Distinguishing Trees in Linear Time *

Antoni Lozano† Mercè Mora‡ Carlos Seara§

15th April 2011

Abstract

A graph is said to be $d$-distinguishable if there exists a $d$-labeling of its vertices which is only preserved by the identity map. The distinguishing number of a graph $G$ is the smallest number $d$ for which $G$ is $d$-distinguishable. We show that the distinguishing number of trees and forests can be computed in linear time, improving the previously known $O(n \log n)$ time algorithm.

1 Introduction

Let $G$ be a connected graph with $n$ vertices. A $d$-labeling of $G$ is a total function $\phi : V(G) \rightarrow \{1, 2, \ldots, d\}$. We say that $\phi$ distinguishes $G$ if $G$ has no label-preserving automorphism different from the identity map. In this case, we say that $\phi$ is a distinguishing $d$-labeling of $G$. Such a labeling is said to break or destroy the symmetries of $G$. The distinguishing number of $G$, $D(G)$, is the minimum number $d$ of labels needed so that $G$ has a distinguishing $d$-labeling. A graph $G$ having a distinguishing $d$-labeling is said to be $d$-distinguishable.

Distinguishing numbers were first introduced by Albertson and Collins [2]. The parameter can be thought of as a measure of the symmetry of a graph, i.e., if $G$ and $G'$ have the same number of vertices but $D(G) > D(G')$, then $G$ is more symmetric than $G'$ because more colors are needed to destroy its automorphisms than those of $G'$.

It is not known if the problem of computing $D(G)$ is polynomially-time solvable or NP-hard. Russell and Sundaram [16] showed that determining if $D(G) > d$ belongs to the class AM, i.e., the set of languages for which there are Arthur-Merlin games. However, when $G$ is restricted to certain graph families such as cycles, hypercubes, acyclic graphs, and planar graphs, the problem can be solved efficiently [2, 3, 4, 5, 6, 9]. See [1, 4, 7, 8, 10, 13, 14, 15, 17] for other works on distinguishing number problems.

For the computation of the distinguishing number of trees and forests with $n$ vertices, Cheng [9] and Arvind and Devanur [3] presented an $O(n \log n)$ time algorithm which uses a binary search to compute the distinguishing number of a tree. Improving this time complexity

---

†Dpt. Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, Spain, antoni@lsi.upc.edu.
‡Dpt. Matemàtica Aplicada II, Universitat Politècnica de Catalunya, Spain, merce.mora@upc.edu.
§Dpt. Matemàtica Aplicada II, Universitat Politècnica de Catalunya, Spain, carlos.seara@upc.edu.

Graphs in this paper are finite, undirected and simple. The vertex-set and the edge-set of a graph $G$ are denoted by $V(G)$ and $E(G)$, respectively. The order of $G$ is the number of its vertices, denoted by $|V(G)|$. For more terminology we follow [18].
was our main motivation for the design of an optimal linear-time algorithm for computing the
distinguishing number of trees and forests, and this is the main result of our paper.

In Section 2 we will focus on the design of a linear time algorithm for rooted trees which
is based on properties proved by Cheng [9] and follow her notation whenever possible. As
a consequence of our result, we show in Section 3 that there are linear-time algorithms for
computing the distinguishing numbers of trees and forests. Finally, in Section 4 we conjecture
logarithmic factor improvements for other graph classes.

2 Distinguishing Rooted Trees

2.1 Preliminaries

We start with some notation. By $\text{Aut}(G)$ we denote the automorphism group of a graph $G$. As
usual, two graphs $G$ and $H$ are isomorphic, denoted by $G \cong H$, if there is a permutation $\pi :$
$V(G) \to V(H)$ which preserves adjacencies, that is, $\{u, v\} \in E(G)$ if and only if $\{\pi(u), \pi(v)\} \in$
$E(H)$ for any $u, v \in V(G)$.

Given a graph $G$ and a labeling $\phi$ of $G$, we represent the corresponding labeled graph by
$(G, \phi)$. In this case, $\text{Aut}(G, \phi)$ consists of all the automorphisms of $\text{Aut}(G)$ which preserve the
labeling $\phi$, that is, $\pi \in \text{Aut}(G, \phi)$ if and only if $\phi \in \text{Aut}(G)$ and $\phi(v) = \phi(\pi(v))$. We also
consider the extension of isomorphism to labeled graphs. Given two labeled graphs $(G, \phi)$ and
$(H, \varphi)$, we say that they are isomorphic if there is a permutation $\pi : V(G) \to V(H)$ which
preserves adjacencies as defined above, but also preserves labels, that is, $\phi(v) = \varphi(\pi(v))$ for
each $v \in V(G)$. In this case, we write $(G, \phi) \cong (H, \varphi)$. Two distinguishing $d$-labelings $\phi$ and $\varphi$
of a graph $G$ are said to be equivalent if $(G, \phi) \cong (G, \varphi)$.

Given a rooted tree $T$, we denote its root by $r(T)$. We also denote by $T_u$ the subtree of $T$
rooted at vertex $u$ of $T$, and we call components of $T$ to all the subtrees $T_u$ of $T$ where $u$ is a
child of $r(T)$. Any isomorphism between two rooted trees $T_1$ and $T_2$ must map $r(T_1)$ into $r(T_2)$.

In the same way, any automorphism of a rooted tree must map the root into itself.

As we will see, the distinguishing number of a rooted tree can be computed using a recursive
formula. Call $D(T, d)$ to the number of inequivalent distinguishing $d$-labelings of a rooted tree $T$.
For instance, if $T$ is a rooted tree consisting of a single path of $k$ vertices, then $D(T, d) = d^k$,
but if $T$ is a full binary tree of (any) depth $k$, then $D(T, 2)$ is just 2 since we can assign two
possible labels to the root while the rest of the tree has a unique distinguishing 2-labeling up to
isomorphism. Clearly, for any graph $G$,

$$D(G) = \min\{d \mid D(G, d) > 0\},$$

and, therefore, computing $D(T, d)$ for any $d$ is all that is needed to compute $D(T)$ for a rooted
tree $T$. We also observe the following relation between $D(T)$ and $D(T, d)$.

Proposition 1. For any rooted tree $T$ and for any $d \geq D(T)$, it holds $D(T, d) \geq d$.

Proof. For any $d \geq D(T)$, $T$ is clearly $d$-distinguishable, so suppose $\phi$ is some $d$-distinguishing
labeling of $T$. Changing the label assigned by $\phi$ to the root gives $d$ inequivalent labelings of $T$,
that is,

$$\phi_i(u) = \begin{cases} 
    i, & \text{if } u = r(T) \\
    \phi(u), & \text{if } u \neq r(T).
\end{cases}$$
for every $i \in \{1, \ldots, d\}$. To see that the labelings are inequivalent, suppose on the contrary that
two such labelings, say $\phi_i$ and $\phi_j$ (where $1 \leq i < j \leq d$), are equivalent. Then, there would be
a mapping $\pi$ between the labeled copies $(T, \phi_i)$ and $(T, \phi_j)$ such that $\phi_i(r(T)) = \phi_j(\pi(r(T))) =
\phi_j(r(T))$, where the last equality holds because any isomorphism between labeled copies of a
rooted tree must map the root to itself. But then, by definition of $\phi_i$ and $\phi_j$, we get $i = j$,
contradicting our assumption that $i < j$.

The existence of the inequivalent $d$-labelings $\phi_1, \ldots, \phi_d$ for $T$ implies that $D(T, d) \geq d$. 

Now, we consider the recursive formula developed in [3] and [9] which counts the number
of inequivalent distinguishing $d$-labelings of a rooted tree, and how to derive the distinguishing
number from it in two ways.

Proposition 2. ([9], Th. 3.2, Cor. 3.3, Th. 4.2) Let $T$ be a rooted tree and $\mathcal{T}$ be the set of the
components of $T$. Suppose that $T$ has exactly $g$ distinct isomorphism classes of trees where the
ith isomorphism class consists of $m_i$ copies of the rooted tree $T_{u_i}$; i.e., $T = m_1T_{u_1} \cup m_2T_{u_2} \cup
\cdots \cup m_gT_{u_g}$. Then,

1. $D(T, d) = d \prod_{i=1}^{g} \left( \frac{D(T_{u_i}, d)}{m_i} \right)$.

2. $D(T) = \min\{d \mid \forall i \in \{1, \ldots, g\} \quad D(T_{u_i}, d) \geq m_i\}$.

3. $D(T) = \max\{\min\{d \mid D(T_{u_i}, d) \geq m_i\} \mid 1 \leq i \leq g\}$.

By Proposition 2(1), in order to compute $D(T)$ for a rooted tree $T$, it is enough to know
a list of values $\{(m_1, u_1), \ldots, (m_g, u_g)\}$, where the degree of $r(T)$ equals $\sum_{i=1}^{g} m_i$ and for each
pair $(m_i, u_i)$, $u_i$ is a child of $r(T)$ and $m_i$ is the multiplicity of the isomorphic copies of $T_{u_i}$, which
appear as components of $T$. We take advantage of the fact that Cheng [9] shows a method to
compute exactly this information. We will assume that a new procedure called COMPUTE-LIST,
given a rooted tree $T$, returns the list $\{(m_1, u_1), \ldots, (m_g, u_g)\}$ defined above (in [9], this can be
accomplished by calling procedure FIND-ISOMORPH, then ESSENTIAL and, finally, taking
the first output). Cheng [9] shows that this can be done in linear time.

2.2 Linear Time Algorithm

We will describe the procedures used in our main algorithm. In the first place, procedure
COLORINGS($T$, $d$), given a rooted tree $T$ and a constant $d$, computes $D(T, d)$ in linear time. We
will not detail the algorithm here since it is already described by Cheng [9] and by Arvind and
Devanur [3]. This procedure is called EVALUATE in [9] and INEQUIV in [3].

Note that, according to Proposition 2(2), given a rooted tree $T$ of order $n$, $D(T)$ can already
be computed by making calls to COLORINGS($T$, $d$) for different values of $d$, $1 \leq d \leq n - 1$.
Since each call takes linear time, using binary search in $d$, it is possible to find $D(T)$ in time
$O(n \log n)$, as it is argued in [3] and [9]. This is precisely the common method in [3, 9] which
works in time $O(n \log n)$. In order to lower it to an overall linear time, we need to carefully call
this procedure for the subtrees of $T$ only when it is needed. To do so, we need the information
contained in the list $\{(m_1, u_1), \ldots, (m_g, u_g)\}$, which will be obtained in linear time by calling to
a procedure COMPUTE-LIST, as stated at the end of Subsection 2.1.

Our algorithm is the following (see Figure 1 for an example).
Distinguishing($T$)

**Input:** A rooted tree $T$ with $n$ vertices

**Output:** $D(T)$

1: if $|V(T)| = 1$ then
2: return 1
3: else
4: $col \leftarrow 0$
5: $\{(u_1, m_1), \ldots, (u_g, m_g)\} \leftarrow$ COMPUTE-LIST($T$)
6: for $i = 1$ to $g$ do
7: $d \leftarrow$ Distinguishing($T_{u_i}$)
8: if $d < m_i$ then
9: while COLORINGS($T_{u_i}, d$) < $m_i$ do
10: $d \leftarrow d + 1$
11: end while
12: end if
13: $col \leftarrow \max\{col, d\}$
14: end for
15: return $col$
16: end if

Figure 1: Each vertex is labeled with the distinguishing number of the rooted subtree it defines.

**Theorem 1.** (Correctness) Given a rooted tree $T$ as input, procedure Distinguishing returns $D(T)$.

**Proof.** We show it by induction on the order of $T$. If $T$ has only one vertex, Distinguishing returns 1 at line 2, which is correct. Suppose now that the order of $T$ is $n > 1$. For each $i$, procedure Distinguishing makes a recursive call on the subtree $T_{u_i}$ in line 7. By induction hypothesis, we can assume that the result is $d = D(T_{u_i})$. Now we can distinguish two cases:

1. If $d \geq m_i$, then by Proposition 1, $D(T_{u_i}, d) \geq d \geq m_i$.
2. If $d < m_i$, then after the while loop in lines 9–11, $d$ is the smallest value satisfying $D(T_{u_i}, d) \geq m_i$.

In any case, the value of $d$ at line 13 for a given $i$ is the smallest value such that $D(T_{u_i}, d) \geq m_i$. Since the value $col$ returned by Distinguishing is the maximum of such values for $1 \leq i \leq g$, it must equal $D(T)$ by Proposition 2(3).
**Theorem 2.** (Complexity) Procedure **Distinguishing** works in $O(n)$ time.

**Proof.** Let $T$ be the rooted tree of order $n$ given as input and let $T_{u_1}, \ldots, T_{u_g}$ be its different components up to isomorphism, with orders $n_1, \ldots, n_g$, respectively. Note that since $m_i$ is the number of isomorphic copies for $T_{u_i}$, it holds that

$$n = 1 + \sum_{i=1}^{g} m_i \cdot n_i. \quad (1)$$

Since line 5 takes linear time, let $a$ stand for a constant such that $an$ bounds the time required by procedure **Distinguishing** to execute lines 1–5 and return $col$ at line 15. Additionally, the for loop between lines 6–14 does some work in constant time, say $b$, plus a maximum of $m_i$ calls to **Colorings**($T_{u_i}, d$). The reason why **Colorings** is called at most $m_i$ times for a given $i$ is that the while loop in lines 9–11 is never executed for $d = m_i$ since, by Proposition 1, $D(T_{u_i}, m_i) \geq m_i$ and the condition of the loop would not hold. Now, since **Colorings** works in linear time, we can bound the overall work done between lines 9 and 11 by $cn_i$ for a constant $c$. Now, if we denote the running time of **Distinguishing** by $R(n)$, we have

$$R(n) \leq an + \sum_{i=1}^{g} (b + c(m_i - 1)n_i + R(n_i)). \quad (2)$$

Let $e$ be a constant such that $e \geq a + b + c$. We will show by induction that the running time of **Distinguishing**($T$) is bounded by $an + e(n - 1)$. For $n = 1$, we have $R(n) \leq a = an + e(n - 1)$.

In the general case $n > 1$, we unfold the summation in Equation (2) and get:

$$R(n) \leq an + bg + c \sum_{i=1}^{g}((m_i - 1)n_i) + \sum_{i=1}^{g}(an_i + e(n_i - 1)),$$

where the last term comes from the induction hypothesis. Now,

$$R(n) \leq an + bg + (e - a) \sum_{i=1}^{g}((m_i - 1)n_i) + a \sum_{i=1}^{g} n_i - eg + e \sum_{i=1}^{g} n_i$$

$$\leq an + bg + a \sum_{i=1}^{g} (m_i - 1)n_i + a \sum_{i=1}^{g} n_i - eg + e(n - 1)$$

$$\leq an - a \sum_{i=1}^{g} (m_i - 1)n_i + a \sum_{i=1}^{g} n_i + e(n - 1)$$

$$\leq an - a(n - 1) + a \sum_{i=1}^{g} n_i + e(n - 1)$$

$$= a(1 + \sum_{i=1}^{g} n_i) + e(n - 1) \leq an + e(n - 1),$$

where both Equation (1) and the fact that $e \geq a + b + c$ have been used. Therefore, we conclude that $R(n)$ is $O(n)$. \qed
3 Distinguishing Trees and Forests

There is an easy way to transform the problem of computing $D(T)$ for a general tree $T$ into the one of computing $D(T')$ for a rooted tree $T'$. This can be done using the concept of tree center, as is done in [3] and [9]. A center of a tree $T$ is a vertex $v$ such that the maximal distance of $v$ to the other vertices is minimized. It is well known that every tree has either one center or two adjacent centers. In the first case, the tree can already be considered a rooted tree with root at its center, while in the second case, a new vertex can be inserted between the centers and then used as its root. As mentioned in [3] and [9], this transformation can be done in linear time.

**Proposition 3.** Given a tree $T$, it is possible to compute a rooted tree $T'$ in linear time such that $D(T) = D(T')$.

As a direct consequence of Proposition 3 and Theorems 1 and 2, we conclude the following.

**Corollary 1.** The distinguishing number of a tree with $n$ vertices can be computed in $O(n)$ time.

In the case of forests, we use the transformation from Cheng's paper [9], which is as follows. Suppose that $F$ is a forest and define the following trees $T_1$ and $T_2$. In the first place, create two vertices $v_1$, and $v_2$ which will act as the respective roots of $T_1$ and $T_2$, respectively. In the second place, transform each connected component of $F$ into a rooted tree as indicated before Proposition 3. Since this can be done in two ways, depending on whether the original tree is unicentral or bicentral, call $F_1$ ($F_2$) to the set of rooted trees obtained from the unicentral (bicentral) trees in $F$ in the way indicated before Proposition 3. Finally, join $v_j$ to the roots of all trees in $F_j$, for $1 \leq j \leq 2$. We can now state the following regarding the above construction.

**Proposition 4.** Given a forest $F$, it is possible to compute two rooted trees $T_1$ and $T_2$ in linear time such that $D(F) = \max\{D(T_1), D(T_2)\}$.

**Proof.** It is well known that the centers of a tree can be computed in linear time. So, given a forest $F$ with $n$ vertices, the above transformation into trees $T_1$ and $T_2$ can be done in time $O(n)$. Moreover, since no automorphism of $F$ can map a unicentral tree into a bicentral tree or vice versa, nontrivial automorphisms of $F$ induce separate nontrivial automorphisms in $T_1$ and $T_2$. Suppose that $d = \max\{D(T_1), D(T_2)\}$. Then, $d$ labels are enough to break symmetries in both $T_1$ and $T_2$, that is, there must be two $d$-labelings $\phi_1$, $\phi_2$ such that both $\text{Aut}(T_1, \phi_1)$ and $\text{Aut}(T_2, \phi_2)$ are trivial. Then, one can define a distinguishing $d$-labeling $\phi$ of $F$: Given a vertex $u$ of $F$, if $u \in T_1$, set $\phi(u) = \phi_1(u)$ and, otherwise, set $\phi(u) = \phi_2(u)$. In case $\text{Aut}(F, \phi)$ was nontrivial, then one of $\text{Aut}(T_1, \phi_1)$ or $\text{Aut}(T_2, \phi_2)$ would be nontrivial. Thus, $\text{Aut}(F, \phi)$ must be trivial, and $D(F) \leq d$. But we can discard the case when $D(F) = d' < d$ since it would make it possible to use the distinguishing $d'$-labeling of $F$ to define a distinguishing $d'$-labeling of $T_1$ and $T_2$, contradicting the definition of $d$ as the maximum between $D(T_1)$ and $D(T_2)$. Therefore, $D(F) = \max\{D(T_1), D(T_2)\}$. \square

Now, it is clear that our algorithm Distinguishing from Section 2 can be used twice in combination with Proposition 4 to yield the following result.

**Corollary 2.** The distinguishing number of a forest with $n$ vertices can be computed in $O(n)$ time.
4 Conclusions and applications

We have shown that the distinguishing number of trees and forests can be computed in linear time, improving the previously known $O(n \log n)$ time algorithm. We believe that our algorithmic technique in Section 2 can be applied to improve by a logarithmic factor (caused by a binary search in the last step of the algorithms) the complexities of computing distinguishing numbers and distinguishing chromatic numbers of the following graph classes: (1) the distinguishing number of (i) planar graphs computed by Arvind et al. [3, 4] and (ii) interval graphs computed by Cheng [10]; (2) the distinguishing chromatic number (due to Collins and Trenk [11], see also [12]) of: (i) trees computed by Cheng [10] and (ii) interval graphs computed by Cheng [10].

References


