# A Linear Time Algorithm for the Role Assignment of Trees

Jeremy Lyle
Clemson University

## 1 Introduction

The concept of role assignments takes its roots from both the application of social network theory and a variation in the theory of graph homomorphisms. Everett and Borgatti, in [EB91], developed role assignments as a way to map a social network so that all individuals of a similar role interact with individuals of different roles in a similar manner. Role assignments also follow naturally as refinements of graph homomorphisms, as evidenced by the definition.

A *role assignment* is a mapping $r$ from an input graph or network, $G_I$, to a set of roles, $G_R$ (a role graph), ie. $r : G_I \to G_R$. For $S \subseteq V$, we define $r(S) = \{r(s) : s \in S\}$. Each role assignment must additionally satisfy the following condition,

$$\forall v \in V(G_I), \quad r(N(v)) = N(r(v)) \tag{1}$$

where $N(v)$ is the open neighborhood of a vertex $v$ in $V(G)$. Note that a role graph $G_R$ may have loops. If a vertex $v \in V(G_r)$ has a loop, then each vertex $u \in r^{-1}(v)$ must be adjacent to a vertex in $r^{-1}(v)$. If a vertex $v \in V(G_r)$ does not have a loop, $r^{-1}(v)$ is an independent set in $G_I$. If $G_R$ is a complete graph (with no loops), then a role assignment is a partition of $G_I$ into independent dominating sets.

As a subset of graph homomorphisms, role assignments first appeared in work by Sailer [Sai78] and by White and Reitz [WR83]. It has reappeared in many places, surfacing again as a particular variety of the generalized $H$-colorings of Kristiansen and Telle, (see [KT00]), and a generalization of fall colorings, partitions of the vertex set of a graph into independent dominating sets, as in [DHH$^+$00].

In general, the task of determining which graphs have a role assignment to a given role graph $G_R$ is a very difficult one. In [RS01], Roberts and Sheng considered the complexity of determining whether a graph has a role assignment to one of the 6 non-isomorphic graphs on two vertices (allowing loops), and showed that for two cases, the question is NP-complete, and

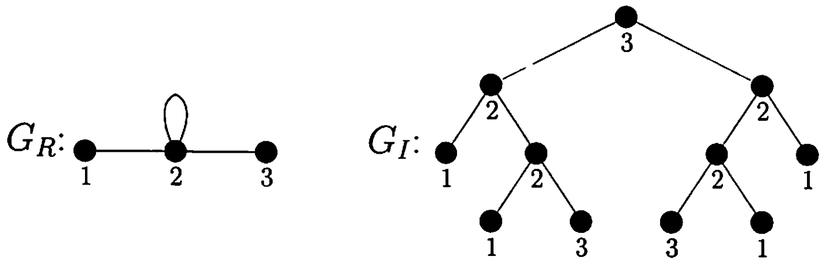

Figure 1: A sample assignment of roles 1, 2, and 3 (the vertices of $G_R$) to the graph $G_I$.

in fact, determining if there is any assignment to some graph on two vertices is an NP-complete question. In [FP05], Fiala and Paulusma further demonstrated the difficulty of finding role assignments by showing that for any connected fixed graph $G_R$ which is not $K_2$, the question of whether an arbitrary graph $G$ has a role assignment to $G_R$ is NP-complete.

Because of the inherent difficulty of this problem, we restrict our attention to the problem of determining role assignments on subclasses of graphs. In [She03], Sheng considered role assignments on the class of triangulated graphs to role graphs with two vertices. In this paper, we will consider the problem of determining a role assignment for any input tree $T$ to a role graph $G_R$.

## 1.1 Role assignments of Trees

We begin by considering the structure of graphs $G_R$, for which there is a role assignment $r$ from some tree $T$ to $G_R$. We note that for any connected graph, there is always a trivial role assignment to a single vertex with a loop, and a trivial identity map. Since every tree is bipartite, it additionally has a role assignment to $K_2$. These three graphs are either trees, or in the case of a single vertex with a loop, a tree with a loop, and we can prove that for a tree $T$, any role graph $G_R$ must be of this form.

**Theorem 1.1.** *Let $T$ be any tree, and $r$ be a role assignment, such that $r : T \rightarrow G_R$, for some role graph $G_R$. Then $G_R$ has no cycles of length greater than or equal to 3.*

*Proof.* We can assume for contradiction that there is some tree $T$ and role assignment $r : T \rightarrow G_R$, such that $G_R$ has a cycle $C_k$ with vertices $c_0, c_1, \ldots, c_{k-1}$, for $k \geq 3$. We can then choose a sequence of vertices from $T$ in the following way. Let $v_0 \in r^{-1}(c_0)$ and $v_{i+1} \in r^{-1}(c_{(i+1) \bmod k})$, such that $(v_i, v_{i+1}) \in E(T)$. We note that such a $v_{i+1}$ must always exist since $r$ is a role assignment, and also, since $T$ is finite, then for some $i$ and $j$ we have $v_i = v_j$, which forms a cycle in $T$, a contradiction. $\square$

This does not preclude $G_R$ from having loops, but every other cycle is forbidden. Therefore, if a tree $T$ has a role assignment to a role graph $G_R$, then $G_R$ is a tree, with the addition of loops on some subset of its vertices.

## 2 Algorithm

In this section, we present an algorithm which, when given an input tree $T$ and a role graph $G_R$, determines if there is a role assignment $r$, such that $r : T \to G_R$. We let $n_T$ be the number of vertices in $T$ and $n_R$ be the number of vertices in $G_R$, and we note that $|E(T)| = n_T - 1$, and $|E(G_R)| \leq 2n_R - 1$ by Theorem 1.1.

For the graph $G_R$, create a set by taking each loop edge, and considering each non-loop edge as two directed edges, and define this set as $\overrightarrow{E}(G_R)$ (See Figure 2 for an example). Then for each vertex $u_j \in V(G_R)$, we construct the set $A_j = \{a_i : e_{a_i} = (u_k, u_j) \in \overrightarrow{E}(G_R)\}$. This set holds the indices of all directed edges whose head is the vertex $u_j$. For the graph $T$, we order the vertices by a post-order traversal of the tree. For each vertex, $v_i$, we construct a boolean array of size $|\overrightarrow{E}(G_R)|$, which we will denote $P_i$, an array representing the possibility that the edge from a vertex to its parent can be assigned to each directed edge $e_j \in \overrightarrow{E}(G_R)$. Additionally, we form the sets $C_i$, which hold the indices of the children of vertex $v_i$ in $T$.

The algorithm processes vertices by working up from the leaves of the tree $T$. At each step, we consider the edge between a vertex $v_i$ and its parent, say $v_p$, and determine the possible edges in $G_R$ that this edge can be mapped to. In order to aid our discussion, we define a *partial role assignment* on a subtree of $T$. We say that the subtree *generated* by a vertex $v_i$, denoted $T\langle v_i \rangle$, is the subtree of descendants of $v_i$. A mapping $r_{i,j}$ is a partial role assignment on $T\langle v_i \rangle$, extendable with edge $e_j = (u_a, u_b) \in \overrightarrow{E}(G_R)$, if

$$\begin{array}{ll} r_{i,j}(N(v)) = N(r_{i,j}(v)) & \text{for } v \in T\langle v_i \rangle - \{v_i\}; \text{ and} \\ N(u_a) - \{u_b\} \subseteq r_{i,j}(N(v_i) - \{v_p\}) \subseteq N(u_a) & \text{where } r_{i,j}(v_i) = u_a \end{array} \tag{2}$$

Now the key point is if there is a partial role assignment on $T\langle v_i \rangle$, extendable with edge $e_j = (u_a, u_b) \in \overrightarrow{E}(G_R)$, then there must be some mapping of the children of $v_i$ to partial role assignments extendable on each of the adjacencies into $u_a$ (with the possible exception of edge $e_j$), and each child must be mapped. The task of constructing and solving this bipartite matching problem is handled by the subroutine MATCHING.

**Procedure 1** MATCHING $(C', A')$

```
  $V_A = \{s_i : i \le |A'|\}$, $V_B = \{t_i : i \le |C'|\}$, $E = \emptyset$, $M^* = \emptyset$
  if $|A'| \le |C'|$ then
    for $k = 1$ to $|V_A|$, $l = 1$ to $|V_B|$ do
      if $P_{c_l}[a_k] = 1$ then
        Add edge $(s_k, t_l)$ to $E$.
    Find a max matching, $M^*$, of $G = (V_A \cup V_B, E)$
```

**Algorithm 1 TRA** (Tree Role Assignments)

```
  INITIALIZATION
  for $i = 1$ to $n_T - 1$ do
    for $j = 1$ to $|\overrightarrow{E}(G_R)|$ do
      For $e_j = (u_{j_1}, u_{j_2})$, let $e_{r_j} = (u_{j_2}, u_{j_1})$
      if every $v_{c_\alpha} \in C_i$, has some $a_j \in A_{j_2}$ such that $P_\alpha[a_j] = 1$ then
        $M^* = \text{MATCHING}(C_i, A_{j_2} - \{r_j\})$.
        if $|M^*| = |A_{j_2} - \{r_j\}|$ then
          $P_i[j] = 1$.
  for $j = 1$ to $n_R$ do
    $M^* = \text{MATCHING}(C_i, A_j)$.
    if $|M^*| = |A_j|$ then
      There is a valid role assignment.
```

At vertex $v_{n_T}$, the root of $T$, we must determine whether there is some role $u_a \in V(G_R)$ which we can assign $v_{n_T}$ so that partial role assignments on the child subtrees of $v_{n_R}$ can be found where one child is assigned to each of the adjacencies of $u_a$ in $G_R$, accomplished by a final matching problem. If a suitable matching is found, then there exists a role assignment from $T$ to $G_R$, and if no matching is found, there is no such role assignment.

## 2.1 Correctness

In order to prove the correctness of the Algorithm **TRA**, we need to show that the array $P_i$ for each vertex in $T_I$ correctly reflects the fact that a partial role assignment either does or does not exist.

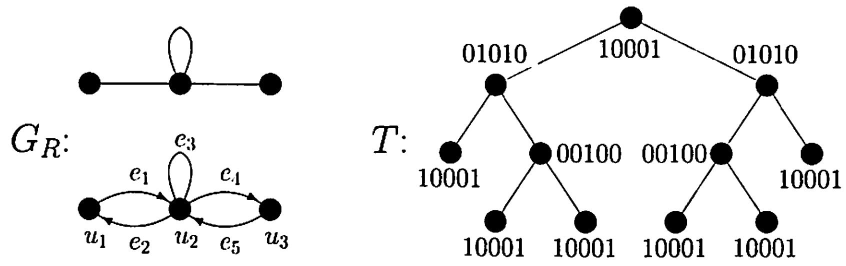

Figure 2: Algorithm execution, with $P_i$ vectors shown for each vertex in $T$, where $A_1 = \{2\}$, $A_2 = \{1, 3, 5\}$, and $A_3 = \{4\}$.

**Lemma 2.1.1.** *For a processed vertex $v_i \in V(T)$ and $e_j = (u_a, u_b) \in \vec{E}(G_R)$, then $P_i[j] = 1$ if and only if there exists a partial role assignment on $T\langle v_i \rangle$, extendable with edge $e_j$.*

*Proof.* We proceed by induction on $i$. First, consider the case where $v_i$ is a leaf. If $P_i[j] = 1$, then $|A_j| - 1 \leq |C_i| = 0$, implying $u_a$ is a leaf in $G_R$, and we can define a partial role assignment which simply takes $v_i$ to $u_a$. On the other hand, if there is a partial role assignment, then $|N(v_i)| - 1 = 0 \geq |N(u_a)| - 1$, so $|N(u_a)| = 1$, which implies $u_a$ is a root, and the conditions to set $P_i[j] = 1$ are trivially satisfied.

Now, we assume that $P_k[j] = 1$ if and only if there exists a partial role assignment on $T\langle v_k \rangle$, extendable with edge $e_j$ for all vertices $v_k$ such that $k < i$. If $P_i[j] = 1$, then a matching exists, which implies that there is a matched assignment of edges $e_{a_j}$ into $u_a$ (possibly excluding $e_j$) to children of $v_i$, say $v_{c_i}$ such that $P_{c_i}[a_j] = 1$. Using our induction assumption, we can then define the partial role assignment $r_{i,j}$ to be $r_{i,j}(v) = r_{c_i, a_j}(v)$ where $v \in T\langle v_{c_i} \rangle$, and $r_{i,j}(v_i) = u_a$. On the other hand, if a partial role assignment exists, then there must be some partial role assignments on each of the subtrees generated by the children, and by the inductive assumption, they must have $P_{c_i}[a_j] = 1$. This then creates a matching of the correct cardinality, and $P_i[j] = 1$. □

Finally, we just need to prove that the final step in the algorithm correctly determines a role assignment for the entire tree.

**Theorem 2.1.** *Algorithm* **TRA** *determines whether a role assignment exists from an input tree $T$ to a role graph $G_R$.*

*Proof.* From Lemma 2.1.1, each child of the root node contains a list of partial role assignments. A suitable matching implies that we can extend these partial role assignments to a full role assignment, where every the mapping of each vertex satisfies the condition of a role assignment, and furthermore, since $G_R$ is connected, every role is assigned. Additionally, a possible role assignment implies there are partial role assignments on each of the children of the root node, which will yield a suitable matching. □

## 2.2 Complexity Analysis

The dominant factor involved in the Algorithm **TRA** is the work involved in solving a bipartite matching problem for every combination of a directed edge of $G_R$ and a vertex of $T$. At each step, we consider the graph $G = (V_A \cup V_B, E)$, where $|V_A| = |A_j|$ or $|A_j| - 1$ and $|V_B| = |C_i|$, with $|A_j| - 1 \leq |C_i|$. This allows us to consider this problem as an *unbalanced* bipartite graph, in which one bipartition is larger, first considered in [AOST94]. In particular, Kao et al. in [KLST01] give a simple adaptation of a bipartite matching algorithm by Gabow and Tarjan [GT89], which yields a running time of $O(\sqrt{n_s} m \log n_s)$ where $n_s$ is the cardinality of the smallest bipartition for the bipartite matching problem.

**Theorem 2.2.** *For any graph $G_R$ and any tree $T$, Algorithm* **TRA** *runs in $O\left(n_T n_R^{2.5} \log n_R\right)$*

*Proof.* The work involved in the initialization for the algorithm is at most $O(n_T n_R)$, the initialization of each array $P_i$, and this is overshadowed by the rest of the of the algorithm, solving a matching problem for every combination of an edge in $G_R$ and a vertex in $T$. Therefore, we get that the running time of the algorithm, $RT$ (**TRA**), is given by,

$$\begin{aligned} RT(\mathbf{TRA}) &= O\left(\sum_{i}^{n_T} \sum_{\substack{e_j=(u_{j_1},u_{j_2}) \\ e_j \in \vec{E}(G_R)}} RT(\text{MATCHING}(C_i, A_{j_2}))\right) \\ &= O\left(\sum_{i}^{n_T} \sum_{j}^{n_R} d(u_j) RT(\text{MATCHING}(C_i, A_j))\right) \end{aligned}$$

From [KLST01], the unbalanced bipartite matching problem can be solved in $O(\sqrt{n_s} m \log n_s)$ time, where $n_s \leq |A_j| \leq 2d(u_j)$, and $m \leq |A_j||C_j| \leq 2d(u_j)d(v_i)$. Since constructing the bipartite graph takes only $O(d(v_i)d(u_j))$ time, we get that $RT(\text{MATCHING}(C_i, A_j)) = O(d(v_i)d(u_j)\sqrt{d(u_j)} \log d(u_j))$.

Substituting this into the running time of Algorithm **TRA**, we get

$$\begin{aligned} RT(\mathbf{TRA}) &= O\left(\sum_{i}^{n_T}\sum_{j}^{n_R} d(u_j)\left(d(v_i)d(u_j)^{1.5}\log d(u_j)\right)\right) \\ &= O\left(\sum_{i}^{n_T} d(v_i)\left(n_R^{2.5}\log n_R\right)\right) = O\left(n_T n_R^{2.5}\log n_R\right) \end{aligned}$$

□

**Corollary 2.1.** *Algorithm* **TRA** *runs in linear time for any fixed graph* $G_R$.

## 2.3 Conclusions

In this paper, we have presented the Algorithm **TRA** , which for any fixed graph $G_R$ will decide in linear time if an input tree $T$ has a role assignment to $G_R$. Additionally, if we allow $G_R$ to be considered as part of the input, Algorithm **TRA** runs in $O\left(n_T n_R^{2.5}\log n_R\right)$. We note at this point, that with a slight modification, we can recover a role assignment if one exists by saving the matchings at each point. Also, if we consider input graphs $G_I$ where $G_I$ is tree with loops on some subset of its vertices, then Theorem 1.1 still holds, and in fact gives an exact characterization of the role graphs which can result. The algorithm can be modified by simply requiring vertices in $G_I$ with loops to be assigned to roles in $G_R$ with loops, and a loop in $G_I$ is present, relaxing the condition that a child subtree must be assigned to the loop edge in $G_R$.

## References

[AOST94] Ravindra K. Ahuja, James B. Orlin, Clifford Stein, and Robert E. Tarjan. Improved algorithms for bipartite network flow. *SIAM J. Comput.*, 23(5):906–933, 1994.

[DHH+00] J. E. Dunbar, S. M. Hedetniemi, S. T. Hedetniemi, D. P. Jacobs, J. Knisely, R. C. Laskar, and D. F. Rall. Fall colorings of graphs. *J. Combin. Math. Combin. Comput.*, 33:257–273, 2000. Papers in honour of Ernest J. Cockayne.

[EB91] Martin G. Everett and Steve Borgatti. Role colouring a graph. *Math. Social Sci.*, 21(2):183–188, 1991.

[FP05] Jiří Fiala and Daniël Paulusma. A complete complexity classification of the role assignment problem. *Theoret. Comput. Sci.*, 349(1):67–81, 2005.

[GT89] Harold N. Gabow and Robert E. Tarjan. Faster scaling algorithms for network problems. *SIAM J. Comput.*, 18(5):1013–1036, 1989.

[KLST01] Ming-Yang Kao, Tak-Wah Lam, Wing-Kin Sung, and Hing-Fung Ting. An even faster and more unifying algorithm for comparing trees via unbalanced bipartite matchings. *J. Algorithms*, 40(2):212–233, 2001.

[KT00] Petter Kristiansen and Jan Arne Telle. Generalized $H$-coloring of graphs. In *Algorithms and computation (Taipei, 2000)*, volume 1969 of *Lecture Notes in Comput. Sci.*, pages 456–466. Springer, Berlin, 2000.

[RS01] Fred S. Roberts and Li Sheng. How hard is it to determine if a graph has a 2-role assignment? *Networks*, 37(2):67–73, 2001.

[Sai78] Lee D. Sailer. Structural equivalence: Meaning and definition, computation and application. *Social Networks*, 1(1):73–90, 1978.

[She03] Li Sheng. 2-role assignments on triangulated graphs. *Theoret. Comput. Sci.*, 304(1-3):201–214, 2003.

[WR83] Douglas R. White and Karl P. Reitz. Graph and semigroup homomorphisms on networks of relations. *Social Networks*, 5(2):193–234, 1983.