ON SOME PROBLEMS ON K-TREES AND PARTIAL K-TREES

By

DARKO SKORIN-KAPOV

B. Sc The University of Zagreb, Yugoslavia, 1978

M. Sc. The University of Zagreb, Yugoslavia, 1983

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

in

THE FACULTY OF GRADUATE STUDIES

(Department of Commerce and Business Administration)

We accept this thesis as conforming

to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

May 1989

87

Department of COMMERCE

The University of British Columbia
Vancouver, Canada

Date June 20, 1989

ABSTRACT

The objective of this thesis is to investigate some structural and algorithmic properties of k-trees and partial k-trees. A k-tree can be constructed from a k-complete graph by recursively adding a new vertex which is adjacent to all vertices of an existing k-complete subgraph. Partial k-trees are graphs embeddable in a k-tree with the same vertex set. They are natural generalizations of forests and series-parallel graphs which are the first two members of the hierarchy of partial k-trees. The many applications of k-trees and partial k-trees have motivated their study from both an algorithmic and a theoretical point of view. For example, k-trees arise in reliable communication network design problems (Farley (1981), Farley and Proskurowski (1982), Neufeld and Colbourn (1983), Wald and Colbourn (1983), Colbourn and Proskurowski (1984)) and in the study of the complexity of certain type of queries in a relational data base system (Arnborg (1979)). Moreover, the class of k-trees is special in the sense that many problems, which are NP-complete for arbitrary graphs, are solvable in polynomial time when restricted to k-trees or partial k-trees (Arnborg and Proskurowski (1989)).

In Chapter 2 of the thesis we analyze a fixed cost spanning forest (FCSF) problem, defined over a graph G, in which some customers require service that can be generated at some facilities' sites. Both the set of customers and facilities' sites are represented by nodes in G. There is a fixed cost for opening each facility and a cost for delivering the service from open facilities to the customers. Customers do not necessarily have to receive the service directly from an open facility, but possibly through other intermediate customers. We develop a linear time algorithm for solving the FCSF problem when the customers and potential facilities' sites are located on a series-parallel network or, equivalently, a partial 2-tree. We further analyze a related cost allocation problem, in which we seek a fair method for allocating the cost of providing the service to the customers. We formulate this cost allocation problem as a cooperative game and show that, in general, the core of this cooperative game may be empty. However, we provide a sufficient condition, which can be verified in polynomial time, for the nonemptiness of the core of this game.

A k-tree can be reduced to the k-complete graph by sequentially removing k-degree vertices with completely connected neighbors. We use this reduction process to develop, in Chapter 3, efficient algorithms for several optimization problems on k-trees and partial k-trees. In particular, we develop a linear time algorithm to find shortest simple paths from a given vertex to all other vertices in a k-tree, we compute the diameter of a k-tree with equal edge lengths in linear time, and we construct an $O(n^{k+2})$ algorithm to solve the Simple Plant Location problem in an n-vertex partial k-tree.

In Chapter 4 we present a new characterization of a k-path between two vertices u and v, in an equal weight k-tree G, by means of minimal k and $k+1$ cliques with respect to certain partial orders defined on the collections of all k and $k+1$ cliques in a k-tree. We use it to develop an $O(n^2)$ algorithm to decompose a vertex set V of a k-tree G to a minimum number of components, such that for any pair of vertices i and j in the same component, the cable distance between i and j is bounded by a positive integer R. We also compute the k-cable diameter of a k-tree with equal edge lengths in linear time.

In Chapter 5 we derive some separation properties of partial k-trees and use them to develop NC algorithms for recognizing partial 2-trees and 3-trees. Explicitly, we prove the existence of a k-separator in a partial k-tree graph and construct a linear time algorithm that finds such a separator in k-trees. This algorithm can be used to obtain a balanced binary decomposition of a k-tree in $O(n \log n)$ time. We derive some other separation properties of partial k-trees and use them to construct a balanced decomposition of an embedding of a k-connected partial k-tree when k = 2 and 3. Finally, we construct NC algorithms for the recognition of a partial k-tree for k = 2 and 3, which run in $O(\log^2 n)$ time using, respectively, $O(n^3)$ and $O(n^4)$ processors.

Table of Contents

List of Figures

List of Tables

Acknowledgement

I would like to express my sincere appreciation to the University of British Columbia and especially to the Faculty of Commerce and Business Administration for their hospitality and financial support given through Dean Earle D. McPhee Memorial Fellowship in Commerce and Business Administration, Leslie G.J. Wong Memorial Fellowship, University of British Columbia Graduate Fellowship and Graduate Travel Grant.

My sincere thanks go to the Division of Management Science for giving me the opportunity to be one of their PhD students.

I am deeply indebted to Professor Daniel Granot, my supervisor, for his guidance, friendliness, many valuable suggestions and comments and financial support. I would also like to thank the other members of my dissertation committee Dr Richard Anstee, Dr Frieda Granot and Dr Maurice Queyranne for their helpful suggestions.

Finally, special thanks with love to my wife Jadranka and our daughters Lea and Nina for their patience and understanding.

To my mother Marija Skorin and
to my father Dinko Skorin

Mojoj majci Mariji Skorin i
mom ocu Dinku Skorinu

Chapter I

INTRODUCTION

The objective of this thesis is to investigate some structural and algorithmic properties of k-trees and partial k-trees. k-trees are a generalization of trees, which can be viewed as 1-trees. They can be constructed from the completely connected (complete) graph on k vertices by the repeated addition of vertices, each of which is connected with k edges to k completely connected vertices. Partial k-trees are partial graphs of k-trees, i.e., graphs obtained by deleting edges from k-trees. They generalize the notion of forests and series-parallel graphs which are respectively, partial 1-trees and partial 2-trees.

The study of the class of k-trees and their partial graphs was motivated by some practical questions concerning the reliability of communication networks in the presence of constrained line-and-site failures (Farley (1981) , Farley and Proskurowski (1982) , Neufeld and Colbourn (1983)) . Therein the vertices V represent sites or nodes in the network, and the edges represent communication lines. Since lines are typically costly, communication between sites is usually not done directly but via other sites. A tree network is an obvious solution. However, for a tree network site and/or line failures are catastrophic. One way to overcome this failures is to increase the connectivity (Hakimi and Tamin (1973)). Farley (1981) proposed isolated failure immune (IFI) networks which can survive site and line failures as long as they are isolated. He also showed that two-trees are minimum IFI networks. (k+1)-trees are IkFI networks in which at most k nonisolated failures are allowed. This, together with the fact that for a fixed k, k-trees have a number of edges which is linear in the size of a vertex set (and thus seem to be economical), makes k-trees attractive for use in the design of reliable communication networks. Studies of k-trees and partial k-trees were also motivated in view of their relevance in modelling the complexity of queries in data base systems. Indeed the evaluation of a certain class of conjuctive data base queries was shown to be equivalent to the embedding of an associated graph, obtained from the query syntax, into a $(k-1)$-tree, for a minimum k (Arnborg (1979)). Moreover, partial k-trees have arisen in some real world problems. As an example, the reliability graph of a battle ship, constructed for weapons effect simulations, had several thousand vertices but was found to be a partial 4-tree (Arnborg (1978)). Further, many examples used in the engineering literature of communication networks are usually partial 4-trees and often series-parallel graphs. Finally, the class of k-trees is special in the sense that many problems which are NP-complete for arbitrary graphs, were shown to be solvable in polynomial time when restricted to this class of graphs; see, for example Arnborg and Proskurowski (1989).

The introduction is organized as follows. In Section 1.1 we review some standard graph-theoretic definitions and in Section 1.2 we discuss several well known optimization problems on general graphs. In Section 1.3 we identify some special classes of graphs. In particular we present some known characteristics of partial k-trees and a number of classes of graphs that are subclasses of partial k-trees. Many graph optimization problems, that are NP-hard on general graphs, can be solved in polynomial time when restricted to these special classes of graphs. Section 1.4 is an overview of the known results on the recognition of partial k-trees and their embeddings into k-trees. Finally in Section 1.5 we give a summary of results obtained in this thesis.

1.1 GRAPH-THEORETIC DEFINITIONS

We present below some basic graph theory notation and definitions. For some recommended graph theory books the reader is referred to Harary (1969) and Berge (1985). A undirected graph or simply a **graph** G is defined to be a pair (V, E), where V is a set $\{ v_1 , v_2 , \cdots, v_n \}$ of elements called vertices (or sometimes nodes) and E is a family $(e_1 , e_2 ,\cdots, e_m)$ of elements of unordered pairs of V, called edges (or sometimes arcs). If an element $e = (u,v)$ of E appears more than once in the family E, it is called **a multiple edge.** An edge of the form (v,v) is called a loop. Graphs without multiple edges are called **simple.**

If the edges are ordered pairs, then $D = (V, E)$, is called a **directed graph** (or a digraph). In a directed graph $D = (V, E)$ the vertices u and v are called the **tail** and the **head** of the directed edge (u,v), respectively. (Note that very often for both undirected and directed graphs an edge from a vertex u to a vertex v is denoted by (u,v) , even though $\{u,v\}$ would be more appropriate for undirected graphs. The concepts of a multiple arc, a simple graph and a loop for directed graphs are analogous to those for undirected graphs.

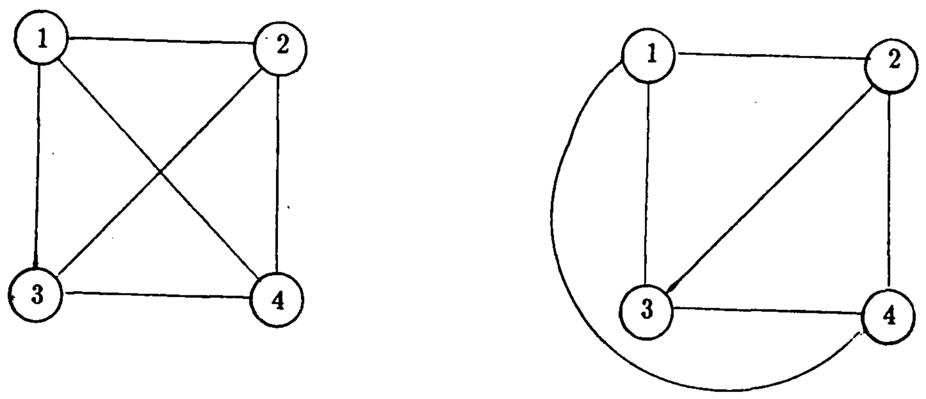


Fig. 1.1 Two drawings of the same graph

We shall say that an edge (u,v) connects the vertices u and v. The vertices u and v are adjacent if there is an edge connecting u and v. The edge (u,v) is said to be incident to the vertices u and v, and conversely. The vertices u and v are called the ends of the edge (u,v). Commonly, graphs are represented by a drawing in which vertices are represented by small circles and edges are lines connecting them (see Fig. 1.1).

Some representations of graphs that are appropriate for computer implementation are an edge list, an incidence matrix and an adjacency matrix. An edge list simply contains an entry for each edge (u,v). Representation by edge lists is the most appropriate for the algorithms studied in the thesis. In the case of the graph in Fig 1.1 such a list contains (1,2), (1,3), (1,4), (2,3), (2,4), (3,4). Each row of a vertex-edge incidence matrix is identified with a vertex and each column with an edge. If the vertices are numbered by the index i and edges by the index k the incidence matrix of an undirected graph G, $A = (a_{ik})$ is defined as follows :

$$a_{ik} = \begin{cases} 1 & \text{if vertex } i \text{ is incident to edge } k \\ 0 & \text{otherwise} \end{cases}$$

In the case of a directed graph D the edge-vertex **incidence matrix** $A = (a_{ik})$ is defined as follows:

$$a_{ik} = \begin{cases} +1 & \text{if } i \text{ is a head of a directed edge } k \\ -1 & \text{if } i \text{ is a tail of a directed edge } k \\ 0 & \text{otherwise} \end{cases}$$

The **adjacency matrix** $B = (b_{ij})$ of an undirected graph $G = (V, E)$ is a matrix with both rows and columns indexed by V and defined as follows : $(b_{ij}) = l$, where l is the number of edges connecting i and j. So G is simple if and only if its adjacency matrix is a $\{0,1\}$−matrix. In the case of a directed simple graph D the adjacency matrix $B = (b_{ij})$ is defined as follows :

$$b_{ij} = \begin{cases} 1 & \text{if there is a directed edge with a head } j \text{ and a tail } i \\ 0 & \text{otherwise} \end{cases}$$

For any vertex $v \in V$, the **neighborhood** of v in a graph G is defined as the set of all vertices adjacent to v. The **degree** of a vertex v is the cardinality of its neighborhood. In the case of a directed graph $D = (V, E)$ we say that the directed edge (u,v) **enters** v and **leaves** u. If W is a set of vertices such that $v \notin W$ and $w \in W$, then (v,w) is said to enter W and to leave $V \setminus W$. If $W \subseteq V$, then $\delta^{-}(W)$ denotes the set of edges in E entering W and $\delta^{+}(W)$ the set of directed edges in E leaving W. We let $\delta^{-}(v)$ and $\delta^{+}(v)$ stand for $\delta^{-}(\{v\})$ and

$\delta^+(\{v\})$ and are called **in-degree of** v and **outdegree of** v, respectively.

A **path** in a graph $G = (V,E)$, $p_{v_0v_t}$ (or $p(v_0,v_t)$) connecting v_0 and v_t is a sequence of the form $(v_0, e_1, v_1, e_2, \cdots, v_{t-1}, e_t, v_t)$, where $v_0, \cdots, v_t$ are vertices and $e_1, \cdots, e_t$ are edges such that $e_i = (v_{i-1}, v_i) \in E$, for $i = 1, 2, \cdots, t$. We refer to v_0, v_t as the endpoints of the path $p_{v_0v_t}$. A path is **simple** if all its vertices and edges are distimct. If $v_0 = v_t$ then the path $p_{v_0v_t}$ is called a **cycle.** A cycle without repeated edges or vertices (except for the endpoints) is called a **simple cycle.** A **directed path** from v_0 to v_t in a digraph $D = (V,E)$ is defined similarly as in the undirected case except that in this case the edges in the path are directed.

A graph is **connected** if any two vertices in it are connected by a path. Connectedness by paths induces an equivalence relation on the vertices. Its classes are called the **connected components** of the graph. A digraph D is called **weakly connected** if its underlying undirected graph is connected. D is **strongly connected** if there is a directed path p_{uv} for any two vertices u and v in D.

A graph $G' = (V',E')$ is a **subgraph** of $G = (V,E)$ if $V' \subseteq V$ and $E' \subseteq E$. If E' is the family of all edges of G which have both ends in V', then G' is said to be induced by V' and denoted by $G'(V')$. A **partial graph** of G contains all its vertices and a subset of its edges. A graph G is an **embedding** of the graph G' if it can be obtained from G' by addition of some edges. A **clique** in the graph $G = (V,E)$ is a set of pairwise adjacent vertices of G. If a graph K is a clique having p vertices we call it the **complete** graph on p vertices and denote it by K_p. A subset S of V is a **separator** of a graph $G = (V,E)$ if the subgraph $G(V \setminus S)$ induced by $V \setminus S$ has two or more connected components. A graph $G = (V,E)$ is called **k-connected** (or sometimes k-point-connected) if the cardinality of any minimal separator of G is at least k.

A **contraction of an** edge $e = (u,v)$ in $G = (V, E)$ is obtained by removing $e = (u,v)$ from E and identifying the vertices u and v. A graph G is **contractible** to a graph G' if G' can be obtained from G by a sequence of edge contractions in G. Edge **extraction** of e in G results in a graph $G \setminus e$ with the same vertex set as G and the edge family $E \setminus \{e\}$. A graph H is a **minor** of a graph G if it can be obtained from G by a finite number of contractions and edge extraction operations.

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be any two graphs. G_2 is **homeomorphic** to G_1 if there exist mappings Ψ and Θ with Ψ being from E_1, onto a set of paths of G_2 and Θ from V_1 into V_2, such that for each edge $(v,w) \in E_1$, the path $\Psi((v,w))$ has ends $\Theta(v)$ and $\Theta(w)$, and no two paths $\Psi((v_1, w_1))$ and $\Psi((v_2, w_2))$ share a vertex except possibly an end of paths.

Let $\mathcal{S} = \{S_1, S_2, \cdots, S_n\}$ be a family of distinct nonempty subsets of S whose union is S. The **intersection graph** of $\mathcal{S}$ is a graph whose vertices are identified with sets in $\mathcal{S}$, with S_i and S_j adjacent whenever $i \neq j$ and $S_i \cap S_j \neq \emptyset$.

In general, graphs and directed graphs are very useful in representing systems or structures that may be considered as set of elements, certain pairs of which are related in a specific way. However, in many applications arising in business, engineering, biological and social sciences, graphs and digraphs are not sufficient to adequately represent the system or structure under study. In such cases, usually numerical values are attached to the vertices and/or edges to represent construction cost, flow capacities, probabilities of destruction and so on, see, for example Ford and Fulkerson (1962). In general, any graph to which such additional structure has been added is called network. For example, a network $G = (V, E)$ is a weighted graph such that each edge $e \in E$ has a 'length' (weight) $w(e)$. For any $u, v \in V$, the length of the path $p(u,v)$ is the sum of its edge weights. Distance between two vertices u and v is defined as the length of the shortest simple path $p(u,v)$.

1.2 COMBINATORIAL OPTIMIZATION PROBLEMS

An **instance of an optimization problem** is a pair (F,c), where F is the set of feasible points and c is the cost function, $c : F \rightarrow R^1$. The problem is to find an $f \in F$ for which $c(f) \leq c(g)$ for all $g \in F$. Such a point f is called an **optimal solution** to the given instance or, for short, an optimal solution. An **optimization problem** is a set I of instances of an optimization problem. For example an instance of a **minimum spanning tree** problem has a given integer $n > 0$ and $n \times n$ symmetric distance matrix (d_{ij}). The problem is to find a spanning tree, (V,E) on n vertices that has minimal total edge length. (By a spanning tree we mean an undirected graph (V,E) that is connected and has no cycles.) For this example $F = \Big\{$all spanning trees (V,E), with $V = \{1,\cdots,n\}\Big\}$ and $c : (V,E) \rightarrow \sum_{(i,j)\in E} d_{ij}$.

One can naturally divide optimization problems into continuous optimization problems and discrete (combinatorial) optimization problems. In combinatorial problems, we usually look for an object from a finite or possibly infinite set. The challenge in combinatorial optimization is to develop algorithms for which the number of elementary computational steps used to obtain an optimal solution of an instance of the problem is acceptably small. Complexity theory categorizes problems in terms of the form of mathematical functions expressing amounts of computation as a function of instance size. If, for example, some algorithm uses a number of elementary operations that can be expressed as third order polynomial of the problem size, n , we say that the algorithm is order n^3, or simply $O(n^3)$. Generally accepted measure of efficient performance of an algorithm in the realm of combinatorial optimization is that of 'polynomial boundness'. An algorithm is considered 'good' if the required number of elementary computational steps to solve an instance of the problem is bounded by a polynomial in the size of the problem. For a given optimization problem a closely related recognition version of the problem can be defined. Given an instance (F,c) of an optimization problem and an integer L , is there a feasible solution $f \in F$ such that $c(f) \leq L$. The class of recognition problems that can be solved by a polynomial time algorithm is called P. A recognition problem A is in the class NP if for a *yes* instance x of A , there exists a certificate for x , whose length is bounded by a polynomial in the size of x and which can be checked for validity in polynomial time. Clearly P is contained in NP. However it is an outstanding open problem whether P = NP. It is strongly suspected that P $\neq$ NP . Let A and B be two recognition problems. It is said that A reduces in polynomial time to B if and only if there exists a polynomial time algorithm $\mathcal{A}$ for A that uses several times as a subroutine at unit cost a (hypothetical) algorithm $\mathcal{B}$ for B . The algorithm $\mathcal{A}$ is called a polynomial-time reduction of A to B . A recognition problem A polynomially transforms to another recognition problem B if, given any string x , we can construct a string y in polynomial time (in the size of x) such that x is a *yes* instance of A if and only if y is a *yes* instance of B . A recognition problem $A \in$ NP is said to be NP-complete if all other problems in NP polynomially transform to A . Problem A is called NP-hard if it can be shown that all problems in NP polynomially reduce to A but it is unknown whether A belongs to NP. Problem that is NP-hard is as hard as any problem in NP. For precise definitions of complexity of an algorithm, P, NP , NP-completeness and NP-hardness, see, for example, Papadimitriou and Steiglitz (1982) and Garey and Johnson (1979). It has been shown that many NP-hard graph problems become solvable in polynomial time when restricted to special structure graphs. Such special graphs are presented in the next section. Below we list several well-known NP-hard problems (Garey and Johnson (1979) to which we further refer in the sequel.

Shortest Path Problem (allowing negative cycles)

$G = (V,E)$ is a weighted graph, such that each edge $e \in E$ has a 'length' (weight) $w(e) \in R^1$. For a given pair of vertices u , $v \in V$ find the length of a shortest simple path $p(u,v)$.

Longest Path

Let $G = (V,E)$ be a weighted graph, such that for each edge $e \in E$, $w(e) \in R^+$. For u , $v \in V$ find the longest simple path $p(u,v)$.

Steiner Tree Problem

Given a connected undirected weighted graph $G = (V,E)$ and a set of vertices D, $D \subseteq V$; find a tree T in G, whose vertex set contains D with a minimum total edge-weight.

Maximum Independent Set

For a given graph $G = (V,E)$ find the maximum size of an independent set. (An independent set in a graph $G = (V,E)$ is a subset $V' \subseteq V$ such that for all u , $v \in V'$, the edge (u,v) is not in E.)

Minimum Dominating Set

For a given graph $G = (V, E)$ compute the minimum size of a dominating set. (A set $D \subseteq V$ is dominating in G if every vertex in V is either in D or it is adjacent to a vertex in D.)

Chromatic Number

For a given graph $G = (V, E)$ compute the size of a smallest set L of colors such that V can be colored with elements from L in such a way that no two adjacent vertices have the same color.

Hamiltonian cycle

For a given graph $G = (V, E)$, decide whether there exist a simple cycle passing through every vertex.

Simple Plant Location Problem in a Graph

Let $G = (V, E)$ be an undirected weighted graph with $w(e) \geq 0$ for all $e \in E$. With each vertex $v \in V$ associate a fixed cost, c_v, to open a facility, and a transportation cost function $g_v(\,.\,)$ which is a nondecreasing function of its argument. Each vertex is viewed as both a potential facility site as well as a customer. Find a subset of vertices F, $F \subseteq V$, that minimizes :

$$\sum_{v \in F} c_v + \sum_{v \in V} g_v(d(v,F)), \quad \text{where } d(v,F) = min_{f \in F} d(v,f).$$

1.3 SPECIAL CLASSES OF GRAPHS

We describe bellow several classes of graphs which were extensively studied in the literature and some of which can be viewed as special cases of the class of partial k-tree graphs. Many optimization problems which are NP-hard for general graphs were shown to be polynomially solvable when restricted to these classes of graphs.

Trees are possibly the simplest class of graphs. Trees are undirected, connected graphs without cycles. **Forests** are undirected graphs that are acyclic but not necessarily connected. $G = (V, E)$ is an **almost tree** with a parameter smaller than or equal to k if and only if for some spanning tree T of G, in each biconnected component of G there are at most k edges of G that are not in T.

Let $G = (V, E)$ be a graph, with $|V| = n$. A linear ordering of G is a bijective mapping $f\colon V \rightarrow \{1, \cdots, n\}$. A linear ordering f of G is said to have bandwidth k if $k = max_{u,v' \varepsilon V} | f(u) - f(v) |$. The **bandwidth** k of G, denoted by bandwidth(G), is the minimum bandwidth of a linear ordering f over all possible orderings of G.

Series-Parallel graphs are undirected graphs that can be constructed recursively from its edges by series and parallel compositions. (Definitions of series and parallel compositions are given in chapter II). **Planar** graphs are those graphs that do not contain K_5 or $K_{3,3}$ homeomorphs. (They can be drawn in the plane in such a way that no two lines representing edges intersect except at a common endpoint.) **Outerplanar** (or 1-outerplanar) graphs are graphs that do not contain graph homemorphic to K_4 or $K_{2,3}$. (Or equivalently they are planar graphs that can be drawn in the plane in such a way that all their vertices are on the same face.) For $k \geq 2$, a graph is k-

outerplanar if and only if it is planar and it has a planar embedding such that if all vertices on the exterior face and all adjacent edges are deleted, then the connected components of the remaining graph are all $(k-1)$-outerplanar.

A graph $G = (V, E)$ is a **Halin graph** if it can be obtained by embedding a tree without vertices of degree 2 in the plane and connecting its leaves by a cycle that crosses none of its edges. $G = (V, E)$ is a **chordal graph** if and only if every cycle with length exceeding three has an edge joining two non-consecutive vertices in the cycle.

k-trees can be defined as follows. The k-clique is a k-tree, and a k-tree of more than k vertices can be constructed by adding a new vertex and new edges connecting it to all vertices of some k-clique of a smaller k-tree. **Partial k-trees** are partial graphs of k-trees.

Many NP-complete problems were solved in polynomial time on trees and forests, using mostly dynamic programming approaches, see, for example, Johnson (1985). Linear time algorithms for finding maximum independent set and minimum dominating set on almost k-trees were developed in Corneil and Perl (1984) and Coppersmith and Vishkin (1985). The technique used therein is to subdivide the problem into a collection of subproblems on the biconnected components, and for each of these to use a tree algorithm as a subroutine. The number of such calls vary, depending on the way how the excess edge influence the solution. Monien and Sudborough (1981) use a dynamic programming approach to show that a number of problems on bandwidth restricted graphs have polynomial solutions. Most of the NP-complete problems remain NP-complete when restricted to planar graphs (Johnson (1985)). However, many NP-complete problems can be solved in polynomial time on series parallel graphs. Indeed, Takamizawa et. al. (1982) provide a general approach to characterize a family of NP-complete problems, that are solvable in linear time on two terminal series-parallel graphs. They consider graph properties defined by means of a finite set of graphs which are forbidden configurations in graphs with these properties. Among other problems they solve the maximum independent vertex set problem, the minimum dominating set problem and the Steiner tree problem. In Bern et. al. (1985) a different general approach was taken for solving combinatorial optimization problems. Namely a general methodology for constructing linear time algorithms is developed therein for graphs which are defined by certain rules of composition. Their approach is applicable for a wide range of problems on series-parallel graphs, outerplanar graphs, Halin graphs, and bandwidth-limited graphs. Chordal graphs can be characterized in terms of 'perfect elimination ordering'. That is, a graph $G = (V,E)$ with $|V| = n$ is a chordal graph if and only if there exist an ordering of the vertex set V, say $v_1, v_2, \cdots, v_n$, such that in the vertex induced subgraph $G(V \setminus \cup \{ v_j : j = 1, \cdots, i-1 \})$ of G, $i \geq 1$, v_i and its adjacent nodes therein form a clique (Golumbic (1980)). A perfect elimination order was used to develop polynomial algorithms for many graph problems that are NP-complete for general inputs (Johnson (1985)). Note that k-trees are chordal graphs (although not all chordal graphs are k-trees).

Arnborg and Proskurowski (1989) have analyzed the complexity of NP-complete problems on partial k-trees, and developed linear time algorithms for the following problems: vertex cover, independent set, dominating set, graph k-colorability, Hamiltonian cycle and network reliability. They have shown that partial k-trees are k-decomposable graphs. (A graph G is k-decomposable if and only if either it has at most $k+1$ vertices, or there is a separator S of G with at most k vertices such that each of the connected components of $G \setminus S$ augmented by S with completely connected vertices is k-decomposable.) Since the component partial k-trees interact only through the separators, solutions to subproblems on the component partial k-trees may be combined to form a solution for the given partial k-tree. Using the k-decomposable property of partial k-trees Arnborg and Proskurowski (1989) and Corneil and Keil (1987) developed a general algorithm paradigm to solve efficiently some difficult problems on graphs with bounded decomposability. We note that since k-trees are chordal graphs these results are related to those obtained by Gavril (1972) in which polynomial algorithms were developed for solving the problems of finding maximum independent set, minimum coloring, maximum clique and minimum clique covering for chordal graphs.

Observe that the class of partial k-trees can be viewed as a unifying framework for some of the special graphs discussed above. Indeed, every graph $G = (V, E)$ is a partial k-tree for some k (in the worst case $k = |V| - 1$). Moreover, with many well known classes of graphs one can associate an upper bound, k_1 , such that these classes can be viewed as partial k-trees for some k not exceeding k_1 . Table 1.1 provides such an upper bound k_1 for several classes of graphs. (A more detailed overview of these results is presented in Bodlaender (1986).

Class of graphs	k_1
Trees, forests	1
Almost trees with parameter k	$k + 1$
Graphs with bandwidth at most k	k
Series parallel graphs	2
Outerplanar graphs	2
Halin graphs	5
k-outerplanar graphs	$3^k - 1$
chordal graphs with max. clique size k	k

Table 1.1 Upper bound of k_1 for special classes of graphs

1.4 RECOGNITION OF k-TREES AND PARTIAL k-TREES

It is easy to check in linear time whether a given graph G is a k-tree by simply removing successively k-degree vertices (every k-tree has at least two such vertices) and its adjacent edges until the graph G becomes empty. Thus if we can reduce a graph G to an empty graph by such an elimination process it is a k-tree. However the recognition of partial k-trees is much more difficult and the characterization of partial k-trees as decomposable graphs (see Section 1.3) does not seem to provide an efficient algorithm for the recognition of this class of graphs. The separator property is lost in partial k-trees. Namely, in a partial k-tree we may find a 'small' separator which cannot be extended to a k-clique in an embedding of this partial k-tree into a k-tree. For example, if we extend the 3-separator $S = \{s_1, s_2, s_3\}$ of the partial 3-tree G in Fig 1.2 to a 3-clique, we obtain a graph which contains K_5 as a subgraph and thus is not a partial 3-tree.

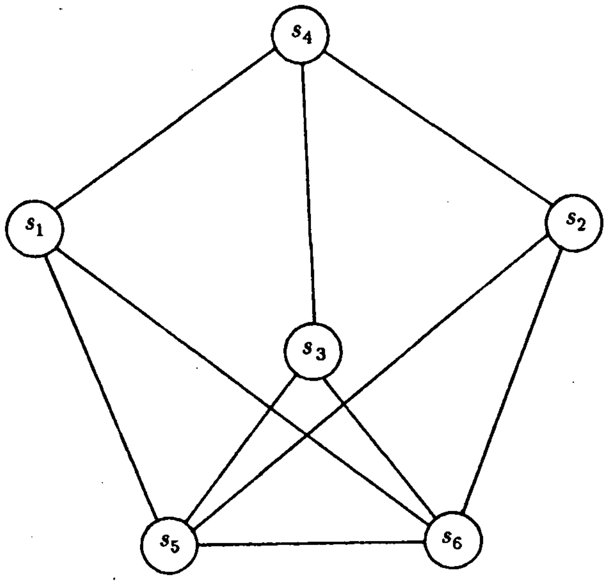


Fig 1.2 Partial 3-tree G

Therefore, since the various polynomial algorithms which solve NP-hard optimization problems on partial k-trees (like vertex cover, chromatic number or graph reliability) require a suitable embedding of the partial k-tree into a k-tree, efforts are devoted to recognize efficiently partial k-trees and find their embeddings into k-trees. Actually, there are two types of recognition problems. The first one is of finding an embedding, of a partial k-tree into a k-tree and the second problem is of finding the minimal value of k, for which the given graph is a partial k-tree. This second recognition problem is important since the above algorithms, though linear in the size of the input, are exponential in k. For $k = 2$ and 3, the recognition and embedding problem has been solved by finding a set of reductions on graphs such that any graph can be reduced to the empty graph if and only if it is a partial k-tree (see Wald and Colbourn (1983) for $k = 2$ and Arnborg and Proskurowski (1986) for $k = 3$). This set of reductions yield $O(n)$ and $O(n \log n)$ time algorithms for the recognition of partial 2-trees and partial 3-trees, respectively. However, already for the case of $k = 4$ there is no easy known generalization of these methods. Arnborg and Proskurowski (1985) presented an incomplete set of reduction rules for partial k-trees which can be used in a heuristic decomposition method. They experimented with these heuristics, and for small values of k (up to 7) most of the graphs which they tested were correctly recognized.

Arnborg et. al. (1987) have shown that the problem of finding the smallest number k such that a given graph is a partial k-tree is NP-complete. However, they also presented therein an $O(n^{k+2})$ time algorithm for the recognition of a partial k-tree when k is fixed. Present research is directed towards a better polynomial time recognition algorithm. Robertson and Seymour (1986) have proved that there exists an $O(n^2)$ algorithm to recognize partial k-trees. However their result is not constructive ; only the existence of such an algorithm is demonstrated and no method is provided to construct it.

With the increasing use of highly parallel computers, it has become desirable to identify optimization problems that can be solved fast in parallel. In particular, there has been a considerable interest in problems which have parallel algorithms running in time bounded by a polynomial in the logarithm of the size of the input (i.e. , in poly-log time) and using a number of processors polynomial in the input size. Such a problem belong to the class NC (Pippenger (1979)). Some of the reasons for this interest in the class NC are:

(i) NC $\subseteq$ P. NC algorithms are more efficient compared to their sequential versions. Further they make use of a 'reasonable' amount of hardware, namely, the processors.

(ii) The class NC is 'robust' under reasonable changes in the underlying machine models of parallel computation.

A very basic issue in parallel complexity theory is to find the most appropriate model of a computer. Different models could lead to different computations and the number of steps executed by the same algorithm would be different, depending on the model of machine considered. The basic differences between various models are concerned with how they deal with read and write conflicts of memory access. The widely excepted model of a RAM (Random Access Machine) consists of a read-only input tape, a write-only output tape, a program and a memory (Aho et.al. (1974)). Fortune and Wyllie (1978) proposed the model PRAM (Parallel Random Access Machine) for parallel computations, which is an extension of the RAM model. In a PRAM model simultaneous reads are allowed, while simultaneous writes are not. Other models of shared memory parallel machines have been proposed in the literature, like CRCW (Concurrent Read Concurrrent Write) PRAM (Vishkin (1983)) in which all the processors have access to a common memory and run synchronously and both simultaneous reading as well as writing on common memory locations are allowed. The NC algorithms that we consider in this thesis assume CRCW PRAM model.

Edenbrandt (1985) constructed a parallel algorithm for the recognition of chordal graphs that requires $O(n^3 m)$ processors and takes $O(log\, n)$ time. Chandrasekharan and Iyengar (1986) improved his processor bound to $O(n^4)$. Tseng (1987) constructed an algorithm that recognizes whether a given 2-connected graph is series-parallel in $O(log^3 n)$ time using $O(n^4)$ processors. Bodlaender (1988) has constructed an algorithm for the recognition of graphs that have a tree width less than k (or, equivalently are partial k-trees), which runs in $O(log\, n)$ time and uses $O(n^{3k+4})$ processors. Thus, parallelism has decreased the running time but is quite inefficient in the use of processors.

1.5 SUMMARY OF THE RESULTS

We provide bellow a summary of the results obtained in the thesis. In chapter II, we analyze the fixed cost spanning forest (FCSF) problem , defined over a network $G = (V, E)$ in which the set of communities D , $D \subseteq V$, require service that can be generated at a set of facilities' sites M , $M \subseteq V$. There is a fixed cost for opening each facility and a cost for delivering the service from open facilities to the communities. Any community receiving the service from an open facility can deliver it to adjacent communities. We develop a linear time algorithm for solving the FCSF problem when the underlying network G has a series parallel structure. We further analyze a related cost allocation problem, in which we seek a fair method for allocating the cost of providing the service to the customers. We formulate this cost allocation problem as a cooperative game and show that, in general, the core of this cooperative game may be empty. However, we provide a sufficient condition, which can be verified in polynomial time, for the nonemptiness of the core of this game.

It is well known that a k-tree can be reduced to the k-complete graph by sequentially removing k-degree vertices with completely connected neighbors. In the third chapter we use this reduction process to develop efficient algorithms for several optimization problems on k-trees and partial k-trees. The complexity of algorithms developed on k-trees and partial k-trees the thesis is expressed in terms of number of vertices and k is assumed throughout to be a fixed constant. Observe that the number of edges of a k-tree is linear in the number of vertices for a fixed k. We develop a linear time algorithm to find shortest simple paths from a given vertex to all other vertices in a k-tree, we compute the diameter (the longest path) and the vertex center of a k-tree with equal edge lengths in linear time and we construct an $O(n^{k+2})$ algorithm to solve the Simple Plant Location problem in an n-vertex partial k-tree.

In chapter IV we study the notion of k-cable and k-path in a k-tree. The **k-cable**, $C(u,v)$, is a collection of k vertex-disjoint paths between vertices u and v in a k-connected graph. We define the **length** of the k-cable, $C(u,v)$, as the sum of the lengths of the k paths in $C(u,v)$ and the **k-cable distance** between two vertices u and v is the length of the shortest k-cable between these two vertices. A **k-path** is an alternating sequence, KP, of distinct k and $k+1$ complete subgraphs $KP = < Q_1 , T_2 , Q_2 , T_3 , \cdots , T_n , Q_n >$, starting and ending with a k-clique and such that T_i contains exactly two of the distinct k-cliques Q_{i-1} and Q_i. A k-path between vertices u and v in a k-tree G is induced by all the minimal k-separators of u and v in G. We provide a new characterization of a k-path between two vertices u and v, in an equal weight k-tree G, in terms of minimal k and $k+1$ cliques with respect to certain partial orders defined on the collections of all k and $k+1$ cliques in G. Further, for an equal weight k-tree $G =$

(V, E) and a positive integer R, $R \geq 2k - 1$, we develop an $O(n^2)$ algorithm to decompose the vertex set V into a minimum number of components, V^1, V^2, $\cdots$, V^l, such that for any pair of vertices i and j, $i, j \in V^t$ and $t = 1, \cdots, l$, the cable distance between i and j does not exceed R. We also develop in chapter IV an algorithm to compute the k-cable diameter of a k-tree with equal edge lengths in linear time.

Finally in chapter V we introduce the notion of a **k-separator** $(k \geq 2)$ of an n-vertex graph $G = (V, E)$. A k-separator is a set of vertices S, $|S| = k$, which induces a partition of $V \setminus S$ into sets V_1 and V_2 satisfying: (i) $|V_i| \leq \frac{k}{k+1} n$, $i = 1, 2$, and (ii) no edge in E connects a vertex in V_1 with a vertex in V_2. We prove the existence of a k-separator for partial k-trees and construct a linear time algorithm for generating such a separator in k-trees. This algorithm can be used to construct a balanced binary decomposition tree of a k-tree in $O(n \log n)$ time. We further derive some other separation properties of partial k-trees which are used to construct a balanced decomposition of an embedding of a k-connected partial k-tree into a k-tree when $k = 2, 3$. Finally, we develop parallel algorithms for the recognition of partial k-trees when $k = 2, 3$. These algorithms require $O(\log^2 n)$ time and use, respectively, $O(n^3)$ and $O(n^4)$ processors.

Chapter II

FIXED COST SPANNING FOREST PROBLEM

Let $G = (V, E)$ be a connected undirected network with a set of vertices V and a set of edges E. The subset D , $D \subseteq V$, is a set of communities and M , $M \subseteq V$, is a set of potential facilities' sites. Open facilities provide service which is required by the communities, and any community receiving the service can in turn deliver it to adjacent communities. Each community in D is required to be connected, perhaps through other communities, to an open facility. There is a cost, $c_i \geq 0$, for opening facility $i \in M$ and a cost $c((i,j)) = c_{ij} \geq 0$, $(i,j) \in E$, if edge (i,j) is used to deliver service. The objective is to provide service to the communities in D at a minimum cost. We will refer to the above optimization problem as the Fixed Cost Spanning Forest (FCSF) problem.

The FCSF problem is equivalent to the Steiner Tree problem on an augmented network G' derived from G by adding a new vertex, denoted by o , and joining it to the set of all potential facilities' sites M . The cost of any newly added edge (o,i) , $i \in M$, in G' is $c_{oi} = c_i$. Clearly, an optimal solution to the FCSF problem can be produced by finding a minimum cost Steiner tree in G' whose vertex set contains $D' = D \cup \{ o \}$. For the other direction suppose that a connected undirected weighted graph $G = (V,E)$ and a set of vertices D , $D \subseteq V$ are given and we want to find a tree T in G, whose vertex set contains D with a minimum total edge weight. To transform this problem to FCSF problem we identify set D with the set of communities and we choose an arbitrary vertex in D and pronounce it a facility site at which a facility can be established at zero cost. Thus, since the Steiner Tree problem is known to be NP-hard (Garey and Johnson (1979)), in general, so is our FCSF problem.

When the underlying network $G = (V, E)$ is a tree, the augmented network $G' = (V', E')$ is a series parallel graph without loops or parallel edges or, equivalently, it is a partial two-tree (Wald and Colbourn (1983)). Therefore, since there exist linear time algorithms for solving the Steiner Tree problem on partial-two trees (see, e.g., Wald and Colbourn (1983), Prodon et al. (1985), Rardin et al. (1982), Takamizawa et al. (1982)) and Cornuejols et. al. (1985) one can employ these algorithms to solve the FCSF problem on a tree network.

The main purpose of this chapter is to develop a linear time algorithm for solving the FCSF problem when the network $G = (V, E)$ is restricted to have a series parallel structure. We further examine a cost allocation problem associated with the FCSF problem, in which one seeks a fair method for allocating the cost of providing the service among the communities.

The rest of the chapter is organized as follows. In Section 2.1 we review standard definitions regarding series-parallel graphs, and in Section 2.2 we develop a linear time algorithm for solving the FCSF problem on a series parallel graph. In Section 2.3 we examine a cost allocation problem associated with the FCSF problem. We formulate this cost allocation problem as a cooperative game and show that, in general, the core of this game may be empty when G is a series parallel network. Nevertheless, we provide a sufficient condition for the nonemptiness of the core of this game, which can be verified in polynomial time.

2.1 PRELIMINARIES

This section reviews several standard graph-theoretic concepts to be employed in the sequel. Series-parallel multigraph (SPM) is associated with two vertices called terminals and can be constructed recursively as follows: A single edge is a SPM; its end vertices are the two terminals. Further, if Q_1 and Q_2 are SPMs with terminals $i(Q_1)$, $j(Q_1)$ and $i(Q_2)$, $j(Q_2)$, respectively, so are the multigraphs obtained by each of the following compositions.

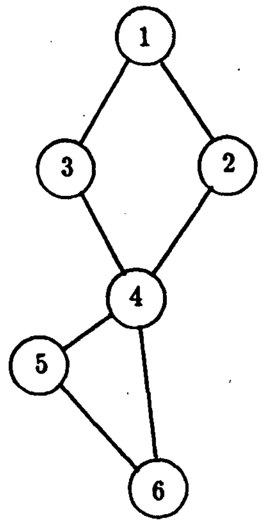
Series composition

Identify a terminal of Q_1 , say $j(Q_1)$, with a terminal of Q_2 , say $i(Q_2)$. The terminals of the new multigraph are $i(Q_1)$ and $\mathrm{j}(Q_2)$.

Parallel composition

Identify one terminal of Q_1 , say $i(Q_1)$, with one terminal of Q_2 , say $i(Q_2)$, and the second terminal of Q_1 , $j(Q_1)$, with $j(Q_2)$. The terminals of the new multigraph are the two vertices where the identifications occur.

The above recursive construction of a SPM G can be represented by a binary decomposition tree (see, e.g., Valdes et al. (1982)). This is a rooted binary tree T in which each vertex Q corresponds to some series-parallel submultigraph of G, denoted by $G(Q)$, obtained as follows. Each vertex of T that has degree one represents a distinct edge of G . If Q is not a degree one vertex, then it is either of a series (S) or parallel (P) type. If Q is a parent of Q_1 and Q_2 , then $G(Q)$ is submultigraph obtained from $G(Q_1)$ and $G(Q_2)$ by the respective series or parallel composition. In particular, if Q is the root of T , $G(Q)$ is the multigraph G . The construction of a binary decomposition tree of a given SPM can be done in linear time, see, e.g., Valdes et al. (1982). Figure 2.1 below illustrates a binary decomposition tree of an SPM.



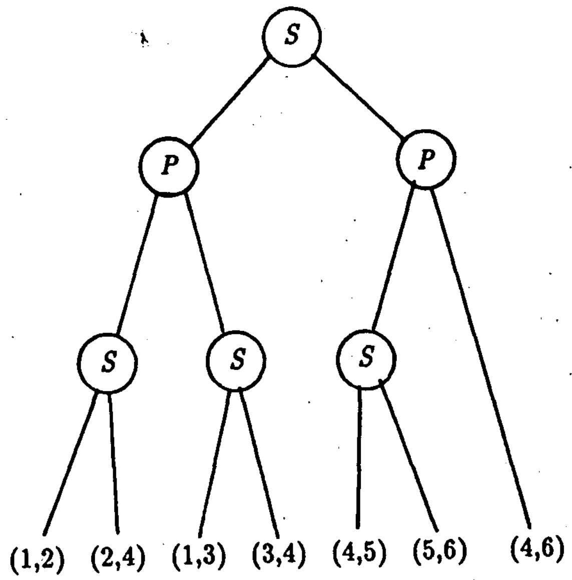


Figure 2.1 a: An SPM G Figure 2.1 b: A binary decomposition tree T of G

An optimal solution , X^* , to the FCSF problem specifies the set of facilities M^*, $M^* \subseteq M$, that are opened at the optimum. Each open facility i , $i \in M^*$, is providing the service to a set of communities through a connected network $T_i^* = (V_i^*, E_i^*)$, which can be assumed to be a tree. The optimal value of the FCSF problem, associated with X^* , is given by :

$$\sum_{i \in M^*} c_i + \sum_{i \in M^*} \sum_{(j,k) \in E_i^*} c_{jk} .$$

The **Directed Steiner Tree** problem, which is closely related to our FCSF problem, is defined with respect to a directed weighted graph $G = (V, E)$. Namely, given a vertex $o \in V$ and a subset of vertices $D \subseteq V$, find a directed tree T in G, rooted away from vertex o and whose vertex set contains D, such that the total edge-weight of T is minimum.

2.2 A LINEAR TIME ALGORITHM

We develop in this section a linear time algorithm for solving the FCSF problem on an SPM $G = (V, E)$ with a set of communities D, $D \subseteq V$, and a set of facilities' sites M, $M \subseteq V$. For simplicity of presentation, we assume that G does not contain parallel edges. The algorithm starts by constructing a binary decomposition tree, T, of the underlying SP graph G. Every vertex q of T designates a series parallel subgraph $Q = (V_Q, E_Q)$ of G, with terminals $i(Q)$ and $j(Q)$. Each such subgraph Q, except for those corresponding to the leaves of T, is either a series or parallel composition of two subgraphs Q_1 and Q_2 of G. That is, if the subgraphs Q, Q_1, Q_2 correspond to vertices q, q_1, q_2, respectively, in T, then q_1 and q_2 are the two immediate successor vertices of q in T, and Q_1 and Q_2 are said to be the sons of Q.

Starting from the leaves of T and proceeding towards its root, for each vertex q of T the algorithm solves certain special cases of the FCSF problem in the corresponding subgraph Q. The solution of these special cases in Q is based on solutions of identical cases in the sons of Q, Q_1 and Q_2, whose composition created Q, and which were previously obtained by the algorithm. At the final stage, when q is the root of T, the corresponding subgraph Q coincides with our SP graph G and an optimal solution for the FCSF on G is at hand.

The special instances of the FCSF problem that we need to solve are derived from some restrictions we impose on the terminals of the SP subgraphs we encounter. Explicitly, for each subgraph Q associated with a vertex q of the binary decomposition tree T, we compute the following optimal values :

I : $f(Q,\emptyset,\emptyset)$,	II : $f(Q,u_Q,\emptyset)$,	III : $f(Q,\emptyset,v_Q)$,
IV : $f(Q,i(Q),\emptyset)$,	V : $f(Q,\emptyset,j(Q))$,	VI : $f(Q,u_Q,v_Q)$,
VII : $f(Q,u_Q,j(Q))$,	VIII : $f(Q,i(Q),v_Q)$,	IX : $f(Q,i(Q),j(Q))$,
X : $f(Q,i(Q),i(Q))$,	XI : $f(Q,j(Q),j(Q))$.	

All the above eleven cases can be viewed as combinations of the following three types of optimal values and their symmetric counterparts : $f(Q,\emptyset,.)$ -optimal value of a FCSF problem in Q when there are no restrictions on the terminal $i(Q)$; $f(Q,u_Q,.)$ - optimal value of a FCSF problem in Q in which it is required that service is delivered to the terminal $i(Q)$ from some facility u_Q , $u_Q \in M \cap V_Q$, and $f(Q,i(Q),.)$ - optimal value of a FCSF problem in Q when a facility can be established at the terminal $i(Q)$ at no cost. Thus, for example, in case VIII we seek the optimal value, $f(Q,i(Q),v_Q)$, of the FCSF problem in Q in which a facility is available at $i(Q)$ at no cost and the service is provided to terminal $j(Q)$ from some facility v_Q , $v_Q \in M' \cap V_Q$ where $M' = M \setminus \{i(Q)\}$, and in case X we seek the optimal value , $f(Q,i(Q),i(Q))$, of the FCSF problem in Q in which an open facility is available at $i(Q)$ at no cost, and the service is provided to terminal $j(Q)$ from the open facility at $i(Q)$.

Now, let $Q = (V_Q, E_Q)$ be any subgraph of G associated with the vertex q of the binary decomposition tree T , and assume that for some optimal solution to the FCSF problem in G vertex v , $v \in V_Q$, receives the service from facility u , $u \notin V_Q$. Then , our algorithm uses throughout the simple observation that the path which is used to deliver the service from u to v

in this optimal solution must pass through at least one of the terminals $i(Q)$ or $j(Q)$ of Q.

Recurrence Rules

In the first step we find $f(Q, . , .)$ for all Q corresponding to the leaves of the decomposition tree T, for all cases I-XI above. That is, using, for example, complete enumeration we solve the above eleven cases of FCSF problems restricted to the different edges of G.

Parallel composition

Let Q_1 and Q_2 be the sons of Q, according to the binary decomposition tree T, such that Q is derived from Q_1 and Q_2 by parallel composition. Then, we identify the terminals of Q_1, Q_2 and Q in accordance with parallel composition, i.e. $i(Q) = i(Q_1) = i(Q_2)$, $j(Q) = j(Q_1) = j(Q_2)$ and we compute $f(Q, . , .)$ as follows :

I $$f(Q,\emptyset,\emptyset) = min \Big\{ [f(Q_1,\emptyset,\emptyset) + f(Q_2,\emptyset,\emptyset)], [f(Q_1,u_{Q_1},\emptyset) + f(Q_2,i(Q_2),\emptyset)], [f(Q_1,i(Q_1),\emptyset) + f(Q_2,u_{Q_2},\emptyset)], [f(Q_1,\emptyset,v_{Q_1}) + f(Q_2,\emptyset,j(Q_2))], [f(Q_1,\emptyset,j(Q_1)) + f(Q_2,\emptyset,v_{Q_2})], [f(Q_1,u_{Q_1},v_{Q_1}) + f(Q_2,i(Q_2),j(Q_2))], [f(Q_1,i(Q_1),j(Q_1)) + f(Q_2,u_{Q_2},v_{Q_2})], [f(Q_1,u_{Q_1},j(Q_1)) + f(Q_2,i(Q_2),i(Q_2))], [f(Q_1,i(Q_1),v_{Q_1}) + f(Q_2,j(Q_2),j(Q_2))], [f(Q_1,i(Q_1),i(Q_1)) + f(Q_2,u_{Q_2},j(Q_2))], [f(Q_1,j(Q_1),j(Q_1)) + f(Q_2,i(Q_2),v_{Q_2})], [f(Q_1,u_{Q_1},j(Q_1)) + f(Q_2,i(Q_2),v_{Q_2})], [f(Q_1,i(Q_1),v_{Q_1}) + f(Q_2,u_{Q_2},j(Q_2))] \Big\}.$$

II $$f(Q,u_Q,\emptyset) = min \Big\{ [f(Q_1,u_{Q_1},\emptyset) + f(Q_2,i(Q_2),\emptyset)], [f(Q_1,i(Q_1),\emptyset) + f(Q_2,u_{Q_2},\emptyset)], [f(Q_1,u_{Q_1},j(Q_1)) + f(Q_2,i(Q_2),v_{Q_2})], [f(Q_1,u_{Q_1},v_{Q_1})$$

$+ f(Q_2, i(Q_2), j(Q_2))]$, $[f(Q_1, i(Q_1), v_{Q_1}) + f(Q_2, u_{Q_2}, j(Q_2))]$, $[f(Q_1, i(Q_1), j(Q_1)) + f(Q_2, u_{Q_2}, v_{Q_2})]$, $[f(Q_1, i(Q_1), i(Q_1)) + f(Q_2, u_{Q_2}, j(Q_2))]$, $[f(Q_1, u_{Q_1}, j(Q_1)) + f(Q_2, i(Q_2), i(Q_2))]$, $[f(Q_1, i(Q_1), v_{Q_1}) + f(Q_2, j(Q_2), j(Q_2))]$, $[f(Q_1, j(Q_1), j(Q_1)) + f(Q_2, i(Q_2), v_{Q_2})] \Big\}$.

III $f(Q, \emptyset, v_Q)$ analogous to II .

IV $f(Q, i(Q), \emptyset) = min \Big\{ [f(Q_1, i(Q_1), \emptyset) + f(Q_2, i(Q_2), \emptyset)]$, $[f(Q_1, i(Q_1), v_{Q_1}) + f(Q_2, i(Q_2), j(Q_2))]$, $[f(Q_1, i(Q_1), j(Q_1)) + f(Q_2, i(Q_2), v_{Q_2})]$, $[f(Q_1, i(Q_1), i(Q_1)) + f(Q_2, i(Q_2), j(Q_2))]$, $[f(Q_1, i(Q_1), j(Q_1)) + f(Q_2, i(Q_2), i(Q_2))] \Big\}$.

V $f(Q, \emptyset, j(Q))$ analogous to IV .

VI $f(Q, u_Q, v_Q) = min \Big\{ [f(Q_1, u_{Q_1}, v_{Q_1}) + f(Q_2, i(Q_2), j(Q_2))]$, $[f(Q_1, i(Q_1), j(Q_1)) + f(Q_2, u_{Q_2}, v_{Q_2})]$, $[f(Q_1, u_{Q_1}, j(Q_1)) + f(Q_2, i(Q_2), v_{Q_2})]$, $[f(Q_1, i(Q_1), v_{Q_1}) + f(Q_2, u_{Q_2}, j(Q_2))]$, $[f(Q_1, u_{Q_1}, j(Q_1)) + f(Q_2, i(Q_2), i(Q_2))]$, $[f(Q_1, i(Q_1), v_{Q_1}) + f(Q_2, j(Q_2), j(Q_2))]$, $[f(Q_1, i(Q_1), i(Q_1)) + f(Q_2, u_{Q_2}, j(Q_2))]$, $[f(Q_1, j(Q_1), j(Q_1)) + f(Q_2, i(Q_2), v_{Q_2})] \Big\}$.

VII $f(Q, u_Q, j(Q)) = min \Big\{ [f(Q_1, u_{Q_1}, j(Q_1)) + f(Q_2, i(Q_2), j(Q_2))]$, $[f(Q_1, i(Q_1), j(Q_1)) + f(Q_2, u_{Q_2}, j(Q_2))] \}$.

VIII $f(Q, i(Q), v_Q)$ analogous to VII .

IX $f(Q, i(Q), j(Q)) = f(Q_1, i(Q_1), j(Q_1)) + f(Q_2, i(Q_2), j(Q_2))$.

X $f(Q, i(Q), i(Q)) = min \Big\{ [f(Q_1, i(Q_1), i(Q_1)) + f(Q_2, i(Q_2), j(Q_2))] , [f(Q_1), i(Q_1), j(Q_1)) + f(Q_2, i(Q_2), i(Q_2))] \Big\}$.

XI $f(Q, j(Q), j(Q)) = min \Big\{ [f(Q_1, \mathrm{i}(Q_1), \mathrm{j}(Q_1)) + f(Q_2, j(Q_2), j(Q_2))] , [f(Q_1, j(Q_1), j(Q_1)) + f(Q_2, i(Q_2), j(Q_2))] \Big\}$.

Series Composition

Let Q_1 and Q_2 be the sons of Q, according to the binary decomposition tree, with corresponding terminals $i(Q_1)$, $j(Q_1)$ and $i(Q_2)$, $j(Q_2)$, respectively, such that Q is derived from Q_1 and Q_2 by series composition. Then, the terminals of Q are $i(Q) = i(Q_1)$ and $j(Q) = j(Q_2)$. (Note that when the composition took place, the terminals $j(Q_1)$ and $i(Q_2)$ were identified, i.e., $j(Q_1) = i(Q_2)$.) We then compute $f(Q , . , .)$ from values of $f(Q_1, . , .)$ and $f(Q_2, . , .)$, previously computed, as follows :

I $f(Q, \emptyset, \emptyset) = min \Big\{ [f(Q_1, \emptyset, \emptyset) + f(Q_2, \emptyset, \emptyset)] , [f(Q_1, \emptyset, v_{Q_1}) + f(Q_2, i(Q_2), \emptyset)] , [f(Q_1, \emptyset, j(Q_1)) + f(Q_2, u_{Q_2}, \emptyset)] \Big\}$.

II $f(Q, u_Q, \emptyset) = min \Big\{ [f(Q_1, u_{Q_1}, \emptyset) + f(Q_2, \emptyset, \emptyset)] , [f(Q_1, u_{Q_1}, v_{Q_1}) + f(Q_2, i(Q_2), \emptyset)] , [f(Q_1, u_{Q_1}, j(Q_1)) + f(Q_2, u_{Q_2}, \emptyset)]$,

$[f(Q_1, j(Q_1), j(Q_1)) + f(Q_2, u_{Q_2}, \emptyset)] \}.$

III $\quad f(Q, \emptyset, v_Q)$ analagous to II .

IV $\quad f(Q, i(Q), \emptyset) = min \Big\{ [f(Q_1, i(Q_1), \emptyset) + f(Q_2, \emptyset, \emptyset)] , [f(Q_1, i(Q_1), u_{Q_1}) + f(Q_2, i(Q_2), \emptyset)] , [f(Q_1, i(Q_1), j(Q_1)) + f(Q_2, u_{Q_2}, \emptyset)] , [f(Q_1, i(Q_1), i(Q_1)) + f(Q_2, i(Q_2), \emptyset)] \Big\} .$

V $\quad f(Q, \emptyset, j(Q))$ analogous to IV .

VI $\quad f(Q, u_Q, v_Q) = min \Big\{ [f(Q_1, u_{Q_1}, \emptyset) + f(Q_2, \emptyset, v_{Q_2})] , [f(Q_1, u_{Q_1}, v_{Q_1}) + f(Q_2, i(Q_2), v_{Q_2})] , [f(Q_1, u_{Q_1}, v_{Q_1}) + f(Q_2, i(Q_2), i(Q_2))] , [f(Q_1, u_{Q_1}, j(Q_1)) + f(Q_2, u_{Q_2}, v_{Q_2})] , [f(Q_1, j(Q_1), j(Q_1)) + f(Q_2, u_{Q_2}, v_{Q_2})] \Big\} .$

VII $\quad f(Q, u_Q, j(Q)) = min \Big\{ [f(Q_1, u_{Q_1}, \emptyset) + f(Q_2, \emptyset, j(Q_2))] , [f(Q_1, u_{Q_1}, j(Q_1)) + f(Q_2, j(Q_2), j(Q_2))] , [f(Q_1, u_{Q_1}, j(Q_1)) + f(Q_2, u_{Q_2}, j(Q_2))] , [f(Q_1, u_{Q_1}, v_{Q_1}) + f(Q_2, i(Q_2), j(Q_2))] \Big\} .$

VIII $\quad f(Q, i(Q), v_Q)$ analagous to VII .

IX $\quad f(Q, i(Q), j(Q)) = min \Big\{ [f(Q_1, i(Q_1), \emptyset) + f(Q_2, \emptyset, j(Q_2))] , [f(Q_1, i(Q_1), v_{Q_1}) + f(Q_2, i(Q_2), j(Q_2))] , [f(Q_1, i(Q_1), i(Q_1)) + f(Q_2, i(Q_2), j(Q_2))] , [f(Q_1, i(Q_1), j(Q_1)) + f(Q_2, u_{Q_2}, j(Q_2))] , [f(Q_1, i(Q_1), j(Q_1)) +$

$$f(Q_2, j(Q_2), j(Q_2))]\Big\} .$$

X $\quad f(Q, i(Q), i(Q)) = f(Q_1, i(Q_1), i(Q_1)) + f(Q_2, i(Q_2), i(Q_2))$.

XI $\quad f(Q, j(Q), j(Q)) = f(Q_1, j(Q_1), j(Q_1)) + f(Q_2, j(Q_2), j(Q_2))$.

Validity of The Algorithm

Lemma 2.2.1 The algorithm solves the FCSF problem on a series-parallel graph.

Proof: At each iteration, starting from the leaves, the algorithm solves cases I-XI for some subgraph Q associated with a vertex q in the binary decomposition tree T. Clearly, after a finite number of iterations we reach the root of T whose associated subgraph coincides with our series parallel graph G. Then, $f(G, \emptyset, \emptyset)$ is the optimal value of the FCSF problem on G. □

Complexity of The Algorithm

Lemma 2.2.2 For a SP graph $G = (V, E)$ with n vertices $(|V| = n)$, a set of communities $D \subseteq V$ and a set of potential facilities' sites $M \subseteq V$, the FCSF problem can be solved in time which is linear in n .

Proof: The binary decomposition tree T can be constructed in linear time (Valdes et al. (1982)).

Further , T has $m \leq 2n - 3$ leaves and $(m - 1)$ vertices which are of degree greater than 1. The proof then follows since the amount of computation associated with each vertex of T requires a constant time. $\square$

Comment Berne et al. (1985) have developed a general methodology for constructing linear time dynamic programming algorithms for a certain class of graph optimization problems on decomposable graphs. Their methodology is applicable to our FCSF problem on SP graphs, and an algorithm based on this methodology would be similar to the algorithm presented here. Although this chapter has less general scope, it has the advantage that the recurrence rules are determined explicitly which is naturally required for any implementation. Moreover, as is shown in the next section, an optimal solution for the FCSF problem can be employed in order to verify, in polynomial time, whether a sufficient condition for the nonemptiness of the core of a cost allocation game associated with this problem is satisfied.

2.3 THE CORE OF THE FCSF GAME ON SP GRAPHS

We examine in this section a cost allocation problem associated with the FCSF problem, and for this purpose we need first to introduce the following game theoretic definitions and notation. Let $N = \{ 1 , 2 , \cdots , n \}$ be a finite set of **players** and let $c : 2^N \rightarrow R$ with $c(\emptyset) = 0$ be a **characteristic function** defined over subsets of N referred to as **coalitions.** If $c(N)$ designates a cost that has to be shared by all the players then the pair $(N; c)$ is called a **(cost) cooperative game,** or simply a **game.** For $x \in R^n$ and $S \subseteq N$, let $x(S) \equiv \sum_{j \in S} x_j$. A **cost allocation** vector x in a game $(N; c)$ satisfies $x(N) = c(N)$, and the solution theory of cooperative games is concerned with the selection of a reasonable subset of cost allocation vectors.

Central to the solution theory of cooperative games is the concept of solution referred to as the core of a game. The **core** of a game $(N;c)$ consists of all vectors $x \in R^n$ such that $x(S) \leq c(S)$ for all $S \subseteq N$, and $x(N) = c(N)$. Observe that the core consists of all allocation vectors x which provide no incentive for any coalition to secede. In general, the core of a game may be empty.

The cost allocation problem associated with the FCSF problem is concerned with the allocation of the cost incurred for satisfying the communities' demand for service. We will formulate this cost allocation problem as a cooperative game and analyze its core. Formally, given a FCSF problem, defined on $G = (V,E)$ with a set of communities $D \subseteq V$ and a set of potential facilities' sites $M \subseteq V$, we denote by $FCSF_S$, for $S \subseteq D$, the FCSF problem obtained from the original problem by simply replacing D by S. Then, the pair $(D;c)$, where $c : 2^{|D|} \rightarrow R$ is such that $c(\emptyset) = 0$ and for each $S \subseteq D$, $c(S)$ is the minimum objective function value of $FCSF_S$, is a cooperative game in characteristic function form, to be referred to as the **Fixed Cost Spanning Forest (FCSF) game**.

In general, the FCSF game generalizes several cooperative games studied in the literature which were used to analyze a variety of cost allocation problems. For example, the FCSF game generalizes Littlechild's (1974) airport game and Megiddo's (1978) tree game. When $D = V$, the core of the FCSF game coincides with the core of the minimum cost spanning tree (m.c.s.t.) game (associated with the augmented network G'), which is known to have a nonempty core, see, e.g., Bird (1976) and D. Granot and G. Huberman (1981). For the case when $D \neq V$, D. Granot and F. Granot (1987) have shown that the core of the FCSF game is not empty provided that G has a tree structure. Unfortunately, this result cannot be extended to cases when G has a more general structure than a tree. Indeed, we provide below an example demonstrating that the core of a FCSF game defined over a graph consisting of a simple cycle may be empty.

Explicitly, consider the network $G = (V, E)$ shown in Figure 2.2 below. Therein, $V = \{1,2,3,4,5,6\}$, $E = \{(1,4), (4,2), (2,5), (5,3), (3,6), (6,1)\}$, $D = \{1,2,3\}$ is the set of communities and $M = \{4,5,6\}$ is the set of possible facilities' sites. Further, we assume that $c_{ij} = 1$ for all $(i,j) \in E$ and that $c_i = 1$ for all $i \in M$.

The core, C, of the FCSF game associated with G is given by:

$$C = \{ x \in R^3 : x_1 \leq 2 , x_2 \leq 2 , x_3 \leq 2 , x_1 + x_2 \leq 3 , x_1 + x_3 \leq 3 ,$$
$$x_2 + x_3 \leq 3 , x_1 + x_2 + x_3 = 5 \} .$$

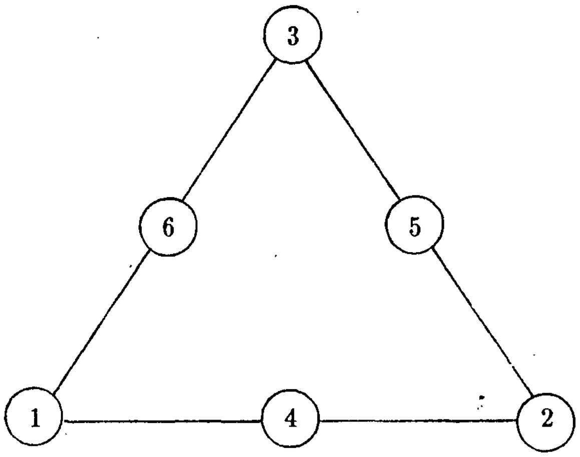


Figure 2.2 $G = (V, E)$

Now, one can easily verify that the core constraints induced by the three two-members coalitions imply that $x_1 + x_2 + x_3 \leq 4\frac{1}{2}$ for any core allocation. Thus, since any core allocation x must distribute the entire cost, i.e., $x_1 + x_2 + x_3 = 5$, we conclude that the core of the FCSF game associated with G , displayed in Figure 2.2 , is empty.

The proof of the nonemptiness of the core of a FCSF game when G is a tree, provided by D. Granot and F. Granot (1987), employs the polyhedral characterization of the Directed Steiner Tree problem in series-parallel graphs given by Prodon et al. (1985). To describe this polyhedral characterization, as applied to our FCSF problem, we first need to replace the graph $G = (V, E)$ with the directed graph $\hat{G} = (\hat{V}, \hat{E})$ in which all undirected edges (i,j) are replaced by two directed edges $<i,j>$ and $<j,i>$, with associated costs $\hat{c}_{ij} = \hat{c}(<i,j>) = \hat{c}(<j,i>) = \hat{c}_{ij} = c_{ij}$. Then, we construct the augmented network $\hat{G}' = (\hat{V}', \hat{E}')$ derived from $\hat{G}$ by adding to $\hat{V}$ the vertex o , and to $\hat{E}$ the directed edges $<o,i>$ for all $i \in M$, with $\hat{c}_{oi} = \hat{c}(<o,i>) = c_i$. We will refer to $\hat{G}'$ as the **directed augmented network** derived from G . Note that finding a minimum cost directed Steiner tree in $\hat{G}'$ which is rooted away from o and whose vertex set contains D , is equivalent to solving the FCSF problem in G .

Further, we need the following notation; for a directed edge $l = <i,j>$ we refer to i as the **tail** and j as a **head** of l , and for a subset of vertices S we denote by $\delta(S)$ the set of all directed edges having their heads, but not their tails, in S . A subset S , $S \subseteq \hat{V}'$, is said to be an **admissible cut-set** of $\hat{G}'$, if $o \notin S \subseteq \hat{V}'$, $S \cap D \neq \emptyset$ (D is the set of communities) and both subgraphs $\hat{G}'(S)$ and $\hat{G}'(V \setminus S)$ of $\hat{G}'$ induced by S and $V \setminus S$, respectively, are connected. We denote by $A_{\hat{G}'}$ the set of all admissible cut-sets of $\hat{G}'$.

It follows from Prodon et al. (1985) that if the directed augmented network $\hat{G}'$ is a series parallel graph, then the incidence vector of a minimum cost directed Steiner tree of $\hat{G}'$, directed away from vertex o and whose vertex set contains $Q \cup \{o\}$, for any $Q \subseteq D$, is an optimal solution of the linear programming problem, $LP(Q)$, defined as follows:

$$LP(Q): \ min\left\{ cx: \ x(\delta(S)) \geq 1 \, , S \in A_{\hat{G}'} \, , \ S \cap Q \neq \emptyset \, , \ x \geq 0 \right\} \tag{2.1}$$

In fact, in view of favorable computational results obtained by Prodon et al. (1985) with instances of the Directed Steiner Tree problem on more general graphs than two-trees, they were led to conjecture that the inequalities describing the polyhedron of the Directed Steiner Tree problem on two-trees (i.e. , the feasible region of $LP(D)$) , while not sufficient to describe that polyhedron for the Directed Steiner Tree problem on more general graphs than two-trees, are nevertheless important in the sense that they often produce optimal integer extreme points. However, notwithstanding this favorable computational experience, $LP(D)$ would fail to produce optimal directed Steiner trees even for the very special case in which G is a partial two-tree. Indeed, let the underlying network G be the simple cycle in Figure 2.2 and consider the directed augmented network $\hat{G}'$ derived from G. All the edge weights in $\hat{G}'$ are assumed to be 1. Then, it is easy to check that the minimum cost directed Steiner tree in $\hat{G}'$, with root o and whose vertex set contains vertices o , 1 , 2 ,3 , has a weight of 5 . However $x^* \in R_+^E$ defined as follows:

$$x^*_{(0,4)} = \tfrac{1}{2} , \qquad x^*_{(0,5)} = \tfrac{1}{2} , \qquad x^*_{(0,6)} = \tfrac{1}{2} ,$$

$$x^*_{(4,1)} = \tfrac{1}{2} , \qquad x^*_{(4,2)} = \tfrac{1}{2} , \qquad x^*_{(5,2)} = \tfrac{1}{2} ,$$

$$x^*_{(5,3)} = \tfrac{1}{2} , \qquad x^*_{(6,1)} = \tfrac{1}{2} , \qquad x^*_{(6,3)} = \tfrac{1}{2} ,$$

$$x^*_{(i,j)} = 0 \quad \text{otherwise} ,$$

is feasible to $LP(D)$ associated with $\hat{G}'$ and has a lower objective function value of 4.5 .

Nevertheless, $LP(D)$ is still useful for the analysis of the FCSF game. Indeed,

Proposition 2.3.1 If the incidence vector of a minimum cost directed Steiner tree in the directed augmented network $\hat{G}'$, rooted away from o and whose vertex set contains D, is an optimal solution to $LP(D)$, then the core of the associated FCSF game is not empty.

Proof: Let $\hat{c}(Q)$ denote the optimal value of $LP(Q)$, $Q \subseteq D$, and consider the cooperative game $(D;\hat{c})$ with $\hat{c}(\emptyset) = 0$. Then, (2.1) provides us with a generalized linear production game formulation of $(D;\hat{c})$, see D. Granot (1986). Further, one can easily verify that the sufficient conditions given therein for the nonemptiness of the core of a generalized linear production game are always satisfied for $(D;\hat{c})$, implying that the core of $(D;\hat{c})$ is not empty. On the other hand, for each $Q \subseteq D$, let $c(Q)$ denote the optimal value of the integer programming problem, $IP(Q)$, derived from $LP(Q)$ by restricting the nonnegative decision variables therein to $0-1$ values. Observe that $c(Q)$ coincides with the characteristic function value assigned to Q, $Q \subseteq D$, in the FCSF game. Clearly, $\hat{c}(Q) \leq c(Q)$, $Q \subseteq D$. Moreover, whenever a minimum cost Steiner Tree in $\hat{G}'$, directed away from vertex o and whose vertex set contains $D \cup \{o\}$, solves $LP(D)$, we have that $c(D) = \hat{c}(D)$. Thus, in this case the core of $(D;\hat{c})$ is a subset of the core of $(D;c)$ and the nonemptiness of the core of the FCSF game then follows since the core of $(D;\hat{c})$ is never empty. □

Moreover, when the underlying network has a series-parallel structure, we can test in polynomial time whether the sufficient condition for the nonemptiness of the core of the FCSF game, given by Proposition 2.3.1, is satisfied. Explicitly, we first find a minimum cost Steiner tree

T , using the algorithm developed in Section 2.2 . Then, we can apply the T-guided algorithm of Prodon et al. (1985) to check if the incidence vector of T is an optimal solution to $LP(D)$. This will be true if and only if the T-guided algorithm terminates with a dual feasible solution whose corresponding objective function value is equal to the weight of T.

Chapter III

ON SOME OPTIMIZATION PROBLEMS ON k-TREES AND PARTIAL k-TREES

The recursive definition of k-trees implies that a k-tree has at least two vertices, called leaves, of degree k , with completely connected neighbors. The neighborhood of a k-leaf v is denoted by $Adj(v)$, and we let K_v denote the k-clique induced by it. Clearly the graph obtained by removing a k-leaf v and its incident edges from a k-tree is a k-tree itself. This defines a **reduction process** for k-trees, and such a reduction process is complete when it ends up with some k-clique, R , the **root** of the reduction process. A reduction sequence can be thought of as giving an **orientation** to a k-tree (Arnborg and Proskurowski (1986)) where vertices are made descendants of k-cliques. Namely, if K is a k-clique, then v is a **descendant** of K in a given reduction sequence if and only if when v was removed, each vertex of $Adj(v)$ was either a member of K or a descendant of it. k-trees are also known to have the following **separation** property (Rosė (1974)): every minimal separator of a k-tree consists of k completely connected vertices.

In this chapter we will use the notion of the reduction process and the separation property to construct, for a fixed k, efficient algorithms for several optimization problems on k-trees and partial k-trees. In particular, in Section 3.1 we follow the reduction process to construct a linear time algorithm that finds shortest simple paths from a given vertex to all other vertices in a weighted k-tree having nonnegative edge weights, and in Section 3.2 we use a similar approach to compute the diameter (longest path) and the vertex center of a k-tree with equal edge lengths in linear time. Finally in Section 3.3 we construct an $O(n^{k+2})$ time algorithm for solving the Simple Plant Location (SPL) problem in weighted partial k-tree graphs, which generalizes Hassin and Tamir's (1986) $O(n^4)$ time algorithm for solving the SPL problem on series parallel graphs (i.e. partial two trees).

3.1 Single Source Shortest Paths Problem in k-trees

Let $G = (V,E)$ be a weighted k-tree with $|V| = n$, such that each (directed) edge e in E has a "length" (weight) $w(e) \geq 0$. For each pair of vertices u and v in V define $d(u,v)$ to be the length of a shortest simple directed path from u to v. Given a vertex s in V, we wish to find the distances from s to all other vertices v in V, as well as distances from all $v \in V$ to s. This problem is recognized as the **Single Source Shortest Path Problem** (SSSP) problem, and we demonstrate below that it is solvable in linear time for k-trees with nonnegative edge weights. For simplicity of presentation, we will only be concerned with the shortest paths from s to all other vertices $v \in V$.

Let v be a k-leaf of $G = (V,E)$ and denote by $G' = (V',E')$ the weighted graph obtained from G by removing v and its incident edges from G and whose edge weights, $w'(e)$, $e \in E'$, have been modified as follows :

$$w'((i,j)) = \begin{cases} min\ \{\ w((i,j))\ ,\ w((i,v)) + w((v,j))\ \} & \text{for } i,j \in Adj(v) \\ w((i,j)) \text{ otherwise} \end{cases} \tag{3.1}$$

Further, let $d'(i,j)$ denote the length of a shortest simple directed path from node i to j in G'.

Lemma 3.1.1 $d'(i,j) = d(i,j)$ for all $i, j \in V'$.

Proof: Let $i, j \in V'$ and assume, on the contrary, that $d'(i,j) = w'(p'_{ij}) < d(i,j) = w(p_{ij})$, where p'_{ij} and p_{ij} are shortest directed simple paths from i to j in G' and G, respectively, and $w'(p'_{ij})$ and $w(p_{ij})$ are their corresponding lengths. Let $A = \{ (u,w) \in p'_{ij} : w'((u,w)) < w((u,w)) \}$. Clearly $|A| \geq 1$. If $|A| = 1$ then $A = \{(u,w)\}$ for some $u,w \in Adj(v)$. Then, by replacing (u,w) in p'_{ij} by (u,v) and (v,w) we obtain the simple directed path p''_{ij} in G for which $w'(p'_{ij}) = w(p''_{ij}) < w(p_{ij})$, contradicting the optimality of p_{ij} in G. So, we must have that $|A| \geq 2$. Let (u_1, w_1) and (u_2, w_2) be in A, such that w_1 and u_2 are not necessarily distinct. Then, upon replacing (u_1,w_1) and (u_2, w_2) by (u_1, v), (v, w_1) and (u_2, v), (v, w_2) in p'_{ij}, we obtain a directed path (not simple) p''_{ij} in G such that $w(p''_{ij}) < w(p_{ij})$. Clearly, p''_{ij} contains a cycle, C, in G which contains node v. Since we assumed that G has nonnegative edge weights, upon the removal of C from p''_{ij} we will obtain a simple path $\overline{p}_{ij}$ in G satisfying $w(\overline{p}_{ij}) < w(p_{ij})$. This contradicts the optimality of p_{ij}, and the proof is complete. □

Lemma 3.1.1 can be used to construct a linear time algorithm to find shortest paths from some designated source node s, in a k-tree $G = (V, E)$, to all other vertices in G.

An algorithm for solving the SSSP problem in k-trees

The algorithm has two phases.

Phase 1 We successively eliminate k-leaves and their incident edges from G while modifying the weights of the remaining edges using (3.1). Since a k-tree has at least two leaves, we can always avoid the elimination of the source node s. The elimination of the k-leaves is continued until we obtain a subgraph $G^k = (V^k, E^k)$ of G, whose number of vertices is just a function of k. For simplicity, we will assume that $|V^k| = k$, which can be achieved by letting G^k be a k-complete graph containing node s.

Next, we solve the SSSP problem in G^k, whose edge weights were derived by successively modifying the edge weights using (3.1). This can be done in $O(k^2)$ elementary operations, which is constant for a fixed k. Let $\hat{d}(s,u)$, $u \in V^k$, be the weight of a shortest path from s to u in G^k. By Lemma 3.1, $\hat{d}(s,u) = d(s,u)$ for all $u \in V^k$, where $d(s,u)$ is the weight of a shortest path from s to u in G.

Phase 2 Let v be the last k-leaf to be eliminated at Phase 1, and consider the k-tree graph $\overline{G}^k$ obtained by adding v and all its incident edges (v,u), $u \in Adj(v)$, to G^k. Let $\overline{G}^k$ have the same edge weights, $\overline{w}(i,j)$, as it had in Phase 1. Now, if $\overline{d}(s,v)$ is the weight of a shortest path from s to v in $\overline{G}^k$ then $\overline{d}(s,v) = min\ \{ \overline{d}(s,u) + \overline{w}(u,v),\ u \in Adj(v) \}$. Moreover, by Lemma 3.1.1, $\overline{d}(s,v) = d(s,v)$ and $\overline{d}(s,u) = d(s,u)$ for $u \in Adj(v)$, which implies

$$d(s,v) = min\ \{d(s,u) + \overline{w}(u,v),\ u \in Adj(v)\ \}. \tag{3.2}$$

Thus, by successively adding k-leaves in the reverse order to which they were eliminated in Phase 1, and by calculating the weights of shortest paths using (3.2) we solve the SSSP problem in G. □

The linear complexity of the above algorithm follows from the fact that in Phase 1 we have at most $n - k$ steps and that each recursive equation used either in Phase 1, to modify the weights, or in Phase 2, to calculate weights of shortest paths, can be computed in constant time.

3.2 Diameter of a k-Tree

Let $G = (V, E)$ be an undirected graph. The **eccentricity** of u, $u \in V$, is $ecc\,(u) = max\,\{\, d(u,v) : v \in V \,\}$. The **diameter** of G is $D = max\,\{\, ecc(u) : u \in V \,\}$. In this section we will compute the diameter of a k-tree $G = (V, E)$ with equal edge lengths in linear time.

The algorithm consists of two phases. In the first phase we will follow a reduction process of the k-tree. Initially, we set $ecc^{K}(u) = 1$ for all $u \in V(K)$ and $K \in \mathcal{K}$, where $\mathcal{K}$ is the family of all k-cliques in G ($|\mathcal{K}| = (n - k)\, k + 1$, $n \geq k$), $V(K)$ is the vertex set of the k-clique K and $ecc^{K}(u)$ is the eccentricity of $u \in V(K)$ in the subgraph of G induced by $V(K)$ and all the descendant vertices of K in the reduction process followed by the algorithm.

Let K_v, where $V(K_v) = Adj(v) = \{\, u_1, \cdots, u_k \,\}$, denote the k-clique induced by the neighbors of a k-leaf v, and for each $u_j \in Adj(v)$ let $\overline{K}_j$ designate the side k-clique induced by $\{v\} \cup Adj(v) \setminus \{u_j\}$. Before a k-leaf node is being eliminated by the reduction process we store, for later purposes, the current eccentricities, $ecc^{\bar{v}}(u) = ecc^{K_v}(u)$, of all vertices $u \in Adj(v)$. After the elimination of v, we compute the eccentricities of vertices u_j, $u_j \in Adj\,(v)$, restricted to the subgraph of the k-tree G induced by $Adj\,(v) \cup \{v\}$ and the descendant vertices of all side k-cliques $\overline{K}_j$ as follows :

$$ecc^{v}(u_j) = max\,\{\, \max_{i \neq j}\; ecc^{\overline{K}_i}(u_j)\,,\; \min_{x \in V(\overline{K}_j)}\; ecc^{\overline{K}_j}(x) + 1 \,\} \tag{3.3}$$

and we update $ecc^{K_v}(u)$, for all $u \in V(K_v)$,

$$ecc^{K_v}(u) = max\,\{\, ecc^{v}(u)\,,\, ecc^{K_v}(u) \,\} \tag{3.4}$$

Clearly, in the last step of the first phase we have $ecc^{R}(u) = ecc\ (u)$, $u \in V(R)$, where R is the k-clique which is the root of the reduction process.

Next, for $u \in V(K)$, $K \in \mathcal{K}$, we denote by $ecc_{D}^{K}(u)$ the eccentricity of u in the subgraph of the k-tree G induced by $V(K)$ and the complement of the set of the descendant vertices of K in the reduction process. In the second phase we initially set $D = max\ \{ecc(u) : u \in V(R)\ \}$ and $ecc_{D}^{K}(u) = 0$, for all $u \in V(K)$ and $K \in \mathcal{K}$, and follow the reduction process in the reverse order. At each step we add a vertex v , compute the exact eccentricity of v in G , $ecc(v)$, and update D as follows :

$$ecc(v) = max\ \{ \max_{j=1,\cdots,k} ecc^{\overline{K}_j}(v)\ ,\ \min_{u \in Adj(v)} ecc^{\bar{v}}(u) + 1\ ,\ \min_{u \in Adj(v)} ecc_{D}^{K_v}(u) + 1\ \} \qquad (3.5)$$

and

$$D = max\ \{\ D\ ,\ ecc(v)\ \}. \qquad (3.6)$$

We further update other eccentricities in the following manner:

for side k-cliques $\overline{K}_j$, $j = 1 ,\cdots, k$ and $u_i \in V(\overline{K}_j) \cap Adj(v)$:

$$ecc_{D}^{\overline{K}_j}(u_i) = max\ \{\ ecc_{D}^{K_v}(u_i)\ ,\ ecc^{v}(u_i)\ ,\ \min_{x \in V(\overline{K}_i)} ecc^{\overline{K}_i}(x) + 1\ \}\ , \qquad (3.7)$$

and for $j = 1\ , \ldots, k$,

$$ecc_{D}^{\overline{K}_j}(v) = max\ \{\ \min_{u \in Adj(v)} ecc^{\bar{v}}(u) + 1\ ,\ \min_{u \in Adj(v)} ecc_{D}^{K_v}(u) + 1\ ,\ \max_{i \neq j} ecc^{\overline{K}_i}(v)\ \} \qquad (3.8)$$

and finally

$$ecc_D^{K_v}(u) = max \{ \; ecc_D^{K_v}(u) \; , \; ecc^v(u) \; , \; u \in Adj(v) \; \} \; . \tag{3.9}$$

Clearly, at the end of the second phase we have computed the diameter, D, of the k-tree G. Since the reduction process has $(n - k)$ steps and each step requires a constant time, the total time required to compute D is $O(n)$.

Remark Since we compute in the above algorithm the eccentricities of all vertices in V, it follows that we can also find in linear time a vertex with the smallest eccentricity. Such a vertex is called a vertex center of G.

3.3 Simple Plant Location (SPL) Problem on Partial k-Trees

Let $G = (V, E)$ be an undirected weighted graph, with $|V| = n$, in which each edge e, $e \in E$, is assumed to have a length $w(e) \geq 0$. Further, each vertex v in V is associated with a fixed cost, c_v, to open a facility at v, and a transportation cost function $g_v(d(v\,,\,.\,))$ which is a nondecreasing function of the distance, $d(v\,,\,.\,)$, from node v. Each vertex is viewed as both a potential facility site as well as a customer. The **Simple Plant Location** (SPL) problem problem, defined with respect to G, is concerned with the location of facilities at vertices in G so as to minimize the sum of the set up costs and the transportation costs.

Formally, the SPL problem seeks a subset of vertices F, $F \subseteq V$, that minimizes $\sum_{v \in F} c_v + \sum_{v \in V} g_v(d(v, F))$, where $d(v, F) = min_{f \in F}\ d(v, f)$. For general graphs, the SPL problem is known to be NP-hard. However, Kolen and Tamir (1986) developed an $O(n^2)$ time algorithm to solve the SPL problem on tree networks and Hassin and Tamir (1986) constructed an $O(n^4)$ time algorithm to solve this problem on series-parallel graphs. The purpose of this section is to develop an algorithm that solves the SPL problem on a partial k-tree graph in $O(n^{k+2})$ elementary operations. Thus, for a fixed k our algorithm is polynomial in the number of vertices, n, of G.

Observe that the embedding of a partial k-tree into a k-tree, for a fixed k, can be done in $O(n^{k+2})$ time, as shown by Arnborg et al. (1987). Therefore, since the addition of edges with infinite length would not affect the solution to the SPL problem, we will assume in the sequel that the partial k-tree graph has been completed into a k-tree. Our algorithm follows a reduction process of a k-tree G. Accordingly, let K be some k-clique with a vertex set $V(K) = \{ u_1, u_2, \cdots, u_k \}$. We will denote by $h(K, f_1, f_2, \cdots, f_k)$ the optimal value of the SPL problem restricted to the subgraph $G(D(K) \cup V(K))$ of G, induced by $V(K)$ and all the descendants of the k-clique K in the reduction process followed by the algorithm, with the stipulation that u_i is served by f_i for $i =$

$1, 2, \cdots, k$, and where f_i can be any node in G. Thus, the optimal value of the SPL problem on G is given by $min\{ h(R, f_1, f_2, \cdots, f_k) : f_i \in V, i = 1, 2, \cdots, k \}$, where R is the root of the reduction process followed by our algorithm.

An algorithm for solving the SPL problem on a k-tree

Suppose that v is the k-leaf which is currently being eliminated in the reduction process followed by the algorithm. Let $Adj(v) = \{ u_1, \cdots, u_k \}$ and denote by $\overline{K}_j$ the 'side' k-clique whose vertex set is $V(\overline{K}_j) \equiv Adj(v) \cup \{v\} \setminus \{u_j\}$, $j = 1, \cdots, k$. Let K_v be the k-clique whose node set is $Adj(v)$.

For any side k-clique $\overline{K}_j$, which was not previously considered, the algorithm first finds the optimal values $h(\overline{K}_j, f_1, \cdots, f_k)$ for all $f_i \in V$, $i = 1, \cdots, k$. In fact, if we denote by $\{ j_1, \cdots, j_k \}$ the node set of $V(\overline{K}_j)$, then it is easy to verify that for all $(f_1, \cdots, f_k)$, $f_i \in V$, $i = 1, \cdots, k$, and side k-clique, $\overline{K}_j$, not previously considered

$$h(\overline{K}_j, f_1, \cdots, f_k) = \sum_{i=1}^{k} c_{f_i} + \sum_{i=1}^{k} g_{j_i}(d(j_i, f_i)) - \sum_{i=1}^{k'} c_{f'_i}(t(f'_i) - 1), \tag{3.10}$$

where f_i is the facility serving j_i, $\{ f'_1, \cdots, f'_{k'} \}$ is the set of all distinct f_i's appearing in $(f_1, f_2, \cdots, f_k)$, and $t(f'_i)$ denotes the number of instances that f'_i appears in $(f_1, f_2, \cdots, f_k)$.

Next, we solve the SPL problem associated with the k-clique K_v. As it will be shown, the optimal value of the SPL problem, $h(K_v, f_1, \cdots, f_k)$, can be expressed in terms of the optimal values, $h(\overline{K}_j, f_1, \cdots, f_k)$, associated with the side k-cliques $\overline{K}_j$, $j = 1, \cdots, k$. First, we need to introduce some notation. We will say that $(f_1, \cdots, f_r)$ is a **facility assigment** for vertices

$\{ l_1 ,\cdots, l_r \}$ if node l_j receives the service from facility f_j , $j = 1 ,\cdots, r$. Let $F = (f_1 ,\cdots, f_k)$ be a facility assignment for $Adj(v) = \{ u_1 ,\cdots, u_k \}$, and let $\hat{V}_j = V(\overline{K}_j) \cap Adj(v)$, $j = 1 ,\cdots, k$. Then, the facility assignment F to $Adj(v)$ induces a facility assignment F_j to $\hat{V}_j$, specifying for each node in $V(\overline{K}_j)$, except node v , the facility providing service to it.

Then, one can verify that for any facility assignment $F = (f_1 ,\cdots, f_k)$ to $Adj(v) = \{ u_1 ,\cdots, u_k \}$,

$$
\begin{aligned}
h(K_v , f_1 ,\ldots, f_k) = min \Big\{ & h(K_v , f_1 ,\ldots, f_k) + \sum_{j=1}^{k} h(\overline{K}_j , (F_j , s)) - \\
& (k-1)\, g_v (d(v,s)) - c_s (\delta_1 (k-1) + \delta_2) - \\
& \delta_3 [\sum_{j=1}^{k} (k-2)\, g_{u_i} (d(u_i ,f_i)) + \sum_{i=1}^{k'} c_{f_i'} (\delta_{4,i} (k-1) \\
& + (1 - \delta_{4,i})(k-2))] - (1 - \delta_3) [\sum_{i=1}^{k} (k-1) \\
& g_{u_i} (d(u_i ,f_i)) + \sum_{i=1}^{k'} c_{f_i'} (\delta_{4,i}\, k + (1-\delta_{4,i}) \\
& (k-1))] : s \in \bigcup_{j=1}^{k} D(\overline{K}_j) \cup \{v\} \cup \{ f_1 ,\cdots, f_k \} \Big\},
\end{aligned}
\tag{3.11}
$$

where $\delta_1 = 1$ if $s \notin \{ f_1' ,\cdots, f_{k'}' \}$ and zero otherwise and $\delta_2 = 1$ if $s = f_i' \in \{ f_1' ,\cdots, f_{k'}' \}$ and $t(f_i') = 1$ and zero otherwise ; $\delta_3 = 1$ if $h(K_v , f_1 ,\cdots, f_k) = 0$ (i.e . K_v was not previously encountered by the algorithm) and zero otherwise , $\delta_{4,i} = 1$ if $t(f_i') > 1$ and zero otherwise, $i \in \{ 1 ,\cdots, k' \}$. Further, (F_j , s) is the facility assignment to $V(\overline{K}_j)$ which consists of the facility assignment F_j to $\hat{V}_j$, induced by the facility assignment F to $Adj(v)$, and v is supplied from s. The set $\{ f'_1, \cdots, f'_{k'} \}$ and $t(f'_i)$, $i = 1, \cdots, k'$ have the same meaning as in (3.10), and $D(K)$ denotes the set of descendant vertices of the k-clique K.

The algorithm will terminate after $n - k$ steps with an optimal value to the SPL problem, $h(R)$, given by $h(R) = min \{ h(R, f_1, \cdots, f_k) : f_i \in V, i = 1, \cdots, k \}$, where R is the root of the reduction process followed by the algorithm.

Proposition 3.3.1 The above algorithm solves the SPL problem on a partial k-tree graph in $O(n^{k+2})$ elementary operations.

Proof: The embedding of a partial k-tree into a k-tree requires $O(n^{k+2})$ time. The computation of distances between all pairs of vertices requires $O(n^2)$ time, using the algorithm developed in Section 3.1. We assume that the computation of $g_v(d(v, x))$, for a given x, requires constant time. Further, in each step of the algorithm we need, in the worst case, $O(kn^k)$ time to compute $h(\overline{K}_j, \cdots)$, using (3.10), for all side k-cliques that were not previously considered by the algorithm. It takes $O(n^{k+1})$ time to compute $h(K_v, \cdots)$ using (3.11). Hence, the total time to solve the SPL problem on a partial k-tree is bounded by $O(n^{k+2}) + O(n^2) + (n - k)(O(kn^k) + O(n^{k+1})) = O(n^{k+2})$. □

Chapter IV

k-CABLES AND k-PATHS IN k-TREES

In this chapter we study the notion of k-path and k-cable in a k-tree. In Section 4.1 we present a new characterization of a k-path in k-trees and develop an $O(n^2)$ algorithm for solving an R k-cable decomposition problem of a k-tree. In Section 4.2 we construct a linear time algorithm for finding the **k-cable diameter** (i.e. length of a longest k-cable) in a k-tree having equal edge weights.

4.1 An R k-cable Distance Decomposition Problem

Let $G = (V, E)$ be a k-tree and R a positive integer, $R \geq 2k - 1$. We are seeking in this section a decomposition of V to a minimum number of components, V^1, $V^2, \cdots,$ V^l, such that for any pair of vertices i and j, $i, j \in V^t$ and $t = 1, \cdots, l$, the cable distance, to be defined below, between i and j does not exceed R. We will refer to this problem as the R k-cable distance decomposition problem. We develop in this section an $O(n^2)$ time algorithm for finding such a decomposition in the special case when G is a k-tree in which all the edge weights are equal to one.

First, we need to introduce some new definitions and notation. The **k-cable,** $C(u,v)$, denotes a collection of k vertex-disjoint paths between vertices u and v in a k-connected graph. The length[1] of the k-cable $C(u,v)$, $W(C(u,v))$, is the sum of the distances of the k paths in $C(u,v)$. The **k-cable distance,** $d_c(u,v)$, between vertices u and v is the length of a shortest k-cable between these two vertices. That is, $d_c(u,v) = min\,\{\, W(C(u,v)) : C(u,v) \in \mathcal{C}(u,v) \,\}$, where $\mathcal{C}(u,v)$ is the collection of all k-cables between u and v.

Apparently, the notion of a k-path, introduced and studied by Proskurowski (1984), plays a major role in the analysis of our decomposition problem.

Definition 4.1.1 (Proskurowski (1984)). A k-path is an alternating sequence, KP, of distinct k and $k + 1$ complete subgraphs $KP = < Q_1 , T_2 , Q_2 , T_3 , \cdots , T_n , Q_n >$, starting and ending with a k-clique and such that T_i contains exactly two of the distinct k-complete subgraphs Q_{i-1} and Q_i.

It was shown by Proskurowski (1984) that in a k-tree G, the vertices of the minimal subgraphs separating two nonadjacent vertices induce a k-path.

Proposition 4.1.1 The k-cable distance between two nonadjacent vertices u and v in a k-tree G

[1] Proskurowski (1984) defined the length of a k-cable, $C(u,v)$, to be the length of a longest path in $C(u,v)$. He also suggested therein other plausible definitions for the length of a k-cable, one of which is the definition of the length of a k-cable used in this thesis.

with edge lengths equal to one is equal to the length, $W(C(u,v))$, of the k-cable $C(u,v)$ induced by the vertices of the unique minimal k-path separating u and v . Moreover, if u and v are not adjacent then $d_c(u,v) = 2k + t$ where t is the number of (k+1)-cliques appearing in the unique k-path between u and v in G . (Observe that the number of (k+1)-cliques in $C(u,v)$ may be zero.)

Proof: Let $C(u,v)$ be the k-cable induced by the k-path $KP(u,v)$, $KP(u,v) = < Q_1 , T_2 , Q_2 , \cdots , T_n , Q_n >$, between u and v , and let $C'(u,v)$ be any other k-cable between u and v . By Theorem 3.2 in Proskurowski (1984), each path p' of C' contains all the vertices of some path p in $C(u,v)$, implying the minimality of $W(C(u,v))$. The construction of the k-path implies that each one of the $(k+1)$-cliques T_i in $KP(u,v)$ contains exactly one edge that belongs to the k-cable $C(u,v)$. The remaining edges in $C(u,v)$ are (u,i) , $i \in Q_1$, and (j,v) , $j \in Q_n$. The proof follows since Q_1 and Q_n are k-cliques. $\square$

Next, we will present an alternative characterization of a k-path which will be used in the sequel. First, we need to introduce some additional notation. Let $G = (V, E)$ be a k-tree and let $\mathcal{K}$ (resp. , $\mathcal{K}'$) denote the collection of all k-cliques (resp., (k+1)-cliques) in G . For a vertex $\overline{v} \in V$, $Q \in \mathcal{K}$, $Q' \in \mathcal{K}'$ we let $\delta_Q(\overline{v}) = max_{v \in V(Q)} d_c(\overline{v}, v)$ and $\delta'_{Q'}(\overline{v}) = max_{v \in V(Q')} d_c(\overline{v}, v)$, where $V(A)$ denotes the vertex set of A . For each pair v_i , $v_j \in V$ we define a partial order on $\mathcal{K}$, $(\mathcal{K} , \leq_{v_i v_j})$ as follows : for Q_1 and Q_2 in $\mathcal{K}$, $Q_1 \leq_{v_i v_j} Q_2$ if and only if $\delta_{Q_1}(v_i) \leq \delta_{Q_2}(v_i)$ and $\delta_{Q_1}(v_j) \leq \delta_{Q_2}(v_j)$; further, $Q_1 <_{v_i v_j} Q_2$ if $Q_1 \leq_{v_i v_j} Q_2$ and at least one of the two inequalities is satisfied as a strict inequality. Similarly, using $\delta'_Q(\overline{v})$, we define the partial orders $(\mathcal{K}' , \leq_{v_i v_j})$ for all pairs v_i , $v_j \in V$.

Definition 4.1.2 The extended k-path between vertices v_i and v_j , $\hat{KP}(v_i, v_j) =$

$< T_1 , Q_1 , T_2 , Q_2 , \cdots, T_n , Q_n , T_{n+1} >$, is the k-path $KP(v_i , v_j)$ extended with the two (k+1)-cliques T_1 and T_{n+1} , where $V(T_1) = \{v_i\} \cup V(Q_1)$ and $V(T_{n+1}) = \{v_j\} \cup V(Q_n)$.

Proposition 4.1.2 The extended k-path $\hat{KP}(v_i , v_j) = < T_1 , Q_1 , \cdots, Q_n , T_{n+1} >$ between vertices v_i and v_j in the k-tree graph $G = (V, E)$ consists of all minimal k and (k+1)-cliques of G with respect to the partial orders $(\mathcal{K} , \leq_{v_i v_j})$ and $(\mathcal{K}' , \leq_{v_i v_j})$, respectively.

Proof We first show that any (k+1)-clique that is not contained in $\hat{KP}(v_i , v_j)$ is not minimal with respect to $(\mathcal{K}' , \leq_{v_i v_j})$. Let M be a (k+1)-clique which is not in $\hat{KP}(v_i , v_j)$, and let x be a vertex in M such that $d_c(v_i , x) = max_{v \in V(M)} d_c(v_i,v)$. Observe that x is not a vertex of any (k+1)-clique in $\hat{KP}(v_i , v_j)$. Now, if $\hat{KP}(v_i , v_j)$ and $\hat{KP}(v_i , x)$ have no (k+1)-cliques in common then it is easy to see that $T_1 <_{v_i v_j} M$. Otherwise, let T_p be the (k+1)-clique in $\hat{KP}(v_i , v_j)$ with the highest index such that T_p is in $\hat{KP}(v_i , x)$. Then, $d_c(v_j , x) \geq max_{v \in V(T_p)} d_c(v_j , v)$, implying that $T_p <_{v_i v_j} M$.

Next, we will show that every (k+1)-clique T in $\hat{KP}(v_i , v_j)$ is minimal. Assume, on the contrary, that there exists a (k+1)-clique M such that $M <_{v_i v_j} T$. By definition, it follows that M cannot be in $\hat{KP}(v_i , v_j)$. Moreover, by the above there exists a (k+1)-clique $T_p \in \hat{KP}(v_i , v_j)$ such that $T_p <_{v_i v_j} M$. Since $(\mathcal{K}' , \leq_{v_i v_j})$ is transitive, that would imply $T_p <_{v_i v_j} T$, which is impossible.

Next, we will show that every k-clique which is not in $\hat{KP}(v_i , v_j)$ is not minimal. So, let M be a k-clique not contained in $\hat{KP}(v_i, v_j)$, and let vertices x_1 and x_2 in M be such that $d_c(v_i , x_1) = max_{v \in V(M)} d_c(v_i , v)$ and $d_c(v_j , x_2) = max_{v \in V(M)} d_c(v_j , x)$. Now, if $\hat{KP}(v_i , x_1)$ and $\hat{KP}(v_i , v_j)$ have no k-cliques in common then $Q_1 <_{v_i v_j} M$. Otherwise, we need to distinguish between two subcases.

In subcase 1, x_1 is not a vertex of any k-clique in $\hat{KP}(v_i, v_j)$. Then, there exists a k-clique, $\overline{Q}_p$, not necessarily contained in $\hat{KP}(v_i, v_j)$ but which is a k-clique of some (k+1)-clique, T_p, of $\hat{KP}(v_i, v_j)$, such that $\overline{Q}_p$ is both a $v_i - x_1$ separator and $v_j - x_2$ separator. Then, $\overline{Q}_p <_{v_i v_j} M$.

In subcase 2, x_1 and, in fact, all vertices of M are vertices of some k-cliques in $\hat{KP}(v_i, v_j)$. Then, M is a k-clique of some (k+1)-clique, T_p, in $\hat{KP}(v_i, v_j)$. Moreover, $d_c(v_i, x_1) = max_{v \varepsilon V(T_p)} d_c(v_i, v)$ and $d_c(v_j, x_2) = max_{v \varepsilon V(T_p)} d_c(v_j, v)$. Then, it is easy to see that $Q_p <_{v_i v_j} M$, where $Q_p = T_p \setminus \{x_1\}$.

Finally, we show that every Q in $\hat{KP}(v_i, v_j)$ is minimal with respect to $(\mathcal{K}, \leq_{v_i v_j})$. Assume that there exists a k-clique M such that $M <_{v_i v_j} Q$. Clearly, M is not in $\hat{KP}(v_i, v_j)$. By the above, there exists a k-clique Q_p in $\hat{KP}(v_i, v_j)$ such that $Q_p <_{v_i v_j} M$. Then, transitivity of $(\mathcal{K}, \leq_{v_i v_j})$ implies that $Q_p <_{v_i v_j} Q$, which is impossible. □

Let $G = (V, E)$ be a k-tree, q – a positive integer, and for any $v_i \in V$ let $V_i = \{ v \in V : d_c(v_i, v) \leq q \}$. Further, denote by $G(V_i)$ the subgraph of G induced by V_i.

Proposition 4.1.3 The subgraphs $G(V_i) = (V_i, E_i)$, $v_i \in V$, are sub-k-trees of G.

Proof Choose $\overline{v} \in V_i$ such that $d_c(v_i, \overline{v}) = max_{v \varepsilon V_i} d_c(v_i, v) = m$. Then, there exists a unique k-path $KP(v_i, \overline{v})$, $KP(v_i, \overline{v}) = < Q_o, T_1, Q_1, T_2, \cdots, T_l, Q_l >$, between v_i and $\overline{v}$ which determines a k-cable $C(v_i, \overline{v})$ of length m. Let Q_l be the k-clique adjacent to $\overline{v}$ in the k-path $KP(v_i, \overline{v})$. Clearly, $d_c(v_e, v_i) < m$ for all $v_e \in Q_l$ and thus the vertex set of Q_l is contained in V_i. It is easy to see that any other vertex $\hat{v}$ adjacent to $\overline{v}$, if exist, would have $d_c(v_i, \hat{v}) > m$. Thus, node $\overline{v}$ is a k-leaf in $G(V_i)$. By eliminating $\overline{v}$ and repeating the above procedure we construct a reduction process for $G(V_i)$, implying that it is a k-tree. $\square$

Recall that a reduction process in a k-tree G induces a descendancy relation of vertices with respect to k-cliques in G. Such a relation can be extended to a descendancy relation among k-cliques and (k+1)-cliques in G. In general, if K_i and K_j are two k-cliques (resp. (k+1)-cliques) in G, then K_i is said to be a descendant of K_j if every node that is either contained in K_i or is a descendant of K_i (resp., some k-clique in K_i) is also either contained in K_j or is a descendant of K_j (resp., descendant of some k-clique in K_j) with respect to the reduction process. This descendancy relation in G can be used to index all $k(n-k)+1$ k-cliques (resp. $n-k$ (k+1)-cliques) in G, such that the index of a k-clique (resp., (k+1)-clique) K_i is smaller than the index of a k-clique (resp., (k+1)-clique) K_j if K_i is a descendant of K_j .

Now, let $G(V_i)$ and $G(V_j)$ be two sub k-trees of G with a corresponding collection of k-cliques (resp., (k+1)-cliques) $\mathcal{K}_i$ and $\mathcal{K}_j$ (resp., $\mathcal{K}'_i$ and $\mathcal{K}'_j$), respectively. Further, let Kr_i and Kr_j be the k-cliques (resp., (k+1)-cliques) having the highest indices, r_i and r_j , in $\mathcal{K}_i$ and $\mathcal{K}_j$ (resp., $\mathcal{K}'_i$ and $\mathcal{K}'_j$), respectively, and assume that $r_i \leq r_j$. Then, it is easy to verify that $\mathcal{K}_i \cap \mathcal{K}_j \neq \emptyset$ (resp., $\mathcal{K}'_i \cap \mathcal{K}'_j \neq \emptyset$) if and only if $K_{r_i} \in \mathcal{K}_j$ (resp., $K'_{r_i} \in \mathcal{K}'_j$) . This result can be used to establish the Helly property of a set of collections of k-cliques (resp., (k+1)-cliques) of sub k-trees of a given k-tree, as follows :

Proposition 4.1.4 Let $G_1, \cdots, G_m$ be a collection of sub k-trees of G with an associated collections of k-cliques (resp., (k+1)-cliques) $\mathcal{K}_1, \cdots, \mathcal{K}_m$ (resp., $\mathcal{K}_1', \cdots, \mathcal{K}'_{m'}$) . Let $\mathcal{K} = \{ \mathcal{K}_1, \cdots, \mathcal{K}_m \}$ and $\mathcal{K}' = \{ \mathcal{K}'_1, \cdots, \mathcal{K}'_m\}$. Then, $\mathcal{K}$ and $\mathcal{K}'$ have the Helly property. That is, if $\mathcal{K}_i \cap \mathcal{K}_j \neq \emptyset$ (resp., $\mathcal{K}'_i \cap \mathcal{K}'_j \neq \emptyset$) for all i and j, then $\cap_{i=1}^{m} \mathcal{K}_i \neq \emptyset$ (*resp.* $\cap_{i=1}^{m} \mathcal{K}'_i$ $\neq \emptyset$) .

Proof: Let K_{r_i} (*resp.*, K'_{r_i}) be the k-clique (resp., (k+1)-clique) with the highest index in $\mathcal{K}_i$ (resp. , $\mathcal{K}'_i$) and assume, without loss of generality, that $r_1 \leq r_j$ for all j. Since $\mathcal{K}_1 \cap \mathcal{K}_j \neq \emptyset$ (resp., $\mathcal{K}'_1 \cap \mathcal{K}'_j \neq \emptyset$) for all j we have $K_{r_1} \in \mathcal{K}_j$ (*resp.* , $K'_{r_j} \in \mathcal{K}'_j$) for all j. □

Theorem 4.1.1 Let $G = (V, E)$ be a k-tree, $V = \{ v_1 , v_2 , \cdots, v_n \}$, $n \geq k+1$, and let R be an even integer, $R \geq 2k$. For each $v_i \in V$ let $G(V_i)$ be the sub k-tree of G induced by $V_i = \{ v \in V\colon\ d_c(v, v_i) \leq \frac{R}{2} + k - 1 \}$, with an associated collection, $\mathcal{K}_i$, of k-cliques. Then, $\cap_{i \in I} \mathcal{K}_i \neq \emptyset$, $I \subseteq \{ 1 , 2 , \cdots, n \}$, if and only if $d_c(v_i , v_j) \leq R$ for all pairs $i, j \in I$.

Proof: Suppose that $\cap_{i \in I} \mathcal{K}_i \neq \emptyset$. Then, it follows from Proposition 4.1.2 that for any pair $i, j \in I$ there exists a k-clique Q_l in the k-path $KP(v_i , v_j)$ which is in $\mathcal{K}_i \cap \mathcal{K}_j$. By definition of $G(V_i)$ we have that $max_{v \in V(Q_l)}\, d_c(v_i , v) \leq \frac{R}{2} + k - 1$ and $max_{v \in V(Q_l)} d_c(v_j , v) \leq \frac{R}{2} + k - 1.$

Now, the k-clique Q_l separates the cable $C(v_i , v_j)$, induced by $KP(v_i , v_j)$, into two components (sides) C_{v_i, Q_l} and C_{v_j, Q_l} , containing v_i and v_j , respectively. From Proposition 4.1.1 it follows that $W(C_{v_i, Q_l}) \leq \frac{R}{2}$ and $W(C_{v_j, Q_l}) \leq \frac{R}{2}$, implying that $d_c(v_i , v_j) = W(C_{v_i, Q_l}) + W(C_{v_j, Q_l}) \leq R$, where, for example, $W(C_{v_j, Q_l})$ denotes the total length of the paths in $C(v_i , v_j)$ which are in the component C_{v_j, Q_l} .

Next, assume that for any pair $i, j \in I$, $d_c(v_i , v_j) \leq R$. Then, let Q_l be a k-clique in $KP(v_i , v_j)$ satisfying $max_{v \in Q_l}\, d_c(v_i , v) = d(v_i , x_1) = \frac{R}{2} + k - 1$. If no such k-clique exist then all k-cliques in $KP(v_i ,v_j)$ are in $\mathcal{K}_i \cap \mathcal{K}_j$. If $KP(v_i ,v_j)$ does not contain a k-clique then v_i and v_j are adjacent and the k-clique containing v_i and v_j is in $\mathcal{K}_i \cap \mathcal{K}_j$. Otherwise, by Proposition 4.1.1, $W(C_{v_i, Q_l}) = \frac{R}{2}$, implying that $W(C_{v_j, Q_l}) \leq \frac{R}{2}$. Therefore, again by Proposition 4.1.1, $d_c(v_j, v) \leq \frac{R}{2} + k - 1$ for all $v \in Q_l$. Thus, $Q_l \in \mathcal{K}_i \cap \mathcal{K}_j$, and by the Helly property of the collections $\mathcal{K}_i$ we have that $\cap_{i \in I} \mathcal{K}_i \neq \emptyset$. □

Theorem 4.1.2 Let $G = (V, E)$ be a k-tree, $V = \{ v_1, v_2, \cdots, v_n \}$, $n \geq k + 1$, and let R be an odd positive integer, $R \geq 2k - 1$. For each $v_i \in V$ let $G(V_i)$ be the sub k-tree of G induced by $V_i = \{ v \in V: d_c(v, v_i) \leq \lfloor \frac{R}{2} \rfloor + k \}$, where $\lfloor a \rfloor$ is the largest integer smaller than a, and let $\mathcal{K}_i'$ be the collection of (k+1)-cliques in $G(V_i)$. Then, $\cap_{i \in I} \mathcal{K}_i' \neq \emptyset$, $I \subseteq \{ 1, \cdots, n \}$, if and only if $d_c(v_i, v_j) \leq R$ for all pairs $i, j \in I$.

Proof: Suppose that $\cap_{i \in I} \mathcal{K}_i' \neq \emptyset$. Then, it follows from Proposition 4.1.2 that for any pair $i, j \in I$ there exists a (k+1)-clique T_l in $KP(v_i, v_j)$ such that $T_l \in \mathcal{K}_i' \cap \mathcal{K}_j'$. Let Q_l and Q_{l+1} be the k-cliques in $KP(v_i, v_j)$ that induce T_l. Clearly, $d_c(v_i, v) \leq \lfloor \frac{R}{2} \rfloor + k - 1$ for all $v \in Q_l$ and thus, $W(C_{v_i, Q_l}) \leq \lfloor \frac{R}{2} \rfloor$. Similarly, $d_c(v_j, v) \leq \lfloor \frac{R}{2} \rfloor + k - 1$ for all $v \in Q_{l+1}$, which implies that $W(C_{v_j, Q_{l+1}}) \leq \lfloor \frac{R}{2} \rfloor$. Thus, $d_c(v_i, v_j) = W(C_{v_i, Q_l}) + W(C_{v_j, Q_{l+1}}) + 1 \leq \lfloor \frac{R}{2} \rfloor + \lfloor \frac{R}{2} \rfloor + 1 = R$.

Next, assume that for every pair $i, j \in I$, $d_c(v_i, v_j) \leq R$. Let T_l be a (k+1)-clique in $\hat{KP}(v_i, v_j)$ satisfying $d_c(v_i, x_1) = max_{v \in T_l} d_c(v_i, v) = \lfloor \frac{R}{2} \rfloor + k$. If no such (k+1)-clique exist then all (k+1)-cliques in $\hat{KP}(v_i, v_j)$ are in $\mathcal{K}_i' \cap \mathcal{K}_j'$. If $\hat{KP}(v_i, v_j)$ does not contain a (k+1)-clique then v_i and v_j are adjacent and the (k+1)-clique containing v_i and v_j is in $\mathcal{K}_i' \cap \mathcal{K}_j'$. Otherwise, $T_l \in \mathcal{K}_i'$. Suppose that T_l is induced by the k-cliques Q_l and Q_{l+1} in $KP(v_i, v_j)$. Then, $W(C_{v_i, Q_{l+1}}) = \lfloor \frac{R}{2} \rfloor + 1$. Therefore, $W(C_{v_j, Q_{l+1}}) = R - \lfloor \frac{R}{2} \rfloor - 1 = \lfloor \frac{R}{2} \rfloor + 1 - 1 = \lfloor \frac{R}{2} \rfloor$, which implies that $d_c(v_j, v) \leq \lfloor \frac{R}{2} \rfloor + k - 1$ for all $v \in Q_{l+1}$. Further, $d_c(v_j, v) \leq \lfloor \frac{R}{2} \rfloor + k$ for all $v \in T_l$, implying that $T_l \in \mathcal{K}_j'$. By the Helly property of the collections $\mathcal{K}_i'$ we conclude that $\cap_{i \in I} \mathcal{K}_i' \neq \emptyset$. $\square$

Observe that in Theorem 4.1.1 and 4.1.2 we restrict R to be at least $2k - 1$. It can be easily shown that if $R < 2k - 1$ then the only decomposition of V for our R k-cable distance decomposition problem coincides with the vertex set V. In general, we solve our R k-cable distance decomposition problem by using the properties of two clique-intersection graphs associated with G, which are constructed as follows. If R is even (resp., odd) then for each $v_i \in V$ let $G^k(V_i)$ (resp., $G^{k+1}(V_i)$ denote the sub k-tree of G induced by $V_i = \{ v \in V\colon d_c(v, v_i) \leq \frac{R}{2} + k - 1 \}$ *(resp.,* $V_i = \{ v \in V\colon d_c(v, v_i) \leq \lfloor \frac{R}{2} \rfloor + k \}$). Let $\mathcal{K}_i$ (resp., $\mathcal{K}'_i$) designate the collection of k-cliques (resp., (k+1)-cliques) in $G^k(V_i)$ *(resp.,* $G^{k+1}(V_i)$), and consider the intersection graph $\tilde{G}^k = (\tilde{V}, \tilde{E}^k)$ *(resp.,* $\tilde{G}^{k+1} = (\tilde{V}, \tilde{E}^{k+1})$, induced by the sub k-trees $G^k(V_i)$ *(resp.,* $G^{k+1}(V_i)$), $v_i \in V$, where $\tilde{V} = \{ \tilde{v}_i : v_i \in V \}$ and $(\tilde{v}_i, \tilde{v}_j) \in \tilde{E}^k$ (resp., $(\tilde{v}_i, \tilde{v}_j) \in E^{k+1}$) if $\mathcal{K}_i \cap \mathcal{K}_j \neq \emptyset$ (resp., $\mathcal{K}'_i \cap \mathcal{K}'_j \neq \emptyset$). Then, following the lead of, for example, Kolen and Tamir (forthcoming), one can use Proposition 4.1.4 to demonstrate that both $\tilde{G}^k$ and $\tilde{G}^{k+1}$ are chordal graphs.

By combining Theorem 4.1.1 , 4.1.2 and the fact that $\tilde{G}^k$ and $\tilde{G}^{k+1}$ are chordal graphs we can construct an $O(n^2)$ time algorithm for generating an R k-cable distance decomposition of the vertex set of a k-tree. Indeed,

Theorem 4.1.3 The R-cable distance decomposition problem can be found in a k-tree $G = (V, E)$ in $O(n^2)$ time .

Proof If R is even (resp., odd) we construct the clique-intersection graph $\tilde{G}^k$ (resp., $\tilde{G}^{k+1}$). By Theorems 4.1.1 and 4.1.2 , a minimum clique cover in $\tilde{G}^k$ or $\tilde{G}^{k+1}$ provides us with a solution to the decomposition problem. The overall complexity of an algorithm which generates the decomposition can be established as follows. From Theorem 4.1.1 (resp., 4.1.2) it follows that for an even (resp., odd) R, $(\tilde{v}_i, \tilde{v}_j) \in \tilde{G}^k$ *(resp.,* $(\tilde{v}_i, \tilde{v}_j) \in \tilde{G}^{k+1}$) if and only if $d_c(v_i, v_j) \leq R$. Thus, the clique-intersection graphs $\tilde{G}^k$ (resp., $\tilde{G}^{k+1}$) will be at hand after all cable distances,

$d_c(v_i, v_j)$, for all v_i , v_j in V have been found. The k-cable distances from a vertex v_i to all other vertices v, $v \in V$, can easily be computed in linear time. Indeed, we initially set $d_c(v_i,u) = 2k - 1$ for all $u \in Adj(v_i)$. Then, we simply successively build up the k-tree G so that whenever a vertex v, $v \in V$, is added to a current sub-k-tree, $d_c(v_i,v) = max_{u \varepsilon Adj(v)} d_c(v_i, u) + 1$. Thus, the construction of $\tilde{G}^k$ (*resp.*, $\tilde{G}^{k+1}$) takes $O(n^2)$ time. Since $\tilde{G}^k$ (resp., $\tilde{G}^{k+1}$) is a chordal graph, its minimum clique cover can be found in $O(n)$ time, see, for example, Gavril (1972) . Therefore, the complexity of the algorithm is dominated by the construction of $\tilde{G}^k$ (resp. $\tilde{G}^{k+1}$) which takes $O(n^2)$ time. □

4.2 k-Cable Diameter of a k-Tree

Let $G = (V,E)$ be a k-tree, $|V| = n$, in which all edge weights are equal to one, and denote by D_c , $D_c = d_c(v_2, v_3) \equiv max_{v_i,v_j \varepsilon V} \; d_c(v_i ,v_j)$, the **k-cable diameter** of G . In this section we develop a linear time alogrithm for finding a **longest k-cable**, $d_c(v_2,v_3)$, and the k-cable diameter, D_c , of a k-tree. Our algorithm is similar to Handler's (1973) algorithm for finding a longest path in a weighted tree.

Before introducing the algorithm, we need to recall some notation introduced in Section 4.1. For any non adjacent pair of vertices v_i , $v_j \in V$, the extended k-path, $\hat{KP}(v_i, v_j)$, induces a shortest k-cable $C(v_i, v_j)$. Further, any (k+1)-clique T_p (resp., k-clique Q_p) contained in $\hat{KP}(v_i, v_j)$ separates $C(v_i, v_j)$ into two sides (components), one of which could be possibly empty, $C_{v_i,T_p}(v_i, v_j)$ and $C_{v_j,T_p}(v_i, v_j)$ (*resp.*, $C_{v_i,Q_p}(v_i, v_j)$ and $C_{v_j,Q_p}(v_i, v_j)$) , not containing v_j and v_i , respectively .

Theorem 4.2.1 Let v_1 be any k-leaf of a k-tree $G = (V, E)$, and let v_2 be any vertex in G

such that $d_c(v_1, v_2) = max_{v\varepsilon V} d_c(v_1, v)$. Further, let $v_3 \in V$ be such that $d_c(v_2, v_3) = max_{v\varepsilon V} d_c(v_2, v)$. Then, $D_c = d_c(v_2, v_3)$.

Proof: Let v_4, v_5 be any pair of vertices in V. We will show that $d_c(v_4, v_5) \leq d_c(v_2, v_3)$. Let T_p and $T_{p'}$ be the (k+1)-cliques in $\hat{KP}(v_1, v_2)$ with the highest indices such that T_p is in $\hat{KP}(v_1, v_4)$ and $T_{p'}$ is in $\hat{KP}(v_1, v_5)$. We will assume, without loss of generality, that $p \leq p'$. Further, let $T_{p'}$ be the (k+1)-clique in $\hat{KP}(v_1, v_2)$ with the lowest index such that $T_{p''}$ is in $\hat{KP}(v_2, v_3)$. Note that T_p and $T_{p'}$ exist since v_1 is a k-leaf and $T_{p''}$ exists since v_2 is a k-leaf. Now, we need to consider two cases.

Case 1: $p'' > p'$. By definition of $T_{p'}$ and $T_{p''}$ and since $p'' > p'$, we have that $T_{p''}$ is contained in $\hat{KP}(v_2, v_4)$. Thus, by the construction of v_3 we have :

$$W(C_{v_3, T_{p''}}(v_2, v_3)) \geq W(C_{v_4, T_{p''}}(v_2, v_4)) \tag{4.1}$$

and by the construction of v_2 we have that :

$$W(C_{v_2, T_{p'}}(v_1, v_2)) \geq W(C_{v_5, T_{p'}}(v_1, v_5)). \tag{4.2}$$

Let $I = W(C_{v_2, T_{p'}}(v_1, v_2)) \cap W(C_{v_1, T_{p''}}(v_1, v_2))$. Then, by (4.1) and (4.2)

$$\begin{aligned} d_c(v_2, v_3) &= W(C_{v_3, T_{p''}}(v_2, v_3)) + W(C_{v_2, T_{p'}}(v_1, v_2)) - I \\ &\geq W(C_{v_5, T_{p'}}(v_1, v_5)) + W(C_{v_4, T_{p''}}(v_2, v_4)) - I. \end{aligned} \tag{4.3}$$

We further claim that

$$W(C_{v_5, T_{p'}}(v_1, v_5)) + W(C_{v_4, T_{p''}}(v_2, v_4)) - I \geq d_c(v_4, v_5). \tag{4.4}$$

Indeed, if $p < p'$ then $T_{p'}$ is in $\hat{KP}(v_2, v_4)$ and (4.4) is satisfied as equality. Otherwise, $p = p'$. Then, if $T_{p'}$ is not in $\hat{KP}(v_4, v_5)$ (4.4) clearly holds, while if $T_{p'}$ is in $\hat{KP}(v_4, v_5)$ then $T_{p'}$ is contained in at least one of the two k-paths $\hat{KP}(v_2, v_4)$ and $\hat{KP}(v_2, v_5)$. Without loss of generality, we can assume that $T_{p'}$ is in $\hat{KP}(v_2, v_4)$, which was shown above to imply (4.4). We conclude therefore by (4.3) and (4.4) that if $p'' > p'$ then $d_c(v_2, v_3) \geq d_c(v_4, v_5)$.

Case 2 $p'' \leq p'$. In this case we clearly have that $T_{p'}$ is in $\hat{KP}(v_2, v_3)$. Let $Q_{p'}$ be the k-clique in $T_{p'}$ such that $Q_{p'}$ is in $\hat{KP}(v_1, v_2)$ and having the highest index among all such k-cliques. By the construction of v_3 we have that :

$$W(C_{v_3, Q_{p'}}(v_2, v_3)) \geq W(C_{v_4, Q_{p'}}(v_2, v_4)) . \tag{4.5}$$

Further, by the construction of v_2, (4.2) holds. Now, by (4.2) and (4.5),

$$\begin{aligned} d_c(v_2, v_3) = \quad & W(C_{v_3, Q_{p'}}(v_2, v_3)) + W(C_{v_2, T_{p'}}(v_1, v_2))) \\ \geq \; & W(C_{v_4, Q_{p'}}(v_2, v_4)) + W(C_{v_5, T_{p'}}(v_1, v_2)). \end{aligned} \tag{4.6}$$

We claim that

$$W(C_{v_4, Q_{p'}}(v_2, v_4)) + W(C_{v_5, T_{p'}}(v_1, v_5)) \geq d_c(v_4, v_5) . \tag{4.7}$$

Indeed, if $p < p'$, $T_{p'}$ is in $\hat{KP}(v_2, v_4)$ and (4.7) is satisfied as equality. If $p = p'$ and $T_{p'}$ is not in $\hat{KP}(v_4, v_5)$, then (4.7) is clearly satisfied. Otherwise, $p = p'$ and $T_{p'}$ is in $\hat{KP}(v_4, v_5)$. In this event, similar to the proof in Case 1, we can assume without loss of generality that $T_{p'}$ is in

$\hat{KP}(v_2, v_4)$, which was shown above to imply (4.7). We conclude that (4.7) holds, which completes the proof. □

Now, as explained in the proof of Theorem 4.1.3 , $d_c(v_1, v_2)$ and $d_c(v_2, v_3)$ can be computed in linear time which implies, by Theorem 4.2.1, that $D_c = d_c(v_2, v_3)$ can be computed in linear time.

Chapter V

NC ALGORITHMS FOR RECOGNIZING PARTIAL 2-TREES AND 3-TREES

In this chapter we introduce the notion of a **k-separator** $(k \geq 2)$ of an n-vertex graph $G = (V, E)$. Formally, a k-separator is a set of vertices S, $|S| = k$, which induces a partition of $V \setminus S$ into sets V_1 and V_2 satisfying (i) $|V_i| \leq (k/k+1)n$, $i = 1, 2$, and (ii) no edge in E connects a vertex in V_1 with a vertex in V_2 . We prove the existence of a k-separator for partial k-trees and construct a linear time algorithm for generating such a separator in k-trees. This algorithm can be used to construct a balanced binary decomposition tree of a k-tree in $O(n \log n)$ time. We further derive other separation properties of partial k-trees which are used to construct a balanced decomposition of an embedding of a k-connected partial k-tree into a k-tree when $k = 2, 3$. Finally, we develop parallel algorithms for the recognition of partial k-trees when $k = 2, 3$. These algorithms require $O(\log^2 n)$ time and use, respectively, $O(n^3)$ and $O(n^4)$ processors. For $k = 2$ and 3 our algorithms improve considerably the processor bound of Bodlaender's (1988) general algorithm for the recognition of partial k-trees, which would require, respectively, $O(n^{10})$ and $O(n^{13})$ processors for these cases.

The chapter is organized as follows. In Section 5.1 we study k-separators of partial k-trees and describe an $O(n \log n)$ algorithm for constructing a balanced binary decomposition tree for k-trees. In Section 5.2 we obtain additional separation properties of partial k-trees, and show that they can be used to construct a binary balanced decomposition of an embedding of a k-connected partial k-tree when $k = 2$ and 3 . In Section 5.3 we develop NC algorithms for the recognition and embedding of a 2-connected (resp., 3-connected) partial 2-trees (resp., 3-trees), and in Section 5.4 we develop an algorithm for recognizing partial 3-trees which are not necessarily 3-connected.

5.1 k-SEPARATORS

Let $G = (V, E)$ be a partial k-tree $(k \geq 2)$ with $|V| = n$. A **k-separator** of G is a set of vertices S, of cardinality $|S| = k$, which induces a partition $\{V_1, V_2\}$ of $V \setminus S$ satisfying

$$|V_i| \leq \frac{k}{k+1}\, n, \quad i = 1,2, \tag{5.1}$$

and

$$\text{no edge in } E \text{ connects a vertex in } V_1 \text{ with a vertex in } V_2 \tag{5.2}$$

Observe that a k-separator extends the notion of a 2-separator defined for series parallel graphs (i.e. partial two trees), see, e.g., Hassin and Tamir (1986) .

Theorem 5.1.1 Every partial k-tree contains a k-separator. Moreover, for a k-tree a k-separator can be constructed in linear time.

Proof: Since any k-separator of a k-tree $G = (V, E)$ is also a k-separator of every partial graph of G, it is sufficient to prove the existence of a k-separator for k-trees. For that purpose, we construct below a linear time algorithm for finding a k-separator of a k-tree G. The algorithm follows a reduction process until reaching, for the first time, a k-leaf vertex v for which the corresponding set of descendant vertices, $D(K_v)$, of the k-clique K_v induced by $Adj(v)$, contains at least $\frac{2}{3}n$ distinct vertices.

Formally, the algorithm has two steps.

Step 1 We initially set $D(A) = \emptyset$ for all k-cliques A of G. In general, let v be a k-leaf which is currently being considered for elimination in the reduction process. Let $V(K_v) = Adj(v) = \{ u_1, \cdots, u_k \}$ and $V(\overline{K}_j) = Adj\,(v) \cup \{v\} \setminus \{u_j\}$, $j = 1, \cdots, k$, where $V(A)$ denotes the vertex set of a graph A. Then, update $D(K_v)$, the set of descendant nodes of K_v, and $|D(K_v)|$ as follows:

$$D(K_v) = \left(\bigcup_{j=1}^{k} D(\overline{K}_j)\right) \bigcup D(K_v) \bigcup \{v\} \tag{5.3}$$

$$|D(K_v)| = \sum_{j=1}^{k} |D(\overline{K}_j)| + |D(K_v)| + 1. \tag{5.4}$$

If $|D(K_v)| < \frac{2}{3}n$, we continue with the reduction process; otherwise, we go to Step 2 where the k-separator is produced. Observe that Step 1 will terminate whenever $n \geq 3k$. If $n < 3k$ every minimal separator of G is a k-separator.

Step 2 We need to distinguish between two cases.

Case 1 $\sum_{j=1}^{k} |D(\overline{K}_j)| + 1 \geq \frac{2}{3}n.$

Let $V_j = D(\overline{K}_j)$, $j = 1, \cdots, k$, and $V_{k+1} = V \setminus \left\{(\cup_{j=1}^{k} V_j) \cup Adj(v) \cup \{v\}\right\}$, and let V_l be such that $|V_l| = max_{j=1,\cdots,k+1} |V_j|$. We will prove that the set of vertices S,

$$S = \begin{cases} Adj(v) & \text{if } l = k+1 \\ V(\overline{K}_l) & \text{otherwise} \end{cases}$$

is the required k-separator. Further, $\{\tilde{V}_1, \tilde{V}_2\}$ where $\tilde{V}_1 = V_l$ and $\tilde{V}_2 = V \setminus (V_l \cup S)$ is the required partition of $V \setminus S$ satisfying (5.1) - (5.2).

Indeed, by assumption, $\sum_{j=1}^{k} |V_j| = \sum_{j=1}^{k} |D(\overline{K}_j)| \geq \frac{2}{3}n - 1$, which coupled with $\sum_{j=1}^{k+1} |V_j| = n - (k+1)$ implies that

$$|V_{k+1}| \leq n - (k+1) - (\tfrac{2}{3}n - 1) \leq \tfrac{2}{3}n . \qquad (5.5)$$

Moreover, since v is the first node encountered by the reduction process for which $|D(K_v)|$

$\geq \frac{2}{3}n$, we have that

$$|V_j| \leq \frac{2}{3}n \ , \ j = 1, \cdots, k. \tag{5.6}$$

Thus, by (5.5) and (5.6)

$$|\tilde{V}_1| = |V_l| \leq \frac{2}{3}n \leq \frac{k}{k+1} \ , \ k \geq 2 \ . \tag{5.7}$$

Furthermore, $|\tilde{V}_1| = |V_l| \geq \frac{n - (k+1)}{k+1}$, which implies that

$$|\tilde{V}_2| \leq n - k - \frac{n - (k+1)}{k+1} \leq \frac{k}{k+1} n \ . \tag{5.8}$$

By the definition of $\tilde{V}_1$, $\tilde{V}_2$ and S we also have that $\tilde{V}_1 \cap \tilde{V}_2 = \emptyset$, $\tilde{V}_1 \cup \tilde{V}_2 = V \setminus S$, and there is no edge in E with endpoints in $\tilde{V}_1$ and $\tilde{V}_2$. Thus, S is the required k-separator.

Case 2 $\sum_{j=1}^{k} |D(\bar{K}_j)| + 1 < \frac{2}{3}n$.

We will prove that if Case 2 holds then $S = Adj(v)$ is the required k-separator. Let $\tilde{D}(K_v)$ denote the set of descendant nodes of K_v just before it was updated by (5.3), which resulted with $|D(K_v)| \geq \frac{2}{3}n$, and construct the new sets $V_1 = \cup_{j=1}^{k} D(\bar{K}_j) \cup \{v\}$ and $V_2 = D(K_v)$. By assumption, $|V_1| < \frac{2}{3}n$. Further, $|V_2| < \frac{2}{3}n$, since otherwise Step 1 would have terminated earlier. Now, let $\tilde{V}_1 = V_1$ if $|V_1| \geq |V_2|$ and $\tilde{V}_1 = V_2$ otherwise, and let $\tilde{V}_2 = V \setminus (\tilde{V}_1 \cup S)$, where $S = Adj(v)$. Since $|V_i| \leq \frac{2}{3}n$, $i = 1,2,$ we have that

$$|\tilde{V}_1| < \tfrac{2}{3}n \leq \tfrac{k}{k+1}n \,, \quad k \geq 2 \,.$$

Moreover, by (5.4) $|V_1| + |V_2| = |D(K_v)| \geq \frac{2}{3}n$, which implies that $|\tilde{V}_1| = max(\,|V_1|, |V_2|\,) \geq \frac{1}{3}n$. Since $|\tilde{V}_1| + |\tilde{V}_2| = n - k$ we derive

$$|\tilde{V}_2| \leq n - k - \tfrac{1}{3}n \leq \tfrac{k}{k+1}n \,, \quad k \geq 2 \,.$$

Again, by the definition of $\tilde{V}_1$, $\tilde{V}_2$ and S we also have that $\tilde{V}_1 \cup \tilde{V}_2 = V \setminus S$, $\tilde{V}_1 \cap \tilde{V}_2 = \emptyset$ and there is no edge with endpoints in $\tilde{V}_1$ and $\tilde{V}_2$.

The above algorithm will produce a k-separator for a k-tree in linear time since Step 1 will be executed at most $n - k$ times, (5.3) and (5.4) require constant time for each iteration and Step 2 requires constant time. □

Let $G = (V, E)$ be a k-tree and S a k-separator of G which induces a partition of $V \setminus S$ into $\{V_1, V_2\}$ that satisfies (1) and (2) . Then, the subgraphs $G(V_1 \cup S)$ and $G(V_2 \cup S)$ induced, respectively, by $V_1 \cup S$ and $V_2 \cup S$ are also k-trees which can be similarly decomposed. We can proceed in this manner to decompose G recursively until we end up with k-cliques. That entire decomposition can be represented by a **balanced binary decomposition tree,** T , which is a rooted binary tree in which the root v represents the graph G , and if some vertex q is a parent of q_1 and q_2 in T then q_1 and q_2 are sub-k-trees of q obtained by the above decomposition of q .

Corollary 5.1.1 Let $G = (V, E)$ be a k-tree. Then, a balanced binary decomposition tree, T, of G can be constructed in $O(n \log n)$ time.

Proof: Let L denote the subset of vertices in T such that the path from the root of T to a vertex in L has l edges. The leaves of T are $k+1$ cliques and there are at most $n-k$ of them. Thus the number of vertices in all subgraphs of G represented by elements in L is bounded by $n + k(n\text{-}k)$. That implies that, the construction of k-separators of subgraphs of G represented by vertices in L takes $O(n)$ time. The proof then follows since T is, by (5.1), at most $O(\log n)$ deep, i.e. the path from the root of T to any of its leafs contains at most $O(\log n)$ edges.

5.2 SOME SEPARATION PROPERTIES OF PARTIAL k-TREES

We derive in this section some separation properties of partial k-trees into k-trees which will be used later to develop NC algorithms for the recognition and embedding of partial k-trees into k-trees when $k = 2$ and 3 .

Lemma 5.2.1 Let $G = (V, E)$ be a graph and S a separator of G which induces a partition of $V \setminus S$ into $\{V_1, V_2\}$ such that $0 \leq |S| = l \leq k$, $|V_i \cup S| \geq k$, $i = 1,2$, and the subgraph of G induced by S, $G(S)$, is an l-clique. Then G is a partial k-tree if and only if the subgraphs $G_1 = G(V_1 \cup S)$ and $G_2 = G(V_2 \cup S)$ induced, respectively, by $V_1 \cup S$ and $V_2 \cup S$ are partial k-trees.

Proof: If G is a partial k-tree then since G_1 and G_2 are subgraphs of G they are also partial k-trees. On the other hand, assume that G_1 and G_2 are partial k-trees and let $\tilde{G}_1 = (V_1 \cup S , \tilde{E}_1)$ and $\tilde{G}_2 = (V_2 \cup S , \tilde{E}_2)$, respectively, be their embeddings into k-trees. Clearly, $G(S)$ is a subgraph of $\tilde{G}_1$ and $\tilde{G}_2$. Therefore, there exist k-cliques K_1 and K_2 contained, respectively, in $\tilde{G}_1$ and $\tilde{G}_2$ and such that both K_1 and K_2 contain the l-clique $G(S)$. Let $v_{l+1}, \cdots, v_k$ and $u_{l+1}, \cdots, u_k$ denote the nodes in $V(K_1)$ and $V(K_2)$, respectively, that are not in S, and consider the graph $\tilde{\tilde{G}}$ obtained from $\tilde{G} = (V, \tilde{E}_1 \cup \tilde{E}_2)$ after the addition of the edges (v_i, u_j), $i = l+1, \cdots, k$, $j = l+1, \cdots, k$ and $j \geq i$. Then, it is easy to see that $\tilde{\tilde{G}}$ is a k-tree, and since G is a partial graph of $\tilde{\tilde{G}}$, it follows that G is a partial k-tree. (Note that in the degenerate case, when $l = 0$, we simply choose for K_1 and K_2 in the above proof any two k-cliques which are contained in $\tilde{G}_1$ and $\tilde{G}_2$.) □

We will use in the sequel the following notation. For a graph $G = (V, E)$ and subsets S_1, S_2 such that $S_1 \subseteq S_2 \subseteq V$ we will denote by $G(S_2 ; K(S_1))$ the subgraph of G induced by S_2 which is augmented with all arcs between pairs of nodes of S_1 if they are missing in G.

Lemma 5.2.2 Let $G = (V, E)$ be a two-connected graph and S, $S = \{s_1, s_2\}$, a separator of G which induces the partition of $V \setminus S$ into $\{ V_1, V_2 \}$ such that $V_i \neq \emptyset$ and $|V_i \cup S| \geq k$, $i = 1,2$. Then, G is a partial k-tree if and only if the subgraphs $G_1 = G(V_1 \cup S ; K(\{ s_1, s_2 \}))$ and $G_2 = G(V_2 \cup S ; K(\{ s_1, s_2 \}))$ are partial k-trees.

Proof: If $G(V_i \cup S ; K(\{s_1, s_2\}))$, $i = 1,2$, are partial k-trees then, by using Lemma 5.2.1 we can conclude that G is a partial k-tree. Thus, suppose that G is a partial k-tree. Since $V_1 \neq \emptyset$ and G is two-connected, there exists a vertex $v_1 \in V_1$ and two disjoint paths, $p(v_1, s_1)$ and $p(v_1, s_2)$, joining v_1 with s_1 and s_2, respectively, in the subgraph $G(V_1 \cup S)$. Therefore, $p(v_1, s_1)$ and $p(v_1, s_2)$ form a simple path, $p(s_1, s_2)$, between s_1 and s_2 in $G(V_1 \cup S)$, and contracting the edges along $p(s_1, s_2)$ would yield a 2-clique with a vertex set $S = \{s_1, s_2\}$. Thus, G is contractible to $G(V_2 \cup S ; K(\{s_1, s_2\}))$, which implies that $G(V_2 \cup S ; K(\{s_1, s_2\}))$

is a partial k-tree. Analogously, we obtain that $G(V_1 \cup S; K(\{s_1, s_2\}))$ is a partial k-tree as well. $\square$

Lemma 5.2.3 Let $G = (V,E)$ be a triconnected simple graph and S, $S = \{ u_1, u_2, u_3 \}$, a separator of G inducing a partition of $V \setminus S$ into $\{ V_1, V_2 \}$ such that $|V_i| \geq 2$, $i = 1, 2$, and $|V_i \cup S| \geq k$, $i = 1, 2$. Then, G is a partial k-tree if and only if the subgraphs $G(V_i \cup S; K(S))$, $i = 1, 2$, are partial k-trees.

Proof: If $G(V_i \cup S; K(S))$ are partial k-trees then, by using Lemma 5.2.1 we can conclude that so is G. So, assume that G is a partial k-tree and let v_1, v_2 be in V_1. By the triconnectivity of G, there exist at least three vertex disjoint paths between v_1 and v_2 in G, and since S is a separator and $|S| = 3$, at least two of these paths are contained in $G(V_1 \cup S)$. Therefore, since G is assumed to be a simple graph, these two paths form a cycle $\mathfrak{C}$ in $G(V_1 \cup S)$ with at least 3 vertices. By the triconnectivity of G there exist 3 vertices, say $\overline{s}_1$, $\overline{s}_2$ and $\overline{s}_3$, in $\mathfrak{C}$ and vertex disjoint paths $p_1(s_1, \overline{s}_1)$, $p_2(s_2, \overline{s}_2)$, $p_3(s_3, \overline{s}_3)$ (some possibly degenerated to a single vertex) in $G(V_1 \cup S)$, such that $p_i(s_i, \overline{s}_i)$ $i = 1, 2, 3$, intersect with $\mathfrak{C}$ only in nodes $\overline{s}_i$, $i = 1, 2, 3$, respectively. Clearly, if we contract edges in the paths $p_i(s_i, \overline{s}_i)$, $i = 1, 2, 3$, and identify s_i with $\overline{s}_i$, for $i = 1, 2, 3$, respectively, we create a cycle $\mathfrak{C}'$ that contains s_1, s_2, s_3. Again, appropriate contractions of edges in $\mathfrak{C}'$ will create a 3-clique $K(S)$. Thus, G is contractible to $G(V_2 \cup S; K(S))$ and we conclude that $G(V_2 \cup S; K(S))$ is a partial k-tree. Analogously, we can show that $G(V_1 \cup S; K(S))$ is a partial k-tree. $\square$

We note that the above separation properties of partial k-tree imply the existence of a balanced decomposition tree, T, of an embedding of a biconnected (resp., triconnected) partial 2-

tree (resp., 3-tree) G into a 2-tree (resp., 3-tree). The root, r, of T is the graph G, and if q_1 and q_2 are sons of r in T then q_1 (resp., q_2) is the graph $G(V_1 \cup S; K(S))$ (resp., $G(V_1 \cup S; K(S)))$, where S is a two-separator (resp. 3-separator of G) satisfying (5.1) and (5.2), and whose existence follows from Theorem 5.1.1. The biconnectivity (resp., triconnectivity) of G implies that $G(V_1 \cup S; K(S))$ and $G(V_2 \cup S; K(S))$ are biconnected (resp., triconnected). Thus, if they have sufficient number of nodes, they can be decomposed in a similar manner. In general, the leaves of the decomposition tree T are biconnected (resp., triconnected) 2-trees (resp., 3-trees) which are not necessarily further decomposable in the same manner since they have less than 9 (resp., 20) nodes.

5.3 NC ALGORITHMS FOR RECOGNITION OF PARTIAL k-TREES FOR k = 2 AND 3

In this section we present NC-algorithms for recognizing a partial 2-tree (resp., three-connected partial 3-tree) graphs G and finding its embedding into a 2-tree (resp., 3-tree). Without loss of generality, we assume here and in Section 5.4 that G is a simple graph.

Algorithm 5.1

Procedure $BBD(G, \alpha, L)$

Input A two-connected (resp., three-connected) simple graph $G = (V, E)$.

Output $\alpha = 1$ if G is a partial 2-tree (resp., 3-tree), and $\alpha = 0$ otherwise. Further, if $\alpha = 1$

then the output L contains an embedding of G into a 2-tree (resp., 3-tree).

Step 0 Set $\alpha = 1$ and $L = \emptyset$.

Step 1

(1.1) If $|E| > 2n - 3$ (resp., $3n - 6$) set $\alpha = 0$ and exit .

(1.2) Else, if $|V| \leq 9$ (resp., 20) check in constant time whether G is a partial 2-tree (resp., 3-tree). If yes, find an embedding, $\overline{G}$, of G into a 2-tree (resp., 3-tree), let $L = L \cup \{\overline{G}\}$. Else, set $\alpha = 0$ and exit .

Step 2 Else, for each pair (resp., triple) of distinct vertices $S = \{s_1, s_2\}$ (resp., $S = \{s_1, s_2, s_3\}$) of V, in parallel, find all the connected components of $G\ (V \setminus S)$ and denote their vertex sets by $V_j^S : j = 1, \cdots, r_S$ for some $r_S \geq 1$.

(2.1) If $|V_j^S| > \frac{2}{3}|V|$ (resp., $|V_j^S| > \frac{3}{4}|V|$) for some $j = 1, \cdots, r_S$ reject S. If all the pairs (resp., triples) S were rejected set $\alpha = 0$ and exit.

(2.2) Else, if $\frac{1}{3}|V| \leq |V_j^S| \leq \frac{2}{3}|V|$ (resp., $\frac{1}{4}|V| \leq |V_j^S| \leq \frac{3}{4}|V|$) for some S and for some j, $1 \leq j \leq r_S$, set $M_1 = V_j^S \cup S$ and $M_2 = (V \setminus V_j^S)$.

(2.3) Else, for an unrejected S compute $q_l = \sum_{j \leq l} |V_j^S|$, $l = 1, \cdots, r_S$, and apply a binary search to find η for which $\frac{1}{4}|V| \leq q_\eta \leq \frac{2}{3}|V|$ (*resp.*, $\frac{1}{4}|V| \leq q_\eta \leq \frac{3}{4}|V|$). Set $M_1 = \cup_{j=1}^{\eta} V_j^S \cup S$ and $M_2 = (V \setminus M_1) \cup S$.

(2.4) Call in parallel $BBD(G(M_1 ; K(S)), \alpha_1, L_1)$ and $BBD(G(M_2 ; K(S)), \alpha_2, L_2)$. If either $\alpha_1 = 0$ or $\alpha_2 = 0$ set $\alpha = 0$ and exit. Else, set $L = L_1 \cup L_2$. Exit.

Theorem 5.3.1 Algorithm 5.1 recognizes whether a two-connected (resp., three-connected) simple graph $G=(V,E)$ with $|V| = n$ is a partial 2-tree (resp., 3-tree) and produces an embedding of G into a 2-tree (resp., 3-tree) in $O(log^2 n)$ time using $O(n^3)$ (resp., $O(n^4)$) processors.

Proof: We first show that the steps of Algorithm 5.1 are valid. If (1.1) in Step 1 is valid then, since every 2-tree (resp., 3-tree) has exactly $2n - 3$ (*resp.*, $2n - 6$) edges, it follows from Lemma 5.2.2 (resp., Lemma 5.2.3) that G is not a partial 2-tree (resp., 3-tree). If all pairs (resp., triples) in Step 2 were rejected, G does not contain a 2-separator (resp., 3-separator) and then, by Theorem 5.1.1, G is not a partial 2-tree (resp., 3-tree). Thus, by Lemma 5.2.2 (resp., Lemma 5.2.3) we can conclude that G is not a partial 2-tree (resp., 3-tree). The biconnectivity (resp., triconnectivity) of G implies the biconnectivity (resp., triconnectivity) of the minors $G(M_i\ ; K(S))$, $i = 1,2$, created in (2.2) and (2.3) of Step 2 respectively. Therefore, by Lemma 5.2.2 (resp., Lemma 5.2.3), G is a partial 2-tree (resp., 3-tree) if and only if all the minors $G(M_i\ ; K(S))$, $i=1,2$, created in Step 2 are partial 2-trees (resp., 3-trees).

Next, we will show that Algorithm 5.1 requires $O(log^2 n)$ time and $O(n^3)$ (resp., $O(n^4)$) processors. In Step 1, (1.2) can be performed in constant time using, for example, the sequential algorithm for the recognition and embedding of partial 2-trees (resp., 3-trees) into 2-trees (resp., 3-trees) of Wald and Colbourn (1983) (resp., Arnborg and Proskurowski (1986)). All connected components of $G(V \setminus S)$ can be found, by the parallel algorithm of Shiloach and Vishkin (1982), in $O(log\, n)$ time and using $O(n + m)$ processors, where $m = |E|$. Note that in our case m = $O(n)$. Since there are $O(n^2)$ (resp., $O(n^3)$) pairs (resp., triples) to be considered simultaneously, Step 2 requires $O(n^3)$ (resp., $O(n^4)$) processors. Further, in Step 2, (2.3) takes $O(log\, n)$ time using

$O(n^2)$ processors in order to compute partial sums, q_l, and to perform a binary search on them. Since in (2.2) and (2.3), $|M_i| \leq \frac{2}{3}|V|$ (resp., $|M_i| \leq \frac{3}{4}|V|$), $i = 1,2$, Algorithm 5.1 will terminate after performing at most $O(log\, n)$ nested calls of Procedure $BBD(G, \alpha, L)$. Thus, our algorithm recognizes whether a two-connected (resp., three-connected) graph G is a partial 2-tree (resp., 3-tree) and delivers in L an embedding of G into a 2-tree (resp., 3-tree) in $O(log^2 n)$ time using $O(n^3)$ (resp., $O(n^4)$) processors. □

Note that in Algorithm 5.1 we have restricted G to be biconnected (resp., triconnected). However, the recognition of partial 2-trees which are not necessarily biconnected can be easily carried out by a slight modification of Algorithm 5.1. Indeed, we can find all biconnected components of G using the parallel algorithm of Tarjan and Vishkin (1985) in $O(log\, n)$ time with $O(n)$ processors. If G does not have biconnected components, then it can be decomposed by zero or one separators into subgraphs of cardinality less than or equal to 3. Then, since every graph on 3 vertices is a partial 2-tree, Lemma 5.2.1 implies that G is a partial 2-tree. Otherwise, we perform Algorithm 5.1 on all biconnected components of G. By Lemma 5.2.1, G is a partial 2-tree if and only if all biconnected components of G are partial 2-trees. Clearly, the modified algorithm requires $O(log^2 n)$ time and $O(n^3)$ processors.

The recognition of partial 3-trees which are not necessarily three-connected requires a major modification of Algorithm 5.1, which is developed in the next section.

5.4 NC-Algorithm for Recognizing Partial 3-Trees

We develop in this section an NC-algorithm for recognizing a partial 3-tree which is not necessarily 3-connected. The performance of this algorithm is identical to that developed in Section 5 for recognizing and embedding 3-connected partial 3-trees. That is, it requires $O(log^2 n)$ time and $O(n^4)$ processors. However, its description is somewhat more involved and it depends on some new separation properties which are developed below. First, we need to introduce a new definition.

For two disjoint paths $p_1 = p_1(v, s_1)$ and $p_2 = p_2(v, s_2)$ in $G = (V, E)$, having only the vertex v in common, the path $p = p(k, l)$ will be called a **bridge path** between p_1 and p_2 if p originates at some node k in p_1, $k \neq v$, *and terminates at vertex* l in p_2, $l \neq v$, but p is otherwise vertex disjoint with p_1 and p_2.

Lemma 5.4.1 Let $S = \{s_1, s_2, s_3\}$ be a separator of $G = (\mathrm{V}, \mathrm{E})$ inducing a partition $\{V_1, V_2\}$ of $V \setminus S$ such that no edge in E connects a vertex in V_1 with a vertex of V_2. Assume that for $i = 1, 2$ there exists a vertex $v_i \in V_i$ and three disjoint paths $p(v_i, s_j)$, $j = 1,2,3$, joining v_i with s_j, $j = 1,2,3$. If for $i = 1$ and 2 there exists a bridge path between a pair of paths among the paths $p(v_i, s_j)$, $j = 1,2,3$, which is contained in $G(V_i \cup S)$, then G is a partial k-tree if and only if $G(V_1 \cup S; K(S))$ and $G(V_2 \cup S; K(S))$ are partial k-trees.

Proof: The existence of a bridge path between some pair of the paths $p(v_i, s_j)$, $j = 1,2,3$, for $i = 1$ and 2 imply that G is contractible to $G(V_1 \cup S; K(S))$ and $G(V_2 \cup S; K(S))$. Thus, if G is a partial k-tree, so are $G(V_i \cup S; K(S))$, $i = 1, 2$. On the other hand, Lemma 5.2.1 implies that if $G(V_i \cup S; K(S))$, $i = 1,2$, are partial k-trees then G must also be a partial k-tree. □

Lemma 5.4.2 Let $S = \{ s_1, s_2, s_3 \}$ be a 3-separator of a 2-connected simple graph $G = (V, E)$

which induces a partition $\{V_1, V_2\}$ of $V \setminus S$ such that $2 \leq |V_i| \leq \frac{3}{4}|V|$, $i = 1,2$, and no edge in E connects a vertex in V_1 with a vertex in V_2. Assume that for $i = 1,2$ there exist vertices $v_i \in V_i$ and three disjoint paths $p(v_i, s_j)$, $j = 1,2,3$, joining v_i with vertices in S. If for $i = 1$ or 2 there exists no bridge path between any pair of paths among the paths $p(v_i, s_j)$, $j = 1,2,3$, which is contained in $G(V_i \cup S)$ then for some $j \in \{1,2,3\}$ and $i \in \{1,2\}$, $\{v_i, s_j\}$ is a separator of G which induces a partition $\{V_1', V_2'\}$ of $V \setminus \{v_i, s_j\}$ such that $\frac{1}{12}|V| - 2 \leq |V_1'| \leq \frac{3}{4}|V|$.

Proof: Assume, without loss of generality, that there is no bridge path between any pair of paths among the paths $p(v_1, s_j)$, $j = 1,2,3$. Since G is connected, for any $v \in V_1$ there exists a node $s(v)$, $s(v) \in S$, and a path from v to $s(v)$, $p(v, s(v))$, in G which does not pass through nodes in $S \setminus \{s(v)\}$. Now, any node v, $v \in V_1$ and $v \neq v_1$, must be in some connected component of $G(V \setminus \{s(v), v_1\})$ which does not contain $S \setminus \{s(v)\}$. Indeed, otherwise there must exist a path, $p(v, \overline{s}(v))$, in G from v to a node $\overline{s}(v)$, $\overline{s}(v) \in S \setminus \{s(v)\}$, which does not pass through either v_1 or $s(v)$. But, the paths $p(v, s(v))$ and $p(v, \overline{s}(v))$ in G induce a bridge path in $G(V_1 \cup S)$ between $p(v_1, s(v))$ and $p(v_1, \overline{s}(v))$, which is a contradiction.

Next, for $t = 1,2,3$ let C_t denote, respectively, the union of all connected components of $G(V \setminus \{v_1, s_t\})$ which do not contain any node in $S \setminus \{s_t\}$. We claim that the sets C_t, $t \in \{1,2,3\}$, are mutually exclusive. Indeed, from the 2-connectivity of G and the definition of the sets C_t it follows that if $v \in C_i$, $i \in \{1,2,3\}$, and $v \neq v_1$, then there exists a path $p(v, s_i)$, $s_i \in S$, which does not pass through $\{v_1\} \cup S \setminus \{s_i\}$. Thus, if $v \in C_i \cap C_j$, $i,j \in \{1,2,3\}$ and $i \neq j$, the paths $p(v, s_i)$ and $p(v, s_j)$ in G would induce a bridge path between $p(v_1, s_i)$ and $p(v_1, s_j)$, which is a contradiction. Thus, we have that $\sum_{t=1}^{3} |V(C_t)| = |V_1| - 1$, where $V(C_t)$ denotes the set of nodes in C_t. Let $|V(C_j)| = max\{ |V(C_t)| : t = 1,2,3 \}$, and set $V_1' = V(C_j)$ and $V_2' = V \setminus (V_1' \cup \{v_1, s_j\})$. Since S is a 3-separator, $|V_1| \geq \frac{1}{4}|V| - 3$ and thus, $|V_1'| \geq \frac{1}{12}|V| - 2$. Clearly, $V(C_t) \subseteq V_1$, $t = 1,2,3$, and thus $|V_1'| \leq \frac{3}{4}|V|$. □

We use the above separation results to construct an NC-algorithm for recognizing a partial 3-tree.

Algorithm 5.2

Procedure $REC(\overline{G}, \alpha)$

Input A simple graph $\overline{G} = (\overline{V}, \overline{E})$.

Output $\alpha = 1$ if $\overline{G}$ is a partial 3-tree and $\alpha = O$ otherwise .

Step 1 Find all biconnected components of $\overline{G}$. If $\overline{G}$ has no biconnected components, set $\alpha = 1$ and exit. Otherwise, perform in parallel procedure $REC1(G_i, \alpha_i)$ for all biconnected components $G_i = (V_i, E_i)$ of $\overline{G}$. If any $\alpha_i = 0$ set $\alpha = 0$ and exit.

Procedure $REC1(G, \alpha)$

Input A simple biconnected graph $G = (V, E)$.

Output $\alpha = 1$ if G is a partial 3-tree and $\alpha = 0$ otherwise .

Step 0 $\alpha = 1$.

Step 1

(1.1) If $|E| > 3n - 6$, set $\alpha = 0$ and exit .

(1.2) Else, if $|V| < 36$ check, in constant time, whether G is a partial 3-tree. If G is not a partial 3-tree set $\alpha = 0$. Exit .

Step 2 Else, for each triple of distinct vertices $S = \{s_1, s_2, s_3\}$ of V, in parallel, find all connected components of $G(V \setminus S)$ and denote their vertex sets by V_j^S, $j = 1, \cdots, r_S$, for some $r_S \geq 1$.

(2.1) If $|V_j^S| > \frac{3}{4}|V|$ for some $j = 1, \cdots, r_S$, reject S. If all triples S are rejected, set $\alpha = 0$ and exit.

(2.2) Else, if $\frac{1}{4}|V| \leq |V_j^S| \leq \frac{3}{4}|V|$ for some S and for some j, $1 \leq j \leq r_S$, set $M_1 = V_j^S \cup S$ and $M_2 = (V \setminus V_j^S)$.

(2.3) Else, for an unrejected S and for all $j = 1, \cdots, r_S$, compute $q_l = \sum_{j \leq l} |V_j^S|$, $l = 1, \cdots, r_S$, and apply a binary search to find an η for which $\frac{1}{4}|V| \leq q_\eta \leq \frac{3}{4}|V|$. Set $M_1 = S \cup (\cup\ V_j^S : j \leq \eta)$ and $M_2 = (V \setminus M_1) \cup S$.

Step 3 Find, in parallel, connected components of $G(M_t \setminus \{s_i, s_j\})$ which do not contain $S \setminus \{s_i, s_j\}$ for $t = 1, 2$, $i \neq j$, $i, j \in \{1, 2, 3\}$. Denote by $C_{i,j}^t$ the union of all such connected components, and let $C_{i,j} = C_{i,j}^1 \cup C_{i,j}^2$ for all $\{i, j\} \in \{1, 2, 3\}$, $i \neq j$.

(3.1) If $C_{1,2} \neq \emptyset$, $C_{2,3} \neq \emptyset$ and $C_{1,3} \neq \emptyset$ let $G_1 = G(M_1 ; K(S))$ and $G_2 = G(M_2 ; K(S))$.

(3.2) Else, if there exist at least two distinct pairs $\{i, j\}$ and $\{p, q\}, \{i, j\} \cup \{p, q\} = \{1, 2, 3\}$ such that $C_{i,j} = \emptyset$ and $C_{p,q} \neq \emptyset$ proceed as follows:

(a) If $|V(C^1_{1,2})| + |V(C^1_{2,3})| + |V(C^1_{1,3})| < |M_1 \setminus S|$ and $|V(C^2_{1,2})| + |V(C^2_{2,3})| + |V(C^2_{1,3})| < |M_2 \setminus S|$, where $V(C^t_{i,j})$ is the vertex set of component $C^t_{i,j}$, then let $G_1 = G(M_1 ; K(S))$ and $G_2 = G(M_2 ; K(S))$.

(b) Else, if for some $t \in \{1,2\}$, $|V(C^t_{1,2})| + |V(C^t_{2,3})| + |V(C^t_{1,3})| = |M_t \setminus S|$ find $|V(C^t_{k,l})| = max\{ |V(C^t_{1,2})|, |V(C^t_{1,3})|, |V(C^t_{2,3})| \}$ and let $G_1 = G(V(C^t_{k,l}) \cup \{s_k, s_l\} ; K(\{s_k, s_l\}))$ and $G_2 = G((V \setminus V(C^t_{k,l})) \cup \{s_k, s_l\} ; K(\{s_k, s_l\}))$.

(3.3) Else, if $C_{1,2} = \emptyset$, $C_{2,3} = \emptyset$ and $C_{1,3} = \emptyset$ find, in parallel, connected components of $G(V \setminus \{s_t , v\})$, for all $t \in \{1,2,3\}$ and $v \in V$, and denote their vertex sets by $V_j(v, s_t)$, $j = 1, \cdots, m(v, s_t)$, for some $m(v, s_t) \geq 1$. If $|V_j(v, s_t)| > \frac{11}{12}|V|$ for some j, $j \in \{1, \cdots, m(v, s_t)\}$, reject the pair $\{v, s_t\}$.

(a) If all pairs $\{v, s_t\}$, $t \in \{1,2,3\}$ and $v \in V$, were rejected let $G_1 = G(M_1 ; K(S))$ and $G_2 = G(M_2 ; K(S))$.

(b) Else, if $\frac{1}{12}|V| - 2 \leq |V_j(v, s_t)|$ for some $j \in \{1, \cdots, m(v, s_t))\}$, set $M'_1 = V_j(v, s_t) \cup \{v, s_t\}$ and $M'_2 = V \setminus V_j(v, s_t)$ and let $G_1 = G(M'_1 ; K(\{v, s_t\}))$ and $G_2 = G(M'_2 ; K(\{v, s_t\}))$.

(c) Else, for an unrejected pair $\{v, s_t\}$ compute $q_l = \sum_{i \leq l} |V_i(v, s_t)|$, $l = 1, \cdots, m(v, s_t)$, and apply a binary search to find η for which $\frac{1}{12}|V| - 2 \leq q_\eta \leq \frac{3}{4}|V|$. Set $M'_1 = \{v, s_t\} \cup (\cup V_j(v, s_t) : j \leq \eta)$, $M'_2 = (V \setminus M'_1) \cup \{v, s_t\}$, $G_1 = G(M'_1 ; K(\{v, s_t\}))$ and $G_2 = G(M'_2 ; K(\{v, s_t\}))$.

(3.4) Call in parallel $REC1(G_1, \alpha_1)$ and $REC1(G_2, \alpha_2)$. If either $\alpha_1 = 0$ or $\alpha_2 = 0$ set $\alpha = 0$. Exit.

Theorem 5.4.1 Algorithm 5.2 recognizes whether a simple graph $\overline{G} = (\overline{V}, \overline{E})$ with $|\overline{V}| = n$ is a partial 3-tree in $O(log^2\ n)$ time using $O(n^4)$ processes.

Proof: We will first prove the validity of Algorithm 6.1. If, in Procedure $REC(\overline{G}, \alpha)$, $\overline{G}$ is found not to contain biconnected components, then it could be decomposed by zero or one separators into subgraphs of cardinality at most 4. Since every graph on 4 nodes is a partial 3-tree, Lemma 5.2.1 implies that $\overline{G}$ is a partial 3-tree. The validity of Step 1 in $REC1(G, \alpha)$ was explained in Algorithm 5.1. If all triples were rejected in Step 2, G does not contain a 3-separator and by Theorem 3.1, G is not a partial 3-tree. Otherwise, in Step 2 algorithm 5.2 finds a 3-separator. In Step 3, if $C_{i,j} \neq \emptyset$ for some i, j, $i \neq j$, then $\{s_i, s_j\}$ is a separator of G, and by the biconnectivity of G and Lemma 5.2.2 it follows that G is a partial 3-tree if and only if G augmented with edge (s_i, s_j) is a partial 3-tree. Thus, by Lemma 5.2.3, if (3.1) holds, then G is a partial 3-tree if and only if $G(M_1\ ;\ K(S))$ and $G(M_2\ ;\ K(S))$ are partial 3-trees. If (3.2) holds then, by Lemma 5.2.2, we can augment G with edge (p, q). Therefore, if 3.2a) is valid, there must exist vertices v_i, $v_i \in M_i \setminus S$, i=1,2, and three disjoint paths from v_i to all $s \in S$ in $G(M_i)$, i=1,2. Thus, by Lemma 5.4.1, since $(p, q\ )$ is a bridge path both in $G(M_1)$ and $G(M_2)$, G is a partial 3-tree if and only if $G(M_1\ ;\ K(S))$ and $G(M_2\ ;\ K(S))$ are partial 3-trees. On the other hand, if 3.2b) holds then, by Lemma 5.2.2, G is a partial 3-tree if and only if $G(\ V(C^t_{k,l}) \cup \{\ s_k,\ s_l\ \}\ ;\ K(\{\ s_k,\ s_l\}))$ and $G((V \setminus V(C^t_{k,l})) \cup \{s_k, s_l\}\ ;\ K(\{s_k, s_l\}))$ are partial 3-trees. If (3.3) holds, then for each $i = 1, 2$ and for each $v \in M_i$ there exist three disjoint paths in $G(M_i)$ from v to all nodes in S. Therefore, by Lemma 5.4.2, if all pairs are rejected in 3.3a), there must exist a bridge path both in $G(M_1)$ and $G(M_2)$ satisfying the stipulations in Lemma

5.4.2 . Then, by Lemma 5.4.1, G is a partial 3-tree if and only if $G(M_1 ; K(S))$ and $G(M_2 ; K(S))$ are partial 3-trees. Otherwise, by Lemma 5.2.2, G is a partial 3-tree if and only if $G(M'_1 ; K(\{ v, s_t \}))$ and $G(M'_2 ; K(\{v, s_t\}))$ are partial 3-trees.

Next, we will show that algorithm 5.2 requires $O(log^2 n)$ time using $O(n^4)$ processors. All biconnected components can be found in Step 0, using the parallel algorithm of Tarjan and Vishkin (1985), in $O(log\, n)$ time using $O(n)$ processors. In Step 1, (1.2) can be carried out in constant time using, for example, the sequential algorithm for recognizing partial 3-trees of Arnborg and Proskurowski (1986). The connected components in Step 2 can be found by the parallel algorithm of Shiloach and Vishkin (1982) in $O(log\, n)$ time with $O(n + m)$ processors. Since there are $O(n^3)$ triples S to be considered simultaneously, Step 2 requires $O(log\, n)$ time and $O(n^4)$ processors. It takes $O(log\, n)$ time using $O(n^2)$ processors to compute, in Step 2.3, partial sums q_l and perform a binary search on them. In Step 3, connected components of $G(M_t \setminus \{s_i, s_j\})$ which do not contain $S \setminus \{s_i, s_j\}$ can be found in $O(log\, n)$ time using $O(n)$ processors. Similarly, connected components of $G\ (V \setminus \{s_t, v\})$ can be found in $O(log\, n)$ time using $O(n)$ processors, and since there are $O(n)$ pairs $\{s_t, v\}$ to be considered simultaneously, Step (3.3) requires $O(log\, n)$ time and $O(n^2)$ processors. Further, 3.3c) takes $O(log\, n)$ time using $O(n^2)$ processors. Finally, observe that in (2.2) $|V_j^S| \geq \frac{1}{4}|V|$, in (2.3) $q_\eta \geq \frac{1}{4}|V|$ and, since S is a 3-separator, in Step 3.2b) $|V(C^t_{k,l})| \geq \frac{1}{12}|V|$. Further, in Step 3.3, $|V_j(v, s_t)| \geq \frac{1}{12}|V| - 2$ and $q_\eta \geq \frac{1}{12}|V| - 2$. Therefore, we can have at most $O(log\, n)$ nested calls of $REC1(G, \alpha)$, and algorithm 5.2 would terminate in $O(log^2 n)$ time using $O(n^4)$ processors. □

Bibliography

A.V. Aho, J.E. Hopcroft and J.D. Ullman, "The Design and Analysis of Computer Algorithms," *Addison-Wesley, Reading*, 1974

S. Arnborg, "Reduced State Enumeration: Another Algorithm for Reliability Evaluation," IEEE Transactions on Reliability 27 (1978), 101-105.

S. Arnborg, "On the Complexity of Multivariable Query Evaluation," FOA Rapport C 20292-D8, National Defence Research Institute, Stockholm, Sweden (1979).

S. Arnborg, "Efficient Algorithms for Combinatorial Problems on Graphs with Bounded Decomposability - A Survey," *BIT* 25 (1985), 2-33.

S. Arnborg, D.G. Corneil and A. Proskurowski, "Complexity of Finding Embeddings in a k-tree," *SIAM Journal on Algebraic and Discrete Methods* 8 (1987), 277-284.

S. Arnborg and A. Proskurowski, "Characterization and Recognition of Partial k-Trees," *Congressus Numerantium*, 47 (1985), 69-75

S. Arnborg and A. Proskurowski, "Characterization and Recognition of Partial 3-trees," *SIAM Journal on Algebraic and Discrete Methods* 7 (1986), 305-314.

S. Arnborg and A. Proskurowski, "Linear Time Algorithms for NP-hard Problems on Graphs Embedded in k-trees," *Discrete Applied Mathematics* 23 (1989) 11-24.

S. Arnborg, A. Proskurowski and D. Corneil, "Forbidden Minors Characterizations of Partial 3-trees," CIS-TR-86-07, University of Oregon, July 1986.

N.W. Bern, E.L. Lawler and A.L. Wong, "Why Certain Subgraph Computations Require Only Linear Time," *Proc. 26th Ann. IEEE Symp. on Foundations of Computer Science* (1985), 117-125.

C. Berge, "Graphs," 2^{nd} *revised edition Amsterdam, North Holland Publishing Company, NY, NY, USA*, 1985.

C. Bird, "On Cost Allocation for a Spanning Tree: A Game Theoretic Approach," *Networks* 6 (1976), 335-350.

H.L. Bodlaender, "Classes of Graphs With Bounded Tree-Width," RUU-CS-86-22, Rijksuniversiteit Utrecht, The Netherlands, November 1986.

H.L. Bodlaender, "NC-Algorithms for graphs with small treewidth," RUV-CS-88-4, Rijksuniversiteit Utrecht, The Netherlands, February 1988.

N. Chandrasekharan and S.S. Iyengar, "NC Algorithms for Recognizing Chordal Graphs and k-Trees," Technical Report 86-020, Department of Computer Science, Louisiana State University, 1986.

C.J. Colbourn and A. Proskurowski, "Concurrent Transmission in Broadcast Networks," *Proc. Int. Conf. Automata, Languages, Programming*, Springer-Verlag LNCS 172 (1984), 128-136.

D.G. Corneil and J.M. Keil, "A Dynamic Programming Approach to the Dominating Set Problem on k-Trees," *SIAM Journal on Algebraic and Discrete Methods* 8 (1987), 535-543.

D.G. Corneil and Y.Perl, "Clustering and Domination in Perfect Graphs," *Discrete Applied Mathematics* 9 (1984), 27-39

G.Cornuejols, J.Fonlupt and D.Naddef, "The traveling Salesman Problem and Some Related Integer Polyhedra," *Mathematical Programming* 33 (1985), 1-27

A. Edenbrandt, "Combinatorial Problems in Matrix Computation," Ph.D. Thesis at Cornell University, Ithaca, NY, July 1985.

A.M. Farley, "Networks Immune to Isolated Failures," *Networks* 11 (1981), 255-268.

A.M. Farley and A. Proskurowski, "Networks Immune to Isolated Line Failures," *Networks* 12 (1982), 393-403.

L.R. Ford Jr. and D.R. Fulkerson, "Flows in Networks," *Princeton University Press, Princeton N.J.* 1962.

S. Fortune and J. Wyllie, "Parallelism in Random Access Machines," *Proceedings of the 10th Annual ACM Symposium on Theory of Computing,* (1978) 114-118.

M.C. Golumbic, "Algorithmic Graph Theory and Perfect Graphs," *Academic Press, New York,* 1980.

M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco (1979).

F. Gavril, "Algorithms for Minimum Coloring, Maximum Clique, Minimum Coloring by Cliques and Maximum Independent Set of a Chordal Graph," *SIAM Journal on Computing* 1 (1972) 180-187.

D. Granot, "A Generalized Linear Production Model: A Unifying Model," *Mathematical Programming* 34 (1986), 212-223.

D. Granot and F. Granot, "A Fixed-Cost Spanning-Forest Problem," University of British Columbia, Working Paper No. 1235, March 1987.

D. Granot and G. Huberman, "Minimum Cost Spanning Tree Games," *Mathematical Programming* 21 (1981) 1-18.

S.L. Hakimi and A.T. Amin, "ON the Design of Reliable Networks," *Networks* 3 (1973), 241-260

G.Y. Handler, "Minimax Location of a Facility in an Undirected Tree Graph," Transp. Sc. 7 (1973), 287-293.

F. Harary, "Graph Theory," *Addison Wesley, Reading, Mass.*, 1969.

R. Hassin and A. Tamir, "Efficient Algorithms for Optimization and Selection on Series-Parallel Graphs," *SIAM Journal on Algebraic and Discrete Methods* 7 (3), July 1986, 379-389.

D.S. Johnson, "The NP-Completeness Column: An Ongoing Guide," *Journal of Algorithms* 6 (1985), 434-451.

A. Kolen and A. Tamir, "Covering Problems" Chapter 6. in *Discrete Location Theory*, L.R. Francis and D.B. Mirchandani, eds., John Wiley, New York (forthcoming).

S.C. Littlechild, "A Simple Expression for the Nucleolus in a Special Case", *International Journal on Game Theory* 3 (1974), 21-29.

N. Megiddo, "Computational Complexity and the Game Theory Approach to Cost Allocation for a Tree", *Mathematics of O.R.* 3 (1978), 189-196.

B. Monien and I.H. Sudborough, "Bandwidth-Constrained NP-complete Problems," *Proceedings of the 13th Annual ACM Symposium on Theory of Computing, New York* (1981), 207-217.

E.M. Neufeldt and C.J. Colbourn, "The Most Reliable Series-Parallel Networks," Dept. of Computer Science, University of Saskatchewan, TR83-7 (1983).

C.H. Papadimitriou and K. Steiglitz, "Combinatorial Optimization Algorithms and Complexity," *Prentice Hall, Inc.*, 1982.

N. Pippenger, "On Simultaneous Resource Bounds," *Proc. 20th Ann. IEEE Symp. on Foundations of Computer Science* (1979), 307-311.

A. Prodon, M. Liebling and H. Gröflin, "Steiner's Problem on Two-Trees," RO 850315, Department of Mathematics, EPF Lausanne, Switzerland, March 1985.

A. Proskurowski, "Separating Subgraphs in k-trees: Cables and Caterpillars," *Discrete Mathematics* 49 (1984), 275-285.

R.L. Rardin, R.G. Parker and M.B. Richey, "A Polynomial Algorithm for a Class of Steiner Tree Problems on Graphs, " ISYE Report Series J-82.5, Georgia Tech., August 1982.

N. Robertson and P.D. Seymour, "Graph Minors II: Algorithmic Aspects of Tree Width," *Journal of Algorithms* 7 (1986), 309-322.

N. Robertson and P.D. Seymour, "Disjoint paths -A survey," *SIAM Journal on Algebraic and Discrete Methods* 6 (1985), 300-305.

D.J. Rose, "On Simple Characterizations of k-Trees," *Discrete Mathematics* 7 (1974), 317-322.

J.C. Sheperdson and H.E. Sturgis,"Computability of Recursive Functions," *Journal of the ACM* 10, (1963), 217- 255

Y. Shiloach and U. Vishkin, " An $O(\log n)$ Parallel Connecting Algorithm," *Journal of Algorithms* 3 (1982), 57-67.

K. Takamizawa, T. Nishizeki and N. Saito, "Linear-Time Computability of Combinatorial Problems on Series-Parallel Graphs," *Journal of the ACM* 29 (1982), 623-641.

R.E. Tarjan and U. Vishkin, "An Efficient Parallel Biconnectivity Algorithm," *SIAM Journal on Computing* 14 (1985), 862-874.

P. Tseng, "Parallel Computation for Edge Weighted Network Problems on Series-Parallel Graphs," Working Paper #1240 , University of British Columbia, December 1987.

J. Valdez, R.E. Tarjan and E.L. Lawler, "The Recognition of Series-Parallel Digraphs," *SIAM Journal on Computing* 11 (1982), 298-313.

U. Viskin, "Implementation of Simultaneous-Memory Access in Models That Forbid It," *Journal of Algorithms* 4, (1983), 45-50

A. Wald and C.J. Colbourn, "Steiner Trees, Partial 2-trees, and Minimum IFI Networks," *Networks* 13 (1983), 159-167.