# An Efficient Algorithm for Edge-Ranking of Series-Parallel Graphs

by

Md. Zakir Hossain

Department of Computer Science and Engineering
Bangladesh University of Engineering & Technology
Dhaka – 1000, Bangladesh.

1999

Submitted to Bangladesh University of Engineering and Technology
in partial fulfillment of the requirements for M. Sc. Engg. (Computer Science & Engineering).

# An Efficient Algorithm for Edge-Ranking of Series-Parallel Graphs

A thesis submitted by

**MD. ZAKIR HOSSAIN**
Roll No. 911806P, Registration No. 80059,
for the partial fulfillment of the degree of
M. Sc. Engg. in Computer Science & Engineering.
Examination held on: December 23, 1999.

Approved as to style and contents by:

[illegible] 23/12/99

DR. MD. ABUL KASHEM MIA — Chairman and Supervisor
Assistant Professor,
Department of Computer Science & Engineering
B. U. E. T., Dhaka – 1000, Bangladesh.

[illegible] 23/12/99

DR. CHOWDHURY MOFIZUR RAHMAN — Member
Head,
Department of Computer Science & Engineering
B. U. E. T., Dhaka – 1000, Bangladesh.

Saidur 23/12/99

DR. MD. SAIDUR RAHMAN — Member
Assistant Professor,
Department of Computer Science & Engineering
B. U. E. T., Dhaka – 1000, Bangladesh.

[illegible] 23\12\99

DR. PRAN KANAI SAHA — Member (External)
Professor,
Department of Electrical and Electronic Engineering
B. U. E. T., Dhaka – 1000, Bangladesh.

# 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 or for publication.

[illegible] 23/12/99

Signature of the
Supervisor

[illegible]
23-12-99

Signature of the
Author

# Contents

# Acknowledgements

I wish to express my profound gratitude to Dr. Md. Abul Kashem Mia for his constant supervision, valuable advice, continual encouragement and constructive criticism without which I might not have made it through the thesis.

I would like to express my utmost appreciation to professor Dr. Md. Kaykobad for his strong recommendation for time extension and many other things without which it was not possible to complete this thesis.

I would like to express my greatest thanks to my friends Dr. Joarder Kamruzzaman and Md. Latiful Haque, who have encouraged me to start the work of the thesis after a long time.

I would also like to thank all my friends and colleagues for their emotional supports.

Special thanks go to Syed Ashraf Ali, Sub-Divisional Engineer of DESA, for his continual inspiration and supports.

Finally, I would like to thank my wife Monira Nargis for her encouragement and supports during the work of the thesis.

# Abstract

This thesis presents an efficient algorithm for finding generalized edge-ranking of series-parallel graphs. A generalized edge-ranking of a graph $G$ is defined as follows: for a positive integer $c$, a $c$-edge-ranking of $G$ is a labeling (ranking) of the edges of $G$ with 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 $i$. A $c$-edge-ranking is optimal if the number of labels used is as small as possible. The $c$-edge-ranking problem is to find an optimal $c$-edge-ranking of a given graph. The $c$-edge-ranking problem has applications in scheduling the complex multi-parts products from its components; it is equivalent to finding a $c$-edge-separator tree of $G$ having the minimum height. The $c$-edge-separator tree provides a parallel computation scheme having the minimum computation time. We give an efficient and simple polynomial-time algorithm for solving the $c$-edge-ranking problem of series-parallel graphs.

# Chapter 1

# Introduction

In this chapter we provide the necessary background and motivation for this study on the edge-ranking of series-parallel graphs. We start by giving a historical background of the development of edge-ranking of graphs. We then give the definition of ordinary edge-ranking and generalized edge-ranking of graphs. Finally, we summarize our new results togather with known ones.

## 1.1 Backgrounds

Recent research efforts in algorithm theory have concentrated on designing efficient algorithms for solving combinatorial problems, particularly graph problems. A graph

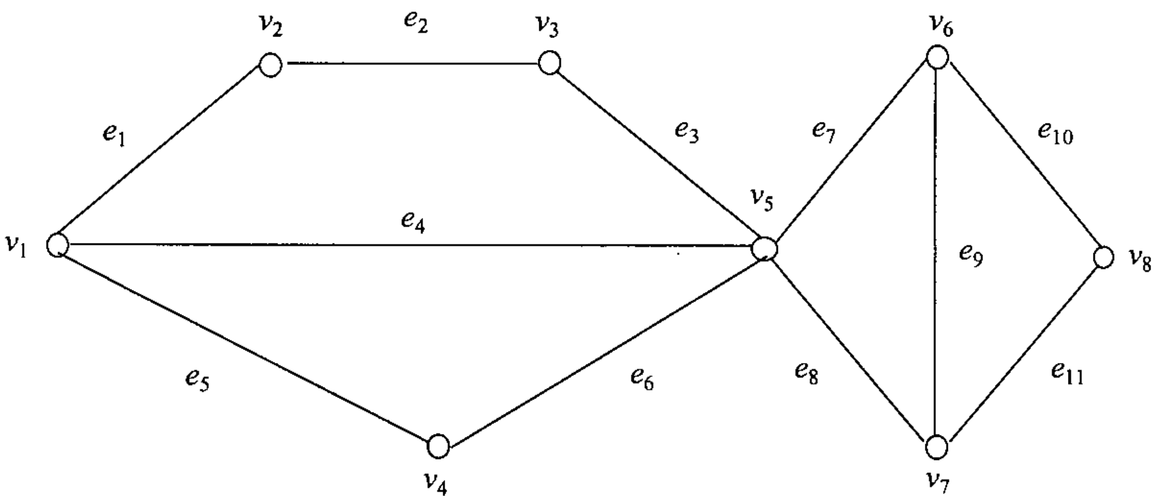


Figure 1.1: A graph with eight vertices and eleven edges.

$G = (V, E)$ with $n$ vertices and $m$ edges consists of a vertex set $V = \{v_1, v_2, v_3, \dots, v_n\}$ and an edge set $E = \{e_1, e_2, e_3, \dots, e_m\}$, where an edge in $E$ joins two vertices in $V$. Figure 1.1 depicts a graph of eight vertices and eleven edges, where vertices are drawn by circles, edges by lines, vertex names next to the circles and edge names next to the lines. Efficient algorithms have been obtained for various graph problems, such as coloring problems, planarity testing problem, maximum flow problem.

The vertex-coloring and the edge-coloring problems are two of the fundamental problems on graph. 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. 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[24]. The ordinary vertex-ranking and edge-ranking problems are restrictions of the vertex-coloring problem and the edge-coloring problem, respectively.

## 1.2 Generalized Edge-Ranking Problem

An ordinary edge-ranking of a given graph $G$ is a labeling of edges of $G$ with positive integers such that every path between two edges with the same label $i$ contains an edge with label $j > i$ [10]. Clearly an edge-labeling is an 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 minimum number of ranks needed for an edge-ranking of $G$ is called the *edge-ranking number* of $G$. An edge-ranking of $G$ using the 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 ordinary edge-ranking problem imply that two adjacent edge cannot have the same rank. Thus the ordinary edge-ranking problem is a restriction of the edge-coloring problem. Figure 1.2 depicts an optimal edge-ranking of the graph $G$ in Figure 1.1 using seven ranks, where the ranks are drawn next to the edges.

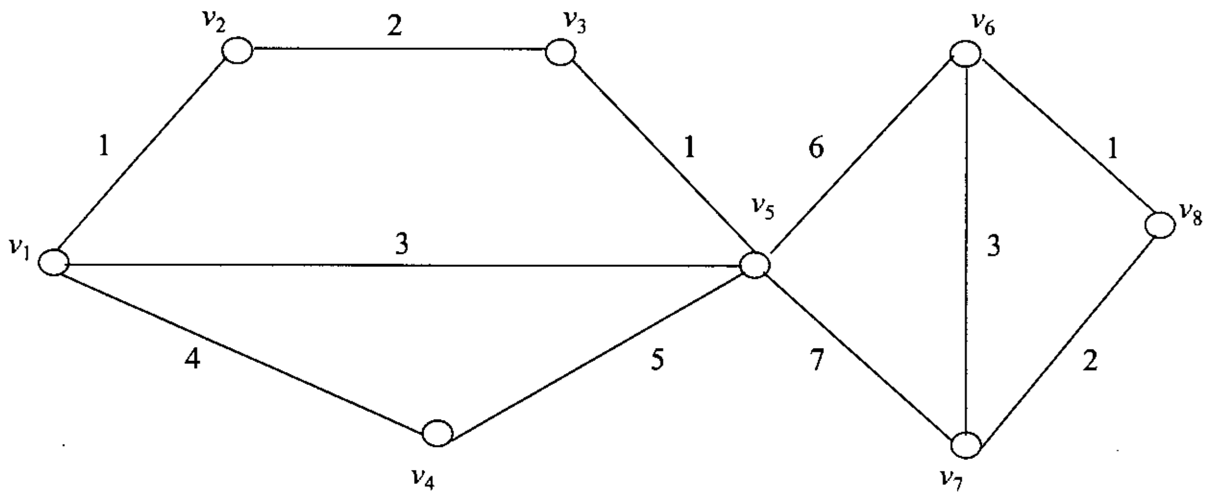


Figure. 1.2 : An optimal edge-ranking of the graph $G$ of Fig. 1.1

A natural generalization of an ordinary edge-ranking is the *c-edge-ranking*. For any positive integer $c$, a $c$-edge-ranking of a 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 $i$ [22]. Clearly an ordinary

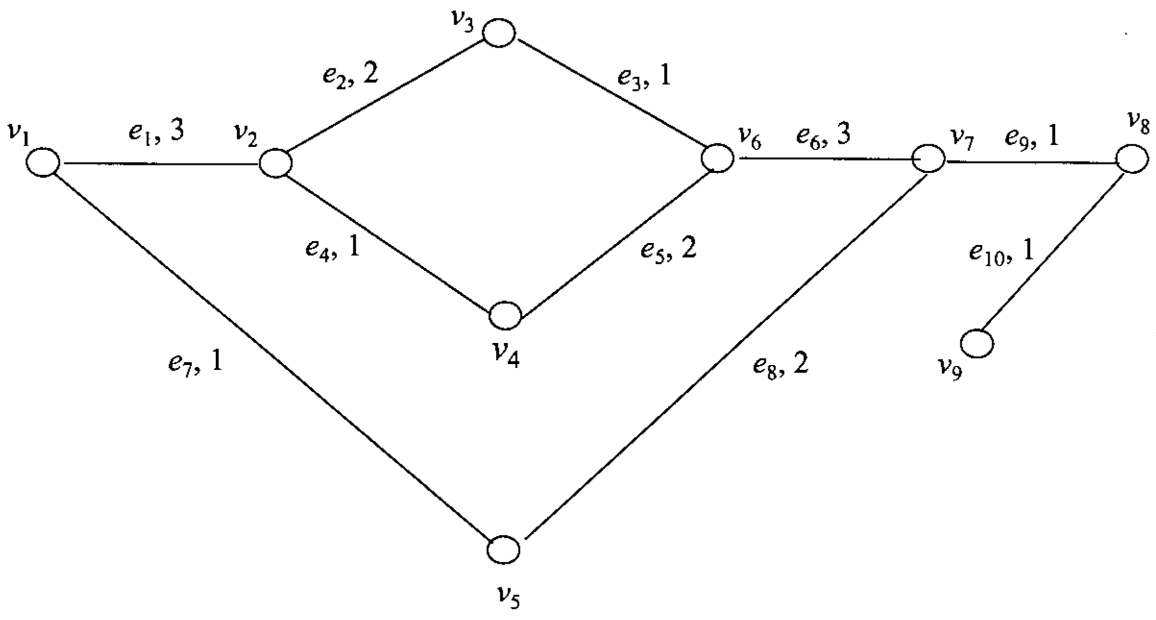


Figure 1.3 : An optimal 2-edge-ranking of a graph $G$.

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* of $G$ and is denoted by $r_c'(G)$. A $c$-edge-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 ranks are drawn next to the edge numbers. Connected components obtained from $G$ by deleting all edges with labels $> i$ for the 2-edge-ranking of the graph in Fig. 1.3 are drawn in ovals in Fig. 1.4.

The problem of finding an optimal $c$-edge-ranking of a graph has applications in scheduling the parallel assembly of a complex multipart product from its components, where the vertices of $G$ corresponds to the basic components and the edges corresponds to assembly operations to be performed between the components [10]. 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

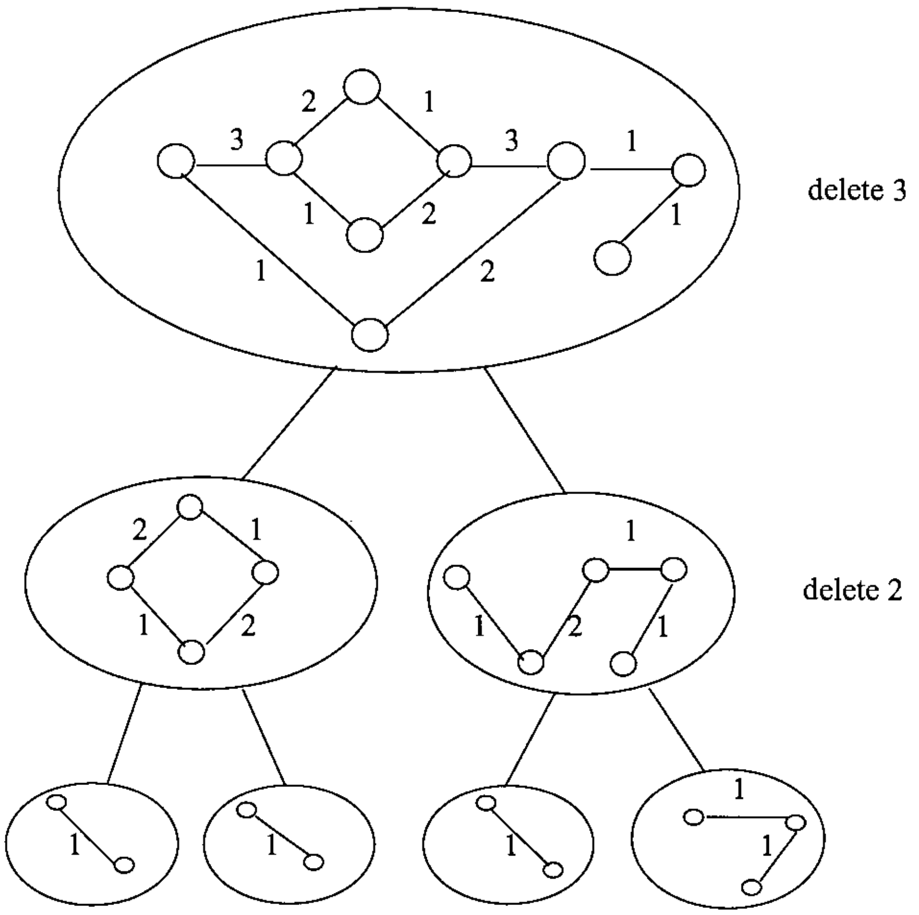


Figure 1.4 (a): A 2-edge-separator tree of the graph in Fig. 1.3.

required for the parallel assembly of a product using the robots is equivalent to finding an optimal $c$-edge-ranking of the graph $G$.

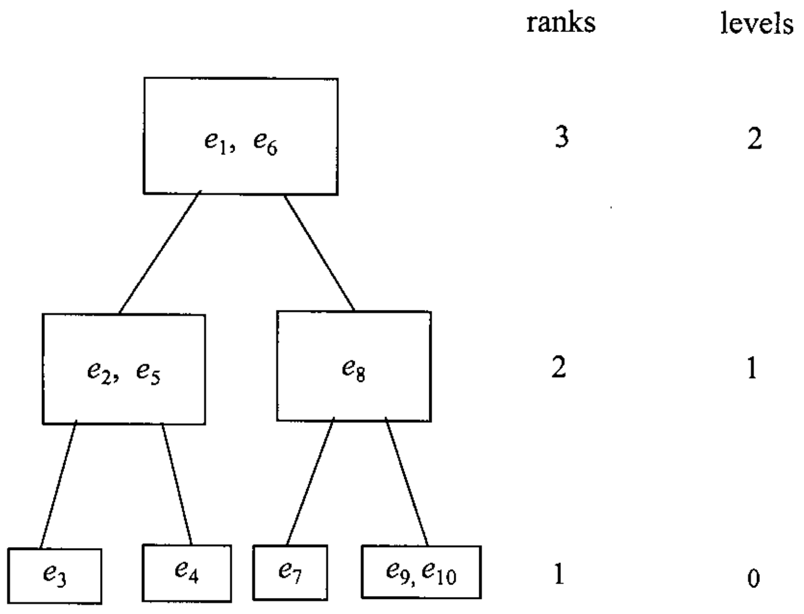


Figure 1.4 (b): A 2-edge-separator tree of the graph of Fig. 1.3.

The $c$-edge-ranking problem of a graph $G$ is also equivalent of 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 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 [16]. Figure 1.4(a) and 1.4(b) illustrates a 2-edge-separator tree of the graph $G$ depicted in Fig. 1.3.

We next review the results on the edge-ranking problem. The problem of finding an optimal edge-ranking was first studied by Iyer *et al.* in 1991 as they found that the problem has an application in scheduling the parallel assembly of multipart products. They gave an $O(n \log_2 n)$ time approximation algorithm for an edge-ranking of trees $T$ using at most twice the minimum number of ranks, where $n$ is the number of vertices in $T$ [10]. Later Lam and Yue have proved that the edge-ranking problem is NP-hard for graphs in general [13]. Zhou *et al.* gave an $O(n^2 \log_2 \Delta)$ time algorithm to solve the $c$-edge-ranking problems on trees for any positive integer $c$, where $\Delta$ is the maximum degree of the tree $T$ [22]. Recently Lam and Yue presented a linear-time algorithm to solve the edge-ranking problem for trees [14]. On the other hand, Kashem *et al.* have obtained a polynomial-time algorithm for finding an optimal $c$-edge-ranking of a given partial $k$-tree with bounded maximum degree for any positive integer $c$ [12]. Since the partial 2-trees are series-parallel graphs [2, 24], their algorithm yields a polynomial-time algorithm for series-parallel graphs. The time complexity of their algorithm is $O(n^{18\Delta+8} \log_2^8 n)$.

## 1.3 Summary

This thesis explores the generalized edge-ranking problem of series-parallel graphs. Our main results can be devided into two parts.

The first part of the results is about the upper bound on the $c$-edge-ranking number $r_c'(G)$ of series-parallel graph $G$. We show that $r_c'(G) = O(\log_{c+1} n)$ for a series-parellel graph $G$ with bounded maximum degree, where $n$ is the number of vertices in $G$ and $c$ is any positive integer.

The second part of the results is to give an efficient polynomial-time sequential algorithm for finding an optimal $c$-edge-ranking of series-parallel graphs with bounded maximum degree for any positive integer $c$. The time complexity of our algorithm is $O(n^{4\Delta+4} \log_{c+1}{}^4 n \log_2 \log_{c+1} n)$. Kashem *et al.* have given a polynomial-time algorithm for finding an optimal $c$-edge-ranking of a given partial $k$-tree with bounded maximum degree for any positive integer $c$. Since the partial 2-trees are series-parallel graphs, their algorithm yields a polynomial-time algorithm for series-parallel graphs. The time complexity of their algorithm is $O(n^{18\Delta+8} \log_2{}^8 n)$.

Our new result togather with known ones are listed in Table 1.1.

| Classes of graphs | Sequential time | Comment on $c$ | Reference |
|---|---|---|---|
| Trees | $O(n^2 \log_2 \Delta)$ | any positive integer | [22] |
| Trees | $O(n)$ | $c = 1$ | [14] |
| Partial $k$-trees with bounded degrees | $O(n^{2\Delta(k+1)^2+2(k+1)+2} \log_2^{k(k+1)+2} n)$ | any positive integer | [12] |
| Series-Parallel Graphs with bounded degrees | $O(n^{18\Delta+8} \log_2{}^8 n)$ | any positive integer | [12] |
| Series-Parallel Graphs with bounded degrees | $O(n^{4\Delta+4} \log_{c+1}{}^4 n \log_2 \log_{c+1} n)$ | any positive integer | Ours |

Table 1.1: Algorithms of the $c$-edge-ranking.

This thesis is organised as follows. Chapter 2 gives preliminaries and characterize the $c$-edge-ranking of series-parallel graphs by "visible edges". Chapter 3 presents an efficient polynomial-time algorithm for finding an optimal $c$-edge-ranking of a given series-parallel graph $G$. Finally, Chapter 4 concludes with a discussion of future works.

# Chapter 2

# Preliminaries

In this chapter we define some basic terms and present characterization of $c$-edge-ranking of series-parallel graphs by "visible edges". Definitions which are not included in this chapter will be introduced as they are needed. We start, in Section 2.1, by giving some definitions of the standard graph theoritical terms used throughout the remainder of the thesis. In Section 2.2 we give the definition and characteristics of series-parallel graphs. In Section 2.3 we define some basic terms used for finding optimal $c$-edge-ranking and characterize the $c$-edge-ranking of series-parallel graphs by the number of visible edges.

## 2.1 Basic Terminology

### 2.1.1 Graphs and Multigraphs

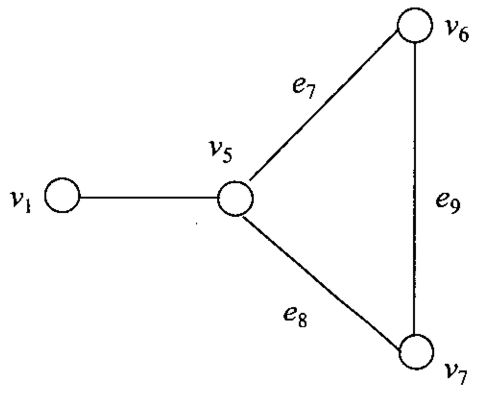
A graph $G = (V, E)$ is a structure which consists of a finite set of *vertices* $V$ and a finite set of *edges* $E$; each edge is an unordered pair of distinct vertices (see Figure 1.1). We call $V(G)$ the vertex-set of the graph $G$ and $E(G)$ the edge-set of $G$. Throughout this thesis the number of vertices is denoted by $n$, that is, $n = |V|$ and the number of edges of $G$ is denoted by $m$, that is, $m = |E|$. 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 a *neighbour* 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 a *multigraph*. Sometimes a simple graph is simply called by a graph.

### 2.1.2 Degree of a Vertex

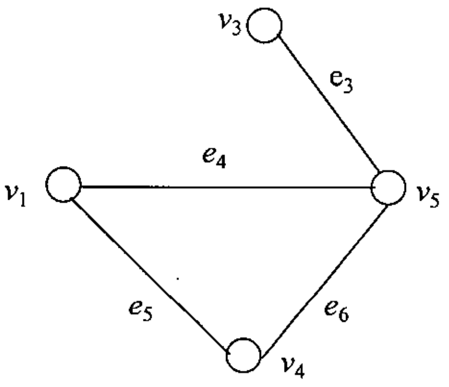
The *degree* of a vertex $v$ in a graph $G$ is the number of edges incident to $v$ and is denoted by $d_G(v)$ or simply by $d(v)$. The maximum degree of $G$ is denoted by $\Delta(G)$ or simply by $\Delta$. A vertex of degree 0 is called an *isolated vertex.*

## 2.1.3 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 $G[V']$. $V'$ consist 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 $G[E']$. Figure 2.1(a) depicts a subgraph of $G$ in fig 1.1 induced by $\{v_1, v_5, v_6, v_7\}$ and fig 2.1(b) depicts a subgraph induced by $\{e_3, e_4, e_5, e_6\}$.



(a)



(b)

Figure 2.1: Subgraph of $G$ in Fig. 1.1. (a) vertex induced subgraph. (b) edge induced subgraph.

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

## 2.1.4 Paths and Cycles

A $v_0 - v_l$ walk in $G$ is an alternating sequence of vertices and edges of $G$, $v_0$, $e_1$, $v_1$, $e_2$, … , $v_{l-1}$, $e_l$, $v_l$ beginning and ending with a vertex, in which each edge is incident to two vertices immediately preceding and following it. If the vertices $v_0$, $v_1$, $v_2$, … , $v_l$ are distinct (except possibly $v_0$, $v_l$), then the walk is called a path and usually denoted by $v_0$, $v_1$, $v_2$, … , $v_l$. The length of the path is $l$, one less than the number of vertices on the path. A path or walk is closed if $v_0 = v_l$. A closed path at least one edge is called a cycle.

## 2.1.5 Connected Components and Separators

A graph is connected if for every pair $\{u, v\}$ of distinct vertices there is a path between $u$ and $v$. A (connected) component of a graph is a maximal connected subgraph. A graph which is not connected is called a disconnected graph. A separator of a connected graph $G$ is a set of vertices whose deletion disconnects $G$.

## 2.2 Series-Parallel Graphs

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

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 denoted by $v_s(G)$ and $v_t(G)$.

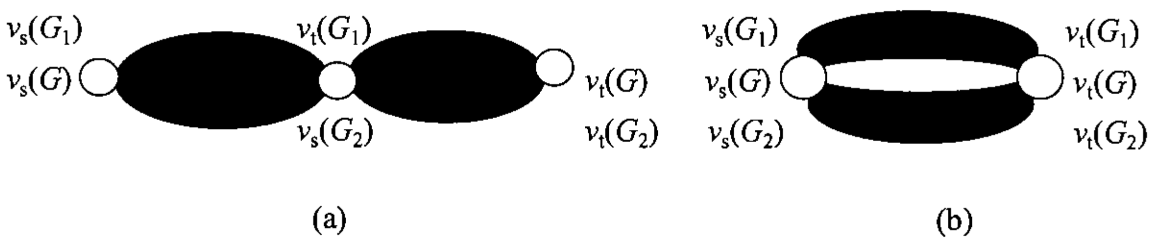


Figure 2.2 : (a) Series and (b) parallel connections.

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 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 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 $G = G_1 \bullet G_2$. (See Fig. 2.2 )

   (b) A graph $G$ obtained from $G_1$ and $G_2$ by identifying vertex $v_s(G_1)$ with vertex $v_s(G_2)$ and vertex $v_t(G_1)$ with vertex $v_t(G_2)$ is 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 $G = G_1 \mid\mid G_2$. (See Fig. 2.2)

Thus a series-parallel graph (partial 2-tree) is a simple graph without multiple edges or self loops.

A series-parallel graph $G$ can be represented by a "binary decomposition tree" $T_b$ [24]. Figure 2.3 illustrates a binary decomposition tree $T_b$ of the graph in Figure 1.3. 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-nodes* and *p-nodes*, respectively. A node $x$ of tree $T_b$ corresponds to a subgraph of $G$, which is denoted by $G_x$. Especially, every *leaf* of $T_b$ represents a subgraph of $G$ induced by an edge $e \in E$. Thus the root of $T_b$ represents the graph $G = (V, E)$.

$G_{x,}$ a subgraph of $G$ associated with a node $x$ in $T_b$, is also a series-parallel graph with two terminals $v_s(G_x)$ and $v_t(G_x)$. Let $S_x$ be the set of two terminals of the series-parallel graph $G_x$,*i.e.*, $S_x = \{v_s(G_x), v_t(G_x)\}$. Every leaf $x$ of $T_b$ represents a subgraph of $G$ induced by two vertices connected by an edge $e_x$. We associate a subgraph $G_x = (V_x, E_x)$ with each node $x$ of tree $T_b$, where

$$V_x = \bigcup \left\{ S_y \mid y = y \text{ or } z \text{ is a descendant of } x \text{ in } T_b \right\}; \text{ and}$$

$$E_x = \begin{cases} e_x \in E, & \text{if } x \text{ is a leaf node in } T_b, \text{ and} \\ E_y \bigcup E_z, & \text{if } x \text{ is an internal node having two childreen } y \text{ and } z. \end{cases}$$

If $x$ is an internal node in $T_b$ having two children $y$ and $z$, then the two edge-sets $E_y$ and $E_z$ are disjoint. The subgraph $G_x$ is an edge-disjoint union of two subgraphs $G_y$ and $G_z$.. If $x$ is a series node, then $G_y$ and $G_z$ is connected in $G_x$ via the common vertex $v$ of $G_y$ and $G_z$, that is, $v = S_y \cap S_z$ through a series connection. On the otherhand, if $x$ is a parallel node, then $G_y$ and $G_z$ is connected in $G_x$ via both the terminals of $G_y$ and $G_z$, that is, $v \in S_y \cup S_z$ through a parallel connection.

## 2.3 Visible Edges

Let $\varphi$ be an edge-labeling of a graph $G = (V, E)$ with positive integers. The *label* (rank) of an edge $e \in E$ is represented by $\varphi(e)$. The number of ranks used by an edge-labeling $\varphi$ is denoted by $\#\varphi$. One may assume without loss of generality that $\varphi$ uses consecutive integers $1, 2, 3, \dots, \#\varphi$ as the ranks.

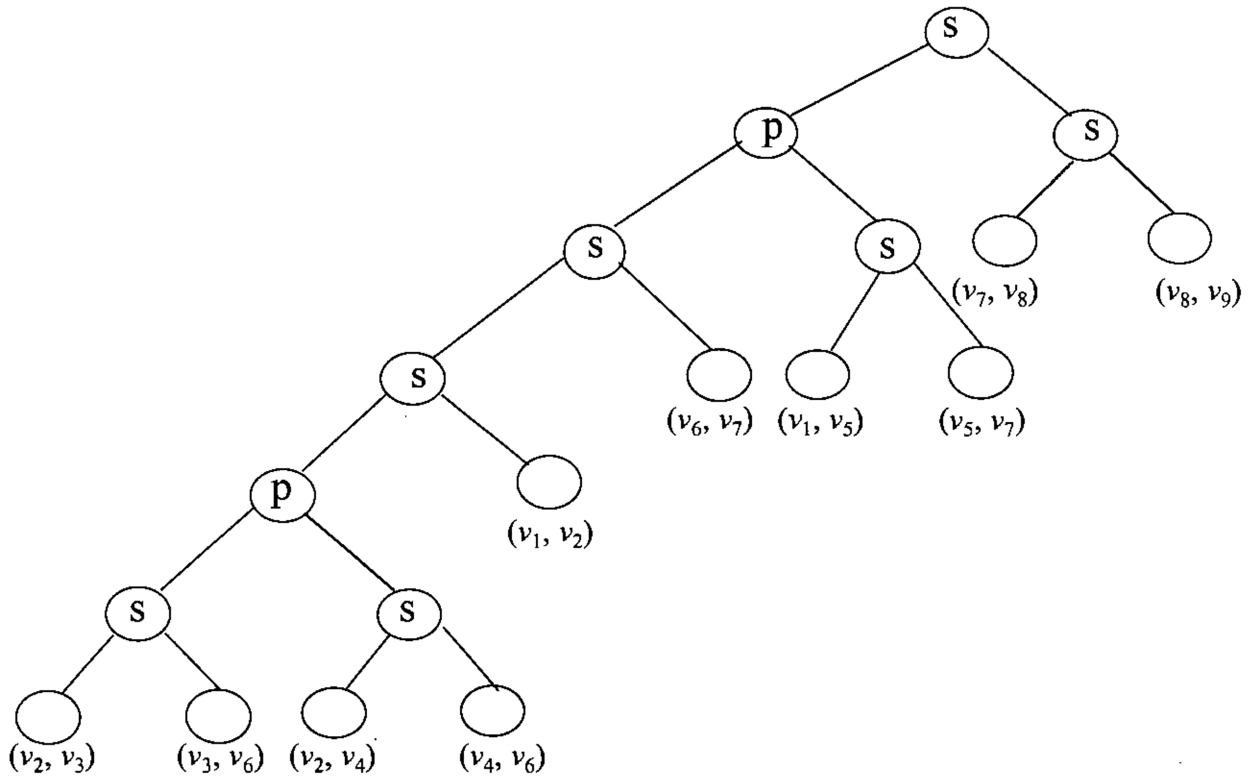


Figure 2.3: A binary decomposition tree of the graph $G$ in Figure 1.3.

For a rank $i$, $1 \leq i \leq \#\varphi$, we denote by $E(G, \varphi, i)$ the set of edges $e$ in $G$ with $\varphi(e) = i$, and let $m(G, \varphi, i) = |(E, \varphi, i)|$. Then $\varphi$ is a $c$-edge-ranking of $G$ if and only if $m(D, \varphi, i) \leq c$ for any $i$, $1 \leq i \leq \#\varphi$ and any connected component $D$ of the graph obtained from $G$ by deleting all edges with ranks $> i$. For a subgraph $G' = (V', E')$ of $G$, we denote by $\varphi|G'$ a *restriction* of $\varphi$ to $G$. Let $\eta = \varphi| G'$, then $\eta(e) = \varphi(e)$ for $e \in E'$ We characterize the $c$-edge-ranking of a series-parallel graph by the number of "visible" edges. An edge $e \in E$ is said to be *visible* from a vertex $v \in V$ under $\varphi$ in $G$ if $G$ has a path from $e$ to $v$ every edge of which has a rank $\leq \varphi(e)$. The rank $\varphi(e)$ of $e$ is also said to be visible from $v$ under $\varphi$ in $G$ if the edge $e$ is so. We then have the following lemma which characterize the $c$-edge-ranking of series-parallel graphs by the number of visible edges.

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

*(a) At most c edges of the same rank are visible from any vertex $v \in S_y \cup S_z$ under $\varphi$ in $G_x$; and*

*(b) If x is an internal node in $T_b$ and has two children y and z, then $\varphi|G_y$ and $\varphi|G_z$ are c-edge-rankings of $G_y$ and $G_z$, respectively.*

**Proof :** $\Rightarrow$: Suppose that $\varphi$ is a $c$-edge-ranking of $G_x$. Then, for any label $i$, deletion of all edges from $G_x$ with labels $> i$ leaves connected components, each having at most $c$ edges with label $i$.

(a) Let $i$ be any rank. Delete all edges with labels $> i$ from $G_x$. Among the connected components of the remaining graph, let $D$ be the one containing a vertex $v \in S_y \cup S_z$. Then exactly $m(D, \varphi, i)$ edges with rank $i$ are visible from $v$ under $\varphi$ in $G_x$. Since $\varphi$ is a $c$-edge-ranking of $G_x$, we have $m(D, \varphi, i) \leq c$. Therefore, at most $c$ edges of rank $i$ are visible from $v$ under $\varphi$ in $G_x$.

(b) Assume that $x$ is an internal node of $T_b$ and has two children $y$ and $z$. Let $\varphi|G_y$ and $\varphi|G_z$ be the restriction of $\varphi$ to $G_y$ and $G_z$, respectively. Since $\varphi$ is a $c$-edge-ranking of $G_x$ and $G_y$ is a subgraph of $G_x$, for any label $i$, deletion of all edges from $G_y$ with labels $> i$ leaves connected components, each having at most $c$ edges with label $i$. Therefore, $\varphi|G_y$ is a $c$-edge-ranking of $G_y$. Similarly $\varphi|G_z$ is a $c$-edge-ranking of $G_z$.

$\Leftarrow$: Suppose for a contradiction that an edge-labeling $\varphi$ satisfies (a) and (b), but $\varphi$ is not a $c$-edge-ranking of $G_x$. Then, there exists a rank $i$ such that deletion of all edges with labels $> i$ from $G_x$ leaves a connected component $D$ such that $m(D, \varphi, i) > c$. Since (a) and (b) hold, $x$ is an internal node of $T_b$ and $D$ is neither a subgraph of $G_y$ nor a subgraph of $G_z$. Furthermore, $G_y$ and $G_z$ have common vertices only in $S_y \cup S_z$. Therefore, $D$ has a vertex $v \in S_y \cup S_z$. Then all $m(D, \varphi, i)$ edges with label $i$ in $D$ are visible from $v$ in $G_x$. Therefore, more than $c$ edges of rank $i$ are visible from $v$ under $\varphi$ in $G_x$, contradictory to (a).

# Chapter 3

# Genaralized Edge-Ranking of Series-Parallel Graphs

This chapter deals with the generalized edge-ranking problem on series-parallel graphs. Lam and Yue have proved that the edge-ranking problem is NP-hard for graphs in general [13]. The edge-ranking problem is one of the few problems for which no efficient algorithms have been obtained except for few class of graphs. Zhou *et al.* gave an $O(n^2 \log_2 \Delta)$ time algorithm to solve the $c$-edge-ranking problem on trees $T$ for any positive integer $c$, where $\Delta$ is the maximum degree of the tree $T$ [22]. Recently Lam and Yue presented a linear-time algorithm to solve the edge-ranking problem for trees [14]. On the other hand, Kashem *et al.* have obtained a polynomial-time algorithm for finding an optimal $c$-edge-ranking of a given partial $k$-tree with bounded maximum degree for any positive integer $c$ [12]. Since the partial 2-trees are series-parallel graphs [2, 24], their algorithm yields a polynomial-time algorithm for series-parallel graphs. The time complexity of their algorithm is $O(n^{18\Delta+8} \log_2^{8} n)$.

In this chapter we give an efficient algorithm to solve the $c$-edge-ranking problem of series-parallel graphs. Our algorithm runs in time $O(n^{4\Delta+4} \log_{c+1}^{4} n \log_2 \log_{c+1} n)$. We use dynamic programming and bottom-up tree computation on the binary decomposition tree

$T_b$ of the graph $G = (V, E)$ from leaf to the root. For each node $x$ of $T_b$ from leaves to the root, we construct all (dominance classes of) $c$-edge-ranking of $G_x$ from those of two subgraphs $G_y$ and $G_z$ associated with the children $y$ and $z$ of $x$.

The remainder of this chapter is organized as follows. Section 3.1 gives some definitions and gives an upper bound on the $c$-edge-ranking number of a series-parallel graph. Sections 3.2, 3.3 and 3.4 define an equivalence class, a peer class and a dominance class, respectively to solve the $c$-edge-ranking problem of series-parallel graphs. Section 3.5 gives an efficient algorithm, verifies the correctness of the algorithm and analyze its time-complexity. Finally Section 3.6 includes a brief conclusion.

## 3.1 Preliminaries

### 3.1.1 Definitions

In this section we give some definitions.

We define the lexicographical order on the set of non-increasing sequences (lists) of positive integers as follows: Let $A = \{a_1, a_2, \dots, a_p\}$ and $B = \{b_1, b_2, \dots, b_q\}$ be two sets (lists) of positive integer such that $a_1 \geq a_2 \geq \dots \geq a_p$ and $b_1 \geq b_2 \geq \dots \geq b_q$, then $A \prec B$ if there exist an integer $i$ such that

(a) $a_j = b_j$, for all $1 \leq j < i$; and

(b) either $a_i < b_i$ or $p < i \leq q$.

We write $A \preceq B$ if $A = B$ or $A \prec B$.

For a list $A$ and an integer $\alpha$, we define a sublist $[\alpha \le A]$ of $A$ as follows:

$$[\alpha \le A] = \{x \in A \mid \alpha \le x\}.$$

Similarly we define sublists $[\alpha < A]$, $[A \le \alpha]$ and $[A < \alpha]$ of $A$. Obviously, if $A \preceq B$ then $[\alpha < A] \preceq [\alpha < B]$ for any $\alpha \ge 1$. For list $L$ and $L'$ we use $L \subseteq L'$ and $L \cup L'$ in their usual meaning in which we regard $L$, $L'$ and $L \cup L'$ as multi-sets.

## 3.1.2 Upper Bound on the *c*-Edge-Ranking Number of a Series-Parallel Graph

In this section we show that the $c$-edge-ranking number $r_c'(G)$ of a series-parallel graph $G$ is $O(\log_{c+1} n)$. We first cite Lemma 3.1 from [23 ].

**Lemma 3.1** *Let $T$ be a tree of $n_T$ $(\ge 1)$ nodes, and let $\alpha$ be any positive integer. Then $T$ has at most $\alpha$ nodes whose removal leaves subtrees each having at most $n_T/q$ nodes, where*

$$q = 2^{\lfloor \log_2(\alpha+3) \rfloor - 1} > \frac{\alpha+3}{4}$$

and $q \ge 2$.

We then have the following lemma on $r_c'(G)$.

**Lemma 3.2** *Let c be any positive integer which is not always bounded, let G be a series-parallel graph of n vertices, and let Δ be the maximum degree of G. Then*

$$r_c'(G) \le 1 + b \log_{c+1} n, \textit{ where}$$

$$b \le \begin{cases} \left\lceil \frac{2\Delta}{c} \right\rceil \log_2(c+1) & \textit{if } c \le 2\Delta; \\ 4 + 2\log_2\Delta & \textit{if } 2\Delta + 1 \le c \le 16\Delta^2 - 1; \text{ and} \\ 4 & \textit{if } c \ge 16\Delta^2. \end{cases}$$

**Proof.** Let $G = (V, E)$ be the series-parallel graph. Let $|V| = n$. Then $|E| \le 2n - 3$. Let $T_b$ be a binary decomposition tree of $G$ and $n_{T_b}$ be the number of nodes in $T_b$. Then the number of leaf nodes are at most $2n - 3$ and hence $n_{T_b} \le 4n - 7$ [24]. We first construct an equivalent tree $T_q$ from the binary decomposition tree $T_b$ of $G$ as follow:

The edge incident to the terminals $v_s$ and $v_t$ of the graph associated with every leaf node is also incident to the terminals of the graph associated with its parent node. To prove this, we have to consider the following two cases separately.

**Case (a)**: The parent node of a leaf node in $T_b$ is a series node.

Let $x$ be an internal node having two children $y$ and $z$. Let $y$ be a leaf node, and $z$ be any node. Let $v_s$ and $v$ be the terminals of $G_y$ and $v$ and $v_t$ be the terminals of $G_z$. Then $v_s$ and $v_t$ are the terminals of $G_x$. Since every leaf node in $T_b$ represents an edge $e \in E$, the edge incident to vertex $v$ in $G_y$ is also incident to vertex $v_s$ in $G_y$. Since $G_y$ and $G_z$ are connected at vertex $v$ through a series connection, the edge $e$ is also incident to vertex $v_s$ in $G_x$.

**Case (b)**: The parent node of a leaf node in $T_b$ is a parallel node.

Let $x$ be an internal node having two children $y$ and $z$. Let $y$ be a leaf node, and $z$ be any node. Let $v_s$ and $v_t$ be the terminals of $G_y$. Since $G_y$ and $G_z$ are connected through a parellel connection at node $x$, then $v_s$ and $v_t$ are also the terminals of $G_z$, and $G_x$. So the edge incident to $v_s$ in $G_y$ is also incident to $v_s$ in $G_x$. Similarly, the edge incident to $v_t$ in $G_y$ is also incident to $v_t$ in $G_x$.

Therefore, we can remove all the leaf nodes from $T_b$.. We call the tree obtained after removing all leaf nodes from $T_b$ as "reduced tree" $T_r$. Then the reduced tree $T_r$ contains at most $2n - 4$ nodes.

Now we construct an equivalent tree $T_q$ from the reduced tree $T_r$ by removing both children nodes of a parallel node and contracting the edges in $T_r$ and repeating these for all parallel nodes in $T_r$. Now we claim that the equivalent tree contains at most $n$ nodes. This can be proved as follows.

During the construction of the graph $G$ by adding vertices one after another from null, if for including a new vertex in the graph $G$, one edge is added in $G$, then $|E| \leq n$ and the parent node of the leaf node corresponding to this edge is a series node in $T_b$. Hence the reduced tree contains at most $n$ nodes. But if for including one vertex in the graph the number of edges in the graph is increased by two. then $|E| \leq 2n - 3$ and one series node, one parallel node and two leaf nodes are added in the binary decomposition tree $T_b$, where the series node is the parent node of the two leaf nodes, and the parallel node is the parent node of the series node. So the reduced tree $T_r$ contains only one series node and one parallel node, where the parallel node is the parent of the series node in $T_r$. Therefore, the reduced tree contains at most $2n - 4$ nodes. Since the terminals of these series node and parallel node in $T_r$ are the same, we can remove both the children of that parent node and contract the edges in $T_r$. We call the tree obtained after removing both the children node of a parallel node and contracting the edges in $T_r$ as "equivalent tree" $T_q$. In this case for including one vertex in the graph $G$, node in the equivalent tree does not increase (because, two nodes are added in the reduced tree, and two nodes are removed from the equivalent tree), that is, the equivalent tree contains at most $n$ nodes in total. Thus we have verified our claim above. Let $n_T$ be the number of nodes in the equivalent tree $T_q$. Then $n_T \leq n$.

Recursively applying Lemma 3.1 to $T_q$, we first construct an α-vertex separator tree $T_\alpha(T_q)$ of tree $T_q$, where $\alpha = \max\{1, \left\lfloor \frac{c}{2\Delta} \right\rfloor\}$. The height $hT_\alpha(n_T)$ of tree $T_\alpha(T_q)$ satisfies the following recurrence relation

$$hT_\alpha(n_T) \leq 1 + hT_\alpha\left(\left\lfloor \frac{n_T}{q} \right\rfloor\right) \tag{3.1}$$

where $q = 2^{\lfloor \log_2(\alpha+3) \rfloor - 1} > \frac{\alpha+3}{4}$.

Solving the recurence (3.1) with $hT_\alpha(1) = 0$, we have

$$hT_\alpha(n_T) \leq \log_q n_T \tag{3.2}$$

We next claim that the $\alpha$-vertex-separator tree $T_\alpha(T_q)$ of tree $T_q$ can be transformed to a $c$-edge-separator tree $T_c(G)$ of $G$ with height

$$hT_c(n) \le \left\lceil \frac{2\Delta}{c} \right\rceil \log_q n.$$

First consider the case in which $c \ge 2\Delta + 1$. In this case $\alpha = \left\lfloor \frac{c}{2\Delta} \right\rfloor$, and any node of $T_\alpha(T_q)$ contains at most $\alpha$ nodes of $T_q$, each corresponding to an edge-separator of $G$ having at most $2\Delta$ edges of $G$. Thus any node of $T_\alpha(T_q)$ correspond to an edge-separator of $G$ having at most $c$ edges of $G$, and hence $T_\alpha(T_q)$ immediately yields a $c$-edge-separator tree $T_c(G)$ whose height $hT_c(n)$ is at most $hT_\alpha(n_T)$. Thus by (3.2) we have

$$hT_c(n) \le hT_\alpha(n_T) \le \log_q n_T \le \log_q n = \left\lceil \frac{2\Delta}{c} \right\rceil \log_q n \qquad (3.3)$$

Next consider the case in which $c \le 2\Delta$. Then $\alpha = 1$, and hence each node $x$ of $T_\alpha(T_q)$ contains at most one node, say $x'$, of $T_q$. Let $s = \left\lceil \frac{2\Delta}{c} \right\rceil$, and replace each node $x$ of $T_\alpha(T_q)$ with $s$ new node $x_1, x_2, x_3, \ldots, x_s$; node $x_j$ is the father of $x_{j+1}$, $1 \le j \le s\text{-}1$, in a new tree. At most $2\Delta$ edges of $G$ are incident to vertices in $S_{x'}$ and these edges form an edge-separator of $G$. We assign each node $x_j$, $1 \le j \le s$, at most $c$ edges so that each of these edges is assigned to some $x_j$, $1 \le j \le s$.

Then the resulting tree has height $\left\lceil \frac{2\Delta}{c} \right\rceil hT_\alpha(n_T)$, and immediately yields a $c$-edge-separator tree $T_c(G)$ of $G$. Thus by (3.2) the height $hT_c(n)$ of $T_c(G)$ satisfies,

$$hT_c(n) \le \left\lceil \frac{2\Delta}{c} \right\rceil hT_\alpha(n_T) \le \left\lceil \frac{2\Delta}{c} \right\rceil \log_q n_T \le \left\lceil \frac{2\Delta}{c} \right\rceil \log_q n.$$

Thus we have verified the claim above.

We finally obtain a $c$-edge-ranking of $G$ from the $c$-edge-separator tree $T_c(G)$ of $G$ as follows: for each $i$, $0 \le i \le hT_c(n)$, label by rank $i + 1$ all edges of $G$ corresponding to the nodes of $T_c(G)$ at lavel $i$. Then the resulting edge-labeling is a $c$-edge-ranking of $G$ using $1 + hT_c(n)$ ranks. Therefore we have

$$\begin{aligned} r_c'(G) &\le 1 + hT_c(n) \\ &\le 1 + \left\lceil \frac{2\Delta}{c} \right\rceil \log_q n. \\ &\le 1 + \left\lceil \frac{2\Delta}{c} \right\rceil \log_q(c+1) \log_{c+1} n \\ &\le 1 + b \log_{c+1} n \end{aligned}$$

where, $b = \left\lceil \frac{2\Delta}{c} \right\rceil \log_q(c+1)$.

Depending upon the value of $c$, we have the following three cases.

**Case 1:** $c \le 2\Delta$.

In this case, $\alpha = 1$ and $q = 2$. Therefore

$$\begin{aligned} b &= \left\lceil \frac{2\Delta}{c} \right\rceil \log_q(c+1) \\ &= \left\lceil \frac{2\Delta}{c} \right\rceil \log_2(c+1). \end{aligned}$$

Case 2: $2\Delta + 1 \leq c \leq 16\Delta^2 - 1.$

In this case, $\alpha \geq 1$, $q \geq 2$, $\left\lceil \frac{2\Delta}{c} \right\rceil = 1$, and $c + 1 \leq 16\Delta^2$. Therefore

$$
\begin{aligned}
b &= \left\lceil \frac{2\Delta}{c} \right\rceil \log_q (c+1) \\
&= \log_q (c+1) \\
&\leq \log_2 16\Delta^2 \\
&= 4 + 2\log_2 \Delta
\end{aligned}
$$

**Case 3** : $c \geq 16\Delta^2.$

In this case, $\alpha = \left\lfloor \frac{c}{2\Delta} \right\rfloor \geq \frac{c - 2\Delta + 1}{2\Delta}$. Therefore in this case we have

$$
\begin{aligned}
q &> \frac{\alpha + 3}{4} \\
&= \frac{\frac{c - 2\Delta + 1}{2\Delta} + 3}{4} \\
&= \frac{1}{2}\left(\frac{c+1}{4\Delta} + 1\right) \\
&\geq \left(\frac{c+1}{4\Delta}\right)^{\frac{1}{2}} \\
&= \frac{(c+1)^{\frac{1}{4}} . (c+1)^{\frac{1}{4}}}{2\sqrt{\Delta}} \\
&\geq (c+1)^{\frac{1}{4}}.
\end{aligned}
$$

Therefore,

$$\begin{aligned} b &= \left\lceil \frac{2\Delta}{c} \right\rceil \log_q (c+1) \\ &= \log_q (c+1) \\ &\leq \log_{(c+1)^{1/4}} (c+1) \\ &= 4. \end{aligned}$$

If $\Delta$ is a bounded integer, then $b = O(1)$ and hence $r_c{}'(G) \leq 1 + b \log_{c+1} n = O(\log_{c+1} n)$ even if $c$ is not a bounded integer. □

## 3.2 Equivalence Class

We use a dynamic programming algorithm to find an optimal $c$-edge-ranking of a series-parallel graph. Consider a binary decomposition tree $T_b$ of $G$. On each node of the binary decomposition tree, a table of all necessary partial solutions of the problem is computed, where each entry in the table represents a dominance class. The table of our algorithm has a size $O(n^{b+1} \log_{c+1}{}^2 n)$. Before defining the dominance class for the $c$-edge-ranking problem, we need to define some terms.

Let a node $x$ of $T_b$ corresponds to a subgraph $G_x = (V_x, E_x)$ of $G = (V, E)$. Let $R = \{1, 2, 3, \ldots, l\}$ be the set of ranks and let $\varphi : E_x \rightarrow R$ be an edge-labeling of the subgraph $G_x$. Iyer *et al.* introduced the idea of a "critical list" to solve the ordinary vertex-ranking problem of trees, to be a list containing the ranks of all edges visible from the root. We use similar idea to define visible list $L(\varphi, v)$, to be a list containing the ranks of all edges $e \in E_x$ visible from a vertex $v \in S_x$, *i. e.*,

$L(\varphi, v) = \{\varphi(e) \mid e \in E_x$ is visible from the vertex $v$ under $\varphi$ in $G_x\}$.

The ranks in the list $L(\varphi, v)$ are sorted in non-increasing order of ranks. The list $L(\varphi, v)$ may contain same ranks with repetition $\le c$. For an integer $i$, we denote by $count(L(\varphi, v), i)$ the number of $i$'s contained in $L(\varphi, v)$, that is, the number of visible edges with rank $i$. Then by lemma 2.1 an edge-labeling $\varphi$ of the subgraph $G_x$ associated with a node $x$ in $T_b$ obtained from extension of the $c$-edge-rankings $\eta$ and $\Psi$ of the subgraphs $G_y$ and $G_z$ associated with the two children $y$ and $z$ of $x$ in $T_b$, respectively will be a $c$-edge-ranking if and only if for any rank $i \in R$,

$$count(L(\varphi, v), i) \le c \text{ for each vertex } v \in S_y \cup S_z.$$

We next define a list-set $\pounds(\varphi)$ as follows:

$$\pounds(\varphi) = \{L(\varphi, v_s), L(\varphi, v_t)\}.$$

For an edge-labeling $\varphi$ of $G_x$, we define a term (hereinafter called as *obstacle*) $\lambda_\varphi$ as follows:

$$\lambda_\varphi = \min\{\lambda \mid G_x \text{ has a path } P \text{ from } v_s \text{ to } v_t \text{ such that } \varphi(e) \le \lambda \text{ for each edge } e \text{ of } P\}.$$

Since $G_x$ is a connected graph with two terminals $v_s$ and $v_t$, so $\lambda_\varphi \in R$. If edge $e \in E_x$ is visible from $v_t$ and $\varphi(e) \ge \lambda_\varphi$, then $e$ is visible from $v_s$ under $\varphi$ in $G_x$. Then clearly,

$$[\lambda_\varphi \le L(\varphi, v_s)] = [\lambda_\varphi \le L(\varphi, v_t)].$$

We next define a pair $R(\varphi)$ as follows:

$$R(\varphi) = (\pounds(\varphi), \lambda_{\varphi}).$$

We call such a pair $R(\varphi)$ the vector of $\varphi$ on node $x$. $R(\varphi)$ is called a *feasible* vector if the edge-labeling $\varphi$ is a $c$-edge-ranking of $G_x$.

A $c$-edge-ranking of $G_x$ is defined to be *extensible* if it can be extended to a $c$-edge-ranking of $G$ without changing the labeling of edges in $G_x$. We then have the following lemma.

**Lemma 3.3** *Let $\varphi$ and $\eta$ be the two c-edge-ranking of $G_x$ such that $R(\varphi) = R(\eta)$, then $\varphi$ is extensible if and only if $\eta$ is extensible.*

**Proof** : It suffices to prove that if $\varphi$ is extensible then $\eta$ is extensible. Suppose that $\varphi$ is extensible. Then $\varphi$ can be extended to a $c$-edge-ranking $\varphi'$ of $G = (V, E)$ such that $\varphi(e) = \varphi'(e)$ for any edge $e \in E_x$. Let $E^* = E - E_x$, and let $G^*$ be the subgraph of $G$ induced by $E^*$. Extend the $c$-edge-ranking $\eta$ of $G_x$ to an edge-labeling $\eta'$ of $G$ as follows:

$$\eta'(e) = \begin{cases} \eta(e), & \text{if } e \in E_x\text{; and} \\ \varphi'(e), & \text{if } e \in E^*. \end{cases}$$

Then it suffices to prove that $\eta'$ is a $c$-edge-ranking of $G$, that is, $m(H_{\eta'}, \eta', i) \le c$ for any connected component $H_{\eta'} = (V_{\eta'}, E_{\eta'})$ of the graph obtained from $G$ by deleting all edges $e \in E$ with $\eta'(e) \ge i$. There are the following two cases to consider.

**Case 1**: $H_{\eta'}$ has no vertex in $S_x$.

In this case $H_{\eta'}$ is a subgraph of either $G_x$ *or* $G^*$, since $G_x$ is connected with $G^*$ only through the vertices in $S_x$. Furthermore $\eta'|G_x = \eta$ and $\eta'|G^* = \varphi'|G^*$ are $c$-edge-ranking of $G_x$ and $G^*$, respectively. Therefore, $m(H_{\eta'}, \eta', i) \leq c$.

**Case 2**: $H_{\eta'}$ has a vertex $w$ in $S_x$.

Since $R(\varphi) = R(\eta)$, we have $L(\varphi, w) = L(\eta, w)$. Hence, deletion of all $e \in E$ with $\varphi'(e) \geq i$ from $G$ leaves a connected component $H_{\varphi'} = (V_{\varphi'}, E_{\varphi'})$ containing the vertex $w$. Since $\varphi'$ is a $c$-edge-ranking of $G$, $m(H_{\varphi'}, \varphi', i) \leq c$. Therefore it suffices to prove that $m(H_{\eta'}, \eta', i) = m(H_{\varphi'}, \varphi', i)$.

Since $R(\varphi) = R(\eta)$, we have $L(\varphi, w) = L(\eta, w)$ for each vertex $v \in S_x$ and $\lambda_\varphi = \lambda_\eta$. Furthermore, $\eta'|G^* = \varphi'|G^*$. Therefore one can observe that $V_{\eta'} \cap S_x = V_{\varphi'} \cap S_x$ and $E_{\eta'} \cap E^* = E_{\varphi'} \cap E^*$. Let $H_{\eta x}$ be the subgraph of $H_{\eta'}$ induced by $E_{\eta'} \cap E_x$, and $H_{\eta'}{}^*$ be the subgraph of $H_{\eta'}$ induced by $E_{\eta'} \cap E^*$. Similarly, let $H_{\varphi x}$ be the subgraph of $H_{\varphi'}$ induced by $E_{\varphi'} \cap E_x$, and $H_{\varphi'}{}^*$ be the subgraph of $H_{\varphi'}$ induced by $E_{\varphi'} \cap E^*$. Then $m(H_{\eta'}, \eta', i) = m(H_{\eta x}, \eta, i) + m(H_{\eta'}{}^*, \eta', i)$ and $m(H_{\varphi'}, \varphi', i) = m(H_{\varphi x}, \varphi, i) + m(H_{\varphi'}{}^*, \varphi', i)$. Since $E_{\eta'} \cap E^* = E_{\varphi'} \cap E^*$ and $\eta'|G^* = \varphi'|G^*$, we have $m(H_{\eta'}{}^*, \eta', i) = m(H_{\varphi'}{}^*, \varphi', i)$. Therefore it suffice to prove that $m(H_{\eta x}, \eta, i) = m(H_{\varphi x}, \varphi, i)$.

We next show that each of the connected component $H_{\varphi x}$ and $H_{\eta x}$ contains at least one vertex in $S_x$. Suppose for a contradiction that a connected component $D$ of $H_{\eta x}$ or $H_{\varphi x}$, say $H_{\eta x}$, contains no vertex in $S_x$. Since $H_{\eta'}$ is a connected graph containing a vertex $w \in S_x$, $w$ is connected to a vertex of $D$ by a path $H_{\eta'}$. However it is impossible because $D$ has no vertex in $S_x$ and $H_{\eta x}$ is connected with $H_{\eta'}$* only through the vertices in $S_x$.

Let $v_s$ be any vertex in $V(H_{\eta x}) \cap S_x = V(H_{\varphi x}) \cap S_x$. Let $D_\eta$ be the connected component of $H_{\eta x}$ that contains vertex $w = v_s$. and let $D_\varphi$ be the connected component of $H_{\varphi x}$ that contains vertex $v_s$. We now claim that $V(D_\eta) \cap S_x = V(D_\varphi) \cap S_x$ and $m(D_\eta, \eta, i) = m(D_\varphi, \varphi, i)$.

Let $v_t \in V(D_\varphi)$, then obviously $\lambda_\varphi \le i$. Since $R(\varphi) = R(\eta)$, we have $\lambda_\varphi = \lambda_\eta \le i$. Therefore, $v_t \in V(D_\eta)$. Hence we have prove that $V(D_\eta) \cap S_x = V(D_\varphi) \cap S_x$. Clearly $m(D_\varphi, \varphi, i) = count(L(\varphi, v_s), i)$ and $m(D_\eta, \eta, i) = count(L(\eta, v_s), i)$. Since $L(\varphi, v_s) = L(\eta, v_s)$, we have $count(L(\varphi, v_s), i) = count(L(\eta, v_s), i)$ and hence $m(D_\varphi, \varphi, i) = m(D_\eta, \eta, i)$. Thus we have verified the claim above.

The claim above implies that $H_{\eta x}$ and $H_{\varphi x}$ have the same number of connected components and there can be at most two connected component of $H_{\eta x}$ and $H_{\varphi x}$, respectively. These can be recognized by $H_{\eta s}$ and $H_{\eta t}$ in $H_{\eta x}$ and $H_{\varphi s}$ and $H_{\varphi t}$ in $H_{\varphi x}$. Furthermore $m(H_{\eta s}, \eta, i) = m(H_{\varphi s}, \varphi, i)$ and $m(H_{\eta t}, \eta, i) = m(H_{\varphi t}, \varphi, i)$. Since $m(H_{\eta x}, \eta, i) = m(H_{\eta s}, \eta, i) + m(H_{\eta t}, \eta, i)$ and $m(H_{\varphi x}, \varphi, i) = m(H_{\varphi s}, \varphi, i) + m(H_{\varphi t}, \varphi, i)$, we have $m(H_{\eta x}, \eta, i) = m(H_{\varphi x}, \varphi, i)$. □

Thus a feasible vector $R(\varphi)$ of $\varphi$ on node $x$ can be seen as an *equivalence class* of extensible $c$-edge-ranking of $G_x$.

## 3.3 Peer Class

Since at parallel connection of two graphs $G_y$ and $G_z$, ranks in the visible list at one terminal $v \in S_x$, of one graph, say $G_y$, may become visible to the other terminal of combined graph $G_x$ via the path between $v_s$ and $v_t$ of $G_z$ when $\lambda_\Psi \leq \lambda_\eta$ and vice versa, we consider this condition in advance by assuming an imaginary path between $v_s$ and $v_t$ of $G_z$ and $G_y$ having different possible obstacle. We call the lists so obtain at the terminals as the peer-lists.

Let $x$ be a node in $T_b$ and has two children $y$ and $z$. Let $\eta$ and $\psi$ be the two $c$-edge-rankings of $G_y$ and $G_z$, respectively. If $x$ is a leaf node in $T_b$, then $G_y$ and $G_z$ are null graphs. Let $\varphi$ be an edge-labeling of $G_x$ extended from $\eta$ and $\psi$. Let $r$, $1 \leq r \leq l$, be a positive integer. Let $r_\varphi$ be a positive integer defined as follows:

$$r_\varphi = \begin{cases} r, & \text{if } 1 \leq r \leq \lambda_\varphi - 1, \text{ and} \\ \lambda_\varphi & \text{if } \lambda_\varphi \leq r \leq l. \end{cases}$$

We define a peer-list $L_p(\varphi, v, r)$ and a pear-list-set $£_p(\varphi)$ for a series-parallel graph as follows:

$$L_p(\varphi, v_s, r) = L(\varphi, v_s) \cup [r_\varphi \leq L(\varphi, v_t) < \lambda_\varphi]$$

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

$$£_p(\varphi) = \{L_p(\varphi, v_s, r), L_p(\varphi, v_t, r)\}$$

For an edge-labeling $\varphi$ of $G_x$, the introduction of the concept of peer-list actually incorporates ranks in the list $[L(\varphi, v) < \lambda_\varphi]$ visible at one terminal $v$, $v \in S_x$, is made visible to the other terminal via an imaginary path between $v_s$ and $v_t$ of $G_x$ having $r$ as the obstacle. Since in parallel connection of two graphs $G_y$ and $G_z$, the ranks visible only at one terminal of $G_y$ may become visible at the other terminal of $G_y$ via the path of $G_z$ having obstacle $\lambda_\psi$ and vice versa, consideration of peer-list actually means to keep in advance the projected list as if another graph having a path with obstacle $r$ is connected in parallel with the graph $G_x$. So, this extra component in the peer-list will be called the look-ahead component.

we next define a pair $R_p(\varphi)$, called peer-vector, as follows:

$$R_p(\varphi) = (£_p(\varphi), \lambda_\varphi).$$

We call such a pair $R_p(\varphi)$ the vector of $\varphi$ on node $x$. $R_p(\varphi)$ is called a feasible vector if the edge-labeling $\varphi$ is a $c$-edge-ranking of $G_x$ and $count(L_p(\varphi, v, r), i) \le c$ for all $v \in S_x$ and all $i \in R$.

Now we give the following two Lemmas for computing the peer-lists $L_p(\varphi, v_s, r)$ and $L_p(\varphi, v_t, r)$ of $G_x$ extended from $G_y$ and $G_z$.

**Lemma 3.4** *Let $x$ be a series node and has two children $y$ and $z$ in $T_b$. Let $G_x$, $G_y$ and $G_z$ be the subgraphs of $G$ associated with the nodes $x$, $y$ and $z$, respectively. Then $G_x = G_y \bullet G_z$. Let $v_s$ and $v$ be the terminal vertices of $G_y$, and $v$ and $v_t$ be the terminals of $G_z$. Let $\eta$ and $\Psi$ be the $c$-edge-rankings of $G_y$ and $G_z$, respectively. Let $\varphi$ be the edge-labeling of $G_x$ extended from $\eta$ and $\Psi$. Let $r$, $1 \le r \le l$, be a positive integer. Let $l_\eta = max\{r, \lambda_\Psi\}$ and let $l_\Psi = max\{r, \lambda_\eta\}$. Then,*

$\lambda_\varphi = \max\{\lambda_\Psi, \lambda_\eta\};$

$L_p(\varphi, v_s, r) = L_p(\eta, v_s, l_\eta) \cup [\lambda_\eta \le L_p(\Psi, v, \lambda_\Psi)] \cup [r \le L_p(\Psi, v_t, \lambda_\Psi) < \lambda_\varphi]$; and

$L_p(\varphi, v_t, r) = L_p(\Psi, v_t, l_\Psi) \cup [\lambda_\Psi \le L_p(\eta, v, \lambda_\eta)] \cup [r \le L_p(\eta, v_s, \lambda_\eta) < \lambda_\varphi]$.

**Proof**: We provide proof for the first part. The second part can be prove analogously. Let $i$ be any rank in $L_p(\varphi, v_s, r)$. Then $i$ is visible from $v_s$ under labeling $\varphi$ in $G_x$ with an imaginary path between the two terminals of $G_x$ having obstacle $r$.

Since $G_x = G_y \bullet G_z$, $l_\eta$ denotes the obstacle for look-ahead component of $\eta$. All ranks in $[\lambda_\eta \le L_p(\Psi, v, \lambda_\Psi)] = [\lambda_\eta \le L(\Psi, v)]$ will be visible from $v_s$ through the path of $\eta$ having obstacle $\lambda_\eta$. All ranks $\ge r$ in the list $L_p(\Psi, v_t, \lambda_\Psi)$ i.e. $[r \le L(\Psi, v_t)]$ will be visible through the imaginary path having obstacle $r$. Since $[\lambda_\Psi \le L(\Psi, v)] = [\lambda_\Psi \le L(\Psi, v_t)]$ and the ranks in the list $[\lambda_\eta \le L(\Psi, v)]$ are already visible from $v_s$ through the path in $\eta$, so which of the rest ranks in the list $[r \le L(\Psi, v_t)]$ are visible through the imaginary path having obstacle $r$ depends upon the relative value of $\lambda_\eta$ and $\lambda_\Psi$ as follows:

**Case a:** $\lambda_\eta \ge \lambda_\Psi$.

In this case, $[\lambda_\eta \le L(\Psi, v_t)] = [\lambda_\eta \le L(\Psi, v)]$ as $[\lambda_\Psi \le L(\Psi, v_t)] = [\lambda_\Psi \le L(\Psi, v)]$; and

then, $[r \le L(\Psi, v_t)] = [r \le L(\Psi, v_t) < \lambda_\eta] \cup [\lambda_\eta \le L(\Psi, v_t)]$

$= [r \le L(\Psi, v_t) < \lambda_\eta] \cup [\lambda_\eta \le L(\Psi, v)]$

but ranks in the list $[\lambda_\eta \le L(\Psi, v)]$ are already visible from $v_s$ through the path in $\eta$. So only ranks in the list $[r \le L(\Psi, v_t) < \lambda_\eta]$ are sufficient for the look-ahead component and since $\lambda_\eta \ge \lambda_\Psi$, so $\lambda_\eta = \max\{\lambda_\eta, \lambda_\Psi\} = \lambda_\varphi$ and $[r \le L(\Psi, v_t) < \lambda_\eta] = [r \le L(\Psi, v_t) < \lambda_\varphi]$.

**Case b:** $\lambda_\eta < \lambda_\Psi$.

In this case, $[\lambda_\Psi \leq L(\Psi, v_t)] = [\lambda_\Psi \leq L(\Psi, v)]$. Then,

$$\begin{aligned}[r \leq L(\Psi, v_t)] &= [r \leq L(\Psi, v_t) < \lambda_\Psi] \cup [\lambda_\Psi \leq L(\Psi, v_t)] \\ &= [r \leq L(\Psi, v_t) < \lambda_\Psi] \cup [\lambda_\Psi \leq L(\Psi, v)].\end{aligned}$$

But since $\lambda_\eta < \lambda_\Psi$, so $[\lambda_\Psi \leq L(\Psi, v)] \subseteq [\lambda_\eta \leq L(\Psi, v)]$ and all ranks in list $[\lambda_\Psi \leq L(\Psi, v)]$ are already visible from $v_s$ through the path in $\eta$. So only ranks in list $[r \leq L(\Psi, v_t) < \lambda_\Psi]$ are sufficient for the look-ahead component and since $\lambda_\eta < \lambda_\Psi$, so $\lambda_\Psi = \max\{\lambda_\eta, \lambda_\Psi\} = \lambda_\varphi$ and $[r \leq L(\Psi, v_t) < \lambda_\Psi] = [r \leq L(\Psi, v_t) < \lambda_\varphi]$. Thus we have either $i \in L_p(\eta, v_s, l_\eta)$ or $i \in [\lambda_\eta \leq L_p(\Psi, v, \lambda_\Psi)]$ or $i \in [r \leq L_p(\Psi, v_t, \lambda_\Psi) < \lambda_\varphi]$. Thus,

$$L_p(\varphi, v_s, r) \subseteq L_p(\eta, v_s, l_\eta) \cup [\lambda_\eta \leq L_p(\Psi, v, \lambda_\Psi)] \cup [r \leq L_p(\Psi, v_t, \lambda_\Psi) < \lambda_\varphi] \qquad (1)$$

Assume that $i$ be any rank in $L_p(\eta, v_s, l_\eta) \cup [\lambda_\eta \leq L_p(\Psi, v, \lambda_\Psi)] \cup [r \leq L_p(\Psi, v_t, \lambda_\Psi) < \lambda_\varphi]$, then either $i \in L_p(\eta, v_s, l_\eta)$ or $i \in [\lambda_\eta \leq L_p(\Psi, v, \lambda_\Psi)]$ or $i \in [r \leq L_p(\Psi, v_t, \lambda_\Psi) < \lambda_\varphi]$. If $i \in L_p(\eta, v_s, l_\eta)$ then clearly $i \in L_p(\varphi, v_s, r)$. If $i \notin L_p(\eta, v_s, l_\eta)$ then either $i \in [\lambda_\eta \leq L_p(\Psi, v, \lambda_\Psi)]$ or $i \in [r \leq L_p(\Psi, v_t, \lambda_\Psi) < \lambda_\varphi]$. If $i \in [\lambda_\eta \leq L_p(\Psi, v, \lambda_\Psi)]$ then $i$ will be visible from $v_s$ through the path in $\eta$ having obstacle $\lambda_\eta$. If $i \notin [\lambda_\eta \leq L_p(\Psi, v, \lambda_\Psi)]$ then $i \in [r \leq L_p(\Psi, v_t, \lambda_\Psi) < \lambda_\varphi]$ and $i$ will be visible from $v_s$ through the imaginary path having obstacle $r$. Hence $i \in L_p(\varphi, v_s, r)$. Thus we obtain,

$$L_p(\varphi, v_s, r) \supseteq L_p(\eta, v_s, l_\eta) \cup [\lambda_\eta \leq L_p(\Psi, v, \lambda_\Psi)] \cup [r \leq L_p(\Psi, v_t, \lambda_\Psi) < \lambda_\varphi] \qquad (2)$$

Therefore from equation (1) and (2), we have

$$L_p(\varphi, v_s, r) = L_p(\eta, v_s, l_\eta) \cup [\lambda_\eta \leq L_p(\Psi, v, \lambda_\Psi)] \cup [r \leq L_p(\Psi, v_t, \lambda_\Psi) < \lambda_\varphi]. \qquad \square$$

**Lemma 3.5** *Let $x$ be a parallel node with two children $y$ and $z$ in $T_b$. Let $G_x$, $G_y$ and $G_z$ be the subgraphs of $G$ associated with the nodes $x$, $y$ and $z$, respectively. Then $G_x = G_y \parallel G_z$. Let $v_s$ and $v_t$ be the terminal vertices of $G_y$ and $G_z$. Let $\eta$ and $\Psi$ be the c-edge-rankings of $G_y$ and $G_z$, respectively. Let $\varphi$ be the edge-labeling of $G_x$ extended from $\eta$ and $\Psi$. Let $r$, $1 \leq r \leq l$ be a positive integer. Then,*

$\lambda_\varphi = \min\{\lambda_\Psi, \lambda_\eta\}$;

$L_p(\varphi, v_s, r) = L_p(\eta, v_s, r) \cup L_p(\Psi, v_s, r)$; and

$L_p(\varphi, v_t, r) = L_p(\eta, v_t, r) \cup L_p(\Psi, v_t, r)$.

**Proof:** We provide proof for the first part. The second part can be prove analogously. Let $i$ be any rank in $L_p(\varphi, v_s, r)$. Then $i$ is visible from $v_s$ under labeling $\varphi$ in $G_x$ with an imaginary path between the two terminals of $G_x$ having obstacle $r$ for the look-ahead component. Since $G_x = G_y \parallel G_z$,we have either $i \in L_p(\eta, v_s, r)$ or $i \in L_p(\Psi, v_s, r)$. Thus,

$$L_p(\varphi, v_s, r) \subseteq L_p(\eta, v_s, r) \cup L_p(\Psi, v_s, r) \tag{3}$$

Assume that $i \in L_p(\eta, v_s, r) \cup L_p(\Psi, v_s, r)$. Then either $i \in L_p(\eta, v_s, r)$ or $i \in L_p(\Psi, v_s, r)$. Then clearly $i \in L_p(\varphi, v_s, r)$ since $G_x = G_y \parallel G_z$. Hence $i \in L_p(\varphi, v_s, r)$. Thus we obtain,

$$L_p(\varphi, v_s, r) \supseteq L_p(\eta, v_s, r) \cup L_p(\Psi, v_s, r) \tag{4}$$

From equation (3) and (4), we have,

$$L_p(\varphi, v_s, r) = L_p(\eta, v_s, r) \cup L_p(\Psi, v_s, r)$$

In the following lemma we define the peer class. □

**Lemma 3.6** *Let $G_x$ be a series-parallel graph. Let $\varphi$ be a c-edge-ranking of $G_x$. Then*

$$R(\varphi) \subseteq R_p(\varphi).$$

**Proof :** Let $\lambda_\varphi$ be the obstacle of the $c$-edge-ranking $\varphi$. Then it suffices to show that for any list-set $£(\varphi) \in R(\varphi)$, there is a peer-list-set $£_p(\varphi) \in R_p(\varphi)$ such that

$$L(\varphi, v) = L_p(\varphi, v, r), \quad \text{where } v \in S_x.$$

Let us choose, $r = \lambda_\varphi$ and $v = v_s$, then we have $r_\varphi = \lambda_\varphi$,and

$$L_p(\varphi, v_s, r) \;=\; L(\varphi, v_s) \cup [r_\varphi \leq L(\varphi, v_t) < \lambda_\varphi]$$

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

Since $[\lambda_\varphi \leq L(\varphi, v_t) < \lambda_\varphi] = \phi$. So,

$L_p(\varphi, v_s, \lambda_\varphi) = L(\varphi, v_s)$. Similarly we can show that,

$L_p(\varphi, v_t, \lambda_\varphi) = L(\varphi, v_t)$. □

Thus a feasible vector $R_p(\varphi)$ of $\varphi$ on node $x$ can be seen as a *peer class* which is a super set of the equivalence class and hence $\varphi$ is an extensible $c$-edge-ranking of $G_x$.

## 3.4 Dominance Class

We define our dominance class as follows.

Let $R_p(x)$ be the set of feasible vectors for $G_x$, Then we apply the following elimination rule on $R_p(x)$. Let $\varphi$ and $\varphi'$ be any two labeling of $G_x$, we say that $\varphi$ *dominates* $\varphi'$ if any of the following conditions is true:

(1) $L_p(\varphi, v_s, r) = L_p(\varphi', v_s, r), L_p(\varphi, v_t, r) \preceq L_p(\varphi', v_t, r)$, and $\lambda_\varphi = \lambda_\varphi'$.

(2) $L_p(\varphi, v_s, r) \preceq L_p(\varphi', v_s, r), L_p(\varphi, v_t, r) = L_p(\varphi', v_t, r)$, and $\lambda_\varphi = \lambda_\varphi'$.

Then the set of feasible vectors obtained from $R_p(x)$ applying the above elimination rule is called *dominance class $R_d(x)$*.

In the following lemma we show that the dominance class $R_d(x)$ on node $x$ can be seen as a set of vectors of extensible $c$-edge-ranking of $G_x$.

**Lemma 3.7** *Let $R_d'(x)$, $R_d(y)$ and $R_d(z)$ be the dominance classes obtained from $R_p'(x)$, $R_p(y)$ and $R_p(z)$, respectively. Let $R_d(x)$ be the dominance classes extended from $R_d(y)$ and $R_d(z)$. Then $R_d'(x) = R_d(x)$.*

**Proof:** We prove the lemma by induction.

**Basis**: Let $x$ be a leaf node in $T_b$. Then $G_x$ corresponds to an edge $e_x = (v_s, v_t)$ in $G$. Let $\varphi$ and $\varphi'$ be the two $c$-edge-ranking of $G_x$. Then $\varphi(e) = \varphi'(e)$ and $\lambda_\varphi = \lambda_\varphi' = \varphi(e) = \varphi'(e)$. $\varphi(e)$ is visible from both the terminals of $G_x$ and so any imaginary path is meaningless. Then $L_p(\varphi, v_s, r) = L_p(\varphi', v_s, r)$ and $L_p(\varphi, v_t, r) = L_p(\varphi', v_t, r)$ for $r = \lambda_\varphi = \lambda_\varphi'$. Note that the only possible value of $r$ is $\lambda_\varphi$. Then Clearly $R_d(x) = R_p(x)$.

**Induction:** Let $x$ be an internal node in $T_b$ having two children $y$ and $z$. Now consider the following two cases, depending on the connection in node $x$.

**Case 1 :** $G_x = G_y \bullet G_z$.

Let $v_s$ and $v$ be the terminal vertices of $G_y$ and $v$ and $v_t$ be the terminal of $G_z$. Then $G_x$ has $v_s$ and $v_t$ as terminals. We show that $R_d'(x) = R_d(x)$. Suppose that, there exists a feasible vector $R_p(\varphi')$ in $R_p'(x)$, obtained from $\eta'(y)$ and $\Psi'(z)$. Now from lemma 3.4 we have,

$$L_p(\varphi', v_s, r) = L_p(\eta', v_s, l_\eta) \cup [\lambda'_\eta \le L_p(\Psi', v, \lambda'_\Psi)] \cup [r \le L_p(\Psi', v_t, \lambda'_\Psi) < \lambda'_\varphi] \qquad (5)$$

$$L_p(\varphi', v_t, r) = L_p(\Psi', v_t, l_\Psi) \cup [\lambda_\Psi' \le L_p(\eta', v, \lambda'_\eta)] \cup [r \le L_p(\eta', v_s, \lambda'_\eta) < \lambda'_\varphi] \qquad (6)$$

Let $\eta(y)$ and $\Psi(z)$ be the two rankings in $R_d(y)$ and $R_d(z)$ such that,

(1) $L_p(\eta, v_s, l_\eta) = L_p(\eta', v_s, l_\eta)$, $L_p(\eta, v, l_\eta) \preceq L_p(\eta', v, l_\eta)$, and $\lambda_\eta = \lambda'_\eta$.

(2) $L_p(\Psi, v_t, \lambda_\Psi) = L_p(\Psi'', v_t, \lambda'_\Psi)$, $L_p(\Psi, v, \lambda_\Psi) \preceq L_p(\Psi'', v, \lambda'_\Psi)$, and $\lambda_\Psi = \lambda'_\Psi$.

(3) $L_p(\Psi, v_t, l_\Psi) = L_p(\Psi'', v_t, l_\Psi)$, $L_p(\Psi, v, l_\Psi) \preceq L_p(\Psi'', v, l_\Psi)$, $\lambda_\Psi = \lambda'_\Psi$.

(4) $L_p(\eta, v_s, \lambda_\eta) = L_p(\eta', v_s, \lambda'_\eta)$, $L_p(\eta, v, \lambda_\eta) \preceq L_p(\eta', v, \lambda'_\eta)$, and $\lambda_\eta = \lambda'_\eta$.

Solving equations (5) and (6) applying the above equations, we get the following result:

$$\lambda'_\varphi = \max\{\lambda'_\eta, \lambda'_\Psi\} = \max\{\lambda_\eta, \lambda_\Psi\} = \lambda_\varphi.$$

$$\begin{aligned} L_p(\varphi', v_s, r) &= L_p(\eta', v_s, l_\eta) \cup [\lambda'_\eta \le L_p(\Psi'', v, \lambda'_\Psi)] \cup [r \le L_p(\Psi'', v_t, \lambda'_\Psi) < \lambda'_\varphi] \\ &= L_p(\eta, v_s, l_\eta) \cup [\lambda_\eta \le L_p(\Psi'', v, \lambda_\Psi)] \cup [r \le L_p(\Psi, v_t, \lambda_\Psi) < \lambda_\varphi] \qquad (7) \end{aligned}$$

$$\begin{aligned} L_p(\varphi', v_t, r) &= L_p(\Psi'', v_t, l_\Psi) \cup [\lambda_\Psi' \le L_p(\eta', v, \lambda'_\eta)] \cup [r \le L_p(\eta', v_s, \lambda'_\eta) < \lambda'_\varphi] \\ &= L_p(\Psi, v_t, l_\Psi) \cup [\lambda_\Psi \le L_p(\eta', v, \lambda_\eta)] \cup [r \le L_p(\eta, v_s, \lambda_\eta) < \lambda_\varphi] \qquad (8) \end{aligned}$$

We have, $L_p(\Psi, v, \lambda_\Psi) \preceq L_p(\Psi'', v, \lambda_\Psi)$, so $[\lambda_\Psi \le L_p(\Psi, v, \lambda_\Psi)] = [\lambda_\Psi \le L_p(\Psi'', v, \lambda_\Psi)]$. Now we consider the following two cases separately:

**Case a:** $\lambda_\eta \ge \lambda_\Psi$.

In this case $[\lambda_\Psi \le L_p(\Psi, v, \lambda_\Psi)] = [\lambda_\Psi \le L_p(\Psi'', v, \lambda_\Psi)]$ as $L_p(\Psi, v, \lambda_\Psi) \preceq L_p(\Psi'', v, \lambda'_\Psi)$. So,

$$[\lambda_\eta \le L_p(\Psi, v, \lambda_\Psi)] = [\lambda_\eta \le L_p(\Psi'', v, \lambda_\Psi)] \qquad (9)$$

We have, from equation (7) and (9),

$$\begin{aligned} L_p(\varphi', v_s, r) &= L_p(\eta, v_s, l_\eta) \cup [\lambda_\eta \le L_p(\Psi'', v, \lambda_\Psi)] \cup [r \le L_p(\Psi, v_t, \lambda_\Psi) < \lambda_\varphi] \\ &= L_p(\eta, v_s, l_\eta) \cup [\lambda_\eta \le L_p(\Psi, v, \lambda_\Psi)] \cup [r \le L_p(\Psi, v_t, \lambda_\Psi) < \lambda_\varphi] \end{aligned}$$

$$L_p(\varphi', v_s, r) = L_p(\varphi, v_s, r) \qquad (10)$$

Since $L_p(\eta', v, \lambda'_\eta) \succeq L_p(\eta, v, \lambda_\eta)$, and $\lambda_\eta = \lambda'_\eta$, then $[\lambda_\eta \leq L_p(\eta', v, \lambda'_\eta)] = [\lambda_\eta \leq L_p(\eta, v, \lambda_\eta)]$. Since $\lambda_\eta \geq \lambda_\Psi$, So, $[\lambda_\Psi \leq L_p(\eta', v, \lambda'_\eta)] \succeq [\lambda_\Psi \leq L_p(\eta, v, \lambda_\eta)]$ (11)

We have from equation (8) and (11),

$$L_p(\varphi', v_t, r) = L_p(\Psi, v_t, l_\Psi) \cup [\lambda_\Psi \leq L_p(\eta', v, \lambda_\eta)] \cup [r \leq L_p(\eta, v_s, \lambda_\eta) < \lambda_\varphi]$$

$$\succeq L_p(\Psi, v_t, l_\Psi) \cup [\lambda_\Psi \leq L_p(\eta, v, \lambda_\eta)] \cup [r \leq L_p(\eta, v_s, \lambda_\eta) < \lambda_\varphi]$$

$$L_p(\varphi', v_t, r) \succeq L_p(\varphi, v_t, r) \tag{12}$$

**Case b:** $\lambda_\eta < \lambda_\Psi$.

We have $L_p(\Psi', v, \lambda'_\Psi) \succeq L_p(\Psi, v, \lambda_\Psi)$ and $\lambda'_\Psi = \lambda_\Psi$, then $[\lambda_\Psi \leq L_p(\Psi', v, \lambda_\Psi)] = [\lambda_\Psi \leq L_p(\Psi, v, \lambda_\Psi)]$. Since $\lambda_\eta < \lambda_\Psi$, so

$$[\lambda_\eta \leq L_p(\Psi', v, \lambda_\Psi)] \succeq [\lambda_\eta \leq L_p(\Psi, v, \lambda_\Psi)] \tag{13}$$

We have from equation (7) and (13),

$$L_p(\varphi', v_s, r) = L_p(\eta, v_s, l_\eta) \cup [\lambda_\eta \leq L_p(\Psi', v, \lambda_\Psi)] \cup [r \leq L_p(\Psi, v_t, \lambda_\Psi) < \lambda_\varphi]$$

$$\succeq L_p(\eta, v_s, l_\eta) \cup [\lambda_\eta \leq L_p(\Psi, v, \lambda_\Psi)] \cup [r \leq L_p(\Psi, v_t, \lambda_\Psi) < \lambda_\varphi]$$

$$L_p(\varphi', v_s, r) \succeq L_p(\varphi, v_s, r) \tag{14}$$

Again we have $L_p(\eta, v, \lambda_\eta) \preceq L_p(\eta', v_s, \lambda'_\eta)$, and $\lambda_\eta = \lambda'_\eta$, so $[\lambda_\eta \leq L_p(\eta', v, \lambda_\eta)] = [\lambda_\eta \leq L_p(\eta, v, \lambda_\eta)]$. Since $\lambda_\eta < \lambda_\Psi$, so

$$[\lambda_\Psi \leq L_p(\eta', v, \lambda_\eta)] = [\lambda_\Psi \leq L_p(\eta, v, \lambda_\eta)] \tag{15}$$

We have from equation (8) and (15),

$$L_p(\varphi', v_t, r) = L_p(\Psi, v_t, l_\Psi) \cup [\lambda_\Psi \leq L_p(\eta', v, \lambda_\eta)] \cup [r \leq L_p(\eta, v_s, \lambda_\eta) < \lambda_\varphi]$$

$$= L_p(\Psi, v_t, l_\Psi) \cup [\lambda_\Psi \leq L_p(\eta, v, \lambda_\eta)] \cup [r \leq L_p(\eta, v_s, \lambda_\eta) < \lambda_\varphi]$$

$$L_p(\varphi', v_t, r) = L_p(\varphi, v_t, r) \tag{16}$$

Therefore, from the $c$-edge-rankings $\eta(y)$ and $\Psi(z)$, we get an edge-labeling $\varphi$ $(x)$ such that

(a) if $\lambda_\eta \geq \lambda_\Psi$ then, $L_p(\varphi, v_s, r) = L_p(\varphi', v_s, r)$, $L_p(\varphi, v_t, r) \preceq L_p(\varphi', v_t, r)$, and $\lambda_\varphi = \lambda'_\varphi$; and

(b) if $\lambda_\eta < \lambda_\Psi$ then, $L_p(\varphi, v_s, r) \preceq L_p(\varphi', v_s, r)$, $L_p(\varphi, v_t, r) = L_p(\varphi', v_t, r)$, and $\lambda_\varphi = \lambda'_\varphi$.

Since $\varphi'$ is a feasible vector, so if $\varphi$ is not a feasible vector, then it is always possible to convert $\varphi$ to a feasible vector by relabeling some edges as follows:

**Case I:** $L_p(\varphi, v_s, r) = L_p(\varphi', v_s, r)$, $L_p(\varphi, v_t, r) \preceq L_p(\varphi', v_s, r)$, and $\lambda_\varphi = \lambda'_\varphi$.

Since $\varphi'$ is a feasible vector and $\varphi$ is not a feasible vector, so there is a rank $\lambda_\Psi \leq i < \lambda_\varphi$ such that $count(L_p(\varphi, v_t, r), i) > c$. Let $\alpha$ be the largest one among all such rank $i$. Then there is a rank $j$, $\alpha < j < \lambda_\varphi$ such that, $count(L_p(\varphi, v_t, r), j) < count(L_p(\varphi', v_t, r), j)$ and hence $count(L_p(\varphi, v_t, r), j) < c$. Let $\beta$ be the smallest rank among all such rank $j$. We also observe that the rank $\alpha$ in the list $L_p(\varphi, v_t, r)$ comes from the list $[\lambda_\Psi \leq L_p(\eta, v, \lambda_\eta)]$ i,e. from the $\eta$ ranking. Now we know the edge incident to $v$ of the graph $G_y$ through which rank $\alpha$ is visible at terminal $v$ under $\eta$ and hence at terminal $v_t$ under $\varphi$, we can hide this rank $\alpha$ from being visible at $v_t$ of $G_x$ by relabeling the edge incident to $v$ under $\eta$ by the rank $\beta$. After this relabeling we update the list $L(\eta, v)$ and $L_p(\varphi, v_t, r)$. If after this relabeling and updating, we have again any rank $\alpha < \lambda_\varphi$ such that $count(L_p(\varphi, v_t, r), \alpha) >$ c then we repeat the process of relabeling and updating list $L(\eta, v)$ and $L_p(\varphi, v_t, r)$ as described above until $count(L_p(\varphi, v_t, r), i) \leq$ c for all rank $i \in R$ and we must reach such a situation because $L_p(\varphi, v_t, r) \preceq L_p(\varphi', v_t, r)$ and $count(L_p(\varphi', v_t, r), i) \leq c$ for all rank $i \in R$..

**Case II:** $L_p(\varphi, v_s, r) \preceq L_p(\varphi', v_s, r)$, $L_p(\varphi, v_t, r) = L_p(\varphi', v_t, r)$, and $\lambda_\varphi = \lambda'_\varphi$.

Since $\varphi'$ is a feasible vector and $\varphi$ is not a feasible vector, so there is a rank $\lambda_\eta \leq i < \lambda_\varphi$ such that $count(L_p(\varphi, v_s, r), i) > c$. Let $\alpha$ be the largest one among all such rank $i$. Then there is a rank $j$, $\alpha < j < \lambda_\varphi$ such that, $count(L_p(\varphi, v_s, r), j) < count(L_p(\varphi', v_s, r), j)$ and hence $count(L_p(\varphi, v_s, r), j) < c$. Let $\beta$ be the smallest rank among all such rank $j$. We also observe that the rank $\alpha$ in the list $L_p(\varphi, v_s, r)$ comes from the list $[\lambda_\eta \leq L_p(\Psi, v, \lambda_\Psi)]$ i,e. from the $\Psi$ ranking. Now we know the edge incident to $v$ of the graph $G_z$ through which rank $\alpha$ is visible at terminal $v$ under $\Psi$ and hence at terminal $v_s$ under $\varphi$, we can hide this rank $\alpha$ from being visible at $v_s$ of $G_x$ by relabeling the edge incident to $v$ under $\Psi$ by the rank $\beta$. After this relabeling we update the list $L(\Psi, v)$ and $L_p(\varphi, v_s, r)$. If after this relabeling and updating, we have again any rank $\alpha < \lambda_\varphi$ such that $count(L_p(\varphi, v_s, r), \alpha) > \mathrm{c}$ then we repeat the process of relabeling and updating list $L(\Psi, v)$ and $L_p(\varphi, v_s, r)$ as described above until $count(L_p(\varphi, v_s, r), i) \leq \mathrm{c}$ for all rank $i \in R$ and we must reach such a situation because $L_p(\varphi, v_s, r) \preceq L_p(\varphi', v_s, r)$ and $count(L_p(\varphi', v_s, r), i) \leq \mathrm{c}$ for all rank $i \in R$..

Now in $R'_d(x)$ obtained from $R'_p(x)$, we keep only one pair $(\{L_p(\varphi', v_s, r), L_p(\varphi', v_t, r)\}, \lambda'_\varphi)$ after applying the elimination rule mentioned above for each individual list $L_p(\varphi', v_s, r)$ and $L_p(\varphi', v_t, r)$. Also we get a feasible vector $(\{L_p(\varphi, v_s, r), L_p(\varphi, v_t, r)\}, \lambda_\varphi)$ in $R_d(x)$, where either $L_p(\varphi, v_s, r) = L_p(\varphi', v_s, r)$ and $L_p(\varphi, v_t, r) \preceq L_p(\varphi', v_t, r)\}$ or $L_p(\varphi, v_t, r) = L_p(\varphi', v_t, r)$ and $L_p(\varphi, v_s, r) \preceq L_p(\varphi', v_s, r)$ in $R_d(x)$ extended from $R_d(y)$ and $R_d(z)$. Hence we can conclude that, $R_d(x) = R'_d(x)$

**Case 2 :** $G_x = G_y \parallel G_z$.

Let $v_s$ and $v_t$ be the terminals of $G_y$ and $G_z$. Then $G_x$ has $v_s$ and $v_t$ as terminals. We show that $R_d'(x) = R_d(x)$. Suppose that, there exists a feasible vector $R_p(\varphi')$ in $R_p'(x)$, obtained from $\eta'(y)$ and $\Psi''(z)$. Now from lemma 3.5 we have,

$L_p(\varphi', v_s, r) = L_p(\eta', v_s, r) \cup L_p(\Psi'', v_s, r)$; and

$L_p(\varphi', v_t, r) = L_p(\eta', v_t, r) \cup L_p(\Psi'', v_t, r)$.

Let $\eta(y)$ and $\Psi(z)$ be the two rankings in $R_d(y)$ and $R_d(z)$ such that,

(1) $L_p(\eta, v_s, r) = L_p(\eta', v_s, r)$, $L_p(\eta, v_t, r) \preceq L_p(\eta', v_t, r)$, and $\lambda_\eta = \lambda'_\eta$.

(2) $L_p(\Psi, v_s, r) = L_p(\Psi'', v_s, r)$, $L_p(\Psi, v_t, r) \preceq L_p(\Psi'', v_t, r)$, and $\lambda_\Psi = \lambda'_\Psi$.

Then according to our choice,

$$L_p(\varphi', v_s, r) = L_p(\eta', v_s, r) \cup L_p(\Psi'', v_s, r)$$

$$= L_p(\eta, v_s, r) \cup L_p(\Psi, v_s, r)$$

$$L_p(\varphi', v_s, r) = L_p(\varphi, v_s, r) \qquad (17)$$

$$L_p(\varphi', v_t, r) = L_p(\eta', v_t, r) \cup L_p(\Psi'', v_t, r)$$

$$\succeq L_p(\eta, v_t, r) \cup L_p(\Psi, v_t, r)$$

$$L_p(\varphi', v_t, r) \succeq L_p(\varphi, v_t, r) \qquad (18)$$

Since $\varphi'$ is a feasible vector, so if $\varphi$ is not a feasible vector, then it is always possible to convert $\varphi$ to a feasible vector by relabeling some edges as follows:

Since $\varphi'$ is a feasible vector and $\varphi$ is not a feasible vector, so there is a rank $i < \min\{r, \lambda_\varphi\}$ such that *count*($L_p(\varphi, v_t, r), i$) $> c$. Let $\alpha$ be the largest one among all such rank $i$. Then there is a rank $j$, $\alpha < j < \min\{r, \lambda_\varphi\}$ such that, $count(L_p(\varphi, v_t, r), j) < count(L_p(\varphi', v_t, r), j)$ and hence $count(L_p(\varphi, v_t, r), j) < c$. Let $\beta$ be the smallest rank among all such rank $j$. We also observe that the rank $\alpha$ in the list $L_p(\varphi, v_t, r)$ comes from both the lists $L_p(\eta, v_t, r)$ and $L_p(\psi, v_t, r)$. Now we know the edge incident to $v_t$ of the graph $G_z$ through which rank $\alpha$ is visible at terminal $v_t$ under $\Psi$ and hence at terminal $v_t$ under $\varphi$, we can hide this rank $\alpha$ from being visible at $v_t$ of $G_x$ by relabeling the edge incident to $v_t$ under $\Psi$ by the rank $\beta$. After this relabeling we update the lists $L_p(\psi, v_t, r)$ and $L_p(\varphi, v_t, r)$. If after this relabeling and updating, we have again any rank $\alpha < \min\{r, \lambda_\varphi\}$ such that $count(L_p(\varphi, v_t, r), \alpha) > c$ then we repeat the process of relabeling and updating the lists $L_p(\psi, v_t, r)$ and $L_p(\varphi, v_t, r)$ as described above until $count(L_p(\varphi, v_t, r), i) \leq c$ for all rank $i \in R$ and we must reach such a situation because $L_p(\varphi, v_t, r) \preceq L_p(\varphi', v_t, r)$ and $count(L_p(\varphi', v_t, r), i) \leq c$ for all rank $i \in R$..

Now in $R'_d(x)$ obtained from $R'_p(x)$, we keep only one pair $(\{L_p(\varphi', v_s, r), L_p(\varphi', v_t, r)\}, \lambda'_\varphi)$ after aplying the elemination rule mentioned above for each individual list $L_p(\varphi', v_s, r)$ and $L_p(\varphi', v_t, r)$. Also we get a feasible vector $(\{L_p(\varphi, v_s, r), L_p(\varphi, v_t, r)\}, \lambda_\varphi)$ in $R_d(x)$, where either $L_p(\varphi, v_s, r) = L_p(\varphi', v_s, r)$ and $L_p(\varphi, v_t, r) \preceq L_p(\varphi', v_t, r)\}$ or $L_p(\varphi, v_t, r) = L_p(\varphi', v_t, r)$ and $L_p(\varphi, v_s, r) \preceq L_p(\varphi', v_s, r)$ in $R_d(x)$ extended from $R_d(y)$ and $R_d(z)$. So we can conclude that, $R_d(x) = R'_d(x)$. □

Thus the dominance class $R_d(x)$ on node $x$ can be seen as a set of vectors of extensible $c$-edge-ranking of $G_x$.

## 3.5 An efficient Algorithm

The main result of this section is the following theorem.

**Theorem 3.1** *For any positive integer c, an optimal c-edge-ranking of a series-parallel graph G with n vertices can be found in time* $O(n^{2b+4} \log_{c+1}{}^4 n \log_2 \log_{c+1} n)$.

**Proof.** Let $T_b$ be a binary decomposition tree of $G$. We first give an algorithm to decide, for a given positive integer $l$, whether $G$ has a $c$-edge-ranking $\varphi$ with $\#\varphi \le l$. We use dynamic programming and bottom-up tree computation on the binary tree $T_b$ : for each node $x$ of $T_b$ rom leaves to the root, we construct all (dominance classes of) $c$-edge-ranking of $G_x$ from those of two subgraphs $G_y$ and $G_z$ associated with the children $y$ and $z$ of $x$. Then using binary search over the range of $l$, $1 \le l \le (1 + b \log_{c+1} n)$, we determine the minimum value of $l$ such that $G$ has a $c$-edge-ranking $\varphi$ with $l = \#\varphi$ and find an optimal $c$-edge-ranking of $G$.

By Lemmas 3.3 and 3.7 the dominance class $R_d(x)$ on $x$ can be seen as a set of vectors of extensible $c$-edge-ranking of $G_x$. Since $|R| = l$, $1 \le r \le l$, and $0 \le count(L_p(\varphi, v, r), i) \le c$ for a $c$-edge-ranking $\varphi$ and a rank $i \in R$, the number of distinct peer-lists $L_p(\varphi, v, r)$ is at most $l.(c+1)^l$ for each terminal $v \in S_x$ of $G_x$. On the otherhand, the number of distinct obstacles $\lambda_\varphi$ is at most $l$. Then by the definition of $R_d(x)$ the number of different dominant vectors on node $x$ is at most $l^2.2(c+1)^l$, since $|S_x| = 2$. One may assume that $c \le |E| \le 2n - 3$ [24] and $l \le 1 + b \log_{c+1} n = O(\log_{c+1} n)$ by Lemma 3.2. Therefore the total number of dominant vectors on node $x$ is $O(n^{b+1} \log_{c+1}{}^2 n)$.

The main step of our algorithm is to compute a table of all dominant vectors on the root of $T_b$ by means of dynamic programming and bottom-up tree computation on $T_b$. If the table is non empty, then the series-parallel graph $G$ corresponding to the root of $T_b$ has a $c$-edge-ranking $\varphi$ such that $\#\varphi \leq l$.

We first show how to find the table of all dominant vectors $R_d(x)$ on a leaf $x$ of $T_b$. This can be done as follows:

(1) enumerate all edge-labelings $\varphi : E_x \rightarrow R$ of $G_x$; and

(2) compute the table of all dominant vectors $R_d(x)$ on $x$ from the edge-labelings $\varphi$ of $G_x$.

Since every leaf of $T_b$ represents a subgraph of $G$ induced by an edge $e \in E$, and $| R | = l$, the number of edge-labelings $\varphi$: $E_x \rightarrow R$ is at most $l$. For each labeling, $\lambda_\varphi$ can be computed in time $O(1)$. Furthermore, the list $L_p(\varphi, v, r)$, $v \in S_x$ can be computed in time $O(l)$. Then checking whether an edge-labeling $\varphi$ is a $c$-edge-ranking of $G_x$ can be done by Lemma 2.1 in time $O(1)$, and if so computing $R_p(\varphi)$ can be done in time $O(1)$. Therefore steps (1) and (2) can be executed for the leaf in time $O(l) = O(\log_{c+1} n)$, and hence the table on $x$ can be found in time $O(\log_{c+1} n)$.

We next show how to compute all dominant vectors on an internal node $x$ of $T_b$ from those on two children $y$ and $z$ of $x$. The dominant vector on an internal node $x$ can be obtained from the cross product of the dominant vectors of the two children $y$ and $z$ of $x$. The cardinality of the resultant vector table is $O(n^{2b+2} \log_{c+1}^4 n)$. Then $\lambda_\varphi$ for each dominant vector can be computed in time $O(1)$. The peer-list can be computed by Lemmas 3.4 and 3.5 in time $O(l)$. The checking whether an edge-labeling $\varphi$ is a $c$-edge-ranking of $G_x$ can be done by Lemma 2.1 in time $O(1)$. If an edge-labeling corresponding to a dominant vector is not a $c$-edge-ranking, then the edge-labeling can be converted to a $c$-edge-ranking by algorithm UPDATE (given later) in time $O(n)$. Therefore, the table of all dominant vectors on an internal node can be computed in time $O(n^{2b+3} \log_{c+1}{}^4 n)$, since $l = O(\log_{c+1} n)$.

We thus have the following algorithm CHECK to determine whether $G$ has a $c$-edge-ranking $\varphi$ with $\#\varphi \le l$ for a positive integer $l$.

```
Algorithm CHECK;
begin
1   obtain a binary decomposition tree $T_b$ of a series-parallel graph $G$;
2   compute a table of all dominant vectors on each leaf node $x$ of $T_b$;
3   for each internal node $x$ of $T_b$, compute a table of all vectors from the dominant vectors on
    the two children $y$ and $z$ of $x$;
4   if possible then
        convert these vectors to feasible vectors by calling the algorithm UPDATE [Algorithm
        UPDATE will be given later];
5   from the table of all feasible vectors, construct a table of dominant vectors by applying the
    elimination rule described in section 3.4;
6   repeat Line 3 to 5 for all internal nodes of $T_b$ upto the root;
7   check whether there exists a dominant vector in the table with $r = \lambda_\varphi$ at the root;
end.
```

Line 1 can be done in time $O(n)$ [2]. Line 2 can be done for each leaf in $O(\log_{c+1} n)$ time. Since there are $O(n)$ leaves, Line 1 cane be done in $O(n \log_{c+1} n)$ time in total for all leaves. As mentioned above, Lines 3, 4 and 5 can be done in $O(n^{2b+3} \log_{c+1}{}^4 n)$ time per node. Since Lines 3, 4 and 5 are executed for $O(n)$ nodes in total in Line 6, Line 6 can be done in

$O(n^{2b+4} \log_{c+1}^4 n)$ time in total. Line 7 can be done in $O(n^{b+1} \log_{c+1}^2 n)$ time. Thus checking whether a series-parallel graph $G$ has a $c$-edge-ranking $\varphi$ such that $\#\varphi \leq l$ can be done in time in $O(n^{2b+4} \log_{c+1}^4 n)$ time in total.

Using binary search technique over the range of $l$, $1 \leq l \leq 1+b \log_{c+1} n = O(\log_{c+1} n)$, one can find the smallest integer $r'_c(G)$ such that $G$ has a $c$-edge-ranking $\varphi$ such that $\#\varphi = r'_c(G)$ by calling CHECK $O(\log_2 \log_{c+1} n)$ times. Therefore an optimal $c$-edge-ranking of a series-parallel $G$ of $n$ vertices can be found in time $O(n^{2b+4} \log_{c+1}^4 n \log_2 \log_{c+1} n)$ for any positive integer $c$. □

We now present the algorithm UPDATE which converts infeasible peer-vectors to feasible peer-vectors if some exists as described in Lemma 3.7.

```
Algorithm UPDATE($R_p(\varphi)$);
begin
8  do while (true)
9    if $R_p(\varphi)$ is a feasible vector then
10       begin
11          keep $R_p(\varphi)$ in the table of feasible vectors on node $x$;
12          return;
13       end
14   else
15       begin
16          if $x$ is a series node in $T_b$ then
17             begin
18                if there is a rank $i$, $\lambda_\psi \leq i < \lambda_\varphi$, such that $count(L_p(\varphi, v_t, r), i) > c$ then
19                   begin
20                      let $\alpha$ be the largest one among all such ranks $i$;
21                      if there is no rank $j$, $\alpha < j < \lambda_\varphi$, such that $count(L_p(\varphi, v_t, r), j) < c$ then
22                         return; [$R_p(\varphi)$ is not a feasible vector]
```

```
23                else
24                    begin
25                        let $\beta$ be the smallest integer among all ranks $j$;
26                        find an edge incident to $v$ of the graph $G_y$ through which rank $\alpha$ is
                          visible at terminal $v$ under $\eta$;
27                        relabel the edge by the rank $\beta$; [thus hide all ranks $< \beta$ including $\alpha$
                          from being visible at terminal $v$ under $\eta$]
28                        update list $L_p(\eta, v, \lambda_\eta)$ and the vector $R_p(\varphi)$;
29                    end
30            end
31        if there is a rank $i$, $\lambda_\eta \le i < \lambda_\varphi$, such that $count(L_p(\varphi, v_s, r), i) > c$ then
32            begin
33                let $\alpha$ be the largest one among all such ranks $i$;
34                if there is no rank $j$, $\alpha < j < \lambda_\varphi$, such that $count(L_p(\varphi, v_s, r), j) < c$ then
35                    return; [$R_p(\varphi)$ is not a feasible vector]
36                else
37                    begin
38                        let $\beta$ be the smallest integer among all ranks $j$;
39                        find an edge incident to $v$ of the graph $G_z$ through which rank $\alpha$ is
                          visible at terminal $v$ under $\Psi$;
40                        relabel the edge by the rank $\beta$; [thus hide all ranks $< \beta$ including $\alpha$
                          from being visible at terminal $v$ under $\Psi$]
41                        update list $L_p(\Psi, v, \lambda_\Psi)$ and the vector $R_p(\varphi)$;
42                    end
43            end
44    end
45  else [$x$ is a parellel node]
46    begin
47        if there is a rank $i$, $i < \min\{r, \lambda_\varphi\}$ such that $count(L_p(\varphi, v_t, r), i) > c$ then
48            begin
49                let $\alpha$ be the largest one among such rank $i$;
50                if there is no rank $j$, $\alpha < j < \min\{r, \lambda_\varphi\}$, such that $count(L_p(\varphi, v_t, r), j) < c$ then
```

```
                  return; [$R_p(\varphi)$ is not a feasible vector]
               else
                  begin
                     let $\beta$ be the smallest integer among all ranks $j$;
                     find the edge incident to $v_t$ of the graph $G_z$ through which rank $\alpha$ is visible at terminal $v_t$ under $\psi$;
                     relabel the edge by the rank $\beta$; [thus hide all ranks < $\beta$ including $\alpha$ from being visible at terminal $v_t$ under $\psi$]
                     update list $L_p(\psi, v_t, r)$ and the vector $R_p(\varphi)$;
                  end
               end
         if there is a rank $i$, $i < \min\{r, \lambda_\varphi\}$ such that $count(L_p(\varphi, v_s, r), i) > c$ then
            begin
               let $\alpha$ be the largest one among such ranks $i$;
               if there is no rank $j$, $\alpha < j < \min\{r, \lambda_\varphi\}$, such that $count(L_p(\varphi, v_s, r), j) < c$ then
                  return; [$R_p(\varphi)$ is not a feasible vector]
               else
                  begin
                     let $\beta$ be the smallest integer among all rank $j$;
                     find the edge incident to $v_s$ of the graph $G_z$ through which rank $\alpha$ is visible at terminal $v_s$ under $\psi$;
                     relabel the edge by the rank $\beta$; [thus hide all ranks < $\beta$ including $\alpha$ from being visible at terminal $v_s$ under $\psi$]
                     update list $L_p(\psi, v_s, r)$ and the vector $R_p(\varphi)$;
                  end
            end
      end
   end
enddo
end.
```

## 3.6 conclusion

We have presented an efficient algorithm for the $c$-edge-ranking of series-parallel graphs. We introduced some new definitions and reduced the table size of dynamic programming by using the concept of dominance class. This enables us to find an optimal $c$-edge-ranking of series-parallel graphs in $O(n^{2b+4} \log_{c+1}{}^{4} n \log_2 \log_{c+1} n)$ time.

# Chapter 4

# Conclusion

This thesis deals with the generalized edge-ranking problem for series-parallel graphs. Since series-parallel graphs arises in many practical cases in VLSI and in electrical circuits, so an efficient algorithm for this particular class of graphs will have many applications in real world.

In Chapter 1 we have introduced the ranking problem and its significance. In Chapter 2, we have characterized the $c$-edge-ranking of series-parallel graphs by the number of visible edges. This characterization helps us to find an optimal $c$-edge-ranking of series parallel graphs.

In Chapter 3, at first we have given an upper bound on the $c$-edge-ranking number of series-parallel graphs. We next present an efficient polynomial-time algorithm for finding an optimal $c$-edge-ranking of a given series-parallel graph with bounded maximum degree $\Delta$ for any positive integer $c$. We have improved the time complexity of our algorithm by using the concept of dominance class.

Our algorithm can be used for $f$-edge-ranking using some modifications. An $f$-edge-ranking of a graph $G$ is a labeling of the edges of $G$ with integers such that, for any label $i$, deletion of all edges with labels $> i$ leaves connected components, each having at most $f(i)$ edges with label $i$. Clearly a $c$-edge-ranking is a special case of an $f$-edge-ranking in which $f(i) = c$ for every rank $i$.

In this thesis, we have given an efficient sequential algorithm to solve the $c$-edge-ranking problem of series-parallel graphs. However, the parallel algorithm for solving the $c$-edge-ranking problem of series-parallel graphs is yet to be explored.

# References

[1] H. L. Bodlaender, Polynomial algorithms for graph isomorphism and chromatic index on partial $k$-trees, *Journal of Algorithms*, 11 (1990), pp. 631–643.

[2] H. L. Bodlaender, A linear-time algorithm for finding tree-decompositions of small treewidth, *SIAM Journal on Computing*, 25 (1996), pp. 1305–1317.

[3] H. L. Bodlaender, J. S. Deogun, K. Jansen, T. Kloks, D. Kratsch, H. Müller, and Zs. Tuza, Rankings of graphs, *SIAM Journal on Discrete Mathematics*, 21 (1998), pp. 168–181.

[4] H. L. Bodlaender and T. Kloks, Efficient and constructive algorithms for pathwidth and treewidth of graphs, *Journal of Algorithms*, 21 (1996), pp. 358–402.

[5] R. Borie, Generation of polynomial-time algorithms for some optimization problems on tree decomposable graphs, *Algorithmica*, 14 (1995), pp. 123–137.

[6] P. de la Torre, R. Greenlaw, and T. M. Przytycka, Optimal tree ranking is in NC, *Parallel Processing Letters*, 2 (1992), pp. 31–41.

[7] P. de la Torre, R. Greenlaw, and A. A. Schäffer, Optimal edge-ranking of trees in polynomial time, *Algorithmica*, 13 (1995), pp. 592–618.

[8] 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.

[9] A. V. Iyer, H. D. Ratliff and G. Vijayan, Optimal node ranking of trees, *Information Processing Letters*, 28 (1988), pp. 225–229.

[10] 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.

[11] M. A. Kashem, X. Zhou, and T. Nishizeki, Algorithms for generalized vertex-ranking of partial *k*-trees, *Proc. of the 3rd Annual International Computing and Combinatorics Conference (COCOON'97), Lecture notes in Computer Science, Springer-Verlag*, 1276 (1997), pp. 212–221.

[12] M. A. Kashem, X. Zhou and T. Nishizeki, Algorithms for generalized edge-ranking of partial *k*-trees with bounded maximum degree, *Proc. of the International Conference on Computer and Information Technology*, (ICCIT '98), pp. 45-51.

[13] T. W. Lam and F. L. Yue, Edge-ranking of graph is hard, *Discrete Applied Mathematics*, 85 (1998), 71–86.

[14] T. W. Lam and F. L. Yue, Optimal edge-ranking of trees in linear time, *Proc. of the Ninth ACM-SIAM Symposium on Discrete Algorithm* (1998), pp. 436–445.

[15] J. W. H. Liu, The role of elimination trees in sparse factorization, *SIAM Journal of Matrix Analysis and Applications*, 11 (1990), pp. 134–172.

[16] N. Megiddo, Applying parallel computation algorithms in the design of serial algorithms, *Journal of the ACM*, 30 (1983), pp. 852–865.

[17] A. Pothen, The complexity of optimal elimination trees, *Technical report CS-88-13, Pennsylvania State University, USA*, 1988.

[18] B. A. Reed, Finding approximate separators and computing tree-width quickly, *Proc. of the 24th Annual ACM Symp. on Theory of Computing*, 1992, pp. 221–228.

[19] R. Robertson and P. D. Seymour, Graph minors, II. Algorithmic aspect of tree-width, *Journal of Algorithms*, 7 (1986), pp. 309–322.

[20] A. A. Schäffer, Optimal node ranking of trees in linear time, Information Processing Letters, 33 (1989/90), pp. 91–96.

[21] J. van Leeuwen, Graph algorithms, In Handbook of theoretic Computer Science, *A: Algorithms and Complexity Theory*, Amsterdam, North Holland Pub. Company, 1990, pp. 527–631.

[22] X. Zhou, M. A. Kashem, T. Nishizeki, Generalized edge-ranking of trees, *Proc. of the 22nd International Workshop on Graph-Theoretic Concepts in Computer Science (WG '96), IEICE Trans. on Fundamentals of Electronics, Communications and Computer Science*, E81–A, 2 (1998), pp. 310–320.

[23] X. Zhou, H. Nagai, and T. Nishizeki, Generalized vertex-ranking of trees, *Information Processing Letters*, 56 (1995), pp. 321–328.

[24] X. Zhou, H. Suzuki and T. Nishizeki, A linear algorithm for edge-coloring of series-parallel multigraphs, *Jurnal of Algorithms* 20, 174–201 (1996).