M.Sc. Engg. Thesis

On Pairwise Compatibility Graphs

By
Md. Shamsuzzoha Bayzid

Submitted to
Department of Computer Science and Engineering
in partial fulfilment of the requirements for the degree of
Master of Science in Computer Science and Engineering

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

June 5, 2010

The thesis titled "**On Pairwise Compatibility Graphs**", submitted by Md. Shamsuzzoha Bayzid, Roll No. 040805019P, Session April 2008, to the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, has been accepted as satisfactory in partial fulfillment of the requirements for the degree of Master of Science in Computer Science and Engineering and approved as to its style and contents. Examination held on June 5, 2010.

Board of Examiners

Candidate's Declaration

This is to certify that the work presented in this thesis entitled "**On Pairwise Compatibility Graphs**" is the outcome of the investigation carried out by me 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.

Md. Shamsuzzoha Bayzid
Candidate

Contents

List of Figures

Acknowledgements

First of all, I would like to declare that all the appraisals belong to the Almighty **ALLAH**.

I would like to express my deep gratitude to my supervisor Professor Dr. Md. Saidur Rahman for introducing me to the fascinating and prospective field of bioinformatics and pairwise compatibility graphs. I have learned from him how to carry on a research work, how to write, speak and present well. I thank him for his patience in reviewing my so many inferior drafts, for correcting my proofs and language, suggesting new ways of thinking, leading to the right way, and encouraging me to continue my research work. I again express my indebtedness, sincere gratitude and profound respect to him for his continuous guidance, suggestions and whole hearted supervision throughout the progress of this work.

I would like to thank every one associated with our Graph Drawing Research Group; in particular, to Muhammad Nur Yanhaona and Md. Jawaherul Alam. I convey my heartfelt reverence to my parents and other family members for giving their best support throughout my work to overcome the tedium of repetitive trials to new findings.

Finally, every honor and every victory on earth is due to Allah, descended from Him and must be ascribed to Him. He has endowed me with good health and with the capability to complete this work. I deeply express my sincere gratitude to the endless kindness of Allah.

Abstract

This thesis deals with pairwise compatibility graphs. Let T be an edge weighted tree, let $d_T(u, v)$ be the sum of the weights of the edges on the path from u to v in T, and let d_{min} and d_{max} be two non-negative real numbers. Then a pairwise compatibility graph of T for d_{min} and d_{max} is a graph $G = (V, E)$, where each vertex $u' \in V$ corresponds to a leaf u of T and there is an edge $(u', v') \in E$ if and only if $d_{min} \leq d_T(u, v) \leq d_{max}$. A graph G is called a pairwise compatibility graph (PCG) if there exists an edge weighted tree T and two non-negative real numbers d_{min} and d_{max} such that G is a pairwise compatibility graph of T for d_{min} and d_{max}. Pairwise compatibility graph is descended from phylogenetic tree which expresses the evolutionary relationships among different species.

Pairwise compatibility graph is not only important for phylogeny applications, but has also given rise to a number of interesting theoretical problems. Unfortunately, however, very few significant results regarding this problem are known to date. Complete characterization of this graph class is not known. It has not been possible yet to give an algorithm for recognizing whether a given graph is pairwise compatibility graph or not. Moreover, the computational complexity of recognizing pairwise compatibility graphs is still unknown. This thesis addresses different important and challenging theoretical problems regarding pairwise compatibility graphs. We deal with the open question regarding whether or not all graphs are pairwise compatibility graphs and settle the corresponding conjecture, which states that every graph is a pairwise compatibility graph, in negative. We also recognize several well known large graph classes as pairwise compatibility graphs. We introduce the new notion of improper PCG, and show that every graph is an improper PCG. In addition, we also present an efficient heuristic algorithm to construct an improper pairwise compatibility tree of any triangulated plane graph.

Chapter 1

Introduction

One of the biggest achievements in the quest for mystery of life is the complete sequencing of human genome. Over the past few decades, major advances in genomic and other molecular research technologies and developments in information technologies have combined to produce an explosive amount of information related to molecular biology. This inundation of genomic data has led to an absolute requirement for computerized techniques to store, organize, and index the information and necessitated the use of specialized algorithms for their effective visualization and analysis. The union between these two subjects is largely attributed to the fact that, life itself is an information technology and an organism's physiology is largely determined by its genes, which at its most basic can be viewed as digital information. This intersection point of information technology and molecular biology has given rise to a new era of science – *bioinformatics* and *computational biology*.

Phylogeny is one of the most widely studied areas of bioinformatics and computational biology. Scientific euphoria has recently centered on the reconstruction of evolutionary relationships among different species. It is assumed and supported by evidences that all life currently on earth is descended from a single common ancestor. Over a period of at least 3.8 billion years, that single original ancestor has split repeatedly into new and independent lineages, i.e., species. On occasion, some of these independent lineages have come back together to form yet other lineages or to exchange genetic information. The evolutionary relationships among these species are referred to as "Phylogeny", and phylogenetic reconstruction is concerned with inferring the phylogeny of groups of organisms. These relationships are expressed as a tree known as phylogenetic tree. Figure 1.1 illustrates a phylogenetic tree. This example has been taken from a presentation titled *Introduction to Phylogenetic Estimation Algorithms* [W09]. The tree is rooted at the most recent common ancestor of the five existing species represented by five leaves. Other internal nodes represent hypothesized or known ancestors. The common practice today is to use biomolecular sequences as representatives of the species set; which is why the nodes of this tree is labeled by DNA sequences. Other than the molecular sequences, morphological data (e.g., color, size, weight etc.), molecular markers (Single Nucleotide Polymorphism (SNP), haplotypes etc.), and gene order and content can be used as the representatives of the species set. These are commonly known as *character data*. Pairwise compatibility graph (PCG) (as defined in Section 1.1) is derived from phylogenetic trees which 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]. Although it is relatively a new area of research, the blend of graph theory and computational biology has attracted much attention from researchers.

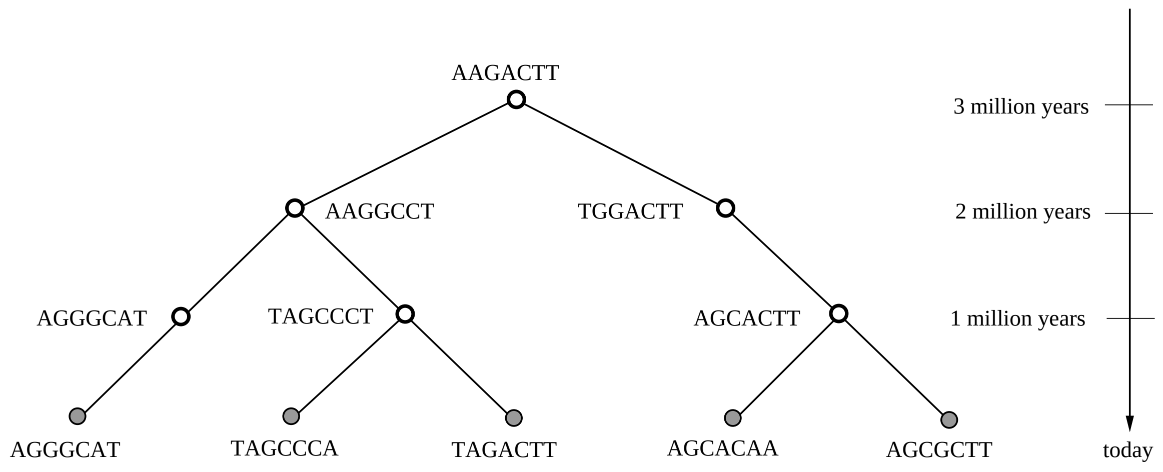

Figure 1.1: A phylogenetic tree [W09].

Since the inception of pairwise compatibility graphs, several interesting problems have been posed in front of us, and hitherto most of these problems have remained unsolved. Among the others, identifying different graph classes as pairwise compatibility graphs is an important concern. Although overlapping of pairwise compatibility graphs with many well-known graph classes like chordal graphs and complete graphs is quite apparent; slight progresses have been made on establishing concrete relationships between pairwise compatibility graphs and other known graph classes. Phillips has shown that every graph of five vertices or less is a PCG [P02] and Yanhaona *et al.* have shown that all cycles, cycles with a single chord, and cactus graphs are PCGs [YHR09]. Seeing the exponentially increasing number of possible tree topologies for large graphs, the proponents of PCGs conceived that all graphs are PCGs [KMP03]. In this thesis we refute the conjecture by showing that not all undirected graphs are PCGs. We have found that the placement of a vertex of a graph as a leaf in an edge weighted tree imposes constraints on the placements of the remaining vertices, and as a result, in some cases, it becomes infeasible to have a tree that generates the graph as a PCG, even when there may exist an exponential number of tree topologies for the given number of graph vertices.

Graph roots and powers have been studied extensively in graph theory. A graph $G' = (V', E')$ is a *k-root* of a graph $G = (V, E)$ if $V' = V$ and there is an edge $(u, v) \in E$ if and only if the length of the shortest path from u to v in G' is at most k. G is called the *k-power* of G' [LKJ00]. A special case of graph power is tree power, which requires G' to be a tree. Given a graph $G = (V, E)$, the *tree k-root problem* asks to construct a tree T on V such that $(u, v) \in E$ if and only if $d_T(u, v) \leq k$. Tree power graphs and their extensions (*Steiner k-power* graphs, *phylogenetic k-power* graphs, etc.) are by definition similar to pairwise compatibility graphs. However, the exact relationship of these graph classes with pairwise compatibility graphs was unknown. In this thesis, we have also investigated the possibility of the existence of such a relationship, and have discovered that tree power graphs and some of their extensions are in fact pairwise compatibility graphs. Such a relationship may serve the purpose of not only unifying related graph classes but also utilizing the method of tree constructions for one graph class in another.

In the rest of this chapter, we provide the necessary background and objectives for this study on pairwise compatibility graphs. In Section 1.1 we describe the pairwise compatibility graphs.

We discuss the applications of phylogenies in Section 1.2. We continue in Section 1.3 with a discussion of the application of pairwise compatibility graphs. We devote Section 1.4 for the literature review, and we detail the objective of this thesis in Section 1.5. Finally, Section 1.6 is a summary of this work and Section 1.7 is the description of the organization of this thesis.

1.1 Pairwise Compatibility Graphs

Let T be an edge weighted tree and let d_{min} and d_{max} be two non-negative real numbers such that $d_{min} \leq d_{max}$. A *pairwise compatibility graph of T for d_{min} and d_{max}* is a graph $G = (V, E)$, where each vertex $u' \in V$ represents a leaf u of T and there is an edge $(u', v') \in E$ if and only if the distance between u and v in T lies within the range from d_{min} to d_{max}. T is called the *pairwise compatibility tree* of G. We denote a pairwise compatibility graph of T for d_{min} and d_{max} by $PCG(T, d_{min}, d_{max})$. A graph G is a *pairwise compatibility graph* (PCG) if there exists an edge weighted tree T and two non-negative real numbers d_{min} and d_{max} such that $G = PCG(T, d_{min}, d_{max})$. Figure 1.2(a) depicts an edge weighted tree T and Fig. 1.2(b) depicts a pairwise compatibility graph G of T for $d_{min} = 4$ and $d_{max} = 7$; there is an edge between a' and b' in G since in T the distance between a and b is six, but G does not contain the edge (a', c') since in T the distance between a and c is eight, which is larger than seven. It is quite apparent that a single edge weighted tree may have many pairwise compatibility graphs for different values of d_{min} and d_{max}. Likewise, a single pairwise compatibility graph may have many trees of different topologies as its pairwise compatibility trees. For example, the graph in Fig. 1.2(b) is a PCG of the tree in Fig. 1.2(a) for $d_{min} = 4$ and $d_{max} = 7$, and it is also a PCG of the tree in Fig. 1.2(c) for $d_{min} = 5$ and $d_{max} = 6$.

In the realm of pairwise compatibility graphs, two fundamental problems are the "tree construction problem" and the "pairwise compatibility graph recognition problem". Given a PCG G, the *tree construction problem* asks to construct an edge weighted tree T, such that G is a pairwise compatibility graph of T for suitable d_{min} and d_{max}.

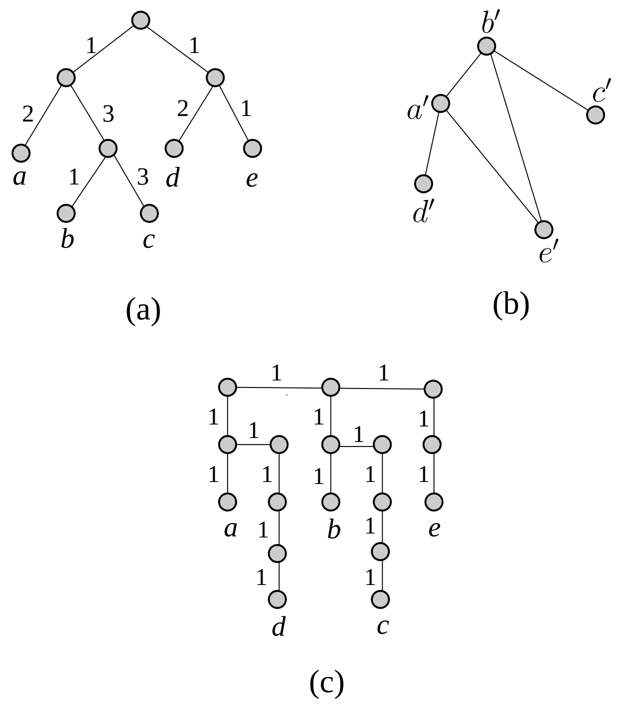

Figure 1.2: (a) An edge weighted tree T_1, (b) a pairwise compatibility graph G, and (c) an edge weighted tree T_2.

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 there is 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* seeks the answer – whether or not a given graph is a *PCG*.

Knowing that not all graphs are *PCG*s (see Chapter 3) we venture to relax the notion of pairwise compatibility graphs that leads to a new notion which we call "improper *PCG*". A graph G is an *improper PCG* if the corresponding pairwise compatibility tree can be constructed by allowing multiple existences of the same leaf; we call these extra leaves *redundancies*.

1.2 Applications of Phylogenies

In this section, we give a brief overview of phylogeny and its applications from [LW05].

Phylogenies are reconstructed on the basis of character data, where a "character" is any feature of an organism that can have different states. A typical biological example of a character is a nucleotide position in a DNA sequence, with the character state being the particular nucleotide (A, G, C, T) in occupying that position. From a mathematical standpoint, a character is just a function that maps the set of taxa to its set of states. Molecular phylogenetics research is concerned not only with the evolutionary history of different organisms, but also with how the different characters evolve in the course of that history.

The uses of phylogenies, beyond elucidating the evolutionary relationships of biological species, are many and growing; here we highlight the most common uses and some of the most intriguing ones.

The most common use of phylogeny is for comparative study [BM91, HP91]. A comparative study is one where a particular question is addressed by comparing how certain biological characters have evolved in different lineages in the context of a phylogeny. This information is used to infer important aspects of the evolution of those characters.

A second common use of phylogenies is to test biogeographic hypothesis. Biogeography is concerned with the geographical distribution of organisms, extant and extinct. For example, a researcher may be interested in whether a particular species have colonized a set of islands a single time or repeatedly. This can be assessed by determining whether all of the species on the island arose from a single most recent mainland common ancestor or whether they are multiple independent mainland species.

One can also use a phylogeny to attempt to infer the amino acid sequence of extinct proteins. This putative extinct proteins can then be synthesized or an artificial gene coding for them can be produced, and the functional characteristics of the proteins that are of interest can be tested.

In a more practical vein, phylogenies can be used to track the evolution of diseases, which can, in turn, be used to design drugs and vaccines that are more likely to be effective against the currently dominant strains. The most prominent example of this use is the flu vaccine, which is altered from year to year as medical experts work to keep track of the influenza types most likely to dominate in a given flu season [BBSCF99].

Finally, phylogenies have even been used in criminal cases, most famously, in a case where a doctor in Louisiana was accused of having deliberately infected his girlfriend with HIV [MMLPGH02]. The phylogenetic evidence featured prominently in the trial and the doctor was ultimately convicted of attempted second degree murder.

In summary, phylogenies are useful in any endeavor where the historical and hierarchical structure of the evolution of species can be used to infer the history of the point of interest.

1.3 Applications of PCGs

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, L02]. We have already discussed the uses of phylogenies. As its origin suggests, pairwise compatibility graphs can be used in parallel with, or at least to support, the phylogeny.

But the most intriguing potential of PCG lies in solving the "clique problem". Kearney *et al.* 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 [KMP03]. A *clique* in a graph is a set of pairwise adjacent vertices, or in other words, an "induced subgraph" which is a "complete graph". In the graph G of Figure 1.3, vertices a, b, c and d form a clique, because each has an edge to all the others. The size (number of vertices in the clique) of this clique is four. The *clique problem* is the problem of determining whether a graph contains a clique of at least a given size k. It is a well known NP-complete problem. The corresponding "optimization problem", the *maximum clique problem*, is to find the largest clique in a graph [CLRS03]. The clique problem is a classical problem in graph theory which finds important applications in different domains. It is a well known NP-complete problem that arises from many applications areas [PX94]. In many applications, the underlying problem can be formulated as a maximum clique problem while in some other applications, a subproblem

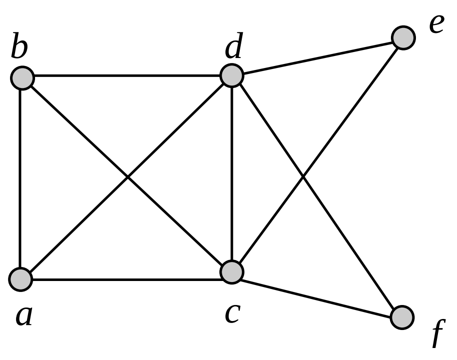

Figure 1.3: Example of a clique.

of the solution procedure consists of finding a maximum clique. It has applications in coding theory, geometry of tiling, problems arising from fault diagnosis, computer vision and pattern recognition [BBPP99]. 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.

PCG can be also associated with problems in distributed computing. *PCG* model the situation where a collection of processors is connected by a tree network and the processors representing the leaves of the tree network (T) are in consideration for a particular computation. A communication step is the amount of time required for a processor to pass information to a neighbor. Each communication step is represented by an edge of weight one. Then the pairwise compatibility graph $G = (T, d_{min}, d_{max})$ represents the possible flow of information among the processors in consideration during k communication steps where $d_{min} \leq k \leq d_{max}$. This application of *PCG* is motivated from the association of graph powers in distributed computing [L92].

1.4 Literature Review

The problems associated with *PCG* have not yet been extensively studied. Since their inception, pairwise compatibility graphs have raised several interesting problems, and most of these problems have remained unsolved. Among the other problems, identifying different graph classes as pairwise compatibility graphs is an important concern. Although overlapping of pairwise compatibility graphs with many well-known graph classes like complete graphs is quite apparent; slight progresses have been made on establishing concrete relationships between pairwise compatibility graphs and other known graph classes. The complete characterization of PCG is not known yet. This implies that 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 [P02]. 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 [YH07]. 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 is 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.

The most important question about PCG is whether every graph is a PCG; in other words, is there always a pairwise compatibility tree T for any arbitrary graph G? Seeing the exponentially increasing number of possible tree topologies for large graphs, the proponents of PCGs conceived that all undirected graphs are PCGs [KMP03]. But this question is still unresolved.

1.5 Objective of This Thesis

For any class of graphs, recognition is a fundamental problem. But in the present context, no examination of *PCG*s would be complete unless the question "Is every graph G a *PCG*?" is addressed. We have already mentioned that this question is still open although it is conjectured that all undirected graphs are *PCG*s. In this research we address the open question and give a negative answer by showing that not all undirected graphs are *PCG*s.

Discovering that not all graphs are pairwise compatibility graphs, we venture to identify which graph classes are *PCG*s and which are not. In this connection, we address some very important graph classes and establish their relationships with *PCG*. We show that tree power graphs and two of its extensions namely, phylogenetic k-power graphs and Steiner k-power graphs are *PCG*s. While proving that not all graphs are *PCG*s, we discover that not all bipartite graphs are *PCG*s. Then we try to find some restricted classes of bipartite graphs that are *PCG*s. Motivated by our experiences that constructing a pairwise compatibility tree is quite complex a task and, in some cases, impossible (which we will prove in Chapter 3), we introduce some flexibility in the concept of *PCG*s that turns into a new notion which we call improper *PCG*.

1.6 Summary of Results

In this thesis, we address the most important open problem regarding pairwise compatibility graphs, and also address the pairwise compatibility tree construction and pairwise compatibility graph recognition problems. The main results of this thesis are as follows.

1. We solve the open problem regarding whether or not every graph is a *PCG* by showing that not all graphs are *PCG*s. (See Section 3.1 for details.)

2. We recognize two subclasses of bipartite graphs as pairwise compatibility graphs. We develop linear-time algorithms for constructing the pairwise compatibility tree for any

graph of those classes. (See Section 3.2 for details.)

3. We show that all tree power graphs and two of their extensions are *PCG*s. We present a linear-time algorithm for finding the pairwise compatibility tree of a tree power graph G from its tree root T. (See Chapter 4 for details.)

4. We introduce a new notion called improper *PCG* and show that every graph is an improper *PCG*. We also present an efficient heuristic algorithm to construct an "improper pairwise compatibility tree" for triangulated plane graphs. (See Chapter 5 for details.)

1.7 Thesis Organization

The rest of the thesis is organized as follows. In Chapter 2 we discuss the relevant ideas and necessary definitions from graph theory and algorithm theory to understand our research work. Chapter 3 describes the solution of the open problem regarding whether every graph is a *PCG*, and recognize two subclasses of bipartite graphs as *PCG*s. Chapter 4 deals with the relationships of tree power graphs and its extensions with *PCG*. We propose a new notion called improper *PCG*s in Chapter 5. Finally, We conclude in Chapter 6 with some future directions.

Chapter 2

Preliminaries

In this chapter we define some basic terminology of graph theory and algorithm theory. Definitions that are not included in this chapter will be introduced as they are needed. We start, in Section 2.1, by giving some definitions of standard graph-theoretical terms used throughout the remainder of this thesis. In Section 2.2 we define different terms related to planar graphs. We devote Section 2.3 to define power graphs and some of their variations. The notion of time complexity is discussed in Section 2.4.

2.1 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 [W03, NR04].

2.1.1 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. 2.1 depicts a graph G where each vertex in $V(G) = \{v_1, v_2, \ldots, v_7\}$ is drawn as a small dark circle and each edge in $E(G) = \{e_1, e_2, \ldots, e_{10}\}$

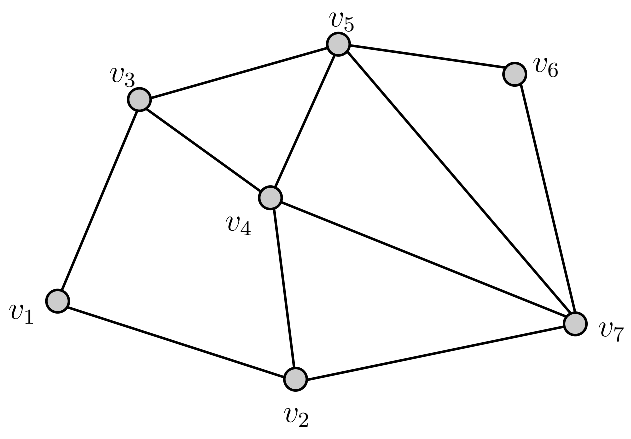

Figure 2.1: A graph with seven vertices and ten edges.

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 be *incident* to vertices u and v. The *degree* of a vertex v in G, denoted by $deg(v)$, 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 to itself. A *complete graph* is a simple graph whose vertices are pairwise adjacent; the complete graph

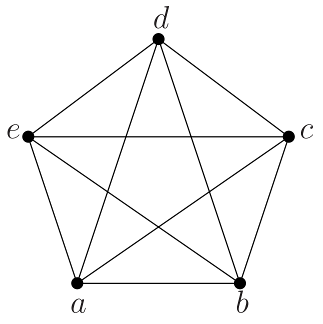

Figure 2.2: A complete graph with five vertices.

with n vertices is denoted by K_n. Figure 2.2 is an example of a complete graph with five vertices. An *independent set* $\{u, v, w, \cdots, z\}$ of G is a set of pairwise non-adjacent vertices. A graph G is *k-partite* if $V(G)$ can be expressed as the union of k independent sets. When $k = 2$ it is called a bipartite graph. A *complete k-partite graph* is a simple graph such that two

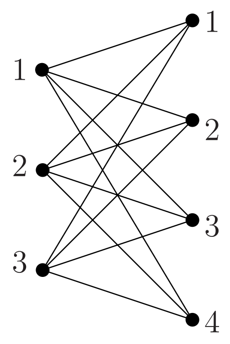

Figure 2.3: A complete bipartite graph.

vertices are adjacent if and only if they are in different partite sets. Figure 2.3 is an example of complete two partite (bipartite) graph.

2.1.2 Subgraphs

A *subgraph* of a graph $G = (V, E)$ is a graph $G' = (V', E')$ such that $V' \subseteq V$ and $E' \subseteq E$; we then write $G' \subseteq G$. If G' contains all the edges of G that join two vertices in V', then G' is said to be the *subgraph induced by* V', and is denoted by $G[V']$. Fig. 2.4 depicts a subgraph of G in Fig. 2.1 induced by $\{v_1, v_2, v_3, v_4, v_6, v_7\}$.

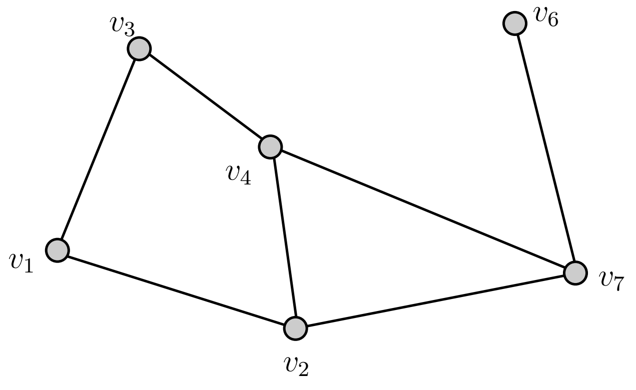

Figure 2.4: A vertex induced subgraph.

We often construct new graphs from old ones by deleting some vertices or edges. If v is a vertex of a given graph $G = (V, E)$, then $G - v$ is the subgraph of G obtained by deleting the vertex v and all the edges incident to v. More generally, if V' is a subset of V, then $G - V'$ is the subgraph of G obtained by deleting the vertices in V' and all the edges incident to them. Then $G-V'$ is a subgraph of G induced by $V-V'$. Similarly, if e is an edge of a G, then $G-e$ is the subgraph of G obtained by deleting the edge e. More generally, if $E' \subseteq E$, then $G-E'$ is the subgraph of G obtained by deleting the edges in E'.

2.1.3 Connectivity

A graph G is a *connected graph* if for every pair $\{u, v\}$ of distinct vertices there is a path between u and v. A graph which is not connected is called a *disconnected graph.* A *connected component* of a graph is a maximal connected subgraph. The graph in Fig. 2.5(a) is a connected graph since there is a path for every pair of distinct vertices of the graph. On the other hand the graph in Fig. 2.5(b) is a disconnected graph since there is no path between v_1 and v_2. The graph in Fig. 2.5(b) has two connected components G_1 and G_2 indicated by dotted lines.

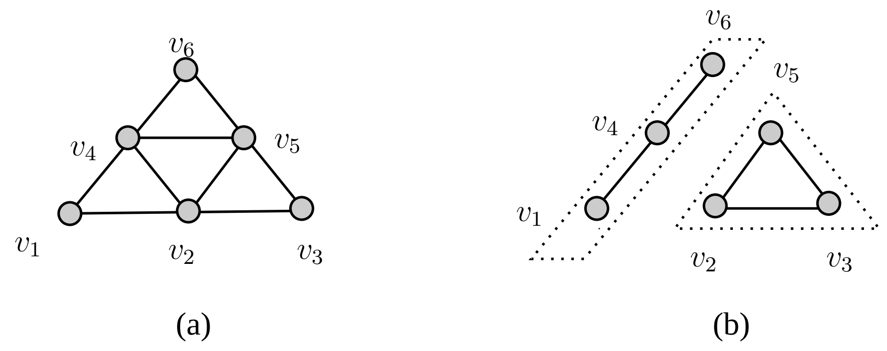

Figure 2.5: (a) A connected graph G and (b) a disconneted graph with two connected components G_1 and G_2.

2.1.4 Paths and Cycles

A v_0, v_l *walk*, $v_0, e_1, v_1, \ldots, v_{l-1}, e_l, v_l$, in G is an alternating sequence of vertices and edges of G, beginning and ending with a vertex, in which each edge is incident to the two vertices immediately preceding and following it. If the vertices $v_0, v_1, \ldots, v_l$ are distinct (except possibly v_0, v_l), then the walk is called a *path* and usually denoted either by the sequence of vertices

$v_0, v_1, \ldots, v_l$ or by the sequence of edges $e_1, e_2, \ldots, e_l$. We denote a path from u to v by P_{uv}, i.e, $P_{uv} = w_0, w_1, \cdots, w_n$ is a sequence of distinct vertices in V such that $u = w_0, v = w_n$ and $(w_{i-1}, w_i) \in E$ for every $1 \le i \le n$. A *sub-path* of P_{uv} is a subsequence $P_{w_j w_k} = w_j, w_{j+1}, ..., w_k$ for some $0 \le j < k \le n$. An *internal vertex* of P_{uv} is any vertex other than u and v that is in P_{uv}. The length of the path P_{uv}, denoted by l_{uv}, is l, one less than the number of vertices on the path. A path or walk is *closed* if $v_0 = v_l$. A closed path containing at least one edge is called a *cycle*.

2.1.5 Trees

A *tree* is a connected graph containing no cycle. Figure 2.6 is an example of a tree. The vertices in a tree are usually called *nodes* . A *rooted tree* is a tree in which one of the nodes is distinguished from the others. The distinguished node is called the *root* of the tree. The root of a tree is generally drawn at the top. In Figure 2.6, the root is v_1. Every node u other than the root is connected by an edge to some other node p called the *parent* of u. We also call u a *child* of p. We draw the parent of a node above that node. For example, in Figure 2.6, v_1 is the parent of v_2, v_3 and v_4, while v_2 is the parent of v_5 and v_6; v_2, v_3 and v_4 are children of v_1, while v_5 and v_6 are children of v_2. A *leaf* is a node of a tree that has no children. An *internal node* is a node that has one or more children. Thus every node of a tree is either a leaf or an internal node. In Figure 2.6, the leaves are v_4, v_5, v_6, v_7 and v_8, and the nodes v_1, v_2 and v_3 are internal nodes.

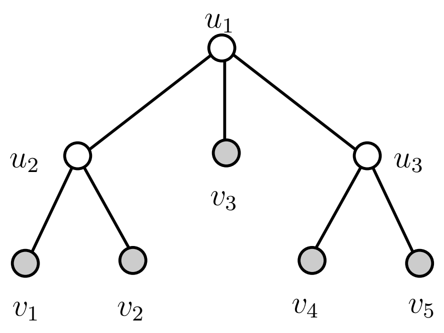

Figure 2.6: Illustration of a tree.

A tree T is *weighted* if each edge is assigned a number as the weight of the edge; we denote the weight of an edge (u, v) by $weight(u, v)$. A *subtree induced by a set of leaves* of T is the minimal subtree of T which contains those leaves. Figure 2.7 illustrates a tree T with six leaves u, v, w, x, y and z, where the edges of the subtree of T induced by u, v and w is drawn by thick lines. We denote by T_{uvw} the subtree of a tree induced by three leaves u, v and w. One can observe that T_{uvw} has exactly one vertex of degree 3. We call the vertex of degree 3 in T_{uvw} the *core* of T_{uvw}. The vertex o is the core of T_{uvw} in Fig. 2.7. The *distance between two vertices u and v* in T, denoted by $d_T(u, v)$, is the sum of the weights of the edges on P_{uv}. A *caterpillar*

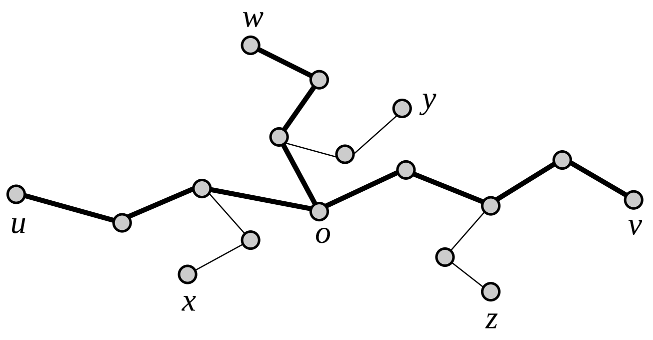

Figure 2.7: Demonstration of a leaf induced subtree.

is a tree in which a single path (*the spine*) is incident to or contains every edge. A *star* is a tree where every leaf has a common neighbour which we call the *base* of the star. In this paper every tree we consider 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 one.

2.2 Planar Graphs and Plane 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].

A graph is a *planar graph* if it can be embedded in the plane so that no two edges intersect geometrically except at a vertex to which they are both incident. Note that a planar graph may have an exponential number of embeddings. Fig. 2.8 shows three planar embeddings of the same planar graph.

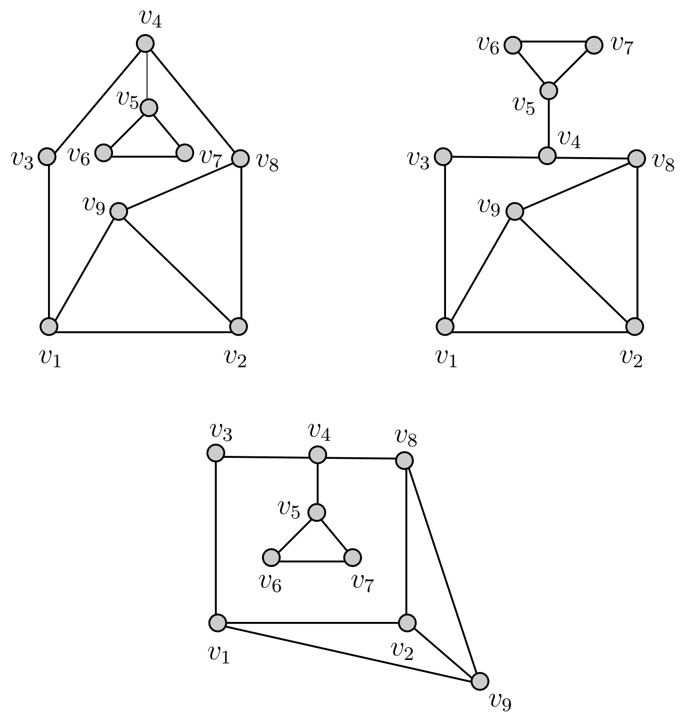

Figure 2.8: Three planar embeddings of the same planar graph.

A *plane graph* is a planar graph with a fixed embedding. A plane graph divides the plane into connected regions called *faces*. We call a plane graph *triangulated plane graph* if all of its faces are triangles.

2.2.1 Dual Graphs

For a plane graph G, we often construct another graph G^* called the *(geometric) dual* of G as follows. A vertex v_i^* is placed in each face F_i of G; these are the vertices of G^*. Corresponding to each edge e of G we draw an edge e^* which crosses e (but no other edge of G and joins the vertices v_i^* which lie in the faces F_i adjoining e; these are the edges of G^*. The construction is illustrated in Fig. 2.9; the vertices v_i are represented by small grey circles and the edges e_i of G are represented by solid lines, whereas the vertices v_i^* are represented by small white circles, and the edges e^* of G^* by dotted 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. We call a dual graph *inner dual* if the outer face is not considered.

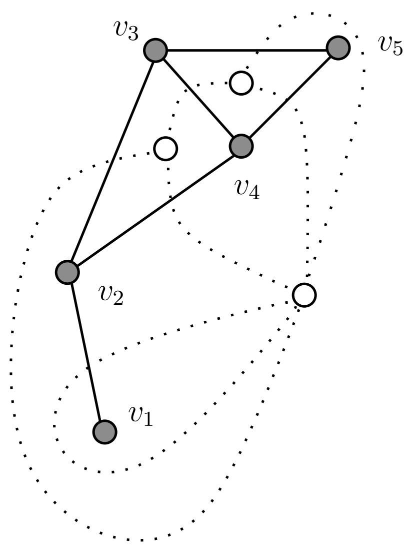

Figure 2.9: A plane graph G and its dual G^*.

2.3 Graph Powers

A graph $G' = (V', E')$ is a *k-root* of a graph $G = (V, E)$ if $V' = V$ and there is an edge $(u, v) \in E$ if and only if the shortest path connecting u and v in G' is at most k. Alternatively, G is called the *k-power* of G' [LKJ00]. The kth power of G is represented as G^k. Figure 2.10(b) illustrates the 2-power graph of the graph G in Fig 2.10(a).

An special case of graph power is tree power which requires G' to be a tree, i.e, a graph $G = (V, E)$ is said to have a tree power for a certain proximity threshold k if a tree T can be constructed on V such that $(u, v) \in E$ if and only if $d_T(u, v) \leq k$. Figure 2.11(b) illustrates the 2-root tree of the graph in Fig. 2.11(a).

"Phylogenetic k-power" and "Steiner k-power" graphs are extensions of tree power graphs. A graph $G = (V, E)$ is called a *phylogenetic k-power graph* if there exists a tree T such that each leaf of T corresponds to a vertex of G and an edge $(u, v) \in E$ if and only if the distance between the two leaves of T corresponding to u and v, which we denote by $d_T(u, v)$, is less than or equal to a chosen proximity threshold k; i.e., $d_T(u, v) \leq k$. Such a tree T is called the

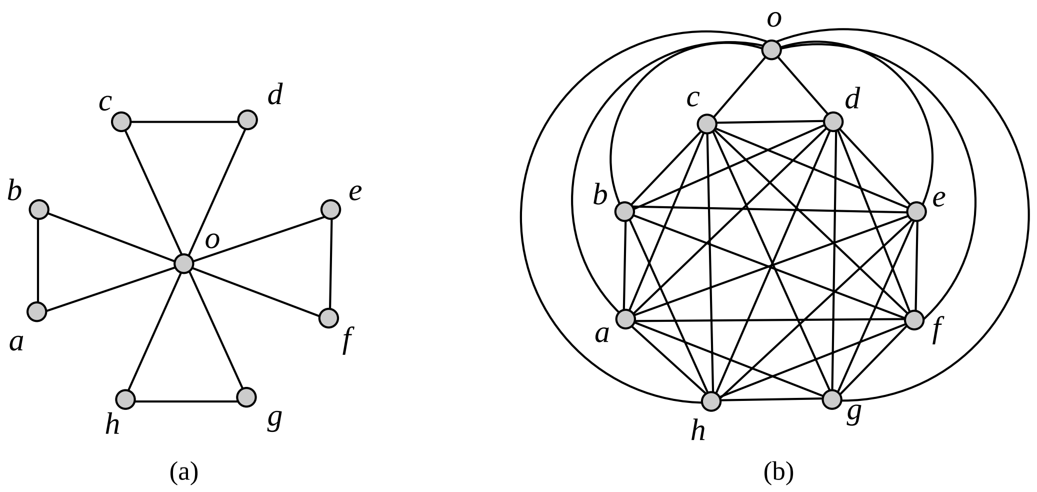

Figure 2.10: (a) A graph G, and (b) the 2-power graph of G.

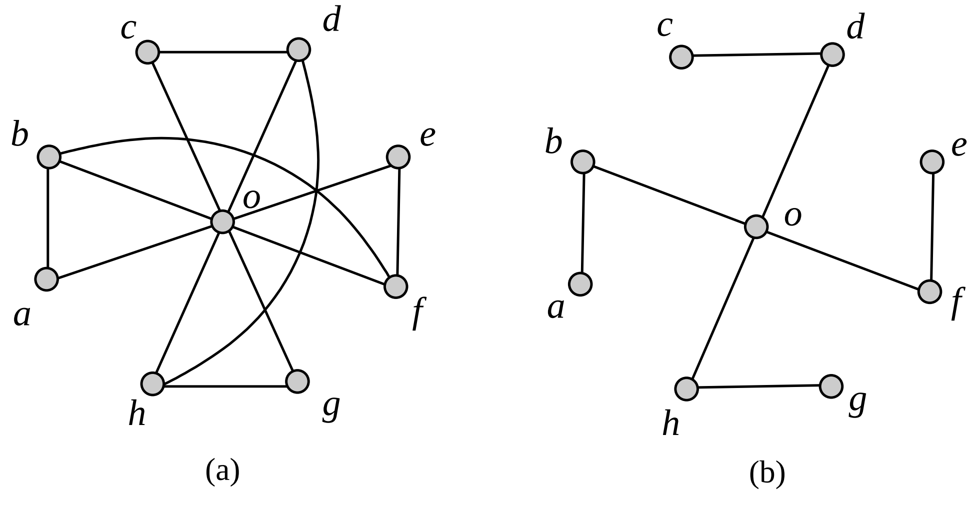

Figure 2.11: (a) A graph G, and (b) the 2-root tree T of G.

k-root phylogeny of G and G is called the *kth phylogenetic power* of T. Phylogenetic k-power graphs are also known as *k-leaf power graphs* [FMRST08, BL09]. Figure 2.12(a) demonstrates the phylogenetic 2-power graph G of the tree T in Figure 2.12(b). We can see that T has four leaves (a, c, e, g), and G has four vertices accordingly.

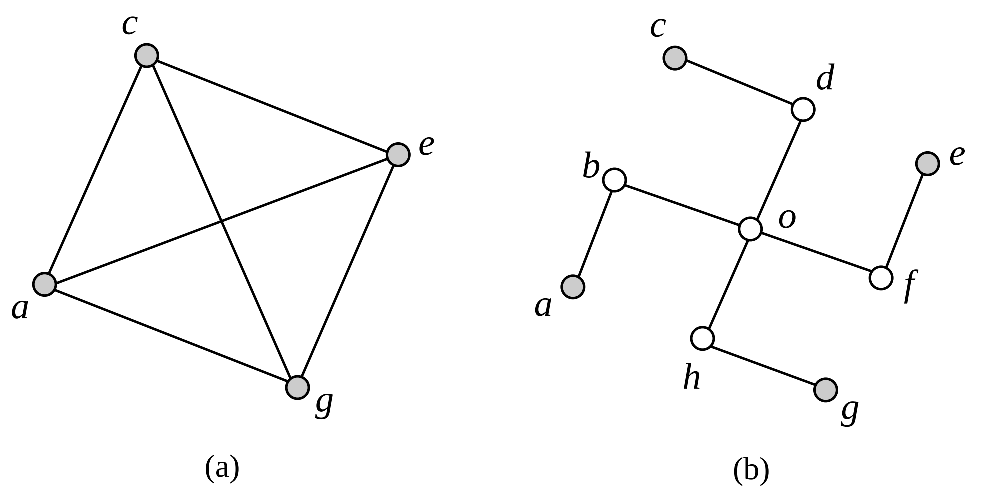

Figure 2.12: (a) A graph G, and (b) the 2-root phylogeny T of G.

In some cases, it is useful to allow the vertices in V to appear as internal nodes in the tree T which leads to a new variant, where vertices of V might appear as internal nodes of T [LKJ00]. For a Steiner k-power graph the corresponding tree can have some internal nodes as well as the leaves that correspond to the vertices of the graph. Such a tree is called a *k-root Steiner tree* of G and G is the *kth Steiner power* of T. The tree T in Fig. 2.13(b) is the Steiner 4-root of the graph shown in Fig. 2.13(a). Here in T the internal vertex o corresponds to the vertex o of G.

2.4 Algorithms and Complexity

In this section we briefly introduce some terminologies related to algorithms and complexity of algorithms. For interested readers, we refer to [GJ79, DPV06, KT05].

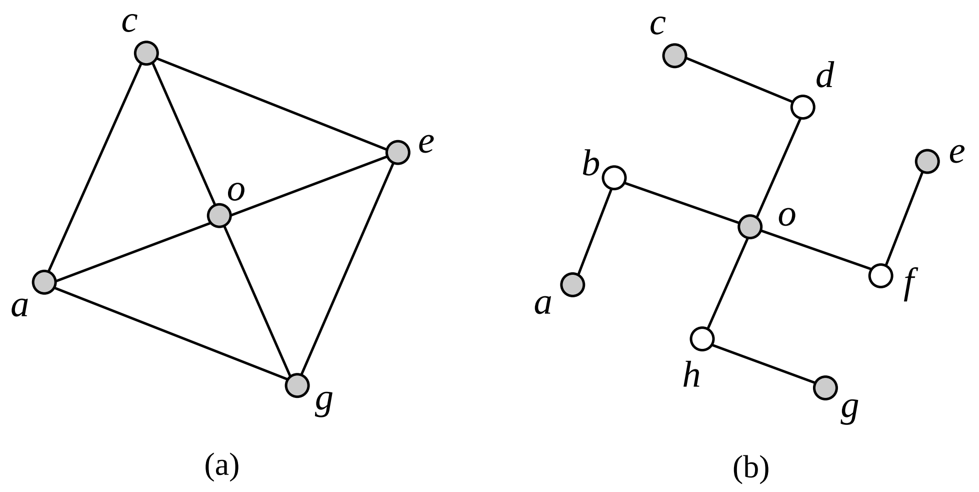

Figure 2.13: (a) A graph G, and (b) the 2-root steiner tree T of G.

2.4.1 Big-O Notation

The most widely accepted complexity measure for an algorithm is the *running time* which is expressed by the number of operations it performs before producing the final answer. Expressing running time in terms of basic computer steps is a simplification. After all, the time taken by one such step depends crucially on the particular processor and even on details such as caching strategy (as a result of which the running time can differ subtly from one execution to the next). Accounting for these architecture-specific minutiae is quite a complex task and yields a result that does not generalize from one computer to the next. It therefore makes more sense to seek an uncluttered, machine-independent characterization of an algorithm's efficiency. To this end, we will always express running time by counting the number of basic computer steps, as a function of the size of the input.

And this simplification leads to another simplification. Instead of reporting that an algorithm takes, say, $5n^3 + 4n + 3$ steps on an input of size n, it is much simpler to leave out lower-order terms such as $4n$ and 3 (which become insignificant as n grows), and even the detail of the coefficient 5 in the leading term (computers will be five times faster in a few years anyway), and just say that the algorithm takes time $O(n^3)$ (pronounced "big oh of n^3"). The reason behind such simplification is that we are often interested only in the "asymptotic behavior", that is, the behavior of the algorithm, when applied to very large inputs, which is insensitive to constant factors and low order terms. We now define this notation precisely. Let $f(n)$ and $g(n)$ are the functions from the positive integers to the positive reals, then we write $f(n) = O(g(n))$ (which means that $f(n)$ grows no faster than $g(n)$) if there exists positive constants c_1 and c_2 such that $f(n) \leq c_1 g(n) + c_2$ for all n.

This cavalier attitude toward constants in case of big-O notation may seem very rude since programmers and algorithm developers are very interested in constants and give tremendous effort in order to make an algorithm run faster even by a factor of 2. But understanding and analyzing algorithms at theoretical level would be impossible without the simplicity afforded by big-O notation.

2.4.2 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 $O(n)$, $O(nlogn)$, $O(n^{100})$, etc. The remaining algorithms are usually referred to as *exponential* or *nonpolynomial*. Examples of such complexity are $O(2^n)$, $O(n!)$, 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.

2.4.3 NP-complete Problems

There are a number of interesting computational problems for which it has not been proved whether there is a polynomial time algorithm or not. Most of them are "NP-complete". In this section we will briefly explain NP-complete problems as well as the complexity class P and NP.

Before going further into details, we first introduce some important concepts. *Decision problems* refer to the algorithmic questions that can be answered by yes or no. For an example, "Is there a truth assignment that satisfies a given boolean formula?" The state of algorithms consists of the current values of all the variables and the location of the current instruction to be executed. A *deterministic algorithm* is one for which each state, upon execution of the instruction, uniquely determines at most one of the following state (next state). All computers, which exist now, run deterministically. In contrast, a *nondeterministic algorithm* is one for which a state may determine many next states simultaneously. We may regard a nondeterministic algorithm as having the capability of branching off into many copies of itself, one for the each next state. Thus, while a deterministic algorithm must explore a set of alternatives one at a time, a nondeterministic algorithm examines all alternatives at the same time.

A problem P_1 is *polynomially reducible* to problem P_2 ($P_1 \leq_p P_2$) if there exists a polynomial time algorithm that transforms every instance I_1 of P_1 to an instance I_2 of P_2 such that the answer to I_1 is "yes" ($I_1 \in P_1$) if and only if the answer to I_2 is "yes" ($I_2 \in P_2$).

The Class NP

NP is the class of problems – solutions of which can be verified deterministically in polynomial time. This means that there is an efficient (low-order polynomial) deterministic checking algorithm C that takes as input the given instance I (the data specifying the problem to be solved), as well as the proposed solution S, and outputs true if and only if S really is a solution to instance I. Moreover the running time of $C(I, S)$ is bounded by a polynomial in $|I|$. We can also define NP as the class of decision problems that can be solved nondeterministically in polynomial time, which is why NP stands for "nondeterministic polynomial time."

The Class P

P is the class of problems that can be solved by deterministic polynomial time algorithm. This implies that there is a deterministic algorithm that takes as input an instance I and has a running time polynomial in I such that if I has a solution, the algorithm returns such a solution; and if I has no solution, the algorithm correctly reports so. Clearly $P \subseteq NP$. But the question, "$P = NP$?" is still unresolved. It is widely believed that $P \neq NP$. However, proving this has turned out to be extremely difficult, one of the deepest and most important unsolved puzzles of mathematics.

The class NP-complete

A problem p is NP-complete if it satisfies the following two conditions.

1. $p \in NP$.

2. For every problem $p' \in NP$, $p' \leq_p p$.

A problem satisfying condition 2 is said to be NP-*hard*, whether or not it satisfies condition 1. NP-complete problems are considered to be the hardest problems in NP. These problems have the following interesting properties.

(a) No NP-complete problem can be solved by any known polynomial algorithm.

(b) If there is a polynomial algorithm for any *NP*-complete problem, then there are polynomial algorithms for all *NP*-complete problems.

2.4.4 Heuristic Algorithms

In computer science, a heuristic algorithm, or simply a heuristic, is an algorithm that is able to produce an acceptable solution to a problem in many practical scenarios, but for which there is no formal proof of its correctness. Alternatively, it may be correct, but may not be proven to produce an optimal solution, or to use reasonable resources. Heuristics are typically used when there is no known method to find an optimal solution, under the given constraints (of time, space etc.) or at all. These algorithms, usually find a solution close to the best one and they find it fast and easily. Sometimes these algorithms can be accurate, that is they actually find the best solution, but the algorithm is still called heuristic until this best solution is proven to be the best. The method used for a heuristic algorithm is one of the known methods, such as greediness, but in order to be easy and fast the algorithm ignores or even suppresses some of the problem's demands. Heuristics rely on ingenuity, intuition, a good understanding of the application and meticulous experimentation to attack a problem [DPV06].

Chapter 3

Pairwise Compatibility Graphs

In this chapter we resolve the open question regarding whether or not every graph is a pairwise compatibility graph by showing that not all graphs are PCGs in Section 3.1. We next recognize some subclasses of bipartite graphs as PCGs in Section 3.2. Finally Section 3.3 is a conclusion.

A graph G is a PCG if there exists an edge weighted tree T and two non-negative real numbers d_{min} and d_{max} such that $G = PCG(T, d_{min}, d_{max})$. Since there are abundance of choices for selecting the topology of T, and real numbers d_{min} and d_{max}, one may think that there would always exist T, d_{min} and d_{max} for a given graph G such that $G = PCG(T, d_{min}, d_{max})$. The proponents of the PCG conjectured that all graphs are PCGs [KMP03]. However, we have observed that regardless of the topology of T and the values of d_{min} and d_{max}, some adjacency relationships among the vertices of a graph makes it impossible to construct an pairwise compatibility tree T for the graph. We have found that the placement of a vertex of a graph as a leaf in an edge weighted tree imposes constraints on the placements of the remaining vertices, and as a result, in some cases, it becomes infeasible to have a tree that generates the graph as a PCG, even when there may exist an exponential number of tree topologies for the given number of graph vertices.

3.1 Not All Graphs are PCGs

In this section we show that not all graphs are pairwise compatibility graphs, as in the following theorem.

Theorem 3.1.1 *Not all graphs are pairwise compatibility graphs.*

To prove the claim of Theorem 3.1.1 we need the following lemmas.

Lemma 3.1.2 *Let T be an edge weighted tree, and u, v and w be three leaves of T such that P_{uv} is the largest path in T_{uvw}. Let x be a leaf of T other than u, v and w. Then, $d_T(w,x) \leq d_T(u,x)$ or $d_T(w,x) \leq d_T(v,x)$.*

Proof. Let o be the core of T_{uvw}. Then each of the paths P_{uv}, P_{uw} and P_{wv} is composed of two of the three subpaths P_{uo}, P_{ow} and P_{ov}. Since $d_T(u,v)$ is the largest path in T_{uvw}, $d_T(u,v) \geq d_T(u,w)$. This implies that $d_T(u,o)+d_T(o,v) \geq d_T(u,o)+d_T(o,w)$. Hence $d_T(o,v) \geq d_T(o,w)$. Similarly, $d_T(u,o) \geq d_T(o,w)$ since $d_T(u,v) \geq d_T(w,v)$. Since T is a tree, there is a path from x to o. Let o_x be the first vertex in $V(T_{uvw}) \cap V(P_{xo})$ along the path P_{xo} from x. Then clearly o_x is on P_{uo}, P_{vo} or P_{wo}. We first assume that o_x is on P_{uo}, as illustrated in Fig. 3.1(a). Then $d_T(v,x) \geq d_T(w,x)$ since $d_T(w,x) = d_T(x,o) + d_T(w,o)$, $d_T(v,x) = d_T(x,o) + d_T(v,o)$ and $d_T(v,o) \geq d_T(w,o)$. We now assume that o_x is on P_{vo}, as illustrated in Fig. 3.1(c). Then $d_T(u,x) \geq d_T(w,x)$ since $d_T(w,x) = d_T(x,o) + d_T(w,o)$, $d_T(u,x) = d_T(x,o) + d_T(o,u)$ and $d_T(u,o) \geq d_T(w,o)$. We finally assume that o_x is on P_{wo}, as illustrated in Fig. 3.1(b). Then $d_T(u,x) = d_T(u,o) + d_T(o,o_x) + d_T(o_x,x)$ and $d_T(w,x) = d_T(w,o_x) + d_T(o_x,x)$. As $d_T(w,o_x) \leq d_T(w,o)$ and $d_T(u,o) \geq d_T(w,o)$, $d_T(u,x) \geq d_T(w,x)$. Likewise, $d_T(v,x) \geq d_T(w,x)$. Thus, in each case, at least one of u and v is at a distance from x that is either larger than or equals to the distance between w and x. $\mathcal{Q.E.D.}$

Lemma 3.1.3 *Let $G = (V,E)$ be a $PCG(T, d_{min}, d_{max})$. Let a, b, c, d and e be five leaves of T and let a', b', c', d' and e' be five vertices of G corresponding to the five leaves a, b, c, d and e of*

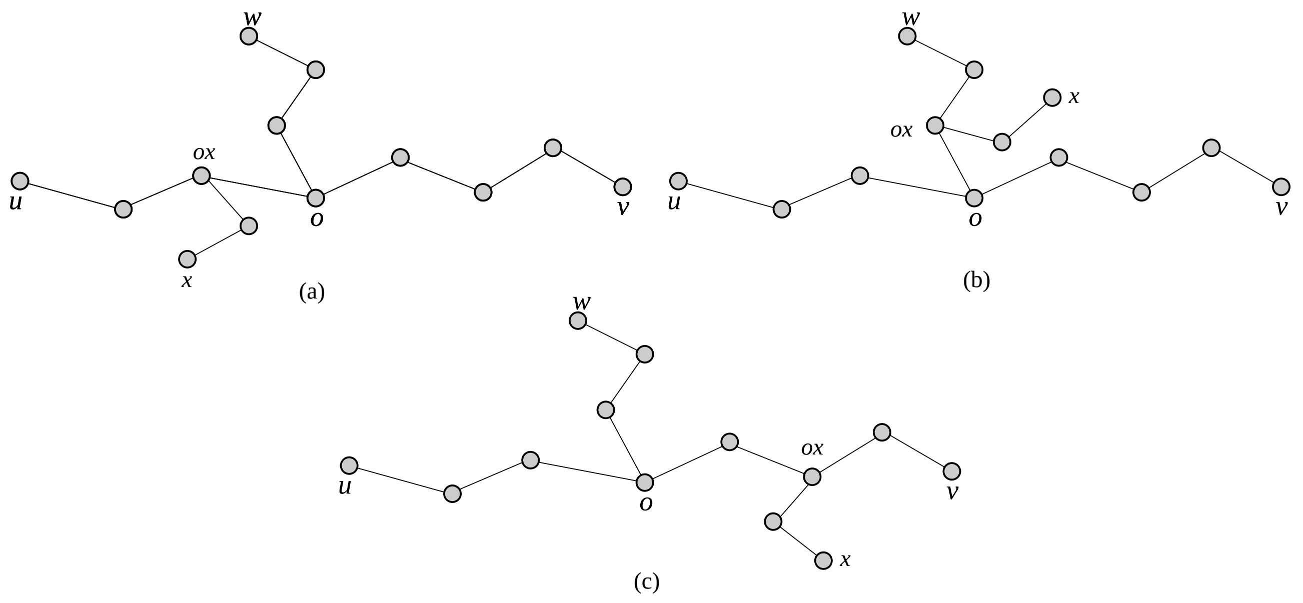

Figure 3.1: Different positions of x.

T, respectively. Let P_{ae} be the largest path in the subtree of T induced by the leaves a, b, c, d and e, and P_{bd} be the largest path in T_{bcd}. Then G has no vertex x' such that x' is adjacent to a', c' and e' but not adjacent to b' and d'.

Proof. Assume for a contradiction that G has a vertex x' such that x' is a neighbor of a', c' and e' but not of b' and d'. Let x be the leaves of T corresponding to the vertex x' of G. Since P_{ae} is the largest path in T among all the paths that connect a pair of leaves from the set $\{a, b, c, d, e\}$, $\max\limits_{y \in \{a,e\}} d_T(x, y) \geq \max\limits_{z \in \{b,c,d\}} d_T(x, z)$ by Lemma 3.1.2. Since both a and e are adjacent to x in G, $\max\limits_{y \in \{a,e\}} d_T(x, y) \leq d_{max}$. This implies that $d_T(x, y) \leq d_{max}$, $y \in \{a, b, c, d, e\}$. Since P_{bd} is the largest path in T_{bcd}, $\max\limits_{y \in \{b,d\}} d_T(x, y) \geq d_T(x, c)$ by Lemma 3.1.2. Without loss of generality assume that $d_T(x, b) \geq d_T(x, c)$. Since b' and x' are not adjacent in G and $d_T(x, b) \leq d_{max}$, $d_T(x, b) < d_{min}$. Then $d_T(x, c) < d_{min}$ since $d_T(x, b) \geq d_T(x, c)$. Since $d_T(x, c) < d_{min}$, c' cannot be adjacent to x' in G, a contradiction. $\mathcal{Q.E.D.}$

Using Lemma 3.1.3 we now present a graph which is not a PCG as in the following Lemma.

Lemma 3.1.4 *Let $G = (V, E)$ be a graph of 15 vertices, and let $\{V_1, V_2\}$ be a partition of the set V such that $|V_1| = 5$ and $|V_2| = 10$. Assume that each vertex in V_2 has exactly three neighbors in V_1 and no two vertices in V_2 has the same three neighbors in V_1. Then G is not a pairwise compatibility graph.*

Proof. Assume for a contradiction that G is a pairwise compatibility graph, i.e., $G = PCG(T, d_{min}, d_{max})$ for some T, d_{min} and d_{max}. Let P_{uv} be the longest path in the subtree of T induced by the leaves of T representing the vertices in V_1. Clearly u and v are leaves of T. Let u' and v' be the vertices in V_1 corresponding to the leaves u and v of T, respectively. Let P_{wx} be the longest path in the subtree of T induced by the leaves of T corresponding to the vertices in $V_1 - \{u', v'\}$. Clearly w and x are also the leaves of T, and let w' and x' be the vertices in V_1 corresponding to w and x of T. Since $|V_1| = 5$, T has a leaf y corresponding to the vertex $y' \in V_1$ such that $y' \notin \{u', v', w', x'\}$. Since G is a PCG of T, G cannot have a vertex adjacent to u', v' and y' but not adjacent to w' and x' by Lemma 3.1.3. However, for every combination of three vertices in V_1, V_2 has a vertex which is adjacent to only those three vertices of the combination. Thus there is indeed a vertex in V_2 which is adjacent to u', v' and y' but not to w' and x'. Hence G can not be a pairwise compatibility graph of T by Lemma 3.1.3, a contradiction. $\mathcal{Q.E.D.}$

Lemma 3.1.4 immediately proves Theorem 3.1.1. Figure 3.2 shows an example of a bipartite graph which is not a PCG. Quite interestingly, however, every complete bipartite graph is a PCG. It can be shown as follows. Let $K_{m,n}$ be a complete bipartite graph with two partite sets $X = \{x_1, x_2, x_3, \cdots, x_m\}$, and $Y = \{y_1, y_2, y_3, \cdots, y_n\}$. We construct a star for each partite set such that each leaf corresponds to a vertex of the respective partite set. Then we connect the bases of the stars through an edge as illustrated in Fig. 3.3. Finally, we assign one as the weight of each edge. Let T be the resulting tree. Now one can easily verify that $K_{m,n} = PCG(T, 3, 3)$.

Taking the graph described in Lemma 3.1.4 as a subgraph of a larger graph, we can show a larger class of graphs which is not PCG, as described in the following lemma.

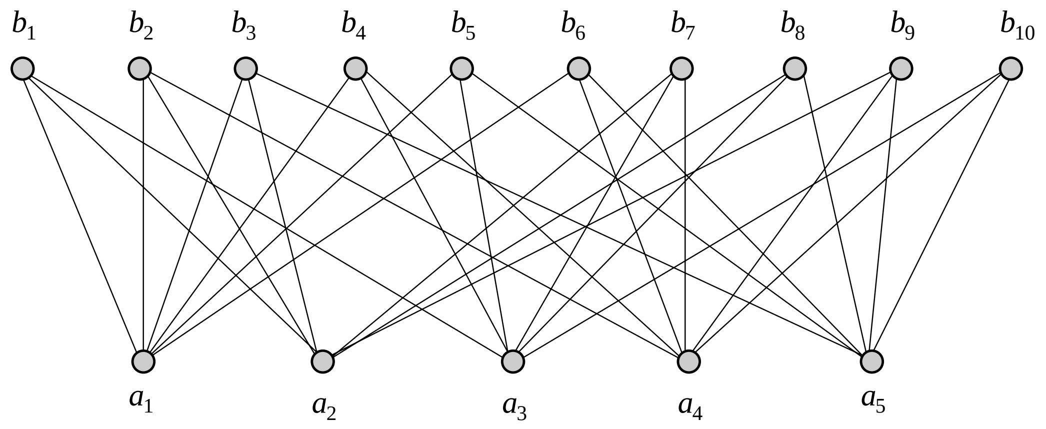

Figure 3.2: Example of a graph which is not a PCG.

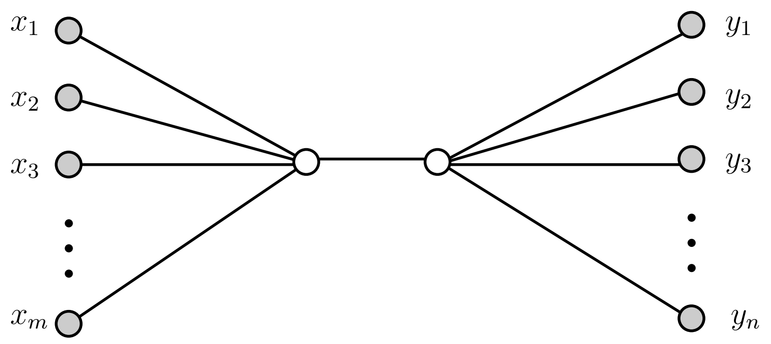

Figure 3.3: Pairwise compatibility tree T of a complete bipartite graph $K_{m,n}$.

Lemma 3.1.5 *Let $G = (V, E)$ be a graph, and let V_1 and V_2 be two disjoint subsets of vertices such that each vertex in V_2 has exactly three neighbors in V_1 and no two vertices in V_2 has the same three neighbors in V_1. Then G is not a pairwise compatibility graph.*

Proof. Assume for a contradiction that G is PCG, i.e., $G = PCG(T, d_{min}, d_{max})$ for some T, d_{min} and d_{max}. Let H be the subgraph of G induced by $V_1 \cup V_2$. Now, let T_H be the subtree of T induced by the leaves representing the vertices in $V_1 \cup V_2$. According to the definition of leaf induced subtree, for any pair of nodes u, v in T_H, $d_{T_H}(u, v) = d_T(u, v)$. Then $H = PCG(T_H, d_{min}, d_{max})$ since $G = PCG(T, d_{min}, d_{max})$. However, H is not a PCG by Lemma 3.1.4, a contradiction. $\mathcal{Q.E.D.}$

3.2 Bipartite Graphs and PCGs

From Theorem 3.1.1, it is evident that not all bipartite graphs are PCGs (see Fig. 3.2). So it would be interesting to find some restricted classes of bipartite graph that are PCGs. We have already showed that complete bipartite graphs are PCGs. We now establish two other subclasses of bipartite graphs as PCGs.

Theorem 3.2.1 *Let $G = (V, E)$ be a bipartite graph with two partite sets P and Q. Let $X \subset P$ and $Y \subset Q$ such that there is no edge in G having one end in X and the other end in Y. Assume that the subgraph of G induced by $(P - X) \cup Q$ is a complete bipartite graph with partite sets $P - X$ and Q, and the subgraph of G induced by $(Q - Y) \cup P$ is a complete bipartite graph with partite sets $Q - Y$ and P. Then G is a PCG.*

Proof. Let G be a bipartite graph as specified in Theorem. 3.2.1 with $|P| = p$, $|Q| = q$, $|X| = r$, and $|Y| = s$. Without loss of generality we can assume that $p \geq q$. We first construct two caterpillars C_p and C_q corresponding to two partite sets P and Q such that each leaf of the C_p and C_q corresponds to a vertex of P and Q, respectively as follows. We make the path $u_1, u_2, \ldots, u_p$ as the spine of C_p and $u'_1, u'_2, \ldots, u'_p$ as the leaves of C_p such that u_i is adjacent to u'_i and $u'_p, u'_{p-1}, \dots, u'_{p-r+1}$ are the r vertices in X. Similarly we make the path $v_1, v_2, \dots, v_q$ as the spine of C_q and $v'_1, v'_2, \dots, v'_q$ as the leaves of C_q such that v_i is adjacent to v'_i and $v'_q, u'_{q-1}, \dots, v'_{q-s+1}$ are the s vertices in Y. C_p and C_q are depicted in Fig. 3.4(a). We next construct a single tree T by connecting C_p and C_q through an edge u_pv_q as illustrated in Fig. 3.4(b). We finally assign the weight of the edges of T as follows. We assign the weight $l_w = 2l$ of the edge u_pv_q, where $l = max\{p, q\}$. We assign weight one to each edge connecting a leaf of the caterpillar to its spine except for the edges incident to the leaves corresponding to the vertices in X and Y, i.e., weight of each edge $u_iu'_i$ and $v_jv'_j$ is one where $1 \leq i \leq p-r$ and $1 \leq j \leq q-s$. We assign $l+1, l, l-1, \dots, l-r+2$ as the weights of the edges $u_pu'_p, u_{p-1}u'_{p-1}, u_{p-2}u'_{p-2}, \dots, u_{p-r+1}u'_{p-r+1}$ respectively. Similarly we assign $l+1, l, l-1, \dots, l-s+2$ as the weights of the edges $v_qv'_q, v_{q-1}v'_{q-1}, v_{q-2}v'_{q-2}, \dots, v_{q-s+1}v'_{q-s+1}$, respectively. This weight assignment is illustrated in Fig. 3.4(c). Let $d_{min} = 2l+2$ and $d_{max} = 4l+1$.

We now show that T is a pairwise compatibility tree of G. The distance between u'_1 and u'_p, and v'_1 and v'_q are $p+l+1$ and $q+l+1$ respectively (see Fig. 3.4(c)). Since $l = max\{p, q\}$, 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, and here we can see that $(2l+1) < d_{min}$. Again the distance between any two leaves 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 leaves corresponding to two vertices that are adjacent in G is $l+2l+(l+1) = 4l+1$ (distance between u'_1 and v'_q), which is within the specified range d_{min} and d_{max}. Again the minimum possible distance between two leaves corresponding to two vertices that are adjacent in G is $1+2l+1 = 2l+2$ (distance between u'_p and v'_q while X and Y are empty), which is also within the specified range. Thus T is a pairwise compatibility tree of G and hence G is a *PCG*. $\mathcal{Q.E.D.}$

Figure 3.5(a) illustrates an example of the bipartite graph as specified in Theorem. 3.2.1 where $p = 5$, $q = 4$, $X = \{p_1, p_2, p_3\}$ and $Y = \{q_3, q_4\}$. Fig. 3.5(b) illustrates the pairwise compatibility tree of the graph in Fig. 3.5(a) obtained by the construction in the proof of

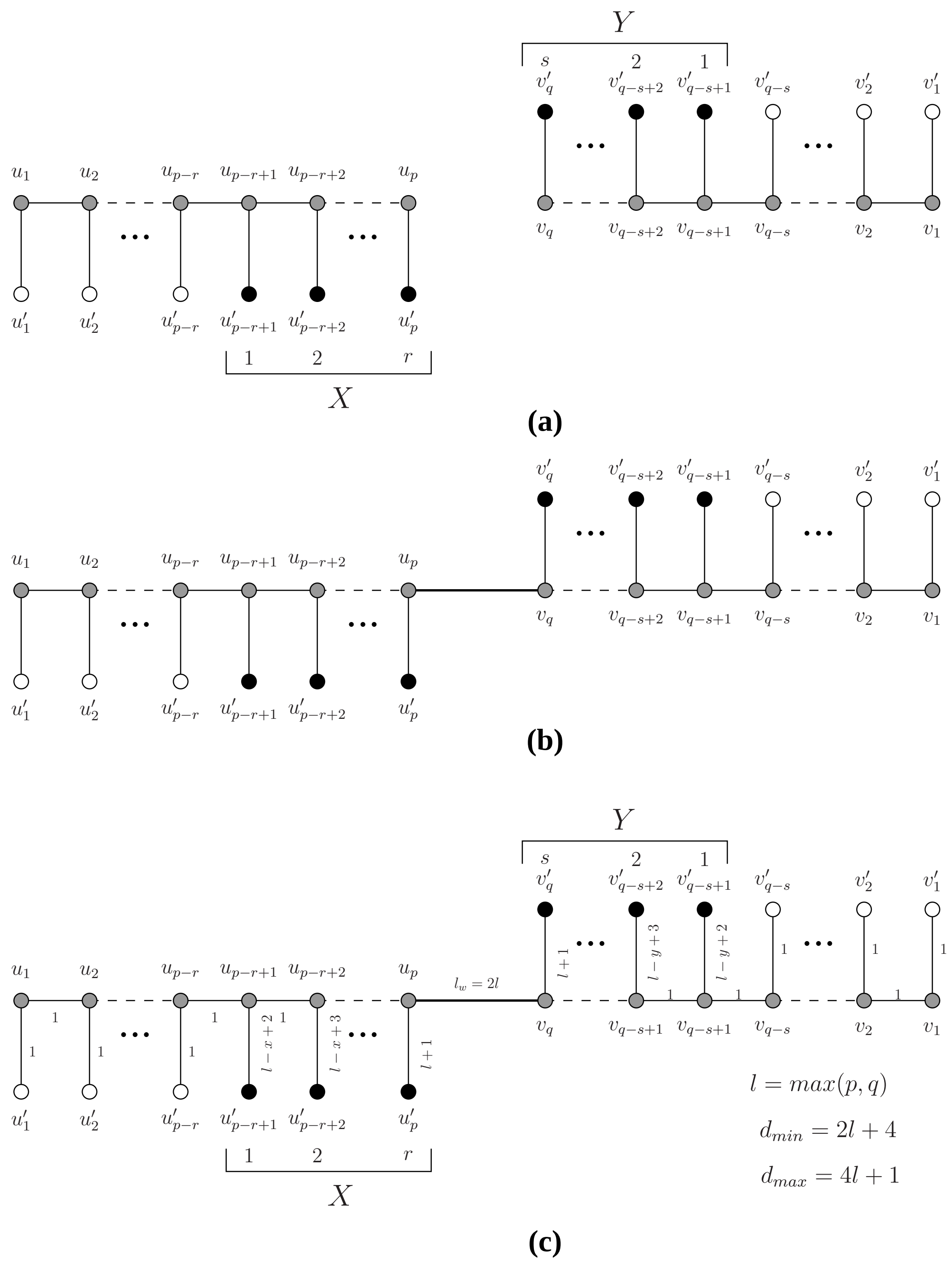

Figure 3.4: (a) Two caterpillars C_p and C_q, (b) connecting C_p and C_q through the edge u_pv_q, and (c) the weight assignment.

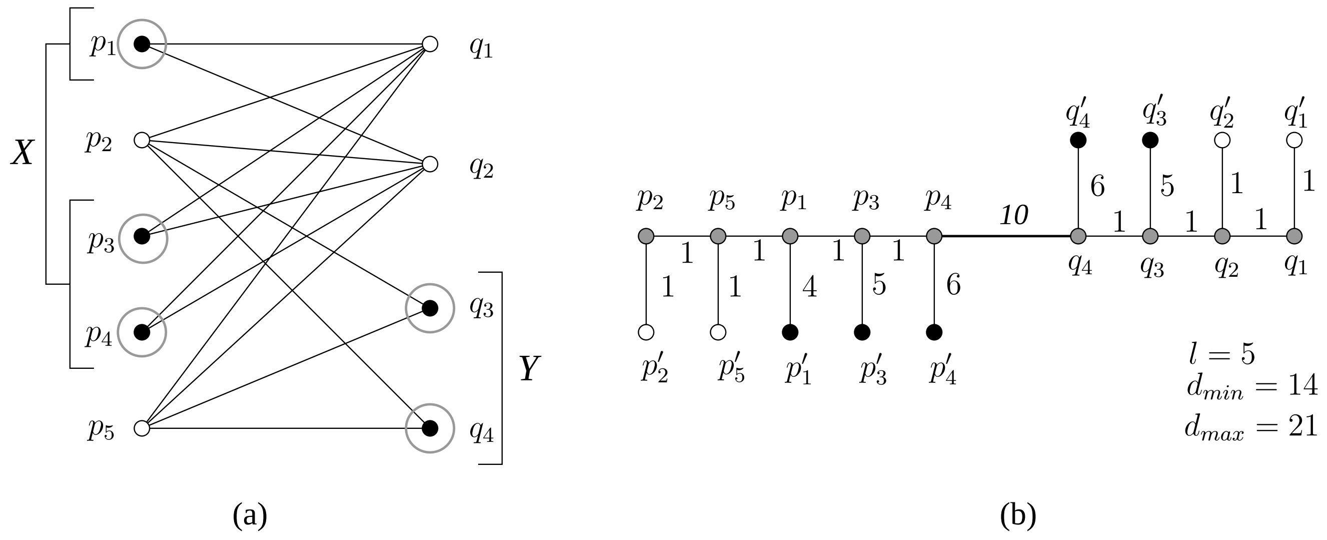

Figure 3.5: (a) A bipartite graph as of Theorem 3.2.1, and (b) its PCG .

Theorem. 3.2.1.

Theorem 3.2.2 *Let $G = (V, E)$ be a bipartite graph with two partite sets P and Q where $|P| = p$, $|Q| = q$ and $p \geq q$. Let $p_1, p_2, \ldots, p_p$ be the p vertices in P and let $q_1, q_2, \ldots, q_q$ be the q vertices in Q. Let $d_m = \max_{v \in P} deg(v)$. Then G is a PCG if any of the following conditions (1) and (2) holds.*

1. *For any vertex $u \in P$, if $deg(u) = d_m$ then its neighbors are $q_i, q_{i+1}, \ldots, q_{i+deg(u)-1}$ where $1 \leq i \leq (q - deg(u) + 1)$.*

2. *If $deg(u) < d_m$, then its neighbors are $q_1, q_2, \ldots, q_{deg(u)}$ or $q_q, q_{q-1}, \ldots, q_{q-(deg(u)-1)}$.*

Proof. Let $G = (V, E)$ be a bipartite graph as specified in Theorem. 3.2.2. We now construct two caterpillars C_p and C_q corresponding to two partite sets P and Q such that each leaf of the C_p and C_q corresponds to a vertex of P and Q, respectively as follows. We make the path $u_1, u_2, \ldots, u_p$ as the spine of C_p and $u'_1, u'_2, \ldots, u'_p$ as the leaves of C_p such that u_i is adjacent to u'_i. Similarly we make the path $v_1, v_2, \ldots, v_q$ as the spine of C_q and $v'_1, v'_2, \ldots, v'_q$ as the leaves of C_q such that v_i is adjacent to $v_i^{'}$. C_p and C_q are depicted in Fig. 3.6(a). Here u_i' and v_i' correspond to p_i and q_i, respectively. We now construct a single tree T by connecting C_p and

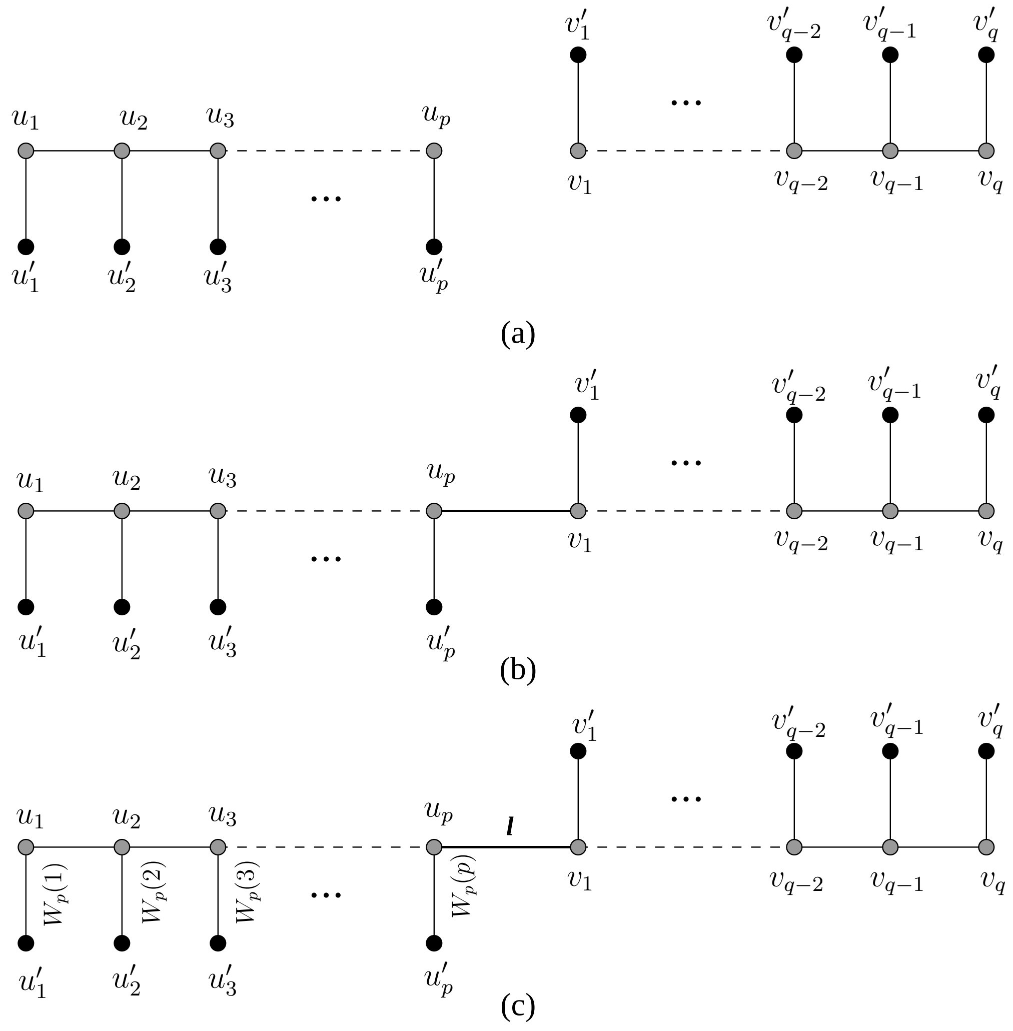

Figure 3.6: (a) Two caterpillars (C_p and C_q), (b) connecting C_p and C_q through the edge u_pv_1, and (c) the weight assignment.

C_q through an edge u_pv_1 as in Fig. 3.6(b). We assign weight one to each edge of C_q. Let l, $W_p(i)$, be the weight of the edge u_pv_1, and the weight of the edge u_iu_i', respectively. Let the neighbors of p_i in G be $q_j, q_{j+1}, \ldots, q_{j+deg(p_i)-1}$; then we define $N_{skip}(i)$ as $j-1$.

Let $d'_{max} = p+q$, $d'_{min} = d'_{max} - (d_m - 1)$ and $l = 2(p+q-1) - d'_{min} + 1$. We now take d_{min} as $d'_{min} + l$ and d_{max} as $d'_{max} + l$. Then we take $W_p(i)$ as follows and show that T is a pairwise compatibility tree of G.

$$W_p(i) = \begin{cases} d'_{max} - (p-i) - (N_{skip}(i) + deg(p_i)) & \text{if } deg_m(i) = d_m(i) \text{ or } deg_m(i) < d_m(i) \text{ and its neighbors are the first } deg_m(i) \text{ vertices of } Q\text{-partite set} \\ d'_{min} - (q - deg(p_i) + 1) - (p-i) & \text{otherwise.} \end{cases} \quad (3.1)$$

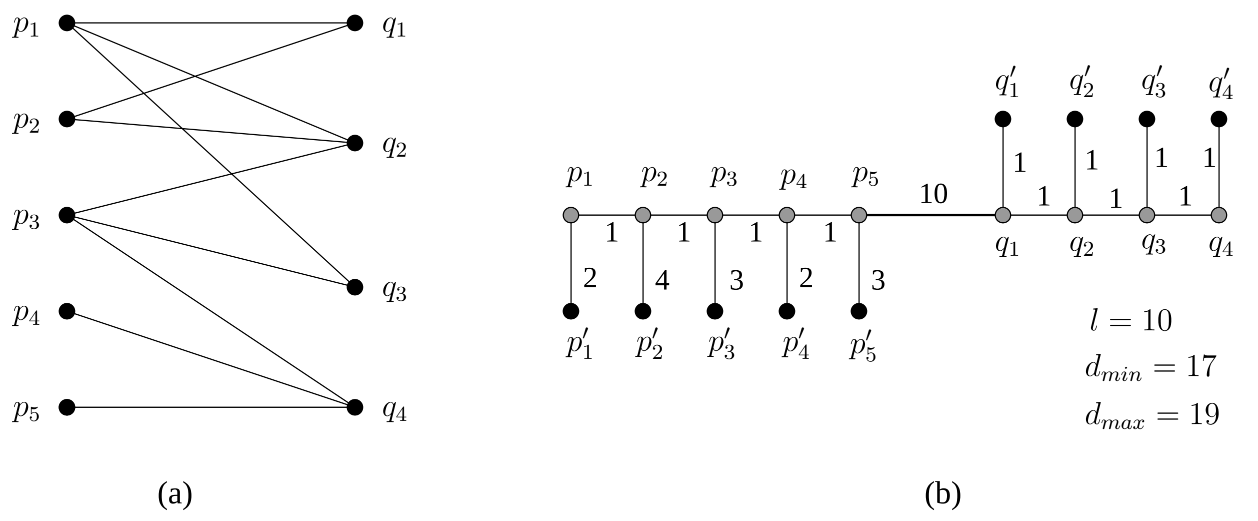

Figure 3.7: (a) An example of a graph as of Theorem 3.2.2, and (b) its PCG.

We have defined $W_p(i)$ in Eqn. 3.1 in such a way that if $deg(p_i) = d_m$ and its neighbors are $q_j, q_{j+1}, \ldots, q_{j+deg(p_i)-1}$ then the distances from u'_i to v'_j, and $v'_{j+deg(p_i)-1}$ are equal to d_{min} and d_{max}, respectively. This implies that the distance from u'_i to v'_t, where $p_iq_t \notin E$ is either less than d_{min} or greater than d_{max}. Next, if $deg(p_i) < d_m$ and its neighbors are $q_1, q_2, \ldots, q_{deg(p_i)}$ then the distance from u'_i to $v'_{deg(p_i)}$ is d_{max}. In this case, the distance from u'_i to v'_t, where $p_iq_t \notin E$ is greater than d_{max}. Finally, if $deg(p_i) < d_m$ and its neighbors are $q_q, q_{q-1}, \ldots, q_{q-(deg(p_i)-1)}$, then the distance from u_i to $v'_{q-(deg(p_i)-1)}$ is d_{min}. Thus the distance from u'_i to v'_t where $p_iq_t \notin E$ is less than d_{min}. Therefore, the distance between two leaves is within the range defined by d_{min} and d_{max} if and only if their corresponding vertices in G are adjacent. Again, d_{min} must be greater than the maximum possible distance between the two leaves of C_p; and the weight of l should be chosen accordingly so that the distance between the two leaves corresponding to two adjacent vertices of G is greater than the maximum possible distance between the two leaves of C_p. The

maximum possible distance between the two leaves of C_p is the distance between leaves u_1 and u_p when $W_p(1)$ and $W_p(p)$ get their maximum possible weight. Again, they get their maximum weight when p_1 and p_p are connected only to the first vertex (q_1) in the Q-partite set. In this situation $W_p(1) = p+q-(1-1)-(p-1+1) = q$ and $W_p(p) = p+q-(1-1)-(p-p+1) = p+q-1$ using the formula as stated in Eqn. 3.1. Now maximum possible distance between two vertices of the same caterpillar is equal to maximum distance between u_1 and u_p
$=$ (maximum weight of leaf u_1) $+$ (maximum weight of leaf u_p) $+(p-1)$
$= q + (p+q-1) + (p-1) = 2(p+q-1)$. Then d_{min} must be greater than $2(p+q-1)$ and we assign $d_{min} = 2(p+q-1)+1$. Now $l = d_{min} - d'_{min} = 2(p+q-1) - d'_{min} + 1$. Therefore, T is a pairwise compatibility tree of G. $\mathcal{Q}.\mathcal{E}.\mathcal{D}.$

Figure 3.7(a) illustrates an example of the bipartite graph as specified in Theorem. 3.2.2. In this example, $p = 5$, $q = 4$ and $d_m = 3$. Note that $deg(p_1) = deg(p_3) = d_m = 3$ and other vertices of P-partite set are of less degree. Two vertices p_1 and p_3 are connected to 3 vertices of Q-partite set having neighbor set $\{q_1, q_2, q_3\}$ and $\{q_2, q_3, q_4\}$ respectively. On the other hand, $deg(p_2) = 2$ which is less than d_m and its neighbors are q_1 and q_2. Similarly vertices p_4 and p_5 have degree one and their neighbor is q_4. Here $N_{skip}(p_1) = 0$, $N_{skip}(p_3) = 1$ and $N_{skip}(p_4) = 3$.

Figure 3.7(b) illustrates the pairwise compatibility tree T of the graph given in Fig. 3.7(a) obtained by the construction in the proof of Theorem. 3.2.2. One can easily verify that T is the pairwise compatibility tree of G.

3.3 Summary

In this chapter we have settled the conjecture that every graph is a pairwise compatibility graph in negative. At the same time, we revealed the fact that not all bipartite graphs are PCGs. We have also recognized two subclasses of bipartite graphs as PCGs.

Chapter 4

Tree Power Graphs and *PCG*s

In this chapter we recognize that tree power graphs and two of its extensions namely phylogenetic k-power graphs and Steiner k-power graphs are pairwise compatibility graphs. We first discuss the importance of establishing such relationships. Next, we continue in Section 4.1 with giving a brief description on tree power graphs and their extensions, and introducing the "tree compatible graphs." In Section 4.2 we establish that every tree compatible graphs are pairwise compatible graphs. Finally Section 4.3 is a conclusion.

For any class of graphs, recognition is a fundamental structural and algorithmic problem. Hence, recognizing tree power graphs and some of its extensions as *PCG*s is very important. But the most intriguing potential of this recognition lies in the fact that there is rich literature on tree roots and powers (more generally graph powers/roots), but few results on *PCG*s. Substantial works have been done on the power of special classes of graphs such as chordal graphs [BP83, LS83], interval graphs [R87], strongly chordal graphs [R92] and circular arc graphs [R92]. Moreover, it is well known that the square of a 2-connected graph has a Hamiltonian cycle [F74], and the Hamiltonian cycle can be found in polynomial time [L80]. Furthermore, Skiena showed that for any non-trivial connected graph G, the graph G^3 is Hamiltonian [S60]. Knowing that tree power graphs and their extensions are *PCG*s, we can utilize the algorithmic properties of these classes in *PCG*.

We have already mentioned that the complete characterization and the computational complexity of recognizing PCGs are not known to date. But a lot of work has been done on power graphs and their extensions. kth power recognition of a tree can be solved in polynomial time for $k > 2$; for bipartite graphs it is NP-complete [L06]. Lau *et al.* showed that for proper interval graphs it is in P [L06]. They also proved the NP-completeness of recognizing squares of chordal graphs, recognizing squares of split graphs, and recognizing chordal graphs that are squares of some graph. For $k \leq 5$, k-leaf powers can be recognized in linear-time, and for $k \leq 4$, structural characterizations are known [BL09].

Tree power graphs, phylogenetic k-power graphs and Steiner k-power graphs have applications in the reconstruction of the evolutionary history of species and admit efficient algorithms for generally difficult problems, that mainly exploit the tree structure of the underlying tree [FMRST08]. Since pairwise compatibility graphs have also been descended from phylogeny, it is quite important to investigate the existence of any relationship of these aforementioned three graphs classes with pairwise compatibility graphs. Such a relationship may serve the purpose of not only unifying related graph classes, but also utilizing the method of tree constructions for one graph class in another.

4.1 Tree Compatible Graphs

To establish the relationship of tree power graphs, phylogenetic k-power graphs, and Steiner k-power graphs with pairwise compatibility graphs, we introduce a generalized graph class which we call "tree compatible graphs." Before defining tree compatible graphs, we first present a brief overview on phylogenetic k-root tree, Steiner k-root tree, and k-root tree of a graph G. Interested readers are referred to [LKJ00] to know more about these graphs.

4.1.1 Phylogenetic k-root Tree

Interspecies similarity is represented by a graph where the vertices are the species and the adjacencies represent evidence of evolutionary similarity. A phylogeny is then constructed from the graph such that the leaves of the phylogeny are labeled by vertices of the graph and two vertices are adjacent in the graph if and only if the corresponding leaves in the tree are connected by a path of length at most k, where k is a chosen proximity threshold. This gives rise to the following algorithmic problem:

Problem	*Phylogenetic k-root Problem*
Input	A graph $G = (V, E)$.
Output	A tree T such that its leaves are labeled by V and for each pair of vertices $u, v \in V, (u, v) \in E$ if and only if $d_T(u, v) \leq k$.

The tree T which is asked to construct in the phylogenetic k-root problem is called a *k-root phylogeny of* G and G is called the *kth phylogenetic power* of T. Lin *et al.* proposed an $O(|V|+|E|)$ time algorithm to determine if a (not necessarily connected) graph $G = (V, E)$ has a 3-root phylogeny, and if so, demonstrate one such phylogeny [LKJ00]. They also proposed an $O(|V| + |E|)$ time algorithm to determine if a connected graph $G = (V, E)$ has a 4-root phylogeny, and if so, demonstrate one such phylogeny.

4.1.2 Steiner k-root Tree

In the context of phylogenetic k-root tree T of G, only the leaves correspond to the vertices of G. However, it is sometimes useful to allow the vertices in V to appear as internal nodes in the output tree T. This gives rise to the following problem:

Problem	*Steiner k-root Problem*
Input	A graph $G = (V, E)$.
Output	A tree T such that its leaves and a subset of internal vertices are labeled by V and for each pair of vertices $u, v \in V, (u, v) \in E$ if and only if $d_T(u, v) \leq k$.

The tree T which is asked to construct in the Steiner k-root problem is called a *k-root Steiner tree* of G and G is the *kth Steiner power* of T. The internal nodes of T that do not correspond to any vertex of G are termed *Steiner points*. Note that a phylogenetic tree of V is a Steiner tree of V where all internal nodes are Steiner points.

4.1.3 k-root Tree

Phylogenetic power and Steiner power are extensions of the standard notion of graph power. An important special case of graph power is tree power. The *tree k-root problem* is as follows.

Problem	*Tree k-root Problem*
Input	A graph $G = (V, E)$.
Output	A tree T on V such that $(u, v) \in E$ if and only if $d_T(u, v) \leq k$.

The tree T which is asked to construct in the tree k-root problem is called a *k-root tree* of G. Whereas recognizing a graph power is NP-complete [MS94], it is possible to determine if a graph has a k-root tree, for any fixed k, in $O(n^3)$ time, where n is the number of vertices in the input graph [KC98].

4.1.4 Tree Compatible Graphs

A graph $G = (V, E)$ is a *tree compatible graph* if there exists a tree T such that all leaves and a subset of internal nodes of T correspond to the vertex set V of G, and for any two vertices $u, v \in V$; $(u, v) \in E$ if and only if $k_{min} \leq d_T(u, v) \leq k_{max}$. Here k_{min} and k_{max} are real numbers. We call G the *tree compatible graph* of T for k_{min} and k_{max}. It is quite evident from this definition that tree compatible graphs comprises tree power graphs, Steiner k-power graphs, and phylogenetic k-power graphs.

4.2 Tree Compatible Graphs are PCGs

As their definition suggests, tree power graphs and their extensions have striking resemblance, in their underlying concept, with PCGs. But does this similarity signify any real relationship? It does indeed; we find that tree power graphs and these two extensions are essentially PCGs, as in the following theorem.

Theorem 4.2.1 *Every tree compatible graph is a pairwise compatibility graph.*

Proof. Let G be a tree compatible graph of a tree T for non-negative real numbers k_{min} and k_{max}. Then to prove the claim, it is sufficient to construct a tree T' and find two non-negative real numbers d_{min} and d_{max} such that $G = PCG(T', d_{min}, d_{max})$.

Clearly $G = PCG(T', d_{min}, d_{max})$ for $T' = T$, $d_{min} = k_{min}$ and $d_{max} = k_{max}$ if every vertex in V corresponds to a leaf in T. We thus assume that V contains a vertex which corresponds to an internal node of T. In this case we construct a tree T' from T as follows. For every internal node u of T that corresponds to a vertex in V, we introduce a surrogate internal node u'. In addition, we transform u into a leaf node by connecting u through an edge of weight λ with u'. Figure 4.1 illustrates this transformation. Here, in addition to the leaves of T, two internal nodes d and e correspond to the vertices in V. T' is the modified tree after transforming d and e into leaf nodes by replacing them by d' and e', respectively.

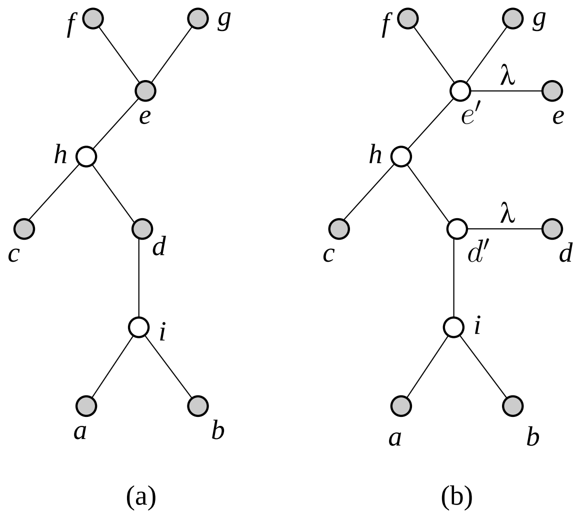

Figure 4.1: (a) T and (b) T'.

The aforementioned transformation transmutes the subset of internal nodes of T that participates in V into a subset of leaves in T'. Let u and v be two arbitrary nodes in T. If u and v are both leaves in T then $d_{T'}(u, v) = d_T(u, v)$. If both u and v are internal nodes of T that are contributing to V then in T' they are two leaf nodes, and $d_{T'}(u,v) = d_T(u,v) + 2\lambda$. Finally, if only one of u and v is transformed to leaf then $d_{T'}(u,v) = d_T(u,v) + \lambda$. We next define $d_{min} = k_{min}$ and $d_{max} = k_{max} + 2\lambda$. Since every vertex $u \in V$ is represented as a leaf in T', T' may be a pairwise compatibility tree of G. We will prove that T' is indeed a pairwise compatibility tree by showing that $G = PCG(T', d_{min}, d_{max})$ for an appropriate value of λ. Note that we cannot simply assign $\lambda = 0$ because, in the context of root finding as well as phylogenetics, an edge of zero weight is not meaningful. For example, if an evolutionary tree contains zero weighted edges then we may find a path of length zero between two different organisms, which is clearly unacceptable. Therefore, we have to choose a value for λ more intelligently.

According to the definition of tree compatible graphs, for every pair of vertices $u, v \in V$, $(u,v) \in E$ if and only if $k_{min} \le d_T(u,v) \le k_{max}$. Meanwhile, we have derived T' from T in such a way that either the distance between u and v in T' remains the same as in T, or increased by at most 2λ. Therefore, if we can prove that $d_{min} \le d_{T'}(u,v) \le d_{max}$ if and only if $k_{min} \le d_T(u,v) \le k_{max}$ then it will imply that $G = PCG(T', d_{min}, d_{max})$. Depending on the nature of the change in the distance between u and v from T to T', we have to consider three different cases.

Case 1: $d_{T'}(u,v) = d_T(u,v)$.

In this case, three possible relationships can exist among $d_T(u,v)$, k_{min} and k_{max}. First, if $d_T(u,v) < k_{min}$ then $d_{T'}(u,v) < d_{min}$ since $d_{min} = k_{min}$. Next, if $k_{min} \le d_T(u,v) \le k_{max}$ then $k_{min} \le d_T(u,v) \le k_{max}{+}2\lambda$. That implies, $d_{min} \le d_T(u,v) \le d_{max}$. Finally, let $d_T(u,v) > k_{max}$. Suppose p is the minimum difference between k_{max} and the length of a path in T that is longer than k_{max}, that is $p = \min\limits_{u,v \in V}\{d_T(u,v) - k_{max}\}$. Then $d_T(u,v) - k_{max} \ge p$. By subtracting 2λ from both side of the inequality we get, $d_T(u,v) - k_{max} - 2\lambda \ge p - 2\lambda$. Which implies $d_{T'}(u,v) - d_{max} \ge p - 2\lambda$. Therefore, if we can ensure that $p > 2\lambda$ then $d_{T'}(u,v)$ will be larger than d_{max}.

Case 2: $d_{T'}(u,v) = d_T(u,v) + 2\lambda$.

In this case, we have to consider three scenarios as we have in case 1. First, if $k_{min} \le d_T(u,v) \le k_{max}$ then $k_{min} \le d_T(u,v) + 2\lambda \le k_{max} + 2\lambda$. Which implies $d_{min} \le d_T(u,v) + 2\lambda \le d_{max}$.

Hence $d_{min} \leq d_{T'}(u,v) \leq d_{max}$. Next, if $d_T(u,v) > k_{max}$ then adding 2λ in both sides we get $d_T(u,v) + 2\lambda > k_{max} + 2\lambda$. That implies $d_{T'}(u,v) > d_{max}$. Finally, let assume that $d_T(u,v) < k_{min}$. Suppose q is the minimum difference between k_{min} and the length of a path in T that is smaller than k_{min}; that is $q = \min_{u,v \in V}\{k_{min} - d_T(u,v)\}$. Then $k_{min} - d_T(u,v) \geq q$. Subtracting 2λ from both sides of the inequality we get $k_{min} - d_T(u,v) - 2\lambda \geq q - 2\lambda$. Which implies $d_{min} - d_{T'}(u,v) \geq q - 2\lambda$. Therefore, if we can ensure that $q > 2\lambda$ then $d_{T'}(u,v) < d_{min}$.

Case 3: $d_{T'}(u,v) = d_T(u,v) + \lambda$.

This case is similar to case 2. By following the same reasoning as in case 2, we can show that $d_{min} \leq d_{T'}(u,v) \leq d_{max}$ if and only if $k_{min} \leq d_T(u,v) \leq k_{max}$, provided $q \geq \lambda$. If we can satisfy the inequality derived from case 2 ($q > 2\lambda$) then the inequality $q > \lambda$ will be immediately satisfied.

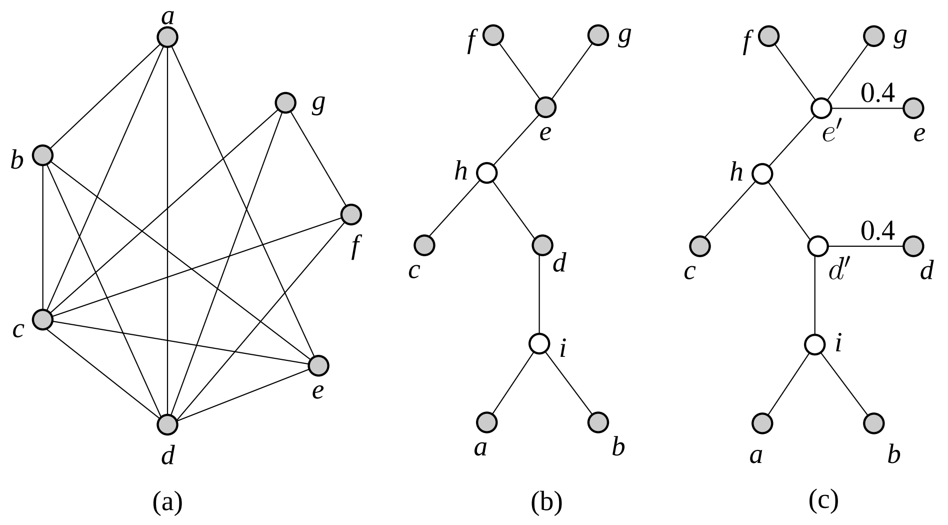

Figure 4.2: (a) A tree compatible graph, (b) the corresponding tree T, and (c) the corresponding pairwise compatibility tree T'.

From our analysis of the three cases above, it is evident that if we can satisfy the two inequalities $p > 2\lambda$ and $q > 2\lambda$ simultaneously then $G = PCG(T', d_{min}, d_{max})$. We can do this by assigning λ any value smaller than $min(p,q)/2$. Thus T' is a pairwise compatibility tree of G, and hence G is a PCG. $\mathcal{Q.E.D.}$

Figure 4.2(a) illustrates an example of a tree compatible graph $G = (V, E)$ and the corresponding tree T is depicted in Fig. 4.2(b). Here $k_{min} = 2$, $k_{max} = 4$, and the weight of every edge is one . Two internal nodes d and e along with the leaves of T correspond to the vertices in V of G. We now transfer T into T' according to the procedure described in Theorem 4.2.1. Figure 4.2(c) illustrates this transformation. Here, $p = q = 1$ and hence we can chose any positive value less than 0.5 for λ. Let $\lambda = 0.4$ and then, $d_{min} = k_{min} = 2$ and $d_{max} = k_{max} + 2\lambda = 4.8$. One can now easily verify that $G = PCG(T', 2, 4.8)$.

4.3 Summary

In this chapter we have established a relationship of pairwise compatibility graphs with the tree power graphs and two of their extensions. We have showed that tree power graphs, phylogenetic k-power graphs and Steiner k-power graphs are pairwise compatibility graphs.

Chapter 5

Improper *PCG*s

In this chapter we introduce a new notion which we call "improper *PCG*s." We start with defining improper *PCG* and other related terms in Section 5.1. In Section 5.2, we show that every graph is an improper *PCG*. We present some preliminary results on triangulated planar graphs in Section 5.3 and present a heuristic algorithm to construct an "improper pairwise compatibility tree" of any triangulated plane graph in Section 5.4. Finally we conclude in Section 5.5.

5.1 Improper *PCG*s

In this section we introduce two new concepts that we call *redundancy* and *improper PCG*. A graph G is an *improper PCG* if the corresponding pairwise compatibility tree can be constructed by allowing multiple existences of the same leaf; we call these extra leaves *redundancies*. Fig. 5.1 shows an example of redundancy and improper *PCG*. Here the pairwise compatibility tree T in Fig. 5.1(b) contains two leaves corresponding to the vertex v_6 of G in Fig. 5.1(a), one of which is called a redundancy (enclosed by grey circle). We call the pairwise compatibility tree T, allowing some redundancies, an *improper pairwise compatibility tree*. We denote the number of redundancies by n_{rd} . In Fig. 5.1(b) $n_{rd} = 1$. For an improper pairwise compatibility tree $n_{rd} \geq 0$. Obviously, a *PCG* is also an improper *PCG*, but the reverse is not necessarily true.

Figure 5.1(c) illustrates a pairwise compatibility tree of G in Fig. 5.1(a)with $n_{rd} = 0$. Given

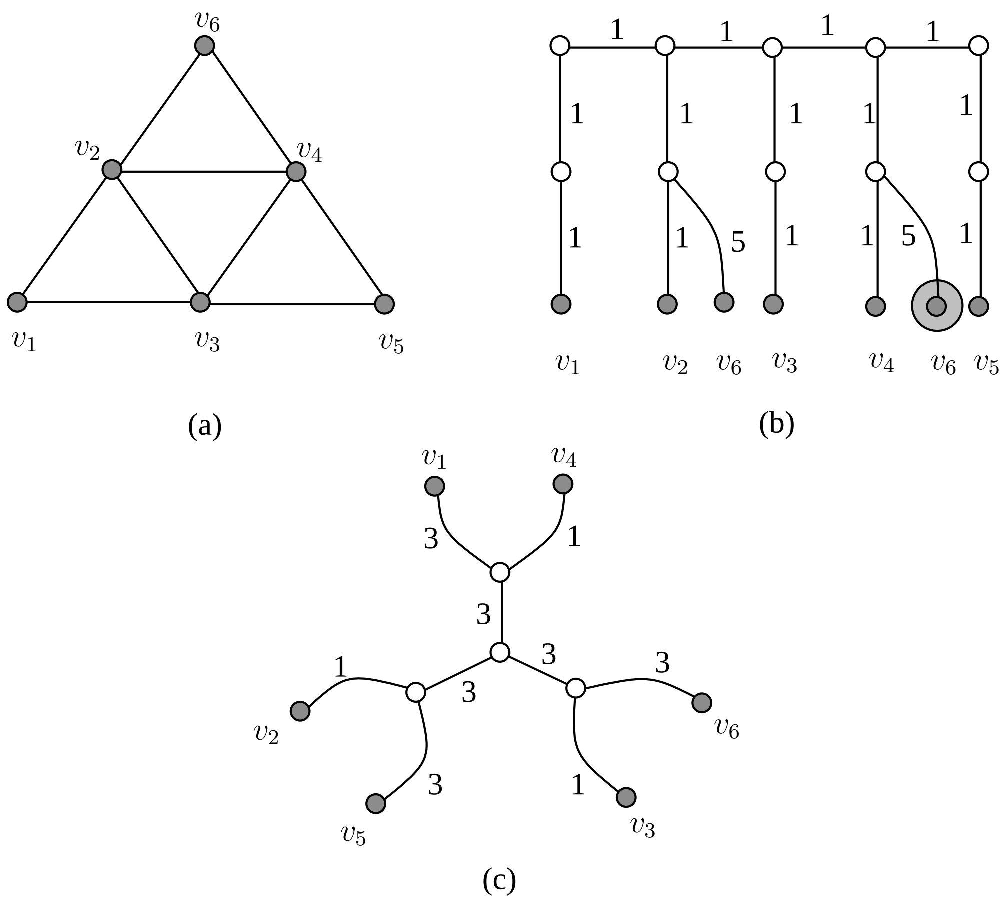

Figure 5.1: (a) A graph G, (b) an improper pairwise compatibility tree of G and (c) a pairwise compatibility tree of G.

a graph G, the *improper pairwise compatibility tree construction* problem asks to construct an edge weighted tree T with $n_{rd} \geq 0$ such that G is an improper PCG of T for suitable d_{min} and d_{max}.

The concept of improper pairwise compatibility graph not only opens up an interesting field of theoretical research, but also possesses some significant practical values. Phylogenetic tree deals with the evolutionary histories of groups of organisms that play a major role in representing the interrelationships among biological entities. But the problem posed by phylogenetics is that genetic data are only available for the present, and fossil records are sporadic and less reliable. Hence what we can do is to use our knowledge of how evolution operates to reconstruct the phylogenetic tree [SE67]. Unfortunately, however, evolutionary history is not something we can see. It has only happened once and only leaves behind clues as to what happened. Systematists use these clues to try to reconstruct evolutionary history. Thus, a phylogenetic tree is based on a hypothesis of the order in which evolutionary events are assumed to have occurred. Hence, nothing is absolute in phylogenetics rather everything is tentative. Therefore, biologists are interested in revealing the tentative patterns of relationships among organisms. In this connection, improper pairwise compatibility tree may serve the purpose of modeling the tentative evolutionary history among different biological entities at the cost of some redundancies. As far as the hypothetical nature of phylogenetic tree is concern, these redundancies will not hamper the tentative pattern of relationships. Moreover, since constructing pairwise compatibility tree is difficult and, in some cases, impossible, improper pairwise compatibility tree can be an effective alternative in modeling evolutionary history.

5.2 Every Graph is an Improper PCG

In this section we show that every graph is an improper PCG, as in the following theorem.

Theorem 5.2.1 *Every graph is an improper PCG.*

Proof. Let $G = (V, E)$ be any graph. Let n_i be the number of neighbors of a vertex $u_i \in V$ and these neighbors are $u_{i1}, u_{i2}, u_{i3}, \ldots, u_{in_i}$. We first construct a star for each vertex $u_i \in V$ with $n_i + 1$ leaves where one leaf corresponds to the vertex u_i and other leaves correspond to its neighbors. We next assign 1 as the weight of the edge incident to the leaf corresponding to u_i, and 2 as the weight of each edge incident to the leaves, that correspond to the neighbors of u_i. Finally, we connect all the bases of these stars to a single node through edges of weight 1 and let the resulting tree be T. Let $d_{min} = d_{max} = 3$. One can easily verify that the distance between two leaves in T corresponding to two adjacent vertices in G is 3. Otherwise, the distance is greater than 3. Thus T is the pairwise compatibility tree of G with some redundancies. Hence G is an improper PCG. $\mathcal{Q.E.D.}$

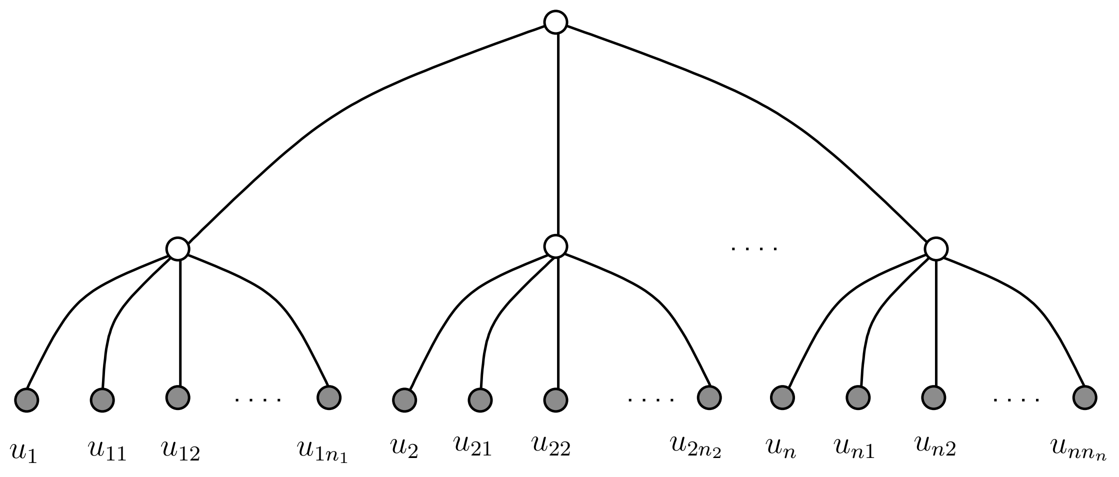

Figure 5.2: An improper pairwise compatibility tree of any graph with n vertices.

This construction process is illustrated in Fig. 5.2. Figure 5.3 illustrates the pairwise compatibility tree of K_n obtained by this method where vertices are labeled as $1, 2, 3, \ldots, n$. This is the worst case of our construction where the required number of leaves is n^2 for a graph with n vertices. And the total number of nodes is $n^2 + n + 1$ (including leaves and all other intermediate nodes).

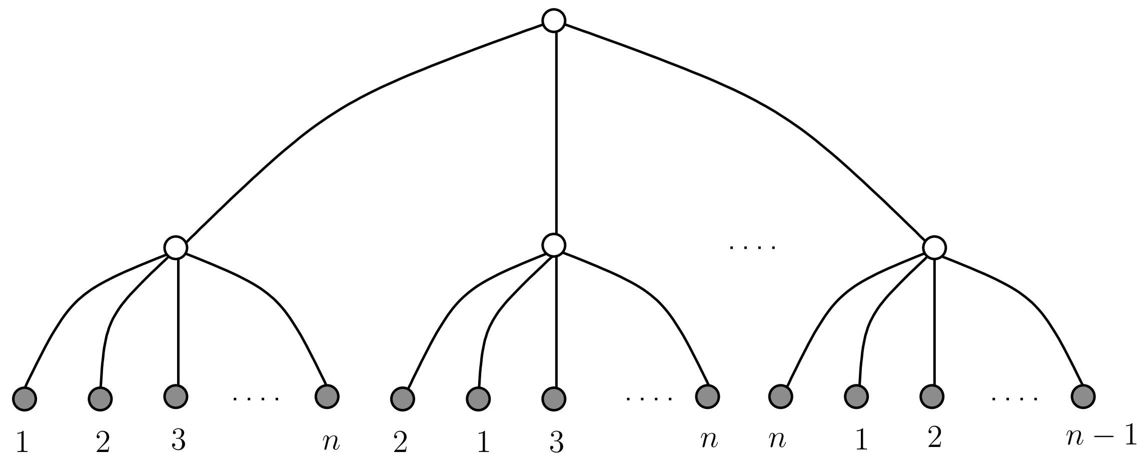

Figure 5.3: An improper pairwise compatibility tree of K_n.

To reduce the number of redundancies, we can take the following strategy. 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 since the connectivity of u and v is already taken in account in the star corresponding to v. We can do so by labeling the vertices of G numerically

$(1, 2, 3, \ldots, n)$ and constructing the star for the lowest numbered vertex first, then for the second lowest and so on. We can now exclude all lower numbered neighbors of any vertex u while constructing its corresponding star. Figure 5.4 illustrates the tree constructed by this method for K_n. Here total number of leaves is, $n + (n-1) + (n-2) + \ldots + 2 = n(n-1)/2 - 1$ and total number of nodes is $n(n-1)/2 - 1 + (n-1) + 1 = (n-1)(n+2)/2$. Hence the following theorem holds.

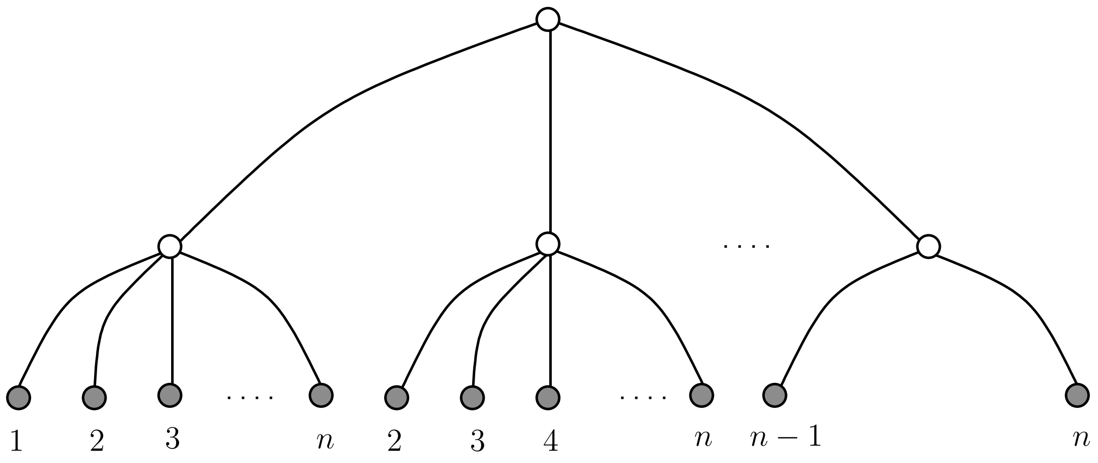

Figure 5.4: Improved construction of improper pairwise compatibility tree of K_n.

Theorem 5.2.2 *An improper pairwise compatibility tree of a graph with n vertices can be constructed with at most $\frac{n(n-1)}{2} - (1 + n)$ number of redundancies.*

5.3 Triangulated Planar Graphs

In this section we discuss some triangulated plane graphs that are *PCG*s. These results will be used in our heuristic algorithm for constructing improper pairwise compatibility tree for any triangulated plane graphs.

The pairwise compatibility tree T of a triangle, as shown in Fig 5.5(a), is a caterpillar as illustrated in Fig. 5.5(b). We assign 1 as the weight of each edge. Let $d_{min} = 3$ and $d_{max} = 4$. Now one can easily verify that T is the pairwise compatibility tree of the triangle.

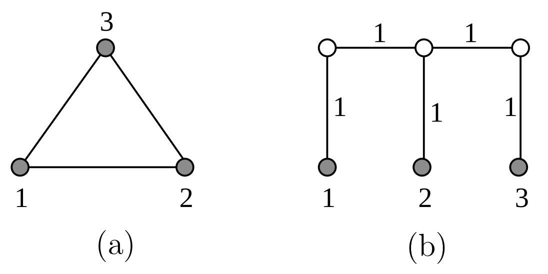

Figure 5.5: (a) A triangle G and (b) a PCG of G.

We now merge multiple triangles in two different ways. First, we merge arbitrarily large number of triangles in such a way 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 numbers, i.e, for an shared edge uv, if u gets label n then the label of v is $n - 1$ or $n + 1$. We call this sort of numbering *sequential numbering*. We denote this type of merging as $type1$. Fig. 5.6(a) shows an example of this sort of graphs. To construct the corresponding pairwise compatibility tree we merge the individual caterpillars for each triangle as shown in Fig. 5.5(b) and let T be the resulting tree as illustrated in Fig. 5.6(b). Values for d_{min} and d_{max} remains the same as it was for a single triangle, i.e, 3 and 4 respectively.

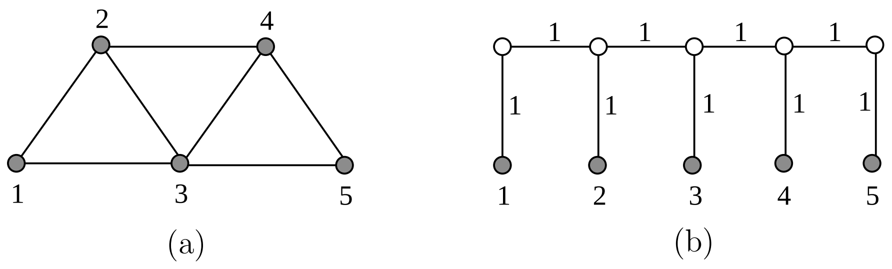

Figure 5.6: (a) Merging of triangles ($type1$), and (b) its PCG.

We now consider another type of merging which we call $type2$. In this case, no two triangles share any edge rather they share only one vertex as shown in Fig. 5.7(a). In this case, we construct a caterpillar for each triangulated face in the same way as illustrated in Fig. 5.5(b) and subdivide the edge that connects a leaf to the spine of the caterpillar. We now merge these caterpillars as in Fig. 5.7(b) and let the resulting tree be T. We assign 1 as the weight of each edge except the edge by which two caterpillars are merged – where we assign 2. Let $d_{min} = 5$

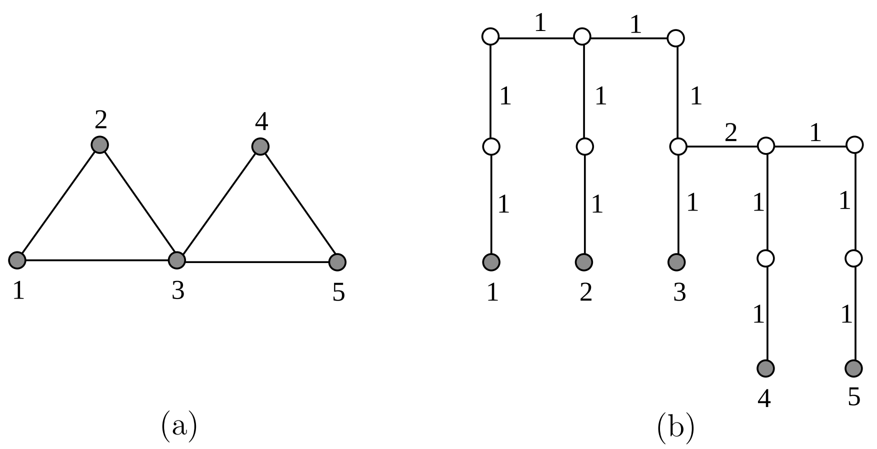

Figure 5.7: (a) Merging of triangles (*type*2), and (b) its *PCG*.

and $d_{max} = 6$. We can see that $d(1,2) = d(2,3) = d(3,4) = 5, d(1,3) = 6$ that are within the range but $d(1,4) = d(2,5) = 9$ which is out of the range. Hence T is the pairwise compatibility tree of the graph as illustrated in Fig. 5.7(a).

We now consider both of these two types simultaneously. Fig. 5.8(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 admits an sequential numbering, i.e., endpoints are labeled with consecutive numbers. Now using the corresponding construction processes of these two types (*type*1 and *type*2), we can construct the pairwise compatibility tree T of the graph shown in Fig. 5.8(a). Fig. 5.8(b) illustrates such construction. One can now easily verify that T is pairwise compatibility graph of G for $d_{min} = 5$ and $d_{max} = 6$. Hence the following theorem holds.

Theorem 5.3.1 *Let G be a triangulated plane graph such that two faces of G can share a vertex, or can share an edge e which admits a sequential numbering. Then G is a pairwise compatibility graph.*

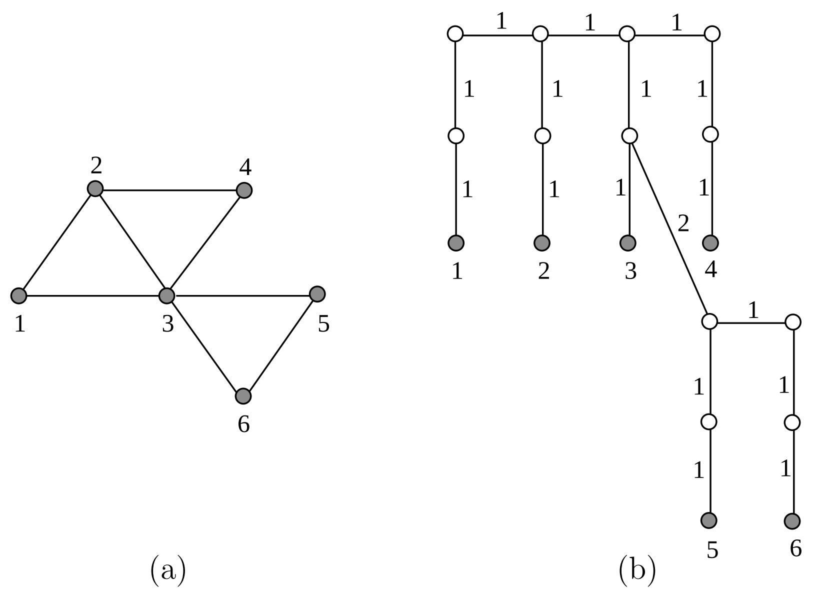

Figure 5.8: (a) A graph G combining the criteria of *type*1 and *type*2, and (b) a pairwise compatibility tree T of G.

5.4 A Heuristic Algorithm for Improper PCG

For any graph G, an improper pairwise compatibility tree T can be constructed with $n_{rd} \geq 0$. If there exists a tree with $n_{rd} = 0$, then G is a PCG. Since we have already shown that not all graphs are PCGs, it is not always possible to construct a tree T with $n_{rd} = 0$. In this context, constructing an improper pairwise compatibility tree with the minimum number of redundancies is an important concern. Motivated by this problem, we now give a heuristic algorithm to efficiently construct an improper pairwise compatibility tree of any triangulated plane graph with smaller number of redundancies.

Let, $G = (V, E)$ be a triangulated plane graph. First, we construct the inner dual $G^* = (V^*, E^*)$ of G. Let G^* is consisting of the subgraphs $G_1^*, G_2^*, G_3^*, \cdots, G_n^*$. For every G_i^*, we construct the improper pairwise compatibility tree for the subgraph of G corresponding to G_i^* as follows. We traverse G_i^* starting from an arbitrary vertex according to depth first traversal (although any kind of traversal will do). While moving from u to v where $u, v \in V(G_i^*)$, we cross an edge $e \in E$ of G. Let f_u and f_v be the faces of G corresponding to u and v of G^*, respectively. If e admits sequential numbering then we merge the tree of the face f_v with the tree corresponding to the face f_u according to the procedure as illustrated in Fig. 5.6. Otherwise we delete one of the two edges (e_1, e_2) of f_v, other than e, as follows. If only one of the $e_i, i \in 1, 2$ is shared with other face then we delete the non-shared edge. If both of the edges are shared, or not shared with other faces then any one of the e_1 or e_2 can be deleted. Let e_d be the edge to be deleted. If e_d is not shared with any other face then v is a leaf of G^*. Then after deletion of e_d, there is just one edge connected to the face as illustrated in Fig. 5.9(a) and we can extend the tree as illustrated in Fig. 5.9(b). If e_d is shared with a face f_d then we traverse

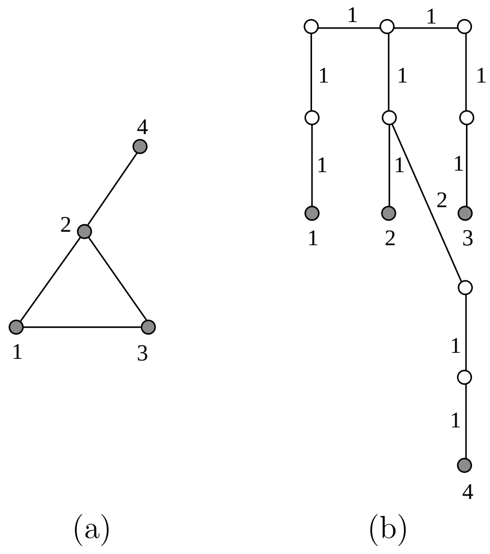

Figure 5.9: (a) A graph G, and (b) a pairwise compatibility tree T of G.

the vertex corresponding to the face f_d. In this case, f_d is connected with the subgraph, the faces of which are already traversed, according to *type*2 (see Fig. 5.7) and we merge the tree for f_d accordingly without deleting any edge of f_d. In this way we traverse the tree T_i and build the pairwise compatibility tree incrementally. Note that, so far, no redundancy has been introduced. We now consider the edges that were deleted. Let, $uv \in E$ be an deleted edge. Then either u or v is present in the improper pairwise compatibility tree, i.e., either u or v corresponds to a leaf. Without loss of generality let u be that vertex. We then introduce a leaf for v and connect it to the parent of the leaf corresponding to u through a path of weight 4 as illustrated in Fig. 5.9(b). Note that the subgraphs of G corresponding to the subgraphs of G^* are connected as *type*2. Hence we can merge the improper pairwise compatibility trees of G_i^*,

$i \in 1, 2, 3, \cdots, n$ according to the procedure as illustrated in Fig. 5.7, and let T be the resulting tree. Now, T is the improper pairwise compatibility tree of G for $d_{min} = 5$ and $d_{max} = 6$. We call our algorithm **Improper-Triangular**.

Fig. 5.10 demonstrates the execution of Algorithm **Improper-Triangular** for an input graph G as illustrated in Fig. 5.10(a). In this figure, triangulated faces of G that are drawn by dotted lines are yet to be traversed, and the traversed faces have been drawn by solid lines. Similarly, the untraversed edges of G^* are drawn by grey lines whereas the traversed edges of G^* are black. The edges of G that has to be deleted (the edge incident between 4 and 6 in this example) are indicated by dashed lines. Figure 5.10(j) shows the improper pairwise compatibility tree of G.

We now have the following theorem on the upper bound of the number of redundancies need to be introduced in Algorithm **Improper-Triangular**.

Theorem 5.4.1 *Algorithm* ***Improper-Triangular*** *constructs an improper pairwise compatibility tree with at most $f/2$ number of redundancies where f denotes the number of faces of G.*

Proof. Let G be a triangulated plane graph and G^* be the inner dual of G. In Algorithm **Improper-Triangular**, while traversing G^*, we need to cross an edge e of G. If e admits a sequential numbering then no edge is deleted, and the corresponding pairwise compatibility tree can be constructed as *type*1. Otherwise we delete an edge e_d which in turn results into a redundancy. After deletion of an edge e_d, the next face f_d of G on the traversal is connected as *type*2, for which the corresponding tree can be constructed without deleting any edge, i.e, by introducing no redundancy. This implies that at most one redundancy may be introduced for two consecutive faces of G on the traversal. Thus the total number of edge deletion which is equal to the number of redundancies is no more than $f/2$ where f denotes the number of faces in G. Therefore, Algorithm **Improper-Triangular** constructs an improper pairwise compatibility tree for any triangulated plane graph with at most $f/2$ redundancies. $\mathcal{Q.E.D.}$

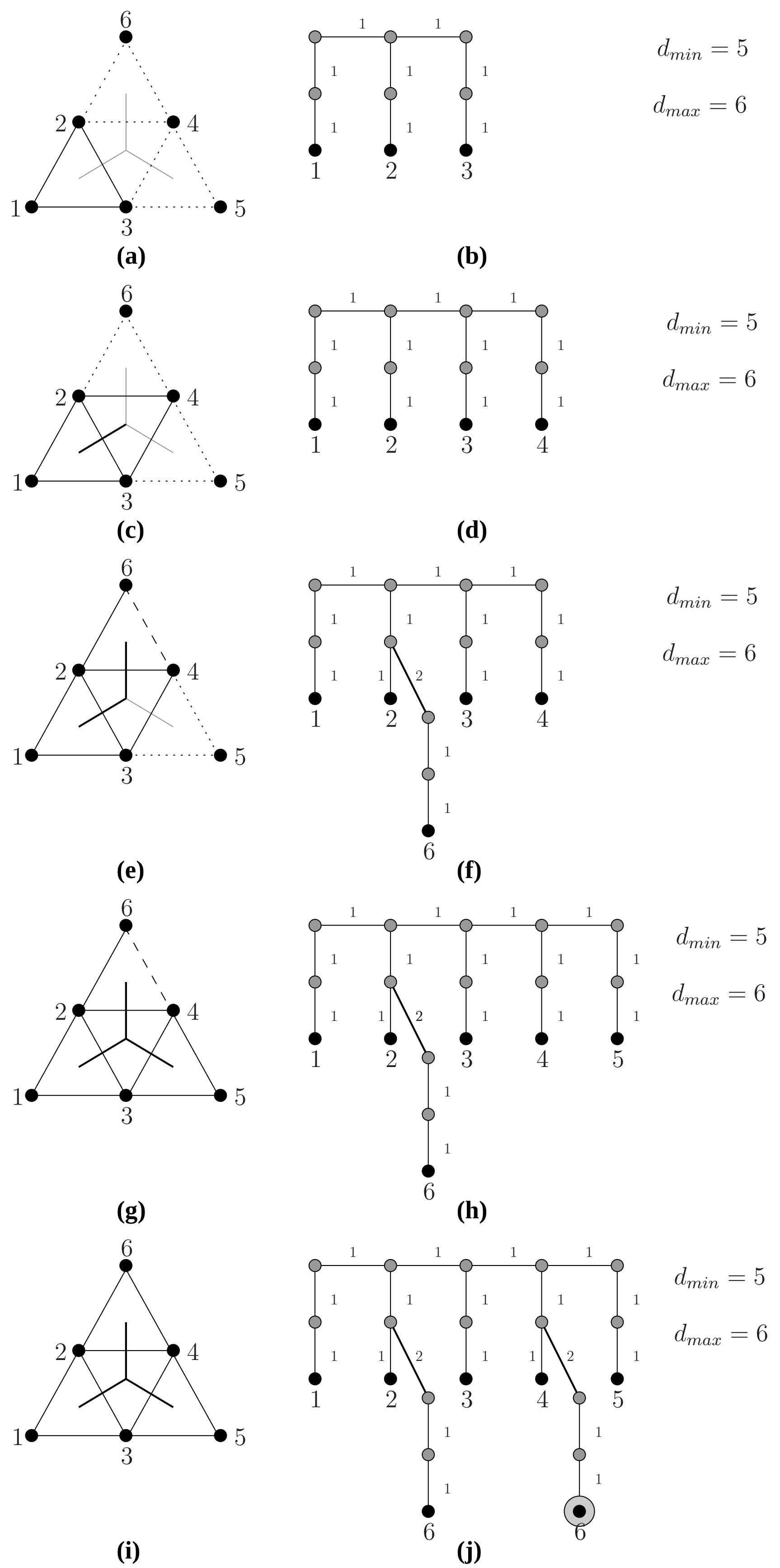

Figure 5.10: Construction of the improper pairwise compatibility tree according to the heuristic algorithm.

5.5 Summary

In this chapter we have introduced the new notion of improper PCGs. We have showed that all graphs are improper PCGs. We have also presented an efficient heuristic algorithm to construct an improper pairwise compatibility tree for any triangulated plane graphs.

Chapter 6

Conclusion

In this thesis we have dealt with different theoretical aspects of pairwise compatibility graphs. We have started with an introductory overview on phylogeny and pairwise compatibility graphs in Chapter 1. In that chapter we have given a precise definition of pairwise compatibility graphs and discussed various practical applications of this graph. We also gave a literature review and described our objective of this thesis.

In Chapter 2 we have introduced the preliminary ideas on graph theory and on pairwise compatibility graphs. We have also discussed tree power graphs and two of its extensions, and complexity theory in detail in this chapter.

In Chapter 3 we have resolved the open question regarding whether or not every graph is a pairwise compatibility graph. We have showed that not all graphs are pairwise compatibility graphs. we have also studied the pairwise compatibility graph recognition problem and established two restricted classes of bipartite graphs as PCG.

In Chapter 4 we have showed that tree power graphs, Steiner k-power graphs and phylogenetic k-power graphs are pairwise compatibility graphs.

In Chapter 5 we have introduced a new notion called improper PCG. We have showed that all graphs are improper PCGs. We have also presented a heuristic algorithm to construct an improper pairwise compatibility tree of any triangulated plane graph.

This thesis goes a long a way to address different important and challenging problems regarding pairwise compatibility graphs. However, the following problems are still open and remained as future works.

1. To study the complete characterization of pairwise compatibility graph. The main challenge here, from the point of view of theoretical research, is to find a necessary and sufficient condition for a graph to be a PCG.

2. To develop an algorithm that can construct a pairwise compatibility tree (if exists) for a given graph. Since there may be multiple pairwise compatibility trees for a single graph, researchers can also put their heads to find the one which describes the evolutionary process more significantly than the others. As this is likely to be NP-hard (as is almost everything in phylogeny estimation), heuristics for these problems need to be developed. Furthermore, since the internal nodes of evolutionary trees are hypothetical, we first need to define the criteria based on which we can classify a tree topology as significant one.

3. To study the complexity of testing whether a given graph is a pairwise compatibility graph or not.

4. To find the smallest graph class that encompasses all of the pairwise compatibility graphs.

5. To find the number of pairwise compatibility graphs on n vertices.

Publications

1. Muhammad Nur Yanhaona, Md. Shamsuzzoha Bayzid and Md. Saidur Rahman, *Discovering Pairwise Compatibility Graphs*, Proc. of The 16th Annual International Computing and Combinatorics Conference (COCOON 2010), Lecture Notes in Computer Science, 6196, pp. 399–408, 2010 (to appear).

   This paper has been invited for submission to a special issue of Discrete Mathematics, Algorithms and Applications.

2. Muhammad Nur Yanhaona, Md. Shamsuzzoha Bayzid and Md. Saidur Rahman, *Not All Graphs are Pairwise Compatibility Graphs*, Proc. of The 3rd Annual Meeting of the Asian Association for Algorithms and Computation (AAAC 2010), pp. 20, 2010.

References

[BBSCF99] R. M. Bush, C. A. Bender, K. Subbarao, N. J. Cox and W. M. Fitch, *Predicting the evolution of human influenza A*, Science, 286, pp. 1921–1925, 1999

[BL09] A. Brandstadt, V. B. Le, *Simplical powers of graphs*, Theoretical Computer Science, 410(52), pp. 5443–5454, 2009

[BP83] R. Balakrishnan and P. Paulraja, *Powers of chordal graphs*, Journal of the Australian Mathematical Society (Series A), 35, pp. 211–217, 1983.

[BM91] D. R. Brooks and D. A. McLennan, *Phylogeny, Ecology and Behaviour*, University of Chicago Presss, Chicago, 1991.

[BBPP99] I. Bomze, M. Budinich, P. Pardalos, and M. Pelillo, *The maximum clique problem*, Handbook of Combinatorial Optimization, 4, pp. 1–74, 1999.

[CLRS03] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, *Introduction to Algorithms*, The MIT Press, Cambridge, 2001.

[D91] P. Damaschke, *Distances in cocomparability graphs and their powers*, Discrete Applied Mathematics, 35(1), pp. 67–72, 1991.

[DPV06] S. Dasgupta, C. H. Papadimitriou and U. V. Vazirani, *Algorithms*, McGraw-Hill, 2006.

[F74] H. Fleischner, *The Square of every two-connected graph is Hamiltonian*, Journal of Combinatorial Theory (Series B), 16(1), pp. 29–34, 1974.

[FMRST08] Michael R. Fellows, Daniel Meister, Frances A. Rosamond, R. Sritharan and Jan Arne Telle, *Leaf Powers and Their Properties: Using the Trees*, Proc. of the 19th International Symposium on Algorithms and Computation, Lecture Notes in Computer Science, 5369, pp. 402 – 413, 2008.

[GJ79] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, New York, 1979.

[HP91] P. H. Harvey and M. D. Pagel, *The Comparative Method in Evolutionary Biology*, Oxford University Press, Oxford, 1991.

[JP04] N. C. Jones, P. A. Pevzner, *An Introduction to Bioinformatics Algorithms*, The MIT Press, Cambridge, 2004.

[KC98] P. E. Kearney and D. G. Corneil, *Tree Powers*, Journal of Algorithms, 29, pp. 111–131, 1998.

[KT05] J. Kleinberg and E. Tardos, *Algorithm Design*, Addison Wesley, 2005.

[KMP03] P. Kearney, J. I. Munro and D. Phillips, *Efficient generation of uniform samples from phylogenetic trees*, Proc. of WABI 2003, LNBI 2812, Springer, pp. 177–189, 2003.

[L02] A. M. Lesk, *Introduction to Bioinformatics*, Oxford University Press, Great Clarendon Street, Oxford, 2002.

[L80] H. T. Lau, *Finding a Hamiltonian Cycle in the Square of a Block*, Ph. D. Thesis, School of Computer Science, McGill University, Montreal, 1980.

[L92] N. Linial, *Locality in distributed graph algorithms. SIAM Journal on Computing*, 21(1), pp. 193–201, 1992.

[LKJ00] G. H. Lin, P. E. Kearney, and T. Jiang, *Phylogenetic k-root and Steiner k-root*, Proc. of International Conference on Algorithms and Computation (ISAAC 2000), LNCS 1969, Springer, pp. 539–551, 2000.

[L06] L. c. Lau, *Bipartite roots of graphs*, ACM Transactions on Algorithms, 2(2), pp. 178-208, 2006.

[LS83] R. Laskar and D. Shier, *On powers and centers of chordal graphs*, Discrete Applied Mathematics, 6(2), pp. 139–147, 1983.

[LS95] Y-L. Lin and S. S. Skiena, *Algorithms for Square Roots of Graphs*, SIAM Journal of Discrete Mathematics, 8(1), pp. 99 – 118, 1995.

[LW05] C.R. Linder and T. Warnow, *Overview of phylogeny reconstruction*, Book chapter, in S. Aluru (editor), Handbook of Computational Biology, Chapman & Hall, CRC Computer and Information Science Series, 2005.

[MMLPGH02] M. L. Metzker, D. P. Mindell, X. M. Liu, R. G. Ptak, R. A. Gibbs and D. M. Hillis, *Molecular evidence of HIV-1 transmission in a criminal case*, Proc. of the National Academy of Sciences of the United States of America, 99(22), pp. 14292–14297, 2002.

[NC88] T. Nishizeki and N. Chiba, *Planar graphs: theory and algorithms*, North-Holland, Amsterdam, 1988.

[MS94] R. Motwani and M. Sudan, *Computing Roots of Graphs is Hard*, Discrete Applied Mathematics, Elsevier Science Publishers B. V., 54(1), pp. 81–88, 1994.

[NR04] T. Nishizeki and M. S. Rahman, *Planar Graph Drawing*, Lecture Notes Series on Computing, 12, World Scientific Publishing Company, 2004.

[P02] D. Phillips, *Uniform sampling from phylogenetic trees*, Master's thesis, University of Waterloo, August 2002.

[PX94] M. Pardalos and J. Xue, *The maximum clique problem*, Journal of Global Optimization, 4(3), pp. 301–328, 1994.

[R87] A. Raychaudhuri, *On powers of interval and unit interval graphs*, Congr. Numer., 59, pp. 235–242, 1987.

[R92] A. Raychaudhuri, *On powers of strongly chordal and circular arc graphs*, Ars Combinatoria, 34, pp. 147–160, 1992.

[S60] M. Skiena, *On an ordering of the set of vertices of a connected graph*, Technical Report No. 412, Publ. Fac. Sci. Univ. Brno, 1960.

[SE67] L. L. Cavalli-Sforza and A. W. F. Edwards, *Phylogenetic analysis: models and estimation procedures*, Evolution, 21, pp. 550–570, 1967.

[W03] D. B. West, *Introduction to Graph Theory*, Prentice Hall of India, New Delhi, 2003.

[W09] T. Warnow, Introduction to Phylogenetic Estimation Algorithms, Presentation, University of Texas at Austin, 26 February, 2009. (http://userweb.cs.utexas.edu/∼phylo/research/resources.html), last accessed on May 10, 2010.

[YH07] M. N. Yanhaona and K. S. M. T. Hossain, *Applications of Graph Theory in Bioinformatics*, B. Sc. Thesis, Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, 2007.

[YHR09] M. N. Yanhaona, K. S. M. T. Hossain and M. S. Rahman, *Pairwise compatibility graphs*, Journal of Applied Mathematics and Computing, 30, pp. 479–503, 2009.

Index