Solving NP-Hard Problems on Graphs That Are Almost Trees and an Application to Facility Location Problems

YURI GUREVICH

University of Michigan, Ann Arbor, Michigan

AND

LARRY STOCKMEYER AND UZI VISHKIN

IBM Thomas J Watson Research Center, Yorktown Heights, New York

Abstract. A general technique is described for solving certain NP-hard graph problems in time that is exponential in a parameter k defined as the maximum, over all nonseparable components C of the graph, of the number of edges that must be added to a tree to produce C; for a connected graph, k is no more than the number of edges of the graph minus the number of vertices plus one. The technique is illustrated in detail for the following facility location problem: Given a connected graph $G(V, E)$ such that each edge has an associated positive integer length and given a positive integer r, place the minimum number of centers on points of the graph such that every point of the graph is within distance r from some center (a "point" is either a vertex or a point on some edge). An algorithm of time complexity $O(|E| \cdot (6r)^{\lceil k/2 \rceil})$ is given. A parallel implementation of the algorithm, with optimal speedup over the sequential version for a fairly wide range for the number of processors, is presented.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*computations on discrete structures*; G.2.2 [**Discrete Mathematics**]: Graph Theory—*graph algorithms; network problems*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Polynomial time algorithms, NP-complete problems, location problems, parallel algorithms

1. *Introduction*

Given a computational problem, a general practice in the theory of algorithms is to classify the input domain for the problem into subdomains on the basis of the complexity of solving the problem when restricted to the subdomain. A typical

The research of the first author was done in part at the IBM Thomas J. Watson Research Center while visiting from the Mathematics Department, Ben-Gurion University, Beer-Sheva, Israel. The research of the third author was done in part at the IBM Thomas J. Watson Research Center as a World Trade Visiting Scientist from the Department of Computer Science, Technion, Haifa, Israel. The general outline of the algorithm was achieved by all three authors. The time bound was later improved by the second two authors. The idea of using k as a parameter in bounding the running time is due to the third author.

Authors' addresses: Y. Gurevich, Department of Computer and Communication Sciences, University of Michigan, Ann Arbor, MI 48109; L. Stockmeyer, IBM Research Lab. K51/281, 5600 Cottle Road, San Jose, CA 95193; and U. Vishkin, Department of Computer Science, Courant Institute of Mathematical Sciences, 251 Mercer Street, New York, NY 10012.

situation is that the problem is NP-complete for some subdomains and has efficient algorithms for others. However, it is possible, as we demonstrate here, that there are intermediate domains for which we are still able to give relatively efficient algorithms. The pragmatic advantage of extending the theory to domains that do not obviously have efficient algorithms in the narrow sense (polynomial time in the length of the input) should be evident. Designers of algorithms need a library of algorithms and algorithmic methods that cover the widest possible domains. Several NP-complete graph problems, including vertex cover, dominating set, and various facility location problems, can be solved in polynomial time for trees but are NP-complete or NP-hard even for planar graphs of degree 3. This suggests that we might extend the "feasible" input domain from trees to slightly denser graphs. The obvious enumeration algorithms for NP-complete graph problems involve an exponential term where the parameter appearing in the exponent is typically the number of vertices or edges. If the problem can be solved in polynomial time for trees, a first goal could be to replace this critical parameter by the number of edges that must be added to a tree to produce the graph. For a connected graph with n vertices and m edges, this number of *additional edges* is $m - n + 1$. The main purpose of this paper is to obtain this goal, improve it, and illustrate it in detail for a particular facility location problem.

One improvement is that the parameter in the exponent can be reduced from the number of additional edges to a parameter $k(G)$ defined as follows:

> Find the maximal nonseparable components $C_1, \ldots, C_t$ of the graph G; for each C_i, let k_i be the number of edges that must be added to a tree to produce C_i (i.e., k_i is the number of edges of C_i minus the number of vertices of C_i plus one); then $k(G)$ is the maximum of k_i for $1 \leq i \leq t$.

Certainly $k(G) \leq$ the number of additional edges of G, and $k(G)$ can be much smaller if the additional edges are spread over several nonseparable components. The idea behind this improvement is that two nonseparable components can communicate through at most one vertex (an articulation vertex) so that choices of solutions made in different components combine additively rather than multiplicatively. Essentially the same principle was used by Lipton and Tarjan [12]. By applying the fact that any planar graph can be separated by $O(\sqrt{n})$ vertices into two pieces of roughly equal size [13], they show that typical NP-complete graph problems can be solved in time $O(\exp(\sqrt{n}))$ for planar graphs.

A related message appears in Garey and Johnson's book on NP-complete problems [8], where, for example, it is shown that the NP-complete partition problem (given n positive integers $a_1, a_2, \ldots, a_n$, partition them into two sets A and B such that the sum of the numbers in A equals the sum of the numbers in B) can be solved in time that is a polynomial in n multiplied by an exponential in the number of bits needed to represent the largest a_i. The message there is that although it is convenient in theory to express the size of a problem instance as a single number, the size of an instance is sometimes more accurately expressed in terms of several independent parameters, and it may be useful to know the running time of an algorithm as a function of more than one parameter.

We choose a particular facility location problem as our main example because the way that two adjacent nonseparable components communicate through an articulation vertex is somewhat involved (e.g., more complicated than in the case of the vertex cover problem) and additional ideas are needed to handle the communication cleanly. "Facility location" problems take place on an undirected

graph $G(V, E)$. Each edge $(u, v) \in E$ has a given *length* $l(u, v)$, a positive integer. We identify the edge (u, v) with a line segment of length $l(u, v)$ in such a way we can talk about *points* on (u, v) at distance x from vertex u and distance $l(u, v) - x$ from vertex v for all real x with $0 \leq x \leq l(u, v)$. Let $P(G)$ denote the set of all such points of G. For $v, \xi \in P(G)$, $d(v, \xi)$ denotes the length of a shortest path from v to ξ. Given a graph G with edge lengths and a positive integer r, the *continuous location problem* is to find the minimum number p and find points $\gamma_1, \ldots, \gamma_p \in P(G)$ such that for every $\xi \in P(G)$ there is a γ_i such that $d(\xi, \gamma_i) \leq r$. Intuitively, we wish to locate the minimum number of "centers" such that every point is within distance r from some center. One motivation is to place the minimum number of emergency facilities on a network of roads such that every point on a road is within a given distance from some facility. Depending on the motivation, another variation of the location problem can be defined by requiring that only vertices must be within distance r from centers. We prefer to describe our algorithm for the continuous location problem as defined above since this case entails complications that do not arise from the other variation. In the final section we outline how our algorithm could be modified to handle the other variation. There is an extensive literature on algorithms for solving these location problems. However, most of the existing work deals only with two extremes of the complexity of the problem. At one extreme, if G is a tree, these location problems can be solved in polynomial time (e.g., see [3, 7, 11, 15]). In particular, Chandrasekaran and Daughety [3] give a time $O(n)$ algorithm for the continuous location problem where n is the number of vertices in the tree. At the other extreme, for general graphs, integer programming techniques have been applied (e.g., see [4, 9]). In general, it is unlikely that these algorithms can find optimal solutions in polynomial time since the location problems are NP-hard even when restricted to planar graphs of degree 3 with unit edge lengths. This is proved by Kariv and Hakimi [11] for the variation in which only vertices must be close to centers. A similar proof shows that the continuous location problem is NP-hard for planar graphs of degree 3, as the reader can easily verify.

As described above, our purpose is to consider cases between these two extremes, where G is a tree with additional edges. In Section 3, using the general idea outlined above, we give an algorithm for the continuous location problem of time complexity $O(m \cdot (6r)^{\lceil k/2 \rceil})$ where m is the number of edges and $k = k(G)$. In Section 4 we show that parallelism can be efficiently utilized in an implementation of the algorithm. In fact, we get parallel time $O(m \cdot (6r)^{\lceil k/2 \rceil}/p)$ for $p \leq (6r)^{\lceil k/2 \rceil}/k$ processors, which is an optimal speedup over the sequential version. In the final section we conclude by mentioning some related results and open questions. Two related results are that the vertex cover problem and the independent set problem can be solved in time $O(m \cdot 2^{k/2})$, and the dominating set problem can be solved in time $O(m \cdot 4^{k/2})$. Coppersmith and Vishkin [5] have improved the time for vertex cover to $O(m + (k'/k) \cdot 2^{k/3})$ where k' is the number of additional edges for the entire graph.

These results suggest that the parameter k is a reasonable parameter to use in bounding the complexity of certain NP-complete graph problems. Once one has settled on a reasonable parameter, another goal is then to reduce the exponential dependence of the complexity on the parameter; for example, Tarjan and Trojanowski [18] give a time $O(2^{n/3})$ algorithm for the independent set problem, whereas the naive algorithm uses time $O(2^n)$. We have taken a first step toward this goal for the parameter k by giving algorithms with $\lceil k/2 \rceil$ in the exponent, where one might expect to see k. It is also interesting to note that for connected regular graphs

of degree 3 (a class for which these problems remain NP-hard), we have $k \leq n/2 + 1$, which gives time $O(n \cdot 2^{n/4})$ for vertex cover and independent set, therefore improving upon [18] for this restricted class of graphs.

2. *Preliminaries*

Before presenting the algorithm, we first give some definitions and introduce some basic concepts. Let $G(V, E)$ be a given graph with edge lengths, and fix an $r > 0$. We assume that G is connected and has no multiple edges or self-loops. Let $\gamma, \xi \in P(G)$ and let $\mathscr{C}$ be a finite subset of $P(G)$. We say that γ *covers* ξ if $d(\gamma, \xi) \leq r$. $\mathscr{C}$ *covers* ξ if there is a $\gamma \in \mathscr{C}$ such that γ covers ξ. $\mathscr{C}$ is a *cover* of G if $\mathscr{C}$ covers every $\xi \in P(G)$.

A graph $G(V, E)$ is said to have a *separation vertex* s (also called an *articulation vertex*) if there exist vertices u and v, $u \neq s$ and $v \neq s$, such that all the paths connecting u and v pass through s. A graph that has a separation vertex is called *separable*, and one which has none is called *nonseparable*. Let $V' \subseteq V$. The induced subgraph $G'(V', E')$ is called a *nonseparable component* if G' is nonseparable and if for every larger V'', $V' \subset V'' \subseteq V$, the induced subgraph $G''(V'', E'')$ is separable. For brevity, "component" will mean "nonseparable component" in the sequel. Let $C_1, C_2, \ldots, C_t$ be the components of G and let $s_1, s_2, \ldots, s_q$ be its separation vertices. The *superstructure* $\tilde{G}(\tilde{V}, \tilde{E})$ is defined as follows:

$$\tilde{V} = \{C_1, C_2, \ldots, C_t, s_1, s_2, \ldots, s_q\},$$
$$\tilde{E} = \{(C_i, s_j) \mid s_j \text{ is a vertex of } C_i\};$$

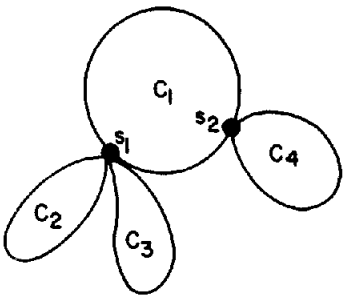
$\tilde{G}(\tilde{V}, \tilde{E})$ is a tree (see Figure 1).

The first phase of our algorithm finds the components, separation vertices, and superstructure of G, using an algorithm of Tarjan [17]. The next goal is to place centers to cover all the components, starting with components that are leaves of the superstructure tree and working up the tree. Suppose we have the situation shown in Figure 2a, where the component C is a leaf of the superstructure, and C is connected to the rest of the graph, G_{rest}, by the separation vertex s. Every point $\xi \in P(C)$ having $d(\xi, s) > r$ must be covered by a center placed on a point of C. A set $\mathscr{C} \subseteq P(C)$ that covers all points $\{\xi \in P(C) \mid d(\xi, s) > r\}$ is called a *partial cover of C with respect to (w.r.t.) s*. Given a partial cover $\mathscr{C}$, let P' be the points of C not covered by $\mathscr{C}$, and define

$$\text{debt}(s, \mathscr{C}) = \begin{cases} \sup\{d(\xi, s) \mid \xi \in P'\} & \text{if } P' \neq \phi, \\ \min\{d(\gamma, s) \mid \gamma \in \mathscr{C}\} - r & \text{otherwise.} \end{cases}$$

When the partial cover $\mathscr{C}$ is clear from context, we write simply debt(s). Note that we always have $-r \leq \text{debt}(s) \leq r$. The reason for the name "debt" should be clear. If debt(s) is positive, there are points of C that must be covered by centers placed outside of C. If debt(s) is negative, there are points of G_{rest} that are covered by centers placed in C. As an aid to the reader's intuition, the effect of a positive debt $b = \text{debt}(s)$ can be simulated by replacing the component C by a new "auxiliary edge" of length b connecting s to a new vertex s' such that no center can be placed on the auxiliary edge (see Figure 2b). The effect of a negative debt $-b = \text{debt}(s)$ can be simulated by replacing C by an auxiliary edge of length $r - b$ connecting s to a new vertex s' and such that there is a center on s' (see Figure 2c).

Our procedures operate recursively on components of a graph. Initially the debts of all vertices are zero, but since a separation vertex could receive a nonzero debt



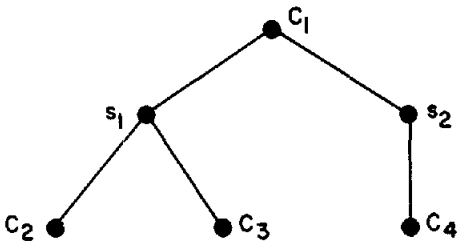
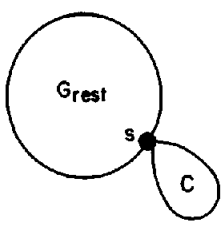
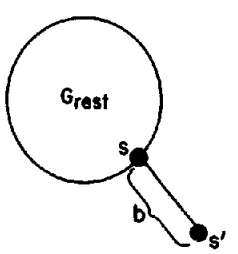


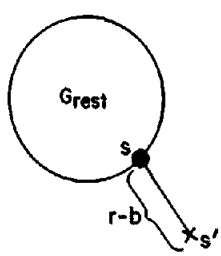
FIG. 1. A graph with four nonseparable components, and its superstructure.



(a)



(b)



(c)

FIG. 2. (a) C is a leaf of the superstructure. (b) The effect of a positive debt b. (c) The effect of a negative debt $-b$; $\times$ denotes the presence of a center on s'.

and later become an internal vertex of some component, we must generalize the definition of partial cover to the case of a component with nonzero debts attached to its vertices. It is also convenient to consider general graphs rather than components. Let G be a graph and let v be a vertex of G. Each vertex u of G has an associated debt(u) in the interval $[-r, r]$. Attach auxiliary edges to all vertices with nonzero debt, as described in the preceding paragraph. Let G_{aux} be $P(G)$ together with all points on auxiliary edges. Let $\mathscr{C}_{\text{aux}}$ be all centers placed on auxiliary edges. A set $\mathscr{C} \subseteq P(G)$ is a *partial cover of* G *w.r.t.* v if $\mathscr{C} \cup \mathscr{C}_{\text{aux}}$ covers every point $\xi \in$

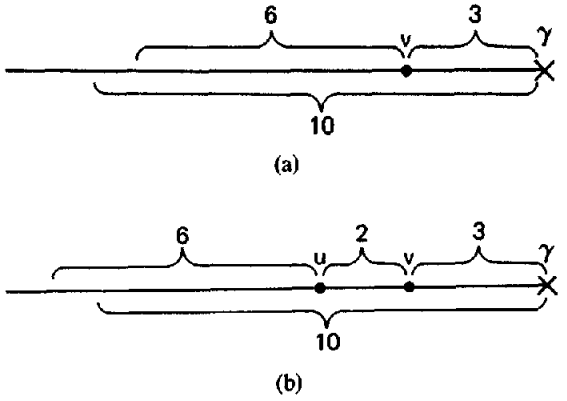


FIG. 3. Illustration of UPDATE. (a) Debt(v) = -7. (b) Debt(v) = 8.

G_{aux} with $d(\xi, v) > r$. Let P' be the points of G_{aux} not covered by $\mathscr{G} \cup \mathscr{G}_{\text{aux}}$. Define

$$\text{debt}(v, \mathscr{G}) = \begin{cases} \sup\{d(\xi, v) \mid \xi \in P'\} & \text{if } P' \neq \phi, \\ \min\{d(\gamma, v) \mid \gamma \in \mathscr{G} \cup \mathscr{G}_{\text{aux}}\} - r & \text{otherwise.} \end{cases}$$

Actually, our algorithm does not compute debts from the definition, it updates debts. Suppose that b_1 is the debt of a vertex at some time during the algorithm, and suppose that the placement of new centers implies that the debt of the vertex should be b_2. The new debt is updated using the following formula.

$$\text{UPDATE}(b_1, b_2) = \begin{cases} b_1 & \text{if } |b_1| > |b_2|, \\ b_1 & \text{if } |b_1| = |b_2| \text{ and } b_1 \leq 0, \\ b_2 & \text{otherwise.} \end{cases}$$

The correctness of UPDATE follows, after a little thought, from the definition of debt. The following example, illustrated in Figure 3, may be helpful. Suppose $r = 10$ and debt(v) = 6 originally. If there is a new center γ such that $d(\gamma, v) = 3$ then it implies a debt of -7 at v. Since $|-7| > |6|$, the center γ also covers all points ξ with $d(\xi, v) \leq 6$ (see Figure 3a). Therefore, we update debt(v) = -7. If at some later time we find a vertex u with $d(u, v) = 2$ and debt(u) = 6, this implies a debt of 8 at v. Since $|8| > |-7|$, this leaves points that are not covered by the debt of -7 (see Figure 3b). Therefore, we update debt(v) = 8.

Define the *cost* of a partial cover $\mathscr{G}$ of G w.r.t. v to be the pair $\langle N, b \rangle$, where N is the cardinality of $\mathscr{G}$ and $b = \text{debt}(v, \mathscr{G})$. We are looking for a partial cover of minimum cardinality, and among those of minimum cardinality we prefer one with minimum debt(v). Formally, we define a lexicographic order between costs as follows: $\langle N_1, b_1 \rangle < \langle N_2, b_2 \rangle$, if either $N_1 < N_2$ or both $N_1 = N_2$ and $b_1 < b_2$. The correctness of this ordering should be obvious if $N_1 = N_2$. If $N_1 < N_2$, then from the partial cover with cost $\langle N_1, b_1 \rangle$ we can obtain another partial cover with cost $\langle N_1 + 1, -r \rangle$ by placing a center on the vertex v. But $N_1 + 1 \leq N_2$ and $-r \leq b_2$, so $\langle N_1, b_1 \rangle$ is no worse than $\langle N_2, b_2 \rangle$.

The final preliminary step is to show that without loss of generality we can assume that centers are placed only on *half-integer* points; that is, points whose distance from a vertex is $z/2$ for some integer $z \geq 0$. The proof of this fact was discovered independently by the authors and by Hang Tong Lau, Brendan D.

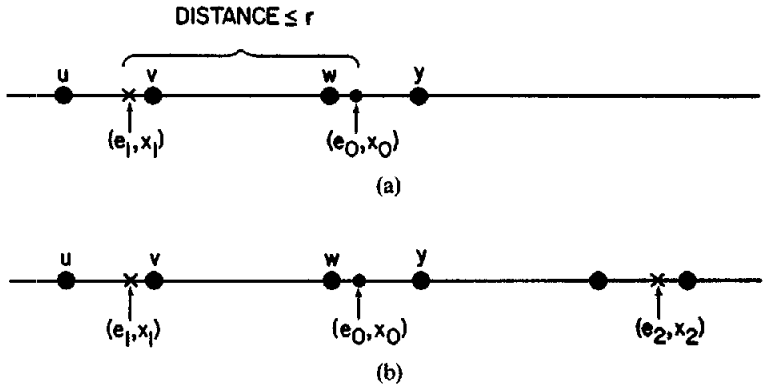


FIG. 4. Illustrating the proof of Lemma 2.1. (a) A path from (e_0, x_0) to (e_1, x_1) of the shortest length that passes through the endpoint v of e_1 and the endpoint w of e_0. (b) An $(e_2, x_2) \in \mathscr{C}$ from which (e_0, x_1) can be reached by a path of length $\leq r$ that passes through y.

McKay, and Ioannis Tollis (personal communication). We give here the proof of Lau, McKay, and Tollis, which is much simpler than our proof. To simplify matters, suppose (without loss of generality, (w.l.o.g)) that all edges are unit length. A point on edge e can be identified by the pair (e, x) where $0 \leq x \leq 1$ is real. (The "origin" endpoint $(e, 0)$ is chosen arbitrarily.)

Given a cover $\mathscr{C}$ of G, form $\mathscr{C}'$ thus: Replace each $(e, x) \in \mathscr{C}$ by (e, x') where

$$x' = \begin{cases} 0 & \text{if } 0 \leq x < \frac{1}{2} \\ \frac{1}{2} & \text{if } x = \frac{1}{2}, \\ 1 & \text{if } \frac{1}{2} < x \leq 1. \end{cases}$$

LEMMA 2.1. *$\mathscr{C}'$ is a cover of G.*

PROOF. Let (e_0, x_0) be an arbitrary point of $P(G)$, and let (e_1, x_1) be a closest point in $\mathscr{C}$ to (e_0, x_0). Consider a path from (e_0, x_0) to (e_1, x_1) of shortest length, and say that this path passes through the endpoint v of e_1 and the endpoint w of e_0 (see Figure 4a).

(1) If $d(v, w) \leq r - 2$, then $d((e_0, x_0), (e_1, x_1')) \leq r$, and we are done.

(2) If $(e_1, x_1) = v$, then $(e_1, x_1) = (e_1, x_1')$, and we are done.

(3) Say then that $d(v, w) = r - 1$ and $(e_1, x_1) \neq v$. Say that v is the origin of e_1 and y is the origin of e_0. In this case, all points (e_0, t) with $0 \leq t < x_1$ cannot be reached from (e_1, x_1) by a path of length $\leq r$ that passes through w. Since (e_1, x_1) is a closest center to (e_0, x_0), there must be an $(e_2, x_2) \in \mathscr{C}$ from which (e_0, x_1) can be reached by a path of length $\leq r$ that passes through y (see Figure 4b). (It is possible that $(e_1, x_1) = (e_2, x_2)$.) Therefore, there is a path ψ of length $\leq 2r$ from (e_1, x_1) to (e_2, x_2) that uses edge e_0. Consider the corresponding path ψ' from (e_1, x_1') to (e_2, x_2'); ψ' uses e_0 also.

Case (i)

$$(0 \leq x_1 < \tfrac{1}{2} \quad \text{or} \quad \tfrac{1}{2} < x_1 \leq 1) \quad \text{and} \quad (0 \leq x_2 < \tfrac{1}{2} \quad \text{or} \quad \tfrac{1}{2} < x_2 \leq 1).$$

$$\text{length}(\psi') < \text{length}(\psi) + \tfrac{1}{2} + \tfrac{1}{2} \leq 2r + 1.$$

$$\text{But length}(\psi') \text{ is integer, so length}(\psi') \leq 2r.$$

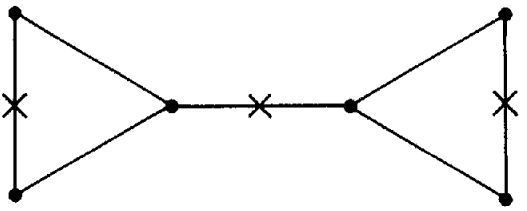


FIG. 5. A graph whose optimal cover requires that centers be placed on half-integer points.

Case (ii)

$$x_1 = \tfrac{1}{2} \quad \text{or} \quad x_2 = \tfrac{1}{2}.$$

$$\text{length}(\psi') < \text{length}(\psi) + \tfrac{1}{2} \leq 2r + \tfrac{1}{2}.$$

But length(ψ') is half-integer, so length(ψ') $\leq 2r$.

In any case, either (e_1, x_1') or (e_2, x_2') has distance $\leq r$ from (e_0, x_0). □

Remark. The use of half-integer points in Lemma 2.1 is necessary, as the example shown in Figure 5 illustrates. If $r = 1$ and all edges have length 1, the minimum cover has three centers and the only way to achieve this minimum is to place the centers on half-integer points as shown in Figure 5.

As a consequence of Lemma 2.1, we can restrict attention to covers and partial covers where all centers are on half-integer points. Similarly, debts take on only half-integer values in the interval $[-r, r]$.

3. *The Algorithm*

In this section we describe an algorithm for the continuous location problem in the case where r and all edge lengths are integers. The worst-case time complexity of the algorithm is $O(m \cdot (6r)^{\lceil k/2 \rceil})$ where m is the number of edges of G and $k = k(G)$ is the parameter defined in the Introduction.

One subroutine SEP(G, v, T) takes a graph G and a vertex v of G and returns G's superstructure as a rooted tree T. Furthermore, if v is a separation vertex of G, then v is the root of T; if v is not a separation vertex, then the root of T is the (unique) component C to which v belongs; in either case we say that v *belongs to the root* of T. Tarjan [17] (see also [1, 6]) shows that SEP can be performed in time $O(m)$ where m is the number of edges of G. By starting the depth-first search at v, we are assured that v belongs to the root of T.

Our algorithm is based on two procedures, GRAPH and COMP, that call each other recursively. GRAPH(G, v, T, $\mathscr{C}$, N) takes a graph G, the superstructure T, and a vertex v that belongs to the root of T, and returns a minimum cost partial cover $\mathscr{C}$ of G w.r.t. v; the number of centers in $\mathscr{C}$ is returned in N. COMP(C, v, $\mathscr{C}$, N) does the same, except that C is a component (so the superstructure is trivial).

The top level procedure for finding a minimum cover of G is

```
TOPLEVEL(G)
  debt(u) ← 0 for all vertices u;
  Let v be an arbitrary vertex;
  SEP(G, v, T);
  GRAPH(G, v, T, 𝒞, N);
  If debt(v) > 0, then (𝒞 ← 𝒞 ∪ {v}; N ← N + 1).
```

Given that GRAPH is correct, it should be clear that TOPLEVEL is correct.

The procedure GRAPH is

```
GRAPH(G, v, T, 𝒢, N)
  𝒢 ← φ;
  N ← 0;
  While T is nonempty do
  begin
    Let C be a leaf component of T;
    If C is the root of T, then s ← v,
      else s ← the father of C in T;
    COMP(C, s, 𝒟, M);
    𝒢 ← 𝒢 ∪ 𝒟;
    N ← N + M;
    Delete C from T;
  end
```

A separation vertex is deleted from T at the time it becomes a leaf of T. Given that COMP is correct, it should be clear that GRAPH is correct.

A concise description of the procedure COMP is

```
COMP(C, s, 𝒢, N)
  If C is an edge (u, s), then EDGE(u, s, 𝒢, N); else
  begin
    δ ← the maximum degree, relative to C, of a vertex in C − {s};
    u ← a vertex of C − {s} with degree δ;
      Let v_1, v_2, ..., v_δ be the vertices of C incident on u;
      Split(C, u) is the graph obtained from C by replacing vertex u by δ vertices
          u_1, u_2, ..., u_δ; the edge that connects u and v_i in C now connects u_i and v_i in
          Split(C, u), for 1 ≤ i ≤ δ;
    SEP(Split(C, u), s, T);
      Let C(x, i) be Split(C, u) with
        debt(u_i) = UPDATE(debt(u), x), and
        debt(u_j) = UPDATE(debt(u), −x) for j ≠ i;
    For each half-integer x with 0 ≤ x ≤ r and
      for each i with 1 ≤ i ≤ δ do
        GRAPH(C(x, i), s, T, 𝒢, N);
    Among the partial covers w.r.t. s returned by GRAPH,
      choose one with minimum cost;
  end
```

In words, the procedure COMP(C, s, $\mathscr{G}$, N) has two cases. If C is an edge (u, s), COMP calls the following procedure EDGE(u, s, $\mathscr{G}$, N). The idea behind EDGE is that we move from u toward s placing centers only when we have to. Thus, the first center is placed at distance $r - \text{debt}(u)$ from u, and thereafter centers are placed at intervals of $2r$. We do not place a center on s since, according to our ordering of costs, we prefer a cost of $\langle N, r\rangle$ to one of $\langle N + 1, -r\rangle$. Procedure EDGE is

```
EDGE(u, s, 𝒢, N)
  N ← ⌈(l(u, s) + debt(u) − r)/2r⌉;
  b ← l(u, s) + debt(u) − 2rN;
  debt(s) ← UPDATE(debt(s), b);
  𝒢 ← { γ | γ is on the edge (u, s) and
        d(u, γ) = r − debt(u) + 2ri for i = 0, 1, 2, ..., N − 1}.
```

We define $\lceil x\rceil = 0$ for $-1 < x \le 0$. EDGE takes time $O(1)$ provided that the set $\mathscr{G}$ is represented symbolically in terms of u, debt(u), r, and N; that is, we do not have to store the points in $\mathscr{G}$ explicitly.

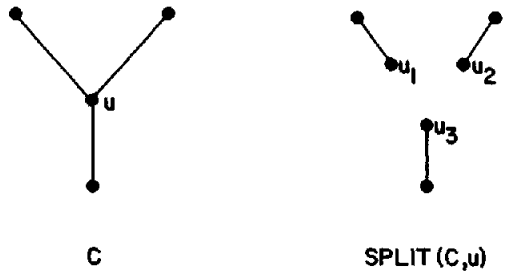


FIG. 6. The transformation of C that produces Split(C, u); the case $\delta = 3$ is shown.

If C is not an edge, COMP(C, s, $\mathscr{L}$, N) finds a vertex u of C such that $u \neq s$ and the degree of u relative to C is maximum. COMP next forms the graph Split(C, u) where u is replaced by vertices $u_1, \ldots, u_\delta$ (see Figure 6). The reason for splitting C is that we reduce its k.

LEMMA 3.1. $k(Split(C, u)) \leq k(C) - \delta + 1$.

PROOF. Let n and m be the number of vertices and edges, respectively, of C and let n' and m' be the number of vertices and edges of Split(C, u). Since C is nonseparable, $k(C) = m - n + 1$. Since Split(C, u) is connected, $k(\text{Split}(C, u)) \leq m' - n' + 1$. But $m' = m$ and $n' = n + \delta - 1$ so the result follows. □

LEMMA 3.2

$$\begin{aligned} &\textit{If} \quad k(C) = 1, \quad \textit{then} \quad \delta = 2. \\ &\textit{If} \quad k(C) \geq 2, \quad \textit{then} \quad \delta \geq 3. \end{aligned}$$

PROOF. If $k(C) = 1$, C is a cycle, so obviously $\delta = 2$.

If $k(C) \geq 2$, C has at least $n + 1$ edges, so C has at least one vertex of degree ≥ 3. If s were the only vertex of C with degree ≥ 3, then deleting s would separate C, contradicting the fact that C is nonseparable. Therefore, C has at least one $u \neq s$ with degree ≥ 3. □

COMP next calls SEP(Split(C, u), s, T). We would now like to call GRAPH recursively on copies of Split(C, u), but we have to account for the fact that paths passing through u in C are lost in Split(C, u). We account for these lost paths by calling GRAPH on several copies of C that differ only in the debts assigned to $u_1, \ldots, u_\delta$. For each half-integer x with $0 \leq x \leq r$ and each integer i with $1 \leq i \leq \delta$, COMP calls GRAPH($C(x, i)$, s, T, $\mathscr{L}$, N). Among the partial covers found, COMP returns one of minimum cost (any center placed on some u_j by GRAPH is placed on u by COMP). Informally, the idea here is that $r - x$ is the distance from u to a closest center γ_0 in some minimum partial cover of C w.r.t. s, and i is the index of an edge e_i on a shortest path from u to γ_0. Attaching a debt of x to u_i forces GRAPH to place a center within distance $r - x$ from u_i. The debts $-x$ attached to u_j for $j \neq i$ simulate the presence of γ_0. But since COMP does not know what x and i are, it tries all the possibilities.

More formally, to see that COMP is correct given that GRAPH is correct, let $\mathscr{L}_{\min}$ be a minimum cost partial cover of C w.r.t. s. Let $\langle N, b \rangle$ be the cost of $\mathscr{L}_{\min}$. Attach auxiliary edges to all vertices w of C with debt(w) $\neq 0$ as described in Section 2. If $b > 0$, attach another auxiliary edge of length $r - b$ to s connecting s to a vertex with a center; this center plays the role of a center that will be placed outside of C to cover the points of C not covered by $\mathscr{L}_{\min}$. Let C_{aux} be all points of C together with all points on auxiliary edges, and let $\mathscr{L}'$ be the centers in $\mathscr{L}_{\min}$

together with all centers on auxiliary edges. Note that $\mathscr{C}'$ covers C_{aux}. Recall that u is the vertex used to split C, and let

$$y = \min\{d(\gamma, u) \mid \gamma \in \mathscr{C}'\}.$$

There are three very similar cases.

(1) $0 < y < r$. Let $\gamma_0 \in \mathscr{C}'$ satisfy $d(\gamma_0, u) = y$, and assume for now that γ_0 is not on an auxiliary edge attached to u. Find a path of length y from γ_0 to u; say that this path uses edge e_i. We claim that GRAPH called on $B = C(r - y, i)$ returns a $\mathscr{C}$ with $\text{cost}(\mathscr{C}) \leq \text{cost}(\mathscr{C}_{\min})$. Since we need to talk about costs relative to both C and B, we use a subscript to indicate which graph is intended. First note that $\mathscr{C}_{\min}$ is a partial cover of B w.r.t. s and

$$\text{cost}_B(\mathscr{C}_{\min}) \leq \text{cost}_C(\mathscr{C}_{\min}).$$

This is true because, if ξ is an arbitrary point in C_{aux}, then the distance from ξ to a closest center in B is $\leq$ the distance from ξ to a closest center in C. For the moment, assume $\xi \neq u$. Let $\gamma_\xi \in \mathscr{C}'$ be a closest center to ξ in C. If a shortest path from ξ to γ_ξ does not pass through u, then clearly the distance cannot increase in B. If the shortest path passes through u, we can assume (since γ_0 is a closest center to u) that $\gamma_\xi = \gamma_0$, and the path starts from ξ, enters u along an edge u_j with $j \neq i$, leaves u along e_i and proceeds to γ_0. But $\text{debt}(u_j) = y - r < 0$ in B means that there is a center at distance y from u_j on an auxiliary edge, so the portion of the path from u_j to γ_0 that is broken in B is replaced by a path along the auxiliary edge. If $\xi = u$, then it follows from definitions that every copy u_j of u is at distance y from some center in B.

If γ_0 is on an auxiliary edge attached to u, we have $\text{debt}(u) = y - r$. In $C(r - y, i)$ we have $\text{debt}(u_j) = y - r$ for all i and j (by the definition of UPDATE). Therefore, the previous paragraph is valid for an arbitrary i and $B = C(r - y, i)$.

Assuming that GRAPH is correct, GRAPH returns a partial cover $\mathscr{C}$ with

$$\text{cost}_B(\mathscr{C}) \leq \text{cost}_B(\mathscr{C}_{\min}).$$

Finally, note that

$$\text{cost}_C(\mathscr{C}) \leq \text{cost}_B(\mathscr{C}).$$

This is true because rejoining $u_1, \ldots, u_\delta$ into one vertex u does not increase $d(\xi, s)$ for ξ an arbitrary point of C_{aux}, so $\text{debt}(s)$ cannot increase when $u_1, \ldots, u_\delta$ are rejoined into u.

(2) If $y = r$, then $\mathscr{C}_{\min}$ is a partial cover of $C(0, 1)$ w.r.t. s, and an argument similar to (1) applies.

(3) If $y = 0$, then $(\mathscr{C}_{\min} - \{u\}) \cup \{u_1\}$ is a partial cover of $C(r, 1)$ w.r.t. s, and the same argument applies.

There is no circularity between COMP and GRAPH in this correctness argument because of Lemma 3.2 (a more formal correctness proof would proceed by induction on $k(G)$).

Note that for $x = 0$ or $x = r$ the cases $C(x, i)$ are equivalent for all i, so it is sufficient to call GRAPH just on $C(x, 1)$. Therefore, the number of calls on GRAPH is at most $\delta(2r - 1) + 2$ which is $\leq 2r\delta$. This fact is used in the complexity evaluation. In practice, the number of calls could be further reduced: Because of the way UPDATE works, only $C(x, i)$ with $x \geq |\text{debt}(u)|$ need be considered.

Let $TC(m, k)$ ($TG(m, k)$, respectively) denote the running time of COMP (GRAPH, respectively), on a component (graph) G with m edges and $k = k(G)$. The following inequalities hold.

$$TG(m, k) \leq \max\{\sum TC(m_i, k_i) \mid \sum m_i = m \text{ and } \max k_i = k\} + \text{cm},$$

$$TC(m, k) \leq \max\{2r\delta \cdot (TG(m, k') + cm) \mid \delta \geq 3 \text{ and } k' \leq k - \delta + 1\} + cm \quad \text{for } k \geq 2 \text{ (recall Lemmas 3.1 and 3.2)},$$

$$TC(m, 0) \leq cm,$$

$$TC(m, 1) \leq (4r) \cdot cm.$$

The terms, cm, where c is a constant, account for calls on the procedure SEP, recursion overheads, and other computations done by the procedure. Let

$$\alpha(k) = \begin{cases} (6r)^{(k+1)/2} & \text{for } k \text{ odd}, \\ \frac{8}{6}(6r)^{k/2} & \text{for } k \text{ even}. \end{cases}$$

The above inequalities are satisfied by taking

$$TG(m, k) \leq 4cm \cdot \alpha(k) - 2cm,$$
$$TC(m, k) \leq 4cm \cdot \alpha(k) - 3cm.$$

Since the time for TOPLEVEL is $TG(m, k) + O(m)$, the claimed time bound, $O(m \cdot (6r)^{\lceil k/2 \rceil})$, follows.

This worst-case time bound could be overly pessimistic. It could happen that Split(C, u) separates into several components, so that choices made in different components would accumulate additively rather than multiplicatively. Since the problem of finding the minimum number of vertices whose removal breaks all cycles of C is an NP-complete problem (the feedback vertex set problem) [8], COMP uses the heuristic of picking vertices of maximum degree. It is quite possible that a more clever choice of splitting vertices would give better performance.

The reader might have noticed that the algorithm is inefficient in some places. For example, it is possible that Split(C, u) could be a component C' together with paths of vertices of degree 2 connecting C' to $u_1, \ldots, u_\delta$. Say u_j is connected to C' by a long path. Each recursive call on GRAPH propagates debt(u_j) along the path to C' using calls on EDGE. For each fixed x, all $C(x, i)$ with $i \neq j$ have the same debt(u_j), so redundant work is being done. We did not attempt to remove these inefficiencies in the description of the algorithm since it would just complicate the exposition without improving the worst-case time bound.

4. *A Parallel Implementation of the Algorithm*

For some efficient sequential algorithms that have polynomial worst-case running times, it is not obvious how to parallelize the algorithm, reducing the running time by more than a constant even if many processors are available. It seems, however, that for exponential time algorithms, where the exponential is due to branching over many choices, a lot can be done. We assume a model of synchronous parallel computation in which many random-access machines communicate via a shared common memory, and all overheads are taken into account (e.g., see [16] and see also [20] for a recent survey of results concerning this model). The framework of the parallel implementation we suggest follows the ideas of the following theorem and its proof.

THEOREM (Brent [2]). *Any synchronized parallel algorithm that runs in parallel time d and consists of x elementary operations can be implemented by p processors within time $\lceil x/p \rceil + d$.*

PROOF. Let x_i denote the number of operations performed by the algorithm at step i, ($\sum_{i=1}^{d} x_i = x$). We now use the p processors to simulate the algorithm. Since all the operations at step i can be executed simultaneously, they can be computed by p processors in $\lceil x_i/p \rceil$ units of time. Thus, the whole algorithm can be implemented by p processors in time

$$\sum_{i=1}^{d} \left\lceil \frac{x_i}{p} \right\rceil \leq \sum_{i=1}^{d} \left(\frac{x_i}{p} + 1 \right) \leq \left\lceil \frac{x}{p} \right\rceil + d. \qquad \square$$

Remark. The proof of Brent's theorem poses an implementation problem: how to assign the processors to their jobs.

Let us go back to our algorithm. Assume that we had all the processors we might have needed in every time unit of the algorithm and we could assign these processors to their jobs in constant time. It is readily seen, and therefore left to the reader, that the time that our algorithm requires under these new assumptions is bounded by

$$\left\lceil \frac{k}{2} \right\rceil \cdot (\text{the time for SEP plus a constant}) = O(km).$$

Here $\lceil k/2 \rceil$ is the number of times we might need to repeatedly apply SEP, charging a given edge. So $O(m \cdot (6r)^{\lceil k/2 \rceil}/km) = O((6r)^{\lceil k/2 \rceil}/k)$ processors would suffice to get this time bound.

The problem of assigning processors to their jobs is solved easily by the following instruction, added at the branchings of COMP over choices of splitting vertices. If p processors are to be assigned among c choices we do the following: If $p \geq c$, the processors are partitioned among the choices, $\lceil p/c \rceil$ or $\lfloor p/c \rfloor$ processors for each choice; else the choices are partitioned among the processors, $\lceil c/p \rceil$ or $\lfloor c/p \rfloor$ choices for each processor. The parallel time we get is therefore

$$O\left(\frac{m \cdot (6r)^{\lceil k/2 \rceil}}{p} \right) \quad \text{for} \quad p \leq \frac{(6r)^{\lceil k/2 \rceil}}{k} \quad \text{processors}$$

without even trying to parallelize the SEP algorithm.

Remark. A further parallelization of our algorithm is possible. Many algorithms on rooted trees that work from the leaves to the root can be parallelized using the "centroid decomposition" technique (e.g., see Megiddo [14]) in order to get $O(\log^2 n)$ parallel time instead of $O(n)$ sequential time, where n is the number of vertices. This, in conjunction with the recent parallel biconnectivity algorithm of Tarjan and Vishkin [19] can be used in order to get parallel time $O(k \log^2 n)$ using sufficiently many processors. Since this involves utilization of known methods and seems tedious, we do not elaborate on this but rather leave it as an exercise for the interested reader.

5. *Related Results and Open Questions*

The algorithm can be modified to solve the other variation of the facility location problem where just vertices must be within distance r from some center. Minor modifications to the procedure EDGE are needed. (The reader probably noticed that EDGE does all the work of actually placing centers, while GRAPH and COMP just provide control.) Lemma 2.1 is replaced by the (trivial) fact that w.l.o.g. centers can be placed on points that are at distance exactly r from some vertex (if there are no such points, then one center placed anywhere covers the entire graph). Because of this fact we suspect that a better time bound is possible.

The basic method of the algorithm can be applied to the vertex cover problem, the independent set problem, and the dominating set problem. All of these problems are NP-complete even for planar graphs of degree 3 [8]. In the vertex cover problem, we are given a graph $G(V, E)$ and we want a set $C \subseteq V$ of minimum cardinality such that every edge in E has at least one endpoint in C. The general outline of the algorithm is followed. When a vertex u is split, there are only two choices: either $u \in C$ or $u \notin C$. This yields the time bound $O(m \cdot 2^{k/2})$. The same bound holds for the independent set problem since C is a vertex cover in $G(V, E)$ iff $V - C$ is an independent set. In the dominating set problem, we want a set $D \subseteq V$ of minimum cardinality such that for every vertex $v \notin D$ there is a $u \in D$ with $(u, v) \in E$. Now there are $\delta + 1$ choices at a splitting vertex u of degree δ, since either $u \in D$ or at least one of u's δ neighbors is in D. This yields the time bound, $O(m \cdot 4^{k/2})$.

Perhaps the most interesting technical question raised by this paper is whether r can be removed from the base of the exponent in the time bound; that is, is there an algorithm for the continuous location problem with time complexity of the form $O(q(n) \cdot c^k)$ or, less optimistically, $O(q(n) \cdot n^{ck})$ for some constant c and some reasonable polynomially bounded function $q(n)$? In a preliminary version of this paper [10], we have taken a first step toward settling this question by giving an algorithm of time complexity $O(|V| \log |V|)$ for the case $k(G) \leq 2$ (i.e., every nonseparable component is either an edge, a simple cycle, or a simple cycle with one chord). Furthermore, this algorithm does not require r and edge lengths to be integers.

ACKNOWLEDGMENTS. As we mentioned above in Section 2, Hang Tong Lau, Brendan D. McKay, and Ioannis Tollis proved independently the fact that, in the case where r and edge lengths are integers, there is a minimum cover with centers on half-integer points. We are grateful to them for permitting us to use their proof, which is much simpler than our proof. We also thank the referees for many useful suggestions.

REFERENCES

1. AHO, A. V., HOPCROFT, J. E., AND ULLMAN, J. D. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, Mass., 1974.
2. BRENT, R. P. The parallel evaluation of general arithmetic expressions. *J. ACM 21*, 2 (1974), 201–206.
3. CHANDRASEKARAN, R., AND DAUGHETY, A. Location on tree networks: p-centre and n-dispersion problems. *Math. Oper. Res 6* (1981), 50–57.
4. CHRISTOFIDES, N., AND VIOLA, P. The optimum location of multi-centres on a graph. *Oper Res Quarterly 22* (1971), 145–154.
5. COPPERSMITH, D., AND VISHKIN, U. Solving NP-hard problems on almost trees: Vertex cover. Report RC 9404, IBM Thomas J. Watson Research Center, Yorktown Heights, N. Y., 1982.
6. EVEN, S. *Graph Algorithms.* Computer Science Press, Potomac, Md., 1979.
7. FREDERICKSON, G. N., AND JOHNSON, D. B. Finding k^{th} paths and p-centers by generating and seaching good data structures. *J. Algorithms 4* (1983), 61–80.
8. GAREY, M. R., AND JOHNSON, D. S. *Computers and Intractability—A Guide to the Theory of NP-Completeness.* W. H. Freeman, San Francisco, 1979.
9. GARFINKEL, R. S., NEEBE, A. W., AND RAO, M. R. The m-center problem: Minimax facility location. *Manage. Sci. 23* (1977), 1133–1142.
10. GUREVICH, Y., STOCKMEYER, L., AND VISHKIN, U. Solving NP-hard problems on graphs that are almost trees and an application to facility location problems. Report RC 9348, IBM Thomas J. Watson Research Center, Yorktown Heights, N.Y., 1982.
11. KARIV, O., AND HAKIMI, S. L. An algorithmic approach to network location problems. I: The p-centers. *SIAM J. Appl. Math 37* (1979), 513–538.

12. LIPTON, R. J., AND TARJAN, R. E. Applications of a planar separator theorem. *SIAM J. Comput. 9* (1980), 615–627.
13. LIPTON, R. J., AND TARJAN, R. E. A separator theorem for planar graphs. *SIAM J. Appl. Math. 36* (1979), 177–189.
14. MEGIDDO, N. Applying parallel computation algorithms in the design of serial algorithms. In *Proceedings of the 22nd IEEE Symposium on Foundations of Computer Science.* IEEE, New York, 1981, 399–408.
15. MEGIDDO, N., TAMIR, A., ZEMEL, E., AND CHANDRASEKARAN, R. An $O(n \log^2 n)$ algorithm for the k^{th} longest path in a tree with applications to location problems. *SIAM J. Comput. 10* (1981), 328–337.
16. SHILOACH, Y., AND VISHKIN, U. Finding the maximum, merging and sorting in a parallel computation model. *J. Algorithms 2* (1981), 88–102.
17. TARJAN, R. E. Depth first search and linear graph algorithms. *SIAM J. Comput. 1* (1972), 146–160.
18. TARJAN, R. E., AND TROJANOWSKI, A. E. Finding a maximum independent set. *SIAM J. Comput. 6* (1977), 537–546.
19. TARJAN, R. E., AND VISHKIN, U. An efficient parallel biconnectivity algorithm. Rep. TR-69, Dept. of Computer Science, Courant Institute, New York Univ., New York, 1983.
20. VISHKIN, U. Synchronous parallel computation—a survey. Rep. TR-71, Dept. of Computer Science, Courant Institute, New York Univ., New York, 1983.

RECEIVED APRIL 1982; REVISED DECEMBER 1983; ACCEPTED DECEMBER 1983