M. Sc. Engineering Thesis

# Parallel Algorithms for Rankings of Series-Parallel Graphs

By
Masud Hasan

Submitted to
Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
M. Sc. Engineering (Computer Science and Engineering)

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

May 07, 2001

# Parallel Algorithms for Rankings of Series-Parallel Graphs

A Thesis submitted by

MASUD HASAN
Student No. 9605036p

For the partial fulfillment of the degree of
M.Sc. Engineering (Computer Science and Engineering)
Examination held on May 7, 2001

Approved as to style and contents by:

[illegible]

| | |
|---|---|
| DR. MD. ABUL KASHEM MIA<br>Assistant Professor<br>Department of Computer Science and Engineering<br>B.U.E.T., Dhaka-1000, Bangladesh | Chairman<br>and<br>Supervisor |

[illegible]

| | |
|---|---|
| DR. CHOWDHURY MOFIZUR RAHMAN<br>Associate Professor and Head<br>Department of Computer Science and Engineering<br>B.U.E.T., Dhaka-1000, Bangladesh | Member<br>(Ex-officio) |

[illegible]

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

Saidur

| | |
|---|---|
| DR. MD. SAIDUR RAHMAN<br>Assistant Professor<br>Department of Computer Science and Engineering<br>B.U.E.T., Dhaka-1000, Bangladesh | Member |

[illegible] 7.5.2001

| | |
|---|---|
| DR. A.H.M. ZAHIRUL ALAM<br>Professor<br>Department of Electrical & Electronics Engineering<br>B.U.E.T., Dhaka-1000, Bangladesh | Member<br>(External) |

# Contents

**Contents** i

**List of Figures** iv

**List of Tables** v

**Acknowledgements** vi

**Abstract** 1

**1 Introduction** 3

1.1 Background . . . . . . . . . . . . . . . . . . . . . . . . . . 3

1.2 Coloring Problems . . . . . . . . . . . . . . . . . . . . . . 4

1.3 Ranking Problems . . . . . . . . . . . . . . . . . . . . . . 4

1.3.1 Vertex-Ranking Problems . . . . . . . . . . . . . . . . 4

1.3.2 Edge-Ranking Problems . . . . . . . . . . . . . . . . . 7

1.4 Parallel Processing . . . . . . . . . . . . . . . . . . . . . . 8

1.4.1 Parallel Models . . . . . . . . . . . . . . . . . . . . . . 9

1.5 Application and Known Results . . . . . 11

1.6 Summary . . . . . 14

**2 Preliminaries** **15**

2.1 Introduction . . . . . 15

2.2 Basic Terminology . . . . . 15

2.2.1 Graphs and Multigraphs . . . . . 15

2.2.2 Degree of a Vertex . . . . . 16

2.2.3 Subgraphs . . . . . 16

2.2.4 Paths and Cycles . . . . . 17

2.3 Some Special Graphs . . . . . 18

2.3.1 Partial $k$-Trees . . . . . 18

2.3.2 Series-Parallel Graphs . . . . . 18

2.4 Visible Vertices and Visible Edges . . . . . 20

2.5 Conclusion . . . . . 22

**3 Vertex Rankings of Series-Parallel Graphs** **23**

3.1 Introduction . . . . . 23

3.2 The Upper Bound for $c$-Vertex-Ranking Number . . . . . 24

3.3 Equivalence Class . . . . . 24

3.4 Dominance Class . . . . . 27

3.5 Tree Contraction Algorithm . . . . . 28

3.5.1 Modified Shunt Operation . . . . . 31

3.6 Parallel Algorithm . . . . . 31

3.7 Conclusion . . . . . 35

**4 Edge Rankings of Series-Parallel Graphs** **36**

4.1 Introduction . . . . . 36

4.2 The Upper Bound for $c$-Edge-Ranking Number . . . . . . . . . 37

4.3 Equivalence Class and Dominance Class . . . . . . . . . . . . 37

4.4 Parallel Algorithm . . . . . . . . . . . . . . . . . . . . . . . . 38

4.5 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . 39

**5 Conclusion** **40**

**Bibliography** **42**

**Index** **45**

# List of Figures

1.1 A graph with five vertices and eight edges. . . . . . . . . . . . 4

1.2 (a) An optimal vertex-ranking of a graph $G$, (b) an optimal edge-ranking of the graph $G$. . . . . . . . . . . . . . . . . . . . . 5

1.3 Optimal 2-vertex-ranking of a graph $G$. . . . . . . . . . . . . . 6

1.4 Optimal $f$-vertex-ranking of a graph $G$. . . . . . . . . . . . . . 7

1.5 The shared memory model . . . . . . . . . . . . . . . . . . . . 9

1.6 (a) An optimal 2-edge-ranking, (b) its corresponding separator tree. . . . . . . . . . . . . . . . . . . . . . . . . . . . 13

2.1 (a) A subgraph induced by vertices, (b) a subgraph induced by edges. . . . . . . . . . . . . . . . . . . . . . . . . . . 17

2.2 A partial 3-tree. . . . . . . . . . . . . . . . . . . . . . . . . 18

2.3 (a) A series connection, (b) a parallel connection. . . . . . . . . 19

2.4 (a) A series-parallel graph, (b) its binary decomposition tree $T_b$. 20

2.5 A ranking of a tree. . . . . . . . . . . . . . . . . . . . . . . 21

3.1 Shunt operation. . . . . . . . . . . . . . . . . . . . . . . . . 29

# List of Tables

1.1 Known results and our results for $c$-vertex-ranking. . . . . . . 12

1.2 Known results and our results for $c$-edge-ranking. . . . . . . . 13

# Acknowledgements

First of all, I would like to thank my supervisor Dr. Md. Abul Kashem Mia for introducing me to this interesting and fascinating field of rankings of graphs, and for teaching me how to carry on a research work. His all out help was the impetus of this work. His profound knowledge in this filed, his keen interest and patience and the scholarly guidance and suggestions at all stages of the work have made it possible to complete this thesis. His careful reading of the manuscript led to improvements in both wording and the technical aspects of the text. I again express my heart-felt and most sincere gratitude to him for his constant supervision, valuable advice and continuous encouragement. Without his patience, concern and efforts this project would have not been possible.

I would like to thank the other members of my graduate committee, Prof. Dr. A.H.M. Zahirul Alam, Prof. Dr. M. Kaykobad, Associate Prof. Dr. Chowdhury Mofizur Rahman and Assistant Prof Dr. Md. Saidur Rahman, for their valuable suggestions.

I would like to give my special thanks to my colleagues who always gave me inspiration. I would like to express my utmost gratitude to my parents for their support, affection and inspiration.

Finally, I would like to express my special thanks to my wife for her endless mental support and inspiration to carry on my work smoothly.

# Abstract

This thesis presents parallel algorithms for generalized vertex-ranking and edge-ranking of series-parallel graphs. A generalized vertex-ranking of a graph $G$ is defined as follows: for a positive integer $c$, a $c$-vertex-ranking of a graph $G$ is a labeling of the vertices of $G$ with integers such that, for any label $i$, deletion of all the vertices with labels $> i$ leaves connected components, each having at most $c$ vertices with label $i$. Similarly a generalized edge-ranking of a graph $G$ is defined. A $c$-vertex(edge)-ranking is optimal if the number of labels used is as small as possible. The problem of finding an optimal $c$-vertex-ranking of $G$ plays an important role for the parallel Cholesky factorization of matrices. The $c$-edge ranking problem has applications in scheduling the manufacture of complex multi-part products; it is equivalent to finding a $c$-edge separator tree of $G$ having the minimum height. In this thesis we present a parallel algorithm for $c$-vertex-ranking of series-parallel graph $G$ that runs in $O(\log_2 n)$ time using $O(c^3 n^7 \log_{c+1}^{13} n)$ operations, where $n$ is the number of vertices in $G$. We also give an algorithm that runs in $O(\log_2 n \log_2 \log_2 n)$ time using $O(c^2 n^5 \log_{c+1}^{9} n)$ operations. For the $c$-edge-ranking of series-parallel graph $G$ our algorithm runs in $O(\log_2 n)$ time using $O(c^3 n^{6\Delta+1} \log_{c+1}^{7} n)$ operations, where $\Delta$ is the maximum vertex-degree of $G$. We also give an algorithm for $c$-edge-ranking of series-parallel graphs that runs in $O(\log_2 n \log_2 \log_2 n)$ time using $O(c^2 n^{4\Delta+1} \log_{c+1}^{5} n)$ operations. The parallel computation model we use is priority CRCW PRAM. Our algorithms assume that a binary decomposition tree of a series-parallel graph is given.

# Chapter 1

# Introduction

In this chapter we provide the necessary background and motivation for this study on the rankings of graphs. Starting with background in Section 1.1, we go to the coloring problems in Section 1.2, ranking problems and its variation in Section 1.3, parallel processing in Section 1.4, application and known results in Section 1.5. Finally Section 1.6 summarizes the chapter.

## 1.1 Background

A graph is a common structure encountered in nature numerous times. A graph $G = (V, E)$ with $n$ vertices and $m$ edges consists of a vertex set $V = \{v_1, v_2, ..., v_n\}$ and an edge set $E = \{e_1, e_2, ..., e_m\}$, where an edge in $E$ joins two vertices in $V$. Fig. 1.1 depicts a graph of five vertices and eight edges. For mathematical formulation of map, distance calculation among cities, layout and circuit design graphs are used extensively. Having huge computing power, recent research efforts in algorithms have encountered on designing efficient algorithms for solving combinatorial problems, particularly graph problems. Efficient algorithms have been obtained for various graph problems, such as coloring problems, planarity testing problem and maximum flow problem.

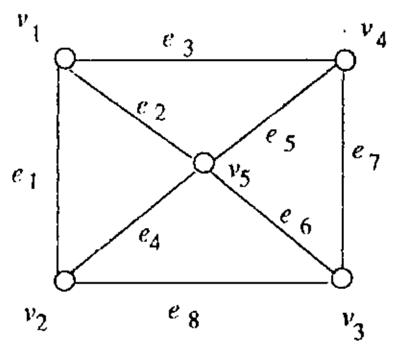


Figure 1.1: A graph with five vertices and eight edges.

## 1.2 Coloring Problems

The vertex-coloring problem and the edge-coloring problem are two of the fundamental problems on graphs. The ***vertex-coloring problem*** is to color the vertices of a given graph with the minimum number of colors so that no two adjacent vertices are assigned the same color. Fig. 1.2(a) depicts a vertex-coloring of a graph using five colors. The ***edge-coloring problem*** is to color the edges of a given graph with the minimum number of colors so that no two adjacent edges are assigned the same color. Fig. 1.2(b) depicts an edge-coloring of a graph using seven colors, where colors are shown next to the edges.

## 1.3 Ranking Problems

The vertex-ranking problem and the edge-ranking problem are restrictions of the vertex-coloring problem and edge-coloring problem, respectively.

### 1.3.1 Vertex-Ranking Problems

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

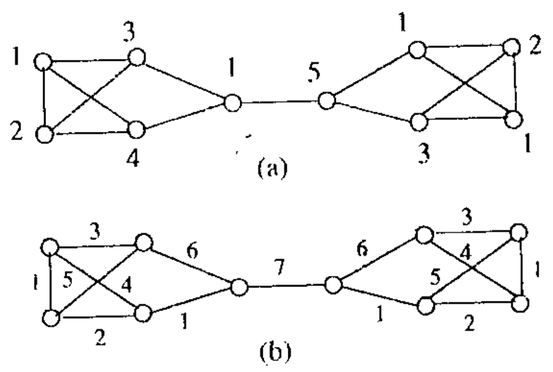


Figure 1.2: (a) An optimal vertex-ranking of a graph $G$, (b) an optimal edge-ranking of the graph $G$.

with the same label $i$ contains a vertex with label $j > i$ [IRV88]. Clearly a vertex-labeling is a 1-vertex-ranking if and only if, for any label $i$, deletion of all vertices with label $> i$ 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 $G$ with minimum number of ranks is called an *optimal vertex-ranking* of $G$. The vertex-ranking problem, also called the *ordered coloring problem* [KMS95], 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 the restriction of the vertex-coloring problem. Fig. 1.2(a) shows an optimal vertex-ranking of a graph $G$ using five ranks, where ranks are shown next to the vertices.

## Generalized Vertex Rankings

Two generalizations of the ordinary vertex-ranking have been introduced [ZNN95]. In this section we define the "$c$-vertex-ranking" of a graph, and generalize the application of ordinary vertex-ranking problem.

### $c$-Vertex-Ranking

A natural generalization of vertex-ranking is the "$c$-vertex-ranking" [ZNN95]. For a positive integer $c$, a *$c$-vertex-ranking* of a graph $G$ is a labeling of the vertices of $G$ with integers such that, for any label $i$, deletion of all the vertices with label $> i$ leaves connected components, each having at most $c$ vertices with label $i$. Clearly an ordinary vertex-ranking is a 1-vertex-ranking. The minimum number of ranks needed for a $c$-vertex-ranking of $G$ is called the *$c$-vertex-ranking number*, and is denoted by $r_c(G)$. A $c$-vertex-ranking of $G$ using $r_c(G)$ ranks is called the *optimal $c$-vertex-ranking* of $G$. A *$c$-vertex-ranking problem* is to find an optimal $c$-vertex-ranking of a given graph. Fig. 1.3 depicts an optimal 2-vertex-ranking of a graph $G$ using four ranks, where ranks are drawn next to the vertices.

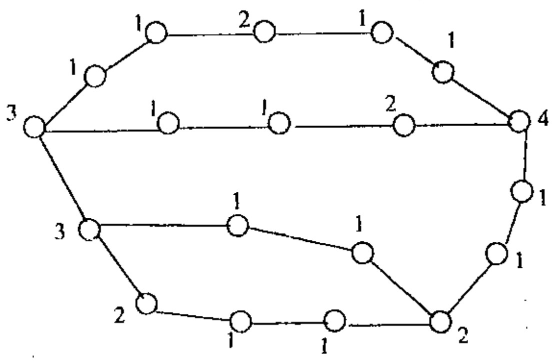
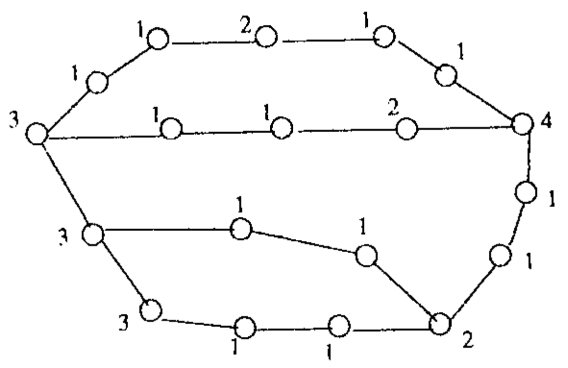


Figure 1.3: Optimal 2-vertex-ranking of a graph $G$.

### $f$-Vertex-Ranking

We may replace the positive integer $c$ by a function $f : \{1, 2, ..., n\} \to N$ to define a more generalized vertex-ranking of a graph as follows: an *$f$-vertex-ranking* of a graph $G$ is a labeling of the vertices of $G$ with integers such that, for any label $i$, deletion of all the vertices with labels $> i$ leaves connected components, each having at most $f(i)$ vertices with label $i$ [ZNN95]. Clearly a $c$-vertex-ranking is a special case of $f$-vertex-ranking where $f(i) = c$ for every rank $i$. The minimum number of ranks needed for an $f$-vertex-ranking of $G$ is called the *$f$-vertex-ranking number*, and is denoted by $r_f(G)$. An $f$-vertex-ranking of $G$ using $r_f(G)$ ranks is called the *optimal $f$-vertex-ranking* of $G$. The *$f$-vertex-ranking problem* is to find an optimal $f$-vertex-ranking of a given graph. The problem is NP-hard in general, since the $c$-vertex-ranking problem is NP-hard. Fig. 1.4 depicts an optimal $f$-vertex-ranking of a graph $G$ using four ranks, where ranks are drawn next to the vertices and the function is given in the right side of the figure.



| Ranks $i$ | $f(i)$ |
|---|---|
| 4 | 1 |
| 3 | 3 |
| 2 | 1 |
| 1 | 2 |

Figure 1.4: Optimal $f$-vertex-ranking of a graph $G$.

## 1.3.2 Edge-Ranking Problems

The edge-ranking problem is defined analogous to the vertex-ranking problem. An ordinary edge-ranking of a graph $G$ is a labeling (ranking) of the edges of $G$ with positive integers such that every path between any two edges with the same label $i$ contains an edge with label $j > i$ [IRV91]. Clearly an edge-labeling is a 1-edge-ranking if and only if, for any label $i$, deletion of all edges with labels $> i$ leaves connected components, each having at most one edge with label $i$. The integer label of an edge is called the *rank* of the edge. The minimum number of ranks needed for an edge-ranking of $G$ is called the *edge-ranking number* of $G$. An edge-ranking of $G$ with minimum number of ranks is called an *optimal edge-ranking* of $G$. The edge-ranking problem is to find an optimal edge-ranking of a given graph. The constraints for the edge-ranking problem imply that two adjacent vertices cannot have the same rank. Thus the edge ranking problem is the restriction of the edge-coloring problem. Fig. 1.2(b) shows an optimal edge-ranking of a graph $G$ in Fig. 1.2(a) using seven ranks, where ranks are shown next to the edges.

**Generalized Edge-Ranking**

The generalization of ordinary edge-ranking can be introduced analogous to the generalized vertex-ranking.

## 1.4 Parallel Processing

The main purpose of parallel processing is to perform computations faster than can be done with a single processor by using a number of processors concurrently. A parallel computer is simply a collection of processors, typically of the same type, interconnected in a certain fashion to allow the coordinate of their activities and the exchange of data. The processors are assumed to be located within a small distance of one another, and are primarily used to solve a given problem jointly. An important goal of parallel processing is to present algorithms that are suitable for implementation on parallel computers.

### 1.4.1 Parallel Models

A formal computational model suitable for sequential processing of any problem is the *random access machine (RAM)*, which assumes the presence of a certain processing unit with a random-access memory attached to it, and some way to handle the input and the output operations. Modeling parallel computation is considerably more challenging given the new dimension introduced by the presence of many interconnected processors. All the models introduces so far are based on directed acyclic graphs, shared memory, and networks. Among these the *shared memory model*, where a number of processors communicate through a common global memory, offers an attractive framework for the development of algorithmic techniques for parallel computations. The shared memory model serves our vehicle for designing and analyzing our parallel algorithms in this thesis.

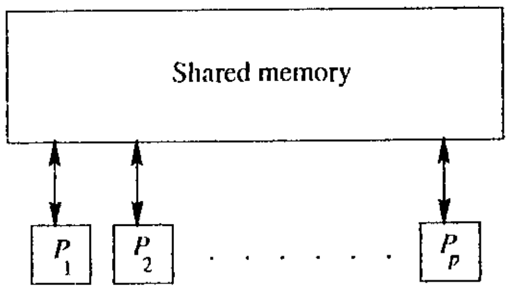


Figure 1.5: The shared memory model

#### Shared memory Model

In this model many processors have access to a single shared memory unit. More precisely, the shared-memory model consists of a number of processors, each of which has its own local memory and can execute its own local program, and all of which communicate by exchanging data through a shared memory unit. Each processor is uniquely identified by an index, called *processor number* or *processor id.* Fig. 1.5 shows a general view of a shared-memory model with $p$ processors. The processors are indexed $1, 2, ..., p$. There are two basic modes of operation of a shared-memory model. In the first mode, called *synchronous*, all the processors operate synchronously under the control of a common clock. A standard name for the synchronous shared-memory model is the *parallel random-access machine (PRAM)* model. In the second mode, called *asynchronous,* each processor operates under a separate clock. Since each processor can execute its own local program, the shared-memory model is a multiple instruction multiple data (MIMD) type. That is, each processor may execute an instruction or operate on data different from those executed or operated on by any other processor during any given time unit. There are several variations of the PRAM model based on the assumptions regarding the handling of the simultaneous access of several processors to the same location of the global memory. The *exclusive read exclusive write (EREW)* PRAM does not allow any simultaneous access to a single memory location. The *concurrent read exclusive write (CREW)* PRAM allows simultaneous access for a read instruction only. Access to a location for a read or a write instruction is allowed in the *concurrent read concurrent write (CRCW)* PRAM. The three principal varieties of CRCW PRAMs are differentiated by how concurrent writes are handled. The *common CRCW* PRAM allows concurrent writes only when all processors are attempting to write the same value. The *arbitrary CRCW* PRAM allows an arbitrary processor to succeed. The *priority CRCW* PRAM assumes that the indices of the processors are linearly ordered, and allows the one with the minimum index to succeed. Other variations of the CRCW PRAM model exist. It turns out that these three models (EREW, CREW, CRCW) do not differ substantially in their computational powers, although CREW is more powerful than the EREW, and the CRCW is the most powerful.

## 1.5 Application and Known Results

The parallel algorithms for vertex-ranking problem and edge-ranking problem have received much attention these days because of their growing number of applications. The parallel algorithms for ranking problem plays an important role in the parallel Cholesky factorization of matrices [DR83, Lui90]. Let $M$ be a sparse symmetric matrix, and let $M'$ be a matrix obtained from $M$ by replacing each nonzero element with 1. Let $G$ be a graph with adjacency matrix $M'$. Then an optimal $c$-vertex-ranking of $G$ corresponds to a generalized Cholesky factorization of $M$ having the minimum recursive depth. It also has applications in VLSI layout [SDG92] and in scheduling the parallel assembly of a complex multi-part product from its components [IRV88]. [RK01] proposed an algorithm of $O(\log_2 n)$ time using $O(n)$ operations on the common CRCW PRAM model for the $c$-vertex-ranking of trees. Whereas [KZN00] proposed an algorithm of $O(\log_2 n)$ time using $O(n^{6(k+1)(a+1)+1} . \log_2^{3(k+1)(3k+2)+1} n)$ operations on the common CRCW PRAM model for the $c$-vertex-ranking of partial $k$-trees, where $a \leq 5$. The problem of finding an optimal $c$-edge-ranking of a graph also has the application in scheduling the parallel assembly of a complex multi-part product from its components, where the vertices correspond to the components and the edges corresponds to assembly operation. 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

| Classes of Graphs | Time | Operations | PRAM | Ref |
|---|---|---|---|---|
| Trees | $O(log_2 n)$ | $O(n)$ | EREW | [RK01] |
| Partial $k$-Trees | $O(log_2 n)$ | $O(n^{6(k+1)(a+1)+1} \cdot log_2^{3(k+1)(3k+2)+1} n)$ where, $a \leq 5$ | Common CRCW | [KZN00] |
| Series-Parallel Graphs | $O(log_2 n)$ | $O(n^{109} \log_2^{73} n)$ | Common CRCW | [KZN00] |
| Our Results | | | | |
| Series-Parallel Graphs | $O(log_2 n \log_2 log_2 n)$ | $O(c^2 n^5 \log_{c+1}^9 n)$ | Priority CRCW | |
| Series-Parallel Graphs | $O(log_2 n)$ | $O(c^3 n^7 \log_{c+1}^1 3n)$ | Priority CRCW | |

Table 1.1: Known results and our results for $c$-vertex-ranking.

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 $G$. Fig. 1.6(b), which shows the separator tree of Fig. 1.6(a), shows that one can assemble in parallel a product of Fig. 1.6(a) using two robots of three hands. Note that, among the three connected components in step 1, there are two connected components which are not isolated vertex. In each step a robot can simultaneously connect at most three connected components of the previous step. For the edge ranking similar results are found. [RK01] proposed an algorithm of $O(\Delta \log_2 n)$ time using $O(\Delta n)$ operations on the common CRCW PRAM model for the $c$-edge-ranking of trees. Whereas [KZN00] proposed an algorithm of $O(\log_2 n)$ time using $O(n^{6(k+1)(b+1)+1} \cdot \log_2^{3(k+1)(3k+2)+1} n)$ operations on the common CRCW PRAM model for the $c$-edge-ranking of partial $k$-trees, where $b \leq 3\Delta$. Table 1.1 and 1.2 summarizes the known results for $c$-vertex-rankings and $c$-edge-rankings for different graphs. The tables also show our results for $c$-vertex-rankings and $c$-edge-rankings for series-parallel graphs.

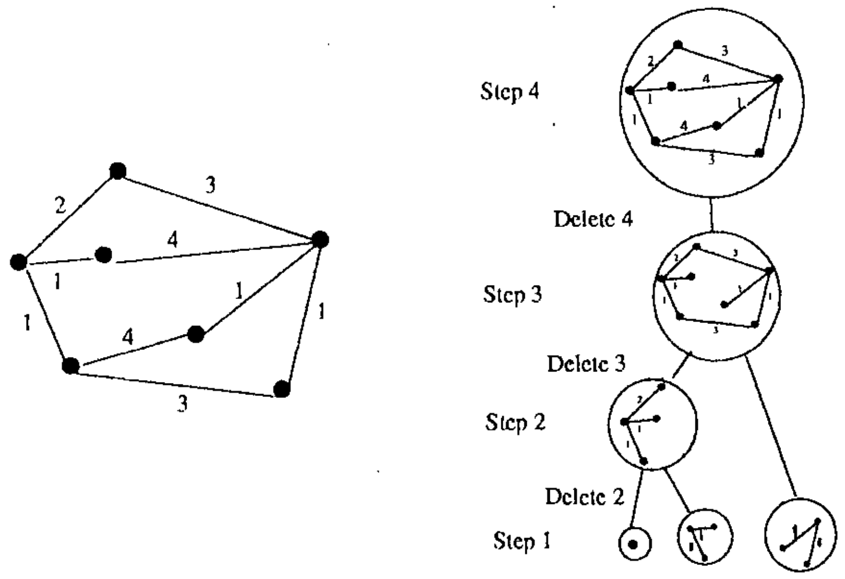


Figure 1.6: (a) An optimal 2-edge-ranking, (b) its corresponding separator tree.

| Classes of Graphs | Time | Operations | PRAM | Ref |
|---|---|---|---|---|
| Trees | $O(\Delta log_2 n)$ | $O(\Delta n)$ | EREW | [RK01] |
| Partial $k$-Trees | $O(log_2 n)$ | $O(n^{6(k+1)(b+1)+1} \cdot log_2^{3(k+1)(3k+2)+1} n)$ where, $b \leq 3\Delta$ | Common CRCW | [KZN00] |
| Series-Parallel Graphs | $O(log_2 n)$ | $O(n^{54\Delta+19} \log_2^{74} n)$ | Common CRCW | [KZN00] |
| Our Results | | | | |
| Series-Parallel Graphs | $O(log_2 n \log_2 log_2 n)$ | $O(c^2 n^{4\Delta+1} \log_{c+1}^5 n)$ | Priority CRCW | |
| Series-Parallel Graphs | $O(log_2 n)$ | $O(c^3 n^{6\Delta+1} \log_{c+1}^7 n)$ | Priority CRCW | |

Table 1.2: Known results and our results for $c$-edge-ranking.

## 1.6 Summary

In this thesis we present parallel algorithms which give the $c$-vertex-ranking and $c$-edge-ranking of a series-parallel graph. Our algorithm for $c$-vertex-ranking runs in $O(\log_2 n)$ time using $O(c^3 n^7 \log_{c+1}^{13} n)$ operations. We also give an algorithm that runs in $O(\log_2 n \log_2 \log_2 n)$ time using $O(c^2 n^5 \log_{c+1}^{9} n)$ operations. The parallel computation model we use is priority CRCW PRAM. Our algorithm for $c$-edge-ranking runs in $O(\log_2 n)$ time using $O(c^3 n^{6\Delta+1} \log_{c+1}^{7} n)$ operations. We also give an algorithm that runs in $O(\log_2 n \log_2 \log_2 n)$ time using $O(c^2 n^{4\Delta+1} \log_{c+1}^{7} n)$ operations. Here also the parallel computation model we use is priority CRCW PRAM.

Rest of the thesis is organized as follows. Chapter 2 gives preliminaries. Chapter 3 and 4 give the parallel algorithms for $c$-vertex-ranking and $c$-edge-ranking for series-parallel graphs, respectively. Finally Chapter 5 concludes with a discussion of the results and future works.

# Chapter 2

# Preliminaries

## 2.1 Introduction

In this chapter we give some basic definitions and elementary characteristics of series-parallel graphs. Definitions that are not included in this chapter will be introduced, as they are needed. We start, in Section 2.2, by giving some definitions of the standard graph-theoretical terms used throughout the remainder of the thesis. In Section 2.3 we give definitions of some special graphs used in the thesis and introduce the binary decomposition tree of a series-parallel graph.

## 2.2 Basic Terminology

### 2.2.1 Graphs and Multigraphs

A graph $G$ is a structure $(V, E)$ which consists of a finite set of vertices $V$ and a finite set of edges $E$, where each edge is an unordered pair of vertices. We call $V(G)$ the *vertex-set* of the graph $G$, and $E(G)$ the *edge-set* of $G$. Throughout 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 $w$ and $v$ are neighbors of each other, and that $e$ is incident to $v$ and

$w$. If a graph $G$ has no "multiple edge" or "loop", 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 are called *multigraph*. Sometimes a simple graph is simply called by a *graph* only if there is no confusion.

### 2.2.2 Degree of a Vertex

The *degree* of a vertex $v$ in a graph $G$ is said to be the number of edges incident to $v$, and is denoted by $d_G(v)$ or simply by $d(v)$. The *maximum degree of* $G$ is denoted by $\Delta(G)$ or simply by $\Delta$.

Fig. 1.1 depicts a graph of five vertices (each of degree 3 except $v_5$ which has degree 4) and eight edges, where vertices drawn by circles, edges by lines, vertex names next to the circles and edge names next to the lines.

### 2.2.3 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 write this as $G' \subseteq G$. 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 $G[V']$. If $V'$ consists of exactly the vertices on which edges in $E'$ are incident, then $G'$ is said to be the *subgraph induced by* $E'$, and is denoted by $G[V']$. Fig. 2.1(a) depicts a subgraph of $G$ in Fig. 1.1 induced by $\{v_1, v_2, v_4, v_5\}$ and Fig. 2.1(b) depicts a subgraph induced by $\{e_1, e_2, e_4\}$. We often construct new graphs from old ones by deleting 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 $G$, then $G - v$ 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 $E'$.

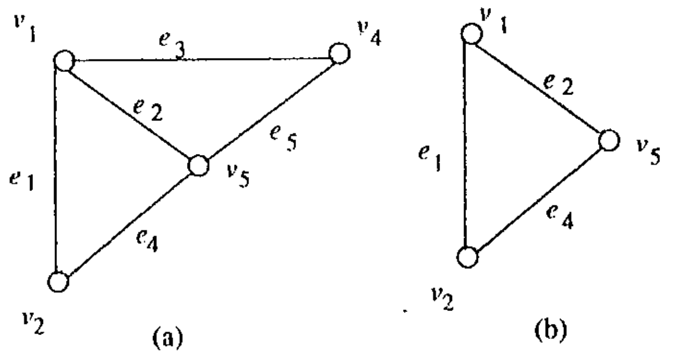


Figure 2.1: (a) A subgraph induced by vertices, (b) a subgraph induced by edges.

### 2.2.4 Paths and Cycles

A $v_0 - v_t$ walk in $G$ is an alternating sequence of vertices and edges of $G$, $v_0, e_1, ..., v_{l-1}, e_l, v_l$ beginning and ending with a vertex, in which each edge is incident to two vertices immediately preceding and following it. If the vertices $v_0, v_1, ..., v_l$ are distinct (except, possibly, $v_0, v_t$), then the walk is called a *path* and is usually denoted by $v_0v_1...v_l$. The *length* of a path is $l$, one less that the number of vertices on the path. A path or a walk is *simple* if it does not include the same edge twice. A path or a walk is *closed* if $v_l = v_0$. A closed path containing at least one edge is called a *cycle*. One example of a walk in $G$ depicted in Fig. 1.1 is $v_1, e_1, v_2, e_8, v_3, e_7, v_4$, which is not closed. One example of a cycle is $v_1v_2v_3v_5v_1$.

## 2.3 Some Special Graphs

### 2.3.1 Partial $k$-Trees

A natural generalization of ordinary trees is the so-called $k$-trees. The class of $k$-trees is defined recursively as follows [Sch89].

1. A complete graph with $k$-vertices is a $k$-tree.

2. If $G = (V, E)$ is a $k$-tree and any $k$ vertices $v_1, v_2, ..., v_k$ induce a complete subgraph of $G$, then $G' = (V \cup \{w\}, E \cup \{(v_i, w) | 1 \leq i \leq k\})$ is a $k$-tree, where $w$ is a new vertex not contained in $G$.

3. All $k$-trees can be formed with rules (a) and (b).

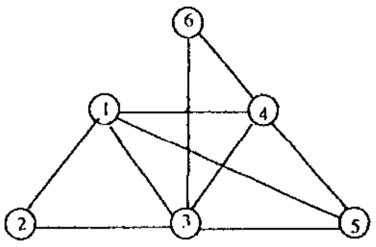


Figure 2.2: A partial 3-tree.

A graph is called a *partial $k$-tree* if it is a subgraph of a $k$-tree. Thus a partial $k$-tree $G = (V, E)$ is a simple graph without multiple edges or self-loops, and $|E| \leq kn$. Fig. 2.2 depicts a partial 3-tree.

### 2.3.2 Series-Parallel Graphs

A (two-terminal) *series-parallel graph* is defined recursively as follows (See Fig. 2.3).

1. A graph $G$ of a single edge is a series-parallel graph. The ends $v_s$ and $v_t$ of the edge are called the *terminals* of $G$ and are denoted by $v_s(G)$ and $v_t(G)$, respectively.

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 a series-parallel with terminals $v_s(G_2)$ and $v_t(G_2)$. Then,

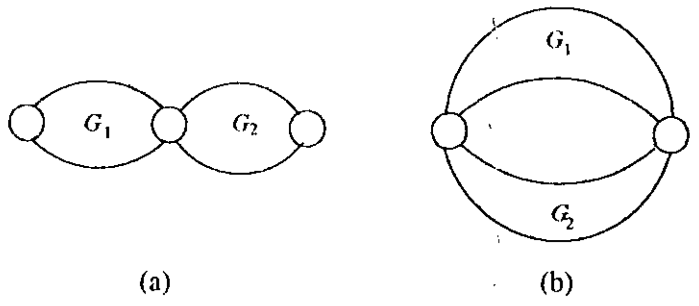


Figure 2.3: (a) A series connection, (b) a parallel connection.

   (a) A graph $G$ obtained from $G_1$ and $G_2$ by identifying vertex $v_t(G_1)$ with $v_s(G_2)$ 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 $G = G_1 \bullet G_2$.

   (b) A graph $G$ obtained from $G_1$ and $G_2$ by identifying vertex $v_s(G_1)$ 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 $G = G_1 \parallel G_2$.

The Terminals $v_s(G)$ and $v_t(G)$ of $G$ are often denoted shortly by $v_s$ and $v_t$.

A series-parallel graph can be represented by a "binary decomposition tree" [TNS82]. Fig. 2.4 illustrates a series-parallel graph graph $G$ and its binary

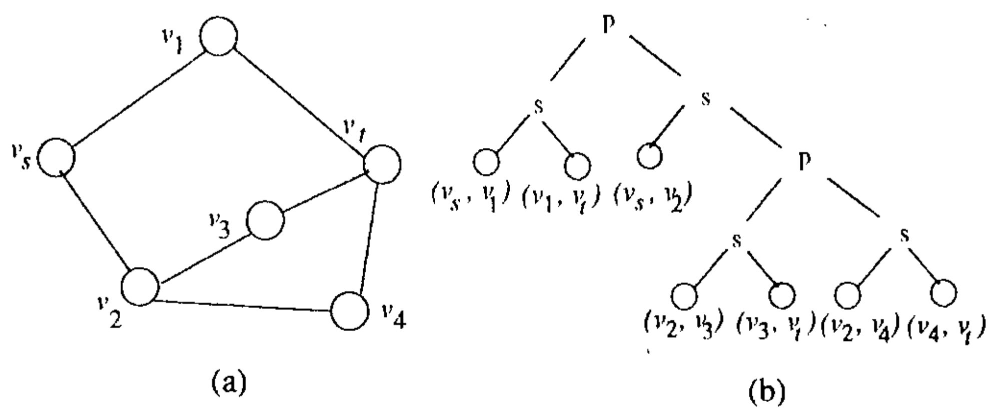

$v_1$
$v_s$
$v_t$
$v_3$
$v_2$
$v_4$
(a)
p
s
s
p
s
s
$(v_s, v_1)$ $(v_1, v_t)$ $(v_s, v_2)$
$(v_2, v_3)$ $(v_3, v_t)$ $(v_2, v_4)$ $(v_4, v_t)$
(b)

Figure 2.4: (a) A series-parallel graph, (b) its binary decomposition tree $T_b$.

decomposition tree $T_b$. Labels s and p attached to internal nodes in $T_b$ indicates series and parallel connections, respectively, and nodes labeled s and p are called *s-* and *p-nodes*, respectively. A node $u$ of tree $T_b$ corresponds to a subgraph of $G$, which is denoted by $G_u$. A leaf of $T_b$, in particular, represents a subgraph of $G$ induced by two vertices, that is, an edge. Let $T(x)$ denote the subtree of $T$ rooted at node $x$. Let $S_x = \{v_x, v_t\}$ be the set of terminals of $G_x$. We associate a subgraph $G_x = (V_x, E_x)$ of $G$ with each node $x$ of $T$, where

$$V_x = \bigcup\{S_y | y = x \text{ or } y \text{ is a descendant of } x \text{ in } T\},$$

$$E_x = \bigcup\{e_y | y \text{ is a leaf node in } T(x)\}.$$

The graph associated with the root of $T$ is the given graph $G$ itself.

## 2.4 Visible Vertices and Visible Edges

Let $\varphi$ be a vertex-labeling of a series-parallel graph $G = (V, E)$ with positive integers. The label (rank) of a vertex $v \in V$ is denoted by $\varphi(v)$. The number of ranks used by a vertex-labeling $\varphi$ is denoted by $\#\varphi$. Without loss of generality it can be assumed that $\varphi$ uses consecutive integers $1, 2, 3..., \#\varphi$ as the ranks. The rank of a vertex $u \in V$ is said to be ***visible*** from a vertex $v \in V$ under $\varphi$ in $G$ if $G$ has a path $P$ from $u$ to $v$ every vertex of which has a rank $\leq \varphi(u)$. Thus the smallest rank visible from $v$ under $\varphi$ is equal to $\varphi(v)$. The list of all visible ranks from $v$ is called ***visible set*** of $v$ and is denoted by $L(\varphi, v)$. Fig. 2.5 depicts the visible list for a ranking of a tree where the number and the number set next to the node represent the rank and the visible list for the node respectively. In this figure, for example, the nodes $a, b, c, h$ are visible from node $a$, so the visible list for node $a$ is the set of ranks of $a, b, c, h$, that is, $\{2,2,2,1\}$. If $\varphi$ is a $c$-vertex-ranking of $G$, then we note that a distinct rank $i$ can appear $\leq c$ times in $L(\varphi, v)$.

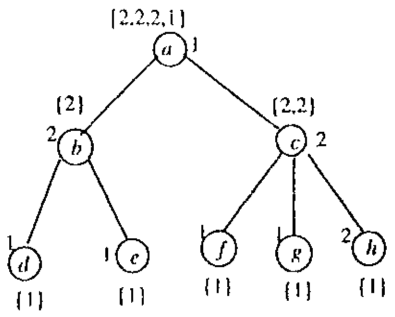


Figure 2.5: A ranking of a tree.

For a subgraph $G_x = (V_x, E_x)$ of $G$, we denote by $\varphi|G_x$ a ***restriction*** of $\varphi$ to $G_x$: let $\varphi' = \varphi|G_x$, then $\varphi'(v) = \varphi(v)$ for $v$ in $V_x$. The following lemma is cited from [KR99].

**Lemma 2.1** *Let $T$ be a binary decomposition tree of a series-parallel graph $G$, and let $x$ be a node in $T$. Then a vertex-labeling $\varphi$ of $G_x$ is a c-vertex-ranking of $G_x$ if and only if,*

1. *at most $c$ vertices of the same rank are visible from any vertex $v \in S_x \bigcup S_y$ under $\varphi$ in $G_x$; and*

2. *if $x$ is an internal node in $T$ and has two children $y$ and $z$, then $\varphi|G_y$ and $\varphi|G_z$ are c-vertex-rankings of $G_y$ and $G_z$, respectively.* □

Similarly let $\varphi$ be an edge-labeling of a series-parallel graph $G = (V, E)$ with positive integers. The label (rank) of an edge $e \in E$ is denoted by $\varphi(e)$. The number of ranks used by an edge-labeling $\varphi$ is denoted by $\#\varphi$. Without loss of generality it can be assumed that $\varphi$ uses consecutive integers $1, 2, 3..., \#\varphi$ as the ranks. The rank of an edge $u \in E$ is said to be ***visible*** from an edge $e \in E$ under $\varphi$ in $G$ if $G$ has a path $P$ from $u$ to $e$ every edge of which has a rank $\leq \varphi(u)$ Thus the smallest rank visible from $e$ under $\varphi$ is equal to $\varphi(e)$. The list of all visible ranks from $e$ is called ***visible set*** of $e$ and is denoted by $L(\varphi, e)$. If $\varphi$ is a $c$-edge-ranking of $G$, then we note that a distinct rank $i$ can appear $\leq c$ times in $L(\varphi, e)$.

For a subgraph $G_x = (V_x, E_x)$ of $G$, we denote by $\varphi|G_x$ a *restriction* of $\varphi$ to $G_x$: let $\varphi' = \varphi|G_x$, then $\varphi'(e) = \varphi(e)$ for $e$ in $E_x$. Then the Lemma 2.1 also holds for $c$-edge-ranking.

## 2.5 Conclusion

In this chapter we have introduced the basic definitions relating to graphs and multigraphs, degree of a vertex, subgraphs, paths and cycles etc. Also we have introduced some special graphs like partial $k$-trees and series-parallel graphs. At the end of thesis chapter we have introduced the definitions and some properties of visible vertices and visible edges. The definitions introduced in this chapter will help understanding the remaining of the thesis.

# Chapter 3

# Vertex Rankings of Series-Parallel Graphs

## 3.1 Introduction

This chapter deals with the parallel algorithm for generalized vertex-ranking problem on series-parallel graphs. Since a series-parallel graph is a partial 2-tree, one can obtain a parallel algorithm for generalized vertex-ranking of series-parallel graphs from the parallel algorithm of partial $k$-trees [KZN00]. However that would cost $O(n^{109} \log_2^{73} n)$ operations in worst case. In this chapter we present a parallel algorithm that gives the $c$-vertex-ranking of a series-parallel graph $G$ in $O(\log_2 n)$ time using $O(c^3 n^7 \log_{c+1}^{13} n)$ operations if a binary decomposition tree $T_b$ of $G$ is given. We also give an algorithm that runs in $O(\log_2 n \log_2 \log_2 n)$ time using $O(c^2 n^5 \log_{c+1}^{9} n)$ operations. The parallel computation model we use is priority CRCW(Concurrent read Concurrent Write) PRAM(Parallel Random Access Machine).

The rest of the chapter is organized as follows. Section 3.2 cites the upper bound for $c$-vertex-ranking number. Sections 3.3 and 3.4 show the equations to calculate the equivalence class and dominance class, respectively. Section 3.5 discusses the traditional tree contraction algorithm and the modified shunt operation used in our algorithm. Section 3.6 finally gives our parallel algorithms.

## 3.2 The Upper Bound for $c$-Vertex-Ranking Number

In this section we cite the result on the $c$-vertex-ranking number $r_c(G)$ of a series-parallel graph $G$ from [K01]. We first cite the following lemma from [ZNN95].

**Lemma 3.1** *Every tree $T$ of $n$ vertices has a vertex whose removal leaves subtrees each having at most $n/2$ vertices.* □

Using Lemma 3.1 [K01] proved the following lemma.

**Lemma 3.2** *For any positive integer $\alpha$, every tree $T$ of $n$ vertices has at most $\alpha$ vertices whose removal leaves subtrees each having at most $\lfloor n/(\alpha+1) \rfloor$ vertices.* □

Using the above two lemmas [K01, KR99] proved the following lemma.

**Lemma 3.3** *Let $c$ be any positive integer which is not always bounded, and let $G$ be a series-parallel graph of $n$ vertices. Then*

$$r_c(G) \leq 1 + 2\log_{c+1} n. \ \square$$

## 3.3 Equivalence Class

Many algorithms on series-parallel graphs use equivalence class. On each node of the tree decomposition, a table of all possible partial solutions of the problem is computed, where each entry in the table represents an equivalent class. The number of operations for parallel algorithm depends on the size of the table. Therefore a suitable equivalence class is always found out. Kashem *et. al* defined a suitable equivalence class for series-parallel graphs [KR99].

Let $T_b$ be a binary decomposition tree of a series-parallel graph $G$. Let $R = 1, 2, ..., m$ be the set of ranks. Let $x$ be a node in $T_b$, and let $\varphi : V_x \to R$ be a vertex labeling of the subgraph $G_x = (V_x, E_x)$. Iyer *et. al.* introduced the idea of a "critical list" to solve the ordinary vertex-ranking problem [IRV88]. A similar idea of *visible list-set* $L(\varphi, v)$ is as follows [KZN00]:

$$L(\varphi, v) = \{\varphi(u) \mid u \in V_x \text{ is visible from } v \text{ under } \varphi \text{ in } G_x\}.$$

Let $L$ be multi-set of integers. Then by *count*$(L, i)$ we denote the number of $i$'s contained in $L$.

For a vertex-labeling $\varphi$ of $G_x$, *minimum obstacle* in the path from $v_s$ to $v_t$ $\lambda_\varphi$, $\lambda_\varphi \in R \bigcup \{0\}$ is defined as follows:

$\lambda_\varphi = \min\{\lambda | G_x$ has a path $P$ from $v_s$ to $v_t$ such that $\varphi(u) \leq \lambda$ for each internal vertex $u \in P\}$.

Let $\lambda_\varphi = 0$ if $(v_s, v_t) \in E_x$. If vertex $u \in V_x$ is visible from $v_t$ and $\varphi(u) \geq \max\{\varphi(v_s), \lambda_\varphi\}$ then $u$ is also visible from $v_s$.

Let $l$ be a positive integer, ranging from 1 to $\lambda_\varphi$. A *peer-list* $L_p(\varphi, v, l)$ and a *peer-list-set* $\mathcal{L}_p(\varphi)$ for a series-parallel graph $G_x$ are defined as follows:

$$L_p(\varphi, v_s, l) = L(\varphi, v_s) \bigcup [l \leq L(\varphi, v_t) < \lambda_\varphi];$$

$$L_p(\varphi, v_t, l) = L(\varphi, v_t) \bigcup [l \leq L(\varphi, v_s) < \lambda_\varphi]; \text{ and}$$

$$\mathcal{L}_p(\varphi) = (L_p(\varphi, v_s, l), L_p(\varphi, v_t, l)).$$

Actually, peer-list from vertex $v_s$ incorporates portion of visible-list at $v_t$, as if there exist another path from $v_s$ to $v_t$ having $l$ as the minimum obstacle.

This extra component in peer-list is called by *look-ahead-component*. A pair $\mathcal{R}_p(\varphi)$, called *peer-vector*, is defined as follows:

$$\mathcal{R}_p(\varphi) = (\mathcal{L}_p(\varphi), \lambda_\varphi).$$

A $\mathcal{R}_p(\varphi)$ is called a *feasible vector* if the vertex-labeling $\varphi$ is a $c$-vertex-ranking of $G_x$ and $count(L_p(\varphi, v, l), i) \leq c$ for all $v \in S_x$ and $i \in R$. We then cite the following lemma from [KR99].

**Lemma 3.4** *Let $\varphi$ and $\eta$ be two c-vertex-rankings of $G_x$, such that $\mathcal{R}_p(\varphi) = \mathcal{R}_p(\eta)$. Then $\varphi$ is extensible if and only if $\eta$ is extensible.* □

If either $G_x = G_y \bullet G_z$ or $G_x = G_y \parallel G_z$ then a vertex labeling of $G_x$ can be obtained from the $c$-vertex-rankings of $G_y$ and $G_z$ by using the following two important lemmas [KR99]:

**Lemma 3.5** *Let $G_x = G_y \bullet G_z$. Let $v_s$ and $v$ be the terminal nodes of $G_y$ and let $v$ and $v_t$ be the terminal nodes of $G_z$. Let $\eta$ and $\psi$ be the c-vertex-rankings of $G_y$ and $G_z$, respectively. Let $l_\eta = max\{l, \psi(v_t), \lambda_\psi\}$, $l_\psi = max\{r, \eta(v_s), \lambda_\eta\}$, $\eta(v) = \psi(v)$ and $\varphi$ be the vertex-labeling of $G$ extended from $\eta$ and $\psi$. Then*

$$L_p(\varphi, v_s, l) = L_p(\eta, v_s, l_\eta) \bigcup [L_p(\psi, v, l) \geq \lambda_\eta], \text{ if} \lambda_\eta < l; \text{ or}$$

$$L_p(\varphi, v_s, l) = L_p(\eta, v_s, l_\eta) \bigcup [L_p(\psi, v_t, \lambda_\eta) \geq l], \text{ if} \lambda_\eta \geq l; \text{ and}$$

$$L_p(\varphi, v_t, r) = L_p(\psi, v_t, l_\psi) \bigcup [L_p(\eta, v, r) \geq \lambda_\psi], \text{ if} \lambda_\psi < r; \text{ or}$$

$$L_p(\varphi, v_t, r) = L_p(\psi, v_t, l_\varphi) \bigcup [L_p(\eta, v_s, \lambda_\psi) \geq r], \text{ if} \lambda_\psi \geq r. \square$$

**Lemma 3.6** *Let $G_x = G_y || G_z$. Let $v_s$ and $v_t$ be the terminal nodes of $G_y$ and let $v_s$ and $v_t$ be the terminal nodes of $G_z$. Let $\eta$ and $\psi$ be the c-vertex-ranking of $G_y$ and $G_z$, respectively. Let $\eta(v_s) = \psi(v_s)$ and $\eta(v_t) = \psi(v_t)$. Let $\varphi$ be the vertex-labeling of $G_x$ extended from $\eta$ and $\psi$. Then*

$$L_p(\varphi, v_s, l) = L_p(\eta, v_s, l) \bigcup L_p(\psi, v_s, l); \text{ and}$$

$$L_p(\varphi, v_t, l) = L_p(\eta, v_t, l) \bigcup L_p(\psi, v_t, l). \ \Box$$

## 3.4 Dominance Class

If $\mathcal{R}_p(x)$ is the set of feasible vectors for $G_x$ then applying the following elimination rule on $\mathcal{R}_p(x)$, Kashem *et. al.* have obtained the dominance class $\mathcal{R}_d(x)$ [KR99]. If $\varphi$ and $\varphi'$ are any two labellings of $G_x$, we say that $\varphi$ *dominantes* $\varphi'$ if any of the following conditions is true:

1. $L_p(\varphi, v_s, l) = L_p(\varphi', v_s, l)$, $L_p(\varphi, v_t, l) \preceq L_p(\varphi', v_t, l)$,

   for $\lambda_\varphi = \lambda_{\varphi'}$, $\varphi(v_s) = \varphi'(v_s)$, and $\varphi(v_t) = \varphi'(v_t)$;

2. $L_p(\varphi, v_s, l) \preceq L_p(\varphi', v_s, l)$, $L_p(\varphi, v_t, l) = L_p(\varphi', v_t, l)$,

   for $\lambda_\varphi = \lambda_{\varphi'}$, $\varphi(v_s) = \varphi'(v_s)$, and $\varphi(v_t) = \varphi'(v_t)$.

The set of feasible vectors obtained from $\mathcal{R}_p(x)$ applying the elimination rule is called *dominance class* $\mathcal{R}_d(x)$. Then we have the following lemma from [KR99]:

**Lemma 3.7** *Let $\mathcal{R}'_d(x)$, $\mathcal{R}_d(y)$ and $\mathcal{R}_d(z)$ be the dominance classes obtained from $\mathcal{R}_p(x)$, $\mathcal{R}_p(y)$ and $\mathcal{R}_p(z)$, respectively. Let $\mathcal{R}_d(x)$ be the dominance class extended from $\mathcal{R}_d(y)$ and $\mathcal{R}_d(z)$. Then $\mathcal{R}'_d(x) = \mathcal{R}_d(x)$.* $\Box$

By Lemma 3.4 and Lemma 3.7 a dominance class $\mathcal{R}_d(x)$ can be seen as a set of extensible $c$-vertex-rankings of $G_x$. Since $|R| = m$, $1 \le \lambda_\varphi \le m$, and $0 \le count(\varphi, v, l) \le c$ for a $c$-vertex-ranking $\varphi$ and a rank $i \in R$, the number of peer-lists $L_p(\varphi, v, l)$ is at most $m(c+1)^m$ for each vertex $v \in S_x = \{v_s, v_t\}$. For each peer-list $L_p(\varphi, v_s, l)$, there can be at most $m$ peer-lists $L_p(\varphi, v_t, l)$ for different $l$. The number of distinct peer-list-sets $\mathcal{L}_p(\varphi)$ can be at most $2m^2(c+1)^m$. Again each of $\varphi(v_s)$ and $\varphi(v_t)$ can assume any value from 1 to $m$. Since by Lemma 3.3 we have $m \le 1 + 2\log_{c+1} n = O(\log_{c+1} n)$, the total number of different feasible vectors on $x$ is $O(cn^2 \log^4_{c+1} n)$..

**Lemma 3.8** *For a node $x$ the dominance class $\mathcal{R}_d(x)$ from the peer vector $\mathcal{R}_p(x)$ can be computed on the priority CRCW model either in $O(1)$ time using $O(\alpha^3 m)$ operations, or in $O(\log_2 \log_2 \alpha)$ time using $O(\alpha^2 m)$ operations, where $\alpha$ is the cardinality of $\mathcal{R}_d(x)$ and $m$ is the length of a list in $\mathcal{R}_d(x)$.*

**Proof:** For each list $L_p(\varphi, v_s, l)$, there are at most $\alpha$ number of lists $L_p(\varphi, v_t, l)$. Computing minimum of such $\alpha$ lists can be done on the priority CRCW either in $O(1)$ time using $O(\alpha^2 m)$ operations, or in $O(\log_1 \log_2 \alpha)$ time using $O(\alpha m)$ operations [Já92]. Since there are at most $\alpha$ lists $L_p(\varphi, v_s, l)$, the dominance class can be computed on the priority CRCW either in $O(1)$ time using $O(\alpha^3 m)$ operations or in $(\log_2 \log_2 \alpha)$ time using $O(\alpha^2 m)$ operations.□

## 3.5 Tree Contraction Algorithm

The tree contraction algorithm was originally introduced by Miller and Reif [MR85]. It takes $O(\log_2 n)$ time using $O(n)$ processors, where $n$ is the number of nodes in the tree. Several authors [ADK+89, GMT88, He91, HY88, KD88] have improved the algorithm as follows so that it takes $O(\log_2 n)$ time using

$O(n/\log_2 n)$ processors. A *structure* is a triple $(A, F_{node}, F_{edge})$ consisting of a set $A$, a *node function* set $F_{node} \subseteq \{f \mid f : A \times A \longrightarrow A\}$, and an *edge function* set $F_{edge} \subseteq \{f \mid f : A \longrightarrow A\}$. A *bottom-up algebraic tree computation tree* on $(A, F_{node}, F_{edge})$ is a binary tree $T_b$ such that: each leaf node $v$ of $T_b$ is labeled by a function $f_u \in F_{node}$; and each edge $e$ of $T_b$ is labeled by a function $f_e \in F_{edge}$. A label $\mathcal{R}(u) \in A$ of each internal node $u$ of $T_b$ is recursively defined as

$$\mathcal{R}(u) = f_u(f_{e_1}(\mathcal{R}(v_1)), f_{e_2}(\mathcal{R}(v_2))),$$

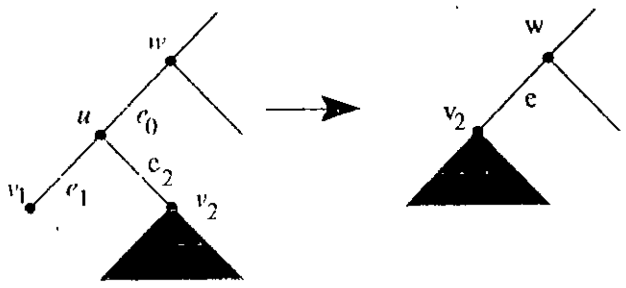


Figure 3.1: Shunt operation.

where $v_1$ and $v_2$ are the left and right child of $u$ in $T_b$, $e_1 = (v_1, u)$, and $e_2 = (v_2, u)$ (see Fig. 3.1). The *bottom-up algebraic tree computation (BATC)* problem on $T_b$ is to compute $\mathcal{R}(u)$ for the root $u$ of $T_b$. In order to solve the BATC problem in parallel, He and Yesha introduced the following *shunt operation* [HY88]. Let $u$ be a node of $T_b$ with left child $v_1$, right child $v_2$, and parent $w$. Let $e_1 = (v_1, u)$, $e_2 = (v_2, u)$, and $e_0 = (u, w)$. Suppose $u$ is the left node of $w$(Fig. 3.1). A shunt operation on $v_1$ is defined as follows: delete $v_1$ and $u$ from $T_b$; make $v_2$ the left child of $w$ with a new edge $e = (v_2, w)$; and assign $e$ a function $f_e$ defined by

$$f_e(\mathcal{R}) = f_{e_0}(f_u(f_{e_1}(\mathcal{R}(v_1)), f_{e_2}(\mathcal{R})))$$

For variable $\mathcal{R} \in A$, if the right child $v_2$ of $u$ is a leaf node, then a shunt operation performed on $v_2$ is defined similarly. Clearly a shunt operation does not affect subsequent evaluation on $T_b$. The following elegant tree-contraction algorithm solves the BATC problem [ADK+89, GMT88, He91, HY88, KD88, MR85]:

*TREE CONTRACTION ALGORITHM*

```
begin
    for each leaf $v$ in parallel do
        index($v$) ← index from left to right with increasing number;
    repeat $\lceil \log n \rceil - 1$ times
        for each left leaf $v$ with odd index in parallel do
            if the $v$'s parent is not the root of $T_b$ then shunt $v$;
        for each right leaf $v$ with odd index in parallel do
            if the $v$'s parent is not the root of $T_b$ then shunt $v$;
        for each leaf $v$ in parallel do
            index($v$) ← index($v$)/2
    compute the label of the root in $T_b$;
end
```

The following theorem was proved in [ADK+89, GMT88, He91, HY88, KD88].

**Theorem 3.1** *The tree contraction algorithm correctly solves the BATC problem. Moreover, if the evaluation of node and edge functions and the shunt* *operation can be done by $p$ processors in $O(1)$ time, this algorithm can be implemented in $O(\log n)$ time using $O(pn/\log n)$ processors, where $n$ is the number of nodes in $T_b$.* □

### 3.5.1 Modified Shunt Operation

A *temporary label* $\mathcal{R}'(u)$ of a node $u$ is a label of $u$ which can be changed.

During the *modified shunt operation* of a node $u$ we delete node $v_1$ and $u$, but do not transfer the label of non-leaf child $v_2$ in place of $u$. Rather we calculate the temporary label $\mathcal{R}'(u)$ based on the temporary label of $v_2$, that is $\mathcal{R}'(v_2)$, and store it in node $v_2$ as new $\mathcal{R}'(v_2)$. We update $\mathcal{R}'(u)$ with the successive change in $\mathcal{R}'(v_2)$. In this way we get the actual value of $u$, that is $\mathcal{R}(u)$, when the actual value of $v_2$, that is $\mathcal{R}(v_2)$, is obtained.

The proof for the following theorem is immediate .

**Theorem 3.2** *If the edge or node function of the modified shunt operation can be done by $p$ processors in $O(1)$ time then Theorem 3.1 also holds for our modified algorithm to solve the BATC problem.* □

## 3.6 Parallel Algorithm

The main result of this chapter can be stated by the following theorem.

**Theorem 3.3** *For any positive integer $c$, an optimal $c$-vertex-ranking of a series-parallel graph $G$ with $n$ vertices can be found on priority CRCW either in*

1. *time $O(\log_2 n)$ using $O(c^3 n^7 \log_{c+1}^{13} n)$ operations; or*

2. *time $O(\log_2 n \log_2 \log_2 n)$ using $O(c^2 n^5 \log_{c+1}^{9} n)$ operations.*

**Proof:** Let $T_b$ be a binary decomposition-tree of $G$. we first give a parallel algorithm to decide whether, for a given positive integer $m$, $G$ has a $c$-vertex-ranking $\varphi$ with $\#\varphi \leq m$. We use tree contraction algorithm on the binary decomposition tree $T_b$ as follows: for each node $x$ of $T_b$ from leaves to root, we construct dominance class of $c$-vertex-rankings of $G_x$ from those of two subgraphs $G_y$ and $G_z$ associated with the children $y$ and $z$ of $x$. If the dominance class at the root of $T_b$ is non-empty, then the series-parallel graph $G$ corresponding to the root of $T_b$ has a $c$-vertex-ranking $\varphi$ such that $\#\varphi \leq m$. Then by using a binary search over the range of $m$, $1 \leq m \leq 1 + 2\log_{c+1} n$, we determine the minimum value of $m$ such that $G$ has a $c$-vertex-ranking $\varphi$ with $m = \#\varphi$ and find an optimal $c$-vertex-ranking of $G$.

We first compute the dominance class for each leaf in parallel. We then show how we use the the tree contraction algorithm on $T_b$. Initially temporary table of dominance class for each internal node is empty. We first number all the leaves, except the first one, from left to right in increasing order starting from 1. Then we contract each odd numbered leaf and its parent, if it is not the root, by using modified shunt operation in parallel. Let $v_1$ and its parent $u$ is to be contracted and $v_2$ be the other child. Also let $w$ is the parent of $u$ (see Fig. 3.1). Then $v_2$ may be a leaf or not. We calculate $\mathcal{R}'(u)$ by the node function $f(u)$ over $\mathcal{R}(v_1)$ and temporary label $\mathcal{R}'(v_2)$ as follows :

$$\mathcal{R}'(u) = f_u(f_{e_1}(\mathcal{R}(v_1)), f_{e_2}(\mathcal{R}'(v_2)))$$

Here the node function is the cross product of the tables of dominance classes of $v_1$ and $v_2$ (which are $\mathcal{R}(v_1)$ and $\mathcal{R}'(v_2)$, respectively) and the application of the elimination rule according to the definition of dominance class and Lemma 3.8. If $\mathcal{R}'(v_2)$ is empty we consider all possible values of $\mathcal{R}'(v_2)$. Our edge function is the selection function. The existing $\mathcal{R}'(u)$ may be empty or not before the node function. Then we have two cases. Case 1: if the existing $\mathcal{R}'(u)$ is empty then we store the newly calculated $\mathcal{R}'(u)$ in node $v_2$ as $\mathcal{R}'(v_2)$ and index it by, if there any, the the pointers by which corresponding $\mathcal{R}'(v_2)$ is indexed or by corresponding $\mathcal{R}'(v_2)$. Case 2: if the existing $\mathcal{R}'(u)$ is not empty then we use the newly calculated $\mathcal{R}'(u)$ to select entries from the existing $\mathcal{R}'(u)$. The selected entries are stores in node $v_2$ as $\mathcal{R}'(v_2)$ and we index it by, if there any, the pointers by which the corresponding $\mathcal{R}'(v_2)$ is indexed or by corresponding $\mathcal{R}'(v_2)$. It is clear that if $v_2$ is a leaf child then before the node function $\mathcal{R}(v_2) = \mathcal{R}'(v_2)$ and $\mathcal{R}'(v_2)$ is not pointed by any one. The above algorithm is cited in the following figure.

```
Procedure c-VERTEX-RANKING
//$u$ has a leaf child $v_1$, a non-leaf child $v_2$ and parent $w$. We contract $v_1$ and $u$.
begin
    if $v_2$ is a leaf then
        compute $\mathcal{R}'(u)$ as follows:
            take the cross product of $\mathcal{R}(v_1)$ and $\mathcal{R}'(v_2)$
            apply the elimination rule
            if the existing $\mathcal{R}'(u)$ is not empty then
                select the values from existing $\mathcal{R}'(u)$
                whose pointer matches with newly calculated $\mathcal{R}'(u)$
        store this $\mathcal{R}'(u)$ as $\mathcal{R}'(v_2)$ at node $v_2$
    else
        compute $\mathcal{R}'(u)$ as follows
            if $\mathcal{R}'(v_2)$ is empty then
                consider all possible values of $v_2$ as $\mathcal{R}'(v_2)$
        take the cross product of $\mathcal{R}(v_1)$ and $\mathcal{R}'(v_2)$
        apply the elimination rule
        if the existing $\mathcal{R}'(u)$ is not empty then
            select the values from existing $\mathcal{R}'(u)$ whose pointer
            matches with newly calculated $\mathcal{R}'(u)$
    store this $\mathcal{R}'(u)$ as $\mathcal{R}'(v_2)$ at node $v_2$ and update
    pointers accordingly
end
```

We now discuss the correctness and the complexity of our algorithm. First we discuss the correctness of our algorithm. Actually the modified shunt operation works by gradually selecting the correct values of $\mathcal{R}'(u)$ from all possible values of $\mathcal{R}'(u)$. So the corrected $\mathcal{R}'(u)$ is always subset of existing $\mathcal{R}'(u)$, if the existing $\mathcal{R}'(u)$ is not empty. In this way the final value of $\mathcal{R}'(u)$ achieved when the $v_2$ is a leaf. So the algorithm works correctly. We now discuss the complexity. We first show how to compute dominance class on a leaf $x$ of $T_b$. Since $|R| = m$, and $|S_x| = 2$, the number of vertex-labellings $\varphi : V_x \rightarrow R$ is at most $m^2$. For each vertex-labeling $\varphi$, $\lambda_\varphi (= 0)$ can be computed in $O(1)$ time. Since $\lambda_\varphi = 0$, the only possible value of $l$ is 0. The peer lists $L_p(\varphi, v, l), v \in S_x$, can be computed in parallel in time $O(1)$ using $O(1)$ operations. Then checking whether a vertex-labeling $\varphi$ is a $c$-vertex-ranking of $G_x$ can be done in time $O(1)$ using $O(1)$ operations and computing $\mathcal{R}(\varphi)$ can be done in time $O(1)$ with $O(1)$ operation. Therefore the peer class on a leaf can be found in time $O(1)$ using $O(m^2) = O(\log^2_{c+1} n)$ operations. We next show how to compute dominance class on an internal node $x$ of $T_b$ from those on two children $y$ and $z$ of $x$. The dominance class on an internal node

$x$ can be obtained from the cross product of the dominance classes of the two children $y$ and $z$. Since the cardinality of the dominance class on each node is $O(cn^2 \log_{c+1}^4 n)$, the cardinality of the cross product is $O(c^2n^4 \log_{c+1}^8 n)$. Since the length of a list is $O(\log_{c+1} n)$, by Lemma 3.8 the dominance class on $x$ can be computed either in $O(1)$ time using $O(c^3n^4 \log_{c+1}^9 n)$ operations if the constant time algorithm is used, or in $O(\log_2 \log_2 n)$ time using $O(c^2n^4 \log_{c+1}^8 n)$ operations if the doubly logarithmic algorithm is used. The node function (i.e. the selection function takes $O(1)$ time using operations no more than the maximum cardinality of a node. Finally the tree contraction takes $O(\log_2 n)$ time and $O(n)$ operations. So total complexity of the algorithm is either $O(\log_2 n)$ time using $O(c^3n^7 \log_{c+1}^{13} n)$ operations or $O(\log_2 n \log_2 \log_2 n)$ time using $O(c^2n^5 \log_{c+1}^9 n)$ operations. The PRAM model used by the algorithm is priority CRCW. $\square$

## 3.7 Conclusion

In this chapter we have introduced our main result for $c$-vertex-ranking. Before that we have discussed the necessary definitions and lemmas from the previous works on $c$-vertex-ranking of series-parallel graphs specially from [KR99]. The contents of this chapter are extensively used in the following chapter of $c$-edge-ranking.

# Chapter 4

# Edge Rankings of Series-Parallel Graphs

## 4.1 Introduction

This chapter deals with the parallel algorithm for generalized edge-ranking problem on series-parallel graphs. Since a series-parallel graph is a partial 2-tree, one can obtain a parallel algorithm for generalized edge-ranking of series-parallel graphs from the parallel algorithm of partial $k$-trees [KZN98]. However that would cost $O(n^{36\Delta+19} \log_2^{73} n)$ operations in worst case. In this chapter we present a parallel algorithm that gives the $c$-edge-ranking of a series-parallel graph $G$ in $O(\log_2 n)$ time using $O(c^3 n^{6\Delta+1} \log_{c+1}^{7} n)$ operations if a binary decomposition tree $T_b$ of $G$ is given. We also give an algorithm that runs in $O(\log_2 n \log_2 \log_2 n)$ time using $O(c^2 n^{4\Delta+1} \log_{c+1}^{5} n)$ operations. The parallel computation model we use is priority CRCW(Concurrent read Concurrent Write) PRAM(Parallel Random Access Machine).

The rest of the chapter is organized as follows. Section 4.2 cites the upper bound for $c$-edge-ranking number. Sections 4.3 show the equations to calculate the equivalence class and dominance class for c-edge-ranking. Section 4.4 finally gives our parallel algorithms.

## 4.2 The Upper Bound for $c$-Edge-Ranking Number

In this section we show that the $c$-edge-ranking number $r'_c(G)$ of a series-parallel graph $G$ is $O(\Delta \log_{c+1} n)$.

Using Lemma 3.3, we have the following lemma.

**Lemma 4.1** *Let $c$ be any positive integer which is not always bounded, and let $G$ be a series-parallel graph of $n$-vertices. Then*

$$r'_c(G) \leq 1 + 2\Delta \log_{c+1} n. \Box$$

## 4.3 Equivalence Class and Dominance Class

A *visible list-set* $\mathcal{R}(\varphi, v)$ for $c$-edge-ranking is defined as follows:

$$\mathcal{R}(\varphi, v) = \{\varphi(e) | e \in E_x \text{ is visible from } v \text{ under } \varphi \text{ in } G_x\}.$$

For an edge-labeling $\varphi$ of $G_x$, *minimum obstacle* in the path from $v_s$ to $v_t$ $\lambda_\varphi$, $\lambda_\varphi \in R \bigcup \{0\}$ is defined as follows: $\lambda_\varphi = \min\{\lambda | G_x$ has a path $P$ from $v_s$ to $v_t$ such that $\varphi(e) \leq \lambda$ for each internal edge $e \in P\}$.

All other definitions are same as in Sections 3.3 and 3.4. We then have the following two lemmas for $c$-edge-ranking.

**Lemma 4.2** *Let $G_x = G_y \bullet G_z$. Let $v_s$ and $v$ be the terminal nodes of $G_y$ and let $v$ and $v_t$ be the terminal nodes of $G_z$. Let $\eta$ and $\psi$ be the c-edge-rankings of $G_y$ and $G_z$, respectively. Let $l_\eta = max\{l, \lambda_\psi\}$, $l_\psi = max\{r, \lambda_\eta\}$, and $\varphi$ be the edge-labeling of $G$ extended from $\eta$ and $\psi$. Then*

$$L_p(\varphi, v_s, l) = L_p(\eta, v_s, l_\eta) \bigcup [L_p(\psi, v, l) \geq \lambda_\eta], \text{ if} \lambda_\eta < l; \text{ or}$$

$$L_p(\varphi, v_s, l) = L_p(\eta, v_s, l_\eta) \bigcup [L_p(\psi, v_t, \lambda_\eta) \geq l], \text{ if} \lambda_\eta \geq l; \text{ and}$$

$$L_p(\varphi, v_t, r) = L_p(\psi, v_t, l_\psi) \bigcup [L_p(\eta, v, r) \geq \lambda_\psi], \text{ if} \lambda_\psi < r; \text{ or}$$

$$L_p(\varphi, v_t, r) = L_p(\psi, v_t, l_\varphi) \bigcup [L_p(\eta, v_s, \lambda_\psi) \geq r], \text{ if} \lambda_\psi \geq r. \square$$

**Lemma 4.3** *Let $G_x = G_y || G_z$. Let $v_s$ and $v_t$ be the terminal nodes of $G_y$ and let $v_s$ and $v_t$ be the terminal nodes of $G_z$. Let $\eta$ and $\psi$ be the c-edge-ranking of $G_y$ and $G_z$, respectively. Let $\varphi$ be the edge-labeling of $G_x$ extended from $\eta$ and $\psi$. Then*

$$L_p(\varphi, v_s, l) = L_p(\eta, v_s, l) \bigcup L_p(\psi, v_s, l); \text{ and}$$

$$L_p(\varphi, v_t, l) = L_p(\eta, v_t, l) \bigcup L_p(\psi, v_t, l). \square$$

By Lemma 3.4 and Lemma 3.7 a dominance class $\mathcal{R}_d(x)$ can be seen as a set of extensible $c$-edge-rankings of $G_x$. Since $|R| = m, 1 \leq \lambda_\varphi \leq m$ and $0 \leq count(\varphi, v, l) \leq c$ for a $c$-edge-ranking $\varphi$ and a rank $i \in R$, the number of peer-lists $L_p(\varphi, v, l)$ is at most $m(c+1)^m$ for each vertex $v \in S_x = \{v_s, v_t\}$. For each peer-list $L_p(\varphi, v_s, l)$, there can be at most $m$ peer-lists $L_p(\varphi, v_t, l)$ for different $l$. The number of distinct peer-list-sets $\mathcal{L}_p(\varphi)$ can be at most $2m^2(c+1)^m$. Since by Lemma 4.1 we have $m \leq 1+2\Delta \log_{c+1} n = O(\log_{c+1} n)$, the total number of different feasible vectors on $x$ is $O(cn^{2\Delta} \log^2_{c+1} n)$.

It is also clear that the Lemmas 3.7 and 3.8 also hold for $c$-edge-ranking.

## 4.4 Parallel Algorithm

The main result of this chapter can be stated by the following theorem.

**Theorem 4.1** *For any positive integer $c$, an optimal $c$-edge-ranking of a series-parallel graph $G$ with $n$ vertices can be found on priority CRCW either in*

1. *time $O(\log_2 n)$ using $O(c^3 n^{6\Delta+1} \log^7_{c+1} n)$ operations; or*
2. *time $O(\log_2 n \log_2 \log_2 n)$ with $O(c^2 n^{4\Delta+1} \log^5_{c+1} n)$ operations.*

**Proof:** For the $c$-edge-ranking our algorithm works as in Procedure *c-VERTEX-RANKING* except the range of $m$ is $1 \leq m \leq 1 + 2\Delta \log_{c+1} n$.

The correctness of the algorithm for the $c$-edge-ranking is analogous to the $c$-vertex-ranking. Also the calculation for the complexity is almost similar. Since for $c$-edge-ranking the cardinality of the dominance class on each node is $O(cn^{2\Delta} \log^2_{c+1} n)$, the cardinality of the cross product is $O(c^2 n^{4\Delta} \log^4_{c+1} n)$. Since the length of a list is $O(\log_{c+1} n)$, by Lemma 3.8 the dominance class on $x$ can be computed either in $O(1)$ time using $O(c^3 n^{6\Delta} \log^7_{c+1} n)$ operations if the constant time algorithm is used, or in $O(\log_2 \log_2 n)$ time using $O(c^2 n^{4\Delta} \log^5_{c+1} n)$ operations if the doubly logarithmic algorithm is used. Finally the tree contraction takes $O(\log_2 n)$ time and $O(n)$ operations. So total complexity of the algorithm is either $O(\log_2 n)$ time using $O(c^3 n^{6\Delta+1} \log^7_{c+1} n)$ operations or $O(\log_2 n \log_2 \log_2 n)$ time using $O(c^2 n^{4\Delta+1} \log^5_{c+1} n)$ operations. The PRAM model used by the algorithm is priority CRCW.

## 4.5 Conclusion

In this chapter we have introduced our main result for $c$-edge-ranking. Before that we have introduced the necessary definitions and lemmas in new fashion suitable for $c$-edge-ranking from those for $c$-vertex-ranking in Chapter 4. With this chapter the main result of our thesis is completed.

# Chapter 5

# Conclusion

This thesis deals with the parallel algorithms for generalized vertex-ranking and edge-ranking for series-parallel graphs. Since series-parallel graphs are of great practical significance, we are glad to present such parallel algorithms.

In Chapter 1 we have introduced the ranking problems and its significance. In Chapter 2 we have characterized the $c$-vertex-ranking and $c$-edge-ranking of series-parallel graphs by the number of visible vertices. In Chapter 3 we have given the parallel algorithm for $c$-vertex-ranking of series-parallel graphs. In Chapter 4 we have given similar parallel algorithm for $c$-edge-ranking of series-parallel graphs.

In $c$-vertex(edge)-ranking when the positive integer $c$ is replaced by a function $f : \{1, 2, ..., n\} \rightarrow N$ such that, for any label $i$, deletion of all the vertices(edges) with labels $> i$ leaves connected components, each having at most $f(i)$ vertices(edges) with label $i$ then it is called $f$-vertex(edge)-ranking. Our parallel algorithms can be extended for $f$-vertex-ranking and $f$-edge-ranking of series-parallel graphs with slight modification. The complexity for the $f$-vertex(edge)-ranking would be similar to the $c$-vertex(edge)-ranking for series-parallel graphs. It is possible to give a parallel algorithm for $f$-vertex-ranking of series-parallel graphs that runs in $O(\log_2 n)$

time using $O(n^{10} \log_2^{13} n)$ operations, where $n$ is the number of vertices in $G$. Also an algorithm can be given for $f$-vertex-ranking of series-parallel graphs that runs in $O(\log_2 n \log_2 \log_2 n)$ time using $O(n^7 \log_2^9 n)$ operations. For the $f$-edge-ranking of series-parallel graphs an algorithm can be given that runs in $O(\log_2 n)$ time using $O(n^{6\Delta+4} \log_2^7 n)$ operations, where $\Delta$ is the maximum vertex-degree of $G$. Also an algorithm can be given for $f$-edge-ranking of series-parallel graphs that runs in $O(\log_2 n \log_2 \log_2 n)$ time using $O(n^{4\Delta+3} \log_2^5 n)$ operations.

Finally, some open problems can be pointed out from our work as follows:

(1) To find the optimal parallel algorithm for $c$-vertex(edge)-ranking of series-parallel graphs.

(2) What is the complexity of parallel algorithms for $c$-vertex(edge)-ranking of other classes of graphs

(3) What is the complexity of parallel algorithms for $c$-vertex(edge)-ranking of perfect graphs specially permutation graphs and chordal graphs.

# Bibliography

[ADK+89] K. Abrahamson, N. Dadoun, D.G. Kircpatrick, and T. Przytycka, A Simple Parallel Tree Contraction Algorithm, *Journal of Algorithm,* 10(1989), pp. 287-302.

[DR83] I.S. Duff and J.K. Reid, The Multifrontal Solution of Indefinite Sparse Symmetric Linear equations, *ACM Transactions on Mathematical Software,* 9(1983), pp. 302-325.

[GMT88] H. Gazit, G.L. Miller, and S.H. Teng, Optimal Tree Contraction in an EREW Model, *Concurrent Computations: Algorithms: Architecture and Technology* (S.K. Tewkesbury, B.W. Dicknson, and S.C. Schwartz, Eds.) Plenum, New York,(1998).

[He91] X. He, Efficient Parallel Algorithms for Series-parallel Graphs, *Journal of Algorithms,* 12(1991), pp. 409-430.

[HY88] X. He and Y. Yesha, Binary Tree Algebraic Computations and Parallel Algorithm for Simple Graphs, *Journal of Algorithms,* 9(1988), pp. 92-113.

[IRV88] A.V. Iyer, H.D. Ratliff, and G. Vijayan, Optimal Node Ranking of Trees, *Information Processing Letters,* 28(1988), pp. 225-229.

[IRV91] A.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.

[Já92] Josjep JáJá, An Introduction to Parallel Algorithms, *Addison-Wesley,* 1992.

[K01] M.A. Kashem, A Separator Theorem of Trees, *Manuscript,* 2001.

[KR99] Md. Abul Kashem and M.Z. Rahman, An Efficient Algorithm for Optimal $c$-Vertex-Ranking for Series-Parallel Graphs, *International Conference on Computer and Information Technology,* (1999), pp. 309-314.

[KZN97] M.A. Kashem, X. Zhou, and T. Nishizeki, Generalized Edge-Ranking of Trees, *Proc. of the 22nd International Workshop on Graph-Theoretical Concepts in Computer Science (WG'96), Lecture Notes in Computer Science, Springer-Verlag,* 1197(1997), pp. 390-404.

[KZN00] M.A. Kashem, X. Zhou, and T. Nishizeki, Algorithms for Generalized Vertex-Ranking of Partial $k$-Trees, *Theoretical Computer Science,* 240(2000), pp. 407-427.

[KMS95] M. Katchalski, W. McCuaig, and S. Seager, Ordered Colorings, *Discrete Mathematics,* 142(1995), pp. 141-154.

[KMW96] T. Kloks, H. Müller, and C.K. Wong, Vertex Ranking of Asteroidal Triple-free Graphs, *Proc. of 7th International Symposium on Algorithms and Computation (ISAAC'96), Lecture Notes in Computer Science, Springer-Verlag,* 1178(1996). pp. 174-182.

[KD88] S.R. Kopsaraju and A.L. Delcher, Optimal Parallel Evaluation of Tree-structured Computation y Ranking, *VLSI Algorithms and Architectures, 3rd Aegean Workshop on Computing, Lecture Notes in Computer Science, Springer-Verlag,* 319(1988), pp. 101-110.

[Lui90] J.W.H. Lui, The Role of Elimination Trees in Sparse Factorization, *SIAM Journal of Matrix Analysis and Application,* 7(1990), pp. 134-172.

[MR85] G.L. Miller and J.H. Reif, Parallel Tree Contraction and Its Application. *Proceedings of the 26th IEEE Symposium on Foundations of Computer Science,* (1985), pp. 478-489.

[RK01] M.Z. Rahman and Md. Abul Kashem, An Optimal Parallel Algorithm for $c$-Vertex-Ranking of Trees, *International Conference on Electrical and Computer Engineering,* (2001), pp. 277-280.

[Sch89] A.A. Schäffer, Optimal Node Ranking of Trees in Linear Time, *Information Processing Letters,* 33(1989/90), pp. 91-96.

[SDG92] A. Sen, H. Deng, and S. Guha, On a Graph Partition Problem with Application to VLSI Layout, *Information Processing Letters,* 43(1992), pp. 87-94.

[TNS82] K. Takamizawa, T. nishizeki, and N. Saito, Linear Time Computability of Combinatorial Problems on Series-Parallel Graphs, *Journal of Associated Computational Mathematics,* 29(1982), pp. 623-641.

[ZNN95] X. Zhou, N. Nagai, and T. Nishizeki, Generalized Vertex-Ranking of Trees, *Information Processing Letters,* 56(1995), pp. 321-328.

# Index

*c*-edge-ranking, 24
*c*-vertex-ranking, 5, 23
*c*-vertex-ranking number, 5, 27
*f*-vertex-ranking, 6
1-edge-ranking, 7
1-vertex-ranking, 4, 5

binary decomposition tree, 21

concurrent read concurrent write, 11
concurrent read exclusive write, 11

dominance class, 31

edge-coloring, 2
edge-ranking, 3, 7, 12
edge-ranking number, 8
equivalence class, 28
exclusive read exclusive write, 11

graph, 1, 16

optimal vertex-ranking, 4

parallel random-access machine, 10
partial *k*-tree, 19

random access machine, 9

series-parallel graph, 26
series-sarallel graphs, 20
shared memory model, 9, 10
simple graph, 16
subgraph, 17

vertex-coloring, 2
vertex-ranking, 3, 4, 12
vertex-ranking number, 4
visible edge, 24
visible vertices, 23