

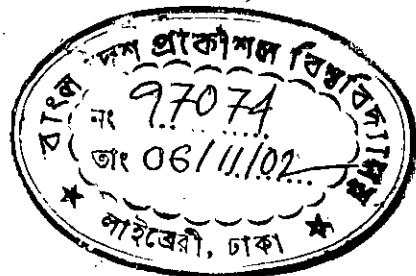
NP-Completeness of Edge-Ranking Problem for Series-Parallel Graphs

by

Md. Emdadul Haque

Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology

October, 2002

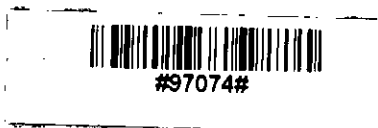


Submitted to

Bangladesh University of Engineering and Technology

in partial fulfillment of the requirements for

M. Sc. Engineering (Computer Science and Engineering)



NP-COMPLETENESS OF EDGE-RANKING PROBLEM FOR SERIES-PARALLEL GRAPHS

A Thesis Submitted by

MD. EMDADUL HAQUE

Student No. 9605045F

for the partial fulfillment of the degree of
M. Sc. Engineering (Computer Science and Engineering).
Examination held on October, 2002.

Approved as to style and contents by:

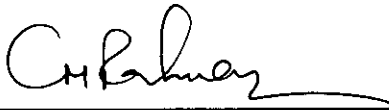


DR. MD. ABUL KASHEM MIA
Associate Professor and Head
Department of Computer Science and Engineering
B.U.E.T., Dhaka-1000, Bangladesh.

Chairman
Supervisor
and
Ex-officio

DR. M. KAYKOBAD
Professor
Department of Computer Science and Engineering
B.U.E.T., Dhaka-1000, Bangladesh.

Member



DR. CHOWDHURY MOFIZUR RAHMAN
Professor
Department of Computer Science and Engineering
B.U.E.T., Dhaka-1000, Bangladesh.

Member

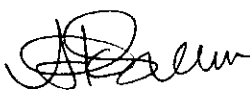


DR. SAIFUR RAHMAN
Professor
Department of Electrical and Electronic Engineering
B.U.E.T., Dhaka-1000, Bangladesh.

Member
(External)

Certificate

This is to certify that the work presented in this thesis paper is the outcome of the investigation carried out by the candidate under the supervision of Dr. Md. Abul Kashem Mia in the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka. It is also declared that neither of this thesis nor any part thereof has been submitted or is being concurrently submitted anywhere else for the award of any degree or diploma.



Signature of the
Supervisor



Signature of the
Author

Contents

Acknowledgements	IX
Abstract	1
1 Introduction	2
1.1 Background	2
1.2 Edge-Ranking Problem	3
1.2.1 Application of Edge-Ranking Problem	5
1.2.2 The c -Edge-Ranking Problem	6
1.3 Vertex-Ranking Problem	9
1.4 Summary	10
2 Preliminary	12
2.1 Graphs and Multigraphs	12
2.1.1 Directed Graph	13
2.1.2 Undirected Graph	14
2.1.3 Paths and Cycles	14
2.1.4 Connected Graphs	14
2.1.5 Subgraph	15
2.2 Tree	15
2.2.1 Binary Tree	17
2.3 Series-Parallel Graph	17
2.4 Edge-Ranking	19
2.4.1 Minimal Cut and Primitive Separator	19

2.4.2	Terminal Edge and Internal Edge	21
2.4.3	Edge Multiplicity and Normal Form	22
2.5	The Theory of NP-Completeness	22
2.5.1	Polynomial Time Algorithms and Intractable Problems	22
2.5.2	Decision Problems	23
2.5.3	Deterministic and Nondeterministic Algorithms	23
2.5.4	The Class P	24
2.5.5	The Class NP	24
2.5.6	Reducibility	24
2.5.7	NP-Completeness	25
2.5.8	NP-Completeness Proof Technique	25
2.5.9	Relation among P, NP and NPC	27
2.6	Conclusion	27
3	The Main Theorem	28
3.1	Introduction	28
3.2	MINIMUM GRAPH BISECTION Problem	29
3.3	The Edge-Ranking Problem is in NP	29
3.4	The EDGE-RANKING Problem	30
3.5	Graph Construction	30
3.6	Conclusion	36
4	Conclusion	37
	References	39
	Index	42

List of Figures

1.1 An optimal edge ranking of a graph and corresponding edge-separator tree	4
1.2 An edge ranking of a graph	5
1.3 An optimal 2-edge-ranking of a graph	6
1.4 An optimal 2-edge ranking of a tree	7
1.5 An optimal 2-edge-separator tree of a tree in Fig 1.4	8
1.6 An optimal vertex-ranking of a graph G	10
2.1 Directed and undirected graphs. (a) A directed graph, (b) An undirected graph, (c) An induced subgraph	13
2.2 A tree with seven vertices	16
2.3 Series (a) and parallel (b) connections	18
2.4 A series-parallel graph with detail connection	18
2.5 A series-parallel graph	18
2.6 An edge-ranking of the series-parallel graph	19
2.7 The edge-ranking in (a) satisfies the minimal cut property, but (b) does not satisfy the minimal cut property	21
2.8 How most theoretical computer scientist view the relationships among P, NP, and NPC. Both P and NPC are wholly contained within NP, and $P \cap NPC = \emptyset$	27
3.1 An example of MINIMUM GRAPH BISECTION problem	29
3.2 (a) the graph G_1 and G_2 and (b) the connector graph H (c) joining G_1 and G_2 with H (d) illustrates total cut of H	31

3.3 (a) A binary tree T with $c = 7$ leaves and height $\lceil \log c \rceil = 3$, and (b) optimal edge-ranking of T using five ranks	32
3.4 A multigraph G_1 constructed from T in Fig. 3.3	33
3.5 A connector graph H with $c = 5$ edges	33
3.6 The series-parallel graph G	34

List of Tables

1.1 Some known results related to edge-ranking problem	11
--	----

Acknowledgements

Foremost, I would like to express my gratitude to my thesis supervisor Dr. Md. Abul Kashem Mia, Associate Professor and Head, Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka-1000. Keen interest of Dr. Kashem in graph ranking has influenced me to carry out a research in edge-ranking of series-parallel graphs. He has always been a great resource for ideas and solution; and his encouragement and support have made difficult times during my thesis exploration less frustrating. His valuable advice, constructive criticism and constant supervision at all stages of this research have made it possible to complete this research.

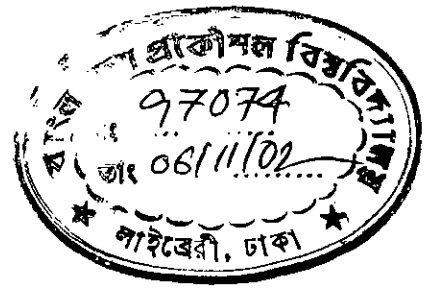
I would like to sincerely thanks the other members of my graduate committee Dr. M. Kaykobad and Dr. Chowdhury Mofizur Rahman, Professors, Department of Computer Science and Engineering, BUET, and Dr. Saifur Rahman, Professor, Department of Electrical and Electronic Engineering, BUET for their suggestions.

I would like to thanks the all-out cooperation and services rendered by the faculty members and staff of the CSE department for many things.

Finally, I would like to thanks my parents for instilling upon me the value of education at young age, and for always encouraging me to pursuit my aspirations.

Abstract

The general goal of this thesis is to prove that the edge-ranking problem is NP-complete for series-parallel graphs. An edge-ranking of a graph G is a labeling of its edges with positive integers such that every path between two edges with the same label i contains an intermediate edge with label $j > i$. An edge-ranking is optimal if it uses the least number of labels among all possible edge-rankings. Many combinatorial problems on general graphs are NP-Complete, but when restricted to series-parallel graphs, many of these problems can be solved in polynomial time. On the other hand, very few problems are known to be NP-complete for series-parallel graphs. These include the subgraph isomorphism problem and the bandwidth problem. Moreover, the subgraph isomorphism problem and the bandwidth problem are NP-complete even for trees. In this thesis, we show that the edge-ranking problem is NP-complete for series-parallel graphs, although the problem can be solved in linear-time for trees.



Chapter 1

Introduction

In this chapter we provide the necessary background and motivation for the study on the edge-ranking of graphs. In Section 1.1 we give the background of the study of NP-complete theory and the edge-ranking problem. We define edge-ranking problem and c -edge-ranking problem in Section 1.2. In Section 1.3 we define the vertex-ranking problem. We conclude by summarizing the new result and known result in Section 1.4.

1.1 Background

NP-completeness theory is one of the most important theoretical developments of algorithm research since its introduction in the early 1970. Its importance arises from the fact that the results have meaning for all researchers who are developing computer algorithms, not only computer scientist but also the electrical engineers, operation researchers etc. A wide variety of common encountered problems from mathematics, computer science, and operations

research are known to be NP-complete, and the collection of such problems continuously to grow almost daily. Indeed, the NP-complete problems are now so pervasive that it is important for anyone concentrated with the computational aspect of these fields to be familiar with the meaning and implementations of this concept. This thesis shows that the edge-ranking problem is NP-complete even for series-parallel graphs.

1.2 Edge-Ranking Problem

An *edge-ranking* of a graph G is a labeling of its edges with positive integers such that every path between two edges with the same label i contains an intermediate edge with label $j > i$ [13]. An edge-ranking is optimal if it uses the least number of distinct labels among all possible edge-rankings. The *edge-ranking problem* is to find an optimal edge-ranking of a given graph. Figure 1.1 depicts an example of an optimal edge-ranking of a graph using six ranks, where ranks are drawn next to the edges. Such a ranking corresponds to a minimum height edge-separator tree of G . Iyer *et al.* [13] first studied the edge-ranking problem as they found the problem has an application in scheduling the assembly of multipart products, where vertices of G correspond to the basic parts and the edges correspond to assembly operation or subprocess. They gave an approximate algorithm for solving the edge-ranking problem on trees, and they closed their paper with an open question: whether the edge-ranking problem on trees or graphs is in P or NP-hard. Later, Lam and Yue [17] proved that the edge-ranking problem is NP-hard for general graphs, although de la Torre *et al.* [5] gave an algorithm of $O(n^3 \log n)$ time for finding an optimal edge-ranking of a tree T , where n is the number of vertices in T . Later, Zhou *et al.* [25] gave a better algorithm running in $O(n^2 \log \Delta)$ time, where Δ is the maximum degree of the tree. Recently, Lam and Yue [18] presented a linear-time algorithm to solve the edge-ranking problem on trees.

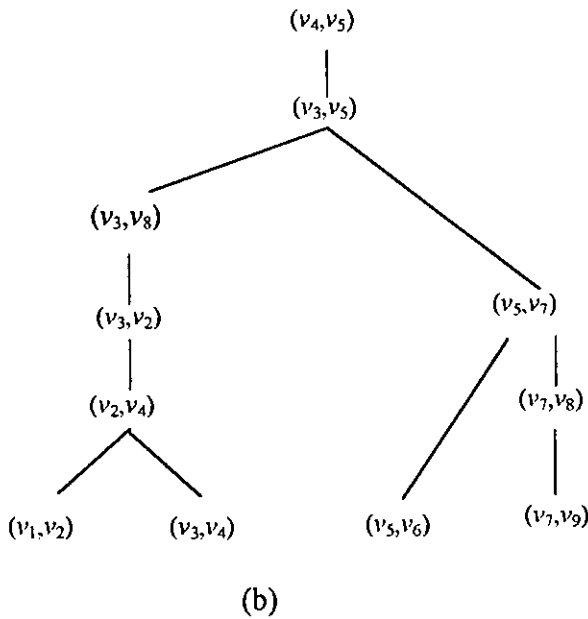
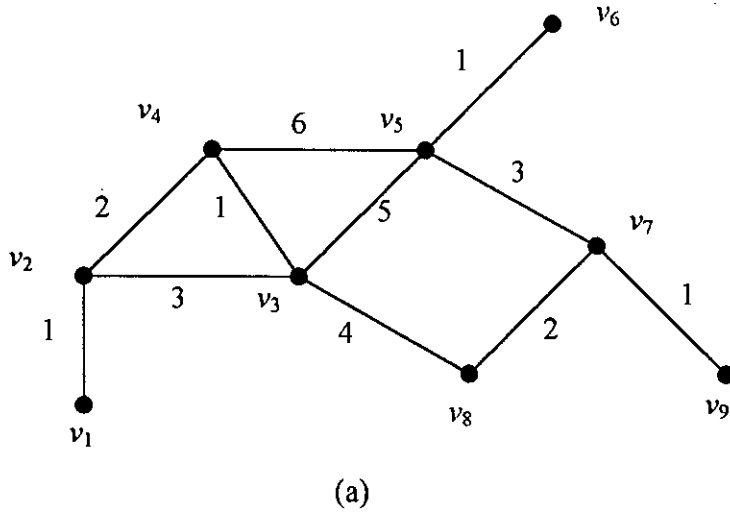


Figure 1.1: (a) An optimal edge ranking of a graph, and (b) corresponding edge-separator tree.

On the other hand, Kashem *et al.* [14] have given a polynomial-time algorithm to solve the generalized edge-ranking problem on partial k -trees with bounded maximum degree. Since a series-parallel graph is a partial 2-tree, their algorithm immediately yields a polynomial-time algorithm for series-parallel graphs with bounded maximum degree. Note that their algorithm

solves the edge-ranking problem for a series-parallel graph G , where the maximum degree of G is constant. In this thesis, we show that the edge-ranking problem is NP-complete for any series-parallel graph.

1.2.1 Application of Edge-Ranking Problem

Edge-ranking of a graph has an important and interesting use in assembly of multipart product. Suppose each edge of the graph shown in Fig 1.2 indicates the sub process in the assembly. And each vertex of the graph indicates the component in the assembly process and the edges corresponds to assembly operations to be performed between the components. How many numbers of steps are needed in the process? The answer of this question can be given by optimally ranking of the edge of the graph. Less number of steps in the process reduces the production cost as well as increases the benefit of the industry. Note that two sub processors cannot share a component at a time. This is why rank of every edge adjacent to a vertex is different. The edges of the following graph shown in Figure 1.2 can optimally rank using five ranks. The optimal edge-ranking of a graph indicates the optimal steps in the assembly process. The edge-ranking problem can also be used in VLSI floor planning and many others processing plants.

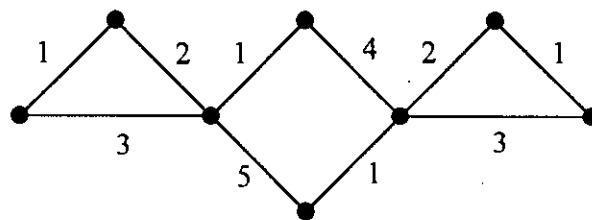


Figure 1.2: An edge ranking of a graph.

1.2.2 The c -Edge-Ranking Problem

A natural generalization of an ordinary edge-ranking is the c -edge-ranking [25]. For a positive integer c , a c -edge-ranking of graph G is a labeling of the edges of G with positive integers such that, for any label i , deletion of all edges with labels $> i$ leaves connected components, each having at most c edges with label i . Clearly an ordinary edge-ranking is a 1-edge-ranking. The minimum number of ranks needed for a c -edge-ranking of G is called the c -edge-ranking number, and is denoted by $r'_c(G)$. A c -edges-ranking of G using $r'_c(G)$ ranks is called an optimal c -edge-ranking of G . The c -edge-ranking problem is to find an optimal c -edge-ranking of a given graph G . Figure 1.3 depicts an optimal 2-edge-ranking of a graph G using three ranks, where the ranks are drawn next to the edges.

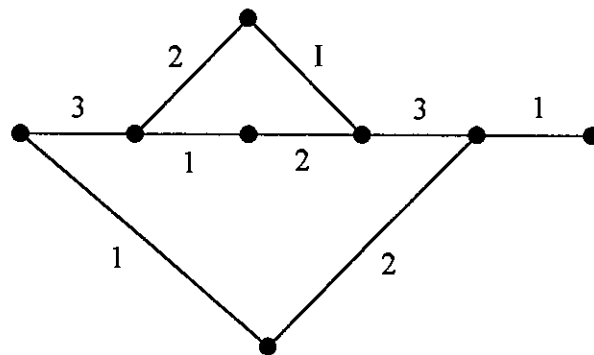


Figure 1.3: An optimal 2-edge-ranking of a graph.

The problem of finding an optimal c -edge-ranking of a graph G has applications in scheduling the parallel assembly of a complex multipart product from its components, where the vertices of G correspond to the basic components and the edges correspond to assembly operations to be performed between the components [8, 12, 13]. Let us consider a robot with $c + 1$ hands which can connect at most $c + 1$ connected components at a time. If we have as many robots as we need, then the problem of minimizing the number of steps required for the parallel assembly of a product using the robots is equivalent to finding an optimal c -edge-ranking of the graph G . One can assemble in parallel a product of Fig. 1.3 in four steps using

robots of two hands. Similarly we can find the 2-edge-ranking for a tree as shown in Figure 1.4.

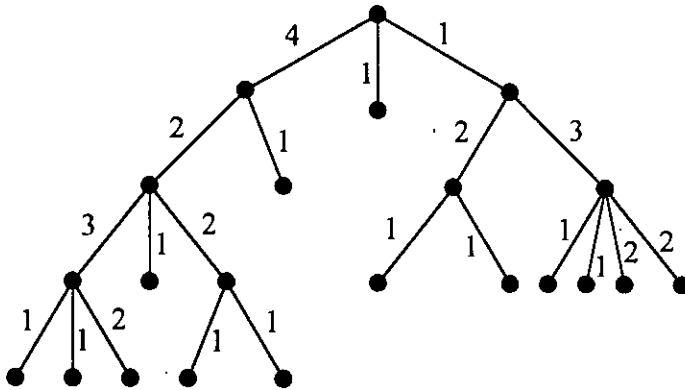


Figure 1.4: An optimal 2-edge ranking of a tree.

The c -edge-ranking problem for a graph G is also equivalent to finding a c -edge-separator tree of G having the minimum height. Consider the process of starting with a connected graph G and partitioning it recursively by deleting at most c edges from each of the remaining connected components until the graph has no edge. The tree representing the recursive decomposition is called a c -edge-separator tree of G . Thus a c -edge-separator tree corresponds to a parallel computation scheme based on the process above, and an optimal c -edge-ranking of G provides a parallel computation scheme having the minimum computation time [21]. Figure 1.5 illustrates a 2-edge-separator tree of the tree depicted in Fig 1.4.

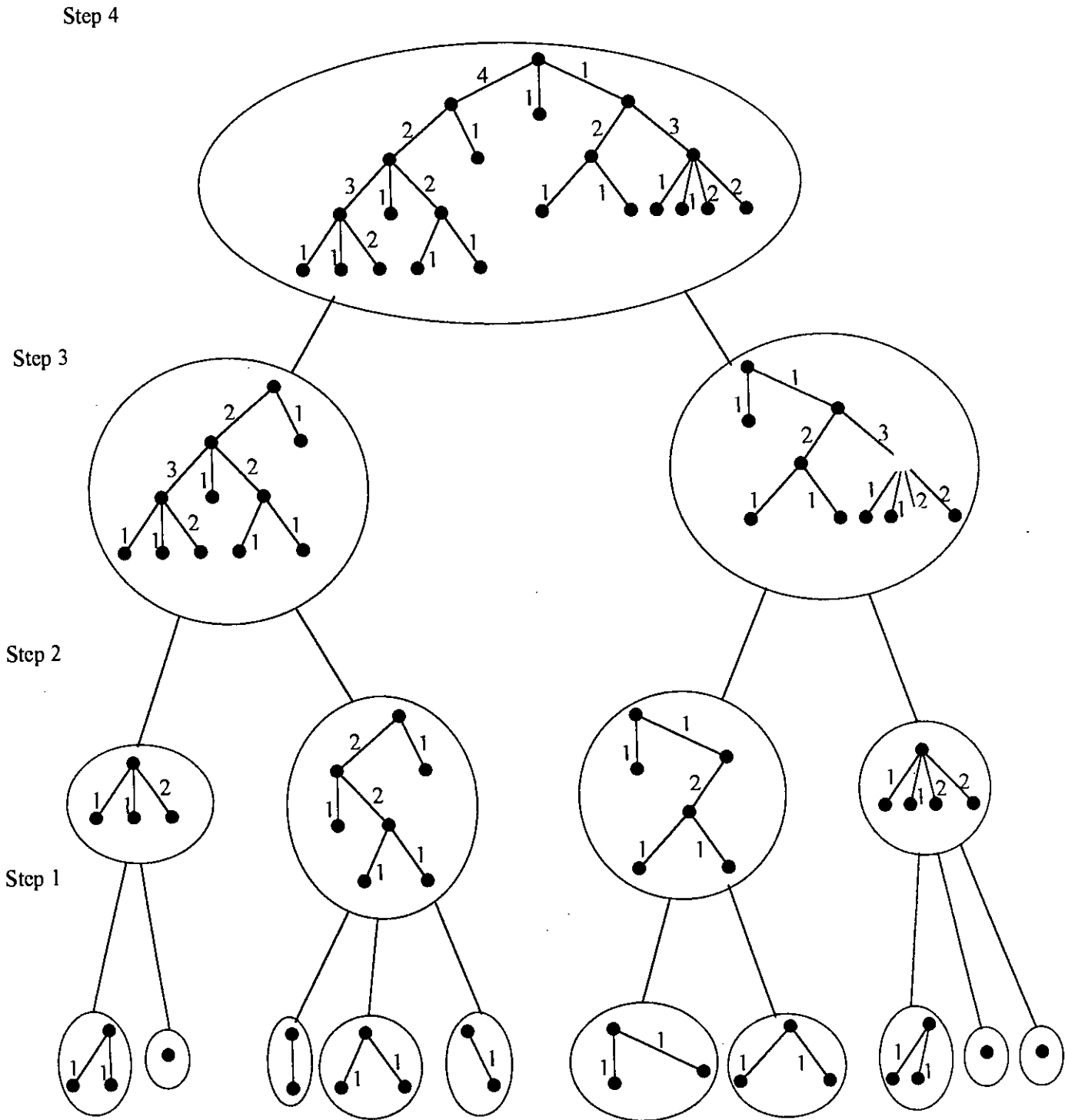


Figure 1.5: An optimal 2-edge-separator tree of a tree in Fig 1.4.

1.3 Vertex-Ranking Problem

An ordinary *vertex-ranking* of a graph G is a labeling (ranking) of the vertices of G with positive integers such that every path between any two vertices with the same label i contains a vertex with label $j > i$ [11]. Clearly a vertex-labeling is a vertex-ranking if and only if, for any label i , deletion of all vertices with labels $> i$ leaves connected components, each having at most one vertex with label i . The integer label of a vertex is called the rank of the vertex. The minimum number of ranks needed for a vertex-ranking of G is called the vertex-ranking number of G . A vertex-ranking of G using the minimum number of ranks is called an *optimal-vertex ranking* of G . The *vertex-ranking* problem is to find an optimal vertex-ranking of a given graph. The constraints for the vertex-ranking problem imply that two adjacent vertices cannot have the same rank. Thus the vertex-ranking problem is a restriction of the vertex-coloring problem. Figure 1.6 depicts an optimal vertex-ranking of a graph G using four ranks, where ranks are drawn next to the vertices.

The vertex-ranking problem has received much attention because of the growing number of applications. The problem of finding an optimal vertex-ranking of a graph G is equivalent to the problem of finding a minimum-height vertex-separator tree of G . The vertex-ranking problem plays an important role for the parallel Cholesky factorization of matrices [7, 20]. Yet other applications of the vertex-ranking problem lie in the field of VLSI-layout [11, 19, 24].

We then review the results on the vertex-ranking problem. The vertex-ranking problem was posed in 1988 by Iyer *et al.* in relation with the applications in VLSI layout and in manufacturing systems [11]. Pothen proved that the vertex-ranking problem is NP-hard in general [2, 22], and hence it is very unlikely that there is a polynomial-time algorithm for solving the problem for general graphs [1]. Hence an approximation algorithm would be useful. An approximation algorithm for graphs in general was given by Bodlaender *et al.*, whose approximation ratio is $O(\log^2 n)$ for the vertex-ranking number [3], where n is the

number of vertices of the input tree. Although the vertex-ranking problem is NP-hard, Iyer *et al.* presented an $O(n \log n)$ time sequential algorithm to solve the vertex-ranking problem for trees [11]. Then Schaffer obtained a linear-time sequential algorithm by refining their algorithm and its analysis [23]. Recently Deogun *et al.* gave sequential algorithms to solve the vertex-ranking problem for interval graphs in $O(n^3)$ time for permutation graphs in $O(n^6)$ time [6]. Bodlaender *et al.* presented a polynomial-time sequential algorithm to solve the vertex-ranking problem for partial k -trees, that is, graphs of treewidth bounded a fixed integer k [2]. Kashem *et al.* presented a polynomial time algorithm to solve the c -vertex-ranking problem for partial k -trees, that is, graphs of treewidth bounded a fixed integer k [15]. Very recently Kloks *et al.* have presented a sequential algorithm for computing the vertex-ranking number of an asteroidal triple-free graph in time polynomial in the number of vertices and the number of minimal separators [16]. On the other hand de la Torre *et al.* presented a parallel algorithm to solve the vertex-ranking problem for trees in $O(\log n)$ time using $O(n^2 / \log n)$ processors on the CREW PRAM [4].

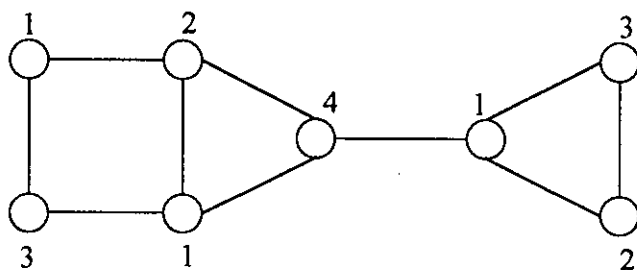


Figure 1.6: An optimal vertex-ranking of a graph G .

1.4 Summary

This thesis gives an important proof of NP-completeness of edge-ranking problem for series-parallel graphs. In this section we summarize our main result. The following table shows the known result about the edge-ranking problem.

Table 1.1 shows that there exists a linear-time algorithm for solving the edge-ranking problem on trees. The complexity of an algorithm for solving the edge-ranking problem on partial k -trees with constant maximum degree is polynomial. The complexity of algorithm of edge-ranking problem on series-parallel ($k = 2$) graphs with constant maximum degree is also polynomial. But there exists no polynomial time algorithm for solving the edge-ranking problem of general series-parallel graphs.

Class of Graph	Time	Reference
Tree	$O(n)$	Lam and Yue [18]
Partial k -tree with constant max. degree	$O(n^{12\Delta(k+1)+2})$	Kashem <i>et al.</i> [14]
Series-parallel graph with constant max. degree	$O(n^{36\Delta+2})$	Kashem <i>et al.</i> [14]
Series-parallel graphs	NP-Complete	Ours

Table 1.1: Some known results related to edge-ranking problem.

This thesis is organized as follows. In Chapter 2 we give preliminary definitions. We prove the main theorem in Chapter 3 with the help of some lemmas that are also discussed in Chapter 3. Finally, Chapter 4 concludes with discussion of the result and future works.

Chapter 2

Preliminary

In this chapter we present some basic terms and necessary observations. Definitions that are not included in this chapter will be introduced, as they are needed. We start, in Section 2.1 by giving the definition of graph and multigraph. We also define directed and undirected graph and other graph theoretic terms in this section. The thesis deals with series-parallel graphs, which are special kinds of graphs. We define the series-parallel graph in Section 2.3. Then we define some terminologies related to our thesis in Section 2.4. At last in Section 2.5 we describe the theory of NP-completeness.

2.1 Graphs and Multigraphs

A *graph* 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 distinct vertices (see Figure 2.1). We call $V(G)$ the *vertex set* of the graph G , and $E(G)$ the *edge set* of G . Through this thesis the number of

vertices of G is denoted by n that is $n = |V|$. If $e = (v, w)$ is an edge, then e is said to join the vertices v and w , and these vertices are then said to be *adjacent*. In this case we also say that w is *neighbor* of v and that e is incident to v and w . If a graph G has no "multiple edges" or "loops", then G is said to be a *simple graph*. *Multiple edges* join the same pair of vertices, while a *loop* joins a vertex to itself. The graph in which loops and multiple edges are allowed is called *multigraph*. Sometimes a simple graph is simply called a *graph* only if there is no danger of confusion.

2.1.1 Directed Graph

A *directed graph* (or *digraph*) G is a pair (V, E) , where V is a finite set and E is a binary relation on V . The set V is called the vertex set of G , and its elements are called *vertices*. The set E is called the edge set of G , and its elements are called *edges*. Figure 2.1 (a) is a pictorial representation of a directed graph on the vertex set $\{1, 2, 3, 4, 5, 6\}$. Vertices are represented

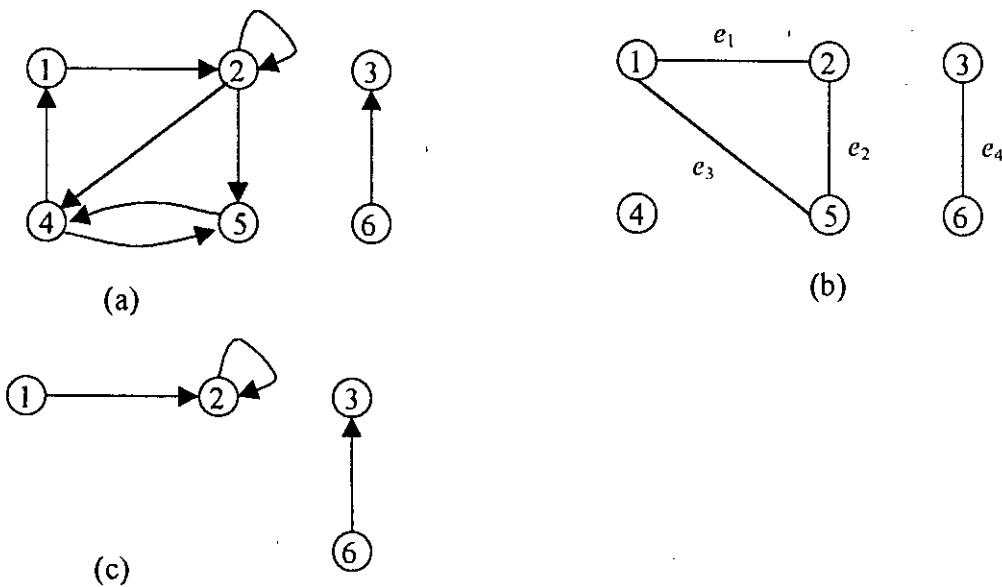


Figure 2.1: Directed and undirected graphs. (a) A directed graph, (b) An undirected graph, (c) An induced subgraph.

by circles in the figure, and edges are represented by arrows. Note that *self-loops* are edges from a vertex to itself—are possible.

2.1.2 Undirected Graph

In an *undirected* graph $G = (V, E)$, the edge set E consists of unordered pairs of vertices, rather than ordered pairs. That is, an edge is a set $\{u, v\}$, where $u, v \in V$ and $u \neq v$. By convention, we use the notation (u, v) for an edge, rather than the set notation $\{u, v\}$, and (u, v) and (v, u) are considered to be the same edge. In an undirected graph, self-loops are forbidden, and so every edge consists of exactly two distinct vertices. Figure 2.1 (b) is a pictorial representation of an undirected graph on the vertex set $\{1, 2, 3, 4, 5, 6\}$.

The *degree* of a vertex in an undirected graph is the number of edges incident on it. For example, vertex 2 in Figure 2.1(b) has degree 2.

2.1.3 Paths and Cycles

A $v_0 - v_l$ *walk* in G is an alternating sequence of vertices and edges of G , $v_0, e_1, v_1, \dots, v_{(l-1)}, e_l, v_l$ beginning and ending with a vertex, in which each edge incident to two vertices immediately preceding and following it. If the vertices v_0, v_1, \dots, v_l are distinct (except, possibly v_0, v_l), then the walk is called a *path* and is usually denoted by $v_0, v_1 \dots v_l$. The *length* of the path is l , one less than the number of vertices on the path. A path is *closed* if $v_0 = v_l$. A closed path containing at least one edge is called a *cycle*. A cycle of length 3, 4, 5, ..., is called a triangle, quadrilateral, pentagon, etc. One example of a walk in G depicted in Fig 2.1(b) is v_1, e_1, v_2, e_2, v_3 which is not closed. One example of cycle is $v_1, e_1, v_2, e_2, v_3, e_3, v_1$ a triangle.

2.1.4 Connected Graphs

An undirected graph is *connected* if every pair of vertices is connected by a path. The *connected components* of a graph are the equivalence classes of vertices under the "is

reachable from" relation. The graph in Figure 2.1(b) has three connected components: $\{1, 2, 5\}$, $\{3, 6\}$, and $\{4\}$. Every vertex in $\{1, 2, 5\}$ is reachable from every other vertex in $\{1, 2, 5\}$. An undirected graph is connected if it has exactly one connected component, that is, if every vertex is reachable from every other vertex.

2.1.5 Subgraph

We say that a graph $G' = (V', E')$ is a *subgraph* of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. Given a set $V' \subseteq V$, the subgraph of G *induced* by V' is the graph $G' = (V', E')$, where $E' = \{(u, v) \in E : u, v \in V'\}$. The subgraph induced by the vertex set $\{1, 2, 3, 6\}$ in Figure 2.1(a) appears in Figure 2.1(c) and the edge set $\{(1,2), (2, 2), (6, 3)\}$

2.2 Tree

A (*free*) *tree* is a connected graph without any cycle. We often omit the adjective "free" when we say that a graph is a tree. Figure 2.2 is an example of a tree. The vertices in a tree are usually called *nodes*. A *rooted tree* is a free 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 Fig. 2.2, the root is v_1 . Every node u other than the root is connected by an edge to some one 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 Fig. 2.2, 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* or *terminal* is node of a tree that has no children. Thus every node of a tree is either a leaf or an internal node, but not both. In Fig. 2.2, the leaves are $v_5, v_6, v_3,$ and v_7 , and the nodes $v_1, v_2,$ and v_4 are internal nodes or *nonterminals*.

The parent-child relationship can be extended naturally to ancestors and descendants. Suppose u_1, u_2, \dots, u_l is a sequence of nodes in a tree such that u_1 is the parent of u_2 , which is a parent of u_3 , and so on. Then node u_1 is called an *ancestor* of u_l and node u_l is called the *descendant* of u_1 . The root is the ancestor of every node in a tree and every node is a descendant of the root. In Fig. 2.2 all nodes other than v_1 are descendants of v_1 , and v_1 is an ancestor of all other nodes.

In a tree T , a node u together with all of its descendants, if any, is called a *subtree* of T . Node u is the root of this subtree. Referring again to Fig. 2.2, node v_3 by itself is a subtree, since v_3 has no descendants other than itself. As another example, nodes v_2, v_5 and v_6 form a subtree, with root v_2 . Finally, the entire tree of Fig. 2.2 is a subtree of itself, with root v_1 . The maximal subtree of T rooted at a vertex $u \in V$ is denoted by $T(u)$. Let $e = (u, v)$ be an edge in

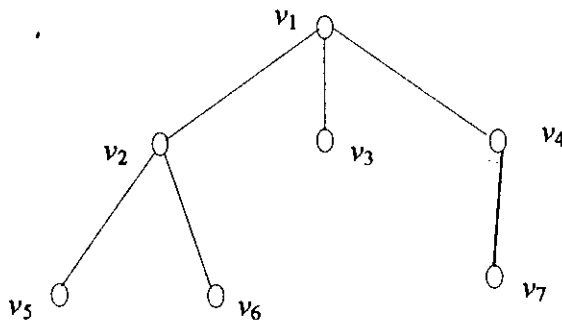


Figure 2.2: A tree with seven vertices.

T such that u is a child of v . Then the tree obtained from $T(u)$ by adding e is denoted by $T(e)$. We denote by $T - T(e)$ the tree obtain from T by deleting all edges and all vertices of $T(e)$ except v .

The *height* of a node u in a tree is the length of the longest path from u to a leaf. The *height* of a tree is the height of the root. The *depth* of a node u in a tree is the length of a path from the root to u . The *level* of a node u in a tree is the height of the tree minus the depth of

u . In Fig.2.2, for example, node v_3 is of height 0, depth 1 and level 1. The tree in Fig. 2.2 has height 2.

2.2.1 Binary Tree

A binary tree can be defined recursively as follows:

A *binary tree* T is a structure defined on a finite set of nodes that either

1. Contains no nodes, or
2. Is comprised of three disjoint sets of nodes: a *root* node, a binary tree called its left *subtree*, and a binary tree called its right *subtree*.

The Figure 2.2 is a binary tree.

2.3 Series-Parallel Graph

A *series-parallel* graph is defined recursively as follows.

1. A graph G of a single edge is a series-parallel graph. The end points v_s and v_t of the edge are called the *terminals* of G and is denoted by $v_s(G)$ and $v_t(G)$.
2. Let G_1 be a series-parallel graph with terminals $v_s(G_1)$ and $v_t(G_1)$, and let G_2 be another series-parallel graph with terminals $v_s(G_2)$ and $v_t(G_2)$.
 - a) A graph G obtained from G_1 and G_2 by identifying vertex $v_t(G_1)$ with vertex $v_s(G_2)$ is a series-parallel graph whose terminals are $v_s(G) = v_s(G_1)$ and $v_t(G) = v_t(G_2)$. Such a connection is called a *series connection*, and G is denoted by $G = G_1 \cdot G_2$. (See Figure 2.3).
 - b) A graph G obtained from G_1 and G_2 by identifying $v_s(G_1)$ with $v_s(G_2)$ and $v_t(G_1)$ with $v_t(G_2)$ is a series-parallel graph whose terminals are $v_s(G) = v_s(G_1) = v_s(G_2)$ and $v_t(G) = v_t(G_1) = v_t(G_2)$. Such a connection is called a *parallel connection*, and G is denoted by $G = G_1 || G_2$. (See Figure 2.3).

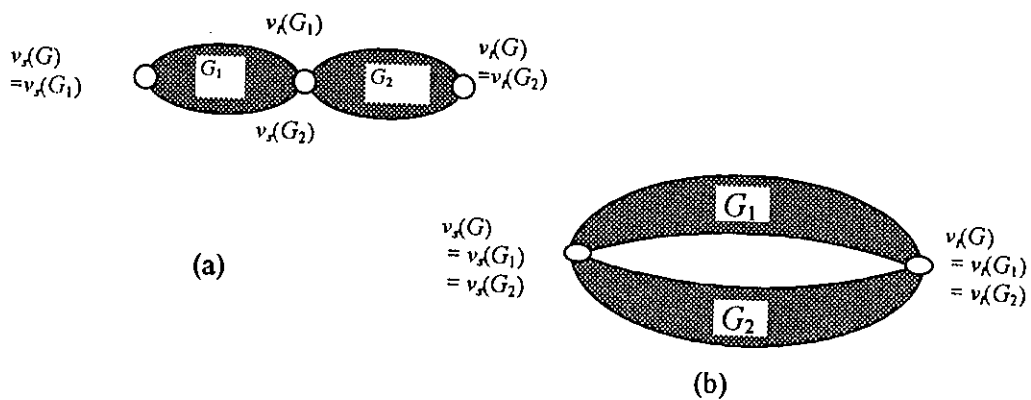


Figure 2.3: Series (a) and parallel (b) connections.

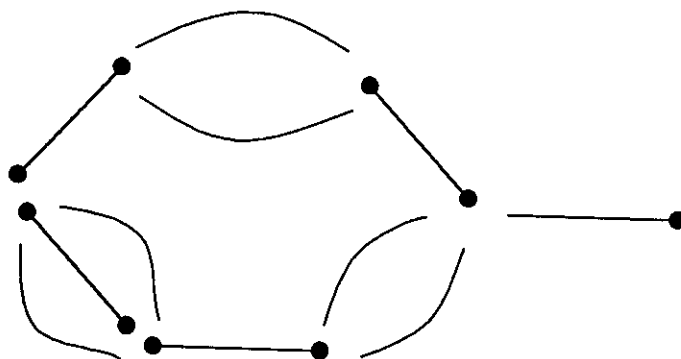


Figure 2.4: A series-parallel graph with detail connection.

After connecting them we get the final series-parallel graph G is shown bellow. The graph G can be ranked optimally using six ranks.

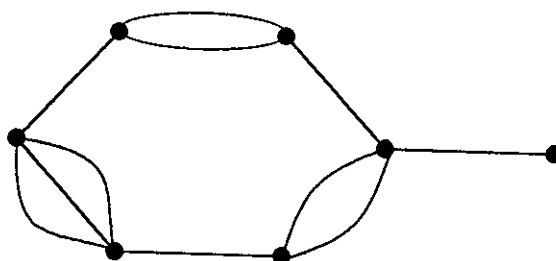


Figure 2.5: A series-parallel graph.

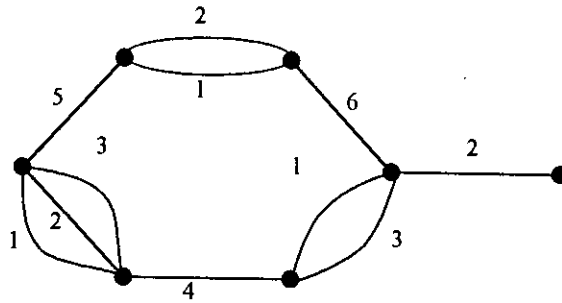


Figure 2.6: An edge-ranking of the series-parallel graph.

2.4 Edge-Ranking

The symbol ψ is used to denote an edge-ranking of a graph. We denote $rank(\psi)$ to be the number of distinct labels used by an edge-ranking ψ , and $rank(G)$ to be the number of distinct labels used by an optimal ranking of G .

2.4.1 Minimal Cut and Primitive Separator

Let $G = (V, E)$ be a series-parallel graph. For any $C \subseteq E$, C is an *edge cut* of G if the removal of C from G disconnects G . An edge cut C of G is said to be *minimal* if the removal of any subset C' of C does not disconnect G . For any minimal cut C of a graph G , the removal of C disconnects G into exactly two connected components.

Let ψ be an edge-ranking of G . Consider the process of removing edges from G in the decreasing order of rank. It should be noted that the edge with highest rank is unique in the graph. If we delete the edge with the highest rank, the resultant graph may be connected or disconnected. If the graph is connected the second highest label in the remaining graph is also unique. Let l be the label of the last edge deleted from G to disconnect the graph. We define the *primitive separator* of ψ to be the set of edges that have labels $\geq l$ under ψ . The

removal of primitive separator from G disconnects it into exactly two components. The remaining edges of the graph other than the primitive separator have labels less than l .

Fact: Let ψ be an optimal edge-ranking of a graph G . Let S be the primitive separator of ψ , and let G_1 and G_2 be the connected components after removing S from G . Then both G_1 and G_2 can be ranked using at most $\text{rank}(G) - |S|$ distinct labels.

We say that ψ satisfies the *minimal cut property* if its primitive separator S forms a minimal cut C in G and the restrictions of ψ to the two connected components resulted from the removal of C from G also satisfy the minimal cut property. We then have the following lemma [17].

Lemma 2.1 *Any graph G has an optimal edge-ranking satisfying the minimal cut property.*

Proof: Given an edge-ranking ψ of G we can rearrange the labels of edges such that the resultant arrangement of ranking satisfies the minimal cut property.

Let S be the primitive separator of ψ and $S' \subseteq S$ be the minimal cut of G . We can rearrange the labels on S such that S' receives the biggest labels. Note that such rearrangement still satisfies the property of ranking. Removing S' from G disconnects it and separates into two connected components G'_1 and G'_2 . If G'_1 and G'_2 consist of one or more edges, we can again and again rearrange the labels on their edges recursively. Let ψ' be the resultant ranking of G . As the process above never introduces new labels, $\text{rank}(\psi') = \text{rank}(\psi)$. If ψ is an optimal then ψ' is optimal too. \square

The following Figure 2.7 shows an example of two different edge-ranking of a graph, one (a) satisfies the minimal cut properties whose primitive separator (of two edges) S forms a minimal cut and the other (b) does not satisfies, whose primitive separator (of three edges) S does not forms a minimal cut.

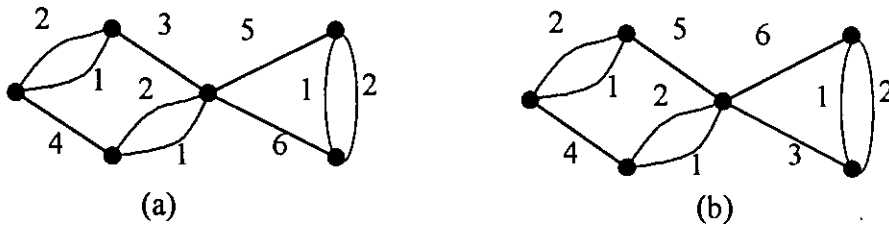


Figure 2.7 The edge-ranking in (a) satisfies the minimal cut property, but (b) does not satisfy the minimal cut property.

2.4.2 Terminal Edge and Internal Edge

In a series-parallel graph G , if any end of any edge has the degree one, then the edge is a *terminal edge*. On the other hand, an edge with both ends having degree more than one is an *internal edge*. The following lemma [17] shows that the existence of an optimal ranking does not include terminal edge in its primitive separator.

Lemma 2.2 *For any graph G containing at least one internal edge, G has an optimal edge-ranking ψ satisfying the minimal cut property such that the primitive separator of ψ contains no terminal edge.*

Proof: Let ψ be an optimal edge-ranking of G satisfying the minimal cut property. Suppose the primitive separator S of ψ contains a terminal edge \hat{e} . Since the removal of \hat{e} disconnects its unit degree endpoint from G , S contains \hat{e} as its only edge and \hat{e} gets the biggest label in ψ . In this case we can construct another optimal edge-ranking ψ' for G from ψ such that \hat{e} gets a label 1 as follows:

$$\psi'(e) = \begin{cases} 1 & \text{if } e = \hat{e} \\ \psi(e) + 1 & \text{otherwise} \end{cases}$$

ψ' uses the same number of distinct label as ψ and \hat{e} no longer lies in the primitive separator. Also, ψ' satisfy the minimal cut property. If the primitive separator of ψ' does not contains any terminal edge then we are done. Otherwise we can repeat the process until we get an optimal edge-ranking for G with an internal edge getting the maximal label. \square

2.4.3 Edge Multiplicity and Normal Form

A *multigraph* is a graph in which a pair of vertices can be connected by one or more parallel edges. Given a multigraph G , the *edge multiplicity* of an edge $e = (u, v)$ is the number of parallel edges connecting u and v in G . If the edge multiplicity of all edges in G is r , then we say that the edge multiplicity of G is r . An optimal edge-ranking ψ is said to be in *normal form* if it satisfies the minimal cut property and its primitive separator contains no terminal edge.

2.5 The Theory of NP-Completeness

The theory of NP-completeness which we present here does not provide a method of obtaining polynomial time algorithms for problems. Nor does it say that algorithms of this complexity do not exist. Instead, what we shall do is to show that many of the problems for which there is no known polynomial time algorithm are computationally related.

2.5.1 Polynomial Time Algorithms and Intractable Problems

Let us say that a function $f(n)$ is $O(g(n))$ whenever there exists a constant c such that $|f(n)| \leq c \cdot |g(n)|$ for all $n \geq 0$. A *polynomial time algorithm* is defined to be one whose time complexity function is $O(p(n))$ for some polynomial function p , where n is used to denote the

input length. Any algorithm whose time complexity function cannot be so bounded is called *exponential time algorithm*.

There is wide agreement that a problem has not been "well-solved" until a polynomial time algorithm is known for it. A problem is *intractable* if it is not well-solved that is so hard that no polynomial time algorithm is not known for the problem.

2.5.2 Decision Problems

As a matter of convenience, the theory of NP-completeness is designed to be applied only to *decision problems*. Such problems have only two possible solutions, either the answer is "yes" or the answer "no". As for example the MINIMUM GRAPH BISECTION problem is a decision problem defined as follows: Given an undirected graph F with $2n$ vertices and a positive integers c , does there exist a partition $V(F) = U \cup W$ with $|U| = |W| = n$ and $|E_F(U, W)| \leq c$? This problem was shown to be NP-complete by Garey, Johnson and Stockmeyer [9].

2.5.3 Deterministic and Nondeterministic Algorithms

The algorithms so far we know have the property that the result of every operation is uniquely defined. Algorithms with this property are termed *deterministic algorithms*. Such algorithms agree with the way programs are executed on a computer. In a theoretical framework we can remove this restriction on the outcome of every operation. We can allow algorithms to contain operations whose outcomes are not uniquely defined but are limited to a specified set of possibilities. The machine executing such operations is allowed to choose any one of these outcomes subject to a termination condition. This leads to the concepts of a *nondeterministic algorithm*. The computer so far been invented cannot execute nondeterministic algorithm.

Example: Consider the problem of searching for an element x in a given set of elements $A(1 : n)$ $n \geq 1$. We are required to determine an index j such that $A(j) = x$ or $j = 0$ if x is not in A . A nondeterministic algorithm for this is

```

j = choice(1 : n)
if A(j) = x then print(j); success endif
printf("0"); failure

```

The above algorithm is a nondeterministic polynomial time algorithm and it is not possible to implement by a computer. The reason is that, it is not possible to determine the existence of a number in an array by a single operation.

2.5.4 The Class P

The class P is the type of problems that can be solved by polynomial time algorithm. For the problem of class P, polynomial time algorithm already exists. For example matrix multiplication algorithm, Prim's minimum spanning tree algorithm, graph traversal algorithm etc. are polynomial time algorithms.

2.5.5 The Class NP

The name NP stands for nondeterministic polynomial. The class NP is the set of problems that can be solved by nondeterministic algorithm in polynomial time or the set of problems whose solution can be verified by a polynomial time algorithm. No deterministic polynomial time algorithm exists for the problems of NP class.

2.5.6 Reducibility

Let L_1 and L_2 be two problems. L_1 reduces to L_2 (also written $L_1 \leq_p L_2$) if and only if there is a way to solve L_1 by a deterministic polynomial time using a deterministic algorithm that solves L_2 in polynomial time.

2.5.7 NP-Completeness

Polynomial-time reductions provide a formal means for showing that one problem is at least as hard as another, to within a polynomial-time factor. That is, if $L_1 \leq_p L_2$, then L_1 is not more than a polynomial factor harder than L_2 , which is why the "less than or equal to" notation for reduction is mnemonic. We can now define the set of NP-complete problems, which are the hardest problems in NP. A problem L is NP-complete if

1. $L \in \text{NP}$, and
2. $L^1 \leq_p L$ for $L^1 \in \text{NPC}$.

That is, in word we can say

A problem is NP-complete if and only if

1. The problem is in NP, and
2. The problem is polynomially reducible from another problem that is already in NP-complete.

If a problem L satisfies property 2, but not necessarily property 1, then we say that L is NP-hard.

2.5.8 NP-Completeness Proof Technique

A salesman wants to visit n cities starting form a city. Visiting each city exactly once he wants to return the starting city so that the total cost of the tour is minimum over all possible tours. This is a *traveling salesman problem* (TSP).

The TSP can be also be defined as a decision problem as follows: Given a graph $G = (V, E)$ that indicate the network of the city (where V indicates cities and E indicates the connection of i to j city). The function $c(i, j)$ indicate the cost from city i to city j , M is a positive number. Does there exist a tour such that total cost of the tour $\leq M$?

Theorem: Traveling salesman problem (TSP) is NP-complete.

Proof: We first show that TSP belongs to NP. For an instance of the problem, we show that we can verify the solution of TSP problem in polynomial time. Given a solution sequence of a TSP problem, the verification algorithm checks that this sequence contains each vertex exactly once, sums up the edge costs, and checks whether the sum is at most M . This process can certainly be done in polynomial time.

To prove TSP is NP-hard, we show that HAM-CYCLE \leq_p TSP (Hamiltonian cycle problem (HAM-CYCLE) already proved as a NP-complete problem). Let $G = (V, E)$ be an instance of HAM-CYCLE. We construct an instance of TSP as follows. We form the complete graph $G' = (V, E')$, where $E' = \{(i, j) : i, j \in V\}$, and we defined the cost function c by

$$c(i, j) = \begin{cases} 0 & \text{if } (i, j) \in E, \\ 1 & \text{if } (i, j) \notin E. \end{cases}$$

The instance of TSP is then $(G', c, 0)$, which is easily formed in polynomial time. So TSP is polynomially reducible from HAM-CYCLE.

We now show that graph G has a hamiltonian cycle if and only if graph G' has a tour of cost at most 0. Suppose that graph G has a hamiltonian cycle h . Each edge in h belong to E and thus cost 0 in G' . Thus, h is a tour in G' with cost 0. Conversely, suppose that graph G' has a tour h' of cost at most 0. Since the costs of the edges in E' are 0 and 1, the cost of the tour h' is exactly 0. Therefore, h' contains only edges in E . We conclude that h is a hamiltonian cycle in graph G .

2.5.9 Relation among P, NP and NPC

Most theoretical computer scientists believe that $P \neq NP$ which leads to the relation among P, NP and NPC shown in the figure. Most scientists also believe NP-complete problems are intractable. The reason is that if any single NP-complete problem can be solved in polynomial time, then every NP-complete has a polynomial time algorithm.

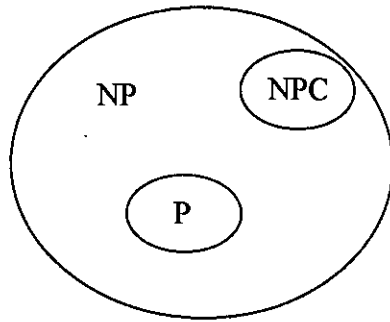


Figure 2.8 How most theoretical computer scientist view the relationships among P, NP, and NPC. Both P and NPC are wholly contained within NP, and $P \cap NPC = \emptyset$.

2.6 Conclusion

In this chapter we present some terminologies like graphs, multigraphs, trees, series-parallel graphs, edge-ranking etc. that are related to our thesis. We introduce some preliminary lemmas in this chapter. At last we present some terminologies related to the NP-Complete theory.

Chapter 3

The Main Theorem

This chapter contains the main theory related to prove the NP-completeness of the edge-ranking problem for series-parallel graphs. In Section 3.2 we define the MINIMUM GRAPH BISECTION problem that is already proved as an NP-complete problem. We define EDGE-RANKING problem as a decision problem in Section 3.4. In Section 3.5 we describe the construction of our graph. Finally we prove that the edge-ranking problem as an NP-complete problem.

3.1 Introduction

This thesis deals with undirected graphs without self-loops. For two disjoint subsets U and W of $V(G)$, we denoted by $E_G(U, W)$ the set of edges e in G such that one end of e is in U and other end is in W . Our main result of this thesis is the following theorem.

Theorem 3.1: *The edge-ranking problem is NP-Complete for series-parallel graphs.*

In the remainder of this chapter we will give a proof of Theorem 3.1.

3.2 MINIMUM GRAPH BISECTION Problem

The MINIMUM GRAPH BISECTION problem is defined as follows: Given an undirected graph F with $2n$ vertices and a positive integers c , does there exist a partition $V(F) = U \cup W$ with $|U| = |W| = n$ and $|E_F(U, W)| \leq c$?

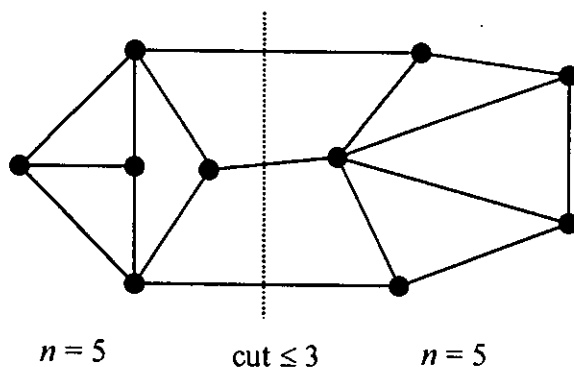


Figure 3.1 An example of MINIMUM GRAPH BISECTION problem.

The above example is an instance of MINIMUM GRAPH BISECTION problem. For $c = 3$ the example is a yes instance of the problem, where $n = 5$. This problem was shown to be NP-complete by Garey, Johnson and Stockmeyer [9].

3.3 The Edge-Ranking Problem is in NP

For a given edge-ranking of a graph, we can verify the correctness of the ranking in polynomial time as follows: We delete all edge of $rank > 1$, if we get at most one edge of

rank 1 in each connected component then we say the graph is valid ranking of rank 1. Similarly, we delete all edge of $rank > 2$, if we get at most one edge of rank 2 in each connected component then we say the graph is valid ranking of rank 2. Similarly we check for all rank ψ of the graph. If all are valid then we say the graph is valid edge-ranking. It can be done in polynomial time, so the edge-ranking problem is in NP.

Therefore, it is sufficient to show that the MINIMUM GRAPH BISECTION problem can be transformed in polynomial time to the edge-ranking problem for series-parallel graphs.

3.4 The EDGE-RANKING Problem

The EDGE-RANKING problem can also be defined as a decision problem as follows: Given a graph G and a positive integer l , does there exist an edge-ranking ψ of G such that $rank(\psi) \leq l$? Given a graph F with $2n$ vertices and a positive integer c (i.e., an instance of MINIMUM GRAPH BISECTION problem), we construct an instance of the edge-ranking problem on a series-parallel graph G . We shall prove that the series-parallel graph G will have a natural upper bound on its ranks if and only if the instance (F, c) of MINIMUM GRAPH BISECTION is a yes instance, i.e., F has a bisection with at most c edges.

3.5 Graph Construction

In this section we describe the constructions of several graph. To describe our construction, we describe the composition of several graphs. The most interesting property of such graphs is that if the resultant graph can be ranked tightly (meeting its lower bound), the individual constituent graphs can also be ranked tightly.

Let G_1 and G_2 be two connected multigraphs. We construct a series-parallel graph G by connecting the graphs G_1 and G_2 with another graph H , where $V(H) = U_1 \cup U_2$ for some $U_1 \subseteq V(G_1)$ and $U_2 \subseteq V(G_2)$, and $E(H) \subseteq (U_1 \times U_2)$. For any $C \subseteq E(H)$, C is said to be a *total cut* of H if the removal of C from H disconnects all the vertices in U_1 from all the vertices in U_2 . Let f_H denote the size of the smallest total cut of H . Then clearly $f_H = |E(H)|$.

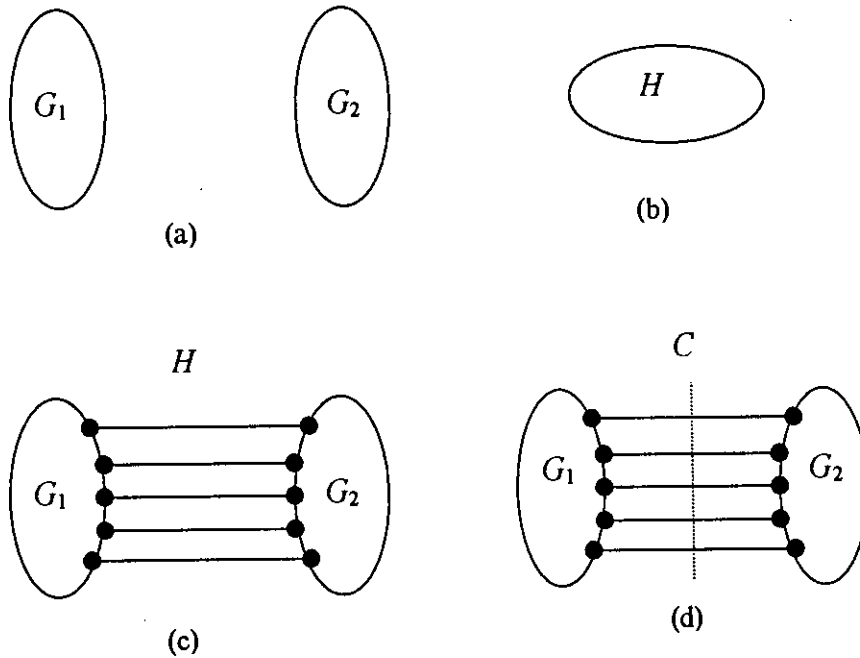


Figure 3.2: (a) the graph G_1 and G_2 and (b) the connector graph H (c) joining G_1 and G_2 with H (d) illustrates total cut of H .

The graphs G_1 , G_2 and H are chosen such a way that a lower bound on the ranks of G_1 and G_2 , and their edge multiplicities are big enough to exceed the value of f_H . Lemma 3.2 shows that with such assumptions the graph G also has a non-trivial lower bound on its ranks.

First we show how to construct the multigraphs G_1 and G_2 . Consider a binary tree T with c leaves and height $\lceil \log c \rceil$ as shown in Figure 3.3. The binary tree T is constructed such a way that the left subtrees are always complete binary trees. Let α be the number of ranks used by an optimal edge-ranking of T , i.e., $rank(T) = \alpha$

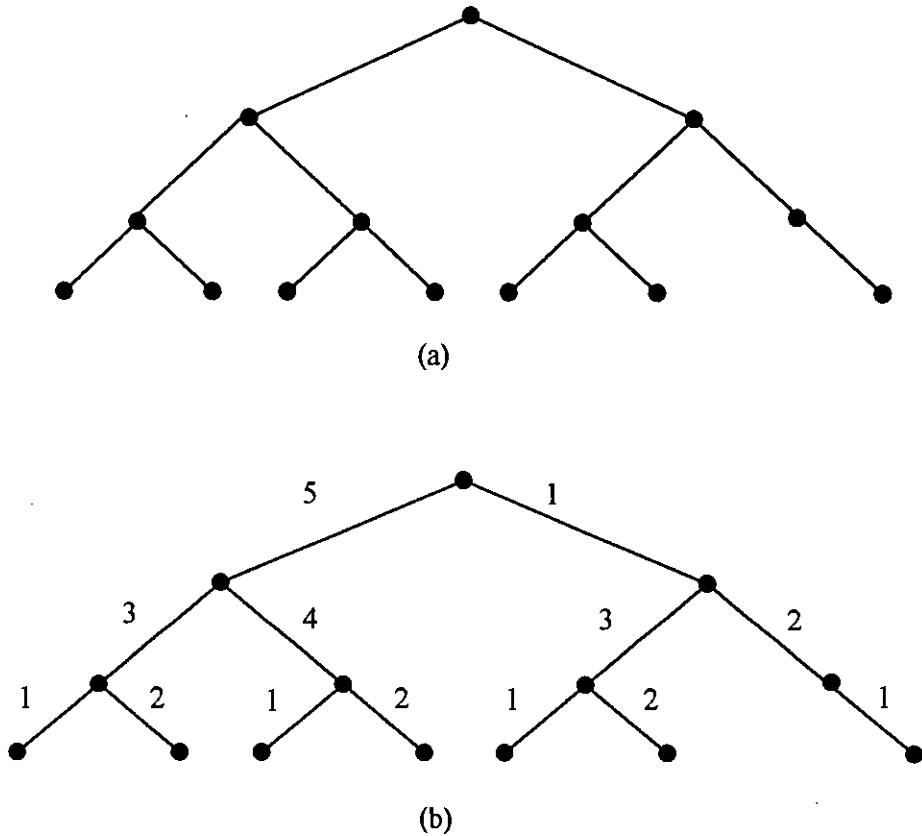


Figure 3.3: (a) A binary tree T with $c = 7$ leaves and height $\lceil \log c \rceil = 3$, and (b) optimal edge-ranking of T using five ranks.

Let m be the minimum integer such that $mn \geq c$. If we replace each edge of the binary tree with mn edges, then we get a multigraph, and we call the multigraph as G_1 . (See Figure 3.4). It should be noted that the edge-multiplicity of all edges in G_1 is mn . Since the parallel edges between two vertices can form a path themselves, all parallel edges between two

vertices must be labeled with distinct ranks. Then the multigraph G_1 can be ranked optimally using $m\alpha$ distinct ranks. The multigraph G_2 is a copy of the multigraph G_1 .

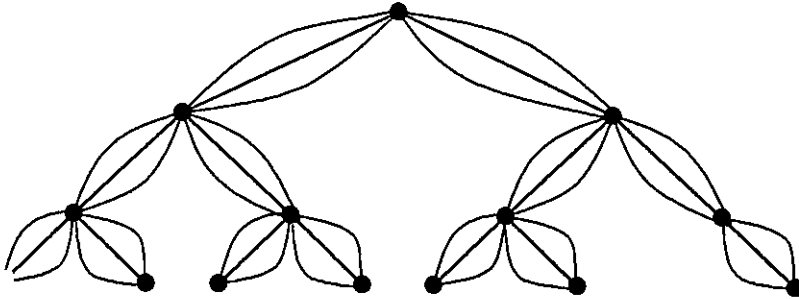


Figure 3.4: A multigraph G_1 constructed from T in Fig. 3.3.

We then construct a connector graph H to connect G_1 and G_2 for the formation of the final series-parallel graph G . The connector H is a disconnected graph of c components each of which is an edge with two vertices as shown in Figure 3.5.

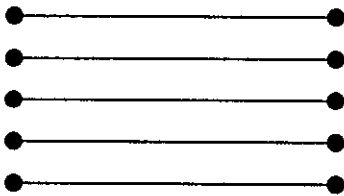


Figure 3.5: A connector graph H with $c = 5$ edges.

Finally, after connecting two multigraphs G_1 and G_2 by connector graph H we get the series-parallel graph G as shown in Figure 3.6.

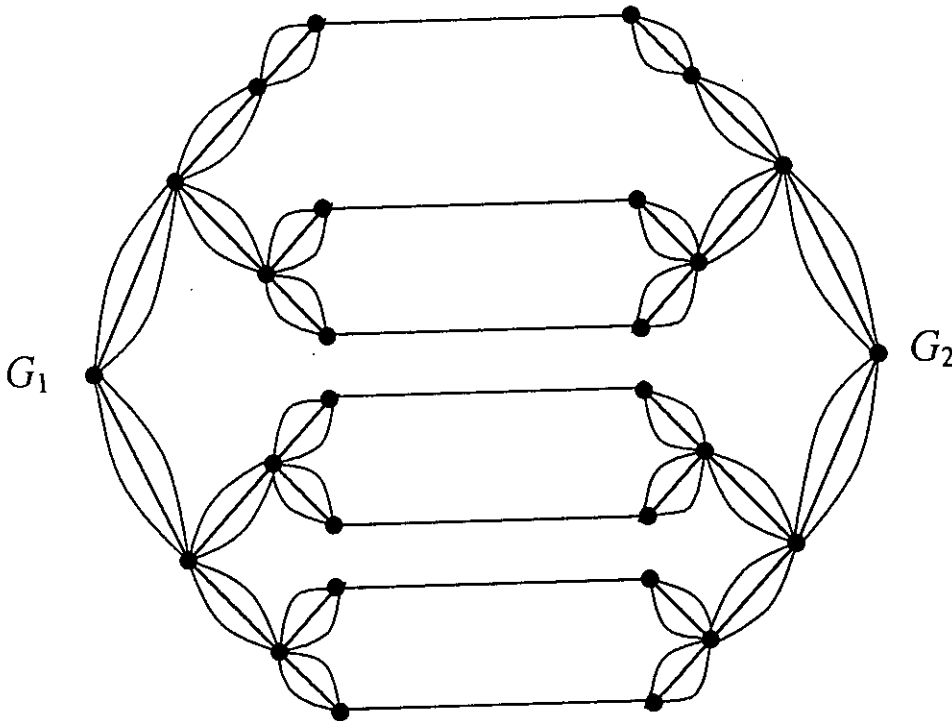


Figure 3.6: The series-parallel graph G .

The number of vertices in the series-parallel graph G is $\leq 2(c + \lceil c/2 \rceil + \lceil c/4 \rceil + \dots + 1) \leq 4c + 2\log c - 2$, and the number of edges is $\leq (4c + 2\log c - 4)mn + c$. Thus G can be constructed from F in polynomial time. We then have the following lemma.

Lemma 3.2 *Let G be a series-parallel graph formed by connecting two multigraphs G_1 and G_2 with the connector H , let $\text{rank}(G_1) = \text{rank}(G_2) = k$, and let the edge multiplicity of both G_1 and G_2 is at least f_H . Then $\text{rank}(G) = f_H + k$.*

Proof: Let ψ be an optimal edge-ranking of G in a normal form. So, its primitive separator S cannot contain any terminal edge and satisfy the minimal cut property. Since the edge multiplicity of both G_1 and G_2 is $\geq f_H$, S cannot contain an edge of G_1 or G_2 . Thus $S = E(H)$. Since there are f_H distinct paths between each edge of G_1 to each edge of G_2 and $\text{rank}(G_1) = \text{rank}(G_2) = k$, the number of ranks needed for an optimal edge-ranking of G is

$$\text{rank}(G) = |S| + k = f_H + k. \quad \square$$

We next show an useful observation when G actually been ranked using $f_H + k$ distinct labels.

Lemma 3.3 *Let G be a series-parallel graph formed by connecting two multigraphs G_1 and G_2 with the connector H , and let $\text{rank}(G) = f_H + k$. Then there exists an optimal edge-ranking ψ of G such that the primitive separator of ψ is a minimal total cut of H .*

Proof: By Lemma 2.2, there exists an optimal edge-ranking ψ of G in a normal form. Since the edge multiplicity of both G_1 and G_2 is $\geq f_H$, its primitive separator S cannot contain an edge of G_1 or G_2 . So $S = E(H)$. Since $f_H = |E(H)|$, S is the minimal total cut of H . \square

We then have the following lemma.

Lemma 3.4 *Let G be the series-parallel graph as constructed above. Then $\text{rank}(G) = k + c$ if and only if the MINIMUM GRAPH BISECTION problem has a yes instance.*

Proof: Let the MINIMUM GRAPH BISECTION problem has a yes instance. Then we show that $\text{rank}(G) = k + c$. A ranking ψ of G using $k + c$ distinct ranks are constructed as follows: the primitive separator S of ψ consists of the c edges from H . The removal of S from G disconnects it into two connected components G_1 and G_2 . Each G_1 and G_2 can be ranked

using k distinct ranks. Edge multiplicity of each graph G_1 and G_2 are $mn \geq c$. Then, by Lemma 3.2, the number of ranks needed for an optimal edge-ranking of G is

$$\text{rank}(G) = \text{rank}(G_1) + f_H = k + c.$$

Conversely, let $\text{rank}(G) = k + c$. Then we show that the MINIMUM GRAPH BISECTION problem has a yes instance. By Lemma 3.3, there exists an optimal edge-ranking ψ of G such that the primitive separator of ψ is a minimal total cut of H . Since $f_H = c$, by deleting this minimal total cut, we have a partition of $V(G)$ such that $V(G) = V(G_1) \cup V(G_2)$ with $V(G_1) = V(G_2)$ and $|E_G(V(G_1), V(G_2))| \leq f_H = c$. \square

Thus we have proved Theorem 3.1.

3.6 Conclusion

This chapter is the most important of our thesis work. First of all we show that the edge-ranking problem is in NP. Then we define the EDGE-RANKING problem as a decision problem. We construct a series-parallel graph that is reducible from MINIMUM GRAPH BISECTION problem in polynomial time. Last of all we prove the NP-completeness of edge-ranking problem for series-parallel graphs with the help of some lemmas.

Chapter 4

Conclusion

The subject of this thesis is an important class of problems, whose status is unknown. No polynomial-time algorithm has yet been discovered for an NP-complete problem, nor has yet been able to prove a superpolynomial-time lower bound for any of them. This so-called $P \neq NP$ question has been one of the deepest, most perplexing open research problems in theoretical computer science since it was posed in 1971 [10].

Most theoretical computer scientist believes that the NP-complete problems are intractable. The reason is that if any single NP-complete problem can be solved in polynomial time, then every NP-complete problem has a polynomial-time algorithm.

In this thesis, we show that the edge-ranking problem is NP-complete even for series-parallel graphs. This is a very interesting problem that is NP-complete for series-parallel graphs, but linear-time solvable for trees.

Then the c -edge-ranking problem is also NP-complete for series-parallel graphs. Many problems on series-parallel graphs exist like ordinary graph. Some of the open problems are as follows:

1. What is the complexity of approximate sequential algorithm for c -edge-ranking of series-parallel graph?
2. What is the complexity of approximate parallel algorithm for c -edge-ranking of series-parallel graph?

References

- [1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.
- [2] H. Bodlaender, J. S. Deogun, K. Jansen, T. Kloks, D. Kratsch, H. Muller, and Zs. Tuza, *Ranking of graphs*, SIAM Journal of Discrete Math, 21 (1998), pp. 168-181.
- [3] H. Bodlaender, J. R. Gilbert, H. Hafsteinsson, T. Kloks, *Approximating treewidth, path width and minimum elimination tree height*, Journal of Algorithms, 18 (1995), pp. 238-155.
- [4] P. de la Torre, R. Greenlaw, and T. M. Przytycka, *Optimal tree ranking is NP*, Parallel Processing Letters, 2 (1992), pp. 31-41.
- [5] P. de la Torre, R. Greenlaw, and A.A. Schffer, *Optimal edge-ranking of trees in polynomial time*, Algorithmica, 13 (1995) pp. 592-618.
- [6] J. S. Deogun, T. Kloks, D. Kratsch, and H. Muller, *On vertex ranking for permutation and other graphs*, Proc. Of the 11th Annual Symposium on Theoretical Aspect of computer Science, Lecture Notes in Computer Science, Springer-Verlag, 775 (1994), pp. 747-758.
- [7] J. S. Deogun and Y. Peng, *Edge ranking of trees*, Congressus Numerantium, 79 (1990) pp. 19-28.
- [8] I. S. Duff and J. K. Reid, *The multifrontal solution space symmetric linear equations*, ACM Transaction on Mathematical Software, 9(1983), pp. 302-325.
- [9] M.R. Garey, D.S. Johnson, and L. Stockmeyer, *Some simplified NP-complete graph problems*, Theoretical Computer Science, 1, 1996, pp. 237-267.

- [10] M.R. Garey, and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman Co., New York, 1979.
- [11] V. Iyer, H.D. Ratliff, and G. Vijayan, *Optimal vertex ranking of trees*, *Processing Letters*, 28(1988), pp. 225-229.
- [12] V. Iyer, H.D. Ratliff, and G. Vijayan, *Parallel assembly of modular products - an analysis*, Technical report PDRC 88-06, Georgia Institute of Technology, 1988.
- [13] V. Iyer, H.D. Ratliff, and G. Vijayan, *On an edge-ranking problem of trees and graphs*, *Discrete Applied Mathematics*, 30 (1991), pp. 43-52.
- [14] M.A. Kashem, X. Zhou, and T. Nishizeki, *Generalized edge-rankings of partial k-trees with bounded maximum degree*, *Proc. of International Conference on Computer and Information Technology*, 1998, pp. 45-51.
- [15] M.A. Kashem, X. Zhou, and T. Nishizeki, *Algorithms for generalized vertex-rankings of partial k-trees*, *Theoretical Computer Science*, 240(2000), pp. 407-427.
- [16] T. Kolks, H. Muller, and C. K. Wong, *Vertex ranking of asteroidal triple-free graphs*, *Proc. of the 77th International Symposium on Algorithms and Computation (ISAA'96)*. *Lecture notes in Computer Science*, Springer-Verlag, 1178 (1996), pp. 174-182.
- [17] T.W. Lam and F.L. Yue, *Edge-ranking of graphs is hard*, *Discrete Applied Mathematics*, 85 (1998), pp. 71-86.
- [18] T.W. Lam and F.L. Yue, *Optimal edge-ranking of trees in linear time*, *Proc. of the Ninth ACM-SIAM Symposium on Discrete Algorithms*, 1998, pp. 436-445.
- [19] C. E. Leiserson, *Area-efficient graph layouts for VLSI*, *Proc. of the 21st Annual IEEE Symposium on Foundations of Computer Science*, 1980, pp. 270-281.
- [20] J. W. H. Liu, *The role of elimination trees in space factorization*, *SIAM Journal of Matrix Analysis and Applications*, 11 (1990), pp. 134-172.
- [21] N. Megiddo, *Applying parallel computation algorithms in the design of series algorithms*, *References 29 Journal of the ACM*, 30 (1983), pp. 852-862.
- [22] A. Pothen, *The complexity of optimal elimination trees*, Technical Report CS-88-13, Pennsylvania State University, U.S.A., 1988.

- [23] A. A. Schaffer, *Optimal vertex ranking of trees in linear time*, Information Processing Letters, 33 (1989), pp. 91-99.
- [24] A. Sen, H Deng, and S. Guha, *On a graph partition problem with application to VLSI layout*, Information Processing Letters, 2 (1992), pp. 31-41.
- [25] X. Zhou, M.A. Kashem, and T. Nishizeki, *Generalized edge-rankings of trees*, IEICE Trans. on Fundamentals of Electronics, Communications and Computer Science, E81-A, 2 (1998), pp. 310-320.

Index

- Δ , 4, 11, 20, 22, 32
- adjacent, 5, 9, 13
- ancestor, 17
- c -edge-ranking, 2, 6, 7, 8
- class NP, 26
- Class P, 26
- connected component, 16, 32
- connected graph, 7, 16
- cycle, 15
- decision problem, 24
- degree, 4, 5, 11, 12, 15, 22, 23
- depth, 18
- descendant, 17
- deterministic algorithms, 25
- deterministic Algorithms, 25
- digraph, 14
- $E(G)$, 13
- edge, 1, 2, 3, 5, 6, 7, 11, 12, 13, 14, 15, 16, 17, 18, 20, 21, 22, 23, 28, 29, 30, 31, 32, 34, 35, 37, 38, 39, 40
- edge-eanking, 3, 5, 6, 20, 30, 31, 32, 39
- edge-ranking problem, 1, 2, 3, 5, 6, 7, 11, 12, 30, 32, 39, 40
- exponential, 24
- f_H , 3
- G_1 , 3
- graph, 1, 3, 5, 6, 7, 9, 10, 12, 13, 14, 15, 16, 18, 19, 20, 21, 22, 23, 25, 26, 27, 28, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40
 - directed graph, 13, 14
 - multigraph, 13
 - subgraph, 16
 - undirected graph, 15, 25
- H , 33
- height, 3, 7, 10, 17, 34
- induced by, 16
- intractable, 24, 28, 40
- leaf, 16, 17
- level, 18
- minimal cut property, 21, 23, 37
- MINIMUM GRAPH BISECTION, 25, 30, 31, 32, 38, 39
- neighbor, 13
- node, 16, 17, 18
- nonterminal, 16
- normal form, 23, 37

NP-complete, 1, 2, 3, 5, 25, 26, 27, 28, 30,
 31, 40
 NP-hard, 3, 10, 27, 28
 parallel connection, 19
 parent, 16, 17
 path, 1, 3, 9, 15, 17, 35
 polynomial, 1, 5, 10, 11, 24, 26, 27, 28,
 29, 31, 32, 37, 39, 40
 primitive separator, 21, 22, 23, 24, 37, 38
 $r'_c(G)$, 6
 rank, 5, 9, 20, 21, 22, 32, 34, 37, 38
 reducibility, 26
 root, 16, 17, 18
 series connection, 18

simple graph, 13
 terminal, 16, 22, 23, 24, 37
 tree, 3, 5, 7, 10, 11, 16, 17, 18, 26, 34, 35
 binary tree, 18
 rooted tree, 16
 subtree, 17, 18
 U_1 , 33
 U_2 , 33
 $V(G)$, 33
 vertex, 2, 5, 9, 10, 13, 14, 15, 16, 17, 18,
 28
 vertex-ranking, 9
 walk, 15

