On Computing the Distinguishing Numbers of Trees and Forests

Christine T. Cheng
Department of Computer Science
University of Wisconsin–Milwaukee, Milwaukee, WI 53211, USA.
`ccheng@cs.uwm.edu`

Submitted: Apr 28, 2005; Accepted: Jan 18, 2006; Published: Feb 8, 2006
Mathematics Subject Classification: 05C, 68R, 68W

Abstract

Let G be a graph. A vertex labeling of G is *distinguishing* if the only label-preserving automorphism of G is the identity map. The *distinguishing number* of G, $D(G)$, is the minimum number of labels needed so that G has a distinguishing labeling. In this paper, we present $O(n \log n)$-time algorithms that compute the distinguishing numbers of trees and forests. Unlike most of the previous work in this area, our algorithm relies on the combinatorial properties of trees rather than their automorphism groups to compute for their distinguishing numbers.

1 Introduction

The notion of distinguishing numbers came about because of the following recreational problem of Rubin's [11]: suppose a professor has a set of n keys on a circular key ring that are indistinguishable to the naked eye. To tell them apart, he attaches a colored marker on each key. What is the fewest number of colored markers needed so he can distinguish the keys from each other? The answer is quite surprising – it is 3 when $n \in \{3, 4, 5\}$, but drops down to 2 when $n \geq 6$. The answer is dependent on the fact that the keyholder was circular. If, for example, the keys were suspended from a straight rod then it is not hard to see that two colors suffice for all $n \geq 2$. This observation motivated Albertson and Collins [2] to generalize the original problem to arbitrary graphs. The vertices of a graph represented the keys and its edges indicate how the keys are connected to each other; hence, the keys on a circular key ring corresponded to C_n while the keys on a straight rod correspondeded to P_n. They asked the following question: given a graph G, what is the minimum number of colors needed to distinguish the vertices from each other? They defined this number as the *distinguishing number of* G. We define it more formally below.

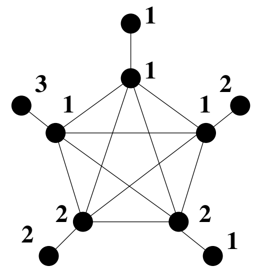

Figure 1: A distinguishing labeling of the graph G_5.

Let G be a graph and u be a vertex of G. An r-labeling of G $\phi : V(G) \rightarrow \{1, 2, \ldots, r\}$ *distinguishes* u if all label-preserving automorphisms of G map u to itself; that is, under the labeling ϕ, u cannot be confused with any other vertex of G. If ϕ distinguishes all the vertices of G, then it is a *distinguishing labeling* of G. Such a labeling is said to break or destroy the symmetries of G because the only member of the automorphism group of (G, ϕ) is the identity map. The *distinguishing number* of G, $D(G)$, is the minimum number of labels needed so that G has a distinguishing labeling.

Given G, Albertson and Collins were interested in the relationship between G's automorphism group, $Aut(G)$, and $D(G)$. It turns out that two graphs that have the same automorphism group need not have the same distinguishing number. For example, let G_n denote the graph on $2n$ vertices obtained from K_n by attaching a pendant vertex to each vertex in K_n (see Figure 1). Clearly, $Aut(G_n) \cong Aut(K_n)$. A labeling of G_n is distinguishing if and only if it assigns different ordered pairs of labels to each pair of vertices consisting of a vertex in K_n and its pendant neighbor. Thus, $D(G_n) = \lceil \sqrt{n} \rceil$ while $D(K_n) = n$. Given a group Γ, Albertson and Collins investigated the possible distinguishing numbers of graphs whose automorphism groups were isomorphic to Γ. For example, they showed that when $Aut(G) \cong \Gamma$ is abelian then $D(G) = 2$ and when it is dihedral then $D(G) \leq 3$. Their work has since been extended by Potanka [10], Russell and Sundaram [12], Tymoczko [13], Klavžar, et al [8], Chan [5, 4, 6], etc. One result that is relevant to us is due to Tymoczko: for every tree T, $D(T) \leq \Delta(T)$, where $\Delta(T)$ is the maximum degree of a vertex in T.

In this paper, we are still interested in determining the distinguishing numbers of a graph family but this time we wish to describe the graph family in terms of its combinatorial structure rather than its automorphism group. In particular, we shall focus on the family of acyclic graphs which consists of trees and forests. We are not the first ones to do this; for example, the distinguishing numbers of cycles, paths, complete graphs and hypercubes [3, 4] are already known. Unlike any of these graph families, however, there is a large number of trees and forests when the number of vertices is fixed at n and so their distinguishing numbers can range from 1 to n. Nonetheless, we shall show that the distinguishing numbers of acyclic graphs can be computed exactly in $O(n \log n)$ time, where n is the number of vertices in the graph. Our algorithm makes use of the following facts which we shall prove later: (i) every tree T' can be mapped to a rooted tree T so that $D(T') = D(T)$, (ii) there is a recursive formula for computing the number of inequivalent distinguishing k-labelings of T, and (iii) the tree ismorphism algorithm [1] can be modified so that the isomorphic subtrees rooted at the children of each vertex v of T can be identified efficiently. We note that for a general graph G, it is not known if the problem of computing $D(G)$ is polynomially-time solvable or NP-hard. Nonetheless, Russell and Sundaram [12] were able to show that determining if $D(G) > k$ belongs to a class of problems called AM, the set of languages for which there are Arthur-Merlin games (see [9] for definition).

2 Preliminaries

Recall that a permutation $\pi : V(G) \to V(G)$ is an *automorphism* of G if π preserves the adjacencies of G; i.e., $\pi(u)$ is adjacent to $\pi(v)$ if and only if u is adjacent to v for every pair of vertices u, v in G. The *automorphism group* of G, $Aut(G)$, consists of all the automorphisms of G. Additionally, the permutation π is an automorphism of the labeled graph (G, ϕ) if π not only preserves the adjacencies of G but the labels of G as well. In other words, $\phi(v) = \phi(\pi(v))$ for each $v \in V(G)$. Similar to the automorphism group of G, $Aut((G, \phi))$ consists of all the automorphisms of (G, ϕ). We define the automorphisms of rooted graphs and rooted labeled graphs in the same way with the extra condition that the root of a graph must always be mapped to itself.

Of particular interest to us are rooted trees. Let T be a rooted tree and v be a vertex of T. We denote its root as $r(T)$, and the subtree of T rooted at v as T_v. Below, we state some properties of automorphisms of T.

Proposition 2.1. *Let T be a rooted tree, $\pi \in Aut(T)$, and v be a vertex of T. The following are true:*

a. π maps the ancestry of v (i.e., its parent $p(v)$, its grandparent $p(p(v))$, etc.) generation by generation, to the ancestry of $\pi(v)$.

b. $T_v \cong T_{\pi(v)}$.

The proposition follows directly from the fact that π preserves the adjacencies of T. We note that it holds true as well if π is an automorphism of (T, ϕ), where ϕ is some labeling of T.

Next, we show that given an unrooted tree T' we can construct a rooted tree T such that $D(T') = D(T)$. A vertex of a graph is a *center* if, among all the vertices of the graph, its maximum distance to any vertex is the least. It is well known that a tree either has one center (i.e., it is *unicentral*) or has two adjacent centers (i.e., it is *bicentral*), and that its center(s) can be determined in linear time. Thus, if T' has a unique center, simply let T be a copy of T'; otherwise, let T be the tree formed by appending a new vertex to the two centers of T' and deleting the edge between the two old centers of T'. In both cases, T has a unique center which we designate as its root $r(T)$.

Lemma 2.2. $D(T') = D(T)$.

Proof: Suppose $D(T') = k$ and ϕ' is a distinguishing k-labeling of T'. Let ϕ be the k-labeling for T where $\phi(v) = \phi'(v)$ if $v \in V(T) \cap V(T')$ and $\phi(v) = 1$ otherwise. Let us now prove that ϕ must be a distinguishing labeling of T as well. Let $\pi \in Aut((T, \phi))$. Then consider the mapping π' where $\pi'(v) = \pi(v)$ for each $v \in V(T')$. Since π maps $r(T)$ to itself, π' is a mapping from $V(T')$ to itself. Moreover, because π preserves the adjacencies of the vertices in T, π' must do so as well for the vertices in T'. In particular, if T' is bicentral, π' maps its two centers to themselves because π maps the children of $r(T)$ to themselves. And, finally, because π preserves the labels of the vertices in (T, ϕ) then π' does so as well for (T, ϕ'). Hence, π' is an automorphism of (T', ϕ'). If π is not the identity map for $V(T)$ then π' is not either for $V(T')$ which leads to a contradiction since ϕ' is a distinguishing k-labeling of T'. Thus, ϕ is a distinguishing k-labeling of T so $D(T) \leq D(T')$. By the same argument, we can show that every distinguishing k-labeling of T' can be transformed into a distinguishing k-labeling of T and so $D(T') \leq D(T)$. The lemma follows. □

Based on the above lemma, we can now restrict our attention to computing the distinguishing numbers of rooted trees.

3 Distinguishing labelings of rooted trees

In this section, we give a characterization of the distinguishing labelings of rooted tree T, and show how we can determine the value of $D(T)$ based on the characterization.

Suppose ϕ is a *distinguishing* labeling of T and v a vertex of T. It must be the case that ϕ, when restricted to T_v, is also distinguishing. In addition, if v and w are distinct children of $r(T)$ and $T_v \cong T_w$, ϕ must label the subtrees *differently*; i.e., (T_v, ϕ) and (T_w, ϕ) must be non-isomorphic. Otherwise, as shown below, (T, ϕ) would have a non-trivial automorphism. We prove in the following lemma that these two conditions are not only necessary but also sufficient conditions for ϕ to be a distinguishing labeling of T.

Theorem 3.1. *Let T be a rooted tree and $CH(T)$ be the set containing all of $r(T)$'s children. Suppose ϕ is a labeling of T, then ϕ is distinguishing if and only if these two conditions hold:*

i. For each v in $CH(T)$, ϕ when restricted to T_v is distinguishing.
ii. For distinct vertices v and w in $CH(T)$, if $T_v \cong T_w$, then $(T_v, \phi) \ncong (T_w, \phi)$.

Proof: Suppose ϕ is a labeling for T and (T_v, ϕ) has a non-trivial automorphism π for some $v \in CH(T)$. Then T has a non-trivial automorphism π', where $\pi'(z) = \pi(z)$ if z is part of the subtree T_v and $\pi'(z) = z$ if z is not in the subtree T_v. Similarly, suppose for distinct vertices $v, w \in CH(T)$, $(T_v, \phi) \cong (T_w, \phi)$. If μ is a label-preserving isomorphism from (T_v, ϕ) to (T_w, ϕ), then T has a non-trivial automorphism μ' where $\mu'(z) = \mu(z)$ if $z \in V(T_v)$, $\mu'(z) = \mu^{-1}(z)$ if $z \in V(T_w)$ and $\mu'(z) = z$ if z is not in the subtrees T_v and T_w. Hence, we have shown that if ϕ does not satisfy conditions *i* or *ii* of the lemma, (T, ϕ) will have a non-trivial automorphism; i.e., ϕ is not a distinguishing labeling.

Suppose the two conditions of the lemma are satisfied by ϕ but ϕ is not distinguishing. There must exist an automorphism of (T, ϕ), say π, and vertex x whose distance from $r(T)$ is a small as possible such that $\pi(x) \neq x$. If x and $\pi(x)$ have distinct parents, from Proposition 2.1(a), π must map $p(x)$ to $p(\pi(x))$ violating the assumption that the distance of x from $r(T)$ is as small as possible. Thus, $p(x) = p(\pi(x))$. Additionally, from Proposition 2.1(b), $(T_x, \phi) \cong (T_{\pi(x)}, \phi)$.

If $p(x) \neq r(T)$, choose u in $CH(T)$ so that $T_{p(x)}$ is a subgraph of $T(u)$. Since ϕ when restricted to $T_{p(x)}$ is not distinguishing, ϕ when restricted to T_u is also not distinguishing.

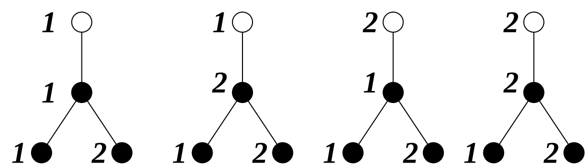

Figure 2: An example of four inequivalent distinguishing labelings of the same tree where the unshaded vertex is the root.

Condition *i* of the lemma is violated. If $p(x) = r(T)$, x and $\pi(x)$ belong to $CH(T)$. Since $(T_x, \phi) \cong (T_{\pi(x)}, \phi)$, condition *ii* of the lemma is volated. But ϕ satisfies both conditions of the lemma; hence, the non-trivial automorphism π of (T, ϕ) does not exist and so ϕ must be distinguishing. $\square$

Suppose ϕ and ϕ' are distinguishing labelings of G. We shall say that the labelings are *equivalent* if $(G, \phi) \cong (G, \phi')$. Figure 2 shows four inequivalent labelings of the same rooted tree all of which are distinguishing. Let $L(G, k)$ denote the set of all distinguishing k-labelings of G. We are interested in $D(G, k)$, the number of equivalence classes in $L(G, k)$. For example, when G is a single node, $D(G, k) = k$. Clearly, $D(G) = \min\{k : D(G, k) > 0\}$.

Theorem 3.2. *Let T be a rooted tree and $\mathcal{T}$ be the set that contains all the subtrees of T whose roots are children of $r(T)$. Suppose $\mathcal{T}$ has exactly g distinct isomorphism classes of subtrees where the jth isomorphism class consists of m_j copies of the rooted tree T_{u_j}; i.e., $\mathcal{T} = m_1 T_{u_1} \cup m_2 T_{u_2} \cup \ldots \cup m_g T_{u_g}$. Then*

$$D(T, k) = k \prod_{j=1}^{g} \binom{D(T_{u_j}, k)}{m_j}.$$

Proof: To create a distinguishing k-labeling for T, we need to assign a label from $\{1, 2, \ldots, k\}$ to $r(T)$, and, according to Theorem 3.1, a distinguishing k-labeling to each copy of T_{u_j} in $\mathcal{T}$ no two of which belong to the same equivalence class in $L(T_{u_j}, k)$ for $j = 1, \ldots, g$. Now suppose ϕ and ϕ' are two distinguishing k-labelings of T. When are they equivalent? It must be the case that (a) $\phi(r(T)) = \phi'(r(T))$ and (b) there is a permutation π_j of $\{1, 2, \ldots, m_j\}$ such that $(T_{u_j,i}, \phi) \cong (T_{u_j,\pi_j(i)}, \phi')$ for $i = 1, \ldots, m_j$ for each j. In other words, for each j, the k-labelings of the m_j copies of T_{u_j} under ϕ and under ϕ' belong to the same m_j equivalence classes in $L(T_{u_j}, k)$. It is straightforward to verify that these two conditions are sufficient as well to guarantee that ϕ and ϕ' are equivalent. This means that equivalence classes of $L(T, k)$ are completely determined by (a) the label of $r(T)$ and (b) the set whose elements are the m_j equivalence classes of $L(T_{u_j}, k)$ that contain the distinguishing labelings of the m_j copies of T_{u_j} for $j = 1, \ldots, g$. Since there are k ways to label $r(T)$, and $\binom{D(T_{u_j},k)}{m_j}$ ways to pick a set of m_j equivalence classes from $L(T_{u_j})$ for $j = 1, \ldots, m$, our result follows. $\square$

The following corollary is immediate.

Corollary 3.3. *For the rooted tree T, $D(T) = k^*$ where $k^* = \min\{k : D(T_{u_j}, k) \geq m_j, \forall j = 1, \ldots, g\}$.*

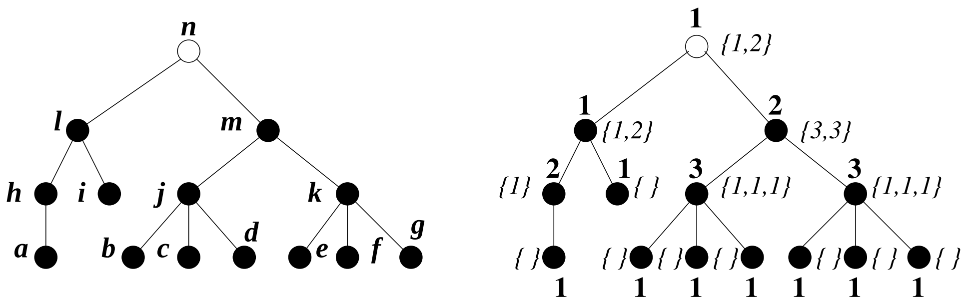

Figure 3: An example of how FIND_ISOMORPH will construct $L(v)$ and $l(v)$ for each vertex v of the graph on the left.

FIND_ISOMORPH(T)

do BFS and construct $B_j = \{v : d(r(T), v) = j\}$ for $j = 0, \dots, h$.

for each $v \in V(T)$
    $l(v) \leftarrow 0$, $L(v) \leftarrow \emptyset$
for $j = h$ to 0
    sort $L(v)$ for each $v \in B_j$
    sort the lists in $\{L(v) : v \in B_j\}$ in lexicographic order
    for each $v \in B_j$
        $l(v) \leftarrow$ rank of $L(v)$ in the sorted list (where ties are not broken)
        add $l(v)$ to $L(p(v))$
return(l, L)

Figure 4: The pseudocode for FIND_ISOMORPH. At the end of this algorithm, two vertices y and z of B_j will have the same label if and only if T_y and T_z are isomorphic.

3.1 Computing $D(T)$

Before we can apply the formula in Theorem 3.2 recursively, we must be able to identify which of the subtrees rooted at each vertex v of T are isomorphic. A brute force approach would be to run the tree isomorphism algorithm [1] on every pair of these subtrees and group together all the isomorphic subtrees. A more efficient way, however, is to simply modify the tree isomorphism algorithm (where we essentially apply the algorithm to just one tree instead of two) so that the problem can be resolved in two sweeps of T. We call our algorithm FIND_ISOMORPH(T); an example and its pseudocode can be found in Figures 3 and 4. In the first sweep of T, run a breadth-first-search (BFS) from $r(T)$ to construct sets B_0, $B_1, \dots B_h$ where B_j contains all vertices that are distance j from $r(T)$, and h is the height of the T. Note that if $v \in B_j$ then all its children are in B_{j+1}. In the second sweep, all vertices v are assigned a label $l(v)$ and a list $L(v)$ which will contain the labels of v's children. Initialize $L(v)$ to the empty set for each vertex v. Start from B_h and move up one level at a time. At each step j, sort $L(v)$ for each $v \in B_j$. Then lexicographically order the set $\{L(v), v \in B_j\}$. Finally, let $l(v)$ be equal to the rank of $L(v)$ in the ordering where ties are not broken. Add $l(v)$ to the list of v's parent $p(v)$.

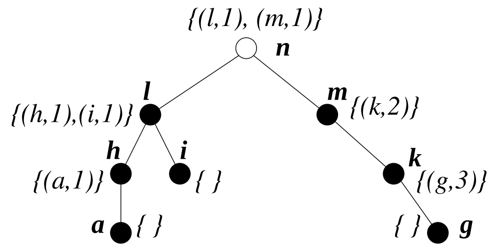

Figure 5: When ESSENTIAL(T, l, L) is applied to the example in Figure 3, the remaining essential vertices are shown above together with their $U(v)$ values.

Lemma 3.4. *Let* y, $z \in B_j$. *At the end of* FIND_ISOMORPH(T), $l(y) = l(z)$ *if and only if the rooted trees* T_y *and* T_z *are isomorphic.*

Proof: We shall show that the lemma is true by induction on j, starting with $j = h$ to $j = 0$. When $j = h$, any vertex in B_h is a leaf. That is, for any $v \in B_h$, T_v consists of a single node, $L(v) = \emptyset$ so $l(v) = 1$. Thus, the lemma is true trivially. Assume that the lemma holds when $j \geq k$ and let $j = k - 1$. If $T_y \cong T_z$, there is some isomorphism that maps y to z and subtrees rooted at y's children to subtrees rooted at z's children. Since y and z's children belong to B_k, by our assumption, the lists $L(y)$ and $L(z)$ are exactly the same. Consequently, they have the same rank in $\{L(v) : v \in B_j\}$ so the algorithm will make $l(y) = l(z)$. On the other hand, if $l(y) = l(z)$, there is a one-to-one correspondence, α, from the children of y to the children of z that preserves the labels. That is, for every child y_i of y, $l(y_i) = l(\alpha(y_i))$. And since y_i and $\alpha(y_i)$ belong to B_k, by assumption, $T_{y_i} \cong T_{\alpha(y_i)}$. If we combine the isomorphism that map the subtrees rooted at y's children to the subtrees rooted at z's children and map y to z, then we have an isomorphism from T_y to T_z. By induction, the lemma holds. □

In Theorem 3.2, we showed that to compute $D(T, k)$, it is necessary and sufficient to know the values of $D(T_{u_j}, k)$ and T_{u_j}'s multiplicity for $j = 1, \ldots g$. We shall say that these u_j's are *essential vertices* of T. But to know these $D(T_{u_j}, k)$'s, we need to also consider the non-isomorphic subtrees rooted at the children of u_j for each j as well. By transitivity, these children of u_j are also essential vertices of T, etc. Thus, we need to only consider a set of essential vertices, $V^E(T)$, of T so that once $D(T_u, k)$ and the multiplicity of T_u is computed for each $u \in V^E(T) \cap B_j$, then $D(T_{u'}, k)$ for each $u' \in V^E(T) \cap B_{j-1}$ can be computed. In algorithm ESSENTIAL(T, l, L) (see Figures 5 and 6), we implement our approach to extract such a set of essential vertices for T.

The set V_j will contain the essential vertices in B_j while the set $U(v)$ is a list that will consist of the ordered pairs (u_j, m_j) defined for T_v in Theorem 3.2. A single sweep of T is performed starting at the only essential vertex of V_0, $r(T)$. At iteration j, for each $v \in V_{j-1}$, the sorted list $L(v)$ is examined. If w's label appears in $L(v)$, we shall let $next(w)$ denote the vertex whose label appears after $l(w)$ in $L(v)$. One child per label together with its multiplicity is added to $U(v)$; this same child is added to V_j. The algorithm ends when the essential vertices in V_{h-1} are examined.

Lemma 3.5. *At the end of* ESSENTIAL(T, l, L), $V^E(T) = \bigcup_{j=0}^{h} V_h$.

ESSENTIAL(T, l, L)

for $j = 0$ to h
    $V_j \leftarrow \emptyset$
$V_0 \leftarrow \{r(T)\}$
for each $v \in V(T)$
    $U(v) \leftarrow \emptyset$
for $j = 1$ to h
    for each $v \in V_{j-1}$ and v not a leaf   /* Construct $U(v)$ */
        let w be the child of v such that $l(w)$ is the first label in $L(v)$
        $count \leftarrow 1$
        while $w \neq nil$
            if $l(w) = l(next(w))$
                $count \leftarrow count + 1$
            else
                add $(w, count)$ to $U(v)$ and w to V_j
                $count \leftarrow 1$
            $w \leftarrow next(w)$
return($U, V_0, V_1, \ldots, V_h$)

Figure 6: The pseudocode for ESSENTIAL(T, l, L). The set V_j contains the essential vertices in B_j.

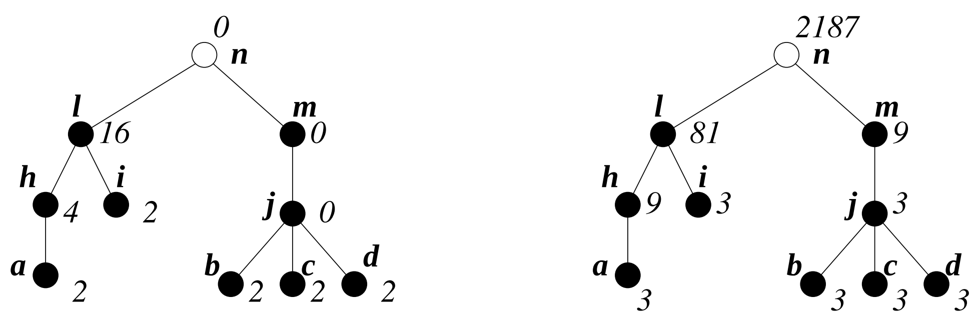

Figure 7: The numbers on the nodes of each tree correspond to $value(v)$ in EVALUATE. For the left tree, k was set to 2; on the right tree, k was set to 3. Since $value(r(T))$ on the left tree is 0, while that of the right tree is positive, we conclude that $D(T) = 3$.

FIND_DIST_TREE(T)

$(l, L) \leftarrow$ FIND_ISOMORPH(T)
$(U, V_0, \ldots, V_h) \leftarrow$ ESSENTIAL(T, l, L)
$left \leftarrow 1$
$right \leftarrow n$
while $right - left > 1$ do
    $k \leftarrow \lceil (left + right)/2 \rceil$
    if EVALUATE($T, U, V_0, \ldots, V_h, k$) > 0
        $right \leftarrow k$
    else
        $left \leftarrow k + 1$
if EVALUATE($T, U, V_0, \ldots, V_h, left$) > 0
    $k \leftarrow left$
    else $k \leftarrow right$
return(k)

EVALUATE($T, U, V_0, \ldots, V_h, k$)

for each leaf v in T
    $value(v) \leftarrow k$
$j \leftarrow h - 1$
while $j \geq 0$ do
    for $v \in V_j$ and v not a leaf
        $value(v) = k \prod_{u_j:(u_j, m_j) \in U(v)} \binom{value(u_j)}{m_j}$
    $j \leftarrow j - 1$
return($value(r(T))$)

Figure 8: The pseudocode algorithm FIND_DIST_TREE(T). Procedure EVALUATE determines the value of $D(T, k)$. The main body of the algorithm performs a binary search to determine the smallest k such that $D(T, k) > 0$.

Once a set of essential vertices has been constructed, we can now compute for $D(T, k)$. We initialize the value for each leaf to k since leaves have exactly k inequivalent distinguishing labelings using k labels. Then starting at V_{h-1}, we apply the formula in Theorem 3.2 one level at a time until we reach the root. If the resulting value is positive, then $D(T) \geq k$. Since $1 \leq D(T) \leq n$, where $n = |V(T)|$, we initially set k to $\lceil (n+1)/2 \rceil$ and perform a binary search to find the smallest k so that $D(T, k) > 0$. We provide an example in Figure 7 and describe our algorithm FIND_DIST_TREE(T) in Figure 8.

Theorem 3.6. *Let T be a rooted tree on n vertices.* FIND_DIST_TREE(T) *computes $D(T)$ correctly in $O(n \log n)$ time.*

Proof: The correctness of FIND_DIST_TREE(T) follows immediately from Theorem 3.2 and Corollary 3.3. To analyze its runtime, assume T has n vertices. In FIND_ISOMORPH(T), the first sweep of T is just a breadth-first search and so takes $O(n)$ time. In the second sweep of T, at iteration j, two types of sorting are done: (i) for each $v \in B_j$, $L(v)$ is sorted, and (ii) the lists in $\{L(v) : v \in B_j\}$ are ordered lexicographically. Now, the labels in each $L(v)$ range from 1 to $|B_{j+1}|$ and $\sum_{v \in B_j} |L(v)| = |B_{j+1}|$. By carefully implementing bucket sort, all the $L(v)$'s can be sorted in $O(|B_{j+1}|)$ time (see exercise C.4.15 of [7]). Similarly, using radix sort, ordering the lists in $\{L(v) : v \in B_j\}$ can be done in $O(|B_{j+1}|)$ time (see pp. 80–84 in [1]). Assigning each $v \in B_j$ a rank takes $O(|B_j|)$ time. Hence, the second sweep of T takes $\sum_{j=0}^{h} O(|B_j| + |B_{j+1}|) = O(n)$ time. Therefore, FIND_ISOMORPH(T) takes $O(n)$ time.

In ESSENTIAL(T, l, L), the list $L(v)$ for each $v \in \cup_{i=0}^{h} V_i$ is examined. But $|L(v)| = deg(v)$ so the runtime of ESSENTIAL(T, l, L) is $O(n)$.

Finally, in the main body of FIND_DIST_TREE(T), there are at most $O(\log n)$ calls to procedure EVALUATE to determine the smallest k such that $D(T, k) > 0$. From the formula in Theorem 3.2, the number of arithmetic operations needed to evaluate $D(T_v, k)$ is proportional to $\sum_{u_j : (u_j, m_j) \in U(v)} m_j$, which equals $deg(v)$. And since all vertices in T may be essential vertices, it takes EVALUATE $O(n)$ time to compute $D(T, k)$. Hence, FIND_DIST_TREE(T) computes $D(T)$ in $O(n \log n)$ time. $\square$

4 Distinguishing Numbers of Forests

Suppose G is a graph with g connected components: $G_1, G_2, \ldots, G_g$. By applying the same arguments we made for Theorem 3.1, we have the following lemma:

Lemma 4.1. *Let G be a graph whose g connected components are $G_1, G_2, \ldots, G_g$. Let ϕ be a labeling of G. Then ϕ is distinguishing if and only if the following two conditions hold:*

i. (G_i, ϕ) *is distinguishing for* $i = 1, \ldots, g$.
ii. *If* $G_i \cong G_j$, *then* $(G_i, \phi) \ncong (G_j, \phi)$ *for every pair of* $i, j \in \{1, \ldots, g\}$.

Let us now consider the case when the graph is a forest F'. Suppose the connected components of F' have g isomorphism classes where the jth isomorphism class contains m_j copies of T'_j; i.e., $F' = m_1 T'_1 \cup m_2 T'_2 \cup \ldots \cup m_g T'_g$. By Lemma 4.1, the trees in the jth isomorphism class must be distinguished. The fewest number of labels that can accomplish this is k_j where $k_j = \min\{k : D(T'_j, k) \geq m_j\}$. Hence, the smallest k such that a k-labeling exists that distinguishes *all* g ismorphism classes is $\max\{k_j, j = 1, \ldots, g\}$. We have proved the following theorem:

Theorem 4.2. *Let F' be a forest whose components have g isomorphism classes where the jth isomorphism class contains m_j copies of T'_j; i.e., $F' = m_1T'_1 \cup m_2T'_2 \cup \ldots \cup m_gT'_g$. Let $k_j = \min\{k : D(T'_j, k) \geq m_j\}$. Then $D(F') = \max\{k_j, j = 1, \ldots, g\}$.*

To compute $D(F')$, we must first identify the isomorphism classes of its connected components. Let F'_1 and F'_2 consist of all the unicentral and bicentral tree components of F respectively. Transform each tree component into a center-rooted tree as in the previous section. Then transform each F'_i into a rooted tree, F_i, by creating a new vertex which is designated as the root of F_i and appending all the centers of the trees in F'_i to this vertex. Finally, run FIND_ISOMORPH(F_i) for $i = 1, 2$. The following must be true.

Lemma 4.3. *Suppose T'_1 and T'_2 are tree components of F'. Then $T'_1 \cong T'_2$ if and only if*

i. the trees belong to the same F'_i and

ii. in FIND_ISOMORPH(F_i)*, the roots of T_1 and T_2 (i.e., their rooted versions) are assigned the same labels (i.e., $l(r(T_1)) = l(r(T_2))$).*

Proof: If $T'_1 \cong T'_2$, they either are both unicentral or both bicentral and so must belong to the same F'_i. Furthermore, their rooted versions, T_1 and T_2 must also be isomorphic. Now, the roots of these trees are children of $r(F_i)$. According to Lemma 3.4, if $T_1 \cong T_2$ and their roots lie on the same level, FIND_ISOMORPH(F_i) assigns the same label to their roots.

Conversely, if T'_1 and T'_2 belong to the same F'_i, they must be both unicentral or both bicentral. Furthermore, the roots of T_1 and T_2 lie on the same level in F_i. Hence, if $r(T_1)$ and $r(T_2)$ were assigned the same label by FIND_ISOMORPH(F_i), then according to Lemma 3.4 the subtrees $T_{r(T_1)} \cong T_{r(T_2)}$. By the way we obtained T_i from T'_i for $i = 1, 2$, this immediately implies that $T'_1 \cong T'_2$. □

Once the isomorphism classes of the tree components of F are identified then, for each class j, we simply have to use EVALUATE to find k_j and output $\max\{k_j, j = 1, \ldots, g\}$.

Theorem 4.4. *Let F' be a forest with n vertices. Its distinguishing number can be computed in $O(n \log n)$ time.*

Proof: Constructing F_1 and F_2 takes $O(n)$ time. Running FIND_ISOMORPH(F_i) for $i = 1, 2$ takes $O(n)$ time. Determining k_j for each j using binary search and EVALUATE takes $O(n \log n)$ time. Finally, finding the maximum among all the k_j's takes $O(n)$ time. □

5 Conclusion

We have presented an $O(n \log n)$-time algorithm for computing the distinguishing number of a tree. There were two important ingredients in our algorithm: (i) the recursive structure of trees and (ii) an efficient algorithm for determining if two trees are isomorphic. The former enabled us to derive a formula for the number of inequivalent distinguishing k-labelings of a tree; the latter allowed us to identify the isomorphic parts of a tree that needed to be distinguished. We then used this algorithm to compute the distinguishing number of a forest. It would be interesting to determine if similar efficient algorithms exist for other graph families. In particular, we ask – can the distinguishing numbers of planar graphs be computed efficiently?

Acknowledgments

Parts of this work were done while the author was at Johns Hopkins University. She would like to thank Lenore Cowen and Alan Goldman who read and discussed parts of this paper with her.

References

[1] A. Aho, J. Hopcroft, and J. Ullman. *The Design and Analysis of Algorithms*. Addison-Wesley, 1974.

[2] M. Albertson and M. Collins. Symmetry breaking in graphs. *Electronic Journal of Combinatorics*, 3:R18, 1996.

[3] W. Bogstad and L. Cowen. The distinguishing number of the hypercube. *Discrete Mathematics*, 283:29–35, 2004.

[4] M. Chan. The distinguishing number of the augmented cube and hypercube powers. Submitted.

[5] M. Chan. The distinguishing number of the direct product and wreath product action. To appear in the *Journal of Algebraic Combinatorics*.

[6] M. Chan. The maximum distinguishing number of a group. To appear in the *Electronic Journal of Combinatorics*.

[7] M. Goodrich and R. Tamassia. *Algorithm Design*. John Wiley and Sons, Inc., 2001.

[8] S. Klavžar, T. Wong, and X. Zhu. Distinguishing labelings of group action on vector spaces and graphs. To appear in the *Journal of Algebra*.

[9] J. Köbler, U. Schöning, and J. Torán. *The Graph Isomorphism Problem: Its Structural Complexity*. Birkhäuser, 1993.

[10] K. Potanka. Groups, graphs and symmetry-breaking. Master's thesis, Virginia Polytechnic Institute and State University, 1998.

[11] F. Rubin. Problem 729. *Journal of Recreational Mathematics*, 11:128, 1979.

[12] A. Russell and R. Sundaram. A note on the asymptotics and computational complexity of graph distinguishability. *Electronic Journal of Combinatorics*, 5:R23, 1998.

[13] J. Tymoczko. Distinguishing numbers for graphs and groups. *Electronic Journal of Combinatorics*, 11(1):R63, 2004.