algorithm instead of computing the time required to output each element.

2.4 Contemporary Results

In this section, we discuss recent results on the generation of subgraphs such as trees, paths, cliques, bicliques, pseudo-cliques, pseudo-bicliques and isolated cliques.

2.4.1 Cliques

Given a simple graph $G(V, E)$. A subset $S \subseteq V$ is a clique if $\forall u, v \in S : (u, v) \in E$. Cliques have various applications in diverse fields like biology, chemistry, networks and data mining (Zadeh et al. 2004; Cazals and Karande 2008; Matsunaga1 et al. 2009). Consequently, the generation of all cliques and maximal cliques has gained a lot of attention in the recent past.

A number of algorithms have been developed to list all cliques and maximal cliques in a given graph (Bron and Kerbosch 1973; Tomita et al. 2006; Makino and Uno 2004). It is proved by Moon and Moser that a given graph can have at most $O(3^{n/3})$ maximal cliques (Moon and Moser 1965). Bron and Kerbosch proposed a depth-first search algorithm to generate all the maximal cliques in a given graph $G$. The algorithm uses a recursive backtracking scheme that considers the graph nodes one by one. It expands the current candidate clique by including the particular node to it or by adding the node to the set of excluded vertices. The excluded vertices cannot be a part of the clique but may have some non-neighbor in the clique (Bron and Kerbosch 1973). Later on, Tomita et al developed an algorithm to list all maximal cliques by deploying pruning techniques used in Bron-Kerbosch algorithm (Tomita et al. 2006).
The Bron-Kerbosch algorithm and its variations are considered as greedy clique generation algorithms. Tsukiyama et al. proposed an output-sensitive (polynomial delay) algorithm to list all maximal independent sets (cliques) in $G$ in $O(|V||E|)$ time, where $V$ is a vertex set and $E$ is an edge set in $G$ (Tsukiyama et al. 1997). Working on the same line, Johnson et al. developed a polynomial time delay algorithm to generate all maximal cliques in lexicographical order (Johnson et al. 1998). The algorithm takes $O(|V||E|)$ time between two consecutive cliques and uses $O(|V|N)$ space, where $N$ is the number of maximal cliques. Later on in 2004, Makino and Uno built two new clique enumeration algorithms (Makino and Uno 2004) based on the idea of Tsukiyama and the reverse search technique (Avis and Fukuda 1996). One of their algorithms is based on matrix multiplication. It enumerates cliques using $O(n^2)$ space in $O(Mul(n))$ time, where $Mul(n)$ is the time required to multiply two matrices of size $n \times n$. The other algorithm lists maximal cliques in $O(\Delta^4N)$ time, where $\Delta$ is the maximal degree of $G$. Their results showed that their algorithms are faster than those of Tsukiyama for sparse graphs.

2.4.2 Bipartite Cliques - Bicliques

In a bipartite graph $G$, the vertex set $V$ consists of two disjoint nonempty sets $V_1$ and $V_2$, and the edge set $E \subseteq V_1 \times V_2$. A bipartite graph is called complete if $E = V_1 \times V_2$. A complete bipartite subgraph is called a biclique and is said to be maximal if it is not a proper subgraph of another biclique.

Bicliques are widely used to model various real-world problems. They have applications in co-clustering, automata theory, artificial intelligence, graph compression, cyber-communities, protein interactions discovery and social networks (Mishra et al. 2004; Ding et al. 2006; Guillaume and Latapy 2006; Murata 2007; Dhillon et al. 2003). Bipartite cliques are the vital structure in different types of graphs. The listing of all bipartite cliques in a graph can be used in recognition algorithms (Prisner 1997). Alexe et al. developed a scheme for enumerating all non-induced bicliques, while Dias
gave a generation algorithm for all induced bipartite cliques in a graph (Alexe et al. 2004; Dias et al. 2007).

The clique enumeration algorithm by Tomita can be extended to list all bicliques (Tomita et al. 2006). Similarly, Makino and Uno also proposed an algorithm to generate all maximal bicliques based on their clique algorithm (Makino and Uno 2004). The algorithm runs with $O(\delta^2)$ time delay and in $O(|V| + |E| + N)$ space, where $\delta$ is the maximum degree of a vertex and $N$ is the total number of maximal bicliques in $G$.

Some of the bicliques generation algorithms have roots in data mining and use techniques of closed pattern detection. There is a one-one correspondence between maximal bicliques and frequent closed item-sets (Li et al. 2007). Thus, frequent item-set mining algorithms can be deployed to mine maximal bicliques.

### 2.4.3 Pseudo-Cliques and Pseudo-Bicliques

Clique and bicliques cannot handle missing data, because of their rigid connectivity requirements. Therefore, researchers are now considering pseudo-structures, that is, pseudo-cliques and pseudo-bicliques to model more natural interactions in real world problems.

There are many ways to define a pseudo-clique. In one model, a pseudo-clique is acquired by deleting fixed number of edges from a complete graph. Another model defines a pseudo-clique in terms of density. In this case, a pseudo-clique is a subgraph that has density greater than a given threshold value. Here, the density is the ratio of the number of edges in the pseudo-clique, to the number of edges in a complete graph of the same size. In the first model, certain number of edges is removed from graphs of any size. Thus, large subgraphs can lose only a few edges, and many trivial vertex sets will become pseudo-cliques. However, in the case of the density based definition, the restraint on the number of edges changes with the size of the subgraph. Furthermore, small subgraphs are classified as pseudo-cliques only if they are cliques.
Many schemes have been proposed to enumerate pseudo-cliques (Mishra et al. 2004; Ding et al. 2006; Uno 2010; Abello et al. 2002; Li et al. 2008). Gibson et al. proposed an algorithm for finding disjoint dense subgraphs in a given graph (Gibson et al. 2006). As this algorithm generates an incomplete list, it may skip some useful dense graphs. The problem of generating all pseudo-bicliques can be compared with the data mining techniques to list pseudo-frequent item sets in a transactional database. Some mining algorithms (Liu et al. 2005; Mishra et al. 2004) deal with the missing item-set by considering it a fault if some of the items are missing in a transaction. These approaches alter just one partition of the biclique; that is, they consider only frequent item sets and do not deal with the frequent transactions (Uno and Arimura 2008). However, in many applications related to chemistry and biology such as discovering protein-protein interactions, it is desirable that both partitions of pseudo-bicliques are changed.

2.4.4 Isolated Cliques and Isolated Bicliques

The concept of isolation was recently introduced to cliques (Ito and Iwama 2009). It computes the connectedness of the clique vertices to the other vertices in the graph. A clique of size $k$ in a given graph $G$ is called $c$-isolated if it has less than $ck$ outgoing edges (outgoing edges connect a clique with the rest of the graph). Here, $c$ is the isolation factor. Clique isolation is a very useful concept as it not only aids in developing efficient algorithms for the generation of maximal cliques, but can also be used in filtering out cliques with distinct properties. It can be used in social, financial and biological networks to discover communities with special features and semantics.

Iwama et al. proposed an algorithm for generating all $c$-isolated cliques in a given graph $G$ (Ito et al. 2005). The algorithm runs in linear time if the isolation factor $c$ is $O(constant)$ and in polynomial time if $c$ is $O(logn)$. The algorithm consists of three stages: trimming, generation and screening. In the trimming stage, unnecessary vertices that cannot be a part of any $c$-isolated clique are removed from the input graph.
G. Next in the generation stage, this algorithm generates all maximal cliques using the technique of fixed point tractability (Fernau 2005) for enumerating all minimal vertex covers in the complement of G. Finally, in the last stage, this algorithm screens the enumerated cliques and removes those cliques that do not satisfy the condition of isolation or maximality.

Recently, Komusiewicz et al. presented new ideas regarding clique isolation by extending Iwama’s concept (Huffner et al. 2009). They proposed an algorithm to generate all “maximal isolated cliques” in a given graph. These cliques are c-isolated and are not subset of any other c-isolated clique. However, Iwama’s algorithm produces “c-isolated maximal cliques”. These cliques are c-isolated and do not exist as subsets in any other clique. Furthermore, Komusiewicz et al. take the notion of isolation to the next level by introducing minimum, maximum and average isolation concepts. A clique of size \( k \) in a given graph is said to be minimum-c-isolated if it has at least one vertex with less than \( c \) outgoing edges and it is called maximum-c-isolated if every vertex in the clique has less than \( c \) outgoing edges. A clique is considered average-c-isolated if the number of its outgoing edges is less than \( ck \). The performance of these three isolation levels is compared by conducting computational experiments on synthetic graphs (with \( G_{n,m,p} \) model), music-artists networks and financial networks (Huffner et al. 2009).

Miyagawa et al. extend the notion of c-isolation to bicliques (H. Miyagawa and Iwama 2007). They define that a bipartite subgraph \( S(V_1, V_2, E) \) of size \( k \) is an isolated biclique, if there are less than \( k \) outgoing edges from \( S \). They developed an algorithm to enumerate all 1-isolated bicliques in a given graph \( G \). The algorithm depends on the value of a parameter called the bipartite factor and denoted by \( \alpha \). It is the ratio of the number of vertices in \( V_2 \) over \( V_1 \). A biclique is said to be \( \alpha \)-biclique if for a real number \( \alpha = 1 \), the size proportion of the parts (of a bipartite graph) is bounded by \( \alpha \). That is, \( |V_1| \leq |V_2| \leq \alpha |V_1| \). The bipartite factor \( \alpha \) drastically affects the running time of the algorithm. In this thesis, we propose another definition of a c-isolated biclique which eliminates the restriction on the part sizes of the bicliques.
2.4.5 Spanning Trees

A spanning tree of an undirected graph on \( n \) nodes is a set of \( n - 1 \) edges that connects all nodes. The generation of all spanning trees in a given graph is a classical problem, that has various applications (Mayeda 1972).

There was much early work on spanning tree enumeration (Chase 1970; Minty 1965). A systematic way to list all spanning trees was developed early by Feussner. Later on, Kapoor and Ramesh (Kapoor and Ramesh 2000) proposed an algorithm to generate the computation tree of relative changes between spanning trees in \( O(N + V + E) \) time, which is optimal. If desired, all spanning trees can be explicitly enumerated using this computation tree.

One of the interesting problems in spanning tree generation is to list all the spanning trees in revolving door order. Smith introduces a method to generate all spanning trees in revolving door order. This algorithm is based on Feussner’s idea for listing all spanning trees (Knuth 2005).

2.4.6 Paths in Cliques

In the previous sections, we discussed the generation of pseudo and isolated cliques. Another interesting problem related to cliques is to generate all paths in revolving door order. Many algorithms have been proposed in the literature to generate all paths in
Table 2.1: Johnson and Trotter permutations for $n = 4$

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Harada developed a scheme to generate all hamiltonian circuits based on the Johnson-Trotter gray code for permutations (Harada 1971; Johnson 1963; Trotter 1962). The Johnson-Trotter method generates permutations by swapping two elements in adjacent positions. Their recursive scheme to list permutations for $n = 4$ is illustrated in Table 2.1. Harada shows that the hamiltonian paths can be obtained from the hamiltonian circuits. However, these hamiltonian paths are not in revolving door order.

Langdon proposed a method to generate all $n!$ permutations in cyclic order (Langdon 1967). This method requires the support of the hardware to efficiently generate all $n$ permutations. The problem of generating all spanning paths in a clique is different from the Langdon method, as the reverse of an undirected path $P$ is same as $P$. 

a given graph (Knuth 2005). However, no research was conducted on generating all paths or spanning paths in a clique in minimal change order.
2.4.7 Conclusion

This chapter briefly reviews the combinatorial structures and current state of knowledge pertaining to the proposed research. It also describes the basic terminologies that are needed to understand and analyze the generation algorithms that are proposed in the next chapters.
Chapter 3

Generation Algorithms for Pseudo-Bicliques

3.1 Introduction

Bicliques are widely used to solve many real life problems as they can capture real-world interactions such as documents and words co-clustering, images and features co-clustering and protein interaction discovery (Li et al. 2007; Li et al. 2008; Ding et al. 2006). However, the rigid connectivity requirement of a biclique makes it unsuitable for dealing with incorrect and missing data. If an edge in a biclique is removed it is no more a biclique. But, it is still a dense structure and can represent natural interactions in real-life situations.

In this thesis, the density based model of a pseudo-biclique is investigated. This model defines a pseudo-biclique as a bipartite subgraph such that the ratio of the number of its edges to the number of edges in a biclique of the same size is no less than a given threshold value. Many different models of pseudo-bicliques were proposed in the literature (Li et al. 2008; Dias et al. 2007; Uno and Arimura 2008; Abello et al. 2002).
The generation of density based pseudo-bicliques is a non-trivial task because straightforward back-tracking and branch-and-bound schemes involve an NP-complete problem (Uno 2010). Secondly, the monotone property does not hold in the family of density based pseudo-bicliques. Therefore, one cannot say that every subset of a pseudo-biclique is also a pseudo-biclique. Because of the anti-monotone property, various techniques proposed in the literature to enumerate combinatorial objects are not applicable to density based pseudo-bicliques.

In this chapter, an efficient algorithm for generating weighted pseudo-bicliques in a given graph $G$ is developed and is extended to enumerate multilevel pseudo-bicliques. The performance of the algorithm is evaluated on randomly generated bipartite graphs and real-world problems. We consider two real-world datasets: stocks-financial ratios and social network for movie ratings. The results are quite promising and show that the algorithm takes linear time to list each pseudo-biclique.

3.2 Preliminaries

A graph $G = (V, E)$ comprises a set of vertices $V$ and a set of edges $E \subseteq V \times V$. Without loss of generality, we assume that $G$ is simple and undirected. A graph $G$ is bipartite if its vertex set $V$ can be partitioned into two disjoint nonempty sets $V_1$ and $V_2$, and every edge in $E$ connects a vertex in $V_1$ and a vertex in $V_2$ such that $E \subseteq V_1 \times V_2$. Note that there is no edge in $G$ that joins any two vertices within $V_1$ or $V_2$. A bipartite graph $G$ is often denoted as $G = (V_1 \cup V_2, E)$. A graph $S = (V', E')$ is a subgraph of a graph $G$ if $V' \subseteq V$ and $E' \subseteq E$. The subgraph $S$ is a biclique (bipartite clique) if every vertex in $V' \cap V_1$ is adjacent to every vertex in $V' \cap V_2$, and it is maximal if no other biclique contains $S$ properly. We denote the degree of a vertex $v$ in a subgraph $S$ by $\text{deg}_S(v)$, the maximum degree by $\Delta(S)$ and the minimum degree by $\delta(S)$.

Next, we introduce our definition of a density based pseudo-biclique. The density of
a bipartite subgraph is defined as the ratio of the number of its edges, to the number of edges in a biclique of the same size.

**Definition 3.2.1.** The density \( \rho(G) \) of a bipartite graph \( G = (V_1 \cup V_2, E) \) is given by

\[
\rho(G) = \frac{|E|}{|V_1||V_2|}.
\]

**Definition 3.2.2.** A pseudo-biclique \( B_{U_1,U_2} \) is a bipartite subgraph of a graph \( G \), if \( \rho(B) \geq \theta \), where \( 0 < \theta \leq 1 \).

Note that the density \( \rho(G) = 1 \) if \( |V_1| = 0 \) or \( |V_2| = 0 \). We label the vertices in \( G(V_1 \cup V_2, E) \) in lexicographic order.

### 3.2.1 Weighted Graph

A weighted graph is defined as a triplet \( G = (V, E, w) \), where \( w : E \to \mathbb{R} \) is a function that assigns a weight (real number) to each edge. The weight of the edge between the vertex \( i \) and the vertex \( j \) is denoted by \( w_{i,j} \) and \( 0 \leq w_{i,j} \leq 1 \). A weighed bipartite graph \( G \) is denoted by \( G = (V_1 \cup V_2, E, w) \). A graph \( S = (V', E', w') \) is a weighted subgraph of \( G \) if \( V' \subseteq V, E' \subseteq E \) and \( w'_{i,j} = w_{i,j} \).

**Definition 3.2.3.** The weighted degree of a vertex \( v \) in a weighted graph \( G(V, E, w) \) is defined as

\[
\deg(v) = \sum_{u \in V} w_{u,v} \quad (3.1)
\]

**Definition 3.2.4.** The density \( \rho(G) \) of a weighted bipartite graph \( G(V_1 \cup V_2, E, w) \), is given by

\[
\rho(G) = \frac{\Sigma_{i \in V_1} \deg(i)}{|V_1| \times |V_2|} = \frac{\Sigma_{j \in V_2} \deg(j)}{|V_1| \times |V_2|} \quad (3.2)
\]

### 3.3 Generation of Weighted Pseudo-Bicliques

In this section, an algorithm for generating weighted pseudo-bicliques is developed. The framework of this algorithm is based on reverse search which is a sophisticated
type of depth first search (Avis and Fukuda 1996). It is widely used in the field of combinatorial generation because of its simplicity and efficiency.

In reverse search, a tree-shaped traversal route (called enumeration tree) is constructed on the family of a combinatorial object. In our case, we have to construct an enumeration tree on the set of pseudo-bicliques to be generated. This can be accomplished by establishing an adjacency relation on this set. In order to develop the adjacency relation, we define a parent for each pseudo-biclique and ensure that the definition of the parent is unique and acyclic. The set of pseudo-bicliques can be generated by traversing the enumeration tree in the depth first search manner. Note that in reverse search scheme, there is no need to save the enumeration tree in the memory. In fact, we develop a technique to discover the children of a pseudo-biclique. We start at the root of the tree and recursively find all its children (pseudo-bicliques) proceeding in depth first manner.

We need to build an adjacency relation on the set of weighted pseudo-bicliques to enumerate all such structures. It is observed that the removal of a vertex with the minimum weighted degree from a bipartite subgraph does not decrease the density of the resultant subgraph. If there are more than one minimum degree vertices, then we consider the one with the minimum index (label). This observation can be used to define an adjacency relation.

**Fact 3.3.1.** Let $v$ be a minimum weighted degree vertex in $G = (V_1 \cup V_2, E, w)$, then

$$
\Sigma_{k \in V_i} \deg_G(k) \geq |V_i| \deg_G(v), i = 1, 2
$$

(3.3)

**Lemma 3.3.1.** Let $B_{U_1, U_2}$ be a pseudo-biclique, and suppose $v \in (U_1 \cup U_2)$. If $\deg_{B_{U_1, U_2}}(v) = \delta(B_{U_1, U_2})$, then $\rho(B_{U_1, U_2} \setminus v) \geq \rho(B_{U_1, U_2})$.

**Proof.** We have to prove that the density of $B_{U_1, U_2} \setminus v$ is no less than the density of $B_{U_1, U_2}$. Without loss of generality, we can assume that $v \in U_2$. Now, we show
\[ \rho(B_{U_1,U_2} \setminus v) - \rho(B_{U_1,U_2}) \geq 0 \]

By substituting the value of density in the left side of the above inequality, we get

\[
\frac{\sum_{i \in U_1} \text{deg}_{B_{U_1,U_2}}(i) - \text{deg}_{B_{U_1,U_2}}(v)}{|U_1| \times |U_2 - 1|} - \frac{\sum_{i \in U_1} \text{deg}_{B_{U_1,U_2}}(i)}{|U_1| \times |U_2|} \]

\[
= \frac{\sum_{i \in U_1} \text{deg}_{B_{U_1,U_2}}(i) - |U_2| \times \text{deg}_{B_{U_1,U_2}}(v)}{|U_1| \times |U_2| \times |U_2 - 1|} \]

\[
\geq \frac{|U_2| \times \text{deg}_{B_{U_1,U_2}}(v) - |U_2| \times \text{deg}_{B_{U_1,U_2}}(v)}{|U_1| \times |U_2| \times |U_2 - 1|} \]

\[
= 0
\]

From the above result, it is evident that the density of a pseudo-biclique is not greater than that of its parent. Consequently, for any pseudo-biclique \( B_{U_1,U_2} \), we can say that \( B_{U_1,U_2} \setminus v \) is also a pseudo-biclique provided that \( v \) is a minimum degree vertex in \( B_{U_1,U_2} \).

Next, we define a unique parent-child relationship among pseudo-bicliques.

**Definition 3.3.1.** Let \( B_{U_1,U_2} \) be a pseudo-biclique then a pseudo-biclique \( B^* = (B_{U_1,U_2} \cup v) \) is a child of \( B_{U_1,U_2} \) if one of the following conditions hold for each \( u \in (U_1 \cup U_2) \)

1. \( \text{deg}_{B^*}(v) < \text{deg}_{B^*}(u) \)
2. \( \text{deg}_{B^*}(v) = \text{deg}_{B^*}(u) \) and the index of \( v \) is less than the index of \( u \)

Figure 3.1 shows a pseudo-biclique \( B_{U_1,U_2} \) comprising of vertices \( \{2, 3, 6, 7, 8\} \). The edge weights are given in a matrix format. It is easy to see that if we add a vertex
Figure 3.1: An illustrative example of a weighted pseudo-biclique

from the set \( T = \{1, 5, 9\} \) to \( B_{U_1, U_2} \), then we get a child of \( B_{U_1, U_2} \). Vertex 4 is not included in \( T \) because \( \deg_{B_{U_1, U_2}}(4) > \delta(B_{U_1, U_2}) \). The weighted degree of vertex 10 is equal to \( \delta(B_{U_1, U_2}) \), yet it is not included in \( T \) because it is lexicographically larger than vertex 3, a minimum degree vertex of \( B_{U_1, U_2} \).

3.3.1 Generation Algorithm

In this section, we present our algorithm. First, we outline the main idea and then suggest some improvements to reduce the computation time of the algorithm. We have carried out computational experiments to prove that our suggested modification significantly improve the time bounds, and the algorithm takes linear time on average to generate each pseudo-biclique.

Generation algorithms that are based on reverse search begin with a trivial structure (an object of minimal size) and recursively find all its children. In our case, each vertex can be considered as a pseudo-biclique. The pseudocode for the generation scheme is given in Algorithm 2. It is written according to the syntax specified in Latex Algorithmic package. Algorithm 2 requires as input a weighted graph \( G(V_1 \cup V_2, E, w) \), a density threshold \( \theta \) and recursively generates all the pseudo-bicliques. Initially, we set \( B_{U_1, U_2} = \emptyset \). For each vertex \( v \in (V_1 \cup V_2) \setminus (U_1 \cup U_2) \), the algorithm enumerates \( B_{U_1, U_2} \cup v \) if it is a pseudo-biclique and lists all its children pseudo-bicliques.
Usually, we are not interested in star shaped graphs. To avoid such structures, we propose to start the algorithm with a subset of vertices that forms a pseudo-biclique. The basic idea is to enumerate all subsets of a given size in $G$ and then run the generation algorithm for each subset. We define a routine PrePseudoBiclique, that takes a graph $G$ and a density threshold $\theta$ as input. It generates all pseudo-bicliques that have exactly two vertices in each partition and have density threshold at least $\theta$. For each generated subset, PrePseudoBiclique routine invokes Algorithm GeneratePseudoBiclique to list all its children (pseudo-bicliques) recursively. In this way, we can avoid small useless pseudo-bicliques. However, there is one drawback in this preprocessing step, that is, we may skip some of the pseudo-bicliques for small density thresholds.

\textbf{Algorithm 1} PrePseudoBiclique

\textbf{Require}: Graph $G(V_1 \cup V_2, E, w)$, density threshold $\theta$

1: for each $(u_i, u_j) \in V_1$, $i \neq j$ do
2: \hspace{1em} Compute $N(u_i) \cup N(u_j)$ \{ $N(u_i)$ denotes neighborhood of vertex $u_i$ \}
3: \hspace{1em} for each $(u_k, u_l) \in \{N(u_i) \cup N(u_j)\}$ do
4: \hspace{2em} if $\rho\{u_i, u_j, u_k, u_l\} > \theta$ then
5: \hspace{3em} GeneratePseudoBiclique($G, \theta, \{u_i, u_j, u_k, u_l\}$)
6: \hspace{1em} end if
7: \hspace{1em} end for
8: end for

\textbf{Algorithm 2} GeneratePseudoBiclique

\textbf{Require}: Graph $G(V_1 \cup V_2, E, w)$, density threshold $\theta$, pseudo-biclique $B_{U_1, U_2}$

1: for each $v \in \{V_1 \cup V_2\} \setminus \{U_1 \cup U_2\}$ do
2: \hspace{1em} if $\rho(B_{U_1, U_2} \cup v) \geq \theta$ then
3: \hspace{2em} if $B_{U_1, U_2} \cup v$ is a child of $B_{U_1, U_2}$ then
4: \hspace{3em} Output $B_{U_1, U_2}$
5: \hspace{3em} GeneratePseudoBiclique($G, \theta, B \cup v$)
6: \hspace{2em} end if
7: \hspace{1em} end if
8: end for

The graphs are represented using adjacency lists. In the above algorithm, two basic
operations are performed before the inclusion of each vertex: find a minimum degree vertex and calculate the density. In simple implementation of the algorithm, the time needed to compute the density is $O(V)$ and the time to find the minimum degree vertex is also $O(V)$. Both of these operations are performed at most $V$ times in an iteration of the algorithm, thus the time to compute a pseudo-biclique is $O(V^2)$. Space requirement of the algorithm is quadratic since no additional space is used other than the weight matrix of the graph.

3.3.2 Maximal Pseudo-Bicliques

Algorithm 2 can be used with slight modification to list all maximal pseudo-bicliques. A pseudo-biclique $B_{U_1, U_2}$ is maximal if and only if there does not exist a vertex $v$ such that $v \notin U_1 \cup U_2$ and $\rho(B_{U_1 \cup v}) \geq \theta$. The existence of $v$ ensures that $B_{U_1, U_2}$ is not maximal. Hence, by addition of a simple constraint in Algorithm 2 we can list only the maximal pseudo-bicliques of the given graph $G(V_1 \cup V_2, E, w)$.

3.3.3 Improvements for Efficient Computation

The time taken by our algorithm can be reduced by keeping record of the minimum degree vertex and the degrees of all vertices in $B_{U_1, U_2}$ under consideration. We use an array to record the degree of each vertex in $B_{U_1, U_2}$. For any $v$, if we know $deg_{B_{U_1, U_2}}(v)$ and the degree sum of all the vertices in $B_{U_1, U_2}$, then the density of $B_{U_1, U_2} \cup v$ can be computed in a constant amount of time. The time required to verify the parent-child relationship can also be improved. In most cases, only a comparison between $deg_{B_{U_1, U_2}}(v)$ and $\delta(B_{U_1, U_2})$ is sufficient to do the job. We can subdivide the task of determining child of $B_{U_1, U_2}$ in one of the following three cases.

1. If $deg_{B_{U_1, U_2}}(v)$ is strictly less than $\delta(B_{U_1, U_2})$, then $B_{U_1, U_2} \cup v$ is a child of $B_{U_1, U_2}$

2. If $deg_{B_{U_1, U_2}}(v)$ is greater than $\delta(B_{U_1, U_2}) + w(v, \delta(B_{U_1, U_2}))$, then it is not a child of
3. Otherwise one of the two possibilities can occur

(a) If $v$ is connected to the minimum degree vertex of $B_{U_1, U_2}$, then verify the parent-child relationship

(b) If $v$ is not connected to the minimum degree vertex of $B_{U_1, U_2}$, then a comparison between the label of $v$ and the minimum degree vertex of $B_{U_1, U_2}$ completes the task

In all of the above cases except 3(a), the verification of a child node can be done in constant time. Only case 3(a) takes $O(V)$ time. Therefore, we can say that cases 1, 2 and 3(b) are constant checks (verifications) and case 3(a) is a non-constant check as it can take linear time.

Next, we show that the cost incurred on constant checks (verifications) can be distributed to the pseudo-bicliques as overhead. The generation of a pseudo-biclique takes $O(V)$, because when a vertex is added to $B_{U_1, U_2}$, then the degrees of all of its adjacent vertices are updated. This operation in fact takes $O(\Delta(G))$ time, where $\Delta(G)$ is the maximum degree of $G$ and $\Delta(G) \leq |V|$. For each $B_{U_1, U_2}$, the number of constant checks that does not yield any child are at most $O(V)$. Therefore, the overhead incurred for constant checks can be included in the generation cost of $B_{U_1, U_2}$. Hence, one can say that overhead of constant checks does not affect the asymptotic time bounds of the algorithm. In Section 3.6, we estimate the number of non-constant checks using computational experiments. It is found that the number of non-constant checks for generating all pseudo-bicliques in $G$ is $O(\beta(G))$, where $\beta(G)$ is the total number of pseudo-bicliques in $G$. 
3.4 Multilevel Pseudo-Bicliques

In many real-world scenarios, multiple dimensions are associated with one clustering entity. For example, in a social network of movie viewers, there are different genres of movies such as action, adventure, drama, romantic, family, musical, and horror. The existing algorithms for the generation of pseudo-bicliques deal with extracting knowledge at one concept level (dimension). However, mostly it is desired to discover information at multiple concept levels. Consider an example of stock and financial ratios; financial analysts wish to cluster stocks on the basis of financial ratios. Nevertheless, it will be more informative for analysts to know that stocks in a cluster are 80 percent similar in liquidity ratios and 50 percent similar in debt ratios.

It is highly desirable to integrate information regarding dimension into the mining process. This will be quite helpful in extracting valuable patterns from the dataset. The existing subspace clustering algorithms that deal with dimensions can not handle missing data. Thus, they can not be used on real world datasets that have noise and missing values. In this section, we propose a multilevel pseudo-biclique generation algorithm to extract multi-dimensional information from the noisy data-set.

We model the data with multiple dimensions using a specialized bipartite graph $G(V_1 \cup V_2, E, w)$, where $V_2$ is further partitioned into non-overlapping vertex sets to represent multiple concept levels. For instance, consider a social network of authors and books. We can model it as a bipartite graph, where the vertices in $V_1$ represent authors, those in $V_2$ represent books and edges represent connections between authors and books. To capture the concept at multiple levels, we divide $V_2$ into disjoint subsets to represent various categories of books such as novel, poetry, drama, and short story. A multilevel pseudo-biclique in this graph denotes a group of authors who have co-authored at-least $d$ books in each category, where $d$ is a given number.

Next, we formally define a multilevel bipartite graph and a multilevel pseudo-biclique. A multilevel bipartite graph $G(V_1 \cup \{V_{2_1} \cup V_{2_2} \cup \cdots \cup V_{2_k}\}, E, w)$ is a bipartite graph,
where $V_2$ is divided into $k$ non-overlapping sets, $V_2$, being the $i^{th}$ subset. A multilevel pseudo-biclique $B(U_1 \cup \{U_2, \cup U_2 \cup \cdots \cup U_{2k}\})$ is a subgraph of $G(V_1 \cup \{V_2, \cup V_2 \cup \cdots \cup V_{2k}\}, E, w)$ such that $U_1 \subseteq V_1$, $U_2 \subseteq V_2$, and $\rho(B_{U_1, U_{2i}}) \geq \theta_i$ for all $1 \leq i \leq k$. Here, $\theta_i$ is the density threshold for $B(U_1, U_{2i})$ and the value of $\theta_i$ can be different for all, $1 \leq i \leq k$. Figure 3.2 shows an example of a multilevel pseudo-biclique.

3.4.1 Generation Algorithm for Multilevel Pseudo-Bicliques

This section extends the generation algorithm for pseudo-bicliques to enumerate multilevel pseudo-bicliques. The following lemma establishes an adjacency relationship on the set of weighted multilevel pseudo-bicliques.

**Lemma 3.4.1.** Let $B(U_1 \cup \{U_{21} \cup \cdots \cup U_{2k}\})$ be a multilevel pseudo-biclique, and suppose $v$ is a minimum degree vertex in $B$. Then, we have

$$\rho(B(U_1 \cup \{U_{21} \cup \cdots \cup U_{2k}\}) \setminus v) \geq \rho(B(U_1 \cup \{U_{21} \cup \cdots \cup U_{2k}\}))$$

**Proof.** There are two cases; $v \in U_1$ or $v \in U_{2i}$. First, let $v \in U_{2i}$, then the removal of $v$ effects only the density of $B(U_1 \cup U_{2i})$. Therefore, from Lemma 3.3.1 $\rho(B(U_1 \cup \{U_{2i} \setminus v\}) \geq \rho(B(U_1 \cup U_{2i})) \geq \theta_i$.

Secondly, let $v \in U_1$. Note that the multilevel pseudo-biclique $B(U_1 \cup \{U_{21} \cup \cdots \cup U_{2k}\})$
can be represented as a collection of \( k \) pseudo-bicliques \( B(U_1 \cup U_2) \), \( 1 \leq i \leq k \). So if \( v \in U_1 \), then \( v \) is the minimum degree vertex in all \( k \) pseudo-bicliques. Thus, the repeated application of Lemma 3.3.1 ensures that the removal of \( v \) does not decrease the density of any of the \( k \) pseudo-bicliques. Hence, \( \rho(B(U_1 \setminus v \cup U_2)) \geq \rho(B(U_1 \cup U_2)) \) for all \( 1 \leq i \leq k \).

In order to enumerate all multilevel pseudo-bicliques, we use reverse search technique based on the parent-child relationship defined in Lemma 3.4.1. The multilevel pseudo-biclique can have different density thresholds for each of the \( k \) pseudo-bicliques. Thus, it has to be ensured that all density constraints are fulfilled after addition of a new vertex. Given a multilevel bipartite graph \( G \) and \( k \) density thresholds, Algorithm 3 recursively lists all maximal multilevel pseudo-bicliques in \( G \).

The time required to generate a multilevel pseudo-biclique increases by a factor of \( k \) as compared to the generation algorithm for weighted pseudo-bicliques. The additional time is required for computing the value of density and verifying the density thresholds for all \( k \) pseudo-bicliques.

### 3.5 Applications: Co-clustering and Social Networks

The concept of weighted and multilevel pseudo-bicliques can be applied to combine related stocks on the basis of financial ratios and vice versa. Financial ratios are valuable indicators of a company’s financial situation and future performance. They are used to examine trends, compare the financial state of the companies and foresee future bankruptcy.

In order to extract meaningful information, the ratios must be compared to historical values of the same company or with the ratios of related companies. For this reason, stocks (or any other security) are clustered based on financial ratios (Eklund et al. 2002; Magnusson et al. 2005). The clusters are then carefully examined to discover
Algorithm 3 GenerateMultiPseudoBiclique

Require: Graph $G(V_1 \cup V_2 \cup \cdots \cup V_k, E, w)$, density thresholds $\theta_1, \theta_2, \cdots, \theta_k$, multilevel pseudo-biclique $B(U_1 \cup \{U_2 \cup U_2 \cup \cdots \cup U_k\})$

1: $\text{isMaximal} := \text{true}$
2: for each $v \notin U_1 \cup U_2 \cup \cdots \cup U_k$ do
3: $\text{isDense} := \text{true}$
4: if $v \in V_1$ then
5: for each $i := 1$ to $k$ do
6: if $\rho(B(\{U_1 \cup V\} \cup U_2)) < \theta_i$ then
7: $\text{isDense} := \text{false}$
8: \end if
9: \end for
10: else if $v \in V_2$ then
11: $i := \text{set}[v]$
12: if $\rho(B(U_1 \cup \{U_2 \cup v\})) < \theta_i$ then
13: $\text{isDense} := \text{false}$
14: \end if
15: \end if
16: if $\text{isDense}$ then
17: $\text{isMaximal} := \text{false}$
18: if $B(U_1 \cup \{U_2 \cup \cdots \cup U_k\}) \cup v$ is a child of $B(U_1 \cup \{U_2 \cup U_2 \cup \cdots \cup U_k\})$ then
19: GenerateMultiPseudoBiclique($G, \theta, B(U_1 \cup \{U_2 \cup \cdots \cup U_k\}) \cup v$)
20: \end if
21: \end if
22: \end for
23: if $\text{isMaximal}$ then
24: output $B(U_1 \cup \{U_2 \cup \cdots \cup U_k\}) \cup v$
25: \end if
related stocks. This helps understand the company’s current financial situation and predict its future behavior. A technique called subspace clustering is used to find clusters in multidimensional data (Parsons et al. 2004). However, these algorithms do not handle missing data. Some researchers proposed that the self-organizing maps (SOM) can be used to combine stocks based on financial ratios (Eklund et al. 2002; Magnusson et al. 2005). SOMs has its own limitations. It is difficult for a user to define clear clusters of entities using SOM because the boundaries of the clusters are hard to differentiate.

In this thesis, we propose the use of weighted pseudo-bicliques to co-cluster stock and financial ratios. Depending on the nature of the study, some ratios are more important than others. For example for value investment, investment valuation ratios are very important but role of other ratios like cash flow indicator cannot be completely ignored. In this scenario, it would be highly beneficial to assign weights to ratios and then cluster them. Furthermore, to provide another level of abstraction we propose the use of multilevel pseudo-bicliques. In this case, we exploit the fact that ratios have multiple classifications such as liquidity ratios, efficiency ratios, debt ratios and so on. Thus, they can be considered as multilevel entities to extract more information.

Pseudo-bicliques are also beneficial in extracting useful information from social networks. For instance, they can help locate users with similar interests in social networks. In this thesis, we conduct experiments on a social network for movie ratings by users. The information extracted from the movie network can be quite useful for social networking sites to recommend movies to their users.

### 3.6 Computational Experiments

We performed computational experiments to evaluate the performance of our algorithms on randomly generated bipartite graphs and real world data. We considered two real world datasets: stocks-financial ratios and social network for movie ratings.
Nodes | Edges | Edge density | Density threshold | Pseudo-bicliques | Non-constant checks | Ratio
--- | --- | --- | --- | --- | --- | ---
500 | 639 | 1% | 95% | 3912 | 842 | 0.21523
500 | 1238 | 2% | 95% | 47678 | 3389 | 0.07108
500 | 1841 | 3% | 95% | 484777 | 8899 | 0.01805
500 | 2527 | 4% | 95% | 1.17e+10 | 21499 | 0.001836
500 | 3149 | 5% | 95% | 4.16e+10 | 41154 | 0.00099

Table 3.1: Experimental results for different edge densities

The results are quite promising and show that on average the algorithm takes linear time to generate each pseudo-biclique. The experiments are performed on Windows 7 environment, using Intel(R) Core(TM)i7 CPU with 6GB RAM. The algorithm is implemented in C++ language using the Boost Graph Library (Boost 2010).

3.6.1 Random Graphs

In this experiment, the ratio of the number of pseudo-bicliques in a graph to the number of non-constant checks (as described in Section 3.3.3) is estimated. The number of pseudo-bicliques depends on factors such as density threshold and graph size. For this purpose, we evaluated the performance of our algorithm on three parameters: density threshold, number of vertices, and edge density. Three different experiments are conducted. In the first experiment, we estimated the ratio for different edge densities. The second experiment examines the effect of various density thresholds and the third deals with different graph sizes. The results of these experiments are shown in Tables 3.1, 3.2, 3.3 and Figure 3.3. Note that the edges in our randomly generated bipartite graphs are uniformly distributed.

Two observations are made from the above experiments. First, the total number of non-constant checks is less than the total number of pseudo-bicliques generated in all three experiments. From this observation, we infer that the cost of non-constant checks can be distributed to the enumerated pseudo-bicliques. Hence, the amount of work done per pseudo-biclique is $O(V)$. Moreover, it is observed that when we
### Table 3.2: Experimental results for different density thresholds

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Edges</th>
<th>Edge density</th>
<th>Density threshold</th>
<th>Pseudo-bicliques</th>
<th>Non-constant checks</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>639</td>
<td>1%</td>
<td>95%</td>
<td>3912</td>
<td>842</td>
<td>0.21523</td>
</tr>
<tr>
<td>1000</td>
<td>2564</td>
<td>1%</td>
<td>95%</td>
<td>219579</td>
<td>7198</td>
<td>0.032781</td>
</tr>
<tr>
<td>1500</td>
<td>5724</td>
<td>1%</td>
<td>95%</td>
<td>2025×10^6</td>
<td>24346</td>
<td>0.010827</td>
</tr>
<tr>
<td>2000</td>
<td>9915</td>
<td>1%</td>
<td>95%</td>
<td>5.91×10^7</td>
<td>56575</td>
<td>0.000975</td>
</tr>
</tbody>
</table>

### Table 3.3: Experimental results for different number of nodes

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Edges</th>
<th>Edge density</th>
<th>Density threshold</th>
<th>Pseudo-bicliques</th>
<th>Non-constant checks</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>639</td>
<td>1%</td>
<td>95%</td>
<td>3912</td>
<td>842</td>
<td>0.21523</td>
</tr>
<tr>
<td>1000</td>
<td>2564</td>
<td>1%</td>
<td>95%</td>
<td>219579</td>
<td>7198</td>
<td>0.032781</td>
</tr>
<tr>
<td>1500</td>
<td>5724</td>
<td>1%</td>
<td>95%</td>
<td>2025×10^6</td>
<td>24346</td>
<td>0.010827</td>
</tr>
<tr>
<td>2000</td>
<td>9915</td>
<td>1%</td>
<td>95%</td>
<td>5.91×10^7</td>
<td>56575</td>
<td>0.000975</td>
</tr>
</tbody>
</table>

Increasing the graph size (or decrease the density threshold), the growth rate of non-constant checks is far less than the growth rate of pseudo-bicliques. Thus, we can deduce that the average cost of computing a pseudo-biclique decreases as the search space of the algorithm increases.

#### 3.6.2 Stock Market: Stocks and Financial Ratios

The concept of weighted and multilevel pseudo-bicliques can be applied to combine related stocks based on financial ratios. The existing algorithms (Li et al. 2008; Liu et al. 2005; Mishra et al. 2004) are incapable of finding maximal multilevel pseudo-bicliques. We obtain two datasets for the year 2007 – 2008 (Portfolio 2010). One consists of stocks and financial ratios of medium size firms and the other contains stocks and ratios of large size firms. There are significant differences in ratios of large public and small private firms (Phillips et al. 2009). Therefore, it is meaningless to compare ratios of the different sized firms.
Figure 3.3: Results of the experiments on random graphs
Categories of Financial Ratios

The choice of ratios plays an important role and greatly depends on the purpose of the analysis which can be value investment, trend analysis or bankruptcy prediction. The weights can be assigned to the ratios according to their importance and relevance.

In our experiments, we consider only the basic categories of ratios such as valuation, liquidity, financial strength, profitability and growth. The ratios included in each category is as follows:

**Valuation** includes dividend yield, price earnings, price/earnings to next year growth rate, price to book, price to cash flow (per share) and price to sales.

**Liquidity** includes current ratio.

**Financial Strength** includes total debt to total equity and annual payout ratio.

**Profitability** includes return on investment and return on equity.

**Growth** includes EPS growth rate, gross margin, net income and sales growth rate.

Datasets and Discretization of Financial Ratios

Financial ratios consist of continuous values. Therefore, we use the hierarchical clustering (Garey and Johnson 2005) method to partition the range of each financial ratio into discrete clusters. There are several variations of the hierarchical clustering, we use the pairwise centroid-linkage method.

We represent the dataset as a bipartite graph. The vertices in the first partition of the bipartite graph represent stocks and in the second partition represent financial ratio intervals. For the medium size firms, we consider 15 financial ratios (specified above) and partition each ratio into 25 clusters (this gives 375 ratio intervals). The bipartite graph for medium firms consists of 603 nodes (228 stocks and 375 financial ratio
We used two different weight vectors, namely $W_1$ and $W_2$, to examine the effect of weights on the number of pseudo-bicliques and clustering. The weight assigned to a ratio specifies its importance (role) in the analysis. The weights are assigned to the ratios by a financial analyst. All the outgoing edges from a ratio will have the same weight. In $W_1$, we assign 0.75 weight to price earnings and return on investment ratio, and the weight of other ratios is fix to one. In $W_2$, we set the weight of price to book and price to cash flow to 0.8, the weight of current ratio, sales growth rate and net income to 0.5 and the weight of other ratios is same as in $W_1$.

For the large firms, we include 15 financial ratios and partition each ratio into 20 clusters. The bipartite graph for large-sized firms consists of 401 nodes (101 stocks and 300 financial ratio intervals) and 1329 edges. We performed two experiments on the dataset of large firms. In one experiment, we find the weighted pseudo-biclique and in the other we look for the multilevel pseudo-bicliques. In the multilevel pseudo-biclique experiment, we grouped 15 financial into three groups. The first group consists of valuation ratios, the second contains growth ratios and the third consists of liquidity, profitability and financial strength ratios. We used the same density threshold for all levels and assign weight one to all the ratios.

The pseudo-bicliques that have multiple edges between stocks and discretized clusters of a financial ratio do not provide any useful information. So, we pruned them to reduce the size of search space. The total number of pseudo-bicliques in a graph is often very large, so it would be more interesting to consider only the maximal pseudo-bicliques. As described above, with slight modification our algorithm can list all maximal pseudo-bicliques.

**Observations and Results**

The results of the experiments are shown in Figures 3.4 and 3.5. We inspect the output set generated by the experiment on medium-sized firms to find the effect of
Figure 3.4: Results of the experiments on medium-sized firms.
weight vectors. The weight vectors greatly help in pruning the unnecessary pseudo-bicliques, see the graphs in Figures 3.4(a) and 3.4(b).

We carefully examined the output generated by two algorithms; weighted and multi-level pseudo-biclique generation algorithm. It is observed that the output sets of both algorithms do not completely coincide. A multilevel pseudo-biclique with the same $\theta$ threshold for each level is a $\theta$ weighted pseudo-biclique. However, this is not true for the maximal pseudo-bicliques. A maximal multilevel pseudo-biclique may not be a maximal weighted pseudo-biclique. In the case of multilevel pseudo-biclique, the error is more evenly distributed, especially when the value of $\theta$ is large. But, this is not true for the weighted pseudo-bicliques, where the error can be skewed.

In this experiment, we compare the number of non-constant checks (Section 3.3.3) with the number of pseudo-bicliques generated. It is interesting to note that the number of non-constant checks are far less than the total pseudo-bicliques. Therefore, we can distribute the overhead of non-constant checks to the pseudo-bicliques. From this, it can be concluded that our algorithm takes linear time on average to list each pseudo-biclique.

3.6.3 Social Network for Movie Ratings by Viewers

This section presents results of the computational experiments conducted on a social network for movie ratings. The dataset for these experiments is obtained from Movielens datasets (MovieLens 2011). It has been collected by the Grouplens research project at the University of Minnesota during the seven-month period from September, 1997 through April, 1998. The data consists of 10000 ratings for 1682 movies by 943 users. The users have rated the movies on the scale of 1 to 5 and each user has rated at least 20 movies.

We can represent the above dataset as a bipartite graph, where the nodes in the first partition are users and the nodes in the second are rankings of movies. We built two
Figure 3.5: Results of the experiments on large-sized firms

bipartite graphs for this dataset. In the first graph, the ranking scale is 1 - 5. In the second graph, the scale is merged into two: likes (includes rankings 3 - 5) and dislikes (includes rankings 1 - 2). The first bipartite graph has 943 users in the first partition and 8410 rankings of 1682 movies in the second. The second graph has 943 users and 3364 rankings. Both graphs have 9420 edges. The results of these experiments are shown in Figure 3.6.

The number of the pseudo-bicliques is very large, so it would be more interesting to output only the maximal pseudo-bicliques. The generated pseudo-bicliques provide groups of users who have similar interests in movies.

This study examines few co-clusters mined by our algorithm and shows that they are quite useful for social networking sites to recommend movies and movie friends to their users. The users in a cluster have similar interest (in movies), so they can recommend the movies that some users (in a cluster) have appreciated to others.

Some of the interesting co-clusters and possible movie recommendations extracted
Nodes | Edge density | Density threshold | Clusters | Pseudo-bicliques | Non-constant checks | Maximal Pseudo-bicliques
--- | --- | --- | --- | --- | --- | ---
9353 | 0.0011878 | 90% | 5 | 7.66E+03 | 152 | 2422
9353 | 0.0011878 | 95% | 5 | 2.20E+03 | 149 | 1253
4307 | 0.0029694 | 90% | 2 | 2.14E+06 | 1728 | 54989
4307 | 0.0029694 | 95% | 2 | 2.59E+04 | 1506 | 3519

Table 3.4: Experimental results for movie rating network

from the results are as follows:

**Co-Cluster 1** Users with Ids \{33, 46, 119, 143, 356\} have the same ratings for the movies: Titanic, Conspiracy Theory and The Game. We can recommend God Father, Crash, The Edge, Wag the Dog and The Jackal and not-recommend Fluber and A thousand Acres

**Co-Cluster 2** Users with Ids \{282, 673, 740\} have the same ratings for the movies: The old lady who worked in the sea, The full Monty, Scream, Liar Liar. We can recommend Contact, Cop Land, The English Patient and The Apostle and not-recommend Crash and A Smile Like Yours

**Co-Cluster 3** Users with Ids \{408, 752, 812, 898\} have the same ratings for the movies: The English Patient, Scream, Air Force One and Everyone says I love you saint . We can recommend Starship Troopers, Apt Pupil, Liar Liar and Everyone Says I Love You and not-recommend The Rainmaker, Air Bud, Lost Highway and Devil’s Own

The co-clusters produced by the experiment on the bipartite graph with cluster size 5 are relatively small in size as compared to co-clusters produced by the experiment with cluster size 2. Furthermore, the results confirmed that the non-constant checks (described in Section 3.3.3) are far less than the number of pseudo-bicliques generated. Thus, one can distribute the overhead of non-constant checks to pseudo-bicliques.
Figure 3.6: Results of the experiment on movie rating network
3.7 Conclusion

In this chapter, we present an algorithm to generate all density-based pseudo-bicliques in a given graph. The algorithm is extended to enumerate all multilevel pseudo-bicliques. The performance of the algorithm is evaluated on randomly generated bipartite graphs and real-world problems. The results show that the algorithm takes linear time on average to generate each pseudo-biclique.
Chapter 4

Generation Algorithm for c-Isolated Bicliques

4.1 Introduction

Bicliques are often used to model web communities and social networks (Li et al. 2007; Murata 2007). A web community is defined as a set of sites that has more connections with members of the community as compared to non-members. A biclique provides no control over the external links of the nodes. So the communities modeled by bicliques may or may not be cohesive. In this chapter, we propose that the c-isolated bicliques can be used to capture strong communities in real world networks.

The concept of subgraph isolation is a new one and is recently introduced to cliques (Ito and Iwama 2009; Huffner et al. 2009). A subgraph is said to be isolated, if the number of its outgoing edges is less than the number of its vertices. The monotone property does not hold in the family of c-isolated bicliques. Therefore, the various combinatorial generation techniques proposed in the literature can not be deployed to list c-isolated bicliques.

In this chapter, we design an efficient algorithm for enumerating all c-isolated bicliques in a given graph $G$. The framework of our algorithm is inspired by the technique used
to enumerate c-isolated cliques (Ito and Iwama 2009). We evaluate the performance of our algorithm on synthetic graphs and real-world problems. The results are very favorable and show that the algorithm takes linear time to generate all c-isolated bicliques.

4.2 Preliminaries

Let $E(X, Y)$ be a set of edges connecting two subgraphs induced by the vertex sets $X$ and $Y$ of an undirected graph $G = (V, E)$. For a vertex set $X \subseteq V$, an outgoing edge is an edge $e \in E(X, V - X)$, and for a vertex $v \in X$, its outgoing edges are the outgoing edges of $X$ that are incident on $v$. A vertex set $X$ is said to be c-isolated, if the number of its outgoing edges is less than $c|X|$, that is, $|E(X, V - X)| < c|X|$. A c-isolated clique is a subgraph $S$ of $G$ such that every pair of vertices in $S$ has an edge between them and its outgoing edges are less than $c|S|$, where $c > 0$. The ‘c’ is called isolation factor.

For a biclique $C(C_1 \cup C_2, E)$ of a bipartite graph $G(V_1 \cup V_2, E)$, the outgoing edges are denoted by $E(C, V - C)$, where $C = C_1 \cup C_2$. Now, the number of outgoing edges from $C_1$ is given by $|E(C_1, V - C)|$, as $C$ is a bipartite clique. Similarly, the number of outgoing edges from $C_2$ is $|E(C_2, V - C)|$. The neighborhood of a vertex $v$ is given by $N(v) = \{u \in V|(u, v) \in E\}$. The degree of $v$ in $G$ is denoted by $\text{deg}(v)$. Note that $\text{deg}(v) = N(v)$.

Next, we introduce the definition of a c-isolated biclique.

**Definition 4.2.1.** A biclique $C(C_1 \cup C_2, E)$ of a bipartite graph $G(V_1 \cup V_2, E)$ is said to be c-isolated if $|E(C_1, V - C)| < c|C_1|$ and $|E(C_2, V - C)| < c|C_2|$, where $c > 0$.

Figure 4.1 shows an example of c-isolated biclique, where $c = 2$. The black vertices in the figure form a 2-isolated biclique and the dotted lines represent the outgoing edges of the biclique.
4.3 Generation of c-Isolated Bicliques

In this section, we address the problem of generating all the c-isolated bipartite cliques in $G$.

**Lemma 4.3.1.** For any c-isolated biclique $C(C_1 \cup C_2, E)$, there are two vertices $v_i \in C_1$ and $v_j \in C_2$ such that each has less than $c$ edges outgoing from $C$.

**Proof.** It is clear from the definition of a c-isolated biclique. \hfill \qed

Given a bipartite graph $G = (V_1 \cup V_2, E)$, we sort the vertices in $V_1$ such that $u < v \Rightarrow \text{deg}(u) < \text{deg}(v)$. Similarly, the vertices in $V_2$ are also sorted. The index of a vertex indicates its position (in its partition) in the sorted order. We also arrange...
the adjacent vertices of each $v$ in non-decreasing order, that is, we sort the adjacency list of each $v \in V$. This can be done in linear time using a sorting technique such as count sort or bucket sort. The index of a vertex indicates its position in the sorted order.

A c-isolated biclique $C(C_1 \cup C_2, E)$ may have more than one vertex with less than $c$ outgoing edges in each partition. The vertices in $C_1$ are sorted by their degrees. Therefore, the vertex with the minimum index in $C_1$ would have less than $c$ outgoing edges, we denote it by $pivot_1$, or in short by $p_1$. Similarly, the minimum index vertex in $C_2$ is said to be $pivot_2$, or in short $p_2$. In bipartite graphs, the neighborhood $N(v)$ of a vertex $v$ constitutes an independent set. Given a pivot $p$, let $N_+(v, p) = \{ u \in N(v) | u > p \} \cup \{ p \}$ and $N_-(v, p) = \{ u \in N(v) | u < p \}$.

Every c-isolated biclique consists of two pivots, we denote them as a pivot pair $(v_i, v_j)$, where $v_i \in V_1$ and $v_j \in V_2$. We can use this information to list all c-isolated maximal bicliques of a graph $G(V_1 \cup V_2, E)$. This can be done by generating for each pivot pair $(v_i, v_j)$ all c-isolated bicliques in the subgraph induced by the vertex set $N_+(v_i, v_j) \cup N_+(v_j, v_i)$. Next, we remove those c-isolated bicliques that are subsets of a c-isolated biclique with another pivot pair $(u_i, u_j)$, where $u_i \in N_-(v_j, v_i)$ and $u_j \in N_-(v_i, v_j)$.

Now we outline the basic idea of the algorithm. The algorithm consists of three stages:

**Trimming stage:** In this stage, the algorithm trims the input graph $G$ by deleting those vertices that can not be a part of any c-isolated biclique. During this stage, we also check the necessary conditions for pivots and terminate the trimming stage if some of the conditions are violated.

**Generation Stage:** During this stage, the algorithm generates all maximal bicliques by removing at most $c$ vertices from each partition.

**Screening Stage:** In this stage, the algorithm scans the bicliques generated in the
previous stage and selects those bicliques that satisfy the c-isolation condition. Finally, we test that each c-isolated biclique is maximal or not.

Now, we give the details of three stages of our algorithm.

4.3.1 Trimming Stage

This stage trims the input graph to reduce the running time of the algorithm. The algorithm builds a candidate set $C$, which is a superset of all c-isolated bicliques for the pivot pair $(v_i, v_j)$. The candidate set $C$ is the union of two independent sets, that is, $C = C_1 \cup C_2$. It is initialized with $N_+(v_i, v_j) \cup N_+(v_j, v_i)$, where $C_1 = N_+(v_j, v_i)$ and $C_2 = N_+(v_i, v_j)$. The algorithm deletes those vertices from $C$ that are not fit to be a part of a c-isolated biclique with the pivot $(v_i, v_j)$. Note that one can delete at most $c - N_-(v_j, v_i) - 1$ vertices from $C_1$ and $c - N_-(v_i, v_j) - 1$ vertices from $C_2$ and still get a c-isolated biclique.

The following tests are performed on $C$ during the trimming stage to reduce the size of $C$. Let $v \in C_1$, then

Test 1: $\deg(v) < |C_2| + c|C_1|$

Test 2: $v$ has less than $c|C_1|$ outgoing edges from $C$

Test 3: $v$ has at least $|C_2| - c$ adjacent vertices in $C$

Test 4: $C_1^h$ has less than $c(c + 1)|C_1^h|$ outgoing edges, where $C_1^h = \{x \in C_1| x \leq h\}$ and $1 \leq h \leq |C_1|$

Similarly, we define the above tests for the vertices in $C_2$.

The first test is inferred from the definition of a c-isolated biclique. This test is important as it takes $O(1)$ time for each vertex, and bound the maximum degree of
any vertex in $C$. The second test ensures the condition of c-isolation. The third test is required because we want to find isolated bicliques by deleting at most $c$ vertices from $C_1$ and $c$ vertices from $C_2$.

Next, we explain the need for the last test. Let $C^h_1$ is the set of $h$ vertices with the lowest indices in $C_1$, that is, $C^h_1 = \{x \in C_1 | x \leq h\}$. We can calculate an upper bound on the outgoing edges from $C$. Note that one can remove at most $(c - 1)$ vertices from each partition of the c-isolated biclique $C$ without violating the isolation condition. Furthermore, the second test ensures that each vertex in $C_1$ has less than $c|C_1|$ outgoing edges. The bound on the number of outgoing edges from $C_1$, is as follows:

$$|E(C_1, V - C)| < (c - 1)c|C_1| + c|C_1| < c(c + 1)|C_1|$$  \hspace{1cm} (4.1)

The following lemma provides the necessary condition for executing Test 4.

**Lemma 4.3.2.** Let $C(C_1 \cup C_2)$ is a c-isolated biclique, then $|E(C^h_1, V - C)| < c(c + 1)|C^h_1|$ and $|E(C^h_2, V - C)| < c(c + 1)|C^h_2|$.

**Proof.**

Suppose $|E(C^h_1, V - C)| > c(c + 1)|C^h_1|$ for any $h$. Let $v_h$ be the largest degree vertex in $|C^h_1|$, then the degree of $v_h$ is at least $c(c + 1)$. The vertices in $|C^h_1|$ are in sorted order. Therefore, all $v_j$ have at least $c(c + 1)$ outgoing edges from $|C^h_1|$, where $j > h$. Thus,

$$|E(C_1, V - C)| = |E(C^h_1, V - C)| + \sum_{i=h+1}^{C_1} (|E(v_i, V - C)|) \geq c(c + 1)|C^h_1| + c(c + 1)(|C_1| - |C^h_1|) = c(c + 1)|C_1|

This contradicts the isolation condition.

Similarly, we can show that $|E(C^h_2, V - C)| < c(c + 1)|C^h_2|$. \hfill $\Box$
4.3.2 Generation Stage

In this stage, the bicliques with the pivot pair \((v_i, v_j)\) are generated. Let \(C(C_1 \cup C_2)\) be a candidate set after the trimming stage. The number of vertices deleted from \(C\) in the trimming stage is given by \(r_1 = |N(v_j) - C_1|\) and \(r_2 = |N(v_i) - C_2|\). We can remove at most \(c - 1\) vertices from \(N(v_i)\) and the same is true for \(N(v_j)\), since otherwise we get too many outgoing edges. At the end of the trimming stage, the number of vertices that can still be removed are at most, \(k_1 = (c - 1) - r_1\) vertices from \(C_1\) and \(k_2 = (c - 1) - r_2\) from \(C_2\).

The basic idea for generating the bicliques is to use enumeration of bipartite vertex covers which can be carried out in linear time. The task is to list the bicliques with at least \(|C_1| - k_1\) vertices in the first bipartite partition and \(|C_2| - k_2\) vertices in the second. This is equivalent to enumerating the constraint bipartite vertex covers with parameters \(k_1\) and \(k_2\) in a bipartite complement graph of \(G\). To see how, consider the following definition of the bipartite complement graph.

**Definition 4.3.1.** The bipartite complement \(\bar{G}(V_1 \cup V_2, \bar{E})\) of a bipartite graph \(G(V_1 \cup V_2, E)\) is a graph that consists of the vertex set \(V_1 \cup V_2\) and the edge set \(\bar{E} = \{(v, w) | v \in V_1 \land w \in V_2; (v, w) \notin E\}\).

Figure 4.2 shows an example of a bipartite graph \(G\) and its complement graph \(\bar{G}\). Given the above definition of bipartite complement \(\bar{G}\) of \(G\), we can state the following:

**Lemma 4.3.3.** If \(S \subseteq G\) is a biclique, then \(G - S\) is a bipartite vertex cover of \(\bar{G}\)

**Proof.** This is clear from the definition of a bipartite complement. \(\square\)

From the above lemma, it is evident that in order to list all bicliques in the candidate set \(C\), we can generate all the bipartite vertex covers in the bipartite complement graph and then take complement of these covers to get the bicliques. At the end of
Figure 4.2: A bipartite graph $G$ and its bipartite complement graph $\overline{G}$

The trimming stage, there are at most $(c - 1) |C_1| + (c - 1) |C_2|$ edges missing from $C$ (because of the isolation factor $c$). This indicates that $C$ is a dense graph and one can construct its bipartite complement graph $\overline{C}$ in $O(E(C) + c|C|)$ time, where $E(C)$ is the number of edges in the subgraph induced by the vertices in $C$.

The generation of constraint bipartite vertex covers with constant parameters $k_1$ and $k_2$ can be performed in $O(1.4^{k_1+k_2}k_1k_2 + (k_1 + k_2)|V|)$ using the technique of fixed point tractability (FPT) (Fernau 2005; Fernau and Niedermeier 2001; Bai and Fernau 2012; Downey and Fellows: 1999). As $k_1 < c$ and $k_2 < c$, thus, we get covers in $O(1.4^{2c}c^2 + 2c|V|)$. As the complements of the bipartite covers can be computed in linear time, hence, the running time of the generation stage is linear.

4.3.3 Screening Stage

In the screening stage, we test each biclique generated in the previous stage, to find if it fulfills the condition of $c$-isolation and maximality. The $c$-isolation condition of
a biclique can be tested in linear time (in terms of the size of a biclique).

The bicliques that fulfill the condition of c-isolation are tested for the maximality condition. Let $D_{ij}$ be a list of c-isolated bicliques for the pivot pair $(v_i, v_j)$ and let $P$ be a list of pivot pairs, $P = \{(v_k, v_l) \mid v_k \in N_-(v_j, v_i) \cup v_i \land v_l \in (N_-(v_i, v_j) \cup v_j) \cap N(v_k)\}$. The task is to check $D' \in D_{ij}$ is maximal, this can be done by comparing it with the c-isolated bicliques generated by all pivots in $P - (v_i, v_j)$. This is sufficient because $v_i$ and $v_j$ can never occur in the bicliques with pivots $k > i$ and $l > j$.

As the c-isolation condition holds, therefore, $|N_-(v_i, v_j)| < c$ and $|N_-(v_j, v_i)| < c$. The number of bicliques for any pivot pair is bounded by $O(2^c)$, as the number of constraint bipartite vertex covers is $O(2^c)$ (Fernau and Niedermeier 2001; Bai and Fernau 2012; Downey and Fellows: 1999). Thus, we have to compare $D'$ with $O(c^2 2^c)$ bicliques. Now, we can give an estimate for the total time taken by the screening stage, that is, $O(c^2 2^c |E|)$.

**Enhanced Screening of c-isolated Bicliques**

In this section, we suggest an improvement in the screening technique of the bicliques to reduce the number of set comparisons (performed in the screening stage). Furthermore, this improvement also facilitates the development of a parallel algorithm that can significantly speed-up the generation process.

Consider the following lemma:

**Lemma 4.3.4.** Let $C$ and $C'$ be two c-isolated bicliques with pivot pairs $(v_i, v_j)$ and $(v_k, v_l)$, respectively. Then,

$C \subseteq C'$ iff $(v_k \in N_-(v_j, v_i)) \cap (v_l \in (N_-(v_i, v_j) \cup v_j) \cap N(v_k))$

where, $N(v_i) \supseteq C_2$ and $N(v_j) \supseteq C_1$.

**Proof.** It follows from the definition of a c-isolated biclique and pivot. $\square$
We check the maximality condition by looking for vertices in $N_-(v_i, v_j)$ that are connected to all vertices in $C_2$ and vertices $N_-(v_j, v_i)$ that are adjacent to all vertices in $C_1$. This step runs in $O(|C|)$ time and benefit us in two ways. First, it speeds up the algorithm by reducing the set comparisons. Second, it makes screening of the bicliques with a particular pivot pair independent from screening of the bicliques with another pivot pair. Thus, we can use parallel computing and simultaneously enumerate the bicliques with different pivot pairs.

**Property 4.3.1.** Let $C(C_1 \cup C_2)$ be a c-isolated biclique with a pivot pair $(v_i, v_j)$. If $C$ is not maximal. Then, $\exists D \subseteq (N(v_i) \cup N(v_j)) \setminus C$, such that $C \cup D$ is a c-isolated biclique.

Here, $D(D_1 \cup D_2)$ is a biclique and all vertices in $D_1$ and $D_2$ are adjacent to all vertices in $C_2$ and $C_1$, respectively.

Property 4.3.1 provides a basic technique for maximality testing with out using the set comparisons. For a c-isolated biclique $C$, we enumerate all maximal bicliques $D \subseteq (N(v_i) \cup N(v_j)) \setminus C$. If $D \cup C$ is c-isolated, then $C$ is definitely not maximal so we remove $C$ from the output list. However, if $D \cup C$ is not c-isolated, then we have to look for a subset $D'$ of $D$, such that $D' \cup C$ is c-isolated. We can do this by deleting highest degree vertices from $D$ till $D \cup C$ becomes c-isolated or $D$ is empty. If we get a c-isolated subset, then $C$ is not maximal, thus, we can delete $C$. In the other case, $C$ is maximal c-isolated biclique in $D \cup C$. We have to show this for all maximal bicliques $D \subseteq (N(v_i) \cup N(v_j)) \setminus C$, to declare $C$ as a maximal c-isolated biclique in $G$.

Now, we estimate the time required by the improved screening stage. For a pivot pair $(v_i, v_j)$, the number of enumerated c-isolated biclique is bounded by $O(2^c)$. In the improved screening idea, we perform two steps. First, for each enumerated c-isolated biclique $C$, we list all the maximal bicliques in $(N(v_i) \cup N(v_j)) \setminus C$, where $(N(v_i) \cup N(v_j)) \setminus C < c$. Gaspers has shown that a list of all the maximal bicliques can be generated in $O(c^{3c/3})$ time (Gaspers et al. 2012). Second, we check if the union of a maximal biclique with $C$ gives us a c-isolated biclique or not (as describe
above). This step takes \( O(c) \) times. Thus, the total time required in the improved screening process for \( |V| \) vertices is

\[
O(2^c)O(c3^{c/3})O(c)|V| = O(c^2 5.76^c |V|)
\]

The improved screening technique has reduced the exponential factor from \( 8^c \) to \( 5.76^c \).

Algorithm 4 \textit{c-IsolatedBicliques} gives the pseudo-code for our c-isolated generation scheme. It calls Algorithm 5 to perform the eight tests specified in the trimming stage. After this, it runs Algorithm 6 to generate all bicliques in the candidate set provided by the trimming stage. Finally, it calls Algorithm 7 to check the generated biclique for the condition of c-isolation and maximality.

\begin{algorithm}
\caption{c-IsolatedBicliques}
\textbf{Require:} Graph \( G(V_1 \cup V_2, E) \), isolation threshold \( c \)
1: Sort the vertices in \( V_1 \) by their degrees and renumber their indices
2: Sort the vertices in \( V_2 \) by their degrees and renumber their indices (starting with \( |V_1| \))
3: Sort the adjacency lists of vertices by their indices
4: \( L := 0 \)
5: \textbf{for} each \( v \in V_1 \) \textbf{do}
6: \quad Set \( N[v]_c \) to first \( c \) neighbors of \( v \)
7: \quad \textbf{for} each \( w \in N[v]_c \) \textbf{do}
8: \quad \quad \( C := \text{TrimmingStage}(G, v, w, c) \)
9: \quad \quad \( L_E := \text{GenerationStage}(G, C, v, w, c) \)
10: \quad \quad \( L_C := L_C \cup \text{ScreeningStage}(G, L_E, v, w, c) \)
11: \quad \textbf{end for}
12: \textbf{end for}
13: Print list of c-isolated bicliques \( L_C \)
\end{algorithm}
Algorithm 5 TrimmingStage

Require: Graph $G(V_1 \cup V_2, E)$, pivots $p_1, p_2$, isolation threshold $c$

{Build Candidate set $C(C_1 \cup C_2)$}

1: $C_1 := N_+(p_2, p_1)$
2: $C_2 := N_+(p_1, p_2)$
3: for each $v \in C_1$ do
4:  if $v$ fails test 1 then remove $v$ from $C_1$
5: end for
6: for each $v \in C_2$ do
7:  if $v$ fails symmetric test 1 then remove $v$ from $C_2$
8: end for
9: for each $v \in C_1$ do
10:  if $v$ fails test 2 or test 3 or test 4 then remove $v$ from $C_1$
11: end for
12: for each $v \in C_2$ do
13:  if $v$ fails symmetric test 2 or test 3 or test 4 then remove $v$ from $C_2$
14: end for
15: $n_1 := N_+(p_2, p_1) - |C_1|$
16: $n_2 := N_+(p_1, p_2) - |C_2|$
17: if $(n_1 > c - N_+(p_2, p_1) - 1)$ OR $(n_2 > c - N_+(p_1, p_2) - 1)$ then
18:  return Ø
19: else
20:  return $C(C_1 \cup C_2)$
21: end if

Algorithm 6 GenerationStage

Require: Graph $G(V_1 \cup V_2, E)$, candidate set $C(C_1 \cup C_2)$, pivots $p_1, p_2$, isolation threshold $c$

1: Compute the bipartite complement $\bar{G}$ of the graph induced by $C$
2: $k_1 := (c - 1) - |(N[p_2]\backslash C_1|$
3: $k_2 := (c - 1) - |(N[p_1]\backslash C_2|$
4: Compute all constraint bipartite vertex covers $L_V C$ of $\bar{G}$ with $k_1$ and $k_2$ as constraint parameters
5: Compute complement of covers in $L_V C$ and store in $L_C \{L_C$ is the list of $c$-isolated bicliques}
**Algorithm 7** \( \text{ScreeningStage} \)

**Require:** Graph \( G(V_1 \cup V_2, E) \), list of isolated bicliques \( L_C \), isolation threshold \( c \)

1: for each \( b \in L_C \) do
2: if \( b \) fails \( c \)-isolation condition then remove \( b \) from \( L_C \)
3: end for
4: for each \( b \in L_C \) do
5: if \( b \) fails maximality condition then remove \( b \) from \( L_C \)
6: end for

### 4.3.4 Time Complexity

In this section, we show that the running time of our algorithm is linear in terms of input, that is, graph size. The algorithm consists of three stages: trimming, generation and screening. First, we compute the running time of the trimming stage.

In this stage, the algorithm builds a candidate set \( C \) and perform the eight tests (described above) to reduce its size. For a pivot pair \((v_i, v_j)\), the construction of the candidate set \( C \) takes \( \deg(v_i) + \deg(v_j) \) time. Note that each vertex \( v_i \in V_1 \) forms a pivot pair with each \( v_j \in N_c(v_i) \), where \( N_c(v_i) \) contains \( c \) smallest indexed vertices in \( N(v_i) \). Alternatively, we can say that each \( v_j \in V_2 \) forms a pivot pair with every \( v_i \in N_c(v_j) \). The time to construct all candidate sets is:

\[
= \sum_{i=1}^{\lvert C_1 \rvert} \sum_{v_j \in N_c(v_i)} (\deg(v_i) + \deg(v_j)) \\
= \sum_{i=1}^{\lvert C_1 \rvert} \deg(v_i) + \sum_{i=1}^{\lvert C_1 \rvert} \sum_{v_j \in N_c(v_i)} \deg(v_j) \\
= c \sum_{i=1}^{\lvert C_1 \rvert} \deg(v_i) + \sum_{j=1}^{\lvert C_2 \rvert} \sum_{v_i \in N_c(v_j)} \deg(v_j) \\
= O(m)
\]

For a pivot pair \((v_i, v_j)\), test 1 takes \( O(\deg(v_i)) \) time and test 1 for the vertices in \( C_2 \) takes \( O(\deg(v_j)) \) time. Summing this over all pivot pairs, we get \( O(m) \) time for these
Next, we calculate the time needed to execute tests 2, 3, 4 for vertices in $C_1$ and the symmetric test for vertices in $C_2$. These tests appear to be time consuming as they scan the adjacency list of vertices in the candidate set. However, it can be shown that the execution of these tests for all pivot pairs does not take more than linear time. For a pivot pair $(v_i, v_j)$, let $C(v_i, v_j)$ consists of the vertices that have passed test 1, and let $S(v_i, v_j) \subseteq C(v_i, v_j)$ be the set of vertices on which remaining tests are applied. We can divide $S$ into two sets; $S_p(v_i, v_j)$, the set of vertices that passed the trimming stage and $S_f(v_i, v_j)$, the set of vertices that do not. It is easy to see that $|S_f(v_i, v_j)| \leq 2c - 2$, since, we quit the trimming stage if more than $c$ vertices from any partition of $G$ fail the tests.

In the trimming stage of pivot pair $(v_i, v_j)$, the adjacency list of vertices in $S(v_i, v_j)$ is scanned constant number of times. As $S(v_i, v_j)$ is a bipartite subset, so it is union of two independent sets $S_1(v_i, v_j) \subseteq V_1$ and $S_2(v_i, v_j) \subseteq V_2$. Tests 2, 3, 4 are applied on the vertices in $S_1(v_i, v_j)$ and the symmetric tests are applied on the vertices in $S_2(v_i, v_j)$. The total time taken by the trimming stage can be expressed as follows:

$$T = \sum_{i=1}^{|C_1|} \sum_{v_j \in N_c(v_i)} \sum_{v_k \in S_1(v_i, v_j)} \deg(v_k) + \sum_{i=1}^{|C_1|} \sum_{v_j \in N_c(v_i)} \sum_{v_k \in S_2(v_i, v_j)} \deg(v_k)$$

$$T = \sum_{i=1}^{|C_1|} \sum_{v_j \in N_c(v_i)} \sum_{v_k \in S(v_i, v_j)} \deg(v_k)$$ (4.2)

**Lemma 4.3.5.** Let $(v_i, v_j)$ be a pivot pair, and suppose that vertices $v_a$ and $v_b \in S_1(v_i, v_j)$. Then $\deg(v_a) \leq \deg(v_b) + c \deg(v_j)$ and $\deg(v_b) \leq \deg(v_a) + c \deg(v_j)$

**Proof.** It is evident as both $v_a$ and $v_b$ have passed test 1. $\square$

Similarly, we can bound the degree of vertices.
Lemma 4.3.6. Given a pivot pair \((v_i, v_j)\) and vertices \(v_a, v_b \in S_2(v_i, v_j)\), we have \(\text{deg}(v_a) \leq \text{deg}(v_b) + c \text{deg}(v_i)\) and \(\text{deg}(v_b) \leq \text{deg}(v_a) + c \text{deg}(v_i)\)

We can express the running time of the trimming stage in terms of \(T_p\), the time taken by the vertices that have passed the trimming stage. This is possible because the vertices that have failed the trimming stage for a pivot \((v_i, v_j)\) are \(|S_f(v_i, v_j)| \leq 2c - 2\).

\[
T = T_p + T_f
\]

\[
T = \sum_{i=1}^{|C_1|} \sum_{v_j \in N_c(v_i)} \sum_{v_k \in S_f(v_i, v_j)} \text{deg}(v_k) + \sum_{i=1}^{|C_1|} \sum_{v_j \in N_c(v_i)} \sum_{v_k \in S_f(v_i, v_j)} \text{deg}(v_k) \quad (4.3)
\]

Here, \(T_p\) and \(T_f\) denote the time taken by the vertices that have passed and failed the trimming stage, respectively. The time taken by the failed vertices in the trimming stage is:

\[
T_f = \sum_{i=1}^{|C_1|} \sum_{v_j \in N_c(v_i)} \sum_{v_k \in S_f(v_i, v_j)} \text{deg}(v_k)
\]

\[
T_f = \sum_{i=1}^{|C_1|} \sum_{v_j \in N_c(v_i)} \sum_{v_k \in S_f(v_i, v_j)} \text{deg}(v_k) + \sum_{i=1}^{|C_1|} \sum_{v_j \in N_c(v_i)} \sum_{v_k \in S_f(v_i, v_j)} \text{deg}(v_k) \quad (4.4)
\]

\(S_f(v_i, v_j)\) is a set of vertices of a bipartite graph, therefore, it is a union of two independent sets \(S_{f1}(v_i, v_j)\) and \(S_{f2}(v_i, v_j)\). From Lemmas 4.3.5 and 4.3.6, and the fact that \(|S_{f1}(v_i, v_j)| < c\), we get

\[
\sum_{v_k \in S_{f1}(v_i, v_j)} \text{deg}(v_k) < c(\text{deg}(v_i) + c(\text{deg}(v_j))
\]

Here, \(v_j\) is a pivot vertex in \(C_2\) and no vertex can occur in more than \(c\) pivot pairs. Thus, we have
\[
\sum_{i=1}^{|C_1|} \sum_{v_j \in N_i(v_i), v_k \in S_f^1(v_i, v_j)} \deg(v_k) < c \sum_{i=1}^{|C_1|} (\deg(v_i) + \deg(v_j)) \\
< c^2 \sum_{i=1}^{|C_1|} \deg(v_i) + c^3 \sum_{j=1}^{|C_2|} \deg(v_j)
\]

Similarly, we can derive an estimate for vertices in \(S_f^2(v_i, v_j)\). Thus, \(T_f\) becomes

\[
T_f < c^2 \sum_{i=1}^{|C_1|} \deg(v_i) + c^3 \sum_{j=1}^{|C_2|} \deg(v_j) \\
< 2c^3(|\sum_{i=1}^{|C_1|} \deg(v_i) + \sum_{j=1}^{|C_2|} \deg(v_j)|) \\
T_f < 2c^3|E|
\]

\(|E|\) is the number of edges in \(G\). Therefore, the total time consumed in the trimming stage is

\[
T < T_p + 2c^3|E| \tag{4.5}
\]

In order to compute \(T_p\), we have to estimate the number of outgoing edges from \(S_p(v_i, v_j)\). As \(S_p(v_i, v_j)\) is a set of vertices of a bipartite graph, therefore it can be partitioned into two independent sets, \(S_p(v_i, v_j) = S_{p1}(v_i, v_j) \cap S_{p2}(v_i, v_j)\).

**Lemma 4.3.7.** \(|E(S_{p1}(v_i, v_j), V - C(v_i, v_j))| < 2(c + 1) \max\{|S_{p1}(v_i, v_j)|, c\}\)

**Proof.** The vertices in \(|S_p(v_i, v_j)|\) have passed the test 4, So

\[
|E((S_{p1}(v_i, v_j)), V - C(v_i, v_j))| < c(c + 1)(|S_{p1}(v_i, v_j)|) \\
< c(c + 1)|C_1|
\]
The above inequality holds if $|C_1| < 2c$. Let $|C_1| \geq 2c$, then $|C_1| < 2|S_{p1}(v_i, v_j)|$. If this is not true then we get a contradiction as follows:

$$|S_{p1}(v_i, v_j)| \leq |C_1| - |C_1|/2 \leq |C_1| - c$$

So, $c(c+1)|C_1| < 2c(c+1)|S_{p1}(v_i, v_j)|$, and the desired inequality holds in this case.

Similarly, we can derive a bound on the number of outgoing edges from vertices in $|V_2|$.

**Lemma 4.3.8.** $|E(S_{p2}(v_i, v_j), V - C(v_i, v_j))| < 2c(c + 1) \max\{|S_{p2}(v_i, v_j)|, c\}$

For each $v_k \in S_p(v_i, v_j)$, its adjacency list is scanned constant number of times. We want to estimate how many times a vertex $v$ can occur in some $S_p(v_i, v_j)$. For this, we define a reverse function $S^{-1}_p(v) = \{(v_i, v_j) | v \in S_p(v_i, v_j)\}$. The vertex $v_i$ can form pivot pairs with at most $c$ vertices from $V_2$, so the reverse function for $v$ can be expressed in terms of the vertices in $V_1$. Let $S^{-1}_{p1}(v) = \{v_x | v \in S_{p1}(v_i, v_x)\}$, where $v_x \in V_2$ and $v_x$ can take at most $c$ values from $V_2$. Thus, $|S^{-1}_p(v)| \leq c |S^{-1}_{p1}(v)|$. Similarly, we can represent the reverse function for $v$ using only vertices in $V_2$, $|S^{-1}_p(v)| \leq c |S^{-1}_{p2}(v)|$.

$T_p$, in terms of reverse function can be expressed as:

$$T_p = \sum_{i=1}^{|C_1|} |S^{-1}_p(v_i)| \deg(v_i) + \sum_{j=1}^{|C_2|} |S^{-1}_p(v_j)| \deg(v_j)$$

$$T_p = \sum_{k=1}^{|C_1|+|C_2|} |S^{-1}_p(v_k)| \deg(v_k)$$

(4.6)

$$T_p \leq c \sum_{k=1}^{|C_1|+|C_2|} |S^{-1}_{p1}(v_k)| \deg(v_k)$$

(4.7)

$$T_p \leq c \sum_{k=1}^{|C_1|+|C_2|} |S^{-1}_{p2}(v_k)| \deg(v_k)$$

(4.8)
Lemma 4.3.9. For a pivot \((v_i, v_j)\)

\[\sum_{v_k \in S_{p1}(v_i, v_j)} |S_p^{-1}(v_k)| < c^2(4c + 3)\max\{|S_{p1}(v_i, v_j)|, c\}\]

Proof. For any \(v_k \in S_{p1}(v_i, v_j)\)

\[
|S_p^{-1}(v_k)| \leq c |S_p^{-1}(v_k)| \\
|S_{p1}^{-1}(v_k)| \leq |S_{p1}^{-1}(v_k) \cap V| \\
\leq |S_{p1}^{-1}(v_k) \cap (V - C(v_i, v_j) + C(v_i, v_j))| \\
\leq |S_{p1}^{-1}(v_k) \cap (V - C(v_i, v_j))| + |S_{p1}^{-1}(v_k) \cap C(v_i, v_j)| \\
\leq |S_{p1}^{-1}(v_k) \cap (C(v_i, v_j) - S(v_i, v_j)) + S(v_i, v_j)| \\
+ |S_{p1}^{-1}(v_k) \cap (V - C(v_i, v_j))| \\
\leq |S_{p1}^{-1}(v_k) \cap (C(v_i, v_j) - S(v_i, v_j))| + \\
|S_{p1}^{-1}(v_k) \cap (S_p(v_i, v_j) + S_f(v_i, v_j))| + |S_{p1}^{-1}(v_k) \cap (V - C(v_i, v_j))| \\
\leq |S_{p1}^{-1}(v_k) \cap (C(v_i, v_j) - S(v_i, v_j))| + \\
|S_{p1}^{-1}(v_k) \cap (S_p(v_i, v_j)) + |S_{p1}^{-1}(v_k) \cap S_f(v_i, v_j))| + \\
|S_{p1}^{-1}(v_k) \cap (V - C(v_i, v_j))|
\]

From above it is clear that we can express \(|S_p^{-1}(v_k)|\) in terms of four quantities. Next, we calculate the value of each quantity.

First, consider the quantity \(|S_{p1}^{-1}(v_k) \cap (C(v_i, v_j) - S(v_i, v_j))|\). As \(S_{p1}^{-1}(v_k) = \{v_i | v \in S(v_i, v_x)\}\), therefore we are only considering vertices in \(C_1\) in this quantity. The
index of any vertex \( v_k \in S_{p1}(v_i, v_j) \) is less than the index of all the vertices \( v_p \in (C(v_i, v_j) - S(v_i, v_j)) \). The vertex \( v_i \) has the smallest index in \( C_1 \). This indicates that any \( v_p \) cannot be in a pivot pair of a c-isolated biclique including any \( v_k \in S_{p1}(v_i, v_j) \).

Thus, \( |S_{p1}^{-1}(v_k) \cap (C(v_i, v_j) - S(v_i, v_j))| = 0. \)

Next, consider the value of \( |S_{p1}^{-1}(v_k) \cap S_p(v_i, v_j)| \). This is same as \( |S_{p1}^{-1}(v_k) \cap S_{p1}(v_i, v_j)| \), as \( S_{p1}^{-1}(v_k) \) consists only of vertices in the first partition of \( G \). Furthermore, note that only the vertices with smallest \( 2c \) indexes in \( S_{p1}(v_i, v_j) \) can form pivot pairs with the vertices in \( S_{p2}(v_i, v_j) \). This is because the vertices in \( S_{p1}(v_i, v_j) \) have at least \( C_2 - c \) neighbors in \( S_{p2}(v_i, v_j) \) and no c-isolated biclique can have a vertex with index less than the pivot. Thus, we can conclude that \( |S_{p1}^{-1}(v_k) \cap S_p(v_i, v_j)| \leq 2c. \)

Next, we have \( |S_{p1}^{-1}(v_k) \cap S_f(v_i, v_j)| \leq c \) as \( |S_{f1}(v_i, v_j)| \leq c - 1. \)

Finally, we get the bound on last value \( |S_{p1}^{-1}(v_k) \cap (V - C(v_i, v_j))| \) from Lemma 4.3.7.

Now, we can provide a bound on \( \sum_{v_k \in S_{p1}(v_i, v_j)} |S_{p1}^{-1}(v_k)| \)

\[
\sum_{v_k \in S_{p1}(v_i, v_j)} |S_{p1}^{-1}(v_k)| < (2c + c)|S_{p1}(v_i, v_j)| + c(2c + 1)max\{|S_{p1}(v_i, v_j)|, c\}
\]

\[
= c(4c + 3)max\{|S_{p1}(v_i, v_j)|, c\}
\]

\[
\sum_{v_k \in S_{p1}(v_i, v_j)} |S_{p1}^{-1}(v_k)| < c^2(4c + 3)max\{|S_{p1}(v_i, v_j)|, c\}
\]

Following the line of the proof of the above Lemma we derive that

**Lemma 4.3.10.** For a pivot \( (v_i, v_j) \)

\[
\sum_{v_k \in S_{p2}(v_i, v_j)} |S_{p2}^{-1}(v_k)| < c^2(4c + 3)max\{|S_{p2}(v_i, v_j)|, c\}
\]

The total time taken by vertices that passed the trimming stage, \( T_p \) can be expressed as:
\[ T_p = \sum_{i=1}^{\lfloor C_1 \rfloor} \sum_{v_j \in N_c(v_i)} \sum_{v_k \in S_p(v_i,v_j)} \deg(v_k) \]
\[ T_p = \sum_{i=1}^{\lfloor C_1 \rfloor} \sum_{v_j \in N_c(v_i)} \left( \sum_{v_k \in S_p(v_i,v_j)} \deg(v_k) \right) + \sum_{v_k \in S_{p2}(v_i,v_j)} \deg(v_k) \]

Thus, \( T_p = T_{p1} + T_{p2} \), where \( T_{p1} \) and \( T_{p2} \) calculates the time taken by vertices in the first and second partition of \( G \) for each pivot pair, respectively. In order to estimate \( T_{p2} \), we divide the set of all pivot pairs into two parts depending on the size of the \( S_{p2}(v_i, v_j) \).

Let \( PP_S = \{(v_i, v_j) \mid |S_{p2}(v_i, v_j)| \leq c \} \) and let \( PP_L = \{(v_i, v_j) \mid |S_{p2}(v_i, v_j)| \geq c \} \). We define a set \( L = \{v_k \mid |S^{-1}(v_k)| < 2c^2(4c + 3) \} \).

**Lemma 4.3.11.** For any \( v_i \in PP_L \), at least half of the elements in \( S_{p2}(v_i, v_j) \) also exist in \( L \).

**Proof.** If the above statement is not true, then more than half of the elements in \( S_{p2}(v_i, v_j) \) will have value greater then \( 2c^2(4c + 3) \). This will contradict Lemma 4.3.9.

We can represent \( T_{p2} \) as the time taken by all pivots \( p_i \) in \( PP_L \) and \( PP_S \).

\[ T_{p2} = \sum_{p_i \in PP_L} \sum_{v_k \in S_{p2}(p_i)} \deg(v_k) + \sum_{p_i \in PP_S} \sum_{v_k \in S_{p2}(p_i)} \deg(v_k) \]

By Lemma 4.3.5 and the fact that for each vertex in \( PP_S \), \( |S_{p2}(v_i, v_j)| < c \), the time needed by all pivots in \( PP_S \) becomes

\[ \sum_{p_i \in PP_S} \sum_{v_k \in S_{p2}(p_i)} \deg(v_k) \]
\[ < \sum_{p_i \in PP_S} c(\deg(v_i) + c \deg(v_j)) \]

Note that \( |PP_S| < c|C_1| \), as each vertex in \( C_1 \) can form a pivot pair with at most \( c \) vertices from \( C_2 \). Furthermore, the vertex \( v_j \) is a pivot and can occur in at most \( c \) pivot pairs. Therefore, the time taken by all the pivots in \( PP_S \) is
Next, we compute the time needed by all pivots in $PP_L$. 

$$
\sum_{p_i \in PP_L} \sum_{v_k \in S_{p2}(p_i)} \deg(v_k)
= \sum_{p_i \in PP_L} \left( \sum_{v_k \in S_{p2}(p_i) \cap L} \deg(v_k) + \sum_{v_k' \in S_{p2}(p_i) \setminus L} \deg(v_k') \right)
$$

By Lemma 4.3.11 at least half of the elements in $S_{p2}(p_i)$ are in $L$. Furthermore, from Lemma 4.3.5 it is clear that we can represent each vertex in $S_{p2}(p_i) \setminus L$ in terms of a vertex in $S_{p2}(p_i) \cap L$ and pivot $v_j$. A pivot $v_j$ can occur in at most $c$ pivot pairs. Thus, the above inequality can be represented as follows:

$$
\leq \sum_{p_i \in PP_L} \sum_{v_k \in L} (2\deg(v_k) + c\deg(v_j))
$$

we can express above equality as inverse function $|S^{-1}v_k|$

$$
\leq 2 \sum_{v_k \in L} |S^{-1}v_k| \deg(v_k) + c \sum_{v_k \in L} |S^{-1}v_k| \sum_{p_i \in PP_L} \deg(v_j)
$$

Using Lemma 4.3.9 and the fact that $v_j$ can occur in at most $c$ pivots

$$
\leq 2 \sum_{v_k \in L} (2c^2(4c + 3))\deg(v_k) + c(2c^2(4c + 3)) \sum_{p_i \in PP_L} \deg(v_j)
\leq 4c^3(4c + 3)|E| + c^2(2c^2(4c + 3))|E|
\leq 4c^3(2c^2 + 4c + 3)|E|
$$

Now, we can give a bound on $T_{p2}$
Similarly, we can compute value for $T_{p1}$, Thus, $T_p$ is

$$T_p \leq 2(2c^3|E| + 4c^3(2c^2 + 4c + 3)|E|)$$

The total time in the trimming stage is

$$T = T_f + T_p$$

$$T \leq 2c^3|E| + 2(2c^3|E| + 4c^3(2c^2 + 4c + 3)|E|) = O(c^5|E|)$$

**Theorem 1.** Algorithm 4 $c$-IsolatedBicliques generates all $c$-isolated bicliques in time $O(c^75.76^c|E|)$, for constant $c$.

**Proof.** Algorithm $c$-IsolatedBicliques consists of three stages: trimming stage, generation stage and screening stage. We have computed the time $T$ for the trimming stage. The generation stage produces all bipartite vertex covers for a given pivot $(v_i, v_j)$ in $O(1.4^{2c}c^2 + 2c|S_p(v_i, v_j)|)$ time using FPT approach, and then convert them to bicliques in $O(|S_p(v_i, v_j)|)$. This stage operates on the vertices in $C(v_i, v_j)$ that have passed the trimming stage. Therefore, we consider only the set $S_p(v_i, v_j)$. The improved screening stage, described in the previous section takes $O(c^25.76^c|S_p(v_i, v_j)|)$ for each pivot pair $(v_i, v_j)$. The total time for the generation and the screening stage for all pivots is

$$E_g \leq \sum_{i=1}^{C_1} \sum_{v_j \in N_c(v_i)} O(1.4^{2c}c^2 + 2c|S_p(v_i, v_j)|) + O(c^25.76^c|S_p(v_i, v_j)|)$$

$$< O(c^25.76^c \sum_{i=1}^{C_1} \sum_{v_j \in N_c(v_i)} |S_p(v_i, v_j)|)$$
The inequality \[ \sum_{i=1}^{C_1} \sum_{v_j \in N_c(v_i)} |s_p(v_i, v_j)| < T_p \] holds. Thus, \( E_S < O(c^{2.576c}T_p) \). The total computation time of the algorithm is:

\[
I_B < T + O(c^{2.576c}T_p) \\
< O(c^5|E|) + O(c^2c^{5.76c}|E|) \\
< O(c^7c^{5.76c}|E|)
\]

### 4.4 Computational Experiments

In this section, we present results of the computational experiments that are conducted on random bipartite graphs and real-world data sets. The results are quite promising and show that linear cost is incurred for generating all c-isolated biclique in a given graph. The experiments are run on Windows 7 environment, using Intel(R) Core(TM)i7 CPU (1.6GHz) with 6GB RAM. The algorithm is implemented in C++ language using the Boost graph library (Boost 2010). The results of the experiments are plotted in log scale.

The output generated by our algorithm consists of c-isolated bicliques. We partitioned it into three parts: star, small and large. The star partition consists of the c-isolated bicliques that form a star graph. The bicliques that have at least three or more vertices in each partition are considered large and the rest are considered small. The large c-isolated bicliques are important, because usually in the real-world problems we are interested in communities with the large number of members. However, in some cases such as web-structure mining, we are interested in isolated stars to extract information about the index structure and detect link farm spams (Uno et al. 2008).

The choice of the isolation factor \( c \) plays an important role and depends on the domain of the problem under consideration. The small values of \( c \) help extract strong communities (the communities that have few links between members and non-members). On
the other hand, the large $c$ is also useful in some scenarios. It allows user to restrict the number of connections with the outside world. This can reduce the output set of million of nodes to a manageable size.

### 4.4.1 Scale Free Random Graphs

We constructed random scale-free graphs to evaluate the performance of our algorithm. The scale-free model is used, because it captures the properties of web and social networks. The classical models for constructing random graphs, such as small world and Gilbert’s $G(n, p)$ model, have a degree distribution that concentrates around the mean. However, many real world networks exhibit a power law degree distribution. Consider an example of world wide web (WWW). In a graph model of WWW, a vertex represents a page and its degree indicates the number of pages it refers. The probability that the popularity of a web-page will increase greatly depends on the existing popularity of the web-page. Thus, the degree distribution in a graph of WWW follows a power law function. In addition to world wide web, the power law distribution is also observed in real world networks such as actor collaboration network (Barabasi and Albert 1999), citation networks (Redner 1998), telephone call graphs, and metabolic networks (Jeong et al. 2001).

Various methods were proposed in the literature to generate random scale-free graphs with power law degree distribution, one such method is preferential attachment (Barabasi and Albert 1999). We constructed a random scale-free bipartite graph using the bipartite preferential attachment algorithm (Batageljy and Brandes 2005). The generated graph consists of 2000 nodes (1000 in each partition), 19033 edges and the minimum degree of the graph is 10. We executed our algorithm on this graph for different values of $c$, the isolation factor. The results are shown in Table 4.1. The table gives the number of generated $c$-isolated bicliques and the time taken to generate all these bicliques for various values of $c$. It also shows how many of the generated $c$-bicliques are star, small or large.
The results show that the isolated bicliques exist in a scale-free graph. This indicates that the isolated communities do occur in real life situations and thus, can be extracted using our algorithm. Furthermore, it is evident from Figure 4.4 that the number of c-isolated bicliques increases with the increase in the isolation factor. For higher values of $c$, we get large number of c-isolated bicliques. However, the growth rate of small-sized bicliques is slow as compared to large ones. It is interesting to note that we do not discover many star bicliques in the scale-free random networks.

<table>
<thead>
<tr>
<th>$c$ (isolation factor)</th>
<th>c-Isolated bicliques</th>
<th>Star</th>
<th>Small</th>
<th>Large</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>969</td>
<td>79</td>
<td>890</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>40</td>
<td>5972</td>
<td>1107</td>
<td>4837</td>
<td>28</td>
<td>21</td>
</tr>
<tr>
<td>60</td>
<td>9672</td>
<td>1637</td>
<td>7829</td>
<td>206</td>
<td>31</td>
</tr>
<tr>
<td>80</td>
<td>13498</td>
<td>1849</td>
<td>10932</td>
<td>717</td>
<td>46</td>
</tr>
<tr>
<td>100</td>
<td>17583</td>
<td>1964</td>
<td>13920</td>
<td>1699</td>
<td>45</td>
</tr>
<tr>
<td>120</td>
<td>22378</td>
<td>1988</td>
<td>17285</td>
<td>3105</td>
<td>48</td>
</tr>
<tr>
<td>140</td>
<td>27084</td>
<td>1991</td>
<td>20384</td>
<td>4719</td>
<td>51</td>
</tr>
<tr>
<td>160</td>
<td>32874</td>
<td>1991</td>
<td>23303</td>
<td>7580</td>
<td>57</td>
</tr>
<tr>
<td>180</td>
<td>41072</td>
<td>1991</td>
<td>25684</td>
<td>13397</td>
<td>58</td>
</tr>
<tr>
<td>200</td>
<td>50127</td>
<td>1991</td>
<td>26888</td>
<td>21248</td>
<td>67</td>
</tr>
</tbody>
</table>

Table 4.1: Experimental results for different values of $c$ in a scale-free graph

4.4.2 Citation Network

In this section, we present the results of our experiment on a real-world dataset, the citation network. The dataset for this experiment is obtained from Stanford Large Network Dataset Collection (SNAP 2012). It consists of a citation graph of “Arxiv HEP-PH” (high energy physics phenomenology), where the nodes are research papers and the edges are citations. This graph includes 34,546 papers and 421,578 citations in the period from January 1993 to April 2003. In this graph, there is a directed edge from the paper $x$ to $y$ only if $x$ cites $y$ and both papers $x$ and $y$ belong to the group “Arxiv HEP-PH”. For our experiment, we convert this graph into a bipartite graph $G(V_1 \cup V_2, E)$ as follows:
Figure 4.3: The time required to generate all c-isolated bicliques in a scale-free graph

Figure 4.4: Distribution of c-isolated bicliques in a scale-free graph
Figure 4.5: Conversion of a directed graph to a bipartite graph $G(V_1 \cup V_2, E)$

- $V_1$ contains papers that cite a paper.
- $V_2$ contains papers that are cited by at least one paper (in the given graph).
- An edge represents a citation. That is, the edge $e(v_i, v_j)$ indicates paper $v_i \in V_1$ cites paper $v_j \in V_2$.

Each edge in $G(V_1 \cup V_2, E)$ is between a vertex in $V_1$ and a vertex in $V_2$. There are 32158 vertices in $V_1$, 28230 vertices in $V_2$ and 421578 edges. The edge density of the graph is 0.000464385. Figure 4.5 shows an example of converting a directed general graph to an undirected bipartite graph. In the converted bipartite graph $G$, some papers are duplicated and they exist in both $V_1$ and $V_2$. However, the edges are not duplicated in $G$. The number of edges in the original graph is equal to the number of edges in the converted bipartite graph. Thus, the running time of the algorithm is not effected.
We run our algorithm on a bipartite graph $G$ with different values of $c$, the isolation factor. The results of this experiment are shown in Table 4.3 and Figures 4.6 and 4.7. The table shows the number of star, small and large $c$-isolated bicliques found in this dataset for different isolation factors. It is observed that the growth rate of the large bicliques is higher as compared to star and small bicliques. Many bicliques are generated even for small values of $c$, this indicates that communities of scientists are coherent. Furthermore, it also shows that scientists in high energy physics phenomenology, usually works in closely netted groups and prefer to cite papers that are published by their group.

We examine some of the discovered $c$-isolated bicliques. It is observed that a $c$-isolated biclique represents related papers. Therefore, they can detect similar papers that may not refer one another. The list of generated bicliques can be used to find isolated research communities. The authors in an isolated communities mostly cite papers authored by the members of their community. Apart from this, carefully scrutinizing the found $c$-isolated bicliques can also aid in detecting plagiarism cases.

Consider an example of a $c$-isolated biclique that we discovered in “Arxiv HEP-PH” citation network with $c = 5$. The first partition of the biclique consists of paper-ids: 9807365, 9910375, 7127, 110005 and the second partition consists of papers with ids: 9806275, 9710445, 9805306, 9707307. When we examine this isolated biclique, we learn that the papers are closely related and have many common authors. The detail of the papers are given in Table 4.2.

4.4.3 Movie Network

We perform experiments on a movie network to discover isolated actor-director communities in a particular time span. By isolated actor-director community, we mean a group of actors who mostly work with a particular set of directors and vise versa.

We acquire the dataset for this experiment from IMDB website (IMDB 2011). It
Table 4.2: Example of a 5-isolated biclique in a citation network

<table>
<thead>
<tr>
<th>c (isolation factor)</th>
<th>c-Isolated bicliques</th>
<th>Star</th>
<th>Small</th>
<th>Large</th>
<th>Time(sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>221</td>
<td>216</td>
<td>5</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1746</td>
<td>1558</td>
<td>178</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>3181</td>
<td>2506</td>
<td>597</td>
<td>78</td>
<td>17</td>
</tr>
<tr>
<td>7</td>
<td>5072</td>
<td>3507</td>
<td>1328</td>
<td>237</td>
<td>48</td>
</tr>
<tr>
<td>9</td>
<td>7189</td>
<td>4497</td>
<td>2197</td>
<td>495</td>
<td>77</td>
</tr>
<tr>
<td>11</td>
<td>10048</td>
<td>5670</td>
<td>3390</td>
<td>988</td>
<td>128</td>
</tr>
<tr>
<td>13</td>
<td>13669</td>
<td>6989</td>
<td>4925</td>
<td>1755</td>
<td>153</td>
</tr>
<tr>
<td>15</td>
<td>18029</td>
<td>8381</td>
<td>6899</td>
<td>2749</td>
<td>250</td>
</tr>
</tbody>
</table>

Table 4.3: Experimental results for different values of c in a citation network

This dataset consists of 2580 directors and 16,608 actors. This dataset is for the time period 1923 – 2000 and contains the information for 15,000 movies. A director-actor connection indicates that the director has worked with the actor in some movie. We build a bipartite graph for this dataset with 2580 nodes (for directors) in the first partition, 16608 nodes (for actors) in the second and 41,075 edges. The results of this experiment are shown in the Table 4.4 and Figure 4.8.

For this dataset, we executed the experiments with large values of c, as actors and directors work in a much larger group as compared to researchers. For instance, a director casts many actors in his movie, while a paper usually consists of 1 – 4 authors. This experiment can be used to detect groups of directors who prefer to work with the same set of actors (with in a given time span). This can be quite useful information.
Figure 4.6: The time required to generate all c-isolated bicliques in a citation network for predicting the social circle of movie stars. Figure 4.9 shows that the algorithm takes linear time to generate the entire list of c-isolated bicliques in a movie network.

Table 4.4: Experimental results for different values of c in a movie network
4.5 Conclusion

In this chapter, we develop an efficient algorithm for enumerating all c-isolated bicliques in a given graph $G$. We provide a comprehensive mathematical proof to show that our algorithm takes linear time to list all c-isolated bicliques. Lastly, we evaluate the performance of our algorithm on synthetic graphs and real-world problems.
Figure 4.8: Distribution of $c$-isolated bicliques in a movie network

Figure 4.9: The time required to generate all $c$-isolated bicliques in a movie network
Chapter 5

Generation Algorithms for Combinatorial Path

5.1 Introduction

This chapter deals with the generation of all paths in a clique in minimal change order. The minimal change order speeds up the generation by listing objects such that successive elements differ in a pre-specified minimal way. The order in which two consecutive objects have distance two is called revolving door order (Nijenhuis and Wilf 1978; Knuth 2005). In the last few decades, many algorithms have been devised to generate combinatorial objects in minimal change order (Korsh and Lafollette 2006; Baril and Vajnovszki 2006). Malcolm Simth (Knuth 2005) proposed a revolving door algorithm for listing all spanning trees in a graph. Recently, Korsh (Korsh and Lafollette 2006) gave a gray code scheme for generating rooted trees. However, there exist no published algorithm in the literature that generates all paths in revolving door order. Harada proposed a method to enumerate all hamiltonian circuits in a complete graph, $K_n$. He shows that the hamiltonian paths can be obtained from the hamiltonian circuits but his paths are not in revolving door order (Harada 1971).

In this chapter, we develop efficient algorithms for generating all spanning paths.
and all paths in a clique in revolving door order. The algorithms are optimal in the sense that they use linear space and operate in constant amortized time (CAT). Our algorithms are based on a recursive structure. Generally, recursive generation algorithms are preferred because they are elegant, flexible and provide insight into the combinatorial structure. We also develop iterative version of our algorithms using counters.

5.2 Preliminaries

Given a simple graph $G(V, E)$. A subset $K \subseteq V$ is a clique if \( \forall u, v \in K : (u, v) \in E \).

In case $K$ is a proper subset of $V$, then it is also known as a complete graph, denoted by $K_n$. We index the vertex set of $K_n$ from 1 to $n$, that is, $V = \{1, \ldots, n\}$. A path $P$ in $K_n$ is a set of edges $\{\{v_0, v_1\}, \ldots, \{v_{k-2}, v_{k-1}\}\}$. For simplicity, a path is represented as a list of vertices. Thus, a path of length $k$ is given by $P = (v_0, \ldots, v_{k-1})$. The reverse of a path is denoted by $\text{reverse}(P) = (v_k, \ldots, v_0)$. To show that two paths, $P_1$ and $P_2$, are different we must show that $P_1 \neq P_2$ and $P_1 \neq \text{reverse}(P_2)$.

![Figure 5.1: A complete graph on 5 vertices](image)

**Definition 5.2.1.** Revolving Door Order is a minimal change order in which two consecutive objects have distance two. That is, they differ in exactly two positions.

Consider two rooms A and B connected by a revolving door that works only if occupied
by two persons, one from each room. The door acts to swap two persons. The problem is to go through all possible combinations of persons in the first room so that each such combination occurs exactly once. The name *revolving door* is first used by Nijenhuis and Wilf (Nijenhuis and Wilf 1978).

The paths in a graph are of different lengths. Therefore, we relax the definition of a revolving door. We define modified revolving door order as a minimal change order in which two consecutive objects have distance one or two. That is, they differ by an addition, deletion or swap of an element.

The following algorithms are developed in this chapter:

**All Spanning Paths:** It generates all spanning paths in $K_n$ in revolving door order.

**All Path Sets:** It generates all paths sets in $K_n$ in revolving door order.

**All $K_n$ Paths:** It generates all paths in $K_n$ in revolving door order. It is built on two algorithms mentioned above.

Our task is not just to generate all paths in revolving door order but to ensure that each path is generated exactly once; and the generation is space and time efficient.
5.3 Revolving Door Algorithms for Spanning Paths

In this section, we present our scheme to generate all undirected spanning paths in a clique on $n$ vertices. Note that this is not same as generating all the orderings of vertices, since we consider the reverse of the path same as the original. A spanning path is a path that includes every vertex of the clique exactly once.

5.3.1 Naive Idea: List Spanning Path Algorithm

In this section, we give a naive idea to generate all spanning paths in $K_n$. It is easy to see that there are $n!/2$ spanning paths in $K_n$. A recursive technique for enumerating all spanning paths is described below:

Let $P = v_1, \ldots, v_k$ be a path of length $k$ and suppose $0 \leq i \leq k$. Consider two operations: append and prepend. The append inserts the vertex $w$ at the end of $P$ and the prepend operation inserts $w$ at the start of $P$.

$$\text{append}(P, w) = v_1, \ldots, v_k, w$$
$$\text{prepend}(P, w) = w, v_0, \ldots, v_k$$

The $\text{rotate}(P)$ operation rotates the given path in cyclic order $P = v_1, \ldots, v_k$ and hence

$$\text{rotate}(P) = v_2, \ldots, v_k, v_1.$$ 

For $n = 3$ consider the following list of paths:
Given a list $\mathcal{L}_{n-1}$ that enumerates all the paths of $K_{n-1}$ and satisfies the above three properties, we can obtain a list $\mathcal{L}_n$ that also satisfies these properties as follows:
Let $L_{n-1} = P_1, P_2, \ldots, P_t$ where $t = (n - 1)!/2$. Consider the list given by:

\[
\begin{align*}
Q_1 &= \text{append}(P_1, v_n) \\
Q_2 &= \text{rotate}(Q_1) \\
Q_3 &= \text{rotate}(Q_2) \\
&\quad \vdots \\
Q_{n-1} &= \text{rotate}(Q_{n-2}) \\
Q_n &= \text{rotate}(Q_{n-1}) = \text{prepend}(P_1, v_n) \\
Q_{n+1} &= \text{append}(P_2, v_n) \\
Q_{n+2} &= \text{rotate}(Q_{n+1}) \\
&\quad \vdots \\
Q_{2n} &= \text{rotate}(Q_{2n-1}) = \text{prepend}(P_2, v_n) \\
Q_{2n+1} &= \text{append}(P_3, v_n) \\
Q_{2n+2} &= \text{rotate}(Q_{2n+1}) \\
&\quad \vdots \\
Q_{3n} &= \text{rotate}(Q_{3n-1}) = \text{prepend}(P_3, v_n) \\
Q_{3n+1} &= \text{append}(P_4, v_n) \\
&\quad \vdots
\end{align*}
\]

In general

\[
Q_{kn+j} = \begin{cases} 
\text{append}(P_{k+1}, v_n), & \text{if } j = 1; \\
\text{rotate}(Q_{kn+j-1}) = \text{rotate}^{j-1}(\text{append}(P_{k+1}, v_n)), & \text{otherwise.}
\end{cases}
\]

where $0 \leq k \leq t$ and $1 \leq j \leq n$. 
Claim 5.3.1.

\[ Q_{kn+n} = \text{prepend}(P_{k+1}, v_n). \]

**Proof.**

Now we outline the proof to show that the above claim holds. Consider the path \( Q_{kn+1} = \text{append}(P_{k+1}, v_n) \). When we apply a rotate operation on this, the last vertex \( v_n \) shifts to left by one position. As this is a cyclic rotate the first vertex of \( Q_{kn+1} \) becomes the last. Further \( n - 2 \) rotate operations will move \( v_n \) to the first position followed by the path \( P_{k+1} \). Thus, \( Q_{kn+n} \) becomes

\[
\begin{align*}
Q_{kn+1} & = P_{k+1}, v_n \\
& \vdots \\
n \text{rotate operations} \\
& \vdots \\
Q_{kn+n} & = v_n, P_{k+1} = \text{append}(P_{k+1}, v_n)
\end{align*}
\]

**Theorem 2.** \( \mathcal{L}_n \) satisfies the following properties:

1. Paths in \( \mathcal{L}_n \) are in revolving door order.

2. The first vertex of every path, except the first one, is the last vertex of the preceding path.

3. For any two distinct paths \( P \) and \( P' \) in \( \mathcal{L}_n \), we have \( P \neq P' \).

4. For any two (not necessarily distinct) paths \( P \) and \( P' \) in \( \mathcal{L}_n \), we have \( P \neq \text{reverse}(P') \).

**Proof.** We proceed by induction on \( n \). The base cases \( (n = 2, 3) \) are easily seen to be true by inspection. Let

\[
\mathcal{L}_{n-1} = P_0, \ldots, P_{l-1}
\]
K3 Spanning Paths

K4 Spanning Paths

insert(Pi,4,4)

rotate(Pi+1)

rotate(Pi+1)

Figure 5.3: Generate $L_4$ from $L_3$
We first show that the paths in $\mathcal{L}_n$ are in revolving door order and the last vertex of each path is the first vertex of the preceding path. The rotate operation maintains these properties. We verify that claim also holds for append and prepend operations.

Consider two successive paths $Q_{k+2n}$ and $Q_{(k+1)n+1}$ in $\mathcal{L}_n$. Note that in $\mathcal{L}_n$, $Q_{(k+1)n+1} = \text{append}(P_{k+2}, v_n)$. From Claim 5.3.1, $Q_{kn+n} = \text{prepend}(P_{k+1}, v_n)$. Hence, $v_n$ is the last vertex of $Q_{(k+1)n+1}$ and the first vertex of $Q_{kn+n}$. To see that $Q_{kn+n}$ and $Q_{k(n+1)+1}$ are in revolving door order, observe that by induction the last vertex $w$ of $P_{k+2}$ is the first vertex of $P_{k+1}$. Hence, $Q_{(k+1)n+1}$ and $Q_{kn+n}$ have the edge $(n, w)$ in common, therefore, the only edges that $Q_{(k+1)n+1}$ and $Q_{kn+n}$ differ on are the ones that $P_{k+1}$ and $P_{k+2}$ differ on. By induction $P_{k+1}$ and $P_{k+2}$ are in revolving door order therefore, $Q_{kn+n}$ and $Q_{n(k+1)+1}$ are also in revolving door order.

Next, we show that $Q_r = Q_s$ implies that $r = s$. If $Q_r = Q_s$, then both paths have $v_n$ at the same position, let say at $i$. By list construction rule, we get $Q_r$ after applying $n-i$ rotate operation on $\text{prepend}(P_k, v_n)$, where $P_k$ is a path in $\mathcal{L}_{n-1}$. That is,

$$Q_r = \text{rotate}^{n-i}(\text{prepend}(P_k, v_n))$$

$$Q_s = \text{rotate}^{n-i}(\text{prepend}(P'_k, v_n))$$

Since $Q_r = Q_s$, then $P_k = P'_k$. Therefore by induction $k = k'$ and hence $r = s$.

Similarly, we can show that two paths in $L_n$ cannot be reverse of each other. Thus, all paths are different.

**Corollary 5.3.1.** $\mathcal{L}_n$ is a list of all spanning paths in $K_n$ in revolving door order. Furthermore, each spanning path in $K_n$ appears in $\mathcal{L}_n$ exactly once.

**Proof.** A simple inductive argument shows that $\mathcal{L}_n$ consist of exactly $\frac{n!}{2}$ spanning paths. Since all paths are distinct, hence, each path must occur in it exactly once. □
5.3.2 Efficient Block Spanning Path Algorithm

In the above scheme, we obtain $L_n$ using $L_{n-1}$. This indicates that we might have to save all paths in $L_{n-1}$ to get the list $L_n$. Hence, the algorithm will take a lot of space and thus, it will not be practical for large $n$. In this section, we develop an algorithm based on the naive idea which generate the paths in $L_n$ on the fly without the need of $L_{n-1}$.

Consider each path as a string of vertices. We define an operation $O_{k}^{i}$, it takes the first $i$ vertices, reverse them, and insert them after the original $k$th vertex. For example, $O_{4}^{2}$ applied to ABCDE yields CDBAE. And if the original string had length $n$, then $O_{n}^{1}$ is often called a cyclic shift; $O_{5}^{1}$ applied to ABCDE yields BCDEA.

Our algorithm is defined recursively. For given $n$, $B_{n}^{0}$ is the empty sequence, and for $n \geq 1$,

$$B_{n}^{i} = B_{n}^{i-1}, O_{n}^{i}, B_{n}^{i-1}, O_{n}^{i}, \ldots, B_{n}^{i-1}$$

where there are $n + 1 - i$ instances of $B_{n}^{i-1}$. For example, $B_{4}^{2}$ is a sequence of $O_{4}^{j}$ where $j = 1, 1, 1, 2, 1, 1, 2, 1, 1, 1$. For $n = 4$, the block algorithm recursion is shown in Figure 5.4.

Figure 5.4: Block structure for $n = 4$
This algorithm has an easy recursive implementation. An iterative version of this algorithm can also be obtained by implementing a special counter of length \( n-2 \). The counter keeps track of the next operation \( O_n^i \). In this counter, the value of \( i^{th} \) index is incremented up to the maximum value of \( n-1-i \). Since, \( O_n^i \) can be implemented in time \( O(i+1) \), hence, the total time taken by the algorithm to generate all \( \frac{n!}{2} \) spanning paths in \( K_n \) is

\[
O\left(\sum_{i=0}^{n-2} (i + 1)(n - i)!\right) = O(n!)
\]

Thus, the algorithm runs in constant amortized time.

**Theorem 3.** \( B_{n-2}^n \) generates all \( \frac{n!}{2} \) paths in revolving door order.

The key observation is the following, which is easily checked:

**Lemma 5.3.1.** The sequence \( O_n^1, O_n^1, \ldots, O_n^1, O_n^j \) where there are \( n-1 \) \( O_n^1 \) operations, has the same net effect as \( O_{n-1}^{j-1} \).

**Proof.** We prove the theorem by induction on \( n \). The base case \( n = 2 \) is trivial. We consider the subsequence of strings after each \( O_n^2 \), because \( O_n^1 \) is just an acyclic shift. By the above, and the definition of \( B_n^i \), this sequence of strings is identical to that generated by \( B_{n-1}^{n-3} \) with the final character \( n \) appended. The operation \( O_n^1 \) is applied to each of these strings \( n-1 \) times to generate all cyclic shifts and maintain revolving door order.

It follows that the strings of \( B_{n-2}^n \) are all distinct and none is the reverse of any other. Since there are \( \frac{n!}{2} \) of them, the only thing to check is that a string and its reverse both do not both. Suppose strings \( x \) and \( y \) are reverse of each other. Then every cyclic rotation of them is in \( B_{n-2}^n \), so let \( x' \) and \( y' \) be the cyclic rotations of \( x \) and \( y \) with \( n \) in the rightmost position. Let \( x'' \) and \( y'' \) be the strings with the \( n^{th} \) symbol removed. It follows that \( x'' \) and \( y'' \) are reverse of each other. This is a contradiction. \( \square \)
5.4 Revolving Door Algorithm for Path Set

In this section, we present an idea which is significant in itself and is also used to generate all paths. Consider the following definition:

**Definition 5.4.1.** A path set is a list of distinct paths $\mathcal{P}S = P_0, P_2, \ldots, P_{t-1}$ such that the paths are in revolving door order and $V(P_i) \neq V(P_j)$ for $i \neq j$. Here, $V(P_i)$ represents a set of vertices of $P_i$.

Next, we present a recursive scheme to generate the path set $\mathcal{P}S$ on $n$ nodes, such that the successive paths in $\mathcal{P}S_n$ differ by an edge. This problem can be restated as: list all $2^n$ sets on $n$ nodes such that two successive sets differ only by the first or the last element. We need this condition because any change (add, delete or swap) to the internal node of a path will change two or more edges of the path. Therefore, two successive path will not be in revolving door order. Many algorithms (Knuth 2005; Savage 1997) have been proposed in the literature for generating all sets and combinations but none of them full fill this requirement.

5.4.1 Naive Scheme

Let us define two operations which are inverse of append and prepend operations define above.

Let $P = v_0, \ldots, v_k, w$, then

$$\text{delrear}(P, v_k) = v_0, \ldots, v_k$$

Similarly, if $P = w, v_0, \ldots, v_k$ then

$$\text{delfront}(P, v_k) = v_0, \ldots, v_k$$

We propose a recursive scheme for generating the path set. For $n = 1$ the paths in the list are given as: $\mathcal{P}S_1 = \{(1), (\emptyset)\}$. Note that $PS_1$ is a complete path set for $K_1$. 
where successive paths differ by a single vertex. Now, assume that we have list

\[ PS_{n-1} = P_0, P_1, \ldots, P_{t-1}, \]

where \( t = 2^{n-1} \), that enumerates the path set for \( K_{n-1} \) in revolving door order. To generate the path set for \( K_n \) we take two successive paths in \( PS_{n-1} \), namely \( P_{2i} \) and \( P_{2i+1} \) (these paths differ by exactly one vertex \( v_k \) by induction), add two new paths between them by inserting \( n \) to \( P_{2i} \) and \( P_{2i+1} \), such that revolving door order is maintained. Continue this procedure, till PS\(_{n-1}\) is completely exhausted. More formally, let

\[ PS_n = Q_0, \ldots, Q_{2t-1} \]

where,

- \( Q_{4j} = P_{2j} \)
- \( Q_{4j+1} = \text{add}(P_{2j}, n) \)
- \( Q_{4j+2} = \text{add}(P_{2j+1}, n) \)
- \( Q_{4j+3} = P_{2j+1} \)

Where,

\[ \text{add}(P, n) = \begin{cases} 
\text{prepend}(P, n), & \text{if } n \text{ is odd} \\
\text{append}(P, n), & \text{if } n \text{ is even} 
\end{cases} \]

**Theorem 4.** \( PS_n = Q_0, \ldots, Q_{2t-1} \) is a complete list of path sets in revolving door order.

**Proof.** The proof is by induction on \( n \). For \( n = 2 \), the claim holds trivially. By induction, the list \( PS_{n-1} \) satisfies the above claim. We obtain the paths in \( PS_n \) by applying either the append or the prepend operation on two successive paths in \( PS_{n-1} \). This ensures that the paths are in revolving door order.

It remains to show that all path are generated exactly once. It is easy to see that the total number of the path in \( PS_{n-1} \) is \( \sum_{k=2}^{n-1} \binom{n-1}{k} = 2^{n-1} \). The list \( PS_n \) includes all the paths in \( PS_{n-1} \) and the paths in \( PS_{n-1} \) with the vertex \( n \) added to either of its
end. Thus, we have $2^n$ paths in $PS_n$. From this, it is obvious that the paths in $PS_n$ are different.

5.4.2 Algorithm

An interesting way to generate a path set is by using a binary counter. Let $x = x_0 \cdots x_n$ be a binary string of length $n + 1$. We define a path set $S(x)$ as follows:

$$S(x) = \{i : x_{i-1} \neq x_i : i > 0\}$$

Consider all the strings with $x_0 = 0$. Let us interpret string $x$ in binary and initialize $y = x + 1$. It is easy to see that two sets $S(x)$ and $S(y)$ differ in exactly one element. Furthermore, one can prove that if $x \neq y$ then $S(x) \neq S(y)$. Thus, we can use a counter to generate the path set as follows:

- Initialize a $n + 1$ bit string $x = 10^n$. Let $S = \{1\}$.
- Repeat $2^n - 1$ times.
  - increment $x$ and let $j$ be the bit that was changed to 1.
  - If $x_{j-1} = 0$ and $j$ is odd $S := \text{prepend}(S, j)$
  - If $x_{j-1} = 1$ and $j$ is odd $S := \text{delfront}(S, j)$
  - If $x_{j-1} = 0$ and $j$ is even $S := \text{append}(S, j)$
  - If $x_{j-1} = 1$ and $j$ is even $S := \text{delrear}(S, j)$

With this simple implementation it is easy to generate the path set in revolving door order. Table 5.1 shows the paths sets for $n = 1, 2, 3, 4$ and 5.
<table>
<thead>
<tr>
<th>No:</th>
<th>$K_1$</th>
<th>$K_2$</th>
<th>$K_3$</th>
<th>$K_4$</th>
<th>$K_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2:</td>
<td>()</td>
<td>12</td>
<td>3 1</td>
<td>1 4</td>
<td>5 1</td>
</tr>
<tr>
<td>3:</td>
<td>2</td>
<td>3 1 2</td>
<td>3 1 4</td>
<td>5 1 4</td>
<td></td>
</tr>
<tr>
<td>4:</td>
<td>()</td>
<td>1 2</td>
<td>3 1</td>
<td>1 4</td>
<td></td>
</tr>
<tr>
<td>5:</td>
<td>2</td>
<td>3 1 2</td>
<td>3 1 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6:</td>
<td>3 2</td>
<td>3 1 2 4</td>
<td>5 3 1 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7:</td>
<td>3</td>
<td>1 2 4</td>
<td>5 3 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8:</td>
<td>()</td>
<td>1 2</td>
<td>3 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9:</td>
<td>2</td>
<td>3 1 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10:</td>
<td>2 4</td>
<td>5 3 1 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11:</td>
<td>3 2 4</td>
<td>5 3 1 2 4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12:</td>
<td>3 2</td>
<td>3 1 2 4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13:</td>
<td>3</td>
<td>1 2 4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14:</td>
<td>3 4</td>
<td>5 1 2 4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15:</td>
<td>4</td>
<td>5 1 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16:</td>
<td>()</td>
<td>1 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17:</td>
<td>2</td>
<td>5 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18:</td>
<td>5 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19:</td>
<td>5 2 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20:</td>
<td>2 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21:</td>
<td>3 2 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22:</td>
<td>5 3 2 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23:</td>
<td>5 3 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24:</td>
<td>3 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25:</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26:</td>
<td>5 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27:</td>
<td>5 3 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28:</td>
<td>3 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29:</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30:</td>
<td>5 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31:</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32:</td>
<td>()</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.1: All path sets for $K_n$


5.5 Note on Generating All Paths

Theorem 5.5.1. It is not possible to generate all the paths in $K_n$ using only insert and delete operations.

Proof. The proof is by contradiction. Assume that we can generate all paths using insert and delete operations. In this case, the number of even length paths and odd length paths should differ by at most one.

The paths of length $k$ in complete graph are:

$$\binom{n}{k} \frac{k!}{2}$$

The total number of the paths in $k_n$ are:

$$\sum_{k=1}^{n} \binom{n}{k} \frac{k!}{2}$$ (5.1)

According to our assumption the number of even and odd length paths can differ by at most one. The difference between even and odd length paths can be written as follows:

$$\sum_{k=\text{even}}^{n} \binom{n}{k} \frac{k!}{2} - \sum_{k=\text{odd}}^{n} \binom{n}{k} \frac{k!}{2} = \sum_{k=1}^{n} \binom{n}{k} \frac{k!}{2} (-1)^k$$

Now we show that

$$\left| \sum_{k=1}^{n} \binom{n}{k} \frac{k!}{2} (-1)^k \right| > 1$$

$$\sum_{k=1}^{n} \binom{n}{k} \frac{k!}{2} (-1)^k = \sum_{k=1}^{n} \frac{n!}{k!(n-k)!} \frac{k!}{2} (-1)^k$$

$$= \frac{n!}{2} \sum_{k=1}^{n} \frac{1}{(n-k)!} (-1)^k$$
Let \( m = n - k \)

\[
A = n!(-1)^n \sum_{m=1}^{n} \frac{1}{m!}(-1)^m
\]

The \( \sum_{m=1}^{n} \frac{1}{m!} \) is an alternating series. This alternating series converges when \( n \to \infty \).

We have

\[
\sum_{m=1}^{\infty} \frac{1}{m!}(-1)^m = \frac{1}{e} \tag{5.2}
\]

We use an error term to bound the next value. Thus,

\[
\frac{1}{e} - \frac{1}{(n+1)!} \leq \sum_{m=1}^{n} \frac{1}{m!}(-1)^m \leq \frac{1}{e} + \frac{1}{(n+1)!} \tag{5.3}
\]

Now we have

\[
|A| \geq n! \left( \frac{1}{e} - \frac{1}{(n+1)} \right) \tag{5.4}
\]

As, for \( n \geq 3 \)

\[
\frac{n!}{e} - \frac{1}{(n+1)} > 1 \tag{5.5}
\]

Therefore,

\[
|A| = n!(-1)^n \sum_{m=1}^{n} \frac{1}{m!}(-1)^m > 1
\]

\[
\sum_{k=1}^{n} \binom{n}{k} \frac{k!}{2}(-1)^k > 1
\]

Thus, it is shown that we can not generate all paths using only delete and insert operations.
5.6 Revolving Door Algorithm for Generating All Paths

A simple idea for generating all paths in $K_n$ is as follows: generate all paths in a path set and apply the block algorithm to each path. This will generate all paths in $K_n$. Unfortunately, the paths might not be in revolving door order. To solve this issue, we can use the reverse operation as a wildcard. That is, by inserting the reverse operation at critical points in right order we can ensure that all paths are generated in revolving door order. Note that the reverse operation does not change any edge in a path, it only reverse the orientation of the path.

Let $P$ be a path of length $k$. Suppose $BU(P)$ denotes the list generated by our block spanning path algorithm for the given path $P$. We assume that the length of the path is implicitly known to us.

Let $PS_{n-1} = (P_0, \ldots, P_{t-1})$ be a list of paths in a path set such that each path appears exactly once in $PS_{n-1}$. Furthermore, suppose we also have paths $Q_0, \ldots, Q_{t-1}$, where $Q_i$ is the last path of $BU(P_i)$. We may need to reverse the current path before/after applying the block unit to get to $Q_i$. Assume that $Q_i$ and $P_{i+1}$ are in revolving door order. Next, we explain how to construct $PS_n$ from $PS_{n-1}$. The list

$$PS_n = R_0, \ldots, R_{2t-1}.$$ 

We define $R_i$ as the output of applying block unit to $S_i$, such that $S_i$ and $R_{i+1}$ are in revolving door order. To do this, we combine the ideas of all spanning path and path set algorithm.

We define $R_{4i}, R_{4i+1}, R_{4i+2}, R_{4i+3}$ and $S_{4i}, S_{4i+1}, S_{4i+2}, S_{4i+3}$ in terms of $P_{2i}, P_{2i+1}$ and $Q_{2i}, Q_{2i+1}$. Since, $Q_{2i}$ and $P_{2i+1}$ are in revolving door order, therefore, there must be a $k$ such that:

$$P_{2i+1} = \text{append}(Q_{2i}, k) \quad \text{or} \quad P_{2i+1} = \text{prepend}(Q_{2i}, k) \quad \text{or} \quad P_{2i+1} = \text{delrear}(Q_{2i}, k) \quad \text{or} \quad P_{2i+1} = \text{delfront}(Q_{2i}, k).$$
We discuss all these cases one by one.

Case 1: $P_{2i+1} = \text{append}(Q_{2i}, v_k)$ for some $k$, and odd $n$. We define,

$$
S_{4i} = Q_{2i},
R_{4i+1} = \text{prepend}(S_{4i}, v_n),
S_{4i+1} = \text{reverse}(BU(R_{4i+1})),
R_{4i+2} = \text{append}(S_{4i+1}, v_k),
S_{4i+2} = \text{reverse}(BU(S_{4i+1})),
R_{4i+3} = \text{delfront}(S_{4i+1}, v_n) = P_{2i+1},
$$

Case 2: $P_{2i+1} = \text{prepend}(Q_{2i}, v_k)$ for some $k$ and even $n$. We define,

$$
S_{4i} = Q_{2i},
R_{4i+1} = \text{append}(S_{4i}, v_n),
S_{4i+1} = BU(\text{reverse}(R_{4i+1})),
R_{4i+2} = \text{prepend}(S_{4i+1}, v_k),
S_{4i+2} = BU(\text{reverse}(R_{4i+2})),
R_{4i+3} = \text{delrear}(S_{4i+2}, v_n) = P_{2i+1},
$$
Case 3: $P_{2i+1} = \text{delrear}(Q_{2i}, v_k)$ for some $k$ and odd $n$. We define,

\begin{align*}
S_{4i} &= Q_{2i}, \\
R_{4i+1} &= \text{prepend}(S_{4i}, v_n), \\
S_{4i+1} &= \text{reverse}(\text{BU}(R_{4i+1})), \\
R_{4i+2} &= \text{delrear}(S_{4i+1}, v_k), \\
S_{4i+2} &= \text{reverse}(\text{BU}(R_{4i+2})), \\
R_{4i+3} &= \text{delfront}(S_{4i+2}, v_n) = P_{2i+1},
\end{align*}

Case 4: $P_{2i+1} = \text{delfront}(Q_{2i}, v_k)$ for some $k$ and even $n$. We define,

\begin{align*}
S_{4i} &= Q_{2i}, \\
R_{4i+1} &= \text{append}(S_{4i}, v_n), \\
S_{4i+1} &= \text{BU}(\text{reverse}(R_{4i+1})), \\
R_{4i+2} &= \text{delfront}(S_{4i+1}, v_k), \\
S_{4i+2} &= \text{BU}(\text{reverse}(R_{4i+2})), \\
R_{4i+3} &= \text{delrear}(S_{4i+2}, v_n) = P_{2i+1},
\end{align*}

Note that in each of the above cases $R_{4i} = P_{2i}$ and we execute $\text{BU}(P)$ to get $Q_{2i}$. To maintain the revolving door order we have to apply the operation $\text{reverse}(P)$ before or after executing the block unit algorithm, $\text{BU}$. Similarly, we get $S_{4i+3}$ by applying the reverse operation and then executing $\text{BU}$.

We need to show that the path set $R_0, \ldots, R_{2t-1}$ have the desired properties. It is clear that $R_0, \ldots, R_{2t-1}$ is a complete list of paths in a path set on $n$ vertices. It remains to show that $S_i$ and $R_{i+1}$ are in revolving door order and $R_{4i+3} = R_{2i+1}$. This is the most critical equality to be established, as the revolving door property of $R_i$’s and $S_i$’s then follows by induction.
We only discuss Case 1, as the remaining cases are similar. Let \( Q_{2i} = v_1, \ldots, v_m \) then \( P_{2i+1} = v_1, \ldots, v_m, k \). Note that in this case,

\[
R_{4i+1} = v_n, v_1, \ldots, v_m.
\]

Hence,

\[
BU(R_{4i+1}) = v_m, v_{m-1}, \ldots, v_2, v_1, v_n
\]

and

\[
S_{4i+1} = \text{reverse}(BU(R_{4i+1})) = v_1, v_n, v_2, \ldots, v_m.
\]

Thus

\[
R_{4i+2} = \text{append}(S_{4i+1}, v_k) = v_1, v_n, \ldots, v_m, v_k
\]

and

\[
S_{4i+2} = BU(R_{4i+2}) = v_k, v_m, \ldots, v_1, v_n.
\]

Thus

\[
S_{4i+2} = \text{reverse}(BU(S_{4i+2})) = v_n, v_1, v_2, \ldots, v_m, v_k.
\]

Finally, this shows that

\[
R_{4i+3} = \text{delrear}(S_{4i+2}, n) = v_1, \ldots, v_m, v_k = P_{2i+1}.
\]

Hence,

\[
S_{4i+3} = Q_{2i+1}.
\]

Where the last equality follows from induction.

The proof that \( R_{4i+3} = P_{2i+1} \) is similar for all four cases. Note that for Cases 3 and 4 the \( \text{reverse}(P) \) operation is performed only if the length of the path \( P \) is greater than 2. The proof of Cases 1 and 3 is shown in Figures 5.5 and 5.6, respectively.
Figure 5.5: Case 1 for generating all paths

<table>
<thead>
<tr>
<th>No</th>
<th>Path Set</th>
<th>Last Block Algo Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2:</td>
<td>3 1</td>
<td></td>
</tr>
<tr>
<td>3:</td>
<td>1 3 2</td>
<td>2 1 3</td>
</tr>
<tr>
<td>4:</td>
<td>1 2</td>
<td></td>
</tr>
<tr>
<td>5:</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>6:</td>
<td>3 2</td>
<td></td>
</tr>
<tr>
<td>7:</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>8:</td>
<td>()</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.2: All $K_3$ paths
<table>
<thead>
<tr>
<th>Insert Vn</th>
<th>Vn</th>
<th>V1</th>
<th>V2</th>
<th>V3</th>
<th>( V_{m-1} )</th>
<th>Vm</th>
<th>Vk</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Block Unit</th>
<th>V1</th>
<th>Vn</th>
<th>V2</th>
<th>V3</th>
<th>( V_{m-1} )</th>
<th>Vm</th>
<th>Vk</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reverse</th>
<th>V1</th>
<th>Vn</th>
<th>V2</th>
<th>V3</th>
<th>( V_{m-1} )</th>
<th>Vm</th>
<th>Vk</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Delete Vk</th>
<th>V1</th>
<th>Vn</th>
<th>V2</th>
<th>V3</th>
<th>( V_{m-1} )</th>
<th>Vm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Block Unit</th>
<th>V1</th>
<th>Vn</th>
<th>V2</th>
<th>V3</th>
<th>( V_{m-1} )</th>
<th>Vm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reverse</th>
<th>V1</th>
<th>Vn</th>
<th>V2</th>
<th>V3</th>
<th>( V_{m-1} )</th>
<th>Vm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Delete Vn</th>
<th>V1</th>
<th>V2</th>
<th>V3</th>
<th>( V_{m-1} )</th>
<th>Vm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.6: Case 3 for generating all paths
5.6.1 Implementation using Counters

The above recursive scheme for generating all paths in $K_n$ can be implemented iteratively using counters. Due to the complicated nature of the recursion, the implementation of the above scheme is a bit tricky. Now, we present an iterative algorithm for all $K_n$ paths using two binary counters. Let $x = x_0 \cdots x_{n-1}$ and $y = y_0 \cdots y_{n-1}$ be two binary strings of length $n$.

In Algorithm 8, Case 1, 2, 3 and 4 are implemented with appropriate $Print(P)$ statements. We increment the counter $x$ and let $k$ be the bit that was changed to 1. Here, $k$ indicates the vertex $v_k$ that is to be inserted. It is not hard to show that the algorithm operates in constant amortized time as it is based on two CAT schemes. We can implement the reverse operation in $O(1)$ time. We have already shown that the cost of the block algorithm is constant amortized. Furthermore, the increment operation of both the counters ($x$ and $y$) takes constant time on average.

Tables 5.2, 5.3, and 5.4 list the paths generated by our scheme for $n = 3, 4$ and 5, respectively. Due to space limitations, these tables only show the path set and the last path generated by the block algorithm.
Algorithm 8  All-$K_n$-Paths(n)

Require: $n$ (the number of vertices in $K_n$)
1: Initialize bit strings $x := 10^n - 1$ and $y := 0^n$
2: Initialize boolean $b := true$
3: if $n = 2$ then $turn := false$
4: Set $P := \{1\}$ and Print($P$)
5: for $i := 0$ to $\frac{2^n}{4}$ do
6:  Increment $x$ and let $k$ be the bit that is changed to $1$
7:  if $n$ is odd then
8:     if $b = true$ then run Case 1 on $P$
9:     else run Case 3 on $P$
10: else
11:     if $b = true$ then run Case 2 on $P$
12:     else run Case 4 on $P$
13: end if
14: Toggle $b$
15: if $i := \frac{2^n}{4}$ then break for-loop
16: Increment $y$ and let $d$ be the bit that is changed to $1$
17: if $d$ is even then reverse($P$)
18: run $BU(P)$
19: if $d$ is odd then reverse($P$)
20: Increment $x$ and let $j$ be the bit that is changed to $1$
21: if $x_{j-1} = 0$ then
22:     if $j$ is odd then append($P, j$)
23:     else prepend($P, j$)
24: else
25:     if $j$ is start vertex of $P$ then delfront($P, j$)
26:     else delrear($P, j$)
27: end if
28: Print($P$)
29: if $d$ is even then reverse($P$)
30: run $BU(P)$
31: if $d$ is odd then reverse($P$)
32: end for
Figure 5.7: All $K_3$ paths
<table>
<thead>
<tr>
<th>No:</th>
<th>Paths</th>
<th>BU Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2:</td>
<td>1 4</td>
<td></td>
</tr>
<tr>
<td>3:</td>
<td>3 4 1</td>
<td>3 1 4</td>
</tr>
<tr>
<td>4:</td>
<td>3 1</td>
<td></td>
</tr>
<tr>
<td>5:</td>
<td>1 3 2</td>
<td>2 1 3</td>
</tr>
<tr>
<td>6:</td>
<td>3 1 2 4</td>
<td>3 1 4 2</td>
</tr>
<tr>
<td>7:</td>
<td>1 4 2</td>
<td>1 2 4</td>
</tr>
<tr>
<td>8:</td>
<td>1 2</td>
<td></td>
</tr>
<tr>
<td>9:</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>10:</td>
<td>2 4</td>
<td></td>
</tr>
<tr>
<td>11:</td>
<td>3 4 2</td>
<td>3 2 4</td>
</tr>
<tr>
<td>12:</td>
<td>3 2</td>
<td></td>
</tr>
<tr>
<td>13:</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>14:</td>
<td>3 4</td>
<td></td>
</tr>
<tr>
<td>15:</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>16:</td>
<td>φ</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.3: All $K_4$ paths
5.7 Conclusion

In this chapter, we provide an efficient scheme to generate all $K_n$ paths in revolving door order using linear space and constant amortized time (CAT). This algorithm is built using all spanning paths and all path set algorithms.
<table>
<thead>
<tr>
<th>No</th>
<th>Paths</th>
<th>BU Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2:</td>
<td>5 1</td>
<td></td>
</tr>
<tr>
<td>3:</td>
<td>1 5 4</td>
<td>4 1 5</td>
</tr>
<tr>
<td>4:</td>
<td>1 4</td>
<td></td>
</tr>
<tr>
<td>5:</td>
<td>3 4 1</td>
<td>3 1 4</td>
</tr>
<tr>
<td>6:</td>
<td>5 3 1 4</td>
<td>4 1 5 3</td>
</tr>
<tr>
<td>7:</td>
<td>3 5 1</td>
<td>1 3 5</td>
</tr>
<tr>
<td>8:</td>
<td>3 1</td>
<td></td>
</tr>
<tr>
<td>9:</td>
<td>1 3 2</td>
<td>2 1 3</td>
</tr>
<tr>
<td>10:</td>
<td>5 3 1 2</td>
<td>2 1 5 3</td>
</tr>
<tr>
<td>11:</td>
<td>3 5 1 2 4</td>
<td>4 2 1 3 5</td>
</tr>
<tr>
<td>12:</td>
<td>3 1 2 4</td>
<td>3 1 4 2</td>
</tr>
<tr>
<td>13:</td>
<td>1 4 2</td>
<td>1 2 4</td>
</tr>
<tr>
<td>14:</td>
<td>5 1 2 4</td>
<td>4 2 5 1</td>
</tr>
<tr>
<td>15:</td>
<td>1 5 2</td>
<td>2 1 5</td>
</tr>
<tr>
<td>16:</td>
<td>1 2</td>
<td></td>
</tr>
<tr>
<td>17:</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>18:</td>
<td>5 2</td>
<td></td>
</tr>
<tr>
<td>19:</td>
<td>2 5 4</td>
<td>4 2 5</td>
</tr>
<tr>
<td>20:</td>
<td>2 4</td>
<td></td>
</tr>
<tr>
<td>21:</td>
<td>3 4 2</td>
<td>3 2 4</td>
</tr>
<tr>
<td>22:</td>
<td>5 3 2 4</td>
<td>4 2 5 3</td>
</tr>
<tr>
<td>23:</td>
<td>3 5 2</td>
<td>2 3 5</td>
</tr>
<tr>
<td>24:</td>
<td>3 2</td>
<td></td>
</tr>
<tr>
<td>25:</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>26:</td>
<td>5 3</td>
<td></td>
</tr>
<tr>
<td>27:</td>
<td>3 5 4</td>
<td>4 3 5</td>
</tr>
<tr>
<td>28:</td>
<td>3 4</td>
<td></td>
</tr>
<tr>
<td>29:</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>30:</td>
<td>5 4</td>
<td></td>
</tr>
<tr>
<td>31:</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>32:</td>
<td>$\phi$</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.4: All $K_5$ paths
Chapter 6

Conclusion and Open Problems

Combinatorial generation is an interesting technique and has applications in various fields. Further studies are needed in this area, and these studies may change our understanding of combinatorial objects in the future. This chapter provides a brief review of the achieved results and describes the open problems.

In this thesis, generation algorithms are developed for combinatorial objects such as pseudo-bicliques, c-isolated bicliques and paths in cliques. Pseudo-bicliques are used to capture real-world interactions in noisy and missing data. In chapter 3, an algorithm is presented for listing all weighted pseudo-bicliques in a given graph $G$. It is based on reverse search technique and is extended to enumerate multilevel pseudo-bicliques. We conducted experiments on random graphs and real-world datasets. The results were quite promising and showed that the algorithm took linear time on average to enumerate each pseudo-biclique in $G$. As a future work, it would be interesting to conduct experiments on huge real-world datasets (with thousands of values) and examine the performance of the algorithm.

This study also considered c-isolated bicliques, another variation of bicliques. The concept of isolation was a new one. In chapter 4, an efficient algorithm was presented to generate all c-isolated bicliques in a given graph in linear time. Furthermore,
computational experiments were carried out on scale-free random graphs and real-world datasets to examine the performance of the algorithm. The results confirmed our theoretical findings and showed that the algorithm can detect $c$-isolated structures in the input graph. This work can be extended to find $c$-isolated bicliques with minimum, maximum, and average isolation levels (Huffner et al. 2009). Moreover, as a future work, one can conduct experiments on real-world datasets using different levels of isolation.

Finally, the enumeration of paths in cliques was considered. An efficient scheme was designed to generate all spanning paths in revolving door order using linear space and constant amortized time (CAT). Furthermore, a CAT algorithm was developed to generate all path sets on $n$ vertices in revolving door order. However, our most significant work on paths was to list all paths in a clique in revolving door order using all spanning paths and all path set schemes. The work leading to path enumeration has generated many interesting and promising ideas. Some of the worth exploring ideas are: generate all paths in an arbitrary graph in revolving door order, list all trees in an arbitrary graph in revolving door order and generate all forests in a graph.


Zadeh, S. M., P. Brzellec, and J. Risler (2004). Cluster-c, an algorithm for the large-scale clustering of protein sequences based on the extraction of maximal