

B. Sc. Engineering Thesis

Graph Modeling for Bioinformatics

By

Md. Shamsuzzoha Bayzid

Student No. 0205043

and

Md. Maksudul Alam

Student No. 0205070

Submitted to

Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
Bachelor of Science in Computer Science and Engineering

Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology (BUET)
Dhaka 1000, Bangladesh

December, 2007

CERTIFICATE

This is to certify that the work presented in this thesis entitled “Graph Modeling for Bioinformatics” is the outcome of the investigation carried out by us under the supervision of Professor Dr. Md. Saidur Rahman in the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology (BUET), Dhaka. It is also declared that neither this thesis nor any part thereof has been submitted or is being currently submitted anywhere else for the award of any degree or diploma.

(Supervisor)

Dr. Md. Saidur Rahman

Professor

Department of Computer Science
and Engineering, BUET, Dhaka.

(Author)

Md. Shamsuzzoha Bayzid

Student No: 0205043

(Author)

Md. Maksudul Alam

Student No: 0205070

Contents

Certificate	ii
Acknowledgements	ix
Abstract	x
I Haplotyping	1
1 Introduction	2
1.1 History of Bioinformatics	3
1.2 Computations in Bioinformatics	4
1.3 Scope of Part I	5
1.4 Summary	6
2 Preliminaries	7
2.1 Cell	7
2.2 Genome Organization	8
2.2.1 DNA	8
2.2.2 Genome	9
2.2.3 Genome Expression	10
2.3 Generation of Genomic Data	10
2.3.1 DNA Sequencing	11
2.3.2 Haplotype Reconstruction and Inference	11
2.3.3 Genome Annotation	11

2.3.4	Analysis of gene expression	12
2.4	Genomic Data Analysis and Applications	12
2.4.1	Phylogeny	13
2.4.2	Study of Single Nucleotide Polymorphism	15
2.4.3	SNPs and Disease Diagnosis	15
2.4.4	SNPs and Personalized Drug Prediction	16
2.5	Summary	16
3	Haplotyping	18
3.1	Haplotype	19
3.2	Individual Haplotyping	19
3.2.1	Terminology	20
3.3	Minimum Error Correction : MEC	22
3.4	A Heuristic Algorithm for MEC Problem	23
3.4.1	Terminologies and Definitions	23
3.4.2	Algorithm - HMEC	25
3.4.3	Implementation and Complexity	27
3.4.4	Approximation to Improve Performance	29
3.4.5	A Simulated Example	29
3.4.6	Performance Evaluation	29
3.4.7	Testing Terminology	32
3.4.8	Result	32
II	Pairwise Compatibility Graphs	35
4	Pairwise Compatibility Graphs	36
4.1	Problem Definition	37
4.2	Applications	38
4.3	Related Works	38

4.4	Scope of Part II	39
4.5	Summary	39
5	Preliminaries	40
5.1	Basic Terminology	40
5.1.1	Graphs	40
5.1.2	Trees	42
5.2	Planar Graphs	43
5.2.1	Planar graphs and plane graphs	43
5.2.2	Dual Graphs	43
5.3	Pairwise Compatibility Graph	44
5.4	Algorithms and Complexity	45
5.4.1	The notation $O(n)$	45
5.4.2	Polynomial algorithms	45
5.4.3	Constant Time	46
6	PCG Graphs	47
6.1	PCG Graphs	47
6.2	Conclusion	55
7	Improper PCG	56
7.1	Improper PCG	56
7.2	Conclusion	61
8	Conclusion	63
	References	64
	Index	66

List of Figures

3.1	Minimum fragment removal using fragment conflict graph	22
3.2	SNP matrix and its partition	24
3.3	An example calculation of <i>Gain</i> measure	28
3.4	An example iteration of HMEC	31
3.5	An example log table	32
3.6	Reconstruction rate vs Coverage value.	33
4.1	A weighted tree and its PCG.	37
5.1	A graph with seven vertices and ten edges.	41
5.2	A complete graph with five vertices.	41
5.3	A complete bipartite graph.	42
5.4	A Tree.	42
5.5	A plane graph G	43
5.6	A plane graph G and its dual graph G^*	44
5.7	A edge weighted tree T and its pairwise compatibility graph G	44
6.1	A complete graph and it's PCG	48
6.2	PCG of a K-partite graph.	49
6.3	A special case of bipartite graph as of Theorem 6.1.3.	49
6.4	General Construction of PCG of the bipartite graph of Theorem 6.1.3.	50
6.5	A bipartite graph as of Theorem 6.1.3 and it's PCG	51
6.6	A special case of bipartite graph as of Theorem 6.1.4.	52

6.7	General Construction of <i>PCG</i> of the bipartite graph of Theorem 6.1.4.	53
6.8	A specific example of graph as of Theorem 6.1.4 and it's <i>PCG</i>	54
7.1	An improper PCG of a graph.	57
7.2	An improper PCG for any graph of n vertices.	57
7.3	Improved construction of improper PCG for graphs with n vertices.	58
7.4	A triangle graph and it's <i>PCG</i>	58
7.5	A special case (<i>case 1</i>) of triangular graph and it's <i>PCG</i>	59
7.6	A special case (<i>case 2</i>) of triangular graph and it's <i>PCG</i>	59
7.7	Combination of <i>case 1</i> and <i>case 2</i>	60
7.8	Construction of the PCG according to the heuristic algorithm.	62

List of Tables

3.1	A chromosome and two haplotypes assembled from it	19
3.2	An SNP matrix.	21
3.3	Different relations between fragments	21
3.4	The pseudocode for the HMEC algorithm	30
3.5	Demonstration of reconstruction rate for different error rate.	34
3.6	Demonstration of execution time.	34

Acknowledgements

First of all, we like to declare that all the appraisal belongs to the Almighty **ALLAH**.

We would like to express our deep gratitude to our supervisor Professor Dr. Md. Saidur Rahman for introducing us to the fascinating and prospective field of bioinformatics and pair-wise compatibility graphs. We have learned from him how to carry on a research work, how to write, speak and present well. We thank him for his patience in reviewing our so many inferior drafts, for correcting our proofs and language, suggesting new ways of thinking, leading to the right way, and encouraging us to continue our research work. We again express our indebtedness, sincere gratitude and profound respect to him for his continuous guidance, suggestions and whole hearted supervision throughout the progress of this work, without which this thesis would never be materialized.

We would like to thank every one associated with our research group. We express our indebtedness to them for their invaluable comments and encouragement throughout the period of this thesis. We also like to thank all of our friends to give us tremendous support.

We are grateful to our families for giving us the moral support to overcome the tedium of repetitive trials to new findings.

Each presentation session was greatly supported by the lab assistants and officials of the department. We would like to thank them for each of the services and facilities they provided us.

Finally, we once again express our belongingness to Allah. All the credits and honor of our achievement goes to Him. He has endowed us with the capability to carry out this work successfully. We deeply express our sincere gratitude to the endless kindness of Allah.

Abstract

Bioinformatics is a highly interdisciplinary field where techniques and concepts from informatics, statistics, mathematics, chemistry, biochemistry, physics, and linguistics merge into a single branch of science. The most prominent and important of these disciplines is information processing. Bioinformatics is the science of using computers to handle biological information.

One of the biggest achievements in the quest for mystery of life is the complete sequencing of human genome. The Human Genome Project successfully discovered the fact that humans are almost 99% identical at the DNA level. Therefore what makes us different is very small region of the whole genome. The smallest possible variation in DNA sequence is the Single Nucleotide Polymorphism. Haplotyping is the process of locating and determining SNPs. In our thesis we will look into the efficient way of detecting haplotypes from erroneous data.

Another important field of bioinformatics is phylogeny. The basic principle of phylogeny is that the origin of similarity is common ancestry. Based on that principle phylogeny tries to determine common ancestral relations. This relationships are easily expressed as a tree known as phylogenetic tree. Pairwise compatibility graph is descended from phylogenetic tree which is an alternate way to express evolutionary relationship.

In our thesis we analyze and compare different haplotyping algorithms based on minimum error correction approach by simulating with original biological data as well as generated data. Most of our task was devoted to simulation. We also study the pairwise compatibility graphs and go a long way to characterize different classes of graphs as pairwise compatible. We also introduce a new notion which we call improper pairwise compatibility graph and also show that every graph fall into this class. The techniques we used to characterize different classes of graphs as pairwise compatibility graph are quite generic and can be utilized in different similar tasks.

Part I

Haplotyping

Chapter 1

Introduction

Analysis of biological experiments and processes is labor intensive and time consuming due to the increasing complexity of the processes and explosive growth of biological data emerging from laboratories worldwide. Hence, the recent challenge is to transform this huge data into knowledge for complete understanding of the biological processes and experiments relating to both health and diseases. The quest for this knowledge has given rise to a new era of science, *bioinformatics*.

Bioinformatics is a highly interdisciplinary field where techniques and concepts from informatics, statistics, mathematics, chemistry, biochemistry, physics, and linguistics merge into a single branch of science. The most prominent and important of these disciplines is information processing. Therefore, “bioinformatics” can be stretched as “**Biological information processing**”. More precisely, bioinformatics derives knowledge from computer analysis of biological data. The development of powerful computers, and the availability of experimental data, that can readily be treated by computation, launched bioinformatics as an independent field.

From the perspective of information processing, bioinformatics can be divided into several areas. The major areas are :

- Develop better tools for **data generation**, capture, and annotation. An example of data generation would be “shotgun sequencing” which identifies the sequence of nucleotides of a target DNA molecule.

- Develop and improve tools for comprehensive functional and relational **analysis**. Numerous tools for phylogenetic analysis of biological data have already been developed to determine the evolutionary relationship among species.
- Develop and improve tools for representing and **comparing** sequence similarity and variation. An example would be BLAST (Basic Local Alignment Search Tool) which aligns two or more sequence and computes different similarity and dissimilarity measures.
- Improve content and utility of biological databases to **store the data** in effective ways. An example would be GenBank. GenBank, developed and housed at NCBI (National Center for Biotechnology information), is the U.S depository for all DNA and protein sequences containing more than 61 million sequence records.
- Create mechanisms to support effective approaches for producing robust, exportable **software** that can be widely shared.

Each of these areas are evolving day by day and are subjects to extensive research. The next section of this chapter contains a snap of the history of bioinformatics. The section following it contains an overview of the computations required in bioinformatics. Finally the scope of this thesis is detailed.

1.1 History of Bioinformatics

The generation of micro-biological data was started first in 1955 when amino acid sequence of a protein (bovine insulin) was announced for the first time by F. Sanger. Since then, hundreds of proteins have been sequenced and analyzed and the need for creating a data bank for these proteins so that the information can be spread very easily to all. In 1973 one such Protein Data Bank (PDB) named Brookhaven was announced. This PDB was fully described in 1977 (<http://www.pdb.bnl.gov>). These PDBs were actually in printed form because of the unavailability of desktop computers and communication facilities at those days.

The first complete genome sequence for an organism (FX174) was published in 1980. The gene consists of 5,386 nucleotide base pairs encoding nine proteins. Just after one year, in 1981, IBM introduced its personal computer in the market and concurrently the smith-waterman

algorithm for sequence alignment was published. The growth of computer performance and biological data is exponential since then and leads to the creation of some new databases like SWISS-PROT in the '80s. Many organizations were also founded in that decade namely Genetics Computer Group (GCG), National Center for Biotechnology Information (NCBI), etc.

Whole genome sequences are published for different single cell organisms like *Haemophilus influenza* of 1.6 Mbp (million base pair), *baker's yeast* of 12.1 Mbp, *E. coli* of 4.7 Mbp in '90s. The Human Genome Project has successfully published the complete sequence of human genome (3000 Mbp) in 2001.

Biochemical methods to sequence, recombine and engineer the DNA sequences have been developed by the 90s. Besides, new modeling and analysis provides new dimensions in the datasets. Thus, the growth of the datasets obviates the necessity of various forms of computation in bioinformatics from the beginning of this decade.

1.2 Computations in Bioinformatics

Computations required in bioinformatics varies largely depending on the objective of the application. Almost all the developed branches of computer science have strong applications in bioinformatics. Some examples of the computational tools that are used in bioinformatics are given below.

Graphs are used to represent relationships among species on different physical and microbiological criteria. For example, the evolutionary relationships among the existing species are expressed in a tree structure called phylogenetic tree. Graphs are also used in problems to analyze biological data.

Numerical simulations of biological systems are used to model systems that are very difficult to be modeled by analytical methods and deterministic operations. For example, the genetic regulatory networks can be modeled by stochastic process. Similarly, host-parasite system, ecosystem etc are well studied through numerical simulation.

Machine learning has many applications in bioinformatics. Generally biological data are huge in quantity but with no established theory. For such data, learning theories provide

methods to gain an insight into the underlying theory of the origin of these data. Besides, statistical analysis can be used in population oriented biology like epidemic controlling, drug designing, etc.

Data mining and advanced database technology are one of the main part of biological information analysis and preservation. The huge amount of data requires efficient processing to maximize their use in research and educational purpose.

1.3 Scope of Part I

There are varieties of applications of graph theory in modeling, representing, analyzing and comparing biological data. Our study covers two major areas concerning the data generation through biological experiments and analyzing represented data.

The first area we covered is computational problems of constructing haplotypes. Haplotypes are sequence of nucleotides that are positioned into some fixed sites in the DNA molecule called the SNP sites. Experimental readout can not completely determine haplotypes due to the lack of precision and correctness in constructing haplotypes, that's why methods are required to build the haplotypes in an optimum way so that the reliability of the haplotypes maximizes. In modeling such optimization problems graphs are extensively used [CvIKT05, BVDL03, BILR05]. We studied several models and in this thesis, we have compiled the problems in a well sorted fashion. Besides, we provide an heuristic algorithm to one of the optimization problems of correcting minimum errors in a set of nucleotide sequences.

The second area we covered is the analysis of relationships among species through phylogeny. Phylogeny is also a well developed branch of bioinformatics. Construction, assembling, visualizing and sampling of phylogenetic trees are major applications of graph in this branch [ACJ03, KMP03, MM81]. In this thesis, we discuss the majority rule method of assembling phylogenetic tree from many small trees. We also discuss a way of arranging haplotypes in phylogenetic order which justifies the relation between the major two areas of our study.

1.4 Summary

In this chapter highlighted bioinformatics from the biological and computational viewpoints. We also placed a short history of bioinformatics to demonstrate the rapid growth of biological data throughout the world. The final section describes the scope of our thesis.

Chapter 2

Preliminaries

Bioinformatics involves the use of techniques of computer science in the field of biology or biochemistry to solve biological problems usually on the molecular level. Hence, it is worth reviewing and initiating these concepts in a brief manner. The first section is about the typical structure of a cell down to molecular level including major biological processes. Generation, assembly, analysis and applications of biological data are discussed in the later sections.

2.1 Cell

The cell is the structural and functional unit of all known living organisms. It is the smallest unit of an organism that is classified as living, and is sometimes called the building block of life. Some organisms, such as bacteria, are unicellular (consist of a single cell). Other organisms, such as humans, are multicellular. Humans have an estimated 100 trillion or 10^{14} cells. Cells range in size from one millimeter down to one micrometer. A typical cell size is 10 μ m and typical cell mass is 1 nanogram. Vital functions of an organism occur within cells, and all cells contain the hereditary information necessary for regulating cell functions and for transmitting information to the next generation of cells.

Cells are highly organized and complicated assembly of large polymeric molecules with specific compartments called organelles. The most important organelle is the nucleus, which houses most of the cellular DNA, the hereditary material of living organisms. Cells are of

two types - prokaryotic cells (e.g. bacteria, amoebae), which lack a defined nucleus and with a simplified internal organization, and eukaryotic cells (body cells of human, animals), which have a more complicated organization including a defined nucleus.

In addition to the nucleus, there are several other organelles in typical eukaryotic cells: the mitochondria, where the cell's energy metabolism is carried out; the rough and smooth endoplasmic reticula, a membranous network in which glycoproteins and lipids are synthesized; Golgi vesicles, which transfers membrane constituents to appropriate places in the cell; and peroxisome, in which fatty acids and amino acids undergo degradation. Animal cells, but not plant cells, contain lysosomes, which degrade unnecessary materials taken in by the cell. Plant cells have chloroplasts, where photosynthesis takes place.

2.2 Genome Organization

2.2.1 DNA

DNA (deoxyribonucleic acid) is the master molecule that contains the genetic instructions used in the development and functioning of all living organisms. DNA contains the instructions needed to construct other components of cells, such as proteins and RNA molecules. The main role of DNA molecules is the long-term storage of genetic information, which is used as a set of blueprints to transfer genetic information from one generation to the next.

DNA is a long polymer made from repeating units called nucleotides. The three dimensional structure of DNA, consists of two long helical strands coiled around a common axis forming a double helix. The DNA double helix is stabilized by hydrogen bonds between the bases attached to the two strands. The four bases found in DNA are adenine (A), cytosine (C), guanine (G) and thymine (T). Each strand of DNA is composed of continually varying sequence of just these four different types of bases called nucleotides.

The DNA segments that carry this genetic information are called genes, but other DNA sequences have structural purposes, or are involved in regulating the use of this genetic information. That means genes are simply portion of nucleotide sequence residing in the DNA

molecule that codes polypeptides or proteins- the main constituents of cells. DNA also contains instructions in form of nucleotide sequence to direct when, which proteins are to be made and in what quantities.

The DNA in the nucleus of a eukaryotic cell is organized among 1 to more than 50 long linear, compact structures. They are called chromosomes. All cells of an organism contain chromosomes of same size and number but they vary among different types of organisms. Each chromosome comprises of a single DNA molecule associated various DNA binding proteins. The total DNA content in the chromosomes of an organism is referred as its genome.

2.2.2 Genome

The genome of an organism is its whole hereditary information and is encoded in the DNA. This includes both the genes and the non-coding sequences of the DNA. More precisely, the genome of an organism is a complete DNA sequence of one set of chromosomes. The term genome can be applied specifically to mean the complete set of nuclear DNA (i.e., the "nuclear genome") but can also be applied to organelles that contain their own DNA, as with the mitochondrial genome or the chloroplast genome. When people say that the genome of a species has been "sequenced," typically they are referring to a determination of the sequences of one set of autosomes and one of each type of sex chromosome, which together represent both of the possible sexes. The study of the global properties of genomes of related organisms is usually referred to as genomics, which distinguishes it from genetics which generally studies the properties of single genes or groups of genes.

Both the number of base pairs and the number of genes vary widely from one species to another, and there is little connection between the two. At present, the highest known number of genes is around 60,000, for the protozoan causing trichomoniasis, almost three times as many as humans have.

The Human Genome is Like a Book: the book is over one billion words long. The book is bound in 5,000 300 page volumes. The book fits into a cell nucleus the size of a pinpoint. A copy of the book (all 5000 volumes) is contained in every cell (except red blood cells) as a

strand of DNA over two miles in length.

2.2.3 Genome Expression

DNA encodes all of the RNA (ribonucleic acid) and protein molecules of the cells of an organism. Proteins are the most abundant and functionally versatile of the cellular macromolecules. Proteins are polymers formed from only 20 different monomers, the amino acids. Many proteins within cells are enzymes, which accelerate (catalyze) biochemical reactions. Proteins also direct their own synthesis and that of other macromolecules, maintain internal cell rigidity, and transport small molecules and ions across membranes.

Protein synthesis from genes does not occur directly. RNA acts as an intermediary molecule. Firstly, a portion of DNA sequence of the large DNA molecule in a chromosome is copied into RNA. The process is called transcription. These RNA copies of segments of the DNA works as templates to direct the synthesis of the protein. This process is called translation as genetic information stored in the form of nucleotide sequence is decoded in the form of amino acid sequence in proteins. DNA can undergo replication (synthesis of new DNA) also. Therefore genetic information in cells owes from DNA to RNA to protein. This fundamental principle genome expression is termed as the central dogma of molecular biology. Genome expression is under fine regulation at various levels.

2.3 Generation of Genomic Data

Bioinformatics primarily deals with a huge range of genomic data gathered from different biological experiment. Computational biology studies the data and derives knowledge from it. Thus generation of genomic data is of prime importance. This section briefly discuss about few effective ways for generation of genomic data.

2.3.1 DNA Sequencing

DNA sequencing is the laboratory technique by which chemical code of the genome is deciphered. DNA sequencing determines the exact order of chemical building blocks, nucleotides (abbreviated A, T, C, and G) that make up the DNA. First, chromosomes are broken into much shorter pieces. After sequencing of the short sequences (in blocks of about 500 bases each, called the read length) they are assembled into long continuous stretches that are analyzed for errors, gene-coding regions, and other characteristics.

The DNA sequence acts as a blueprint, determining what species of organism is produced, and within a species, the DNA sequence of each individual is unique. The DNA sequence that makes up a genome may include coding DNA (gene) and also non-coding DNA. Non-coding DNA does not encode a protein but may regulate where and when genes and proteins are active.

Advances in molecular biology, genomics, and robotics have resulted in automatic sequencing of DNA. Today, the DNA sequence of an entire microbial genome can be determined in just a few weeks rather than several years as in the past. The resulting DNA sequence maps are being used by 21st century scientists to explore biology phenomena.

2.3.2 Haplotype Reconstruction and Inference

A haplotype, is simply the genetic constitution of an individual chromosome. It also refers to a set of single nucleotide polymorphisms (SNPs) found to be statistically associated on a single chromatid (one-half of a replicated chromosome). Such information is most valuable to investigate the genetics behind common diseases.

Haplotypes may be used to compare different populations. Haplotype diversity refers to the uniqueness of a particular haplotype in a given population. Haplogroups are large groups of haplotypes that define genetic populations and are often geographically oriented.

2.3.3 Genome Annotation

Genome annotation is the process of marking the genes and other biological features in a DNA sequence. In its earliest days, gene finding was based on painstaking experimentation on living

cells and organisms. Today, with comprehensive genome sequence and powerful computational resources at the disposal of the research community, gene finding has been redefined as a largely computational problem.

The target genome is searched for sequences that are similar to extrinsic evidence in the form of the known sequence of a messenger RNA (mRNA) or protein product. Given an mRNA sequence, it is trivial to derive a unique genomic DNA sequence from which it had to have been transcribed. Given a protein sequence, a family of possible coding DNA sequences can be derived by reverse translation of the genetic code. Once candidate DNA sequences have been determined, it is a relatively straightforward algorithmic problem to efficiently search a target genome for matches, complete or partial, and exact or inexact.

2.3.4 Analysis of gene expression

The expression of many genes can be determined by measuring mRNA levels with multiple techniques including microarrays, expressed cDNA sequence tag (EST) sequencing, serial analysis of gene expression (SAGE) tag sequencing, massively parallel signature sequencing (MPSS), or various applications of multiplexed in-situ hybridization. All of these techniques are extremely noise-prone and/or subject to bias in the biological measurement, and a major research area in computational biology involves developing statistical tools to separate signal from noise in high-throughput gene expression studies. Such studies are often used to determine the genes implicated in a disorder: one might compare microarray data from cancerous epithelial cells to data from non-cancerous cells to determine the transcripts that are up-regulated and down-regulated in a particular population of cancer cells.

2.4 Genomic Data Analysis and Applications

Bioinformatics creates the tools to store, manage, analyze, compare genomic data. Vast amounts of sequence are now stored in organized computer databases. The genome sequence has been interpreted using computational tools combined with biological knowledge. Computer

software associated with the database is being used for easier data retrieval and data analysis process. Sequence analysis tools can also translate the DNA sequence into protein sequence and can provide information on the predicted physical properties of the protein such as molecular weight. Sequence comparisons also can be used to categorize groups of related gene or sequences into families. Sequences in the same family suggest that the genes or proteins perform similar functions. Another use for sequence comparisons is studying the relatedness and evolution of different genes or organisms.

Here are some research areas where important relationships and predictions are being generated by genomic data analysis with the help of bioinformatics tools:

- Gene number, exact locations, and functions
- Gene regulation
- DNA sequence organization
- Chromosomal structure and organization
- Non-coding DNA types, amount, distribution, information content, and functions
- Coordination of gene expression, protein synthesis, and post-translational events
- Predicted vs experimentally determined gene function
- Evolutionary conservation among organisms
- Correlation of SNPs (single-base DNA variations among individuals) with health and disease
- Disease-susceptibility prediction based on gene sequence variation
- Genes involved in complex traits and multi-gene diseases

Some uses of genomic data are discussed in this section which will be elaborated throughout this thesis.

2.4.1 Phylogeny

Phylogeny is the description of biological relationships based on classification according to similarity of one or more sets of characters or on a model of evolutionary processes. Phylogenetic relationship based different characters are consistent and support one another. Phylogeny is

usually expressed as trees called phylogenetic trees.

The goals of phylogeny is to work out the relationships among species, populations, individuals or genes (generally referred as taxa). Relationship is taken as assignment of a scheme of descendants of a common ancestor. The results are usually presented as an evolutionary tree.

A phylogenetic tree is a two dimensional graph composed of nodes representing the taxa and branches representing the relationships among the taxa. A tree is a connected graph in which there is exactly one path (consecutive set of edges beginning at one point and ending at the other) between every two points. A particular node may be selected as a root. Abstract trees may be rooted or unrooted. Unrooted tree displays the topology of relationship. But it does not show the pattern of the descent. Rooted trees are directed graphs in which each edge is a one-way street and the ancestor-descendant relationship implies the direction of each edge. If every node of rooted trees has two descendants, they are called Binary trees. Numbers are often assigned to the edges of a graph to signify a distance between the nodes connected by the edges. Thus the sizes of the edges can be drawn proportional to the assigned lengths. The length of a path through the graph is the sum of the edge lengths.

In phylogenetic trees edge length signify either some measure of the dissimilarity between two species or the period since their separation. There are two approaches for derivation of phylogenetic trees

- Phenetic approach proceeds by measuring a set of distances between species to generate a tree by hierarchical clustering procedure.
- Cladistic approach considers possible pathways of evolution, inferring the characteristics of the ancestor at each node.

There are two types of data used for building phylogenetic trees:

- Distance-based: A matrix of distances between the species is used as input (e.g., the alignment score between them or the fraction of residues they agree on).
- Character-based: Each character (e.g., a base in a specific position in the DNA) is examined separately.

2.4.2 Study of Single Nucleotide Polymorphism

Single nucleotide polymorphisms or SNPs (pronounced "snips") are DNA sequence variations of single nucleotide (A,T,C,or G) in the genome sequence. For example, a change in the DNA sequence AATTAC to ATTTAC is a SNP. When a variation occurs in at least 1% of the population, it is considered as an SNP. SNPs are found in both coding (gene) and non-coding regions of the genome. Most SNPs occur outside of "coding sequences". SNPs found within a coding sequence are of particular interest to researchers because they are more likely to alter the biological function of a protein.

SNPs, make up about 90% of all human genome sequence variation. Two of every three SNPs are due to the replacement of cytosine (C) with thymine (T). These variations in DNA sequence are believed to be associated with humans respond to disease; environmental insults such as bacteria, viruses, chemicals; and therapies. Thus SNPs has become of great value for biomedical research and for developing pharmaceutical products or medical diagnostics. SNPs are easier to follow in population studies because they are evolutionarily stable - not changing much from generation to generation.

2.4.3 SNPs and Disease Diagnosis

Each person has a unique SNP pattern. In most cases, SNPs do not cause disease, they only serve as biological markers for pinpointing a disease on the human genome map, because they are usually located near a gene found to be associated with a certain disease. Thus SNPs help to determine a person's genetic predisposition to a particular disease based on genes and hereditary factors. Researchers may also identify relevant genes associated with a disease by studying stretches of DNA that have been found to harbor a SNP associated with a disease trait. Thus SNPs will also allow researchers a better evaluation about the impact of non-genetic factors like behavior, diet, lifestyle, and physical activities diseases.

To create a genetic test to screen a disease in which the disease-causing gene has already been identified, scientists may compare SNP patterns of a group of individuals affected by the disease with that of individuals unaffected by the disease. This association study can indicate

which pattern is most likely associated with the disease-causing gene. Then SNP profiles that are characteristic of a variety of diseases can be established and a physician would easily screen individuals for susceptibility to a disease just by analyzing their DNA samples for specific SNP patterns. SNP maps may help them identify the multiple genes associated with such complex diseases as cancer, diabetes and mental disorder, etc.

2.4.4 SNPs and Personalized Drug Prediction

A treatment proved effective in one patient for a particular disease may be found ineffective in others. SNPs are also believed to be useful in helping to determine and understand why individuals differ in their abilities to absorb or clear certain drugs, as well as to determine why an individual may experience an adverse side effect to a particular drug while other may not have any. The most appropriate drug for an individual could be determined by analyzing a patient's SNP pattern. When a medicine works well for a group of people, researchers tries to find out the DNA markers (SNPs) that are alike for these people. Scientists could identify which medicines are best for any one person by using these markers. For example, when a person is in need of medicine, doctors will compare the person's SNP pattern with several SNP patterns of different groups of individuals having same disease and will prescribe individualized therapies specific to a patient's needs. The prediction of the appropriate treatment to the right person is referred to as "personalized drug prediction". Personalized medicine would allow pharmaceutical companies to bring many more drugs to market and allow doctors to suggest a drug that will be most effective for that individual patient.

Therefore, the recent discovery of SNPs are on the way of a revolution in the process of disease detection and the practice of preventative and curative medicine

2.5 Summary

This chapter is written as a suitable starting point for the readers who lack necessary biological background to read the rest of the thesis. The very basics of bioinformatics is actually deeply

rooted in the concepts and discoveries of biologists. That's why, as a student of computer science, to study of bioinformatics is a difficult job and requires much time and effort to grasp the ideas inherent in this beautiful research area.

Chapter 3

Haplotyping

One of the biggest achievements in the quest for mystery of life is the complete sequencing of human genome. The Human Genome Project successfully discovered the fact that humans are almost 99% identical at the DNA level. Therefore what makes us different is very small region of the whole genome. The smallest possible variation in DNA sequence is the Single Nucleotide Polymorphism abbreviated as SNP and pronounced as “snip”. It is also the prominent kind of variation in humans. Let us describe SNP more elaborately because haplotyping is the process of locating and determining SNPs.

An SNP is a specific nucleotide, placed inside a DNA molecule which is otherwise identical for all of us, whose value varies, in a statistically significant way, within a population. For a chromosome all the sites where SNP occurs have been well identified in the human genome project. The base at an SNP site is called allele. The possible variations in a particular SNP site over the entire population are most of the times between two alleles. Such SNP is called bi-allelic. It is still to be explained the reason why bi-allelic SNPs are prominent than multi-allelic SNPs which has three or more different bases. Snips are the most extensive research topic in recent years for the computational biologists.

3.1 Haplotype

Several computational problems related to SNPs have been devised in few years. One of the popular problems is Haplotyping which deals with generating haplotypes from genomic sequence data. Diploid organisms are organized in pairs of chromosomes. A diploid organism has one copy of a specific chromosome inherited from father and another copy of that same chromosome inherited from mother. For each SNP site one can be homozygous (same allele on both chromosomes) or heterozygous (different alleles). The values of a set of SNPs on a particular chromosome copy define a haplotype. An example is shown in Table. 3.1. Two copies of the same chromosome of an individual are aligned and the SNP sites are shown by capital letters. The individual is heterozygous at SNPs 1 and 3 and homozygous at SNPs 2 and 4. The haplotypes are TGGT and AGCT.

paternal copy:	atcatcTcaagtGgaattGctcTctaa			
maternal copy:	atcatcAcaagtGgaattCctcTctaa			
Haplotype 1:	T	G	G	T
Haplotype 2:	A	G	C	T

Table 3.1: A chromosome and two haplotypes assembled from it

Two major category of problems related to haplotyping are

- **Individual Haplotyping** – The Haplotype Assembly Problem
- **Population Haplotyping** – The Haplotype Inference Problem

The following sections elaborate the idea of haplotyping problems in details.

3.2 Individual Haplotyping

Haplotyping an individual consists of determining a pair of haplotypes, one for each copy of a given chromosome. This pair of haplotypes completely define the SNP fingerprints of an individual for a specific chromosome. Given a sequence of bases of a specific chromosome, we

just need to check all the SNP sites to generate the haplotypes. But, the situation is a bit complicated for generating haplotypes from sequencing data. Sequencing data for a genome does not contain the total sequence of bases for a specific chromosome, rather it provides sequences of a set of fragments of the whole genome. Hence the actual problem of individual haplotyping is to find two haplotypes from the set of overlapping fragments of the both copies of a chromosome where fragments may have error and to which copy of the chromosome a fragment belongs is not determined. The very general formulation of the problem is given below [BILR05].

“Given a set of fragments obtained by DNA sequencing from the two copies of a chromosome, find the smallest amount of data to remove so that there exist two haplotypes compatible with all the data remaining.”

Before describing some of the optimization problems of individual haplotyping, the mathematical terminology is presented below.

3.2.1 Terminology

Let, S be the set of k bi-allelic SNP sites over which the haplotypes will be constructed. Let, F be the set of m fragments produced from two copies of the chromosome. Each fragment contains information of nonzero number of SNPs in S . Because the SNPs are bi-allelic, let the two possible alleles for each SNP site be 0 and 1 where they can be any two elements of the set $\{A, T, G, C\}$. Since all the nucleotides are same at the sites other than SNP sites, we can remove these extraneous sites from all the fragments and consider the fragments as sequences of SNP sites only. Thus each fragment $f \in F$ is a string of symbols $\{0, 1, -\}$ of length k where ‘ $-$ ’ denotes an undetermined SNP named as *hole*. All the fragments can be arranged in an $m \times k$ matrix $M = \{M_{ij}\}, i = 1, \dots, m, j = 1, \dots, k$, where row i is a fragment of F and column j is an SNP of S . This matrix is called SNP matrix. An example of an SNP matrix is given in Table. 3.2.

The Consecutive sequence of ‘ $-$ ’ that lies between two non-hole symbols is called a gap. A *gapless* SNP matrix is the one that has no gap in any of the fragments. In the example, the

----1101-----
-----0001110101----
11010010011-----
---10100---010-----
-----10110101011
010111-----01011

Table 3.2: An SNP matrix.

first, second and third rows have no gaps while the fourth and sixth rows both have one gap.

Two fragments f and g are said to have *conflict*, if there exist an SNP position s where two fragments disagree, i.e. $M[f, s] = 0$ and $M[g, s] = 1$ or vice versa. If two fragments do not have conflict on any SNP, they are said to *agree*. Some pairs of fragments are given in Table. 3.3 to illustrate the idea of *conflict* and *agree*.

relation	pair of fragments
conflict	0111010101100 ----010001---
agree	0111010101100 ----010101---
agree	011101----- -----110001
conflict	010001----- -----00001---

Table 3.3: Different relations between fragments

An SNP matrix is *error-free* if it can be partitioned into two classes of non-conflicting fragments. For a specific SNP matrix there may be more than one such partitions. From each non-conflicting partition of fragments, a haplotype can be constructed by just taking the common allele of the non-conflicting fragments for a particular SNP site. Let, partition of rows

in M are M_l and M_r where rows in each set are pairwise non-conflicting. From these two partitions, the construction of haplotypes H_l and H_r by combining the rows can be described as

$$H_{ij} = \begin{cases} 1 & \text{if } M_{ij} = 1 \text{ for at least one row} \\ 0 & \text{if } M_{ij} = 0 \text{ for at least one row} \\ - & \text{if } M_{ij} = - \text{ for all fragments} \end{cases} \quad (3.1)$$

where $i \in \{l, r\}$ and $j = 1, 2, \dots, n$. Although actual haplotypes can not have any hole, haplotype assembly problems can introduce holes in the haplotypes if there is no allelic information in the fragments of the partition. Thus the general problem can now be redefined as to *finding an error-free matrix from a given SNP matrix through optimal changes*.

The following sections deal with an approach of finding error-free matrix.

3.3 Minimum Error Correction : MEC

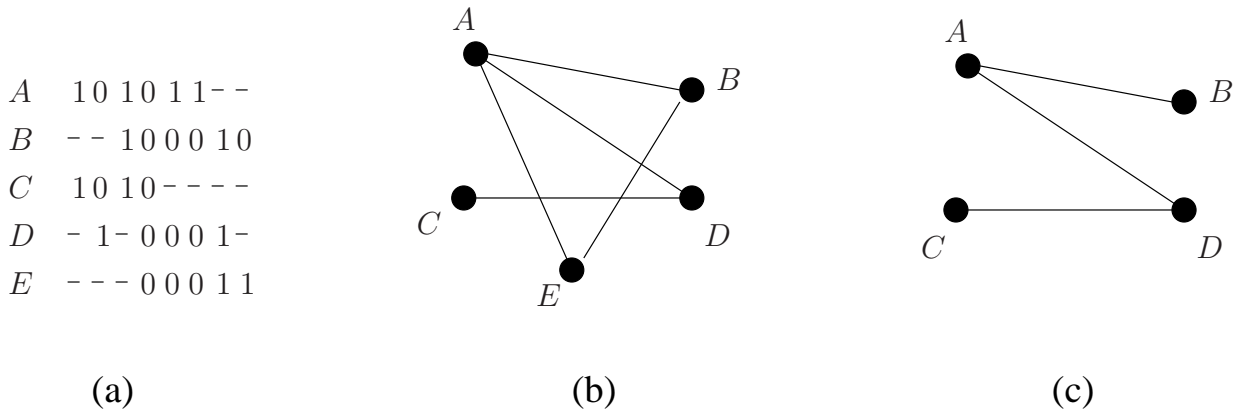


Figure 3.1: Minimum fragment removal using fragment conflict graph

With the advent of sophisticated sequencing methods, errors in SNP matrices are getting smaller. As a result, correcting these errors rather than removing a whole row or column is more preferable approach. Erroneous SNP values can be corrected by just flipping it to the other allele. Obviously it would have not been possible if it were a multi-allele (more than two possible symbols in an SNP) SNP matrix. A good example of saving information by correcting

the errors would be for the SNP matrix in Fig. 3.1. In that matrix fragment E has conflict with fragment B just for the last SNP that is common to them. Thus flipping that SNP of fragment E to 0 would result into an error-free matrix having the partition $\{A,C\}$ and $\{B,D,E\}$. MEC retains the information of the other four alleles of fragment E, while MFR would delete all the five alleles from the matrix.

Problem : *Minimum Error Correction – MEC*

Input : An SNP matrix M .

Output : The smallest set of SNP alleles whose flipping makes M error-free.

MEC problem for gapless SNP matrix is NP-hard [CvIKT05]. Hence, MEC is more difficult than the gapless MFR or MSR which have polynomial time algorithm. For 1-gap case MEC is proved to be APX-Hard. It has been showed that haplotype assembly problem has reasonable input size for practical exact algorithms [Hüf05]. An exact algorithm based on branch and bound technique is available. It searches all possible pairs of haplotypes to find the solution [WWLZ05]. Now, we present a heuristic algorithm to find the solution of a minimum error correction problem.

3.4 A Heuristic Algorithm for MEC Problem

3.4.1 Terminologies and Definitions

Before describing the algorithm, let us redefine the terminologies. Let, $M = \{M_{ij}\}, i = 1, \dots, m, j = 1, \dots, k$, is an SNP matrix of dimension $m \times k$ where $M_{ij} \in \{0, 1, -\}$. There are m fragments; each of which has either a fixed allele (i.e. $\{0, 1\}$) or a gap (i.e. ‘-’) in each of its k SNP sites.

Conceptually, an SNP matrix $M = \langle M_1, M_2, \dots, M_m \rangle$ can be viewed as an ordered set of m fragments where a fragment $M_i = \langle M_{i1}, M_{i2}, \dots, M_{ik} \rangle$ is an ordered set of k alleles or holes (see Fig. 3.2(a)). A fragment M_i is called to *cover* the j th SNP if $M_{ij} \in \{0, 1\}$ and called to *skip* the j th SNP if $M_{ij} = -$. Let, M_s and M_t be two fragments. The distance between two fragments, $D(M_s, M_t)$, is defined as the number of SNPs that are covered by both of the

fragments and have different alleles. For example in Fig. 3.2(a) the $D(M_3, M_2) = 4$. Hence,

$$D(M_s, M_t) = \sum_{j=1}^k d(M_{sj}, M_{tj}) \quad (3.2)$$

where $d(x, y)$ is defined as

$$d(x, y) = \begin{cases} 1 & \text{if } x \neq - \text{ and } y \neq - \text{ and } x \neq y \\ 0 & \text{Otherwise} \end{cases} \quad (3.3)$$

In Table. 3.2, the distance between second and third fragment is 2, as they differ in the seventh and ninth SNP sites (columns).

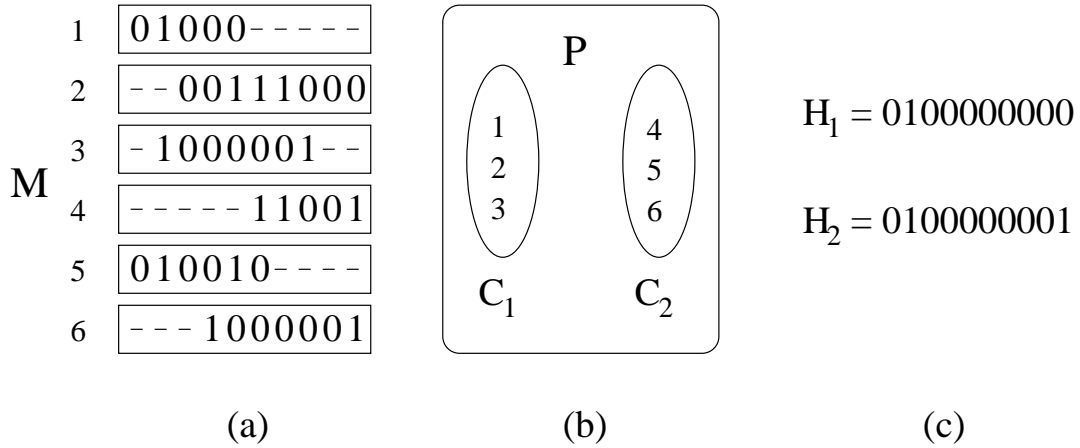


Figure 3.2: SNP matrix and its partition

Two fragments M_s and M_t are said to be *conflicting* if $D(M_s, M_t) > 0$. Let $P(C_1, C_2)$ be a *partition* of M , where C_1 and C_2 are two sets of fragments taken from M so that $C_1 \cup C_2 = M$ and $C_1 \cap C_2 = \emptyset$ [WWLZ05]. In Fig. 3.2(b), an arbitrary partition, corresponding to the SNP matrix of Fig. 3.2(a), is shown. Then an SNP matrix M is an *error-free* matrix if and only if there exists a partition $P(C_1, C_2)$ of M such that for any two fragments $x, y \in C_i, i \in \{1, 2\}$, x and y are non-conflicting, i.e., $D(x, y) = 0$. Such a partition is called an error-free partition. The partition in the Fig. 3.2(b) is not error free since $D(M_1, M_2) > 0$ in C_1 and $D(M_5, M_6) > 0$ in C_2 . A haplotype $H_i, i \in \{1, 2\}$ is constructed from its corresponding fragment class C_i using

the formula

$$H_{ij} = \begin{cases} 1 & \text{if at least one fragment in } C_i \text{ has a 1 in } j\text{th SNP;} \\ 0 & \text{if at least one fragment in } C_i \text{ has a 0 in } j\text{th SNP;} \\ - & \text{if all the fragments in } C_i \text{ skips } j\text{th SNP.} \end{cases} \quad (3.4)$$

where C_i is called the defining class of haplotype H_i and $H_{ij}, i \in \{1, 2\}$ and $j = 1, \dots, k$, denotes the j th element of haplotype H_i . Thus haplotype construction from an SNP matrix may introduce holes in the haplotypes if there is no allelic information in the fragments of the partition.

Now we take a focus to the general minimum error correction problem. If a matrix M is *not* error-free, there will be no error-free partition P . For such M there will be at least one conflicting pair of fragments in each of the classes of all possible partitions. That's why it is impossible to construct a haplotype that is non-conflicting with all the fragments in its defining class of fragments. If we are given a partition $P(C_1, C_2)$ and two haplotypes H_1 and H_2 constructed from P then the number of errors $E(P)$ that must be corrected can be readily calculated by the following formula

$$E(P) = \sum_{i=1}^2 \sum_{f \in C_i} D(f, H_i) \quad (3.5)$$

The MEC problem asks to find a partition P that minimizes the error function $E(P)$ over all such partitions of an SNP matrix M .

3.4.2 Algorithm - HMEC

Now, to minimize the $E(P)$, we need to search all possible partitions of a matrix M . This would certainly require running time exponential to the number of fragments in M . But such a search is not possible because we don't have the real haplotypes to calculate $E(P)$. Hence, we have to approximate $E(P)$ by constructing two haplotypes from the given partition P .

For best approximation we should construct haplotypes which are minimum conflicting with the fragments of their corresponding collections. Therefore, for each SNP site, the haplotype H_i should take the allele that is present in majority of the fragments in C_i . In case of ties, 0 is

avored because it is the more common than 1. In Fig. 3.2(c) the two haplotypes H_1 and H_2 are shown for the partition P in Fig. 3.2(b) which are constructed in this method. To define this construction more mathematically, let $N_j^0(C_i)$ is the number of fragments in a collection C_i that have 0 in j th SNP. Similarly, $N_j^1(C_i)$ defines the number of 1s. Thus to minimize the number of errors $E(P)$ for a specific partition P , the haplotype should be constructed following the rule

$$H_{ij} = \begin{cases} 1 & \text{if } N_j^1(C_i) > N_j^0(C_i); \\ 0 & \text{if } N_j^0(C_i) \geq N_j^1(C_i) \text{ and } N_j^0(C_i) \neq 0; \\ - & \text{if } N_j^1(C_i) = N_j^0(C_i) = 0, \end{cases} \quad (3.6)$$

where $i \in \{1, 2\}$ and $j = 1, 2, \dots, k$.

To find the best partition we will use a local search heuristic. The algorithm iteratively searches a better partition with respect to the current one and chooses it to move to. Because the algorithm searches from within a small set of partitions, the chosen partition may not lead to the optimum solution and the algorithm has a chance to fall into the local optimum solution. The algorithm is inspired from the famous Fiduccia and Mattheyses (FM) algorithm for bipartitioning a hypergraph minimizing the cut size.

This algorithm starts with an arbitrary partition as for example $P(M, \phi)$ and iteratively searches a better partition. In each iteration the algorithm performs a sequence of transfer of distinct fragments from their present collection to the other one so that the partition becomes less erroneous. It should be noted that, a fragment's transfer of collection can both increase or decrease the error function $E(P)$.

Let, the partition before transferring a fragment f is P_p and the partition resulted is P_n . We define the gain of the transfer as $Gain(f) = E(P_p) - E(P_n)$. Let, $F = \langle f_i \rangle, i = \{1, 2, \dots, m\}$ is an ordering of all the fragments in a partition P in such a way that fragment f_i will precede fragment f_j if all the fragments preceding f_i have been transferred to form an intermediate partition P_i and $Gain(f_i) \geq Gain(f_j)$ over P_i . Thus the first intermediate partition P_1 is the current partition P_c of the ongoing iteration. We also define the cumulative gain of a fragment ordering F up to its n th fragment as $CGain(F, n) = \sum_{j=0}^n Gain(f_j)$. Note that, all the gains

used to compute the cumulative gain are calculated over different intermediate partitions. The maximum cumulative gain, $MCGain(F)$ is defined as

$$MCGain(F) = \max_{1 \leq i \leq m} CGain(F, i)$$

Now, in each iteration the algorithm finds the ordering F_c of current partition P_c and transfers only those fragments of F_c that can achieve the $MCGain(F_c)$. The fragment that is the last to be transferred is referred as f_{max} . Thus, the algorithm iterates from one partition to another reducing the error function. Since our algorithm is local search algorithm, it continues whenever $MCGain(F_c) > 0$ and stops as soon as $MCGain(F_c) \leq 0$.

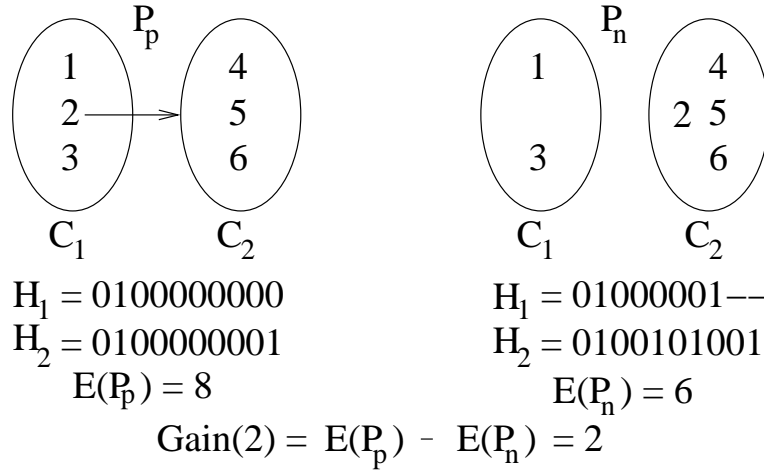
3.4.3 Implementation and Complexity

There are several issues to discuss about the above described algorithms. We will discuss the data structure for each such issue.

To find F_c in each iteration the algorithm repeatedly transfers the fragment that is not transferred previously in this iteration and has maximum gain over all such fragments. To accomplish this we use a locking mechanism. At the beginning of each iteration all the fragments are set free. The free fragment with maximum gain is found out and tentatively transferred to the other collection. After the transfer the fragment is locked at the new collection. This tentative transfer creates the first intermediate partition P_1 . The algorithm then finds the next free fragment with maximum gain in P_1 and transfer and lock that fragment to create the P_2 . Thus, free fragments are transferred until all the fragments are locked and the order of the transfer (F_c) is stored in the log table along with the cumulative gains ($CGain$). $MCGain$ is the maximum $CGain$ and f_{max} is the fragment corresponding to $MCGain$ in the log table.

Although after finishing all such tentative transfers P_c has been changed to an undefined partition, the algorithm checks the log to find the $MCGain(F_c)$ and f_{max} and rollback the transfer of all the fragments that were transferred after f_{max} . When the rollback completes the P_c becomes ready for the next iteration.

While tentatively transferring a free fragment, the algorithm needs to find the fragment with maximum gain among the free fragments (which are not yet transferred). This requires

Figure 3.3: An example calculation of *Gain* measure

calculating gains for each of them. To calculate the $\text{Gain}(f) = E(P_p) - E(P_n)$ for a fragment we need to calculate two error values of two different partitions; the present intermediate partition and the next partition which will be resulted if f is transferred. Each of these error functions requires calculation of two new haplotypes from their corresponding collections (see Fig. 3.3). Although $E(P_p)$ and the haplotypes of P_p can be found from the previous transfer, calculation of $E(P_n)$ requires construction of haplotypes of P_n . Since, the difference between P_p and P_n is only one transfer, we can introduce differential calculation of haplotypes $H_{nj}, j \in \{1, 2\}$ of next partition from the haplotypes of $H_{pi}, i \in \{1, 2\}$ of present partition. For this purpose, the algorithm stores $N_j^1(C_{pi})$ and $N_j^0(C_{pi})$ values of the present partition. After a transfer these values will either be incremented or decremented by 1 or remain the same. Hence, it is now possible to construct $H_{nj}, j \in \{1, 2\}$ in $O(k)$ time. To compute $E(P_n)$ from the haplotypes requires $O(mk)$ time. Therefore, running time to compute the $E(P_n)$ as well as to compute $\text{Gain}(f)$ is $O(mk + k)$.

For each intermediate partition $P_i, i = 1, \dots, n$ we need to compute *Gain* measures for $m - i$ unlocked fragments to find the maximum one. The transfer of this fragments require updating of $N_j^1(C_i)$ and $N_j^0(C_i), i \in \{1, 2\}$ and $j = 1, 2, \dots, k$. So, it also needs $O(k)$ time to run. Finally, there will be m such transfer in each iteration. Thus each iteration will require $O(m(m(mk + k) + k)) \sim O(m^3k)$ running time.

3.4.4 Approximation to Improve Performance

For large SNP matrix $O(m^3k)$ running time is critical to the performance of the algorithm. We can use an approximation in the calculation of the $Gain(f)$ by using only the fragment f and not using the $m - 1$ other fragments. The approximate gain should be

$$AppxGain(f) = D(H_i^p, f) - D(H_j^n, f) \quad (3.7)$$

where H_{pi} is the haplotype of f 's present collection C_i of partition P_p and H_{nj} is the haplotype of f 's next collection C_j of partition P_n . This function ignores the effect of fragments other than f on $Gain(f)$ but reduces the run time of calculating gain to $O(k)$. The total run time of each iteration will be $O(m^2k)$

3.4.5 A Simulated Example

In Fig. 3.4 we present an example iteration of HMEC. We consider that the current partition $P_c = P_1$ is the partition given in Fig. 3.2(b) for the matrix M . All the intermediate partitions $P_i, i \in \{1, \dots, 7\}$ are shown sequentially and the gains of each fragment over the intermediate partitions are shown on the right of each partition. The free fragment with maximum gain is marked in each intermediate partition. For example, the maximum gaining fragment on P_2 is fragment 6 with gain 2. After each transfer the transferred fragment is locked by a circle. Here, the ordering F_c of the fragments is $\langle 2, 6, 5, 1, 4, 3 \rangle$ which is also the order of locking of the fragments. In the log this ordering will be stored along with the $CGains$ (see Fig. 3.5). All the tentative transfers after f_{max} have to be rolled back so that the P_2 becomes the next P_c .

3.4.6 Performance Evaluation

We ran our program on real Biological data as well as simulation data to demonstrate the performance of our program. We also compared our program with most recent Genetic algorithm.

Algorithm HMEC(M)

```

1:   $P_c = P(M, \phi)$ 
2:  FREE_LOCKS()
3:  CLEAR_LOG()
4:  repeat always
5:      while there is an unlocked fragment in  $P_c$  do
6:          begin
7:              find a free fragment  $f$  so that  $Gain(f)$  is maximum
8:              transfer  $f$  to the other collection
9:              update the haplotypes after the transfer
10:             LOCK( $f$ )
11:             LOG_RECORD( $f, Gain(f)$ )
12:          end
13:      FREE_LOCKS()
14:      check the log and find  $MCGain(F_c)$  and  $f_{max}$ 
15:      if  $MCGain(F_c) > 0$ 
16:          begin
17:              set new  $P_c$  by rolling back the transfers
18:                  that occurred after the transfer of  $f_{max}$ 
19:              calculate haplotypes of  $P_c$ 
20:              CLEAR_LOG()
21:              continue the loop
22:          end
23:      else
24:          begin
25:              terminate the algorithm and output current haplotypes
26:          end
27:      end repeat

```

Table 3.4: The pseudocode for the HMEC algorithm

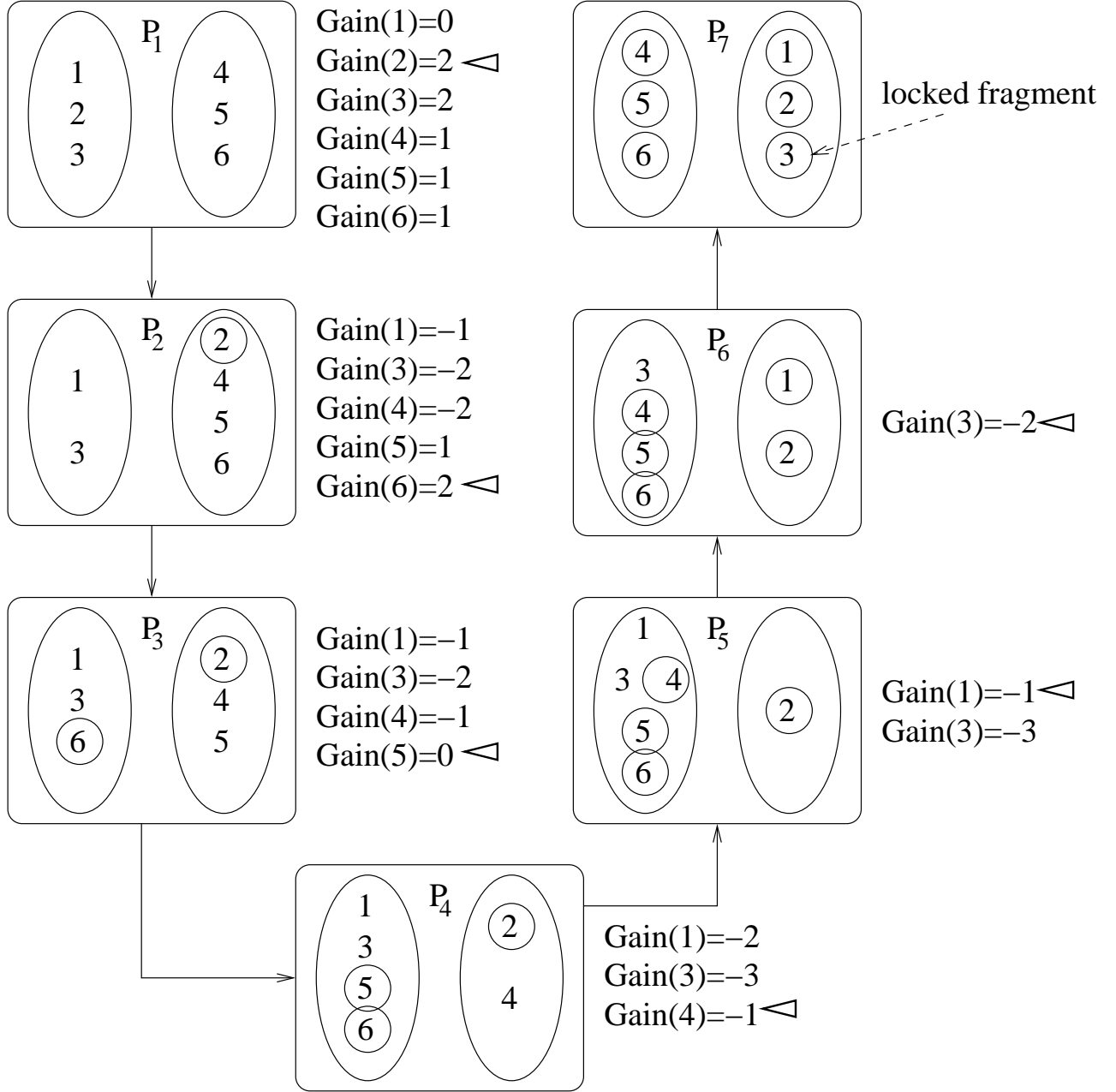


Figure 3.4: An example iteration of HMEC

Log Table	
E_c	CGain
2	2
6	4
5	4
4	3
1	2
3	0

MCGain




Figure 3.5: An example log table

3.4.7 Testing Terminology

Now we proceed to our testing terminologies. We first sample the original haplotype pair into many fragments with certain coverage and error. Each fragment works as distinct sample of the same specimen. Here coverage indicates how many columns of SNP matrix have been sampled out. The remaining slots are gap. Then we introduce some specific amount of error into these samples. The number of fragments, coverage and error rate are user given input for our simulated sequencing technique. The simulation was controlled in several ways. We varied the error rate while no of fragments and coverage were kept constant. Again coverage was varied while no of fragments and error rate were kept constant. Every time we compared our result with that of the Genetic algorithm.

3.4.8 Result

We tested our algorithm and the Genetic algorithm thoroughly with data of various coverage and error rate. The reconstruction rate of our algorithm is very much comparable to that of the genetic algorithm and many of the times it is better. The reconstruction capability of our algorithm approaches better with the increase of coverage value. Fig. 3.6 illustrates the nature of two algorithm for various coverage value. The sharp slope of the corresponding graph of our algorithm is the clear testimony of superiority for our algorithm. We also simulate the algorithms for different error rate keeping the coverage value constant. Table depicts that for

advanced sampling technique, that is for low error rate, the performance of our algorithm is simply tremendous. And for higher error rate it is also very much remarkable and comparable to that of the Genetic algorithm. Our algorithm simply outperform the genetic algorithm, when time needed to reconstruct the haplotype pair is the main concern. The table illustrates how fast our algorithm is compared to the genetic algorithm. For an SNP matrix of 964 columns it takes only 0.04 seconds whereas Genetic algorithm takes 111 seconds when programs are executed on a Pentium-III processor.

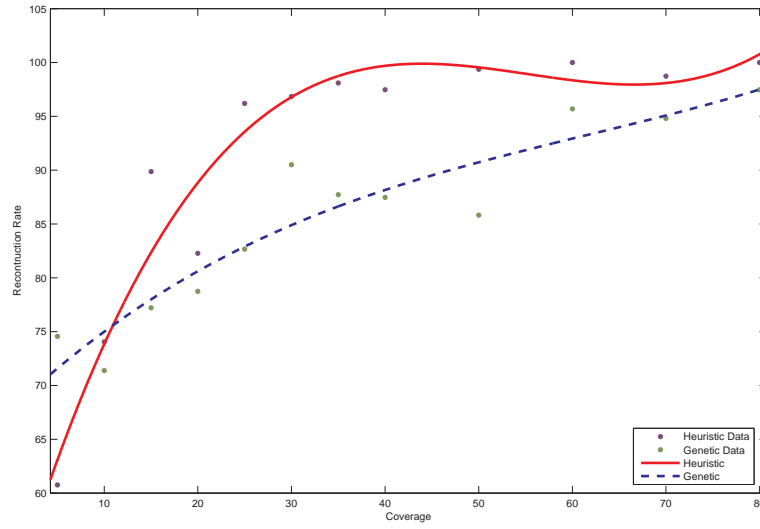


Figure 3.6: Reconstruction rate vs Coverage value.

Another Salient feature of our algorithm is it's deterministic nature. In every execution, our algorithm generates same result for same data whereas Genetic algorithm, which is fully random in nature, generates different result in different execution. We have observed standard deviation up to 11 for Genetic algorithm whereas that of our algorithm is obviously 0.

Table 3.5: Demonstration of reconstruction rate for different error rate.

Error rate (percent)	HMEC (percent)	Genetic (percent)
2	100	86.453
5	99.37	91.202
7	98.73	91.264
15	84.81	86.708
20	84.81	88.164
25	93.67	82.406
30	79.75	88.355
35	92.40	81.898
40	70.89	78.482
50	85.44	78.986

Table 3.6: Demonstration of execution time.

Length of haplotype	HMEC (sec)	Genetic (sec)
79	0.01	9.483
158	0.01	17.806
316	0.01	35.101
632	0.03	71.142
862	0.04	100.064
964	0.04	111.119

Part II

Pairwise Compatibility Graphs

Chapter 4

Pairwise Compatibility Graphs

One of the most widely studied areas of biology is phylogeny. It is description of biological relationships among different species. A statement of phylogeny among objects is based on the notion of common ancestry. The basic principle of phylogeny is that the origin of similarity is common ancestry. Based on that principle phylogeny tries to determine common ancestral relations. This relationships are easily expressed as a tree known as phylogenetic tree. Pairwise compatibility graphs are derived from phylogenetic trees. This is an alternative way of viewing evolutionary relationships.

Phylogeny is the description of biological relationships among different species. A statement of phylogeny among objects is based on the notion of common ancestry. The basic principle of phylogeny is that the origin of similarity is common ancestry. Based on that principle phylogeny tries to determine common ancestral relations. This relationships are easily expressed as a tree known as phylogenetic tree. Pairwise compatibility graphs are derived from phylogenetic trees. This is an alternative way of viewing evolutionary relationships. Dealing with a sampling problem in a phylogenetic tree Kearney *et al.* introduced the concept of pairwise compatibility graphs [KMP03].

In this chapter we give the definition of pairwise compatibility graph, their applications, description of previous works and our results.

4.1 Problem Definition

Let T be an edge weighted tree and $G = (V, E)$ be a graph such that, each vertex of G correspond to a leaf of T and an edge $(u, v) \in E$ if and only if distance between the two leaves of T corresponding to u and v is within a given limit. We call G a pairwise compatibility graph of T . Fig. 4.1(a) depicts an edge-weighted tree T and Fig. 4.1(b) depicts a pairwise compatibility graph G of T , where $d_{min} = 5$ and $d_{max} = 6$. Here, G has the edge (a, b) since the distance between the leaves corresponding to a and b is 5 in T , but G does not contain the edge (a, d) since the distance between the leaves corresponding to a and d in T is 8 which is greater than d_{max} . All the remaining edges of G are drawn following the same rule.

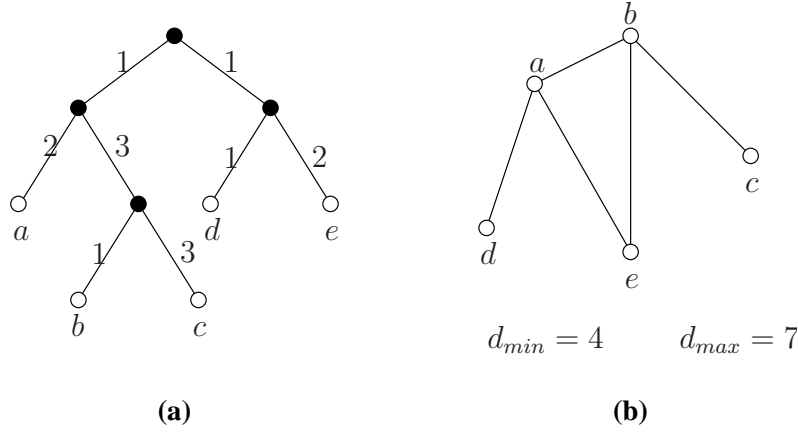


Figure 4.1: A weighted tree and its PCG.

Given a tree T and two limits, d_{min} and d_{max} , the construction of a *PCG* is an easy and trivial problem. In our thesis we focus on the more complex counterpart of the problem which is just the reverse of this, i.e., we are given with a graph G and we have to find out a tree T and suitable limit of distance, d_{min} and d_{max} , such that G becomes the *PCG* of T . The problem can be formalized as follows.

Problem : *Pairwise Compatibility Tree construction*

Input : A graph $G = (V, E)$.

Output : A tree T and distance limit, d_{min} and d_{max} , such that each vertex of G corresponds to a leaf of T and an edge $(u, v) \in E$ if and only if distance between the two leaves of T

corresponding to u and v is within the given limit.

The pairwise compatibility graph recognition problem asks to determine whether a given graph is a pairwise compatibility graph or not.

4.2 Applications

Pairwise compatibility graphs have many applications in phylogenetic reconstruction which is the reconstruction process of evolutionary relationship of a set of species from biological data [JP04, Les02]. Usually phylogenetic relationship is expressed as a tree, known as phylogenetic tree. Dealing with a sampling problem in a phylogenetic tree Kearney *et al.* introduced the concept of pairwise compatibility graphs [KMP03]. They showed that “the clique problem” is polynomially solvable for pairwise compatibility graphs if the pairwise compatibility tree construction problem can be solved in polynomial time. The clique problem asks to determine a maximum set of pairwise adjacent vertices in a graph [CLRS01]. It is an well known NP-complete problem that arises from many applications areas [PX94]. Thus, the pairwise compatibility graph recognition problem and the pairwise compatibility tree construction problem have great potential from the view point of research and practitioner purpose.

4.3 Related Works

The problems associated with *PCG* have not yet been extensively studied. The smallest graph class that encompasses all of the pairwise compatibility graphs is not known at all. It is known that every graph of five vertices or less is a *PCG* [Phi02]. However, no result on the *PCG* recognition problem for arbitrary large graphs is known. Not many well known classes of graphs can be characterized as *PCG* yet. Kearney *et al.* showed that nearly every problem on *PCGs* remained unsolved and posed some open problems [KMP03].

But most recently some structural properties of pairwise compatibility graphs have been unveiled. Tozammel and Yanhaona in their undergraduate thesis determined the relationship between pairwise compatibility graphs and chordal graphs [HY07]. They also proved that

chordal graphs are pairwise compatibility graphs in a restricted case. They showed that number of nontrivial component is not affected when value of d_{max} is set to certain large value and only d_{min} is varied to construct different *PCGs*. They solved the open problem, whether any (or every) cycle of length greater than five a pairwise compatibility graph, given by Kearney *et al.* [KMP03]. They prove that, all chordless cycles and single chord cycles are pairwise compatibility graphs. They gave a linear time algorithm for constructing trees from chordless cycles and single chord cycles.

4.4 Scope of Part II

In this thesis we characterize some classes of graphs as *PCG*. We prove that every complete graphs (K_n) and complete k -partite graphs are *PCG*. We also characterize two restricted classes of bipartite graph as *PCGs*.

We also introduce the notion of *ImproperPCG* that allows some redundancies. We shall describe these terms in Section. 7.1. We prove that every graph is an *improper PCG*. Here, we also give an efficient algorithm to construct the pairwise compatibility tree for any planar graph with all the faces triangulated allowing some redundancies.

The pairwise compatibility tree construction methods, that we describe in this thesis, are very much generic and adoptable in other similar problems.

4.5 Summary

In this part of the thesis, we characterize some classes of graphs as pairwise compatibility graphs and introduce the new notion of *improperPCG*. We give some efficient construction that can generate pairwise compatibility tree in linear time.

The rest of the thesis is organized as follows. Chapter 5 is devoted for preliminaries. Chapter 6 deals with our findings about *PCG*. In Chapter 7 we present our notion of *improper PCG* and give some associated characterization and algorithms.

Chapter 5

Preliminaries

In this chapter we define some basic terminology of graph theory. Definitions and terminologies not included in this chapter will be introduced as they are needed. We start, in Section 5.1, by giving some definitions of standard graph-theoretical terms used throughout the remainder of this thesis. In Section 5.1.2 we define various terms about trees. We devote Section 5.2 to define terms related to planar graphs. We define pairwise compatibility graph in Section 5.3. In Section 5.4 we define various terms about algorithms and complexity.

5.1 Basic Terminology

In this section we give definitions of some theoretical terms used throughout the remainder of this thesis. Interested readers are referred to detailed texts of the literature [Wes00, NR04].

5.1.1 Graphs

A *graph* G is a structure (V, E) which consists of a finite set of *vertices* V and a finite set of *edges* E ; each edge is an unordered pair of vertices. The sets of vertices and edges of G are denoted by $V(G)$ and $E(G)$ respectively. Fig. 5.1 depicts a graph G where each vertex in $V(G) = \{v_1, v_2, \dots, v_7\}$ is drawn as a small dark circle and each edge in $E(G) = \{e_1, e_2, \dots, e_{10}\}$ is drawn by a line segment. An edge connecting vertices u and v in V is denoted by (u, v) . If $(u, v) \in E$, then two vertices u and v are said to be *adjacent* in G ; edge (u, v) is then said to

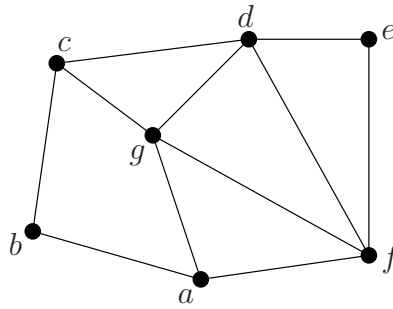


Figure 5.1: A graph with seven vertices and ten edges.

be *incident* to vertices u and v . The *degree* of a vertex in G is the number of edges incident to it in G .

A graph is called a *simple graph* if there is no “loop” or “multiple edges” between any two vertices in G . *Multiple edges* join the same pair of vertices, while a loop joins a vertex itself. A

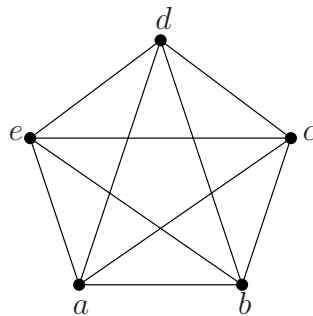


Figure 5.2: A complete graph with five vertices.

complete graph is a simple graph whose vertices are pairwise adjacent; the complete graph with n vertices is denoted by K_n . Figure 5.2 is an example of a complete graph with five vertices. A graph G is *k-partite* if $V(G)$ can be expressed as the union of k independent sets. (The definition of independent set may be required). When $k = 2$ it is called a bipartite graph. A *complete k-partite graph* is a simple graph such that two vertices are adjacent if and only if they are in different partite sets. Figure 5.3 is an example of two partite (bipartite) graph.

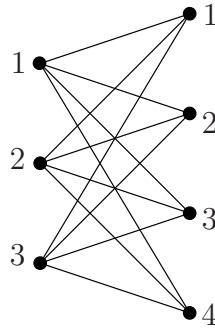


Figure 5.3: A complete bipartite graph.

5.1.2 Trees

A *tree* is a connected acyclic graph. Figure 5.4 is an example of a tree T . The vertices in a tree are usually called nodes. The vertices of degree one in a tree are called *leaves* and the other vertices are called *internal vertices*.

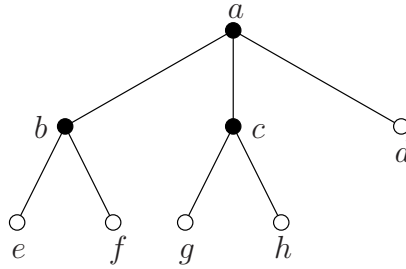


Figure 5.4: A Tree.

In Figure 5.4, the leaves are v_5, v_6, v_7, v_8 , and the internal nodes are v_1, v_2, v_3, v_4 . The weight of an edge (u, v) is denoted by $weight(u, v)$. The *parent* of a vertex v in a tree T denoted by $parent(v)$ is the immediate ancestor internal vertex in the tree. Length of the path between u and v in T denoted by $d(u, v)$ is the sum of the weights of the edges of the path. A *caterpillar* is a tree in which a single path (*the spine*) is incident to or contains every edge. A *star* is a tree where each of the leaves has a common parent which we call the *base* of the star. In this paper every tree we considered is a weighted tree. We use the convention that, if an edge of a tree has no number assigned to it then its default weight is 1.

5.2 Planar Graphs

In this section we give some definitions related to planar graphs used in the remainder of the thesis. For readers interested in planar graphs we refer to [NC88].

5.2.1 Planar graphs and plane graphs

A graph G is *planar* if it has a drawing in the Euclidean plane without edge crossings. A *plane graph* is a planar graph with a fixed planar embedding. A planar graph may have exponential number of planar embedding. Fig. 5.5 shows an example of plane graph. A plane graph G divides the Euclidean plane into connected regions called *faces*. A planar graph where each of the faces is a triangle is called triangular planar graph.

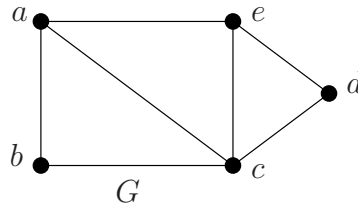
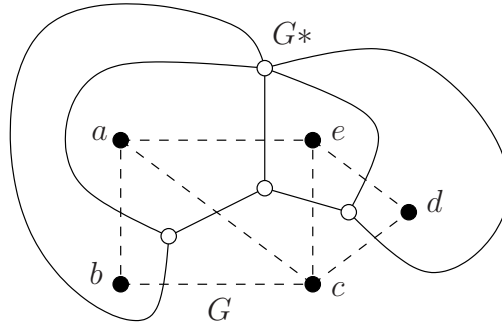


Figure 5.5: A plane graph G .

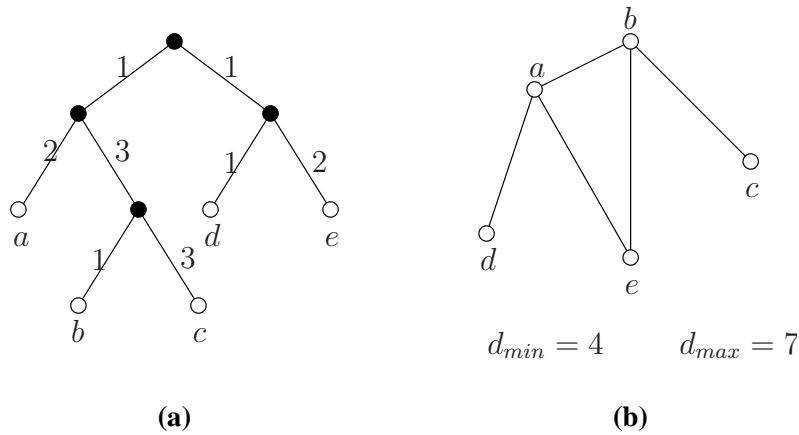
5.2.2 Dual Graphs

For a plane graph G , we often construct another graph G^* called the (*geometric*) *dual*. The *dual* of a plane graph $G = (V, E)$ is a graph $G^* = (V^*, E^*)$, where for every face of G , there is a corresponding vertex in V^* and the edges of G is defined as follows : if two faces X and Y of G share a common edge e , there is an dual edge e^* of e that joins the two vertices in V^* that represents the faces X and Y respectively. The construction is illustrated in Fig. 5.6; the vertices v_i^* are represented by small white circles, and the edges e^* of G^* by solid lines while the edges of plane graph G is represented by dashed lines. G^* is not necessarily a simple graph even if G is simple. Clearly the dual G^* of a plane graph G is also plane.

Figure 5.6: A plane graph G and its dual graph G^* .

5.3 Pairwise Compatibility Graph

Let T be an edge weighted tree and $G = (V, E)$ be a graph such that, each vertex of G correspond to a leaf of T and an edge $(u, v) \in E$ if and only if distance between the two leaves of T corresponding to u and v is within a given limit. We call G a pairwise compatibility graph of T . Fig. 4.1(a) depicts an edge-weighted tree T and Fig. 5.7(b) depicts a pairwise compatibility graph G of T , where $d_{min} = 5$ and $d_{max} = 6$.

Figure 5.7: A edge weighted tree T and its pairwise compatibility graph G .

In this paper we use the term *PCG* as the acronym of pairwise compatibility graph.

5.4 Algorithms and Complexity

In this section we introduce some terminologies related to complexity of algorithms. We can define algorithm as a precise method usable by a computer to solve a problem. An algorithm must solve any instances of a problem and terminate after a finite number of operations. The most widely accepted complexity measure for an algorithm are the running time. The *running time* is the number of operations it performs before producing the final answer. The number of operations required by an algorithm is not the same for all problem instances. Thus, we consider all inputs of a given size together, and we define the complexity of the algorithm for that input size to be the worst case behavior of the algorithm on any of these inputs. Then the running time is a function of size n of the input.

5.4.1 The notation $O(n)$

In analyzing the complexity of an algorithm, we are often interested only in the "asymptotic behavior", that is, the behavior of the algorithm when applied to very large inputs. To deal with such a property of functions we shall use the following notations for asymptotic running time. Let $f(n)$ and $g(n)$ are the functions from the positive integers to the positive real numbers, then we write $f(n) = O(g(n))$ if there exists positive constants c_1 and c_2 such that $f(n) \leq c_1 g(n) + c_2$ for all n . Thus the running time of an algorithm may be bounded from above by phrasing like "takes time $O(n^2)$ ".

5.4.2 Polynomial algorithms

An algorithm is said to be *polynomially bounded* (or simply *polynomial*) if its complexity is bounded by a polynomial of the size of a problem instance. Examples of such complexities are $O(n)$, $O(n \log n)$, $O(n^{100})$, etc. The remaining algorithms are usually referred as *exponential* or *non-polynomial*. Example of such complexity are $O(2n)$, $O(n!)$, etc.

When the running time of an algorithm is bounded by $O(n)$, we call it a *linear time* algorithm or simply a *linear* algorithm.

5.4.3 Constant Time

In computational complexity theory, *constant time* refers to the computation time of a problem when the time needed to solve that problem doesn't depend on the size of the data that is given as input. Constant time is notated as $O(1)$.

For example, accessing the elements in the array takes constant time as we can pick up an element using the index and start working with it. However finding the minimum value in an array is not a constant time operation as we need to scan each element of the array and then decide the minimum of those elements. Hence it is a linear time operation and takes $O(n)$ time.

Chapter 6

PCG Graphs

In this chapter we identify different well known classes of graphs as *PCG*.

Now we proceed to our findings. We present our findings through theorems in the rest of the chapters.

6.1 PCG Graphs

Theorem 6.1.1 *Every complete graph (K_n) is a PCG.*

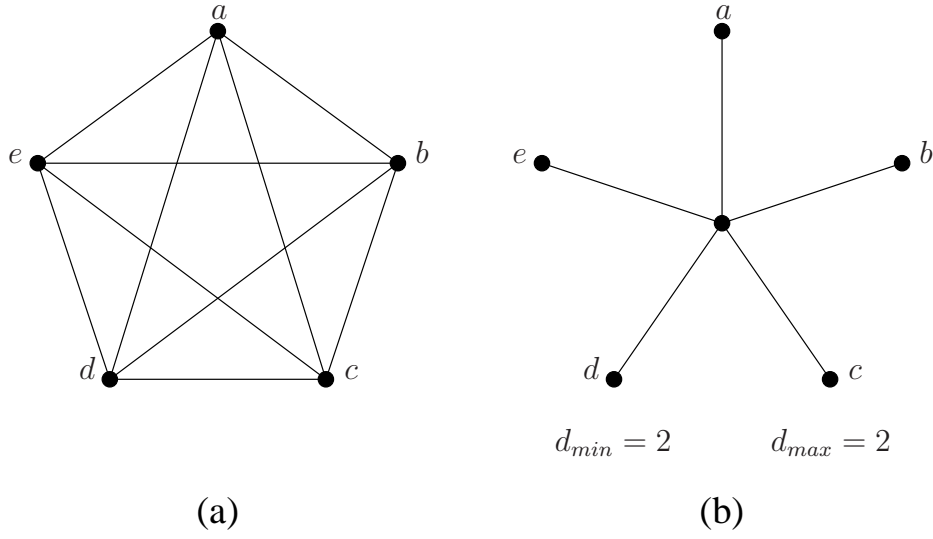
Let, $K_n = (V, E)$ be a complete graph. We construct a star such that all the vertices of K_n appear as the leaf of the star as illustrated in Fig. 6.1. The weight of each of the edge of the star is one. Let, $d_{min} = d_{max} = 2$. Now we are done. The resulting tree is the pairwise compatibility tree of K_n .

All the vertices of K_n are pairwise adjacent and we can see that for all $u, v \in V$, the distance between the corresponding nodes in the tree is 2 which is within the specified range.

In the next theorem, we show that every complete k -partite graph is a *PCG*.

Theorem 6.1.2 *Every complete k -partite graph is a PCG.*

Let, $G = (V, E)$ be a k -partite graph. Let, m_1, m_2, \dots, m_k be k independent sets of G and n_1, n_2, \dots, n_k be the corresponding number of vertices of these independent sets. We denote the j^{th} vertex of i^{th} independent set as $m_{i,j}$. Now, we organize each partite set as a star as

Figure 6.1: A complete graph and it's *PCG*.

Theorem. 6.1. Then we connect the bases of all the stars to a single node as illustrated in Fig. 6.2. Let, the weight of each of the edge of the resulting tree (T) is one. Now if we take $d_{min} = d_{max} = 4$, the resulting tree (T) is the pairwise compatibility tree of G .

Here, for any $u, v \in m_i$; $i \in 1, 2, \dots, k$, the distance between the corresponding nodes in T is 2 which is less than d_{min} and for any $u \in m_i$ and $v \in m_j$, where $i, j \in 1, 2, \dots, k$ and $i \neq j$, the corresponding distance in T is 4 which is within the range. So, G is the *PCG* of T for the specified choice of $d_{min} = d_{max} = 4$.

So far we deal with complete graph and complete k -partite graph which is relatively easy problem. In the following two theorem we focus on the two restricted set of general bipartite graph.

Theorem 6.1.3 *A bipartite graph $K_{m,n}$ is a PCG if there exist two nonempty sets X and Y where $X \in m$ and $Y \in n$ such that there is no edge uv , $u \in X$ and $v \in Y$ and $K_{m-X,n}$ and $K_{m,n-Y}$ are complete bipartite graphs.*

Fig. 6.3 illustrates an example of the bipartite graph as specified in Theorem. 6.1.3. Here $m = 5$, $n = 4$, $X = \{1, 2, 3\}$ and $Y = \{1, 2\}$. Now we give a constructive proof of Theorem. 6.1.3.

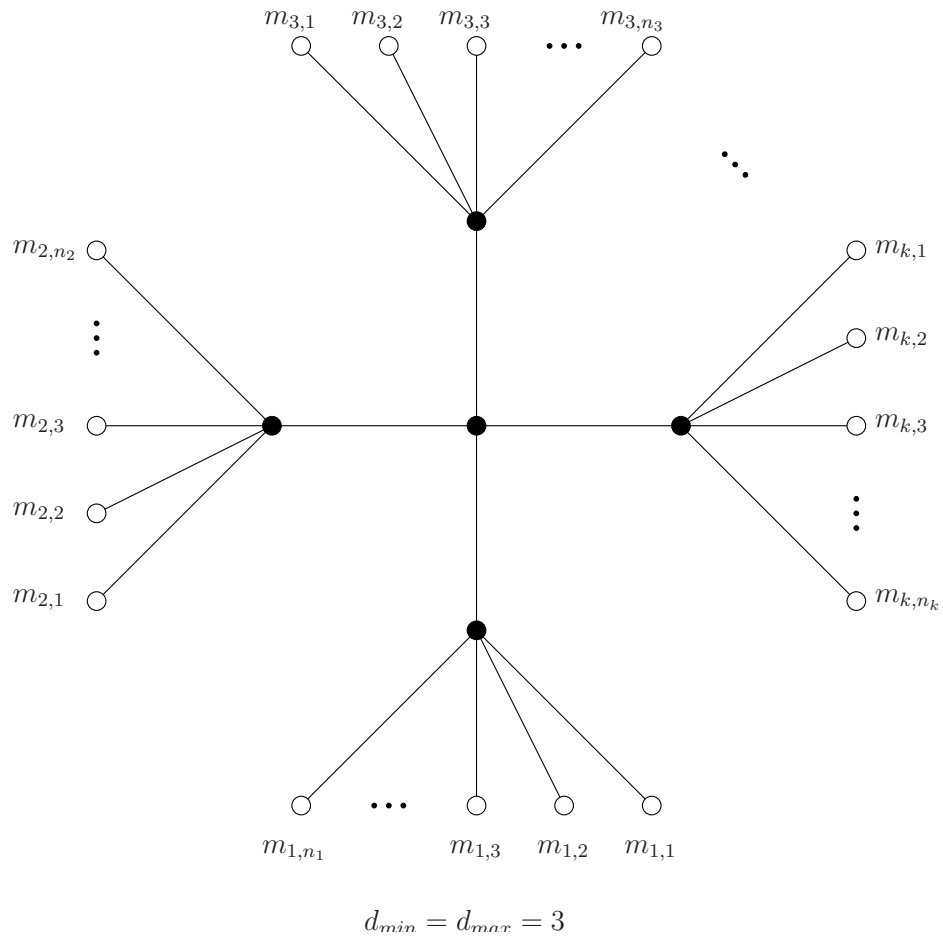


Figure 6.2: PCG of a K-partite graph.

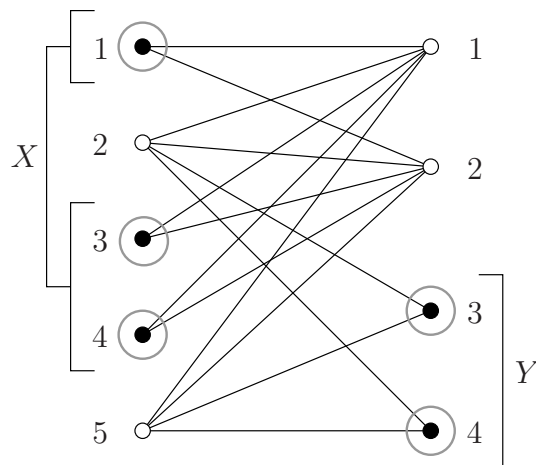


Figure 6.3: A special case of bipartite graph as of Theorem 6.1.3.

Let, $G = (V, E)$ be a bipartite graph $(K_{m,n})$ with $|X| = x$ and $|Y| = y$. Now construct two caterpillars (C_m and C_n) with the vertices of two partite sets as the leaves of the caterpillars such that the rightmost x leaves of C_m and the leftmost y leaves of C_n correspond to X and Y respectively. Fig. 6.4(a) depicts this construction. Now we construct a single tree (T) by

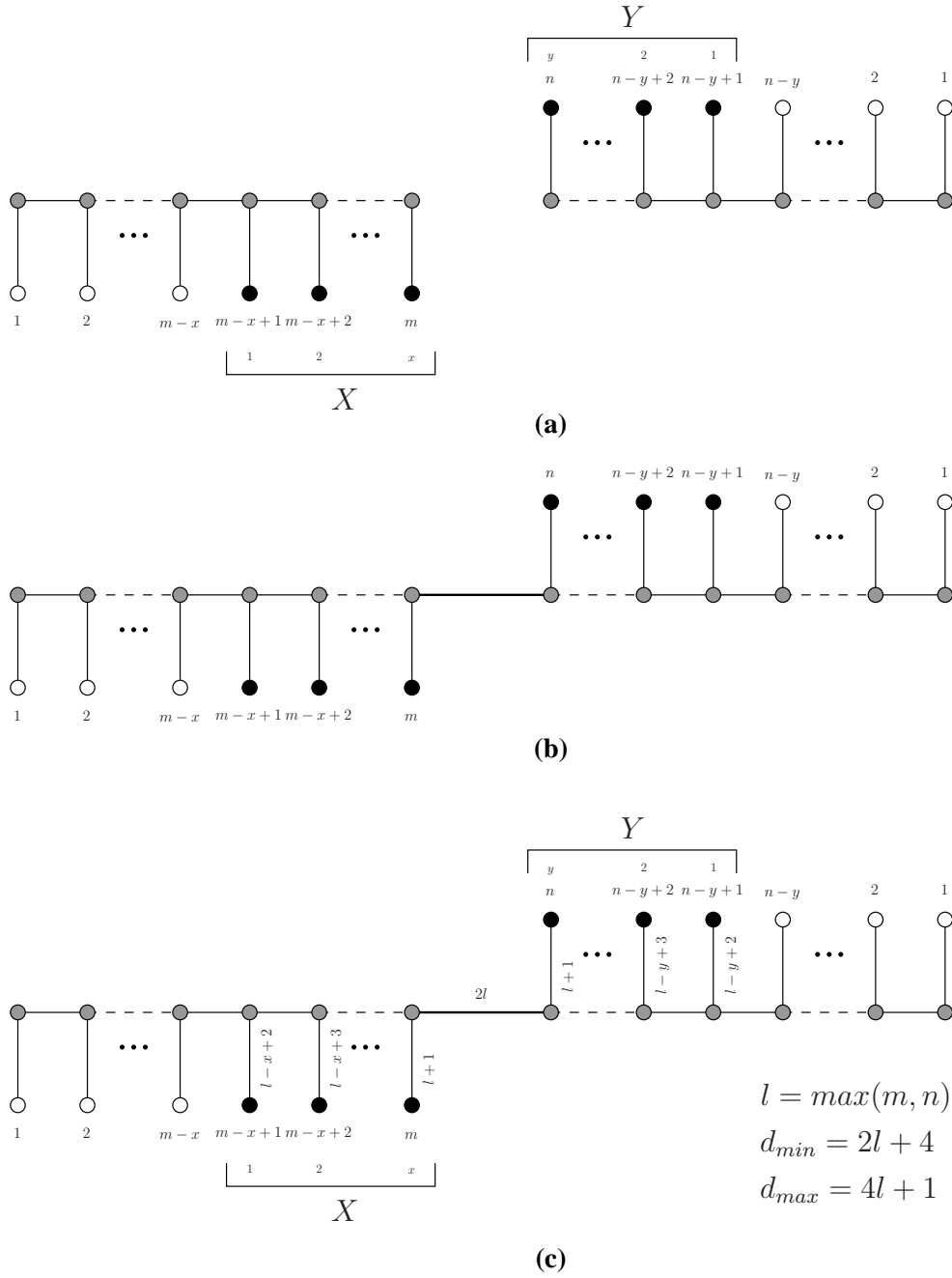


Figure 6.4: General Construction of *PCG* of the bipartite graph of Theorem 6.1.3.

connecting C_m and C_n through a linking edge as illustrated in Fig. 6.4(b). Now we give suitable weight to the edges of T such that T becomes the pairwise compatibility graph of G for a specific choice of d_{min} and d_{max} . Let, the weight of the linking edge is l_w and $l = \max\{m, n\}$. Then we take $l_w = 2l$. The weights of all the edges connecting the leaves to the spine of the caterpillar is 1 except for the leaves correspond to X and Y . The weights of the edges connecting x vertices are $l+1, l, l-1, \dots, l-x+1$ where $l+1$ is the weight of the edge connected to the rightmost vertex and $l-x+1$ is for the leftmost vertex of X . And similarly the weights of the edges connecting y vertices are $l+1, l, l-1, \dots, l-y+1$. But here $l+1$ is the weight of the edge connected to the leftmost vertex and $l-x+1$ is for the rightmost vertex of Y . This weight allocation is illustrated in Fig. 6.4(c). Now if we take $d_{min} = 2l + 4$ and $d_{max} = 4l + 1$, the resulting tree (T) generates G . And hence we are done. If we apply this construction to the graph shown in Fig. 6.3, the resulting tree would be as illustrated in Fig. 6.5.

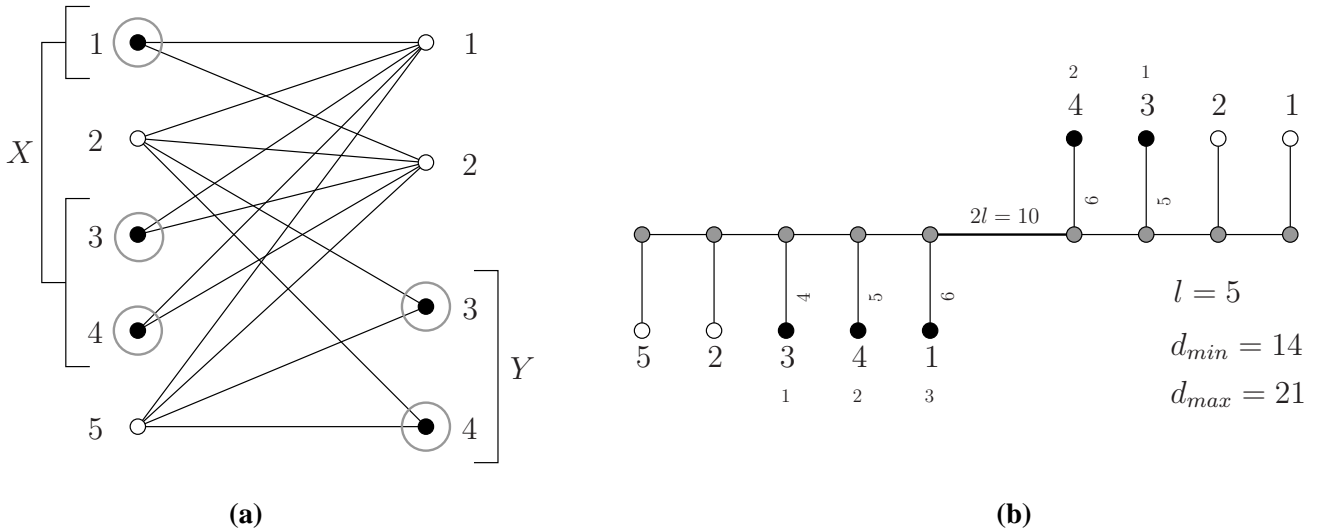


Figure 6.5: A bipartite graph as of Theorem 6.1.3 and it's PCG .

Now we take a look why these choices of weights serve our purpose. The distance between the first and the last leaf of C_m and C_n are $m+l+1$ and $n+l+1$ respectively [see figure]. Since $l = \max\{m, n\}$, the maximum possible distance between two leaves of the same caterpillar is $2l+1$. The distance between any two leaves of the same caterpillar should be out of the range.

So d_{min} must be greater than $2l + 1$. And the distance between the two leaves of different caterpillar that are connected in G must be within the range. Hence we take $l_w = 2l$. Now the distance between any two leaf u and v where $u \in X$ and $v \in Y$ is $(l + 1) + (l + 1) + 2l = 4l + 2$ which is greater than d_{max} . The maximum possible distance between two vertices that are connected in G is $l + 2l + (l + 1) = 4l + 1$ (distance between the corresponding leaves of the first vertex of the larger partite set and the last vertex of the smaller partite set). And the minimum distance is $2 + 2l + 2 = 2l + 4$ while $x = y = 1$ (distance between the $(m - 1)th$ leaf of m -partite set and 2^{nd} leaf of n -partite set). Fig. 6.4(c) illustrates all about these weight calculations.

Theorem 6.1.4 *A bipartite graph $K_{m,n}$ with two partite sets m and n (without loss of generality, we assume $m \geq n$) is a PCG if any vertex $u \in \{1, 2, \dots, m\}$ is connected to d_m or less no. of sequential vertices in n -partite set where $d_m = \max\{\text{degree}(u)\}; u \in \{1, 2, \dots, m\}$ such that if $\text{degree}(u) = d_m$ then it's neighbors are arranged sequentially from any position in n -partite set, otherwise it's neighbors must be the first or last $\text{degree}(u)$ number of vertices of n -partite set.*

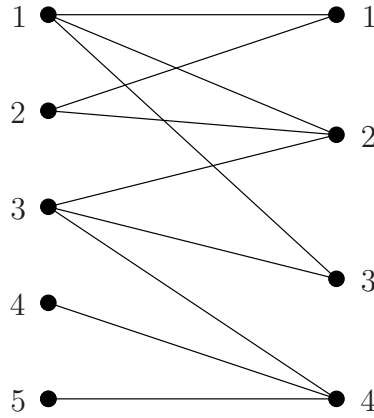


Figure 6.6: A special case of bipartite graph as of Theorem 6.1.4.

Fig. 6.6 illustrates an example of the bipartite graph as specified in Theorem. 6.1.4. Here $m = 5$, $n = 4$ and $d_m = 3$. Here $\text{degree}_m(1) = \text{degree}_m(3) = d_m$ (here $\text{degree}_m(i)$ denotes the degree of the i th vertex of m -partite set) and other vertices of m -partite set are of less degree. Two vertices 1 and 3 are connected to 3 sequential vertices of n -partite set having neighbors

$\{1, 2, 3\}$ and $\{2, 3, 4\}$. Again $\text{degree}_m(2) = 2$ which is less than d_m and its neighbors are the first two vertices of other partite set. Similarly, vertices 4 and 5 are of degree one and their neighbor is the last vertex of the n -partite set.

Now we proceed to the constructive proof of Theorem. 6.1.4. Let, $G = (V, E)$ be a bipartite graph ($K_{m,n}$). Now construct two caterpillars (C_m and C_n) with the vertices of two partite sets as the leaves of the caterpillars as illustrated in Fig. 6.7(a). Now we construct a single

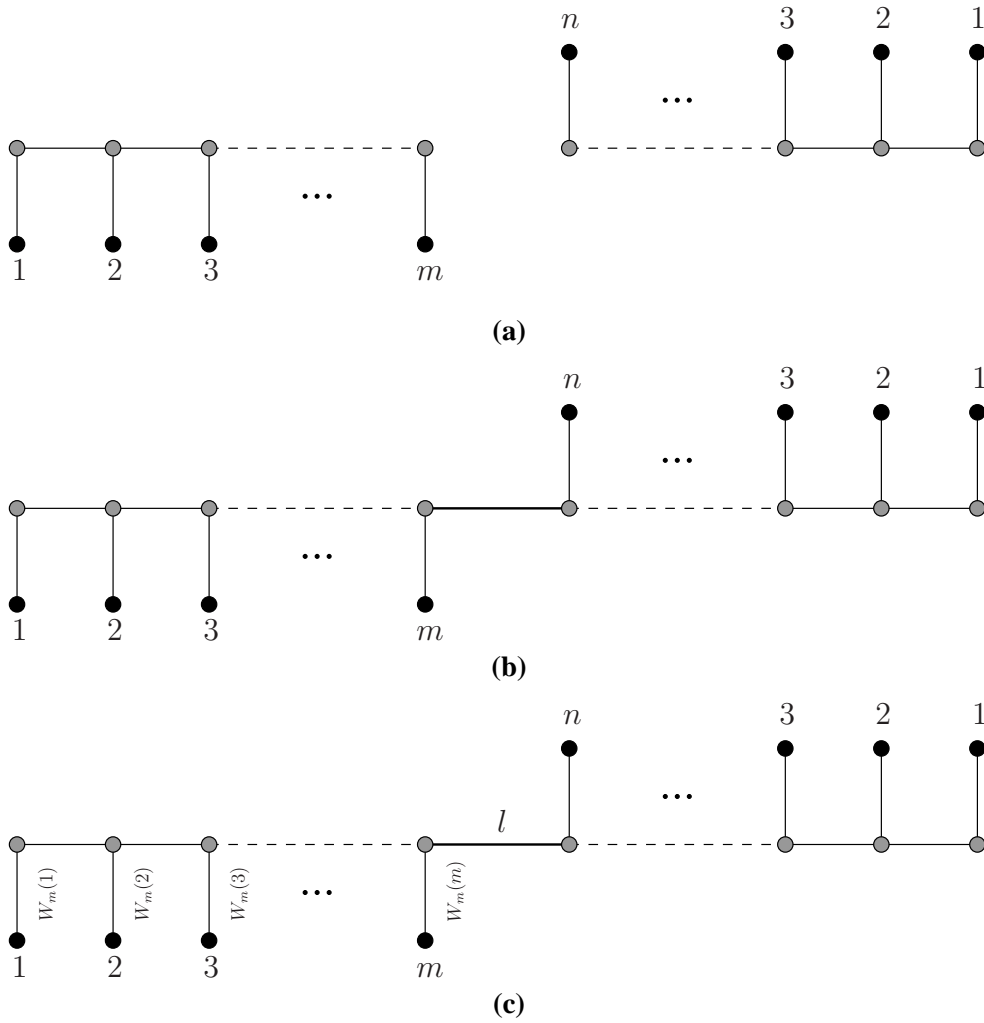


Figure 6.7: General Construction of *PCG* of the bipartite graph of Theorem 6.1.4.

tree (T) by connecting C_m and C_n through a linking edge as illustrated in Fig. 6.7(b). The weight of all the edges of C_n is one. Let, l , $W_m(i)$, $\text{label}(i)$ and $N_{\text{skip}}(i)$ denote the weight of the linking edge, the weight of the edge that connects the i th leaf of C_m to its spine, the label

(numbering) of the i th vertex and the number of vertices that has to be skipped to reach the first neighbor respectively. In Fig. 6.6, $N_{skip}(1) = 0, N_{skip}(3) = 1, N_{skip}(4) = 3$ etc. Now we define the following distances.

$$d'_{max} = m + n$$

$$d'_{min} = d'_{max} - (d_m - 1)$$

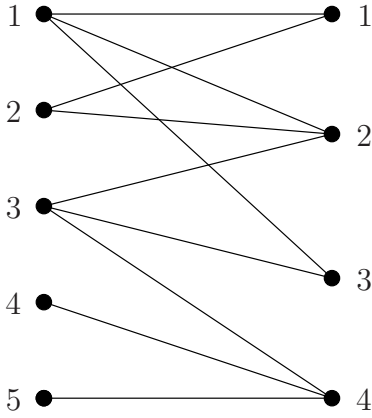
$$l = 2(m + n - 1) - d'_{min} + 1$$

$$d_{min} = d'_{min} + l$$

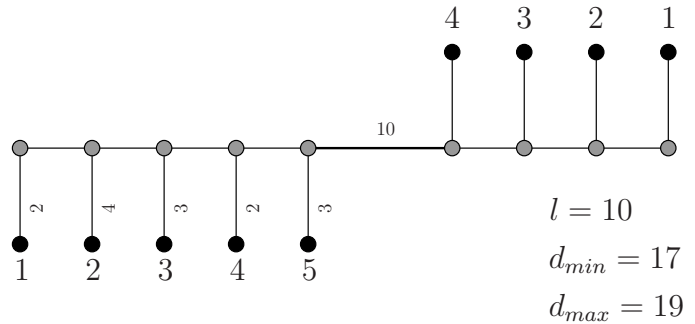
$$d_{max} = d'_{max} + l$$

$W_m(i)$ is defined as

$$W_m(i) = \begin{cases} d'_{min} - ((m - \text{label}(i)) + N_{skip}(i) + 1) & \text{if } \begin{array}{l} \text{degree}_m(i) = d_m(i) \text{ or } \text{degree}_m(i) < \\ d_m(i) \text{ and its neighbors are the first } \\ \text{degree}_m(i) \text{ vertices of } n\text{-partite set} \end{array} \\ d'_{max} - (\text{degree}_m(i) - 1) - (m - \text{label} + 1) & \text{otherwise} \end{cases}$$



(a)



(b)

Figure 6.8: A specific example of graph as of Theorem 6.1.4 and its *PCG*.

The pairwise compatibility tree T of the graph G shown in Fig. 6.6 is illustrated in Fig. 6.8.

Now we take a look at these weight calculations to see how they come. Here, without loss of generality, we assume that $m \geq n$. d'_{min} and d'_{max} do not take the weight of the linking edge

in account. For the same reason as stated in Theorem. 6.1.3, d_{min} must be greater than the maximum possible distance between the two leaves of C_m . And l should be tuned accordingly so that distance between two connected vertices is greater than the maximum possible distance between the two leaves of C_m . The maximum possible distance between the two leaves of C_m is the distance between leaf 1 and m when they get their maximum possible weight. Again they get their maximum weight when they are connected only to the first vertex the n -partite set. Using the formula $W_m(i) = d'_{max} - (degree_m(i) - 1) - (m - label + 1)$ as stated in Eqn. 6.1 we get $W_m(1) = m + n - (1 - 1) - (m - 1 + 1) = n$ and $W_m(m) = m + n - (1 - 1) - (m - m + 1) = m + n - 1$ that is maximum possible weight of leaf 1 and m are n and $m + n - 1$ respectively. Now maximum possible distance between two vertices of the same caterpillar is

maximum distance between leaf 1 and m

$$= \text{maximum weight of leaf 1} + \text{maximum weight of leaf } m + (m - 1)$$

$$= n + (m + n - 1) + (m - 1)$$

$$= 2(m + n - 1). \text{ Then } d_{min} \text{ must be greater than } 2(m + n - 1) \text{ and we take } d_{min} = 2(m + n - 1) + 1.$$

Now $l = d_{min} - d'_{min} = 2(m + n - 1) - d'_{min} + 1$. In Eqn. 6.1 the calculation is such that if $degree_m(i) = d_m(i)$ or $degree_m(i) < d_m(i)$ and its neighbors are the first $degree_m(i)$ vertices of n -partite set then the distance to it's first neighbor is d_{min} and otherwise the distance to it's last neighbor is d_{max} .

6.2 Conclusion

In this chapter we characterize some classes of graphs as *PCG* and also give constructive proof of our theorems.

Chapter 7

Improper PCG

In this chapter we introduce two new concepts that we call *redundancy* and *improper PCG*. We also identify some properties associated with improper *PCG*. We have proved that every graph is improper *PCG*. We also give a heuristic algorithm to construct improper pairwise compatibility tree of triangular planar graphs.

7.1 Improper PCG

When there is multiple existence of the same leaf then we call the extra leaves as redundancy. A graph G is an improper *PCG* if the corresponding pairwise compatibility tree can be constructed introducing some redundancies. Fig. 7.1(b) shows an example of redundancy and improper *PCG* corresponding to the graph shown in Fig. 7.1(a). Here the node labeled with number 6 appears twice, one of which is called redundancy (enclosed with grey circle). We denote the number of redundancies as n_{rd} . In an improper *PCG*, $n_{rd} \geq 0$. Obviously, a *PCG* is also an improper *PCG* but the reverse is not necessarily true.

Now we proceed to the fact that every graph is an improper *PCG*.

Theorem 7.1.1 *Every graph is an improper PCG.*

Let, $G = (V, E)$ be any graph. First, we construct a star for each vertex $u \in V$ where the corresponding vertex and its neighbors are the leaf of the star. The weight of the edge

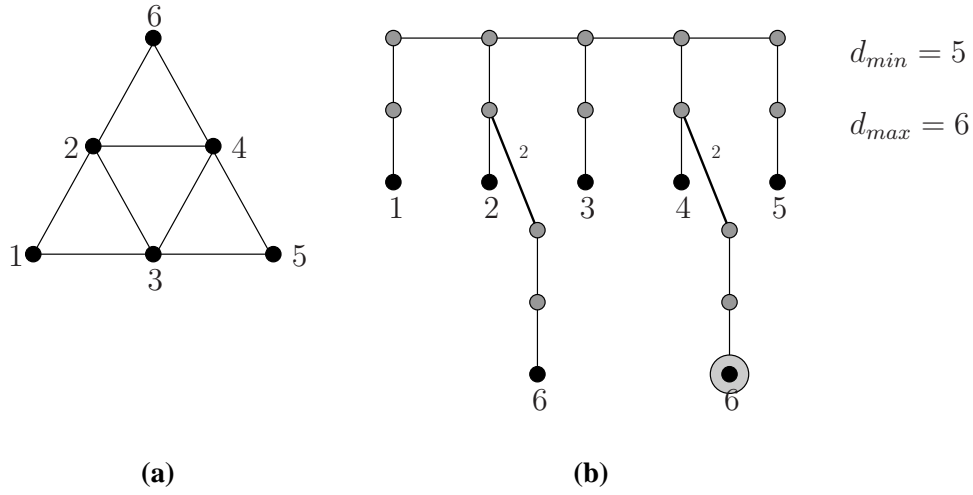
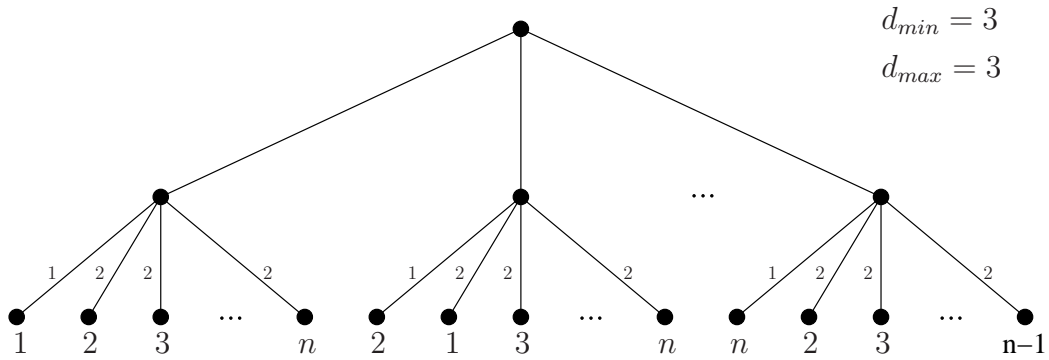


Figure 7.1: An improper PCG of a graph.

connecting u to the base of the star is 1 whereas the other edges incident to its neighbors are of weight 2. Then we connect all the bases of the stars to a single node through edges of weight 2. Now if we take $d_{min} = d_{max} = 3$, we are done.

Fig. 7.2 illustrates the pairwise compatibility tree of K_n constructed by this method and this is also the worst case of our construction where the number of leaves required is n^2 for a graph with n vertices. And the total number of nodes is $n^2 + n + 1$ (including leaf and all other intermediate node).

Figure 7.2: An improper PCG for any graph of n vertices.

For better performance, while we are constructing the star for a vertex $u \in V$, we can exclude its neighbor $v \in V$ if the corresponding star of v is already been constructed. Because the connectivity of u and v is already taken in account in the star corresponding to v . That is,

if we number the vertices of G numerically and construct star for the lowest numbered vertex first, then for the next lowest and so on, then we can exclude all lower numbered neighbors of any vertex while constructing the corresponding star. Fig. 7.3 illustrates the tree constructed by this method for K_n . Here total number of leaves is, $n + (n-1) + (n-2) + \dots + 2 = n(n-1)/2 - 1$ and total number of nodes is $n(n-1)/2 - 1 + (n-1) + 1 = (n-1)(n+2)/2$.

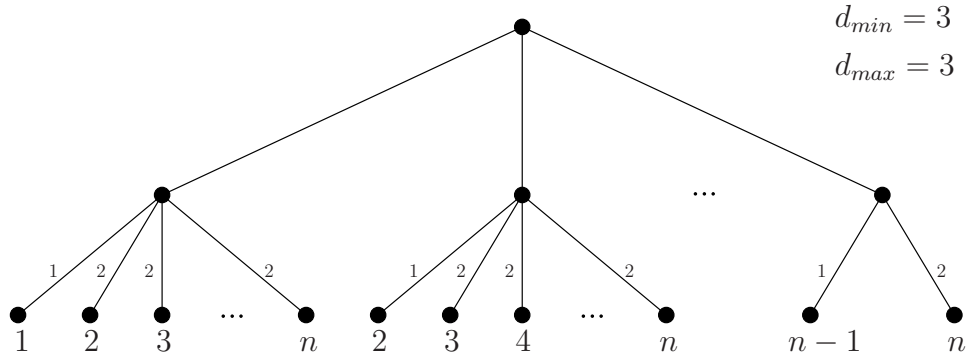


Figure 7.3: Improved construction of improper PCG for graphs with n vertices.

Before proceed to our heuristic algorithm we discuss some special cases of planar triangular graphs for which pairwise compatibility graph can be constructed without any redundancy. The pairwise compatibility graph of just a triangle is a caterpillar as illustrated in Fig. 7.4(a), (b) where $d_{min} = 3$ and $d_{max} = 4$. Now we consider our first case which we call *case1* and here we merge arbitrarily large number of triangles such that two consecutive triangles share an edge and the vertices of the graph can be numbered in such a way that the endpoints of an edge shared by two triangles get consecutive number, i.e, for an shared edge uv , if u get label n then the label of v is $n-1$ or $n+1$. We call this sort of numbering, *sequential numbering*

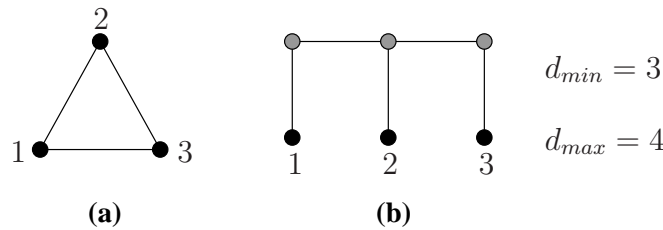


Figure 7.4: A triangle graph and its PCG .

Fig. 7.5 shows an example of this sort of graphs. Now we can merge the individual caterpil-

lars for each of the triangle as shown in Fig. 7.4(b) and we get the tree as in Fig. 7.5(b). Choice for d_{min} and d_{max} remains the same as it was for a single triangle, i.e, 3 and 4 respectively.

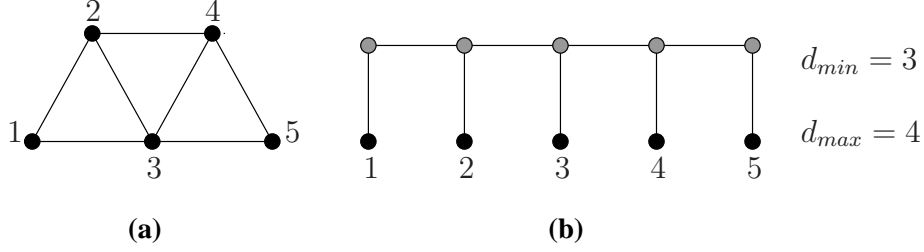


Figure 7.5: A special case (*case 1*) of triangular graph and it's *PCG*.

Now we consider another case which we call *case 2*. We can extend a triangular planar graph by merging arbitrarily large number of triangles such that they do not share any edge rather they share only one vertex as shown in Fig. 7.6(a).

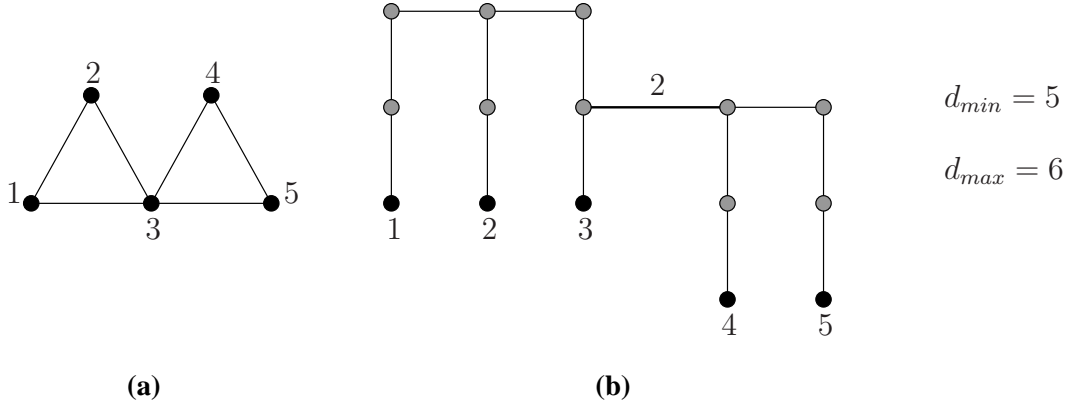


Figure 7.6: A special case (*case 2*) of triangular graph and it's *PCG*.

Fig. 7.6(b) shows the corresponding pairwise compatibility tree. Weight of all the edges are 1 except those that are labeled as 2. Here $d_{min} = 5$ and $d_{max} = 6$. Here we can see that $d(1, 2) = d(2, 3) = d(3, 4) = 5$, $d(1, 3) = 6$ that are within the range but $d(1, 4) = 9$ which is expectedly out of the range. We shall use these two cases (denoted by *case 1* and *case 2*) in our heuristic algorithm described in section.

Now we consider both the cases simultaneously. Fig. 7.7(a) shows an example of such a graph where two triangles can share either a single vertex or an edge and for the later case the shared edge is such that it's endpoints are labeled with consecutive numbers (as described

above). Now merging the corresponding construction processes of these two cases, we can construct the pairwise compatibility tree and Fig. 7.7(b) illustrates such construction. Here $d_{min} = 5$ and $d_{max} = 6$. Measuring the distances between different leaves, the validity of such construction can be easily verified. Hence we characterize a restricted class of planar triangular graph, that fall in either of the above two cases or both the cases simultaneously, as *PCG*.

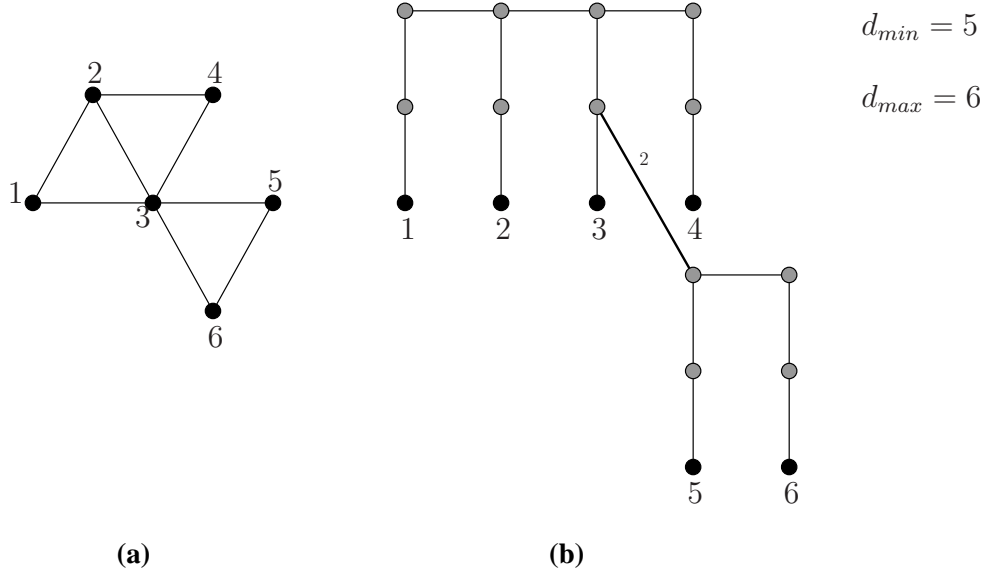


Figure 7.7: Combination of *case 1* and *case 2*.

Now we give an heuristic algorithm to construct pairwise compatibility tree of any planar triangular graph allowing redundancies ($n_{rd} \geq 0$).

Let, $G = (V, E)$ be an planar triangular graph. First, we construct the internal dual $G^* = (V^*, E^*)$. Now we have to traverse G^* starting from an arbitrary vertex $u \in V^*$. While moving from one u to v where $u, v \in V^*$, we cross an corresponding edge $e \in E$. If e has sequential numbering then we add the tree of the corresponding face of v (as was *case 1*) as in Fig. 7.5. Otherwise we delete one of the two edges (e_1, e_2) other than e of the corresponding face of v . If $e_i, i \in 1, 2$ is not shared with any other face then e_i is the edge to be deleted. If both the edges are shared with other face or not, then any one of e_1 and e_2 can be deleted. If any one or both of e_1 and e_2 is shared then after deletion of one, the next face on the traversal is connected as *case 2* as illustrated in Fig. 7.6. If none of e_1 and e_2 is shared then after deletion of one we have just one edge connected to the face and can extend the tree as in Fig. 7.7. In

this way we traverse the entire dual graph and build the tree incrementally.

Now we have to consider the edges that was deleted. Let, $uv \in E$ be an deleted edge. Then either one or both of the endpoints are already present in the tree. we choose the leaf corresponding to u or v that is present in the tree and connect the other one to the parent of the earlier. Fig. 7.8 illustrates this procedure.

7.2 Conclusion

In this chapter we describe our notion of *improperPCG* and also present our findings associated with it.

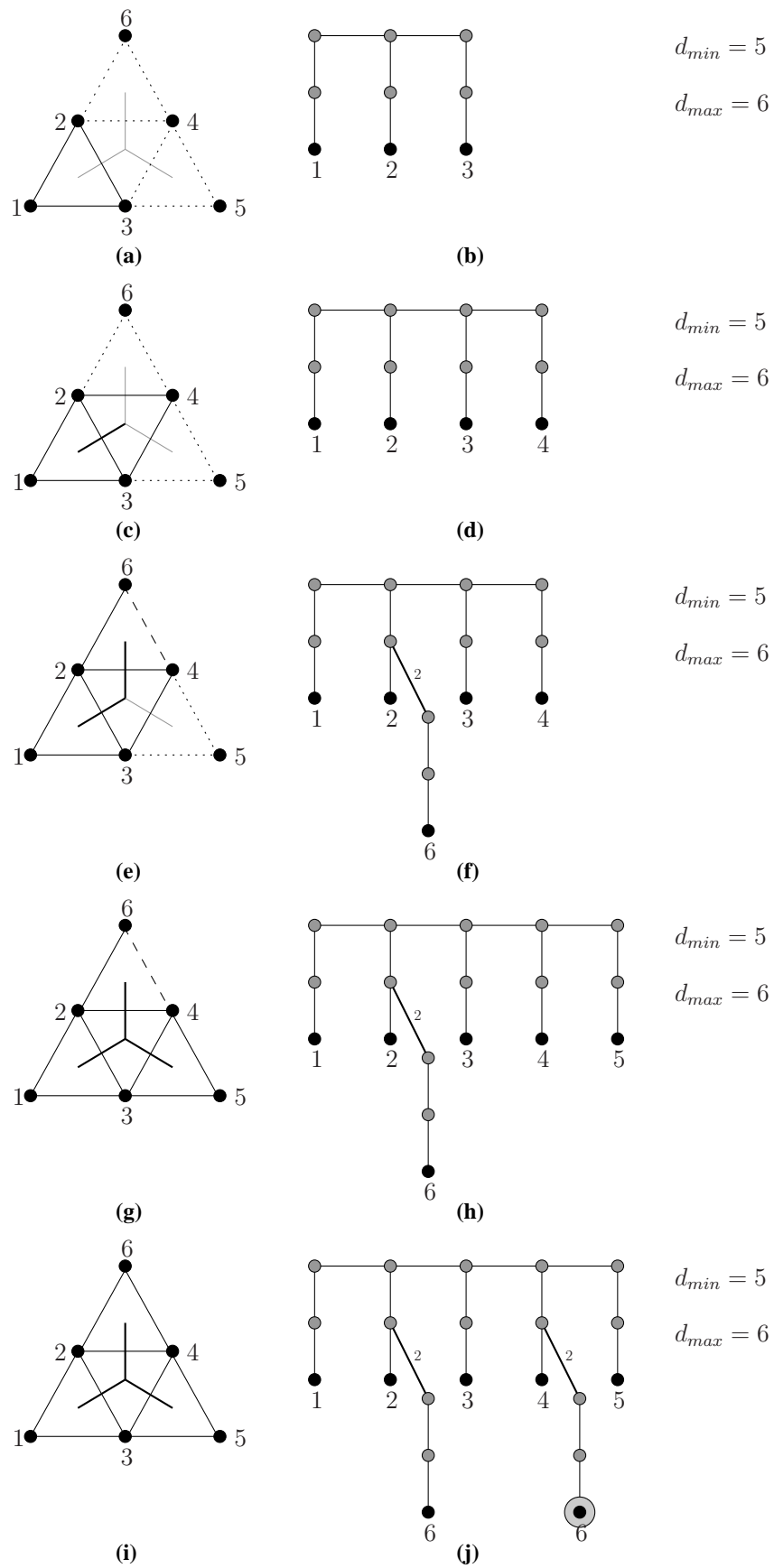


Figure 7.8: Construction of the PCG according to the heuristic algorithm.

Chapter 8

Conclusion

This thesis focuses on the utilization of the graph theory in bioenergetics. This thesis is organized around two important field of bioinformatics : haplotyping and phylogeny.

Haplotyping can be done in many approaches. Most of our work in this field was devoted to comparing two algorithms to generate haplotype for an individual from SNP fragments. We compare a heuristic algorithm based on minimum error correction with a genetic algorithm and qualify the former as superior.

In this thesis we were mainly concerned with pairwise compatibility graphs. We have established some important classes of graph as *PCG*. We also provide efficient construction method to construct the trees that can generate those *PCGs*. Moreover, we introduce the notion of *improperPCG* and *redundancy* and also unveil the fact that every graph is an improper *PCG*. The construction methods that we provided in this thesis is very much generic and hence offer great opportunity and flexibility to utilize them in recognizing more classes of graph as *PCG*.

Identifying the boundary of *PCG* graphs i.e. which graphs are *PCG* and which are not is still to be done and we think that it would be an interesting research area. We left it as a future work and interested researchers may explore this fascinating area of pairwise compatible graph.

References

- [ACJ03] N. Amenta, F. Clarke, and K. St. John, *A linear-time majority tree algorithm*, in Benson and Page [BP03], pp. 216–227.
- [BILR05] V. Bafna, S. Istrail, G. Lancia, and R. Rizzi, *Polynomial and apx -hard cases of the individual haplotyping problem*, Theoretical Computer Science **335** (2005), no. 1, 109–125.
- [BP03] G. Benson and R. D. M. Page (eds.), *Algorithms in bioinformatics, third international workshop, wabi 2003, budapest, hungary, september 15-20, 2003, proceedings*, Lecture Notes in Computer Science, vol. 2812, Springer, 2003.
- [BVDL03] P. Bonizzoni, G. D. Vedova, R. Dondi, and J. Li, *The haplotyping problem: An overview of computational models and solutions*, Journal of Computer Science and Technology **18** (2003), no. 6, 675–688.
- [CLRS01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*, 2nd ed., The MIT Press, 2001.
- [CvIKT05] R. Cilibrasi, L. van Iersel, S. Kelk, and J. Tromp, *On the complexity of several haplotyping problems*, WABI (R. Casadio and G. Myers, eds.), Lecture Notes in Computer Science, vol. 3692, Springer, 2005, pp. 128–139.
- [Hüf05] F. Hüffner, *Algorithm engineering for optimal graph bipartization*, WEA (S. E. Nikolettseas, ed.), Lecture Notes in Computer Science, vol. 3503, Springer, 2005, pp. 240–252.

- [HY07] K. S. M. T. Hossain and M. N. Yanhaona, *Pairwise compatibility graphs*, B. Sc. Thesis, Bangladesh University of Engineering and Technology, June 2007.
- [JP04] N. C. Jones and P. A. Pevzner, *An introduction to bioinformatics algorithms*, The MIT Press, Cambridge, 2004.
- [KMP03] P. E. Kearney, J. I. Munro, and Derek Phillips, *Efficient generation of uniform samples from phylogenetic trees*, in Benson and Page [BP03], pp. 177–189.
- [Les02] A. M. Lesk, *Introduction to bioinformatics*, Oxford University Press, Great Clarendon Street, Oxford, 2002.
- [MM81] T. Margush and F. R. McMorris, *Consensus n -trees*, Bulletin of Mathematical Biology **43** (1981), no. 2, 239–244.
- [NC88] T. Nishizeki and N. Chiba, *Planar graphs : theory and algorithms*, North-Holland mathematics studies, vol. 140 / 32, North-Holland, Dordrecht, Amsterdam, 1988.
- [NR04] T. Nishizeki and M. S. Rahman, *Planar graph drawing*, Lecture Notes Series on Computing, vol. 12, World Scientific Publishing Company, Singapore, September 2004.
- [Phi02] D. Phillips, *Uniform sampling from phylogenetics trees*, Master’s thesis, University of Waterloo, August 2002.
- [PX94] P. M. Pardalos and J. Xue, *The maximum clique problem*, Journal of Global Optimization **4** (1994), no. 3, 301–328.
- [Wes00] D. B. West, *Introduction to graph theory*, Prentice Hall, 2000.
- [WWLZ05] R. S. Wang, L. Y. Wu, Z. P. Li, and X. S. Zhang, *Haplotype reconstruction from snp fragments by minimum error correction*, Bioinformatics **21** (2005), no. 10, 2456–2462.

Index

- algorithm
 - complexity, 45
 - constant time, 46
 - exponential, 45
 - heuristic, 60
 - linear, 45
 - linear time, 45
 - non-polynomial, 45
 - polynomial, 45
 - polynomially bounded, 45
- allele, 18, 23
 - bi-allelic, 18
 - heterozygous, 19
 - homozygous, 19
- asymptotic behavior, 45
- base, 8
- bioinformatics, 2, 7
 - areas, 2
- BLAST, 3
- cell, 7
 - prokaryotic, 8
- chordless, 39
- chromosome, 9
- clique, 38
- conflict, 21
- cycle
 - chordless, 39
- data mining, 5
- diploid, 19
- DNA, 8
 - gene, 8
 - replication, 10
 - sequencing, 11
- edge, 40
 - crossings, 43
 - dual, 43
 - multiple, 41
 - weighted tree, 37, 44
- face, 39
- gain, 26
- gap, 20
- gapless, 20
- GenBank, 3
- genetic algorithm, 29
- genetic information, 8

- genome, 9
 - annotation, 11
 - sequence, 3
- genomics, 9
- graph, 40
 - bipartite, 41
 - complete, 41, 47
 - complete k-partite, 41
 - dual, 43
 - k-partite, 41
 - planar, 43
 - plane, 43
 - simple, 41
 - triangular planar, 43, 56
- haplotype, 11, 19
 - construction, 22
- haplotyping, 19
 - individual, 19
 - population, 19
- HMEC, 23
- hole, 20
- Human Genome Project, 4
- integer, 45
- loop, 41
- machine learning, 4
- marker, 16
- maximum cumulative gain, 27
- MEC, 22
- NCBI, 3, 4
- NP-complete, 38
- nucleotide, 8
- nucleus, 7
- pairwise
 - compatibility graph, 44
 - compatibility tree, 38
- Pairwise compatibility graph, 36
- PCG, 37
 - improper, 39, 56
- PDB, 3
- personalized drug prediction, 16
- phylogenetic tree, 14
 - character-based, 14
 - cladistic, 14
 - distance-based, 14
 - phenetic, 14
- phylogeny, 13, 36
- planar embedding, 43
- protein synthesis, 10
- redundancy, 39, 56
- RNA, 10
- search heuristic, 26
- set
 - independent, 41
 - partite, 41

simple graph, 41

SNP, 15, 18

SNP matrix, 20

star, 42, 56

taxa, 14

tree, 42

 edge-weighted, 44

triangulated, 39