SIAM J. ALG. DISC. METH.
Vol. 7, No. 3, July 1986

006

EFFICIENT ALGORITHMS FOR OPTIMIZATION AND SELECTION ON SERIES-PARALLEL GRAPHS*

R. HASSIN† AND A. TAMIR†

Abstract. It is well known that a series-parallel multigraph G can be constructed recursively from its edges. This construction is represented by a binary decomposition tree. 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 leaf (tip) of T represents a distinct edge of G. If q is not a leaf then it is either of a series or parallel type. If q_1 and q_2 are the two sons of q on T, $G(q)$ is the submultigraph obtained from $G(q_1)$ and $G(q_2)$ by the respective series or parallel composition.

In this paper we use this tree to develop efficient algorithms for several optimization and selection problems defined on graphs with no K_4 homeomorph. In particular, we provide a linear time algorithm to find the shortest simple paths from a given vertex to all other vertices. We also construct an $O(n^4)$ algorithm to solve the uncapacitated plant location problem, where n is the number of vertices.

We prove that graphs with no K_4 homeomorph contain a 2-separator, and then use it to show that the kth longest path in the set of all intervertex distances can be selected in $O(n \log^3 n)$ time.

Key words. series-parallel graphs, graph decomposition, selection algorithms, uncapacitated plant location problem

AMS(MOS) subject classification. 90C35

1. Introduction. It is well known that various optimization problems that are NP-hard on general graphs can be solved efficiently on trees, by polynomial algorithms. Some of these algorithms (e.g. [KH], [KT], [MTZC]) are of the dynamic programming nature. They represent the given tree as a rooted tree, and then recursively solve subproblems associated with a sequence of subtrees, (while starting with the tips of the rooted tree). In other words, these algorithms rely heavily upon the existence of an efficient construction that recursively generates larger components from previous ones, and terminate with the given tree, T. The key property of this construction is that each component T_i, generated in the process, is associated with some distinguished vertex, say v_i, such that, every simple path connecting a vertex in T_i with a vertex in $T - T_i$ must contain v_i.

Another property of trees which is useful in designing efficient divide-and-conquer algorithms, is the existence of a 1-separator (e.g. [GOL], [KH]). A 1-separator of an n-vertex tree $T=(V, E)$ is a vertex v for which there exists a partition of $V-\{v\}$ consisting of V_1 and V_2 with,

(1) $V_1 \cap V_2 = \varnothing$,

(2) $|V_i| \leqq (2/3)n$, $i=1, 2$, and

(3) no edge in E connects a vertex in V_1 with a vertex in V_2.

Another class of graphs for which an efficient construction (or decomposition) scheme is available is the class of series-parallel graphs [VTL]. Such a construction is represented by a binary decomposition tree that can be found in linear time [VTL]. Here, each component G_i of the underlying graph G, generated in the process is associated with two vertices, say $\{u_i, v_i\}$ such that every simple path connecting a vertex in G_i with a vertex in $G - G_i$ must contain either u_i or v_i.

In this paper we focus on graphs, not necessarily biconnected, that contain no K_4 homeomorph. (Note that this class of graphs contains all trees.) Our main goal is to extend some of the results mentined above with respect to trees. We will use the

* Received by the editors May 3, 1984, and in revised form July 17, 1985.
† Department of Statistics, Tel Aviv University, Tel Aviv 69978, Israel.

above binary decomposition tree to design efficient recursive algorithms for several optimization problems. We will deal mainly with problems which are not affected by adding to a graph edges with infinite length, (e.g. shortest paths problems). (This will enable us to convert a graph with no K_4 homeomorph into an equivalent series parallel graph.) In particular, we produce a linear time algorithm to find the shortest simple paths from a given vertex to all other vertices. We also construct an $O(n^4)$ algorithm to solve the simple plant location problem on an n-vertex graph with no K_4 homeomorph. (The latter is NP-hard for general graphs.)

We also use the binary decomposition tree to prove the existence and find, in linear time, a 2-separator of a graph with no K_4 homeomorph. (A 2-separator of an n-vertex graph $G=(V, E)$ is a pair of vertices $\{u, v\}$ associated with a partition of $V-\{u, v\}$ to sets V_1 and V_2 satisfying (1)-(3) above.)

We mentioned above that the 1-separator of a tree has been used in designing efficient divide and conquer algorithms on trees. We claim that the 2-separator is useful for these purposes on graphs with no K_4 homeomorph. This is demonstrated in § 7, where we study the selection of the kth longest simple path in the set of intervertex distances. This set is of $O(n^2)$ cardinality. However, we will show that the kth largest element in this set can be found in $O(n \log^3 n)$ time (i.e., without explicitly generating the entire set). This result generalizes similar results for tree graphs, [FJ2], [MTZC].

At this point it is worth mentioning the seminal works of Robertson and Seymour, [RS1], [RS2] that extend the concept of a binary decomposition tree, described above, for general graphs. Their results are quite general and therefore are not always most efficient when applied to certain classes of graphs. Specifically, when we apply their results to graphs with no K_4 homeomorph, we get weaker results than those reported above. In particular, their work implies the existence of a 3-separator, instead of the 2-separator result reported above. This is rather crucial for our purposes, since our selection algorithm for the kth longest path will fail if 2-separators are replaced by 3-separators. Also, their general work does not yield linear time algorithms for finding the separators and the binary decomposition tree.

2. Graph-theoretic definitions. A *multigraph* $G=(V, E)$ consists of a finite set of *vertices* V and a finite multiset of *edges* E. Each edge is a pair (u, v) of distinct vertices. If all the edges of G are ordered pairs G is called a *directed* multigraph, and if all its edges are unordered pairs G is an *undirected* multigraph. If E is a set, G is a *graph*, i.e. G contains no parallel edges.

A vertex v is a *cut vertex* of G if the removal of v, and all the edges incident to v, from G results in a disconnected graph. G is *biconnected* if it has no cut vertex. A biconnected *component* of G is a maximal submultigraph of G which is biconnected. A multigraph G contains a subgraph *homeomorphic* to a graph H, if H can be obtained from G by a sequence of the following operations:

(a) remove an edge,

(b) remove an isolated vertex

(c) if a vertex v has degree two, remove v and replace the two distinct edges (u, v) and (v, w) incident to v by an edge (u, w).

This paper focuses on a class of multigraphs, called series-parallel, which are defined as follows:

We say that edges e_1 and e_2 are *in series* if they have a single common vertex which is of degree two. The edges e_1 and e_2 are *parallel* if they have the same set of end vertices. By a *series construction* of an edge $e=(u, v)$ we mean the replacement of e by two edges in series, i.e., by (u, w) and (w, v), such that w is a new vertex. By

a *parallel construction* of an edge $e=(u, v)$ we mean the replacement of e by two parallel edges e_1 and e_2, having u and v as their end vertices. By a *series reduction* of two edges in series $e_1=(u, w)$ and $e_2=(w, v)$ we mean the replacement of e_1 and e_2 by a new edge $e=(u, v)$. By a *parallel reduction* of two parallel edges $e_1=(u, v)$ and e_2 we mean the replacement of e_1 and e_2 by a new edge $e=(u, v)$.

A *series-parallel multigraph* (SPM) is a multigraph that can be obtained by a sequence of series and parallel constructions of edges starting from a single edge. Alternatively, G is SPM if it can be reduced to a single edge by a sequence of series and parallel reductions.

3. Basic results on series-parallel multigraphs. The basic theorem on a SPM is due to Dirac [DIR] and Duffin [DUF].

THEOREM 3.1. *Let G be a multigraph. Then each biconnected component of G is series-parallel if and only if G does not contain a subgraph homeomorph to K_4, the complete graph on four vertices.*

The next result shows that a multigraph that does not contain K_4 as a homeomorph, can be augmented by edges to obtain a biconnected SPM.

THEOREM 3.2. *Let G be a connected multigraph that does not contain K_4 as a homeomorph. If G has k biconnected components, then G can be augmented by at most $k-1$ edges such that the resulting multigraph is a biconnected SPM.*

Proof. The proof is by induction on k, the number of biconnected components. Let $G_1, \cdots, G_k$, $k \geqq 2$, be the biconnected components of G. Each pair of biconnected components has at most one vertex in common. Suppose without loss of generality that G_1 and G_2 have a vertex in common, say u. Let u_i be a vertex in G_i, $i=1, 2$, which is adjacent to u (i.e., (u, u_i) is an edge of G_i). Connect u_1 and u_2 by an edge, and call the new multigraph G'. Clearly G_1 and G_2, with the augmented edge, form a biconnected submultigraph which we denote by $G_{1,2}$. The biconnected components of G' are $G_{1,2}$, $G_3, \cdots, G_k$. A multigraph does not contain a K_4 homeomorph if and only if none of its biconnected components does. Thus, it suffices to show that $G_{1,2}$ does not contain a K_4 homeomorph.

Suppose that $G_{1,2}$ contains a K_4 homeomorph, H, and let x, y, z and v be the four vertices of H of degree 3. Assume first that none of G_1 and G_2 contains all of these four vertices. Thus, without loss of generality, let x be in G_1, y be in G_2, and $x \neq u$, $y \neq u$. Since x and y are of degree 3, there exist in H three (intermediate) vertex disjoint paths connecting x and y. But, this is clearly not possible since the connection in G' between G_1 and G_2 is only via u and or u_1. Next assume without loss of generality that x, y, z and v are all vertices of G_1. Since G_1 has no K_4 homeomorph, it follows (without loss of generality) that there is a path, $P(x, y)$, in H connecting x and y, which contains a subpath $P(u, u_1)$, consisting of no edges of G_1. Replacing $P(u, u_1)$ by the edge (u, u_1) of G_1 we obtain from H a K_4 homeomorph H', which is contained in G_1. This is impossible since G_1 has no K_4 homeomorph.

Remark 3.3. Since the biconnected components of a graph can be obtained in linear time, [TAR], the augmentation process described in the proof of Theorem 3.2 can also be performed in linear time.

SPM's can be constructed recursively as follows. Each is associated with two vertices called *terminals*.

(1) A single edge is SPM. Its end vertices are the two terminals.

(2) If G_1 and G_2 are SPM's with terminals x, y and z, w respectively, so are the multigraphs obtained by each of the following compositions.

(2.1.) *Series composition.* Identify a terminal of G_1, say x, with a terminal of G_2, say z. The terminals of the new multigraph are y and w.

(2.2.) *Parallel composition.* Identify one terminal of G_1, say x, with one terminal of G_2, say z, and the second terminal of G_1, y, with w. The terminals of the new multigraph are the two vertices where the identifications occur.

The above recursive construction of an SPM G is represented by a *binary decomposition tree* [VTL]. 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 leaf (tip) of T represents a distinct edge of G. If q is not a leaf then it is either of a series (S) or parallel (P) type. Let q_1 and q_2 be its two sons. $G(q)$ is the submultigraph obtained from $G(q_1)$ and $G(q_2)$ by the respective series or parallel composition. In particular, if r is the root of T $G(r)$ is the multigraph G. Figure 5 of [VTL] illustrates a binary decomposition tree of an SPM. We note that a linear time algorithm that finds the binary decmposition tree of a given SPM is presented in [VTL].

Remark 3.4. Given a SPM G, a binary decomposition tree associates two terminals with G. Suppose that G is also biconnected. If (u, v) is an edge of G it follows [DUF, Corollary 4] that there exists a binary decomposition tree of G that induces the vertices u and v as the two terminals of G. (This tree can be found in linear time by the algorithm of [VTL] that constructs a binary decomposition tree for a directed series-parallel multigraph.)

In the following sections we will utilize the binary decomposition tree representation of a SPM to produce efficient algorithms for several combinatorial problems defined on series-parallel graphs.

4. Finding a 2-separator of a graph with no K_4 homeomorph. In this section we assume that $G=(V, E)$ is a graph, i.e., it has no parallel edges. Suppose that G has n vertices and $n>2$. A set of vertices $X \subseteq V$ is a *k-separator* of G if the following conditions are met: There exist subsets of vertices V_1 and V_2 such that V_1, V_2, X are pairwise disjoint, $V_1 \cup V_2 \cup X = V$, there is no edge $(u, v) \in E$ such that $u \in V_1$ and $v \in V_2$, and $|X| = k$, $|V_i| \leqq \frac{2}{3}n$, $i=1, 2$.

It is well known [KH] that if G is a tree it has a 1-separator. (For example, the centroid of a tree, [GOL], [KH], is a 1-separator.) We will extend this result and prove that a graph that does not contain a K_4 homeomorph has a 2-separator. We will also exhibit a linear time algorithm for finding a 2-separator.

Using Theorem 3.2 we restrict our analysis to biconnected series-parallel graphs. These graphs are planar and have at most $3n-6$ edges. Given such a graph let T be its binary decomposition tree. As noted above each leaf of T corresponds to a distinct edge of T. Thus, T has $m \leqq 3n-6$ leaves and $m-1$ vertices which are of degree greater than one.

Since G is biconnected and $n>2$, there exists a one-to-one mapping of the vertices of G to the set of edges. Such a mapping can be constructed, for example, by considering some spanning tree of G, say $T(G)$. Then, root $T(G)$ at a leaf of $T(G)$, make it an out-tree, and map each vertex to the edge directed into it and the root into an incident edge not included in $T(G)$. Therefore, we now suppose that each vertex of G corresponds to a distinct leaf of T. (Note that when $m>n$ certain leaves of T are not assigned a vertex of G). Each leaf of T which is assigned a vertex of G is given a unit weight while all other vertices of T have zero weight. Call the leaves with unit weight *weighted leaves.* In linear time, [GOL], [KH], we can find a vertex of T (called a *weighted centroid*), say q, such that each of the (at most 3) connnected components of T, obtained by removing q from T, contains at most $n/2$ weighted leaves. Since $n>2$ q itself is not a leaf and it corresponds to a series-parallel subgraph of G, say $G(q)$. See Fig. 1.

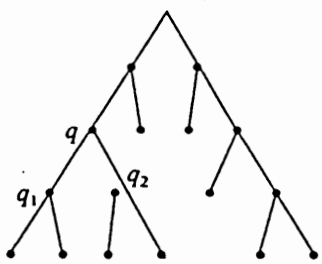

FIG. 1

Let q_1 and q_2 be the two sons of q on T corresponding to the SP subgraphs $G(q_1)$ and $G(q_2)$ respectively. Let u_i, v_i denote the two terminals of $G(q_i)$, $i=1,2$. Let V_i denote the vertex set of $G(q_i)$, $i=1,2$.

Denote by T_i, $i=1,2$, the connected component of T which is obtained by removing q, and contains q_i. T_i contains at most $n/2$ weighted leaves, and $T_1 \cup T_2$ contains at least $n/2$ weighted leaves. Since each weighted leaf in $T_1 \cup T_2$ corresponds to a distinct vertex in $V_1 \cup V_2$, it follows that $|V_1 \cup V_2| \geqq n/2$. Also, each vertex in V_i, $i=1,2$, which is not a terminal corresponds to a distinct weighted leaf in T_i. Thus $|V_i| \leqq n/2+2$, $i=1,2$.

Define $V_3 = V-(V_1 \cup V_2)$. Then $|V_3| \leqq n/2$.

Suppose first that $|V_3| = \max\{|V_1|, |V_2|, |V_3|\}$. Then from $|V_1|+|V_2|+|V_3| \geqq n$ we obtain $n/3 \leqq |V_3| \leqq n/2$. There is no edge of G connecting a vertex in V_3 with a vertex of $G(q)$ which is not one of the two terminals of $G(q)$. (Recall that $G(q)$ is the composition of $G(q_1)$ and $G(q_2)$.) Thus, the two terminals of $G(q)$ constitute a 2-separator of G. It separates V_3 from the remaining vertices of V, i.e., $V-V_3$. Clearly $|V-V_3| \leqq \frac{2}{3}n$ since $|V_3| \geqq n/3$.

Next assume, without loss of generality, that $|V_1| = \max\{|V_1|, |V_2|, |V_3|\}$. Then $n/3 \leqq |V_1| \leqq n/2+2$. There is no edge of G that connects a vertex which is not a terminal of $G(q_1)$ with a vertex in $V-V_1$. Therefore the two terminals of $G(q_1)$ separate the nonterminal vertices of G_1 from $V-V_1$. We have $|V-V_1| \leqq \frac{2}{3}n$ since $|V_1| \geqq n/3$. Also the number of nonterminal vertices of G_1 is $|V_1|-2 \leqq n/2$.

The complexity of the above scheme for locating the 2-separator is governed by the complexity of constructing T, the binary decomposition tree of G, and finding a weighted centroid on it. As mentioned above both take linear time.

We have proved the following.

THEOREM 4.1. *Let $G=(V, E)$ be an n-vertex graph that does not contain K_4 as a homeomorph. There exist two vertices u, v in V and two subsets V_1, $V_2 \subseteq V$, $|V_i| \leqq \frac{2}{3}n$, $i=1,2$, such that $V_1 \cup V_2 \cup \{u, v\} = V$, and there is no edge of G connecting a vertex in V_1 with a vertex in V_2. Furthermore, the vertices, u, v and the sets V_1 and V_2 can be found in linear time.*

Remark 4.2. The above theorem can easily be extended to the weighted version where each vertex v in V is associated with a positive weight w_v. In this case the condition $|V_i| \leqq \frac{2}{3}n$ is replaced by $\sum_{v \in V_i} w_v \leqq \frac{2}{3}\sum_{v \in V} w_v$.

Remark 4.3. From the above discussion it follows that the 2-separators which we find consist of a pair of vertices $\{u, v\}$ that are terminals of some series-parallel subgraph. Therefore if we add the edge (u, v) to the underlying graph $G=(V, E)$ the resulting graph will also not contain K_4 as a homeomorph.

5. Linear time algorithms for finding shortest paths on series-parallel graphs. We have already noted above that a binary decomposition tree of a series-parallel graph

can be generated in linear time [VTL]. With the aid of this tree one can design efficient algorithms for solving a variety of combinatorial problems defined on the underlying series-parallel graph. For example, linear time algorithms for finding a minimal vertex cover, a minimal dominating set or a Hamiltonian cycle can easily be constructed, utilizing the decomposition tree.

In this section we demonstrate the use of the decomposition tree to solve edge-weighted problems on graphs that do not contain K_4 as a homeomorph. Although we will focus on the problem of finding shortest (simple) paths, the approach can easily be modified to solve other problems like the matching problem and the travelling salesman problem. In particular, since the dual of a series-parallel graph is also series-parallel, the method described in [HAS] combined with the algorithm of this section yields a linear time algorithm for the maximum flow problem on series-parallel graphs.

To this end let $G=(V, E)$ be a connected directed graph with $|V|=n$, that does not contain K_4 as a homeomorph. Each (directed) edge e in E has a "length" (weight) w_e (which is not necessarily nonnegative). For each pair of vertices u and v define $d(u, v)$ to be 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 vertices v in V. This problem is recognized as the *single source shortest paths problem*.

If the graph may contain negative cycles (we do not exclude this possibility), the problem is NP-hard for general planar graphs. If $w_e \geqq 0$ for each edge, then this problem is solvable for any n-vertex planar graph in $O(n\sqrt{\log n})$ time, [FRED]. (In particular, this bound is applicable to our class of graphs, since these graphs are planar.) However, we will show that the single source shortest paths problem is solvable in linear time for graphs with no K_4 homeomorph.

Without loss of generality we assume that G is biconnected and series-parallel, since otherwise we may apply the procedure described in the proof of Theorem 3.2, and add to G edges with infinite length. We consider (as noted in Remark 3.4), a decomposition tree T, that induces the vertex s (from which the distances $d(s, v)$, $v \in V$, are sought for) as one of the terminals of G.

We now give a brief overview of the algorithm, which consists of two phases. In the first phase we start with the leaves of T and recursively compute for each subgraph of G, $G(q)$, (that corresponds to a vertex q of T), the distances, restricted to $G(q)$, $d(q, i(q), j(q))$ and $d(q, j(q), i(q))$ between its two terminals $i(q)$ and $j(q)$. The recursive equations are rather straightforward and therefore they are omitted.

In the second phase we start with the root of the tree (corresponding to the graph G), and recursively compute the distances in G from s to the terminals of each subgraph $G(q)$. Note that each vertex of G is a terminal of at least some subgraph $G(q)$. Therefore, at the end of the second phase we will have the distances $d(s, v)$ for all vertices in V. The second phase uses the following auxiliary distance function. If x is a terminal of $G(q)$, let $d'(q, s, x)$ denote the length of a directed shortest simple path from s to x that does not contain any edge of $G(q)$. Starting with the root of T, we recursively compute $d'(q, s, x)$ and $d(s, x)$. The recursive formulae are again quite clear. For example, suppose that $G(q)$ is generated from $G(q_1)$ and $G(q_2)$ by a series composition with, say, $i(q)=i(q_1)$ and $j(q)=j(q_2)$, (i.e., $j(q_1)=i(q_2)$). Then,

$$d'(q_1, s, i(q_1)) = d'(q, s, i(q)),$$

$$d'(q_1, s, j(q_1)) = d'(q, s, j(q)) + d(q_2, j(q_2), i(q_2)),$$

$$d(s, i(q_1)) = d(s, i(q)),$$

and

$$d(s, j(q_1)) = \text{Min}\,[(d'(q, s, i(q)) + d(q_1, i(q_1), j(q_1))),$$
$$(d'(q, s, j(q)) + d(q_2, j(q_2), i(q_2)))].$$

For the benefit of brevity we omit the recursive equations for the other cases.

We claim that the above algorithm solves the single source shortest paths problem in linear time. The linear complexity follows from the facts that the decomposition tree has $O(n)$ vertices, and that each recursive equation used in either one of the two phases can be computed in constant time.

6. The uncapacitated plant location problem. The uncapacitated (or simple) plant location problem is defined on a n-vertex undirected graph $G = (V, E)$. Each edge e in E is assumed to have a positive length w_e. Each vertex v in V is associated with a fixed cost c_v (the set up cost of a facility at v), and a transportation cost function $f_v(\cdot)$, which is a nondecreasing function of its argument. Each vertex is viewed as both a potential site for a service center (facility) as well as a customer. The uncapacitated plant location problem is to locate facilities at vertices so as to minimize the sum of the setup costs and the transportation costs. Since the transportation cost functions are nondecreasing each customer will use the closest facility to him. Formally the problem is to find a subset of vertices $S \subseteq V$ that minimizes

$$\sum_{v \in S} c_v + \sum_{v \in V} f_v(d(v, S)),$$

where

$$d(v, S) = \text{Min}_{u \in S}\, d(v, u).$$

This problem is NP-hard on general planar graphs. However, $O(n^2)$ algorithms are given in [KT], for the case when G is a tree. In this section we will present an $O(n^4)$ algorithm solving the problem for graphs containing no K_4 as a homeomorph.

As in the previous section, the procedure will be based on a recursive approach on the corresponding binary decomposition tree.

Since we can add to G edges with infinite length, we use Theorem 3.2 and assume without loss of generality that G is a biconnected series-parallel graph and T is its binary decomposition tree. We will use the terminology that customer v is served by facility u if u is the closest established facility to v.

Let q be a vertex of T, $G(q)$ its respective series-parallel subgraph of G and $i(q)$ and $j(q)$ the two terminals of $G(q)$. For each pair of vertices u, v in V we define $h(q, u, v)$ to be the solution value of the uncapacitated plant location problem restricted to the subgraph $G(q)$, with the constraint that $i(q)$ is served by u and $j(q)$ is served by v. u and v are not restricted to be in $G(q)$. Formally, if we let $V(q)$ denote the vertex set of $G(q)$, $h(q, u, v)$ is defined by

$$h(q, u, v) = \text{Min}_{S \subseteq V} \left\{ \sum_{w \in S} c_w + \sum_{w \in V(q)} f_w(d(w, S)) \right\}$$
$$\text{s.t. } i(q) \text{ is served by } u \text{ and } j(q) \text{ is served by } v.$$

The solution to the uncapacitated plant location problem on G is, therefore, given by $\text{Min}_{u,v \in V} h(r, u, v)$, where r is the root of the decomposition tree. The functions $h(q, \cdot, \cdot)$ will be evaluated, recursively, starting with the leaves of T. Suppose that $G(q)$ is a composition of $G(q_1)$ and $G(q_2)$. For a series composition with, say, $i(q) = i(q_1)$,

$j(q)=j(q_2)$, (i.e., $j(q_1)=i(q_1)$) we obtain

$$h(q, u, v)=\min_{w\in V(q)\cup\{u,v\}}\{h(q_1, u, w)+h(q_2, w, v)-f_{j(q_1)}(d(j(q_1), w))-c_w-c_u\delta(u, v, w)\},$$

where $\delta(u, v, w)$ is equal to 1 if $u=v=w$ and 0 otherwise. For a parallel composition with $i(q)=i(q_1)=i(q_2)$ and $j(q)=j(q_1)=j(q_2)$ we have $h(q, u, v)=h(q_1, u, v)+h(q_2, u, v)-f_{i(q)}(d(i(q), u))-f_{j(q)}(d(j(q), v))-c_u-c_v\varepsilon(u, v)$, where $\varepsilon(u, v)$ is equal to 1 if $u\neq v$ and 0 otherwise.

To compute the complexity of this recursive procedure, we assume that it takes a constant time to compute any cost function $f_v(x)$ at a given value of x. To compute these functions we need the distances between all pairs of vertices in G. If we use the algorithm given in § 5, this task can be performed in $O(n^2)$ time. It follows from the above recursive formulae that the complexity bound of the proecure is $O(kn^3)$, where $k\geqq 1$ is the number of vertices of the decomposition tree of the series type.

7. Finding the kth longest distance on series parallel graphs. Let $G=(V, E)$ be an n-vertex directed graph with nonnegative edge-lengths $w_e\geqq 0$, $e\in E$, and consider the (multi) set of intervertex distances

$$S=\{d(u, v)\mid u, v\in V, u\neq v\}.$$

($d(u, v)$ is the length of a shortest simple directed path from u to v.)

The cardinality of S is $O(n^2)$, and therefore if S is given explicitly, the kth largest element in S can be found in $O(n^2)$ time by a standard algorithm [AHU]. For general graphs no algorithm is known for finding even the largest element in S in time which is of a lower order than the time required to compute S explicitly. (The best time bound known is $O(n^3(\log\log n/\log n)^{1/3})$ [FRE].) When G is a tree, S can be given a succinct representation that requires only $O(n\log n)$ space. The construction of this representation takes $O(n\log n)$ time, and it enables the selection of the kth largest element in S in $O(n\log n)$ time, [FJ2], [MTZC], i.e., the time is sublinear in the size of S. (We note that for trees the largest element in S can be found in linear time, [HAN].)

As discussed in [FJ2], [MTZC], the ability to select efficiently in the set S gives rise to efficient algorithms for a variety of center location problems.

In this section we extend the results of [FJ2], [MTZC] and show that if G does not contain K_4 as a homeomorph, the kth largest element in S can be selected in $O(n\log^3 n)$ time, using $O(n\log n)$ space.

The algorithm is a divide-and-conquer scheme, which is based extensively on the linear time algorithms of §§ 4 and 5 to compute 2-separators and distances.

The first phase of the algorithm deals with the succinct representation of S. We partition S into $\bar{m}=O(n\log n)$ subsets such that the k_jth largest element in the jth subset, for all $j=1, \cdots, \bar{m}$, can be found in a total time of $O(\bar{m}\log n)=O(n\log^2 n)$.

The partitioning phase starts by finding a 2-separator of G consisting of a pair of vertices u, v. Let V_1, V_2 be the two subsets of V separated by u, v (Theorem 4.1). Without loss of generality we assume that the three sets V_1 and V_2 and $\{u, v\}$ are mutually disjoint.

At this stage we partition only the subsets of S corresponding to distances from vertices in V_1 to vertices in V_2 and from vertices in V_2 to vertices in V_1. (We then proceed recursively by decomposing $V_1\cup\{u, v\}$ and $V_2\cup\{u, v\}$.)

For any pair of vertices $x\in V_1$ and $y\in V_2$, any simple path connecting the two must contain at least one of the separators u and v. Moreover, since all edge lengths

are nonnegative, we have

$$d(x, y) = \text{Min}\{d(x, u) + d(u, y), d(x, v) + d(v, y)\},$$
$$d(y, x) = \text{Min}\{d(y, u) + d(u, x), d(y, v) + d(v, x)\}.$$

For any disjoint subsets of vertices $X, Y \subseteq V$ we will define

$$S(X, Y) = \{d(x, y) : x \in X, y \in Y\}.$$

Focusing first on $S(V_1, V_2)$ we partition it to $S^-(V_1, V_2)$ and $S^+(V_1, V_2)$, where

$$S^-(V_1, V_2) = \{d(x, u) + d(u, y) | x \in V_1, y \in V_2, d(x, u) - d(x, v) + d(u, y) - d(v, y) \leqq 0\},$$
$$S^+(V_1, V_2) = \{d(x, v) + d(v, y) | x \in V_1, y \in V_2, d(x, u) - d(x, v) + d(u, y) - d(v, y) > 0\}.$$

Defining $a_x = d(x, u)$, $c_x = d(x, u) - d(x, v)$, $x \in V_1$ and $b_y = d(u, y)$, $d_y = d(u, y) - d(v, y)$, $y \in V_2$, we note that $S^-(V_1, V_2)$ (and similarly $S^+(V_1, V_2)$) are both of the nature of the sets described in Appendix 1. The sequences $\{a_x\}$, $\{c_x\}$, $x \in V_1$, and $\{b_y\}$, $\{d_y\}$, $y \in V_2$ can be constructed in linear time if we find the 2-separator $\{u, v\}$ by the algorithm in § 4 and the distances from (and to) u and v by the algorithm in § 5.

Using Appendix 1 and noting that $|V_i| = O(n)$, $i = 1, 2$, we conclude that $S(V_1, V_2)$ can be partitioned into $m' = O(n)$ lists and stored in a data structure (of $O(n)$ space), on which we can select the k_jth largest element in the jth list for all $j = 1, \cdots, m'$ in a total time of $O(n \log n)$.

Clearly the same partitioning will be applied to the set $S(V_2, V_1)$. From Appendix 1 we see that the complexity bound of this stage of the recursion is $O(n \log n)$.

We now have to proceed with the distances within the set of vertices $V_1 \cup \{u, v\}$ (and similarly the distances within $V_2 \cup \{u, v\}$).

Let G_i, $i = 1, 2$ denote the subgraph of G induced by the vertices $V_i \cup \{u, v\}$. To compute the distances within, say, $V_1 \cup \{u, v\}$, we may not restrict our discusion to G_1, since the shortest path connecting u to v (or v to u) may not be included in G_1. However, this difficulty can easily be resolved by adding to G_i, $i = 1, 2$, the (directed) edges (u, v) and (v, u) with edge-lengths $d(u, v)$ and $d(v, u)$ respectively. (Note that $d(u, v)$ and $d(v, u)$ have already been computed earlier). To proceed recursively with G_i, $i = 1, 2$, we need the property that G_i, $i = 1, 2$, with the above augmented edges does not contain K_4 as a homeomorph. Indeed, this is guaranteed by Remark 4.3. To make sure that each pair of vertices is taken into account precisely once, when we deal with the set of distances within $V_2 \cup \{u, v\}$ we will not record the distances between u and v.

We now estimate the total number of lists created during the entire partitioning process. Let $f(n)$, denote the maximum number of such lists created in such a partition of a graph with n vertices. Then

$$f(n) \leqq c_1 n + f(n_1) + f(n_2),$$

where $n_i = |V_i \cup \{u, v\}| \leqq \frac{2}{3}n + 2$, $i = 1, 2$, and $n_1 + n_2 = n + 2$. Therefore, $f(n) = O(n \log n)$.

To estimate the total space needed for the process, $g(n)$, we note that $g(n)$ satisfies

$$g(n) \leqq c_2 n + g(n_1) + g(n_2)$$

with n_i as above, and thus $g(n) = O(n \log n)$. Finally, the running time, $T(n)$, of the partitioning phase satisfies

$$T(n) \leqq c_3 n \log n + T(n_1) + T(n_2)$$

where $n_i \leqq \frac{2}{3}n + 2$ $i = 1, 2$, and $n_1 + n_2 = n + 2$. It thus follows that $T(n) = O(n \log^2 n)$.

Next we turn to the selection phase. At this point S is represented by $\bar{m} = O(n \log n)$ lists (subsets), say, $S_1, \cdots, S_{\bar{m}}$. These lists are maintained in a data structure on which the total time to select the k_jth largest element from S_j for all $j = 1, \cdots, \bar{m}$ is $O(\bar{m} \log n)$, (Appendix 1). Since each list is of cardinality $O(n)$, we can use the algorithm of [FJ1] to select the k-th largest element (i.e., kth longest distance) in S in $O(n \log^3 n)$ time. (Note that the time bound $O(\bar{m} \log n)$ mentioned in [FJ1] is based on the assumption that the total time to select the k_jth largest element from S_j for all $j = 1, \cdots, \bar{m}$ is $O(\bar{m})$. Since in our case the latter is replaced by $O(\bar{m} \log n)$, our total time bound is $O(\bar{m} \log^2 n) = O(n \log^3 n)$.)

Finally we note that if we only wish to find the largest element in S, i.e., the *diameter* of the graph, we do not need the first phase. In fact, the diameter can be found recursively by finding at each iteration, the largest element in $S(V_1, V_2) \cup S(V_2, V_1)$. The latter can be found in $O(n)$ time, thus yielding an $O(n \log n)$ total time for the scheme that finds the diameter.

Appendix 1. Representation of and selection in the set $R = \{a_i + b_j | c_i + d_j \leqq 0,\ i = 1, \cdots, n,\ j = 1, \cdots, m\}$. Suppose that we are given four sequences of real numbers, $\{a_i\}$, $i = 1, \cdots, n$, $\{c_i\}$, $i = 1, \cdots, n$, $\{b_j\}$, $j = 1, \cdots, m$, $\{d_j\}$, $j = 1, \cdots, m$. Define the (multi-) set R by

$$R = \{a_i + b_j | c_i + d_j \leqq 0,\ i = 1, \cdots, n, j = 1, \cdots, m\}. \tag{A.1}$$

Our goal is to provide a partition of the multiset R, (whose cardinality is $O(mn)$) into m lists $R'_1, \cdots, R'_m$, and a data structure (requiring a total space of $O(m+n)$), for storing these lists, on which the following operations can be implemented: Given integers $k_1, \cdots, k_m$, select the k_jth largest element in R'_j for all $j = 1, \cdots, m$, in $O((m+n) \log n)$ total time. (The total time consumed in constructing this data structure will be shown to be $O(m \log m + n \log n)$.)

For each j, $j = 1, \cdots, m$, define the set $R_j = \{a_i | c_i + d_j \leqq 0,\ i = 1, \cdots, n\}$. Sort the sequences $\{c_i\}$, $i = 1, \cdots, n$, and $\{d_j\}$, $j = 1, \cdots, m$, and assume without loss of generality that $c_1 \leqq c_2 \cdots \leqq c_n$, and $d_1 \leqq d_2 \leqq \cdots \leqq d_m$. Next, define $I(j) = \{i | c_i + d_j \leqq 0\}$, $j = 1, \cdots, m$, to note that

$$I(1) \supseteq I(2) \supseteq \cdots \supseteq I(m). \tag{A.2}$$

Rewriting R_j, $j = 1, \cdots, m$, as $R_j = \{a_i | i \in I(j)\}$, define h_j, $j = 1, \cdots, m$, to be the difference between adjacent sets, i.e.,

$$h_j = |I(j)| - |I(j+1)|.$$

The subset R_{j+1} is obtained from R_j by deletion of h_j elements. Thus using a 2-3 tree data structure, [AHU], we can delete the respective elements, and find the k_jth largest element in R_j for all $j = 1, \cdots, m$, in a total time of $O((\sum_{j=1}^{m-1} h_j) \log n + m \log n)$. By (A.2) this bound is $O((m+n) \log n)$.

The total effort to construct this data structure is dominated by the sorting of the sequences, $\{c_i\}$, $i = 1, \cdots, n$ and $\{d_j\}$, $j = 1, \cdots, m$, i.e., $O(n \log n + m \log m)$.

The multiset R, defined by (A.1), can now be represented as the union of the following m subsets, $R'_1, \cdots, R'_m$, where $R'_j = \{b_j\} + R_j$, $j = 1, \cdots, m$. Using the above we conclude that the k_jth largest element in R'_j for all $j = 1, \cdots, m$, can be found in $O((m+n) \log n)$ total time by selecting the elements in R_j, $j = 1, \cdots, m$ and adding the appropriate b_j value.

REFERENCES

[AHU] A. V. AHO, J. E. HOPCROFT AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[BBT] W. W. BEIN, P. BRUCKER AND A. TAMIR, *Minimum cost flow algorithms for series parallel networks*, Discrete Appl. Math., 10 (1985), pp. 117–124.

[COL] R. COLE, *Searching and storing similar lists*, Technical Report 88, Dept. Computer Science, Courant Institute of Mathematical Sciences, New York Univ. New York, Oct., 1983.

[DIR] G. A. DIRAC, *A property of 4-chromatic graphs and some remarks on critical graphs*, J. London Math. Soc., 17 (1952), pp. 85–92.

[DUF] R. J. DUFFIN, *Topology of series parallel graphs*, J. Math. Anal. Appl., 10 (1965), pp. 303–318.

[FRE] M. FREDMAN, *New bounds on the complexity of the shortest path problem*, SIAM J. Comput., 5 (1976), pp. 83–89.

[FRED] G. N. FREDERICKSON, *Shortest path problems in planar graphs*, in Proc. 24th IEEE Symposium on Foundations of Computer Science, Tucson, AZ, Nov. 1983, pp. 242–247.

[FJ1] G. N. FREDERICKSON AND D. B. JOHNSON, *The complexity of selection and ranking in $X + Y$ and matrices with sorted columns*, J. Comput. System Sci., 24 (1982), pp. 197–208.

[FJ2] ———, *Finding kth paths and p-centers by generating and searching good data structures*, J. Algorithms, 4 (1983), pp. 61–80.

[GOL] A. J. GOLDMAN, *Optimal center location in simple networks*, Transport. Sci., 5 (1971), pp. 212–221.

[HAN] G. Y. HANDLER, *Minimax location of a facility in an undirected tree graph*, Transport. Sci., 7 (1973), pp. 287–293.

[HAS] R. HASSIN, *Maximum flow in (s, t) planar networks*, Inform. Proc. Letters, 13 (1981), p. 107.

[KH] O. KARIV AND S. L. HAKIMI, *An algorithmic approach to network location problems, Part I. The p-centers*, SIAM J. Appl. Math., 37 (1979), pp. 513–538.

[KT] A. KOLEN AND A. TAMIR, *Covering problems*, to appear in Discrete Location Theory, L. R. Francis and P. B. Mirchandani, eds., John Wiley, New York, 1986.

[MTZC] N. MEGIDDO, A. TAMIR, E. ZEMEL AND R. CHANDRASEKARAN, *An $O(n \log^2 n)$ algorithm for the kth longest path in a tree with applications to location problems*, SIAM J. Comput., 10 (1981), pp. 328–337.

[RS1] N. ROBERTSON AND P. D. SEYMOUR, *Graph minors* I: *Excluding a forest*, J. Comb. Theory B, 35 (1983), pp. 39–61.

[RS2] ———, *Graph minors* II: *Algorithmic aspects of tree width*, submitted for publication.

[ST] G. STEINER, *A compact labeling scheme for series parallel graphs*, Discrete Appl. Math., 11 (1985), pp. 281–297.

[TAR] R. E. TARJAN, *Depth-first search and linear graph algorithms*, SIAM J. Comput., 1 (1972), pp. 146–160.

[TNN] N. TAKAMIZAWA, T. NISHIZEKI AND N. SAITO, *Linear-time computability of combinatorial problems on series parallel graphs*, J. ACM, 29 (1982), pp. 623–641.

[VTL] J. VALDES, R. E. TARJAN AND E. L. LAWLER, *The recognition of series parallel diagraphs*, SIAM J. Comput., 11 (1982), pp. 298–313.