JOURNAL OF ALGORITHMS **29,** 111–131 (1998)
ARTICLE NO. AL980949

Tree Powers

Paul E. Kearney*

Department of Computer Science, University of Toronto, Toronto, ON, Canada M5S 3G4

and

Derek G. Corneil†

Department of Computer Science, University of Waterloo, Waterloo, ON, Canada N2L 3G1

Received March 26, 1996

We present the first polynomial algorithm for recognizing tree powers. A graph G is a *tree power* if there is a tree T and a positive integer k such that $T^k \cong G$, where x and y are adjacent in T^k if and only if $d_T(x, y) \leq k$. We also show that a natural extension of tree power recognition is NP-complete, namely, given a graph G and a positive integer r, determine if there is a tree power within r edges of G.

1. INTRODUCTION

Root and *root finding* are concepts familiar to most branches of mathematics. In graph theory, H is a *root* of $G = (V, E)$ if there exists a positive integer k such that

$$xy \in E \quad \Leftrightarrow \quad d_H(x, y) \leq k,$$

where $d_H(x, y)$ is the length of a shortest path in H from x to y. If H is a kth root of G then we write $G \cong H^k$ and call G the *kth power* of H. Powers of graphs are analogous to powers of numbers since $A(H^k) = A(H)^k$, where $A(H)$ denotes the adjacency matrix of H with 1s on the diagonal and $A(H)^k$ is reduced such that nonzero entries are replaced by 1s.

* E-mail: pkearney@math.uwaterloo.ca.
† E-mail: dgc@cs.toronto.edu.

0196-6774/98 $25.00

The literature is rich with results on graph roots and powers. Substantial work has been done on the powers of special classes of graphs such as chordal graphs [2, 13], interval graphs [20], cocomparability graphs [3], strongly chordal graphs [17, 21], and circular arc graphs [21]. A related area of research concerns algorithms defined on classes of graph powers such as Hamiltonian cycle on graph powers [15] and Hamiltonian cycle on squares of 2-connected graphs [14].

For any class of graphs, recognition is a fundamental problem. In the present context, no examination of graph powers and graph roots would be complete unless the question "Does G have a root?" is addressed. Despite the importance of recognition, substantial results in the area of graph powers and roots are scarce. Characterizations of graph squares [19], square roots of directed graphs [7], and graph powers in general [4] have been given, yet these characterizations have not yielded polynomial recognition algorithms. Motwani and Sudan [18] have shown that unrestricted graph power recognition is NP-complete. Although this result testifies to the difficulty of graph power recognition, it does not shed light on restricted graph power recognition problems.

In this paper we present an $O(n^3)$ algorithm for recognizing kth powers of trees, where n is the number of vertices in the input graph. Not only is this the first polynomial algorithm for tree power recognition, but it is one of the first recognition algorithms for nontrivial classes of graph powers. Stated explicitly, the problem we solve is

Tree Power Recognition (TPR):

Instance: A graph $G = (V, E)$ and positive integer k.

Problem: Is there a tree T such that $T^k \cong G$?

As stated, TPR is a decision problem. Our algorithm is constructive in the sense that it solves TPR by producing a tree root T such that $T^k = G$. Note that we solve for T such that $T^k = G$ instead of the weaker $T^k \cong G$, since we permit vertices of G to be labelled. An example of a tree and its cube is given in Figure 1. Although TPR is parameterized by k, $G = (V, E)$

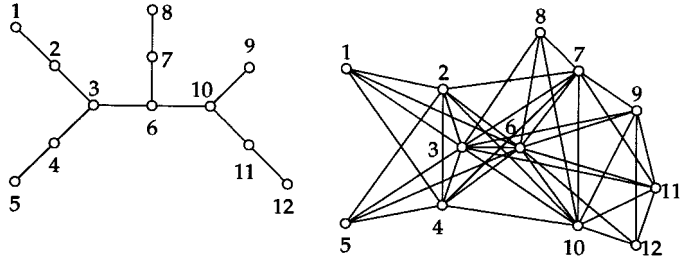

FIG. 1. A tree and its cube.

is a tree power if and only if G is a kth tree power for some k, $1 \leq k < |V|$. Hence, it suffices to provide a polynomial algorithm for TPR to determine if G is a tree power.

Tree power recognition has received recent attention from Lin and Skiena [15], who give a polynomial algorithm for recognizing squares of trees. Hamada [10] has given an algorithm that finds all kth tree roots of an input graph with the restriction that it be known a priori that the input graph is a kth tree power. This restriction is necessary since the algorithm does not produce the bijection between vertices of the input graph and vertices of a generated tree. As a result, verifying a solution reduces to an instance of graph isomorphism. Hence, Hamada's algorithm does not solve the recognition problem.

Although our interest in tree powers is largely theoretical, graph powers are associated with problems in distributed computing [16]. In particular, tree powers model the situation where a collection of processors is (minimally) connected by a tree network T. A communication step is the amount of time required for a processor to pass information to a neighbor. Hence, T^k represents the information exchange after k communication steps.

Having shown TPR to be polynomially solvable, it is natural to ask if more general related problems are also polynomially solvable. To this end we investigate the following problem:

Closest kth Tree Power (CTP(k))

Instance: A graph $G = (V, E)$ and a nonnegative integer r.

Problem: Is there a kth tree power $G' = (V, E')$ such that $|E \vartriangle E'| \leq r$?

Here $E \vartriangle E'$ is the set $E \cup E' - E \cap E'$, which is the disparity between G and G'.

Observe that CTP(k) reduces to an instance of TPR when $r = 0$. The extension of TPR to CTP(k) is analogous to the extension of interval graph recognition to completion and sandwich problems defined on interval graphs [6, 9, 8]. These problems attempt to model error in the input graph when the input graph is obtained from experimental data, for example, in the application of interval graphs to DNA physical mapping [8]. The assumption is that although the input graph contains errors and may not have an exact solution, there is a graph close (edgewise) to the input graph which does. In the same way, CTP(k) has application to the scenario where a graph G is to be implemented by some tree network T. In the case that G is not an exact tree power, it is desirable to find the closest (edgewise) graph to G that is.

We sketch a proof that CTP(k) is NP-complete when $k = 3$. Our approach can be extended to show NP-completeness for larger values of k.

For fixed values of the parameter r, CTP(k) is solvable in polynomial time by calling our tree power recognition algorithm $O(n^r)$ times.

The remainder of this paper is organized as follows. In Section 2 we review graph theoretic terminology and concepts used throughout the paper. In Section 3 we begin with an overview of the tree power recognition algorithm and follow with a detailed description of the algorithm and supporting theory. Finally, in Section 4 we briefly discuss the NP-completeness of CTP(k).

2. DEFINITIONS

Let $G = (V, E)$ be an undirected and simple graph. $G[S]$ denotes the subgraph of G induced by $S \subseteq V$. $G(v)$ is the connected component of G containing vertex v. $N(v)$ denotes the *open neighborhood* of v, that is, $\{u \in V: uv \in E\}$. $N[v]$ denotes the *closed neighborhood* of v, that is, $N(v) \cup \{v\}$. We write $u \sim v$ if $N[u] = N[v]$ and call u and v *twins*. If $N[v] = V$ then we say v *dominates* G. With respect to a set $U \subseteq V$, we say v *G-dominates* U if $U \subseteq N[u]$ in G.

Vertices u and v are *neighborhood compatible* if $N(u) \subseteq N(v)$ or $N(v) \subseteq N(u)$. A set $U \subseteq V$ is *neighborhood orderable* if for each pair of vertices $u, v \in U$, u and v are neighborhood compatible. A vertex $v \in V$ is *simplicial* if $N[v]$ is complete and *simple* if $N[v]$ is complete and neighborhood orderable. A *simple elimination ordering* of V is an ordering $v_1, v_2, \ldots, v_n$ of V such that v_i is simple in the subgraph induced by $\{v_i, v_{i+1}, \ldots, v_n\}$, for $1 \leq i \leq n$.

With respect to a tree $T = (V, E)$ we define the following terms. $P(x, y)$ denotes the unique path in T from x to y and $d(x, y)$ denotes the number of edges in $P(x, y)$. T' is a *subtree of* T *at* v if T' is a connected component of $T - \{v\}$. We say v *inserts* onto $P(x, y)$ at vertex $p \in P(x, y)$ if $P(x, y) \cap P(x, v) \cap P(v, y) = \{p\}$ in T. The pair (u, v) is an *extreme pair* in T if $d(u, v)$ realizes the diameter of T. If u is part of an extreme pair then u is called an *extreme leaf* of T. The center of T, center(T), is the set of vertices in T that are within distance $\lceil \text{diam}(T)/2 \rceil$ of all vertices in T. It is well known that the center of a tree is either a vertex or two adjacent vertices.

3. RECOGNIZING TREE POWERS

Let the input instance to TPR be $G = (V, E)$ and k. Since tree powers are necessarily connected, we assume G is connected. Let T be a kth tree root of G, assuming one exists.

An ideal strategy for reconstructing T from G would be to locate a leaf v of T in G and then to determine v's neighbor r in T. We call r the *root* of v. The leaf v is then removed from G and we continue determining leaves and their roots recursively. Our algorithm is based on this simple paradigm; however, tests used to isolate leaves and their roots recursively can be complex.

We begin by introducing terms used to describe the algorithm. Let i denote the current iteration and V_i denote the set of unprocessed vertices, that is, $V - V_i$ is the set of vertices pruned from T thus far. T_i denotes the graph with edges $\{vr\colon v \in V - V_i, r \text{ the root of } v\}$. Hence, T_i is a partial solution to the input instance at iteration i. Stated more explicitly, T_i satisfies the following invariant:

There exists a kth tree root T of G such that

—T_i is a collection of subtrees of T and

—$T[V_i]$ is a tree.

Figure 2 depicts the state at iteration i. The shaded area is T_i, the dashed edges compose $T[V_i]$, and the vertices inside the dashed box compose V_i. During iteration i, r may be selected as a leaf of $T[V_i]$ with root u. During some previous iteration v was selected as a leaf with root r. Note that $T_i(v)$ denotes the connected component of T_i that contains v.

The framework of our algorithm is given in Table 1. To complete our description we need to describe how leaves of $T[V_i]$ are located in G and how the root of a leaf is determined. The reader is encouraged to refer to Table 1 throughout this section. In particular, observe that the paradigm of finding a leaf and its root is broken into two cases: when V_i consists solely

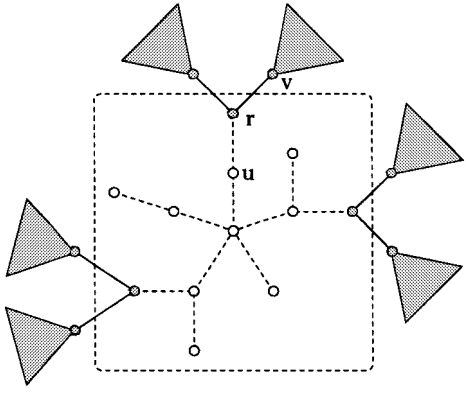

FIG. 2. The state at iteration i.

TABLE 1
TPR Recognition Algorithm

```
V_1 := V
T_1 := ∅
i := 1
if G is complete then
    return any star with |V| − 1 leaves
end if
while not all vertices of V_i dominate G do
    find a leaf v ∈ V_i which is simple in G[V_i]
    find the root r ∈ V_i of v
    V_{i+1} := V_i − {v}
    T_{i+1} := T_i + {rv}
    i := i + 1
end while
while T_i not connected do
    L_i = {v ∈ V_i: T_i(v) − {v} ≠ ∅}
    for each v ∈ L_i do
        find root r ∈ V_i of v
        T_i := T_i ∪ {rv}
    end for
    V_{i+1} := V_i − L_i
    T_{i+1} := T_i
    i := i + 1
end while
for each v ∈ V_i do
    T_i := T_i ∪ {vc}, c a vertex in center(T_i)
end for
return T_i
```

of vertices which dominate G and when it does not. Before beginning, we note that there is potential for confusion when discussing G and T simultaneously. Unless otherwise noted, statements concerning adjacencies are with respect to G and not T.

3.1. *Looking for Leaves*

Define the *core* of G, core(G), to be those vertices (if any) that dominate G. For illustration, let $G = T^7$, where T is the tree depicted in Figure 3. The core of G is the set $\{21, 22, \ldots, 32\}$. Locating a leaf of $T[V_i]$ in $G[V_i]$ is broken into two cases: when V_i consists solely of vertices of core(G) and when it does not. We address the latter case first. In the degenerate case where G is initially complete then any star on V (a tree with $|V| - 1$ leaves) is a kth tree root for G if $k \geq 2$.

We begin with a sufficient condition for a vertex of G to be a leaf of T:

SIMPLICIAL LEAF LEMMA. *If v does not dominate G and v is simplicial in G then v is a leaf of any kth tree root of G.*

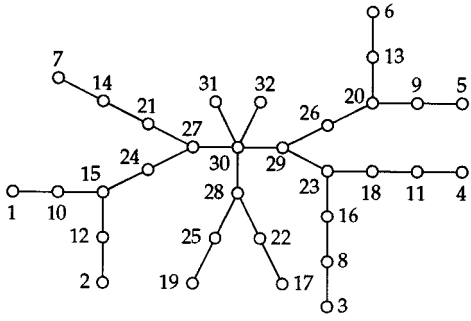

FIG. 3. A tree root T.

Proof. Let v be simplicial in $G = (V, E)$ but not dominate G and let T be a kth tree root of G such that $\deg(v) \geq 2$ in T. Let a and b be neighbors of v in T and let T_a be the subtree of T at v containing a; define T_b analogously. If there is a vertex $a' \notin T_a$ such that $d_T(a', v) = k$ then $aa' \notin E$ implying v is not simplicial. Hence, $d_T(a', v) < k$ for all $a' \notin T_a$. An analogous argument shows $d_T(b', v) < k$ for all $b' \notin T_b$. It follows that $d_T(v', v) < k$ for all $v' \in V$ implying v dominates G, contradicting our initial assumption; hence $\deg(v) = 1$ in T. ∎

The converge of the simplicial leaf lemma is not generally true. For example, vertex 7 of the tree depicted in Figure 3 is a leaf, but not simplicial in T^7 (7 is adjacent to 1 and 20 in T^7, but 1 and 20 are not adjacent). However, the converse does hold for a restricted class of leaves:

EXTREME LEAF LEMMA. *If v is an extreme leaf of T then v is simple in T^k.*

Proof. Let (v, v') be an extreme pair in T. Suppose $x, y \in N(v)$, but $xy \notin E(G)$. Since vx, vy are edges of T^k, but xy is not, then $d_T(v, x), d_T(v, y) \leq k < d_T(x, y)$. It is then easy to see that either $d_T(v, v') < d_T(v', x)$ or $d_T(v, v') < d_T(v', y)$—contradicting the maximality of $d_T(v, v')$. This proves that v is simplicial.

Suppose $x, y \in N(v)$ and x, y are not neighborhood compatible, hence, there exists $x' \in N(x) - N(y)$ and $y' \in N(y) - N(x)$. Since xx', yy' are edges of T^k but $xy', x'y$ are not edges of T^k then $d_T(x, x'), d_T(y, y') \leq k < d_T(x, y'), d_T(y, x')$ implying $P(x, x') \cap P(y, y') = \emptyset$ in T. Again, using a straightforward case-by-case analysis, it can be shown that either $d_T(v, v') < d_T(v', x')$ or $d_T(v, v') < d_T(v', y')$—contradicting the maximality of $d_T(v, v')$. We conclude that $N[v]$ is neighborhood orderable. ∎

Recall that v being simple implies v is simplicial. A corollary of the extreme leaf lemma is that a simple elimination order of T^k can be obtained from an extreme leaf elimination order of T. This provides a

short proof that tree powers are *strongly chordal*, namely, there is an ordering $v_1, v_2, \ldots, v_n$ of V such that for each i, j, k, and l, if $i < j$, $k < l$, $v_k, v_l \in N[v_i]$, and $v_k \in N[v_j]$ then $v_l \in N[v_j]$. It turns out that G is strongly chordal if and only if G has a simple elimination order [5]. The extreme leaf lemma also guarantees the existence of a leaf v of $T[V_i]$ which is simple in $G[V_i]$. We will capitalize on this fact in Section 3.2 when we determine v's root.

Simplicial tests for leaves are utilized in [15] to determine leaves of tree square roots. However, for $k > 2$ we need to devise a stronger test in order to determine leaves recursively since eventually $G[V_i]$ will be complete and simplicial tests will fail to distinguish leaves from nonleaves since all vertices are simplicial in a complete graph. The test we develop is based upon neighborhood compatibility. The motivation for this is that the neighborhoods of a leaf and its root are compatible and no vertex neighborhood can be strictly contained in the neighborhood of a leaf. These properties form the signature of a leaf in $T[V_i]$.

INDUCTIVE LEAF LEMMA. *If*

1. $v \in V_i$ *does not G-dominate* $V(T - T_i(v))$, *and*
2. *there exists* $u \in V_i$ *such that* $N(v) \subseteq N(u)$ *in* $V(T - T_i(v))$ *and*
3. *there does not exist* $w \in V_i$ *such that* $N[w] \subset N[v]$ *in* $V(T - T_i(v))$

then v *is a leaf of* $T[V_i]$.

Proof. Let v and u satisfy the conditions of the lemma but $\deg(v) \geq 2$ in $T' = T - T_i(v) \cup \{v\}$. Let T_u be the subtree of T' at v containing u and let w be any vertex of T' adjacent to v but not in T_u. Since $N(v) \subseteq N(u)$ in $V(T - T_i(v))$, v dominates $V(T - T_i(v) - T_u)$; otherwise $N(v) \not\subseteq N(u)$. It follows that $N[w] \subseteq N[v]$ in $V(T - T_i(v))$. Since $N[w] \not\subset N[v]$, it follows that $N[w] = N[v]$ in $V(T - T_i(v))$. This can only occur if v dominates $V(T - T_i(v))$ which is a contradiction. ∎

For example, consider the tree T in Figure 4 and let $G = T^3$. Using the simplicial leaf lemma, vertices 1, 2, 4, 5, and 7 are correctly identified as

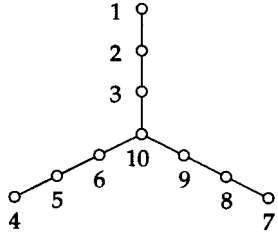

FIG. 4. Neighborhood compatibility versus simplicial tests for leaves.

leaves; hence, $V_6 = \{3, 6, 8, 9, 10\}$. All $v \in V_6$ now dominate $G[V_6]$, and so the simplicial leaf lemma is no longer applicable. However, the inductive leaf lemma correctly identifies 3, 6, and 8 leaves of $T[V_6]$.

It remains to be shown how the inductive leaf lemma can be used to find leaves of $T[V_i]$ given that not all vertices in V_i dominate G. We say the pair $v, u \in V_i$ satisfies the induction leaf lemma if v and u satisfy conditions 1, 2, and 3 as stated in the lemma.

LEMMA 1. *If not all vertices in V_i dominate G, then there is a pair of vertices $v, u \in V_i$ which satisfies the inductive leaf lemma.*

Proof. Since $T[V_i]$ is a tree it has an extreme pair (u, v). Let $w \in V_i$ such that w does not dominate G; in particular, suppose w is not adjacent to x in G. Since (u, v) is an extreme pair of $T[V_i]$, $d_T(u, w), d_T(v, w) \leq d_T(u, v)$. Let $p = P_T(u, v) \cap P_T(u, w) \cap P_T(v, w)$ and let x insert onto $P_T(u, v)$ at q. If $q \in P_T(u, p)$ then $d_T(x, w) \leq d_T(x, v)$ since $d_T(u, w) \leq d_T(u, v)$. If $q \in P_T(v, p)$ then $d_T(x, w) \leq d_T(x, u)$ since $d_T(v, w) \leq d_T(u, v)$. Hence, either v does not dominate G (v is not adjacent to x) or u does not dominate G (u is not adjacent to x). Assume the former without loss of generality.

If v does not G-dominate $V(T_i(v))$ then $d_T(v, v') > k$ for some $v' \in V(T_i(v))$. It follows that $d_T(u, v') > k$; hence, u does not G-dominate $V(T - T_i(u))$. Otherwise, v does not G-dominate $V(T - T_i(v))$. It follows that either v does not G-dominate $V(T - T_i(v))$ or u does not G-dominate $V(T - T_i(u))$. Without loss of generality, assume the former.

Let r be v's root in $T[V_i]$; hence, $N(v) \subseteq N(r)$ in $V(T - T_i(v))$. If there does not exist $w \in V_i$ such that $N[w] \subset N[v]$ in $V(T - T_i(v))$ then the inductive leaf lemma is satisfied by v and r. Otherwise, let $w \in V_i$ such that $N[w] \subset N[v]$ in $V(T - T_i(v))$. Observe that

—w does not dominate $V(T - T_i(w))$ since $N[w] \subset N[v]$ in $V(T - T_i(v))$ and all vertices in $T_i(w)$ are closer to w than to v in T, and

—$N[w] \subseteq N[v]$ in $V(T - T_i(w))$ since $N[w] \subset N[v]$ in $V(T - T_i(v))$.

Hence, either w and v satisfy the inductive leaf lemma (where w is substituted for v and v is substituted for u) or there is some $w' \in V_i$ such that $N[w'] \subset N[w]$ in $V(T - T_i(w))$. In the latter case, the argument can be repeated where v is replaced by w and w is replaced by w'. Hence, we have a recurrence which must terminate by the finiteness of V_i. ∎

We summarize leaf finding when V_i contains vertices not in core(G), that is, V_i contains vertices that do not dominate G:

If $G[V_i]$ is not complete then the simplicial leaf lemma can be invoked; any simplicial vertex of $G[V_i]$ will be a leaf of $T[V_i]$. In fact, by

the extreme leaf lemma, there will be a simple vertex in $G[V_i]$; hence, select any simple vertex of $G[V_i]$ to be our leaf.

If $G[V_i]$ is complete then all vertices of $G[V_i]$ will be simple. Since the simplicial leaf lemma is no longer applicable, we appeal to the inductive leaf lemma to find a leaf of $T[V_i]$.

Now we address the case when all vertices of V_i are in core(G). Note that an element of core(G) is never chosen as a leaf in the first loop of the algorithm; hence, V_i contains *all* vertices of core(G). We want to use the same paradigm of finding a leaf and then determining its root. To achieve this we need some preliminary results. We assume $|V_i| \geq 2$ since otherwise T_i is connected and there is nothing left to do.

LEMMA 2. *If $v \in V_i$, V_i = core(G), and $T_i(v) - \{v\}$ is nonempty then v is a leaf of $T[V_i]$.*

Proof. Let $T_i(v) - \{v\} \neq \emptyset$ and suppose v is adjacent to a and b in $T[V_i]$, where T_a denotes the subtree of T at v containing a and T_b denotes the subtree of T at v containing b. Let $p \in T_i(v)$ be adjacent to v in T. Since a dominates G, it follows that p G-dominates $V(T - T_a)$. Similarly, since b dominates G, p G-dominates $V(T - T_b)$. It follows that p dominates G, but then $p \in V_i$ contradicting $p \in T_i(v)$. That is, p would not have been chosen as a leaf at an earlier iteration. We conclude that v is a leaf of $T[V_i]$. ∎

If v is a leaf of $T[V_i]$ but $T_i(v) - \{v\}$ is empty then the only constraint on v's location in T is that it be within distance k of all other vertices in T. It suffices to let v's root in T be a vertex in center(T). For example, if $G = T^7$, where T is the tree depicted in Figure 3, then vertices 31 and 32 are examples of core leaves which have empty subtrees.

Lemma 2 and the discussion above motivate the following definition:

$$L_i = \{v \in V_i : T_i(v) - \{v\} \neq \emptyset\}.$$

Hence, leaves in L_i are the "important" leaves of $T[V_i]$. All other leaves can be rooted at center(T) at the end of the algorithm.

Observe that center(T) $\subseteq V_i$ since if some $v \in V_i$ dominates G then those vertices in center(T) must also dominate G. Define $d_T(v, \text{center}(T))$ to be the distance in T from v to the closest element of center(T).

LEMMA 3. *All elements of L_i are equidistant from* center(T) *in T.*

Proof. Suppose $u, v \in L_i$ such that $d_T(u, \text{center}(T)) < d_T(v, \text{center}(T))$. Since $u \in L_i$ there exists $w \in T_i(u)$ adjacent to u in T.

It is a well-known fact that the center of a tree is the middle vertex (or edge) of every longest path in the tree. Let $P(s, t)$ be a longest path in T. Without loss of generality, assume u inserts onto $P(s, t)$ no closer to t than to s. Since $w \notin V_i$ = core(G), w does not dominate G; in particular,

$wt \notin E$. If v inserts onto $P(s,t)$ no closer to t than to s then $d_T(u, \text{center}(T)) < d_T(v, \text{center}(T))$ implies that $d_T(w, \text{center}(T)) \leq d_T(v, \text{center}(T)) \leq k$ (since v dominates G). This contradicts $wt \notin E$. It follows that v inserts onto $P(s,t)$ closer to t than to s.

However, $wt \notin E$, $ut \in E$, and u and w adjacent in T imply that $d_T(u,t) = k$. Yet, $d_T(u,t) < d_T(v,s)$ since s and t are equidistant from center(T). It follows that $d_T(v,s) > k$ contradicting v dominates G. ∎

Lemma 3 permits us to determine leaves recursively. Since the leaves of L_i are equidistant from the center of T, if V_{i+1} is defined to be $V_i - L_i$ then L_{i+1} will be the roots of vertices in L_i. The vertices of L_{i+1} will then also be equidistant from center(T).

3.2. *Determining the Root of a Leaf*

Now that we have found a leaf v of $T[V_i]$ which is simple in $G[V_i]$, we want to determine v's root r in $T[V_i]$. Initially, all vertices of $N(v) \cap V_i$ are candidates for v's root. Our goal is to reduce $N(v) \cap V_i$ to a set root(v) of twins in G and then to select r from among these vertices. This process is complicated by the facts that determining v's root is dependent upon the graph T_i constructed thus far and that v's root is not necessarily unique.

By the inductive leaf lemma, v is the extremity of a chain of neighborhood compatible vertices. From this perspective we would expect r to be an immediate predecessor of v in this chain. This motivates the following definition.

DEFINITION 4. The set of *nearest neighbors* of v in G, NN(v), is defined to be

$$\{u \in N(v) : N[v] \subset N[u] \text{ and } \nexists w \text{ such that } N[v] \subset N[w] \subset N[u]\}.$$

NEAREST NEIGHBOR LEMMA. *If $G = T^k$, v is simple in G, v is a leaf of T, and v does not dominate G then v's root in T is a nearest neighbor of v.*

Proof. Let $G = (V, E)$ and suppose v's root in T is $r \notin \text{NN}(v)$. Since v does not dominate G, $N[v] \subset N[r]$ in G. It follows that there exists w such that $N[v] \subset N[w] \subset N[r]$.

Let $w_2 \in N[w] - N[v]$ such that $d_T(v, w_2) = k + 1$ and let $w_1 \in P_T(v, w_2)$ such that $d_T(v, w_1) = k$. Let $r_2 \in N[r] - N[w]$ such that $d_T(v, r_2) = k + 1$ and let $r_1 \in P_T(v, r_2)$ such that $d_T(v, r_1) = k$. Since $r_2 \notin N[w]$ and $r_1 \in N[w]$, it follows that $d_T(w, r_1) = k = d_T(v, r_1)$. Hence, r_1 inserts onto $P_T(v, w)$ at the midpoint p of $P_T(v, w)$. Since w_2 is closer to w than to v in T, w_2 inserts onto $P_T(v, w)$ at q such that w is closer to q than to p. The above is depicted in Figure 5.

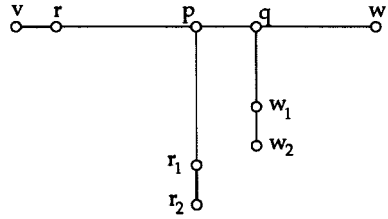

FIG. 5. Vertices v, r, and w in the proof of the nearest neighbor lemma.

Since $vw \in E$, $d_T(v,w) \leq k$, hence, $d_T(v,p) \leq k/2$. However, $d_T(v,r_1) = d_T(v,p) + d_T(p,r_1) = k$, hence, $d_T(p,r_1) \geq k/2$. It follows that $d_T(r_1,w_1) \geq k$ since $d_T(v,r_1) = d_T(v,w_1)$. Since $r_1, w_1 \in N[v]$ implying $r_1w_1 \in E$ it follows that $d_T(r_1,w_1) = k$. This yields a contradiction since $r_2 \in N(r_1) - N(w_1)$ and $w_2 \in N(w_1) - N(r_1)$ implying $N[v]$ is not neighborhood orderable. ▮

If v does not dominate $G[V_i]$ then the nearest neighbor lemma can be applied so that $N(v) \cap V_i$ is reduced to the set NN(v). Observe that $x \sim y$ in $G[V_i]$ for all pairs $x, y \in$ NN(v). This follows from the fact that $N(v)$ is neighborhood compatible in $G[V_i]$ and by definition of $x, y \in$ NN(v), neither $N[x] \subset N[y]$ nor $N[y] \subset N[x]$ in $G[V_i]$. If v does dominate $G[V_i]$ then $N(v) \cap V_i = V_i$, hence, $G[V_i]$ is complete since v is simple in $G[V_i]$. Hence, $x \sim y$ in $G[V_i]$ for all pairs $x, y \in N(v) \cap V_i$.

It follows that we need a procedure for deciding between each pair $x, y \in N(v) \cap V_i$ when $x \sim y$ in $G[V_i]$ but $x \nsim y$ in G. Let x and y be a pair of vertices satisfying these conditions. This implies the existence of $u \notin V_i$ such that $u \in N(x) - N(y)$ or $u \in N(y) - N(x)$. Assume the former without loss of generality and let $V_i \cap V(T_i(u)) = \{r_u\}$, that is, r_u is the root of the connected component of T_i containing u. Observe that r_u is strictly closer to x than y. Let w be the most distant vertex from r_u on $P(r_u,u)$ in $T_i(u)$ such that wy is an edge of G. We use the structure of $T_i(u)$, which is known, to eliminate either x or y as a candidate for v's root.

If $vw \notin E$ then x cannot be v's root since, if it were, $d_T(v,w) \leq d_T(w,y) = k$ implying $vw \in E$.

If $vw \in E$ then y cannot be v's root since, if it were, $d_T(w,y) = k < d_T(v,w)$ implying $vw \notin E$.

The above procedure allows us to reduce $N(v) \cap V_i$, in conjunction with the nearest neighbor lemma if v does not dominate $G[V_i]$, to a set root(v) of twins in G.

Since all elements of root(v) are twins in G, they differ at iteration i only by the components of T_i they are contained in. For example, let $G = T^6$, where T is the tree depicted in Figure 3. Vertices 16 and 20 are twins in G yet when 9 is chosen as a leaf, assuming leaves are chosen in the order they are labelled, 16 cannot be 9's root. If 16 were chosen as 9's root then $d_T(3, 9) = 3$ contradicting the fact that 3 and 9 are not adjacent in G. However, 20 can be selected as 9's root.

Define the *root* of a connected component S of T_i to be that unique vertex v such that $S \cap V_i = \{v\}$. In Figure 2, v is the root of $T_i(u)$. We want to determine if two leaves of $T[V_i]$, u and v, can share the same root r. Restated, we want to test the hypothesis that the tree $T_r = T_i(u) \cup T_i(v) \cup \{ru, rv\}$ is a subtree of some kth tree root T of G, where $r \in V_i$ is a potential root for u and v (i.e., $r \in \text{root}(u) \cap \text{root}(v)$). The notions of *internal consistency* and *external consistency* will prove useful in testing this hypothesis:

DEFINITION 5. We say $T_i(u)$ and $T_i(v)$ are *internally consistent at* $r \in V_i$ if T_r is a kth tree root of $G[V(T_r)]$. $T_i(u)$ and $T_i(v)$ are called *externally consistent at* $r \in V_i$ if, for each $u' \in T_i(u)$ and $v' \in T_i(v)$ such that $d_{T_r}(r, u') = d_{T_r}(r, v')$, $N(u') = N(v')$ in $V - V(T_r)$.

LEMMA 6. *Let u and v be leaves of $T[V_i]$ and* root(u) = root(v). *There is a kth tree root T of G containing T_i as a subgraph and with u and v sharing the same root r if and only if $T_i(u)$ and $T_i(v)$ are internally and externally consistent at r.*

Proof. Internal and external consistency of $T_i(u)$ and $T_i(v)$ at r are necessary if they share the root r in some kth tree root of G. We prove the sufficiency of these conditions.

Let u and v be leaves of $T[V_i]$ and let $T_i(u)$ and $T_i(v)$ be internally and externally consistent at r. Without loss of generality, let $T_i(u)$ be no deeper than $T_i(v)$. Let T' be a kth tree root of G containing T_i as a subgraph. Let r be the root of v in T'. Let r' be the root of u in T' and assume $r \neq r'$. Let T be the tree $T' - \{ur'\} \cup \{ur\}$; hence, $T_i(u)$ is "pruned" at r' and then "grafted" at r.

We claim T is a kth tree root of G. To prove this it suffices to show that for each $x \in T_i(u)$ and $y \notin T_i(u)$, $xy \in E$ if and only if $d_T(x, y) \leq k$.

Suppose $y \in T_i(v)$. Since $T_i(u)$ and $T_i(v)$ are internally consistent, $T_i(u) \cap T_i(v) \cup \{ru, rv\}$ is a kth tree root of $G[V(T_i(u) \cup T_i(v) \cup \{r\})]$; hence, $xy \in E$ if and only if $d_T(x, y) \leq k$.

Suppose $y \notin T_i(u) \cup T_i(v)$. Since $T_i(u)$ is no deeper than $T_i(v)$, and $T_i(u)$ and $T_i(v)$ are externally consistent at r, there exists $x' \in T_i(v)$ such that $d_T(r, x') = d_T(r, x)$ and $N(x) = N(x')$ in $V - V(T_i(u) \cup T_i(v))$. It follows that $xy \in E$ if and only if $x'y \in E$ if and only if $d_T(x', y) \leq k$ if

and only if $d_T(x, y) \leq k$. This completes the proof that T is a kth tree root of G. ∎

We can now summarize how a root r of a leaf v is determined. If v dominates $G[V_i]$ then $N(v) \cap V_i = V_i$; hence, $G[V_i]$ is complete. It follows that $x \sim y$ in $G[V_i]$ for all pairs $x, y \in N(v) \cap V_i$. We can then reduce $N(v) \cap V_i$ to root(v) using the tests described above.

If v does not dominate $G[V_i]$, then r is an element of NN(v) by the nearest neighbor lemma. By definition of NN(v) and the fact that v is simple, it follows that $x \sim y$ in $G[V_i]$ for all pairs $x, y \in$ NN(v). We can then reduce NN(v) to root(v) using the tests described above.

The final step is to consider each $r \in$ root(v) such that $T_i(r) - \{r\}$ is nonempty, that is, those r already serving as roots. Lemma 6 is used to determine if r is v's root or not. If no $r \in$ root(v) that already serves as a root is necessarily v's root then arbitrarily pick any other element of root(v) to be v's root. This selection is arbitrary since all such elements are indistinguishable. We note that the same process of root selection is used for leaves in core(G) and for leaves not in core(G).

3.3. *Complexity Analysis*

At each iteration of the algorithm an edge vr is added to the tree being constructed, where v is a leaf of $T[V_i]$ and $r \in V_i$ is v's root. We give a brief analysis of the complexity of leaf and root finding. Let n be the number of vertices in G.

When $G[V_i]$ is not complete, simple vertices of $G[V_i]$ are leaves of $T[V_i]$. We assume a simple elimination ordering of the vertices of G is obtained before the algorithm beings. Simple elimination orderings can be found in $O(n^3)$ time [1, 11].

If $G[V_i]$ is complete but by $V_i \not\subseteq$ core(G) then leaves of $T[V_i]$ are determined using the inductive leaf lemma. Using appropriate data structures, a leaf of $T[V_i]$ can be found in $O(n^2)$ time. If $V_i \subseteq$ core(G) then leaves of $T[V_i]$ can be determined using Lemmas 2 and 3 in $O(n)$ time.

Once the leaf v is determined, the algorithm then finds v's root r. If v does not dominate $G[V_i]$ then NN(v) can be determined in $O(n^2)$ time. If v does not dominate $G[V_i]$ then we do not need to determine NN(v). NN(v) is reduced to the set root(v) in $O(n^2)$. Finally, internal and external consistency checks are made for each vertex in root(v). These checks require $O(n)$ time each, resulting in $O(n^2)$ time to check all potential roots in root(v). Hence, v and r are each determined in $O(n^2)$ time which results in an overall running time of $O(n^3)$ for the algorithm.

4. CLOSEST TREE POWER

We have seen that tree powers can be recognized in polynomial time. An obvious question is whether we can efficiently solve more general problems such as recognizing graphs that are within r edges of being tree powers. We formalize this problem as

Closest kth Tree Power (CTP(k))

Instance: A graph $G = (V, E)$ and a nonnegative integer r.

Problem: Is there a kth tree power $G' = (V, E')$ such that $|E \triangle E'| \leq r$?

Here $E \triangle E'$ is the set $E \cup E' - E \cap E'$ which is the edgewise difference between G and G'.

We show that CTP(k) is NP-complete when $k = 3$, and so NP-complete in general. The technique used can be extended to larger values of k. We begin with some preliminary definitions. If $G = (V, E)$, T is a tree with vertex set V, and k is a positive integer then

$$E_k(T, G) = \{xy: x, y \in V \text{ and } xy \in E \Leftrightarrow d^T(x, y) > k\}$$

is the disparity between T^k and G, that is, $E_k(T, G) = E \triangle E'$, where E' is the edge set of T^k.

M is a *binary dissimilarity matrix* for the set S if M is a symmetric matrix with rows and columns indexed by S such that $M(x, y) \in \{1, 2\}$ for all $x \neq y \in S$ and $M(x, x) = 0$ for all $x \in S$.

T is a 2-*ultrametric* for the set S if T is a rooted tree with leaves labelled by elements of S and edges with weights taken from the set $\{k/2: k \text{ is a positive integer}\}$ such that all leaves are equidistant from the root and there are at most two distinct interleaf distances. An example of a 2-ultrametric for the set $\{a, b, c, d, e, f\}$ is given in Figure 6.

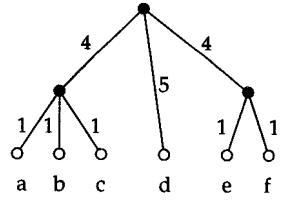

FIG. 6. A 2-ultrametric for $\{a, b, c, d, e, f\}$. Numbers indicate edge weights. Note that the two distinct interleaf distances are 2 and 10.

Given a binary dissimilarity matrix M on S and a 2-ultrametric T on S define

$$E(T, M) = \sum_{x, y \in S} |d_T(x, y) - M(x, y)|.$$

Hence, $E(T, M)$ measures how well T matches the interleaf distances specified by M.

It is clear that CTP(3) is in NP. We reduce the problem FUT to CTP(3) to complete the proof of NP-completeness. FUT has been shown to be NP-complete by Křivánek and Morávek [12].

Fitting Ultrametric Trees (FUT):

Instance: Binary dissimilarity matrix M on S where $|S| \geq 4$ and even, and a positive integer k.

Problem: Is there a 2-ultrametric T on S such that $E(T, M) \leq k$?

To facilitate this, we first establish some structural results.

LEMMA 7. *Let G and T share vertex set V where $A \subseteq V$. If $|A| \geq 2m + 1$ and for each $a \in A$ there exists $v \in V$ such that $av \in E_3(T, G)$ then $|E_3(T, G)| > m$.*

Proof. Let $E' \subseteq E_3(T, G)$ be those pairs containing a vertex of A. Let H be the graph edge induced by E'. Partition A into A_1 and A_2 where $a \in A_1$ if there exists $v \notin A$ such that $av \in E'$. Let $A_2 = A - A_1$. Define $r = |A_1|$. Since each $a \in A$ is on an edge, there are at least $\lceil (2m + 1 - r)/2 \rceil$ edges in $H[A_2]$. This implies that H has at least $r + \lceil (2m + 1 - r)/2 \rceil > m$ edges. It follows from $|E_3(T, G)| \geq |E'|$ that $|E_3(T, G)| > m$. ∎

If T is a tree and U a subset of T's vertices then T_U denotes the minimal subtree of T connecting vertices of U. That is, $v \in T_U$ if and only if v is on a path in T between two vertices of U.

LEMMA 8. *Let G and T share vertex set V where A and B are disjoint subsets of V. Let $G[A]$ and $G[B]$ be complete such that there are no $A \times B$ edges in G and $|A| \geq 2m + 1$, $|B| \geq 2m + 1$, for some $m \geq 0$. If $|E_3(T, G)| \leq m$ then $T_A \cap T_B = \emptyset$.*

Proof. By Lemma 7, there exists $a \in A$ and $b \in B$ such that a and b do not participate in any errors. That is, for all $x \in V$, $ax \in E$ if and only if $d_T(a, x) \leq 3$. Similarly for b.

Consider the path $P_T(a, b)$. For any $a' \in A$ and $b' \in B$, $d_T(a, a')$, $d_T(b, b') \leq 3 < d_T(a, b'), d_T(b, a')$ by the above. This implies $P_T(a, a') \cap P_T(b, b') = \emptyset$. This necessitates $P_T(a_1, a_2) \cap P_T(b_1, b_2) = \emptyset$, for all $a_1, a_2 \in A$ and $b_1, b_2 \in B$. It follows that $T_A \cap T_B = \emptyset$. ∎

LEMMA 9. *If T is a solution to the instance (M, S, k) of* FUT *then there is a solution T' with root r such that for each $s \in S$, $P_{T'}(r, s)$ has exactly two edges and all edges of T' have weight $1/2$.*

Proof. Let r be the root of T. Let the distinct interleaf distances in T be a and b where $a \leq b$. If $v \neq r$ has exactly two neighbors a and b in T then remove v from T and add the edge ab such that $d_T(a, b)$ is preserved. If rs is an edge of T then bisect rs with a new vertex s' such that $d_T(s, s') = a/2$ and $d_T(r, s') = (b - a)/2$. We may also assume that all leaves of T, except possibly r if r is a leaf, are elements of S.

T now has the property that $P_T(r, s)$ has exactly two edges such that the edge incident with r has weight $b/2$ and the edge incident with s has weight $a/2$, for all $s \in S$.

It now suffices to show that all edges of T can be given weight $1/2$ without increasing $E(T, M)$. Define A to be the number of S pairs x, y such that $M(x, y) = 1$ but $d_T(x, y) = a$; define B to be the number of S pairs x, y such that $M(x, y) = 1$ but $d_T(x, y) = b$; define C to be the number of S pairs x, y such that $M(x, y) = 2$ but $d_T(x, y) = a$; and define D to be the number of S pairs x, y such that $M(x, y) = 2$ but $d_T(x, y) = b$. Hence $E(T, M) = A|1 - a| + B|1 - b| + C|2 - a| + D|2 - b|$.

If $a > 2$ then let T' be T except that edges incident with leaves of T get weight 1 and edges incident with r get weight $(b - 2)/2$. It follows that the two distinct interleaf distances of T' are 2 and b; furthermore, $E(T', M) \leq E(T, M)$. It follows that we may assume $1 \leq a \leq 2$.

Similarly, if $b > 2$ then let T' be T except that edges incident with r get weight $(2 - a)/2$. It follows that the two distinct interleaf distances of T' are a and 2; furthermore, $E(T', M) \leq E(T, M)$.

It follows that we may assume $1 \leq a \leq b \leq 2$ which implies $(a, b) \in \{(1, 1), (1, 2), (2, 2)\}$. If both a and b are 1 then let T' be the tree with all $s \in S$ and r adjacent to a vertex r'. Let all edges have weight $1/2$. It follows that $E(T', M) \leq E(T, M)$ and T' satisfies the lemma.

If both a and b are 2 then let T' be the tree with each $s \in S$ adjacent to a unique vertex s' and all s' adjacent to r. Let all edges have weight $1/2$. It follows that $E(T', M) \leq E(T, M)$ and T' satisfies the lemma.

Finally, if $a = 1$ and $b = 2$ then T already satisfies the lemma since all edges of T must have weight $1/2$. ∎

THEOREM 10. FUT $\leq_p$ CTP(3).

Proof. Let (M, S, k) be an instance of FUT. Construct an instance (G, m) of CTP(3) where $G = (V, E)$ as follows. V is partitioned into sets A, B, X, S, and $\{y, z\}$ where all but S induce complete graphs in G, and A, B, and X have $\max\{2k + 1, |S|\}$ vertices. Let $\{y, z\}$ dominate G, A

dominate X, X dominate B, and B dominate S. For vertices $s, s' \in S$ define $ss' \in E$ if and only if $M(s, s') = 1$. No other edges exist. See Figure 7 for a depiction of G. Finally, let $m = k$.

The above construction can obviously be performed in polynomial time. It remains to be shown that (M, S, k) has a solution T_S if and only if (G, m) has a solution T. Suppose T_S is a solution to (M, S, k) with the structural properties permitted by Lemma 9.

We construct a tree with vertex set V as follows: Identify z with the root of T_S. Partition B into B_S and B_0 where each $b \in B_S$ is identified with a vertex v in T_S adjacent to some $s \in S$. For each $s \in S$, create the edge bs where $b \in B_S$ is identified with the vertex v of T_S and vs is an edge of T_S. Let z be adjacent to all $b \in B$. Pick some vertex x_A in X and make all $a \in A$ adjacent to x_A. Let all $x \in X$ be adjacent to y, and to complete the tree, let y be adjacent to z. T is depicted in Figure 8.

Due to the construction of T, pairs in $E_3(T, G)$ are of the form ss' where $s, s' \in S$. Observe that $ss' \in E_3(T, G)$ exactly when

$$\big(d_T(s, s') > 3 \text{ and } ss' \in E\big) \quad \text{or} \quad \big(d_T(s, s') \leq 3 \text{ and } ss' \notin E\big).$$

But $ss' \in E$ if and only if $M(s, s') = 1$. It follows that $|E_3(T, G)| = E(T_S, M) \leq k = m$.

Conversely, let T be a tree with vertex set V such that $|E_3(T, G)| \leq m$. By Lemma 8, $T_A \cap T_B = \varnothing$. Let pq be an edge on the connecting path from T_A to T_B in T where $p \in T_A$. Let $A_1 = \{a \in A\colon d_T(a, p) = 1\}$ and $A' = A - A_1$. Similarly, let $B_1 = \{b \in B\colon d_T(b, q) = 1\}$ and $B' = B - B_1$.

Suppose $|B_1| > m$. It follows that $A_1 = \varnothing$ else $|E_3(T, G)| > m$. If multiple subtrees of T at p have vertices of A then $|E_3(T, G)| > m$ since vertices in distinct subtrees will be at least distance 4 apart. If vertices of A exist in one subtree of T at p then $p \notin T_A$ since, by definition, $p \in T_A$ only if p lies on some path between two vertices of A. By contradiction, $|B_1| \leq m$; hence, $|B'| > m$.

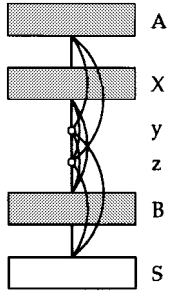

FIG. 7. The graph G constructed in the reduction.

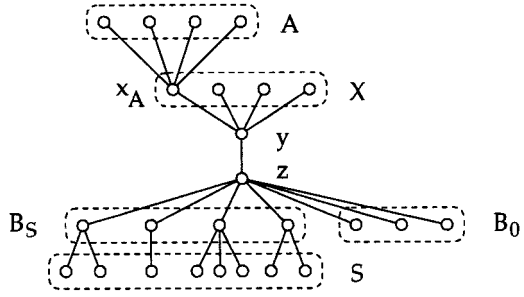

FIG. 8. The tree T constructed in the reduction.

If $|A'| > m$ then for some $x \in X$, either x is not within distance 3 of all vertices in A' or not within distance 3 of all vertices in B' since $d_T(a', b') \geq 5$, for all $a' \in A'$, $b' \in B'$. This implies $|E_3(T, G)| > m$. We conclude that $|A'| \leq m$ and $|A_1| > m$.

It follows from $|A_1| > m$ that $B_1 = \emptyset$, else $|E_3(T, G)| > m$. Consider $x \in X$. Let T_p be the connected component of $T - \{q\}$ containing p. If $x \in T_p - \{p\}$ then x is more than distance 3 from all B' vertices implying $|E_3(T, G)| > m$. Hence, $x \notin T_p - \{p\}$. However, x must be within distance 3 of all A_1 vertices; otherwise $|E_3(T, G)| > m$. It follows that for any $x \in X$ either x is adjacent to q or $x = p$ or $x = q$. In particular, at least $2m - 1$ vertices of X are adjacent to q. It follows that $A' = \emptyset$; otherwise $a' \in A'$ would be more than distance 3 from all X vertices adjacent to q implying $|E_3(T, G)| > m$. Similarly, all B vertices are distance 2 from q; otherwise $|E_3(T, G)| > m$.

If multiple subtrees of T at q have vertices of B then $|E_3(T, G)| > m$ since vertices in distinct subtrees will be distance 4 apart. It follows that all vertices of B are adjacent to a single vertex r, where r is adjacent to q.

If $s \in S$ then s cannot be adjacent to any vertex in $A \cup X \cup \{p, q, r\}$; otherwise s would be within distance 3 of more than m A or X vertices, implying $|E_3(T, G)| > m$. Suppose $s \in S$ is not adjacent to any $b \in B$. It follows that s is more than distance 3 from all but one vertex of B; hence, $|E_3(T, G)| > m$. We conclude that each vertex of S is adjacent to a vertex of B; hence, all vertices of S are distance 2 from r. It follows that $p \in X$ and $q, r \in \{y, z\}$. We define B_S to be those vertices of B adjacent to a vertex s of S. Let $B_0 = B - B_S$.

We can now conclude that T has structure consistent with the tree depicted in Figure 8. In particular, all errors contributing to $E_3(T, G)$ are of the form ss', where $s, s' \in S$. If we let $T_S = T[B_1 \cup S \cup \{r\}]$, where edges receive weight $1/2$, then $E(T_S, M) = |E_3(T, G)| \leq m = k$. ∎

5. CONCLUDING REMARKS

Several open problems present themselves. A bottleneck in our algorithm for tree power recognition is finding simple elimination orders of graphs. An obvious question is whether the algorithm could be altered so that a simple elimination order is not necessary for recognition. In general, can kth tree powers be recognized in time less than $O(n^3)$?

Another interesting area of research is the recognition of graphs which have tree-like roots. For example, what is the recognition complexity of those graphs having roots with $n - 1 + r$ edges, for some constant r?

Finally CTP(2) remains open.

ACKNOWLEDGMENTS

The authors gratefully acknowledge the support of NSERC and the Bellairs Research Institute of McGill University. The authors thank Ryan Hayward, Anna Lubiw, and Jerry Spinrad for helpful suggestions.

REFERENCES

1. R. P. Anstee and M. Farber, Characterizations of totally balanced matrices, *J. Algorithms* **5** (1984), 215–230.
2. R. Balakrishnan and P. Paulraja, Powers of chordal graphs, *Austral. J. Math. Ser. A* **35** (1983), 211–217.
3. P. Damaschke, Distances in cocomparability graphs and their powers, *Discrete Appl. Math.* **35** (1992), 67–72.
4. F. Escalante, L. Montejano, and T. Rojano, Characterization of n-path graphs and of graphs having nth root, *J. Combin. Theory Ser. B* **16** (1974), 282–289.
5. M. Farber, Characterizations of strongly chordal graphs, *Discrete Math.* **43** (1983), 173–189.
6. M. R. Garey and D. S. Johnson, "Computers and Intractibility: A Guide to the Theory of NP-Completeness," Freeman, New York, 1979.
7. D. P. Geller, The square root of a digraph, *J. Combin. Theory Ser. B* **5** (1968), 320–321.
8. M. C. Golumbic, H. Kaplan, and R. Shamir, "On the Complexity of DNA Physical Mapping," Technical Report 271–293, Computer Science Dept., Tel Aviv University, 1993 (to appear in *Adv. in Appl. Math.*).
9. M. C. Golumbic and R. Shamir, Complexity and algorithms for reasoning about time: A graph-theoretical approach, *J. Assoc. Comput. Mach.* **40** (1993), 1108–1133.
10. T. Hamada, Graphs which are n-th powers of trees, *Sci. Univ. Tokyo J. Math.* **15-1** (1979).
11. A. J. Hoffman, A. W. J. Kolen, and M. Sakarovitch, Totally-balanced and greedy matrices, *SIAM J. Alg. Disc. Math.* **6** (1985), 721–730.
12. M. Křivánek and J. Morávek, NP-hard problems in hierarchical-tree clustering, *Acta Inform.* **23** (1986), 311–323.
13. R. Laskar and D. Shier, On powers and centers of chordal graphs, *Discrete Appl. Math.* **6** (1983), 139–147.

14. H. T. Lau, ''Finding a Hamiltonian Cycle in the Square of a Block,'' Ph.D. thesis, School of Computer Science, McGill University, Montreal, 1980.
15. Y.-L. Lin and S. S. Skiena, Algorithms for square roots of graphs, *SIAM J. Discrete Math*. **8** (1995), 99–118.
16. N. Linial, Locality in distributed graph algorithms, *SIAM J. Comput*. **21** (1992), 193–201.
17. A. Lubiw, ''Γ-Free Matrices,'' M.S. thesis, Department of Combinatorics and Optimization, University of Waterloo, Waterloo, 1982.
18. R. Motwani and M. Sudan, Computing roots of graphs is hard, *Discrete Appl. Math*. **54** (1994), 81–88.
19. A. Mukhopadhyay, The square root of a graph, *J. Combin. Theory Ser. B* **2** (1967), 290–295.
20. A. Raychaudhuri, On powers of interval and unit interval graphs, *Congr. Numer*. **59** (1987), 235–242.
21. A. Raychaudhuri, On powers of strongly chordal and circular arc graphs, *Ars Combin*. **34** (1992), 147–160.