Parallel Algorithm for Generalized Vertex-Colorings of Partial *k*-Trees

by

Mohammed Eunus Ali

Department of Computer Science and Engineering

Bangladesh University of Engineering and Technology

October 2002

Submitted to
Bangladesh University of Engineering and Technology

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

PARALLEL ALGORITHM FOR GENERALIZED VERTEX-COLORINGS OF PARTIAL *K*-TREES

A Thesis submitted by

MOHAMMED EUNUS ALI
Student No. 040005033P
for the partial fulfillment of the degree of
M. Sc. Engineering (Computer Science and Engineering).
Examination held on October 08, 2002.

Approved as to style and contents by:

[illegible]

Dr. Md. Abul Kashem Mia Associate Professor & Head Department of Computer Science and Engineering B.U.E.T., Dhaka – 1000, Bangladesh.	Chairman, Supervisor and Ex-officio

[illegible]

Dr. M. Kaykobad Professor Department of Computer Science and Engineering B.U.E.T., Dhaka – 1000, Bangladesh.	Member

[illegible]

Dr. Chowdhury Mofizur Rahman Professor Department of Computer Science and Engineering B.U.E.T., Dhaka – 1000, Bangladesh.	Member

[illegible]

Dr. Md. Kamrul Hasan Associate Professor Department of Electrical and Electronics Engineering B.U.E.T., Dhaka – 1000, Bangladesh.	Member (External)

Certificate

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

[illegible]

Signature of the Supervisor

[illegible]

Signature of the Author

Parallel Algorithm for Generalized Vertex-Colorings of Partial *k*-Trees

by

Mohammed Eunus Ali

Department of Computer Science and Engineering

Bangladesh University of Engineering and Technology

October, 2002

List of Figures

List of Tables

Acknowledgement

First and foremost, I would like to acknowledge my gratitude to Dr. Md. Abul Kashem Mia, Associate Professor and Head, Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology. His constant supervision, scholarly guidance, valuable advice and encouragement have been a great impetus to this thesis work. His vast experience and in-depth knowledge in graph theory and parallel algorithms have helped significantly to achieve a smooth completion of this thesis work.

I would like to thank the members of the graduate committee, Dr. M. Kaykobad, and Dr. Chowdhury Mofizur Rahman, Professors, Department of Computer Science and Engineering, BUET, and Dr. Md. Kamrul Hasan, Associate Professor, Department of Electrical and Electronic Engineering, BUET, for their valuable suggestions.

I would like to acknowledge the all-out co-operation and services rendered by the faculty members and staffs of the CSE Department. Finally, I express my ever gratefulness to all who contributed to this thesis work.

Abstract

In this thesis we present an efficient parallel algorithm for solving the generalized vertex-coloring (l-vertex-coloring) problem on partial k-trees. Let l be a positive integer, and let G be a graph with nonnegative integer weights on edges. Then the optimal l-vertex-coloring problem on G is an assignment of colors to the vertices of G in such a way that any two vertices u and v in G get different colors if the distance between u and v is at most l and the number of colors used is as small as possible. The l-vertex-coloring problem has applications in various scheduling problems. In this paper we give a parallel algorithm to find an l-vertex-coloring of a partial k-tree G with the minimum number of colors. Our algorithm takes $O(\log_2 n)$ parallel time using $O\left(\alpha \times (l+2)^{6\alpha(k+1)} \times n + n^3\right)$ operations on the common CRCW PRAM model, where n is the number of vertices in G and α is the number of colors in the color-set. The previously known best algorithm takes $O(\log_2 n)$ parallel time using $O(n(\alpha+1)^{2^{6(k+1)(l+2)}} + n^3)$ operations on the common CRCW PRAM.

Chapter 1

Introduction

In this chapter we provide the necessary background and motivation for the study on the coloring of graphs. In Section 1.1, we give a historical background of the development of coloring of graphs. In Section 1.2, we give a generalization of ordinary vertex-coloring problem and provide the necessary motivation for generalization. Finally Section 1.3 summarizes our results together with the known ones.

1.1 Background

Recent research efforts in algorithm theory have concentrated on designing efficient algorithms for solving combinatorial problems, particularly graph problems. A graph $G = (V, E)$ with n vertices and m edges consists of a vertex set $V = \{v_1, v_2, \ldots, v_n\}$ and an edge set $E = \{e_1, e_2, \ldots, e_m\}$, where an edge in E joins two vertices in V. Figure 1.1 depicts a graph of 7 vertices and 9 edges, where vertices are drawn by circles, edges by lines,

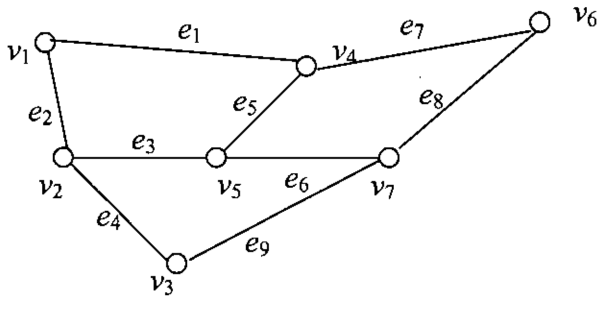

Figure 1.1: A graph with 7 vertices and 9 edges.

with vertex names next to the vertices and edge names next to the edges. Efficient algorithms have been obtained for various graph problems, such as coloring problems, planarity testing problem and maximum flow problem.

1.1.1 Vertex-coloring Problem

The vertex-coloring problem is one of the most fundamental problems on graphs. A *vertex-coloring* of a graph G is an assignment of colors to the vertices of G in such a way that any two adjacent vertices get different colors [Wes99]. The *vertex-coloring problem* is to find a vertex-coloring that requires the minimum number of colors. The minimum number of colors needed to vertex-color a graph is called the *chromatic number* of the graph G and is denoted by $\chi(G)$. Figure 1.2 depicts an optimal vertex-coloring of a graph G using 3 colors, where the colors are written next to the vertices.

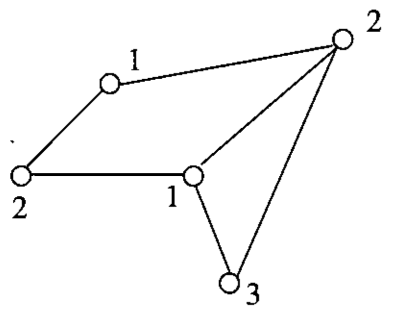

Figure 1.2: An optimal vertex-coloring of a graph G.

Vertex coloring arises in a variety of scheduling and clustering applications. Compiler optimization is the canonical application for coloring, where we seek to schedule the use of a finite number of registers. In a program fragment to be optimized, each variable has a range of times during which its value must be kept intact, in particular, after it is initialized and before its final use. Any two variables whose life spans intersect cannot be placed in the same register. A graph can be constructed where there is a variable associated with each vertex and an edge between any two vertices indicates that the variable life spans intersect. A coloring of the vertices of this graph assigns the variables to classes such that two variables with the same color do not clash and so can be assigned to the same register. The most important application of vertex-coloring is in scheduling of any kind. If the vertices of a graph G represent a set of university courses, with edges between courses with common students, then the chromatic number is the minimum number of periods needed to schedule examinations without conflicts. One of the most famous problems in graph theory also involves coloring. It is known as 4-color problem. A map is a partition of the plane into connected regions. It requires determining whether the regions of every map can be colored using at most four colors so that no two neighboring regions have the same color.

1.2 Generalized Vertex-Coloring

There are many generalizations of ordinary vertex-coloring. In this section we describe a generalized vertex-coloring, called an l-vertex-coloring [ZKN00].

1.2.1 l-vertex-coloring

A natural generalization of the ordinary vertex coloring is the l-vertex-coloring. Let l be a positive integer and G be a graph with positive integer weights on all its edges. Then an *l-vertex-coloring* of G is an assignment of colors to the vertices of G in such a way that any two vertices u and v in G get different colors if $dist(u,v) \leq l$, where $dist(u,v)$ is the length of the shortest path between u and v in G. Clearly an ordinary vertex-coloring is an l-vertex-coloring. Figure 1.3 shows a 3-vertex-coloring of a graph G using 2 colors, where colors are drawn next to the vertices and edge-weights are given next to the edges.

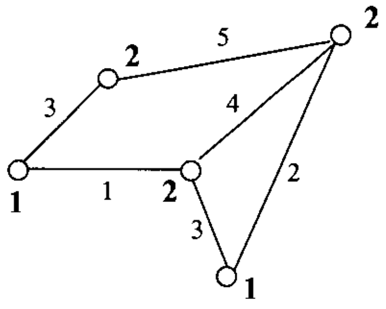

Figure 1.3: A 3-vertex-coloring of a graph using 2 colors.

The minimum number of colors needed for an l-vertex-coloring of a graph G is called the *l-chromatic number* of G and is denoted by $\chi_l(G)$. An l-vertex-coloring of G using $\chi_l(G)$ colors is called an *optimal l-vertex-coloring* of G. The *l-vertex-coloring problem* is to find an optimal l-vertex-coloring of a given graph G.

Since the ordinary vertex-coloring problem is NP-hard [GJ79], the l-vertex-coloring problem is NP-hard in general [ZKN00]. So it is very unlikely that there exists an polynomial time algorithm to solve the l-vertex-coloring problem for general graphs [GJ79]. However, Zhou *et al.* presented a polynomial-time algorithm to solve the l-vertex-coloring problem for partial k-trees, that is, graphs of tree-width bounded by a fixed integer k [ZKN00]. Their sequential algorithm takes $O(n(\alpha+1)^{2^{2(k+1)(l+2)+1}}+n^3)$ time. They claimed that the same problem can be solved in $O(\log_2 n)$ parallel time using $O(n(\alpha+1)^{2^{6(k+1)(l+2)}}+n^3)$ operations on the common CRCW PRAM. In this paper, we show that the l-vertex-coloring problem for a partial k-tree can be solved in $O(\log_2 n)$ parallel time using $O\left(\alpha \times (l+2)^{6\alpha(k+1)} \times n + n^3\right)$ operations on the common CRCW PRAM model, where n is the number of vertices in G and α is the number of colors in the color-set. The l-vertex-coloring problem on a weighted graph $G = (V, E)$ can be easily reduced to the ordinary vertex-coloring problem on a new non-weighted graph $G_l = (V, E_l)$ such that $(u,v) \in E_l$ for any two vertices u and v in V if and only if $dist(u, v) \leq l$ in G. Therefore, one may expect that the l-vertex-coloring problem for a partial k-tree G can be efficiently solved by applying a linear-time algorithm [BPT92] to solve an ordinary vertex-coloring problem for partial k-tree. However, it is not the case, because G_l is not always a partial k-tree.

The applications of the l-vertex-coloring are all the applications of vertex-coloring. The examination-scheduling problem in Section 1.1.1 can be generalized now so that there is a gap of l periods between any two examinations of courses having common students.

1.3 Summary

This thesis gives a parallel algorithm for solving l-vertex-coloring problem on partial k-trees. In this section, we summarize our main results. The known results along with our new results are given in Table 1.1.

Classes of Graphs	Algorithm	Time	Operations	PRAM model	Ref.
Partial k-trees	Sequential	$O(n(\alpha+1)^{2^{2(k+1)(l+2)+1}}+n^3)$	--	--	Zhou et al. (IEICE 2000)
Partial k-trees	parallel	$O(\log_2 n)$	$O(n(\alpha+1)^{2^{6(k+1)(l+2)}}+n^3)$	CRCW	Zhou et al. (IEICE 2000)
Partial k-trees	parallel	$O(\log_2 n)$	$O(n\alpha(l+2)^{6\alpha(k+1)}+n^3)$	CRCW	Ours

Table 1.1: Known results for solving l-vertex-coloring problem along with our new results.

The symbol α in the table 1.1 stands for the number of colors used in the l-vertex-coloring.

The thesis is organized as follows. Chapter 2 gives preliminary definitions. We give an efficient parallel algorithm for l-vertex-coloring problem on k-trees in Chapter 3. Finally Chapter 4 concludes with the results and future works.

Chapter 2

Preliminaries

In this chapter we present some basic terms and easy observations. Definitions that are not included in this chapter will be introduced, as they are needed. In Section 2.1, we start with some definitions of the standard graph-theoretical terms used throughout the thesis. In Section 2.2 we discuss about properties of trees. In Section 2.3 we define parallel computer models most widely used in parallel algorithm development and analysis. In Section 2.4 we discuss about partial k-tree and decomposition of k-tree. Finally in Section 2.5 we show a way to transform a rooted tree into an equivalent binary tree so that we can apply bottom-up algorithm on the equivalent binary tree.

2.1 Basic Terminology

2.1.1 Graphs and Multigraphs

A *graph* is a structure (V, E) which consists of a finite set of vertices V and a finite set of edges E; each edge is an unordered pair of distinct vertices. We call $V(G)$ the *vertex-set* of graph G and $E(G)$ the *edge-set* of G. Throughout this thesis, the number of vertices of G is denoted by n, that is $n = |V|$. If $e = (v, w)$ is an edge, then e is said to join the vertices v and w and these vertices are 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 the 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 *graph*, if doing so creates no confusion.

2.1.2 Subgraphs

A *subgraph* of a graph $G = (V, E)$ is graph $G' = (V', E')$ such that $V' \subseteq V$ and $E' \subseteq E$; we write this as $G' \subseteq G$. If G' contains all the edges of G that join two vertices in V'

then G' is said to be the *subgraph induced by* V' and is denoted by $G[V']$. If V' consists of exactly the vertices on which edges in E' are incident then G' is said to be the *subgraph induced by* E' and is denoted by $G[E']$.

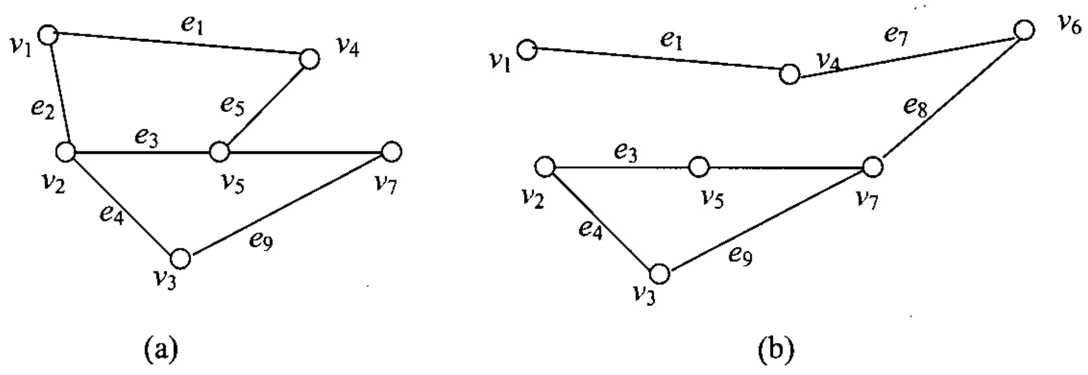

Figure 2.1: Subgraphs of G in Fig. 1.1: (a) vertex-induced subgraph, and (b) edge induced subgraph.

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 a 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 subgraph of G obtained by deleting the vertices in V' and 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 e. More generally, if $E \subseteq E'$ then $G - E'$ is a subgraph of G obtained by deleting the edges in E'.

2.1.3 Weighted Graphs

A graph where each of the edges has a positive weight associated with it is called a *weighted graph.* Now if N is the set of all positive integers then we can define the weight function for the edges as w: $E \rightarrow N$. Fig. 2.2 shows a weighted graph with five vertices and six edges.

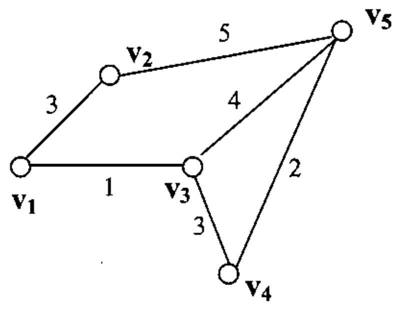

Figure 2.2: A weighted graph with five vertices and six edges.

2.1.4 Paths and Distances

A v_0–v_l walk in G is an alternating sequence of vertices and edges of G, $v_0, e_1, v_1, \ldots, v_{l-1}, e_l, v_l$, 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, $v_1, \ldots, v_l$ are distinct (except possibly v_0, v_l), then the walk is called a *path* and is usually denoted by $v_0\, v_1 \ldots v_l$. The *length* of a path in an unweighted graph 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 of length at least one is called a *cycle* where $v_0 = v_l$ is the only vertex repetition.

In a weighted graph, the length of a path is determined by weights of the edges constituting the path. So the length $w(P)$ of a path P is defined as $w(P) = \sum_{e \in P} w(e)$.The distance between any two vertices in a graph is the length of the shortest path in the graph between the two vertices. We denote the distance from a vertex u to another vertex v by $dist(u,v)$. Now if the shortest path in G from u to v is P, then $dist(u,v) = w(P)$.

2.2 Trees

A (free) tree is a connected graph without any cycles. We often omit the word "free" when we say that a graph is a tree. Fig 2.2 is an example of a tree. A rooted tree is a free tree in which one of the nodes is distinguished from others. This distinguished node is called the *root* of the tree. The root of the tree is generally drawn at the top. In Fig. 2.3, the root is node 1.

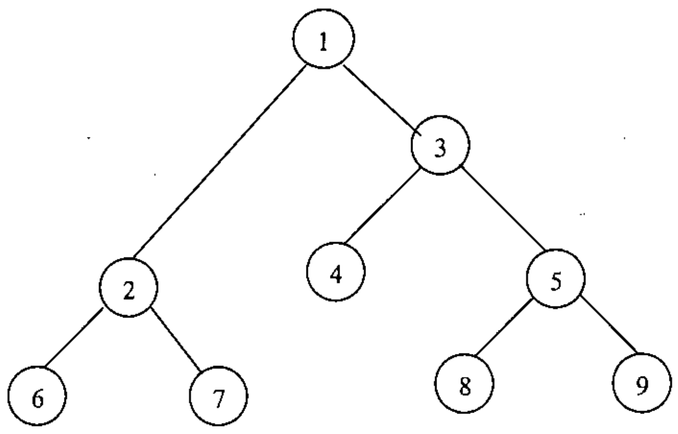

Figure 2.3: A tree with 9 vertices.

Every vertex u other than the root is connected by an edge to some other vertex p called the *parent* of u. We also call u a *child* of p. We draw the parent of a node above that node. For example, in Fig. 2.3, node 1 is the parent of node 2 and node 3. Alternately, nodes 6 and 7 are children of node 2. A *leaf* is a node of a tree that has no child. Thus every node of a tree is either a leaf or an internal node, but not both. In Fig. 2.3, the leaves are 4, 6, 7, 8 and 9 and nodes 1, 2, 3 and 5 are internal nodes.

The parent-child relationship can be extended naturally to ancestors and descendants. Suppose, u_1, u_2, …, u_l is a sequence of nodes in a tree such that u_1 is the parent of u_2, which is the parent of u_3 and so on. The node u_1 is called an *ancestor* of u_l and node u_l is called a *descendant* of u_1. The root is the ancestor of every node in a tree and every node is a descendant of the root. In Fig. 2.3, all nodes other than node 1 are descendants of node 1 and node 1 is an ancestor of all other nodes.

In a tree T, a node u together with all of its descendants, if any, is called a *subtree* of T. Node u is the root of this subtree. Referring again to Fig. 2.3, node 6 by itself is a subtree, since node 6 has no descendant. Again, nodes 2, 6 and 7 form a subtree with root 2. Finally the entire tree of Fig. 2.3 is a subtree of itself with root 1. The height of a node u in a tree is the length of the longest path from u to a leaf under u. The height of a tree is the height of a root. The depth of a node u in the tree is the length of a path from the root to u. In Fig 2.3, for example, node 3 is of height 2 and depth 1. The tree has height 3.

2.3 PRAM Models

The RAM model has been used successfully to predict the performance of sequential algorithms. The PRAM model is a natural extension of the basic sequential model [Jos 92]. The PRAM model consists of a number of processors, each of which has its own local memory and can execute its own local program. The processors communicate by exchanging data through a shared memory unit. Each processor is uniquely identified by an index, called a *processor id.* All the processors operate synchronously under the control of a common clock. Fig. 2.4 shows a general view of a PRAM model with p processors. These processors are indexed 1, 2, ..., p. Shared memory is also referred to as global memory.

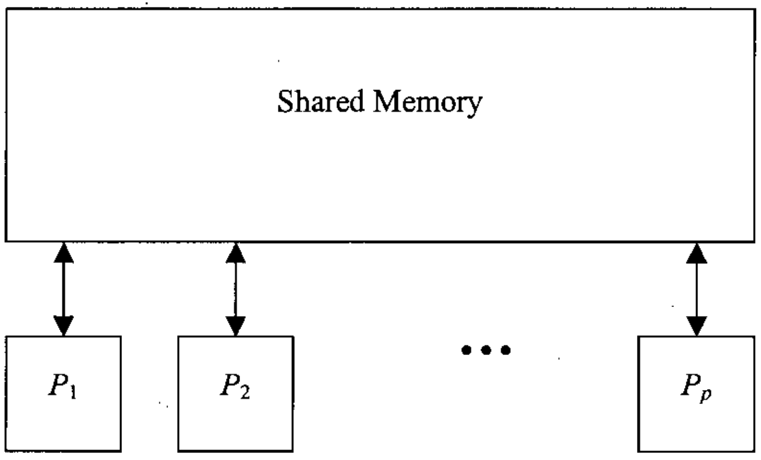

Figure 2.4: The PRAM model.

There are two basic modes of operation of a shared-memory model. In *asynchronous* mode, each processor operates under a separate clock and it is the programmer's responsibility to set appropriate synchronization points whenever necessary. In *synchronous* mode, all the processors operate synchronously under the control of a common clock. A standard name for the synchronous shared-memory model is the *parallel random-access machine* (PRAM) model. Since each processor can execute its own local program, the shared memory model is a *multiple instruction multiple data* (MIMD) type. That is, each processor may execute an instruction and operate on data different from those executed or operated on by any other processor during any given time unit.

There are several variations of the PRAM model based on the assumption regarding the handling of the simultaneous access of several processors to the same location of the shared-memory. The *exclusive read exclusive write* (*EREW*) PRAM does not allow any simultaneous access to a single memory location. The *concurrent read exclusive write* (*CREW*) PRAM allows simultaneous access for a read instruction only. The *concurrent read concurrent write* (*CRCW*) allows simultaneous access for a read or a write instruction. The three principal varieties of CRCW PRAMs are differentiated by how concurrent writes are handled. The *common* CRCW PRAM allows concurrent writes only when all processors are attempting to write the same value. The *arbitrary* CRCW PRAM allows an arbitrary processor to succeed. The *priority* CRCW PRAM assumes that the indices of the processors are linearly ordered and allows the one with minimum index (or the maximum index) to succeed.

2.4 Partial *k*-Trees

A k-tree can be recursively defined as follows [Bod90]:

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

A graph is called a *partial k-tree* if it is a subgraph of a k-tree. Thus a partial k-tree $G = (V, E)$ is a simple graph, and $|E| < kn$. The treewidth of a graph G is the minimum integer k such that G is a partial k-tree.

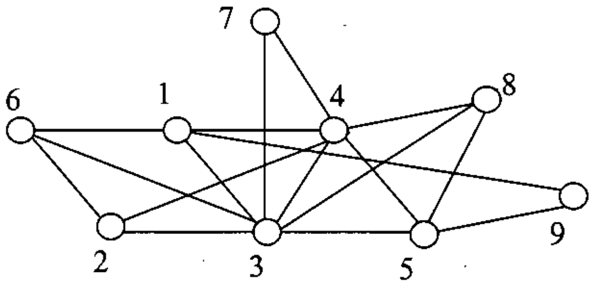

Figure 2.5: A partial 3-tree.

A *tree-decomposition* of a graph $G = (V, E)$ is a pair (T, S), where $T = (V_T, E_T)$ is a tree and $S = \{X_x \mid x \in V_T\}$ is a collection of vertex-set of V satisfying the following three conditions [RS86]:

- $\bigcup_{x \in V_T} X_x = V$;
- for every edge $e = (v, w) \in E$, there exists a node $x \in V_T$ with $v, w \in X_x$; and
- for all $x, y, z \in V_T$, if node y lies on the path from node x to z in T, then $X_x \cap X_z \subseteq X_y$.

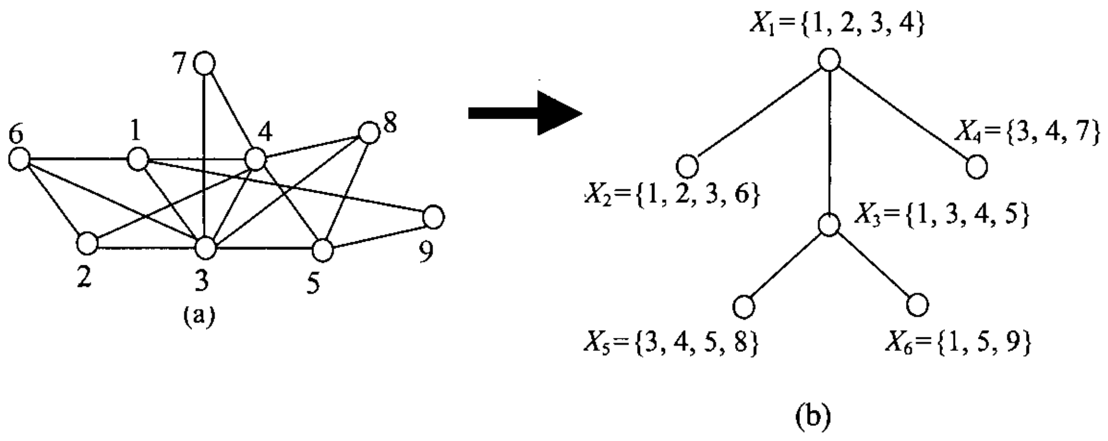

Figure 2.6 (a) A partial 3-tree, and (b) a tree-decomposition of the graph in (a).

Figure 2.6(b) depicts a tree-decomposition of a graph in Fig. 2.6(a). The *width of a tree-decomposition* (T, S) is $\max_{x \in V_T} |X_x| - 1$. The *tree-width of a graph* G is the minimum width of a tree-decomposition of G, taken over all possible tree-decomposition of G. A graph G with tree-width $\leq k$ is called a *partial k-tree*. Every partial k-tree G has a tree-decomposition (T, S) with width $\leq k$ and $n_T \leq n$, where n_T is the number of nodes in T [Bod96]. We then cite the following theorem from [BH95, BK96, Ree92].

Theorem 2.1 *It can be checked whether the tree width of a graph G is k and find a corresponding tree-decomposition in $O(\log_2^2 n)$ parallel time using $O(n)$ operations on the EREW PRAM*

2.5 Equivalent Binary Tree of a Tree

To solve the l-vertex-coloring for k-trees, we have to transform an arbitrary rooted tree into a regular binary tree. We now show how an arbitrary rooted tree can be reduced to a regular binary tree T, which is the canonical binary tree representation of T.

Let (T, S) be a tree-decomposition of G with width $\leq k$ and $n_T \leq n$. We can transform it to a *binary tree-decomposition* as follows [Bod90]: regard T as rooted tree by choosing an arbitrary node as the root, and replace each internal node x having d children, say $y_1, y_2, \ldots, y_d$ with $d+1$ new nodes $x_1, x_2, \ldots, x_{d+1}$ such that $X_x = X_{x_1} = X_{x_2} = \ldots = X_{x_{d+1}}$, where x_i, $1 \leq i \leq d$, is the father of x_{i+1} and the ith child y_i of x, and x_{d+1} is a leaf of the tree. This transformation can be done in $O(\log_2 n)$ parallel time using $O(n)$ operations on the EREW PRAM. The resulting tree-decomposition (T, S) of $G = (V, E)$ has the following characteristics:

- the width of (T, S) is $\leq k$, and the number n_T of nodes in T is $O(n)$;
- each internal node x of T has exactly two children, say y and z, and either $X_x = X_y$ or $X_x = X_z$; and
- for each edge $e = (v, w) \in E$, there is at least one leaf x in T such that $v, w \in X_x$.

We then cite the following theorem from [BH95].

Theorem 2.2 A *binary tree-decomposition* T of partial k-tree with height $O(\log_2 n)$ and width at most $3k+2$, can be constructed from a given *tree-decomposition* in $O(\log_2 n)$ parallel time using $O(n)$ operations on the EREW PRAM.

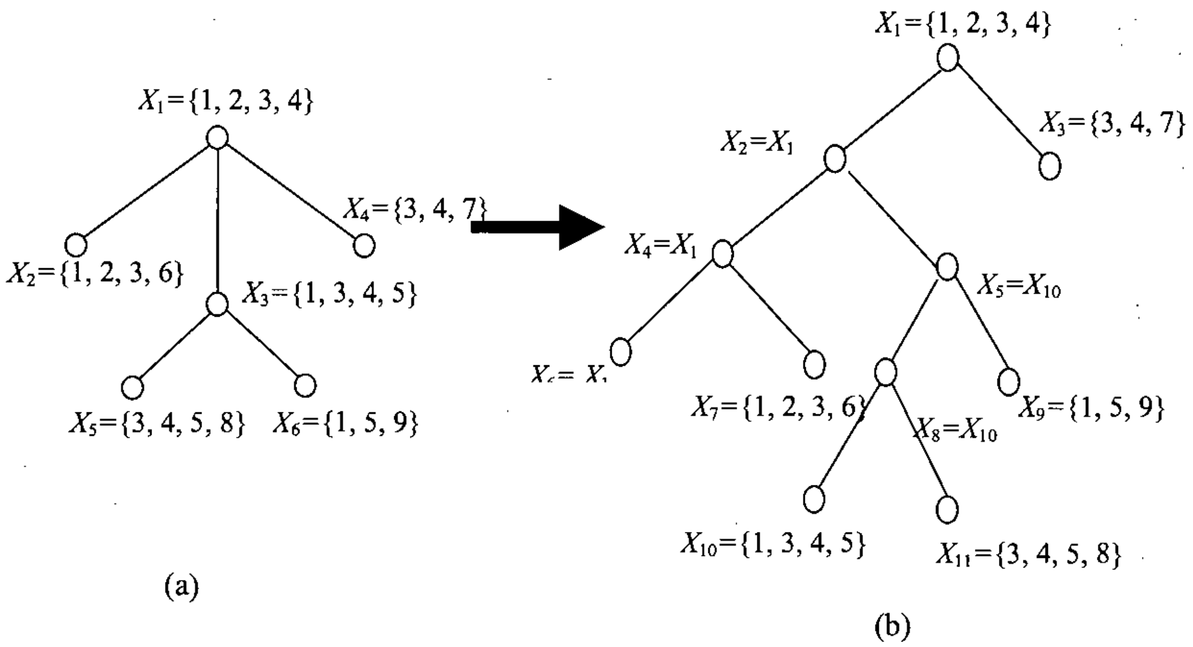

Figure 2.7 (a) a tree-decomposition of the graph and (b) binary-tree representation of the tree in (a).

Let (T, S) be a binary tree-decomposition of partial k-tree $G = (V, E)$. We associate a subgraph $G_x = (V_x, E_x)$ of G with each node x of T, where

$V_x = \bigcup\{X_y \mid y = x \text{ or } y \text{ is a descendant of } x \text{ in } T\}$; and

$E_x = \{(u, v) \in E \mid u, v \in V_x\}$.

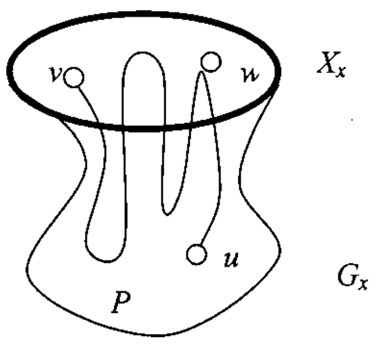

Figure 2.8 Graph G_x

Fig 3.8 depicts graph G_x, where X_x is indicated by an oval drawn in thick line. Thus G is associated with the root of T.

Chapter 3

Parallel Algorithm for *l*-Vertex-Coloring of Partial *k*-Trees

This chapter deals with parallel algorithm for solving the generalized vertex coloring problem on partial *k*-trees. Since ordinary vertex coloring problem is NP-hard [GJ79], the *l*-vertex-coloring problem is also NP-hard in general. So it is very unlikely that there exists an efficient algorithm to solve the *l*-vertex-coloring problem for general graphs. But polynomial time algorithms have been developed for special subset of graphs – partial *k*-trees in particular. Zhou *et al.* [ZKN00] presented a polynomial sequential time algorithm that determines whether any *l-vertex-coloring* exists for partial *k*-trees in time $O\left(n^3 + n \times (\alpha+1)^{2^{2(k+1)(l+2)+1}}\right)$, where α is the number of colors. They claimed that it is possible to solve the same problem in $O(\log_2 n)$ parallel time using $O(n(\alpha+1)^{2^{6(k+1)(l+2)}} + n^3)$ operations on the common CRCW PRAM. It may be noted that partial *k*-trees are graphs of treewidth bounded by a fixed constant *k* [RS86].

In this chapter we present an parallel algorithm that takes $O(\log_2 n)$ time using $O\left(\alpha \times (l+2)^{6\alpha(k+1)} \times n\right)$ operations on the common CRCW PRAM model to determine *l*-vertex-coloring of partial *k*-trees. We then used this algorithm together with parallel search technique over a bounded range of positive integers to determine the optimal number of colors.

The *l*-vertex-coloring problem on a weighted graph $G = (V, E)$ can be easily reduced to the ordinary vertex-coloring problem on a new non-weighted graph $G_l = (V, E_l)$ such that $(u, v) \in E_l$ for any two vertices *u* and *v* in *V* if and only if $dist(u, v) \leq l$ in *G*. Therefore, one may expect that the *l*-vertex-coloring problem for a partial *k*-tree *G* can be efficiently solved by applying a linear-time algorithm [BPT92] to solve an ordinary vertex-coloring problem for partial k-tree. However, it is not the case, because G_l is not always a partial k-tree.

The remainder of this chapter is organized as follows. Section 3.1 gives some preliminary definitions and easy observations. In Section 3.2 we define an equivalence class to solve the l-vertex coloring of a partial k-tree. Section 3.3 gives a parallel algorithm for l-vertex-coloring of partial k-trees. Finally, Section 3.4 concludes with our parallel l-vertex-coloring problem of partial k-trees.

3.1 Preliminaries

In this section we define some terms and easy observations. Let $G = (V, E)$ denote a graph with vertex set V and edge set E. We often denote by $V(G)$ and $E(G)$ the vertex set and the edge set of G, respectively. We denote by n the number of vertices in G. An edge joining vertices u and v is denoted by (u, v). Let N be the set of all positive numbers, and let $w: E \rightarrow$ N be an edge-weight function for G. For a path P in G, we define the *length* $w(P)$ of P as $w(P) = \sum_{e\in P} w(e)$. For two vertices u and v of G, we define the distance $dist(u, v)$ between u and v as follows: if there is a path from u to v in G, then $dist(u, v) = \min\{w(P): P \text{ is a path from } u \text{ to } v \text{ in } G\}$; otherwise, $dist(u, v) = \infty$.

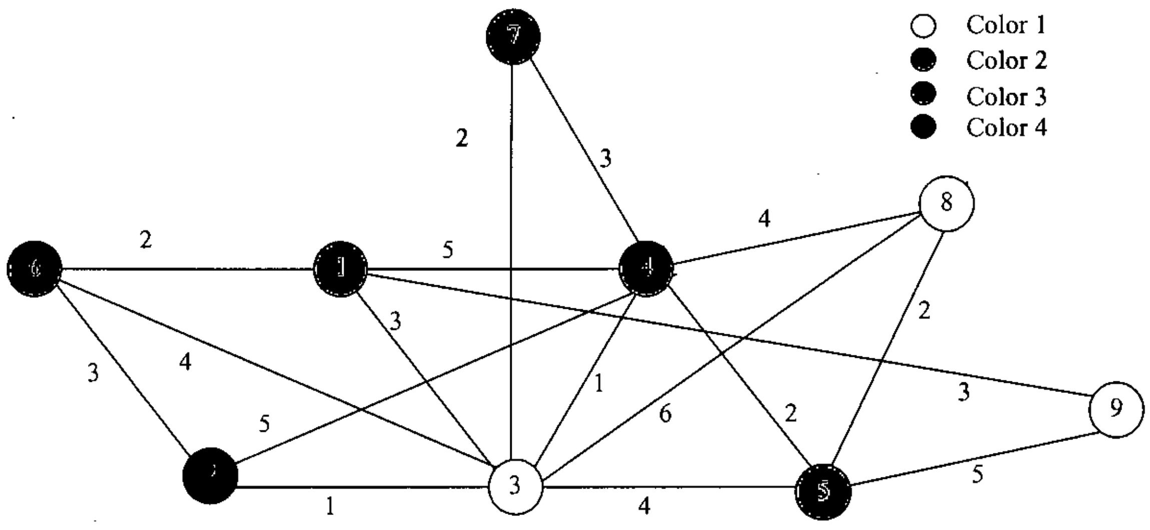

Figure 3.1: 4-vertex-coloring of a partial 3-tree using 4 colors

Definition 3.1 Let l be a positive integer and C be a set of colors. Then a function $\varphi : V \to C$ is an l-vertex-coloring of a partial k-tree G if $\varphi(u) \neq \varphi(v)$ for any two vertices u and v such that $dist(u, v) \leq l$.

Definition 3.2 The minimum number of colors needed to perform an l-vertex-coloring of T is called l-chromatic number of T and is denoted by $\chi_l(T)$.

Let φ be a vertex-labeling of a partial k-tree $G = (V, E)$ with positive integers. The label (color) of a vertex $v \in V$ is denoted by $\varphi(v)$. The number of colors used by a vertex-labeling φ is denoted by #φ. *Clearly* $\chi_l(T) \leq$ #φ *for an* l*-vertex-coloring of* G. One may assume without loss of generality that φ uses consecutive integers 1, 2, …, #φ as the colors. Then C is the color set having colors 1, 2, …, #φ.

Let (T, S) be a binary tree-decomposition of partial k-tree $G = (V, E)$. We associate a subgraph $G_x = (V_x, E_x)$ of G with each node x of T, where

$V_x = \bigcup\{X_y \mid y = x$ or y is a descendant of x in $T\}$; and

$E_x = \{(u, v) \in E \mid u, v \in V_x\}$.

Thus G is associated with the root of T. For a subgraph $G_x = (V_x, E_x)$ of G, $x \in V_T$, we denote by $\varphi| G_x$ the *restriction* of φ to G_x : let $\eta = \varphi| G_x$, then $\eta(v) = \varphi(v)$ for $v \in V_x$. We then have the following lemma.

Lemma 3.1: *Let (T, S) be a binary tree- decomposition of a partial k-tree G, and let x be a node in T. Then a vertex-labeling φ of G_x is an l-vertex-coloring of G_x if and only if*

(a) *any two vertices u and w with $\varphi(u) = \varphi(w)$, $u, w \in V_x$, are within a distance from a vertex $v \in X_x$ such that $dist(u, v) + dist(v, w) > l$.*

(b) *if x is an internal node in T and has two children y and z, then $\varphi| G_y$ and $\varphi| G_z$ are l-vertex colorings of G_y and G_z, respectively.*

Proof:

$\Rightarrow$ Let φ be an l-vertex coloring of G_x. Then any two vertices $u, w \in V_x$ with $\varphi(u) = \varphi(w)$ satisfy $dist(u, w) > l$.

If x is a leaf, then $V_x = X_x$. Since $dist(u, w) > l$, then for any vertex $v \in X_x$ clearly $dist(u, v) + dist(v, w) > l$.

Let x be an internal node. Then there may be two cases.

Case 1:*Both u and w are either in G_y or in G_z.*

Since $dist(u, w) > l$, clearly for any vertex $v \in X_x$ we have $dist(u, v) + dist(v, w) > l$.

Case 2: *One of u and w is in G_y and the other in G_z.*

In this case any path between u and w must contain a vertex $v \in X_x$. Since $dist(u, w) > l$, clearly $dist(u, v) + dist(v, w) > l$.

Since G_y and G_z are sub-graph of G_x, clearly $\varphi| G_y$ and $\varphi| G_z$ are l-vertex colorings of G_y and G_z, respectively.

$\Leftarrow$ Suppose for a contradiction that a vertex-labeling φ satisfies (a) and (b) but φ is not an l-vertex-coloring of G_x. Then there exist two vertices $u, w \in V_x$ with $\varphi(u) = \varphi(w)$ such that $dist(u, w) \leq l$. Since (a) and (b) hold, x is an internal node of T. Then both u and w neither belong to G_y nor G_z; one of u and w is in G_y and the other is in G_z. Furthermore G_y and G_z have common vertices only in X_x. Then shortest path between u and w must contain a vertex $v \in X_x$. Then $dist(u, v) + dist(v,w) \leq l$. This contradicts (a). □

3.2 Equivalence Classes

Many algorithms on partial k-trees use dynamic programming. On each node of the tree-decomposition, a table of all possible partial solutions of the problem is computed, where each entry in the table represents an equivalence class. The time complexity of an algorithm mainly depends on the size of this table. Therefore, we shall find a suitable equivalence class for which the table has a polynomial size. The table of our algorithm has a size $O(n\alpha(l+2)^{6\alpha\,(k+1)})$. List-set can be seen as an equivalence class in our algorithm.

Let (T, S) be a binary tree-decomposition of a partial k-tree $G = (V, E)$. Let $C = \{1, 2, \ldots, \alpha\}$ be the set of colors. Let x be a node in T and let $\varphi : V_x \to C$ be a vertex-labeling of the subgraph $G_x = (V_x, E_x)$. We now define a color-list $L(\varphi, u)$ and a list-set $L(\varphi)$ as follows:

$L(\varphi, u) = \{(i, d_i) \mid$ color $i \in C$ has already been assigned to a vertex v in G_x under φ and d_i is the minimum value of $dist(u, v)$ with $\varphi(v) = i$ and $u \in X_x\}$; and

$L\varphi) = \{L(\varphi, u) \mid u \in X_x\}$.

A list-set $L(\varphi)$ is called *feasible* if the vertex-labeling φ is an l-vertex-coloring of G.

An l-vertex-coloring of G_x, $x \in V_T$, is defined to be *extensible* if it can be extended to an l-vertex-coloring of G without changing the labeling of vertices in G_x. We then have the following lemma.

Lemma 3.2: *Let φ and η be two l-vertex-colorings of G_x such that $L(\varphi) = L(\eta)$. Then φ is extensible if and only if η is extensible.*

Proof: It is sufficient to prove that if φ is extensible, then η is extensible. Suppose φ is extensible. Then φ can be extended to an l-vertex-coloring φ' of $G = (V, E)$ such that $\varphi'(v) = \varphi(v)$ for any vertex $v \in V_x$. Let $V^* = V - V_x$ and let G^* be the sub-graph of G induced by V^*. We can extend the l-vertex-coloring η of G_x to a vertex labeling η' of G as follows:

$$\eta'(v) = \begin{cases} \eta(v) \text{ if } v \in V_x; \text{ and} \\ \varphi'(v) \text{ if } v \in V^*. \end{cases}$$

Then it suffices to prove that η' is an l-vertex-coloring to G, that is, any two vertices $v, w \in V$ with $\eta'(v) = \eta'(w)$ are in a distance such that $dist(v, w) > l$. Now we have two cases.

Case 1: *Both v and w are either in G^* or in G_x.*

In this case $\eta' \mid G_x = \eta$ and $\eta' \mid G^* = \varphi' \mid G^*$ are the l-vertex-colorings of G_x and G^*, respectively. Then v and w are in a distance such that $dist(v, w) > l$.

Case 2: *One of v and w is in G_x and the other is in G^*.*

In this case G_x and G^* have common vertices only in X_x. Then the shortest path between v and w must contain a vertex $u \in X_x$. Now it is sufficient to prove that $dist(v, u)+dist(u, w)>l$.

Since $L(\varphi) = L(\eta)$, we have $L(\varphi, u)= L(\eta, u)$. Hence $\varphi(u)= \eta(u)$. Therefore, $\varphi'(u) = \varphi(u) = \eta(u) = \eta'(u)$. Furthermore, $\eta' | G^* = \varphi' | G^*$. Let $v \in V^*$. Therefore $\eta'(v) = \varphi'(v)$. Since $L(\varphi, u) = L(\eta, u)$, there exists a vertex $a \in V_x$ such that $\varphi(a) = \eta(w)$ and $dist(u, a) = dist(u, w)$. Since φ' is an l-vertex-coloring of G, any two vertices $v, a \in V$ with $\varphi'(a) = \varphi'(v)$ are in a distance such that $dist(v, a) > l$, that is, $dist(v, u) + dist(u, a)>l$. Since $dist(u, a) = dist(u, w)$, we have $dist(v, u) + dist(u, w) > l$. Hence clearly $\text{dist}(v, w) > l$. Therefore $\eta'$is an l-vertex-coloring of G. Thus $\eta'$is extensible. □

Example: Let us consider 4-vertex-coloring of the following graph with 3 colors. Consider the partial coloring η of G_x the subgraph of the partial 3-tree of Fig 3.2. Edge weights are written next to the edges.

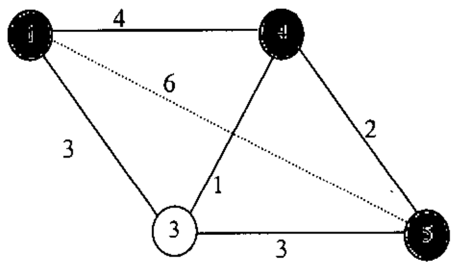

Figure 3.2: A partial 4-vertex-coloring of a subgraph associated with a leaf node x of T

The color list for vertex $v1$is L(η,v1) = (3, 0,4,∝). Similarly, color lists for the vertices $v3$, $v4$, and $v5$ are L(η,v3) = (0,3 ,1,∝), L(η,$v4$) = (1, 2,0,∝), and L(η,$v5$) = (3,0,2,∝) respectively.

Lemma 3.3 The number of distinct color-lists for each vertex is at most $(l + 2)^{\alpha}$. Since $|X_x| \leq 3(k + 1)$, number of distinct list-sets associated with each child of x of T is at most $(l + 2)^{3\alpha(k+1)}$.

Proof: Immediate. □

3.3 An Efficient Parallel Algorithm

We will now show in Lemma 3.4 that if x is an internal node in T with two children y and z then l-vertex-coloring of G_x can be obtained from l-vertex-colorings of G_y and G_z.

We shall use the notation $d_i(\varphi, u)$ to indicate the distance component of (i, d_i) in the color-list $L(\varphi, u)$, *i.e.* $d_i(\varphi, u) = d_i$, where $(i, d_i) \in L(\varphi, u)$. We then have the following lemma.

Lemma 3.4: *Let x be an internal node in T with two children y and z. Let u be any vertex in X_x. Let η and ψ be the l-vertex-colorings of G_y and G_z, respectively. Let $\eta(v) = \psi(v)$ for all vertices $v \in X_y \cap X_z$, and let φ be the vertex-labeling of G_x extended from η and ψ. Then $L(\varphi, u) = \{(i,d_i) \mid i \in C\}$ and $d_i = \min\{d_{i\eta}, d_{i\psi}\}$,where $d_{i\eta} = \min_{v \in X_y \cap X_z}\{d_i(\eta, v) + dist(u, v)\}$ and $d_{i\psi} = \min_{v \in X_y \cap X_z}\{d_i(\psi, v) + dist(u, v)\}$).*

Proof: Let i be any color in the set of color and $(i, d_i) \in L(\varphi, u)$. Then we have that d_i is the minimum of all the distances from u to vertices in G_y and G_z with color i.

To determine the extended color-list, we need to consider the elements in $L(\eta, u)$ and $L(\psi, u)$. Furthermore, u may be connected to $v \in X_x$ through a path. So we also need to consider the elements in $L(\eta, v)$ and $L(\psi, v)$. Therefore $d_i = min\{d_{i\eta}, d_{i\psi}\}$,where $d_{i\eta} = min_{v \in X_x \cap X_z}\{d_i(\eta, v) + dist(u, v)\}$ and $d_{i\psi} = min_{v \in X_x \cap X_z}\{d_i(\psi, v) + dist(u, v)\}$).

Thus $L(\varphi, u) \subseteq \{(i, d_i) \mid d_i = min\{d_{i\eta}, d_{i\psi}\}\}$. (1)

Let i be any color and $(i, d_i(\eta, u)) \in L(\eta, u)$, $(i, d_i(\eta, v)) \in L(\eta, v)$, $(i, d_i(\psi, u)) \in L(\psi, u)$ and $(i, d_i(\psi, v)) \in L(\psi, v)$. Then if $d_i(\eta, u)$ is the minimum of all the distances from u to the vertices of with color i in G_x then $(i, d_i(\eta, u)) \in L(\varphi, u)$. Since $dist(u, u) = 0$, one can write $(d_i(\eta, u) + dist(u, u)) \in L(\varphi, u)$. Otherwise if $(d_i(\eta, v) + dist(u, v))$, $v \in X_x$ and $v \neq u$, is the minimum of all the distances from u to the vertices with color i in G_x then $(i, d_i(\eta, v) + dist(u, v)) \in L(\varphi, u)$. Similarly we may find the cases such that $d_i = d_i(\psi, u) + dist(u, u)$ or $d_i = d_i(\psi, v) + dist(u, v)$ for any $v \in X_x$ and $v \neq u$.

Thus $L(\varphi, u) \supseteq \{(i, d_i) \mid d_i = min\{d_{i\eta}, d_{i\psi}\}\}$. (2)

From equations (1) and (2), we have $L(\varphi, u) = \{(i, d_i) \mid d_i = min\{d_{i\eta}, d_{i\psi}\}\}$ □

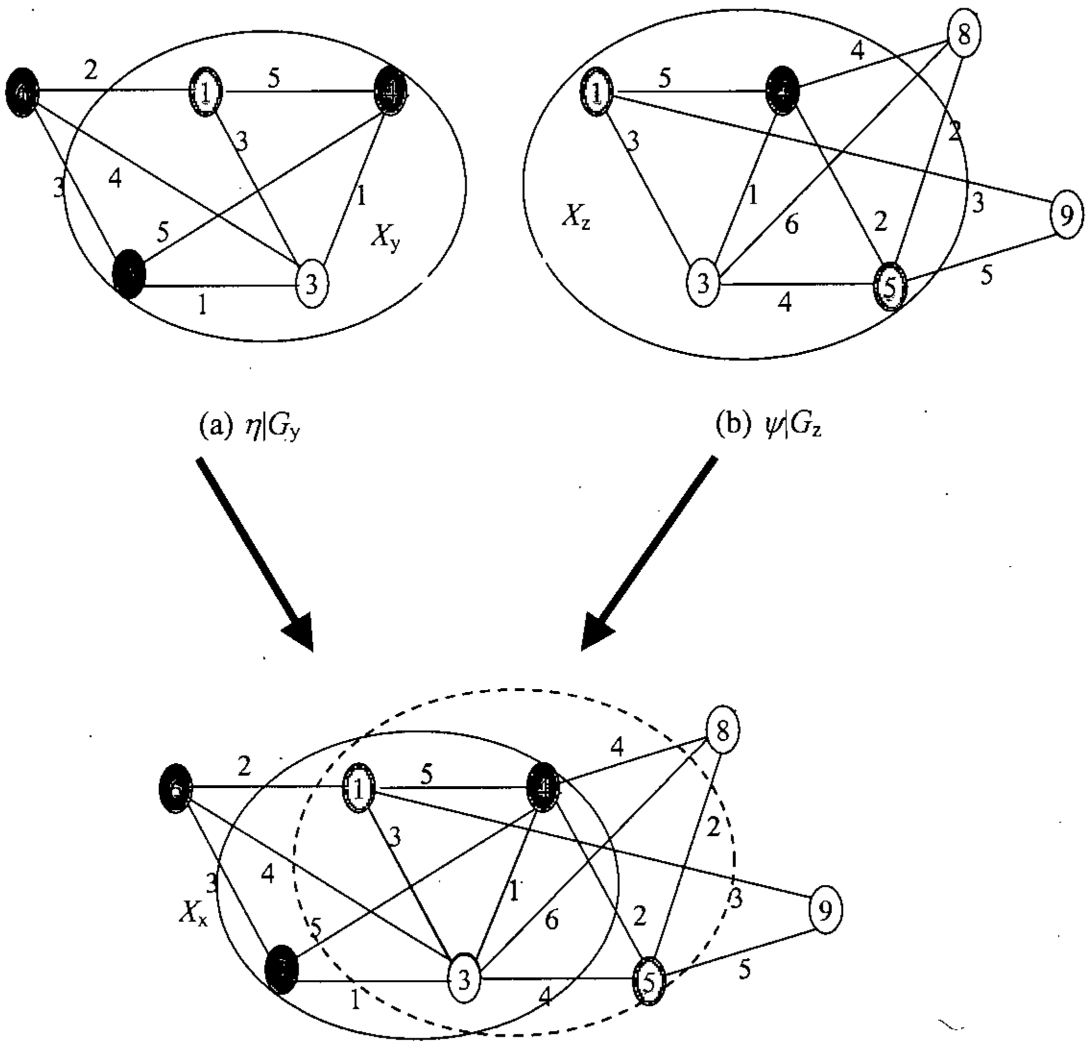

Figure 3.3: Merging of $\eta|G_y$ and $\psi|G_z$ to form $\varphi|G_x$

Example: Let x be an internal node in T with two children y and z. Let η and ψ be the l-vertex-colorings of G_y and G_z, respectively. Let $\eta(v) = \psi(v)$ for any vertex $v \in X_y \cap X_z = \{v1, v3, v4\}$. Now L($\eta$) ={$v$1(3,0,2,4), v2(1,4,2,0), v3(0,3,1,1), v4(1,4,0,2)} and L(ψ) ={v1(3,0,4,∝), v3(0,3,1,∝), v4(1,2,0,∝), v5(3,0,2,∝)} are given. We can calculate L(φ) from L(η) and L(ψ) by using Lemma 3.4. Now, L(φ) ={v1(3,0,2,4), v2(1,4,2,0), v3(0,3,1,1), v4(1,2,0,2)}.

Lemma 3.5: *Let x be a leaf node in T and let φ is a coloring of G_x. Then $L(\varphi)$ is a feasible vector if and only if for all pair of vertices $u,v \in V_x$ with $\varphi(u) = \varphi(v)$, $dist(u,v) > l$.*

Proof: Immediate.

We have to check whether there exists a possible l-vertex-coloring of G_x from the previously obtained l-vertex-colorings of two subgraphs G_y and G_z. We then have the following Lemma.

Lemma 3.6: *Let x be an internal node in T with two children y and z. Let $u \in X_x$, and let $v \in X_y \cap X_z$. Let η and ψ be the l-vertex-colorings of G_y and G_z, respectively. Then η and ψ can be combined to form an l-vertex-coloring φ of G_x if and only if for all possible pairs (u, v), we have $dist(u,v) + d_i(\eta,u) + d_i(\psi,v) > l$, $1 \le i \le \alpha$, provided $d_i(\eta,u)$ and $d_i(\psi,v)$ contain the color distance of two distinct vertices.*

Proof: At first we have to check whether $d_i(\eta,u)$ and $d_i(\psi,v)$ contain the color distance of a common vertex. Since, the common vertices must belong to $X_y \cap X_z$, so we have to check whether such a common vertex, $w \in X_y \cap X_z$ exists or not. If such a w exists then we need not to check any l-vertex-coloring condition, since η and ψ are the l-vertex-colorings of G_y and G_z, respectively.

If $d_i(\eta, u)$ and $d_i(\psi, v)$ do not contain the color distance of a common vertex, then we know, $d_i(\eta,u) = \min_{w_1 \in Gy}\{dist(u,w_1) \mid \eta(w_1) = i\}$, .and $d_i(\psi,v) = \min_{w_2 \in Gz}\{dist(v,w_2) \mid \psi(w_2) = i\}$.

Then the distance between two vertices $w_1 \in G_y$ and $w_2 \in G_z$ having the same color i in the subgraph G_x is $dist(u,v) + d_i(\eta,u) + d_i(\psi,v)$. According to the definition of l-vertex-coloring the distance between any two vertices having the same color must be greater than l. If $dist(u,v) + d_i(\eta,u) + d_i(\psi,v) > l, 1 \le i \le \alpha$ for all pairs (u,v), then the labeling will be an l-vertex-coloring of G_x.

Assume that φ be an *l-vertex-coloring of* G_x Then for all pairs of vertices u, $v \in V_x$,we have $dist(u,v) > l$ with $\varphi(u) = \varphi(v)$. So it must be true that for all possible pairs u, v, $u \in X_x$ and $v \in X_y \cap X_z$, $dist(u,v) + d_i(\eta,u) + d_i(\psi,v) > l, 1 \le i \le \alpha$, provided $d_i(\eta,u)$ and $d_i(\psi,v)$ contain the color distance of two distinct vertices. □

3.3.1 Algorithm

The main result of this section is the following theorem.

Theorem 3.7 *Let G be a partial k-tree of n vertices given by its tree-decomposition with height* $O(\log_2 n)$ *and width at most 3k + 2. Then an optimal l-vertex coloring of G can be found in* $O(\log_2 n)$ *parallel time using* $O\left(\alpha \times (l+2)^{6\alpha(k+1)} \times n + n^3\right)$ *operations on the common CRCW PRAM for any positive integer l and any bounded integer k.*

In the remaining of this section we prove the theorem 3.7.

The following general lemma is well known [BDJ98].

Lemma 3.8 *Let A be a given algorithm with* $O(\log_2 n)$ *parallel computation time. If A involves a total number of q operations, then A can be implemented using p processors in* $O(q/p + \log_2 n)$ *parallel time.*

If there is an algorithm A which solves the l-vertex-coloring problem in $O(\log_2 n)$ parallel time using a total of $q = O\left(\alpha \times (l+2)^{6\alpha(k+1)} \times n\right)$ operations, then by adapting lemma 3.8 with choosing $p = q/\log_2 n$ one can know that A can be implemented using $O(q/\log_2 n)$ processors in $O(\log_2 n)$ parallel time.

Thus by Theorem 3.7 and Lemma 3.8 we have the following corollary.

Corollary 3.9 *For any positive integer l and any bounded integer k, the l-vertex-coloring problem for partial k-trees can be solved in* $O(\log_2 n)$ *parallel time with a polynomial number of processors on the common CRCW PRAM.*

The following lemma is also a well-known [Jos92].

Lemma 3.10 *Given a parallel algorithm that can be implemented to run in time t on a p-processor common CRCW PRAM, this algorithm can be implemented on a p-processor EREW PRAM to run in* $O(t \log_2 p)$ *time.*

Thus by Corollary 3.9 and Lemma 3.10 we have the following corollary.

Corollary 3.11 *For any positive integer l and any bounded integer k, the l-vertex-coloring problem for partial k-trees can be solved in* $O(\log_2^2 n)$ *parallel time with a polynomial number of processors on the EREW PRAM.*

In the remaining section we prove the Theorem 3.7. Let (T, S) be a binary tree-decomposition of a partial k-tree G with width $\leq 3k + 2$. We first give a parallel algorithm to decide, for a given positive integer $\alpha \leq n$, whether G has an l-vertex-coloring φ with $\#\varphi \leq \alpha$. We use parallel dynamic programming and bottom-up tree computation on the binary tree T: for each node x of T from leaves to the root, we construct all (equivalence classes of) l-vertex-colorings of G_x from those of two subgraphs G_y and G_z associated with the children y and z of x. Then, by using a parallel search over the range of α, we determine the minimum value of α such that G has an l-vertex-coloring φ with $\alpha = \#\varphi$, and find an optimal l-vertex-coloring of G.

A feasible vector $L(\varphi)$ of φ on x can be seen as an equivalence class of extensible l-vertex-coloring of G_x. Each color i can be placed in the color-list with the distance $d_i = 0, 1, 2, \ldots, l$ or ∞. Therefore the number of distinct color-lists for each vertex is at most $(l + 2)^{\alpha}$. Since $|X_x| \leq 3(k + 1)$, number of distinct list-sets associated with each child of x of T is at most $(l+2)^{3\alpha(k+1)}$. Therefore total number of different feasible vectors on a leaf node x is at most $(l+2)^{3\alpha(k+1)}$ for any fixed integer k.

We first show how to find the table all feasible vectors $L(\varphi)$ on a leaf x of T. This can be done as follows:

1. Enumerate all vertex-labeling $\varphi: Vx \rightarrow C$ of G_x.
2. Compute all feasible vectors $L(\varphi)$ on x from the vertex-labeling φ of G_x.

Since $|V_x| \leq 3(k+1)$ and $|C| \leq \alpha$, the number of vertex labeling φ: $V_x \rightarrow C$ is at most $\alpha^{3(k+1)}$. Furthermore the color-lists $L(\varphi, u)$, $u \in X_x = V_x$ can be computed in $O(1)$ time. So, $L(\varphi)$ can also be computed in $O(1)$ time for a bounded integer k.

To check whether φ is an l-vertex coloring of G_x, we have to check the Lemma 3.5. Since, $|V_x| \leq 3(k+1)$, so the total number of check to be made is ${}^{3(k+1)}C_2 \approx 3(k+1)(3k+2)/2$. It can be computed for a node x can be made in O(1) parallel time using O(k^2) operations on the EREW PRAM. So, table of all feasible vectors on a leaf x of T can be computed for a leaf in O(1) parallel time using $O(\alpha^{3(k+1)})$ operations on the EREW PRAM. Since there are $O(n)$ leaves, such table for all leaves can be computed in in O(1) parallel time using $O(n\alpha^{3(k+1)})$ operations on the EREW PRAM.

We next show how to compute all feasible vectors on an internal node x of T from those on two children y and z of x. One may assume that $X_x = X_y$. By the definition of $G_x = (V_x, E_x)$, we have $V_x = V_y \cup V_z$ and $E_x = E_y \cup E_z$. Let η and ψ, respectively, be the l-vertex-colorings of G_y and G_z, such that $\eta(v) = \psi(v)$ for any vertex $v \in X_y \cap X_z$, and let φ be the vertex-labeling of G_x extended from η and ψ. Then $\varphi \mid G_y = \eta$ and $\varphi \mid G_z = \psi$. Fig. 3.4 illustrates G_x, G_y and G_z where X_x, X_y and X_z are drawn in ovals.

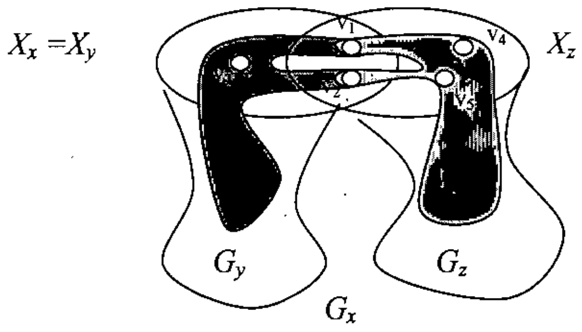

Figure 3.4: Graphs G_x, G_y, and G_z.

First we have to check whether there exists a possible l-vertex-coloring of G_x from the previously obtained l-vertex-coloring of two subgraphs G_y and G_z.This can be checked by using Lemma 3.6, in $O(1)$ parallel time using $O(\alpha k^2)$ operations on the CREW PRAM.

We next show how to compute $L(\varphi)$ from $L(\eta)$ and $L(\psi)$. $L(\varphi)$ can be obtained using Lemma 3.4. The obtained $L(\varphi)$ is a feasible vector and φ is an l-coloring of G_x. It can also be done in O(1) parallel time using $O(\alpha k^2)$ operations on the CREW PRAM.

The table of all feasible vectors on an internal node x can be obtained from the pairs of tables of all feasible vectors on the two children y an z of x, and the number of these pairs is $O((l + 2)^{6\alpha(k+1)})$.To compute the table on x in $O(1)$ parallel time , we need common CRCW PRAM model. A concurrent read capability is needed, because each feasible vector of the table y needs to simultaneously accessed by all the $(l + 2)^{3\alpha(k+1)}$ processors corresponding to the table on z and each feasible vector of the table z needs to simultaneously accessed by all the $(l + 2)^{3\alpha(k+1)}$ processors corresponding to the table on y. A concurrent write up of the same value is required, because different pairs of feasible vectors on y and z may compute the same feasible vector on x.

We know the number of distinct list-sets associated with each child of x of T is at most $(l + 2)^{3\alpha(k+1)}$.Therefore number of distinct pair of list-set between two children of x is at most $(l + 2)^{6\alpha(k + 1)}$. While checking each pair of list-set and computing the valid list-set on x can also be done in $O(1)$ parallel time using $O(\alpha k^2)$ operations on the CREW PRAM. Thus computing a table of all valid list-sets on x from those on the two children of x takes $O(1)$ parallel time using $O(k^2\alpha(l + 2)^{6\alpha(k + 1)}) \approx O(\alpha(l + 2)^{6\alpha(k + 1)})$ operations on the common CRCW PRAM, where k is a bounded integer.

We thus have the following parallel algorithm to find an optimal l-vertex-coloring of a partial k-tree G, where h is the height of the tree.

```
Parallel Algorithm l-vertex-coloring
begin
1  Find all pair shortest path for the given partial $k$-tree $G$ in parallel
2  Obtain a binary tree decomposition $T$ with height $O(\log_2 n)$ of partial $k$-tree $G$.
3  for $m: = 1$ to $\alpha$ in parallel do
   begin
4     compute a table of all feasible vectors on each leaf $x$ of $T$ for $C = \{1, 2, \ldots, m\}$ in
      parallel
5    for $l: = 1$ to $h$ do   $\{ h = O(\log_2 n)\}$
6      for each internal node $x$ in $T$ at level $l$ in parallel do
7         compute a table of all feasible vectors from those on the two children of $x$;
8    check whether there exist a feasible vector in the table at the root;
   end
9  Apply parallel search on the range of color-set $1 \leq \alpha \leq n$ and find the smallest integer $m$
   such that there exists a feasible vector in the table at the root for $C = \{1, 2, \ldots, \alpha\}$, and
   output $m$ as $\chi_l(G)$;
End.
```

Line 1 can be computed in $O(\log_2 n)$ parallel time using $O(n^3)$ operations [ZKN00].

Line 2 can be computed in $O(\log_2 n)$ parallel time using $O(n)$ operations on the EREW PRAM [BH95].

Lines 4-8 are executed for all m in parallel in Line 3. We now show that Lines 4-8 can be done in $O(\log_2 n)$ parallel time using $O(n\alpha(l+2)^{6\alpha(k+1)})$ operations on the common CRCW PRAM.

In Line 4 of the algorithm, we compute a table of all feasible list-sets on each leaf x of T. Thus computing a table of all feasible list-sets on each leaf x of T can be computed for a leaf in $O(1)$ parallel time using $O(\alpha^{3(k+1)})$ operations on the EREW PRAM as mentioned before. Since there are $O(n)$ leaves, Line 4 can be computed in $O(1)$ parallel time using $O(n\alpha^{3(k+1)})$ operations on the EREW PRAM for all leaves.

Line 7 can be done $O(1)$ parallel time using $O(n\alpha(l+2)^{6\alpha(k+1)})$ on the common CRCW PRAM for each node. Since $h = O(\log_2 n)$, Lines 5 and 6 can be done in $O(\log_2 n)$ parallel time using $O(n\alpha(l+2)^{6\alpha(k+1)})$ operations on the common CRCW PRAM. Line 8 can be done in $O(1)$ parallel time using $O(n\alpha(l+2)^{6\alpha(k+1)})$ operations on the common CRCW PRAM. Thus Lines 4-8 can be done in $O(\log_2 n)$ parallel time using $O(n\alpha(l+2)^{6\alpha(k+1)})$ operations on the common CRCW PRAM.

For optimization we apply parallel search on the range of color-set $1 \leq \alpha \leq n$ in Line 9. Line 9 can be done in $O(\log_2 n)$ sequential time using $O(\log_2 n)$ operations on the EREW PRAM.

Thus an optimal l-vertex-coloring of a partial k-tree G of n vertices can be found in $O(\log_2 n)$ parallel time using $O(\alpha(l+2)^{6\alpha(k+1)}n + n^3)$ operations on the common CRCW PRAM for any positive integer l and any bounded integer k.

3.4 Conclusion

In this chapter, we have given a parallel algorithm to find an optimal l-vertex-coloring of a weighted partial k-tree G. Our algorithm runs in $O(\log_2 n)$ parallel time using $O(n(l + 2)^{6\alpha(k + 1)} \alpha + n^3)$ operations on the common CRCW PRAM for any positive integer l and any bounded integer k. This is the parallel algorithm of an *l-vertex-coloring problem* of a partial k-trees that guarantees an optimal solution.

Chapter 4

Conclusion

This thesis deals with a generalized vertex-coloring problem on partial k-trees.

In Chapter 2 we have defined some basic terms needed for solving generalized coloring problems. We also described different PRAM models. Finally we described techniques of tree decomposition of an arbitrary tree into equivalent binary tree. These techniques played key roles in our algorithms.

Chapter 3 gives a parallel algorithm to solve the l-vertex-coloring problem of partial k-trees. Our parallel algorithm finds an optimal l-vertex-coloring of a k-tree G in $O(\log_2 n)$ parallel time using $O(\alpha \times (l+2)^{6\alpha(k+1)} \times n + n^3)$ operations on the CRCW PRAM model, where n is the number of vertices in G and α is the number of colors. The previously known best parallel algorithm takes $O(\log_2 n)$ parallel time using $O(n(\alpha+1)^{2^{6(k+1)(l+2)}} + n^3)$ operations on the common CRCW PRAM to solve the same problem [ZKN00]. Though, the time and PRAM model are same in both algorithms, the number of operations is single exponential in our algorithm. In previous algorithm the number of operations is double exponential. Thus our algorithm is better than the existing one.

In this thesis, we have given parallel algorithm for a generalized coloring of partial k-trees. However, following problems are still open:

1. Is there an upper bound on the number of colors needed for an l-vertex-coloring of partial k-trees?
2. What is the complexity of sequential and parallel algorithms for l-vertex-coloring of other graph classes?

References:

[BH95] H. L. Bodlaender, T. Hagerup, Parallel algorithms with optimal speedup for bounded treewidth, *Proc* 22nd *Int. Colloq. on Automata, Languages and Programming, Lecture Notes in Computer Science* **10**, Springer, Berlin, pp. 268-279, 1995.

[BDJ98] H. L. Bodlaender, J.S. Deogum, K.Jansen, T. Kloks, D. Kratsch, H. Müller, Zs. Tuza, Rankings of graphs, SIAM J. Discrete Math., **21** pp.168-181, 1998.

[BK96] H. L. Bodlaender, T. Kloks, Efficient and constructive algorithms for the pathwidth and treewidth of graphs, J. Algorithms, **21** pp.358-402, 1996.

[Bod90] H. L. Bodlaender, Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees. *Journal of Algorithms*, **11(4)**, pp.631-643, 1990.

[Bod96] H. L. Bodlaender, A linear time algorithm for finding tree-decompositions of small treewidth, SIAM J. Comput. **25**, pp.1305-1317, 1996.

[BPT92] R. B. Borie, R. G. Parker and C. A. Tovey, Automatic generation of linear time algorithms from predicate calculus descriptions of problems on recursively constructed graph families, *Algorithmica* **7**, pp. 555-581, 1992.

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

[Jos92] J. Joseph, *An Introduction to Parallel Algorithms*, Addison-Wesley Inc., 1992.

[Ree92] B.A. Reed, Finding approximate separators and computing tree-width quickly, Proc. 24th Ann. ACM Symp. on Theory of Computing, pp 221-228, 1992.

[RS86] N. Robertson, P.D. Seymour, Graph minors. II. Algorithmic aspects of tree-width, J. Algorithms 7, pp 309-322, 1986.

[Wes99] D. B. West, *Introduction to Graph Theory*, 2nd ed., Prantice-Hall Inc., 2002.

[ZKN00] X. Zhou, Y. Kanari and T. Nishizeki, Generalized vertex-colorings of partial k-trees. *IEICE Trans. on Fundamentals*, pp. 555-581, 2000.

Index