Computing role assignments of chordal graphs*

Pim van 't Hof[1,**], Daniël Paulusma[1,**] and Johan M. M. van Rooij[2]

[1]School of Engineering and Computing Sciences, Durham University,
Science Laboratories, South Road, Durham, DH1 3LE, England.
{pim.vanthof,daniel.paulusma}@durham.ac.uk
[2]Department of Information and Computing Sciences, Universiteit Utrecht,
PO Box 80.089, 3508 TB Utrecht, The Netherlands.
jmmrooij@cs.uu.nl

Abstract. In social network theory, a simple graph G is called k-role assignable if there is a surjective mapping that assigns a number from $\{1,\ldots,k\}$, called a role, to each vertex of G such that any two vertices with the same role have the same sets of roles assigned to their neighbors. The decision problem whether such a mapping exists is called the k-Role Assignment problem. This problem is known to be NP-complete for any fixed $k \geq 2$. In this paper, we classify the computational complexity of the k-Role Assignment problem for the class of chordal graphs. We show that for this class the problem can be solved in linear time for $k = 2$, but remains NP-complete for any $k \geq 3$. This generalizes earlier results by Sheng and answers her open problem.

1 Introduction

All graphs considered in this paper are undirected, finite and simple, i.e., without loops or multiple edges, unless otherwise stated. Given two graphs, say G on vertices $u_1,\ldots,u_n$ and R on vertices $1,\ldots,k$ called *roles*, an *R-role assignment* of G is a vertex mapping $r : V_G \to V_R$ such that the neighborhood relation is maintained, i.e., the roles of the neighbors of each vertex u in G are exactly the neighbors of role $r(u)$ in R. Let $N_G(u)$ denote the set of neighbors of u in the graph G and let $r(S) = \{r(u) \mid u \in S\}$ for any subset $S \subseteq V_G$. The condition that r is an *R-role assignment* of G can be formally expressed as

$$\text{for all } u \in V_G : r(N_G(u)) = N_R(r(u)).$$

An R-role assignment r of G is called a *k-role assignment* of G if $|r(V_G)| = |V_R| = k$. An equivalent definition states that r is a k-role assignment of G if r maps each vertex of G to a positive integer so that $|r(V_G)| = k$ and $r(N_G(u)) = r(N_G(u'))$ for any two vertices u and u' with $r(u) = r(u')$. See Figure 1 for an example.

Role assignments are introduced by Everett and Borgatti [9], who call them role colorings. They originate in the theory of social behavior. The *role graph* R

* An extended abstract of this paper has been presented at FCT 2009.
** This author has been supported by EPSRC (EP/D053633/1).

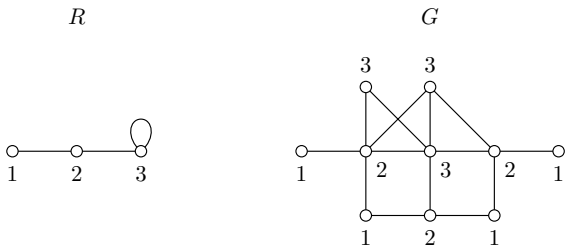

Fig. 1. A role graph R and a graph G with an R-role assignment which is also a 3-role assignment of G (example based on the figure in [24]).

models roles and their relationships, and for a given society we can ask if its individuals can be assigned roles such that relationships are preserved: each person playing a particular role has exactly the roles prescribed by the model among its neighbors. This way one investigates whether large networks of individuals can be compressed into smaller ones that still give some description of the large network. As persons of the same social role may be related to each other, the smaller network can contain loops. In other words, given a *simple* instance graph G on n vertices does there exist a *possibly non-simple* role graph R on $k < n$ vertices in such a way that G has an R-role assignment? From the computational complexity point of view it is interesting to know whether the existence of such an assignment can be decided quickly (in polynomial time). This leads to the following two decision problems.

R-Role Assignment
Input: a simple graph G.
Question: does G have an R-role assignment?

k-Role Assignment
Input: a simple graph G.
Question: does G have a k-role assignment?

Known results and related work. A *graph homomorphism* from a graph G to a graph R is a vertex mapping $r : V_G \to V_R$ satisfying the property that the edge $r(u)r(v)$ belongs to E_R whenever the edge uv belongs to E_G. If for every $u \in V_G$ the restriction of r to the neighborhood of u, i.e., the mapping $r_u : N_G(u) \to N_R(r(u))$, is bijective, we say that r is *locally bijective* [1, 19]. If for every $u \in V_G$ the mapping r_u is injective, we say that r is *locally injective* [10, 11]. If for every $u \in V_G$ the mapping r_u is surjective, r is an R-role assignment of G. In this context, r is also called a *locally surjective* homomorphism from G to R.

Locally bijective homomorphisms, also called graph coverings, have applications in distributed computing [2, 3, 7] and in constructing highly transitive regular graphs [5]. Locally injective homomorphisms, also called partial graph coverings, have applications in models of telecommunication [11] and frequency

assignment [12]. Besides social network theory [9, 21, 23], locally surjective homomorphisms also have applications in distributed computing [8].

The main computational question is whether for every graph R the problem of deciding if an input graph G has a homomorphism of given local constraint to the fixed graph R can be classified as either NP-complete or polynomial time solvable. For locally bijective and injective homomorphisms there are many partial results, see e.g. [11, 19] for both NP-complete and polynomial time solvable cases, but even conjecturing a classification for these two locally constrained homomorphisms is problematic. This is not the case for the locally surjective constraint and its corresponding decision problem R-Role Assignment.

Roberts and Sheng [23] have shown that the k-Role Assignment problem is already NP-complete for $k = 2$. Fiala and Paulusma [13] have shown that the k-Role Assignment problem is also NP-complete for any fixed $k \geq 3$ and classify the computational complexity of the R-Role Assignment problem. Let R be a fixed role graph without multiple edges but possibly with loops. Then the R-Role Assignment problem is solvable in polynomial time if and only if one of the following three cases holds: either R has no edge, or one of its components consists of a single vertex incident with a loop, or R is simple and bipartite and has at least one component isomorphic to an edge. In all other cases the R-Role Assignment problem is NP-complete, even for the class of bipartite graphs [13]. If the instance graphs are trees, then the R-Role Assignment problem becomes polynomial time solvable for any fixed role graph R [14].

A graph is *chordal* if it does not contain an induced cycle of length at least 4. Chordal graphs are also called *triangulated* graphs. This class contains various subclasses such as trees, split graphs and indifference graphs (graphs whose vertices can be assigned some real values such that two vertices are adjacent if and only if their assigned values are sufficiently close). Due to their nice properties, chordal graphs form an intensively studied graph class both within structural graph theory and within algorithmic graph theory. Sheng [24] presented an elegant greedy algorithm that solves the 2-Role Assignment problem in linear time on chordal graphs with at most one vertex of degree 1. She also characterized all indifference graphs that have a 2-role assignment.

Our results and paper organization. In Section 2, we present a linear time algorithm for the 2-Role Assignment problem on chordal graphs. This settles an open problem of Sheng [24]. Unlike the greedy algorithm of Sheng [24], which uses a perfect elimination scheme of a chordal graph with at most one vertex of degree 1, our algorithm works for any chordal graph G by using a dynamic programming procedure on a clique tree decomposition of G. Section 3 contains our second result. Here we prove that, for any fixed $k \geq 3$, the k-Role Assignment problem is NP-complete for chordal graphs. Section 4 contains the conclusions and mentions some open problems.

2 Computing a 2-role assignment in linear time

In this section, we prove the following result.

Theorem 1. *The* 2-Role Assignment *problem can be solved in linear time for the class of chordal graphs.*

We will start by discussing the different 2-role assignments. Following the notation of Sheng [24], the six different role graphs on two vertices are:

$$R_1 := (\{1,2\}, \emptyset)$$
$$R_2 := (\{1,2\}, \{22\})$$
$$R_3 := (\{1,2\}, \{11,22\})$$
$$R_4 := (\{1,2\}, \{12\})$$
$$R_5 := (\{1,2\}, \{12,22\})$$
$$R_6 := (\{1,2\}, \{11,12,22\})$$

These six role graphs are depicted in Figure 2.

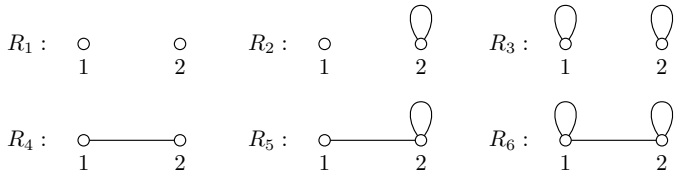

Fig. 2. The six different role graphs on two vertices.

Let G be a chordal graph. If G contains at most one vertex, then G has no 2-role assignment. Suppose $|V_G| \geq 2$. If G only contains isolated vertices, then G has an R_1-role assignment. If G contains at least one isolated vertex and at least one component with at least two vertices, then G has an R_2-role assignment. If G is disconnected but does not have isolated vertices, then G has an R_3-role assignment. Now assume that G is connected and has at least two vertices. If G is bipartite, then G has an R_4-role assignment. If G is not bipartite, then G has a 2-role assignment if and only if G has an R_5-role assignment or an R_6-role assignment.

We claim that we only have to check whether G has an R_5-role assignment. This is immediately clear if G has a vertex of degree 1, as such a vertex must be mapped to a role of degree 1 and R_6 does not have such a role. If G does not have any degree 1 vertices, we use the following result by Sheng [24].

Theorem 2 ([24]). *Let G be a chordal graph with at most one vertex of degree 1 and no isolated vertices. Then G has an R_5-role assignment.*

We will now present a linear time algorithm that decides whether a chordal graph G has an R_5-role assignment and if so outputs an R_5-role assignment of G. From the above, it is clear that this suffices to prove Theorem 1. Our algorithm is to be viewed as being independent from the linear time algorithm of Sheng [24] for computing an R_5-role assignment of a chordal graph with at most one vertex of degree one and no isolated vertices. Before presenting our algorithm we first make some basic observations on chordal graphs.

2.1 On chordal graphs

Let $G = (V, E)$ be a chordal graph. A *clique* in G is a subset $K \subseteq V$ such that $G[K]$ is a complete graph, where $G[K]$ denotes the subgraph of G induced by K. A clique in G is *maximal* if it is not properly contained in any other clique in G. Let $\mathcal{K}$ denote the set of maximal cliques of G. The *clique graph* $C(G)$ of G has as its vertex set $\mathcal{K}$, and two vertices of $C(G)$ are adjacent if and only if the intersection of the corresponding maximal cliques is non-empty. Moreover, every edge K_1K_2 of $C(G)$ is given a weight equal to $|K_1 \cap K_2|$. Chordal graphs can be represented using so-called clique trees, and many different definitions and characterizations of clique trees have appeared in the literature (see for example [6]). We use a characterization due to Bernstein and Goodman [4]: a tree T with vertex set $\mathcal{K}$ is a *clique tree* of G if and only if T is a maximum weight spanning tree of $C(G)$. See Figure 3 for an example of a chordal graph G and a clique tree of G.

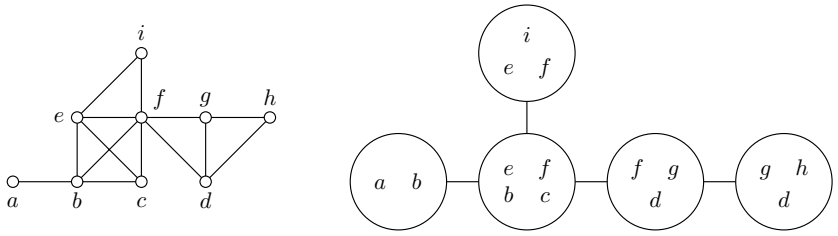

Fig. 3. A chordal graph G (left) and a clique tree T of G.

We refer to a set $K \in \mathcal{K}$ as a *bag* of T. We define the notions *root bag*, *parent bag*, *child bag* and *leaf bag* of a clique tree similar to the notions root, parent, child and leaf of a "normal" tree. If the bag $K_r \in \mathcal{K}$ is the root bag of a clique tree T of G, then we say that T is *rooted at* K_r. A *descendant* of a bag K is a bag K^* such that K lies on the (unique) path from K^* to the root bag K_r in T. Every bag $K \neq K_r$ of a clique tree T has exactly one parent bag K' in T. We say that a vertex $v \in K$ is *given to the parent bag* K' if $v \in K \cap K'$, i.e., if v is both in the child bag K and in the parent bag K'. We say that vertex $v \in K$ *stays behind* if $v \in K \setminus K'$, i.e., if v is in the child bag K but not in the parent bag K'. The characterization of a clique tree given above immediately implies the following observation.

Observation 1 *Let G be a connected chordal graph with at least two maximal cliques. Let $T = (\mathcal{K}, \mathcal{E})$ be a clique tree of G rooted at K_r. At least one vertex of any bag $K \neq K_r$ of T is given to the parent bag of K and at least one vertex stays behind. Moreover, $|K| \geq 2$ for all $K \in \mathcal{K}$.*

It is well-known that a connected graph is chordal if and only if it has a clique tree [18]. We will make use of the following results.

Theorem 3 ([20]). *Let $G = (V, E)$ be a chordal graph. Then $\sum_{K \in \mathcal{K}} |K| = \mathcal{O}(|V| + |E|)$.*

Theorem 4 ([6, 15]). *A clique tree of a connected chordal graph $G = (V, E)$ can be constructed in $\mathcal{O}(|V| + |E|)$ time.*

2.2 An outline of our algorithm

Our algorithm for solving the R_5-Role Assignment problem on chordal graphs takes as input a chordal graph $G = (V, E)$, and either outputs an R_5-role assignment of G, or outputs `NO` if such a role assignment does not exist. If the graph G is disconnected, then the algorithm described below is executed on each of the connected components of G. In that case, the algorithm outputs an R_5-role assignment of G if and only if it found an R_5-role assignment of every connected component of G, and outputs `NO` otherwise. We assume from now on that the input graph G is connected.

The algorithm starts by computing a clique tree T of G, and then executes two phases. . In Phase 1, the algorithm assigns a *label* to every vertex, and decides whether or not G is R_5-role assignable. If so, then the labels of the vertices are used in Phase 2 to determine which *role* must be assigned to each vertex in order to obtain an R_5-role assignment of G.

Phase 1. Decide whether or not G has an R_5-role assignment

In Phase 1, the algorithm processes the bags of T in a "bottom-up" manner, starting with the leaf bags of T, and processing a bag K only after all its child bags have been processed. When processing bag K, the algorithm computes a label $L_K(v)$ for each vertex $v \in K$; this label $L_K(v)$ will be referred to as the *K-label* of v. Initially, each vertex $v \in K$ is assigned a label $L_K(v) = 0$. Thereafter, our algorithm updates the labels of the vertices of this bag in order to maintain information about the possible roles that these vertices can get in a possible R_5-role assignment of G, as well as information about the possible roles of their neighbors. To this end, it uses the labels defined in Table 1. This updating process first generates a new label for v in K based on the labels that v has in the child bags of K; thereafter it updates all labels in K based on the different labels that are now present in K.

Labels of two vertices can be conflicting if they represent information on the possible roles of the vertices that cannot be combined to an R_5-role assignment. For example, no two vertices in a bag can both get label 1, because this would mean each of them must have role 1. Our algorithm first checks if there are any conflicting labels. If so, it outputs `NO`. Otherwise, it updates the labels in order to maintain Invariant 1 below. Here, a *solution* on G is an R_5-role assignment of G. A *partial solution* on a subgraph H of G is a mapping $r' : V_H \to \{1, 2\}$ such that no two adjacent vertices x, y of H have roles that are forbidden by R_5 (i.e., we do not have $r'(x) = r'(y) = 1$), and every vertex x in H not adjacent to a vertex in $G - H$ has neighbors with the roles required by R_5 (at least one neighbor y with role $r'(y) = 2$, and if $r'(x) = 2$, also at least one neighbor z with role $r'(z) = 1$).

Invariant 1 *Let V' be the set of vertices of G that do not belong to any descendant of a bag K'. Then a partial solution on $G[V' \cup K']$ can be extended to a solution on G if and only if it satisfies the constraints given by the labels $L_K(v)$ of the vertices $v \in (K \cap K')$ for every child bag K of K'.*

Recall that K_r is the root of the clique tree. Suppose that at some moment bag K_r is processed. We observe that $V' \cup K' = K_r$ in Invariant 1 if $K' = K_r$. Hence, our algorithm ensures that a partial solution on $G[V' \cup K'] = G[K_r]$ can be extended to a solution on G if and only if it satisfies the constraints given by the labels of the vertices on $K \cap K_r$ for every child bag K of K_r. Our algorithm will now decide if such a partial solution on $G[K_r]$ exists. If so, it finds one and goes to Phase 2. If not, it outputs `NO`.

Phase 2. Produce an R_5-role assignment of G

When Phase 2 starts, we know that an R_5-role assignment exists for G. Now, the algorithm will construct an R_5-role assignment as follows. The partial solution on $G[K_r]$ found at the end of Phase 1 is propagated to a solution of G in a "top-down" manner, processing a bag K only after its parent bag has been processed.

A bag K is processed as follows. Each vertex $v \in K$ that already has been assigned a role at an earlier step in Phase 2 keeps this role; Invariant 1 ensures that such a role satisfies the constraints given by $L_K(v)$. Each vertex in K without a role gets a role. The algorithm does this in a greedy way by considering these vertices one by one and assigning them a role that satisfies the constraints imposed by their labels. This leads to an R_5-role assignment of G.

We now present Phase 1 and Phase 2 in detail. When doing this we show that Invariant 1 is maintained throughout Phase 1. As such we immediately prove that our algorithm is correct.

2.3 Phase 1 in detail

Table 1 shows what labels a vertex v in a bag K can have. We observe that Phase 2 is only executed if G is indeed R_5-role assignable. We implicitly assume this in Table 1 and in the remainder of this section, whenever we write that some vertex gets some role in Phase 2.

Initially, every vertex v in each bag K is assigned the label $L_K(v) = 0$. During an execution of the algorithm, this label $L_K(v)$ will be updated: the arrows in Figure 4 represent all possible transitions between two labels. This figure will be clarified in detail later on. For now, we only note that no arrows point downwards in Figure 4. This corresponds to the fact that labels in a higher level contain more information than labels in a lower level. For example, if a vertex v in bag K has a label 2_2 and one of its neighbors in K gets label 2, then we change the label $L_K(v)$ into 2 before processing the parent bag of K. After all, label 2 contains more information than label 2_2, as label 2 contains the information that at least one neighbor of v will get role 2 in Phase 2.

$L_K(v) = 0$	initial label of every vertex
$L_K(v) = 1$	v must get role 1 in Phase 2
$L_K(v) = 2$	v must get role 2 in Phase 2; it is ensured that v gets a neighbor with role 1 and a neighbor with role 2
$L_K(v) = 1^*$	v belongs to a set $J \subseteq K$ of at least two vertices that are all labeled 1^* because they are given to K from the same child bag in which a vertex with label 2_1 is left behind; exactly one vertex in J must get role 1 (and hence all others must get role 2)
$L_K(v) = 2_1$	v must get role 2 in Phase 2; it is ensured that v gets a neighbor with role 2, but we must still make sure that v gets a neighbor with role 1
$L_K(v) = 2_2$	v must get role 2 in Phase 2; it is ensured that v gets a neighbor with role 1, but we must still make sure that v gets a neighbor with role 2
$L_K(v) = 1\|2$	v may get either role 1 or 2 in Phase 2; in the latter case it is ensured that v gets a neighbor with role 1 and a neighbor with role 2
$L_K(v) = 1\|2_1$	v may get either role 1 or 2 in Phase 2; in the latter case it is ensured that v gets a neighbor with role 2, but we must still make sure that v gets a neighbor with role 1
$L_K(v) = 1\|2_2$	v may get either role 1 or 2 in Phase 2; in the latter case it is ensured that v gets a neighbor with role 1, but we must still make sure that v gets a neighbor with role 2

Table 1. The different labels a vertex v can have.

We will now give a detailed description of the label assignments in Phase 1. At each step of this description, we will prove that these label assignments maintain Invariant 1.

Let K be the bag that is currently being processed. As soon as K is processed, the algorithm deals with the next bag until all bags have been processed. Recall that the order in which this is done is such that a bag is processed only if all its child bags have been processed.

The algorithm distinguishes between the following three phases. Phase 1a deals with the case in which $K \neq K_r$ and K is a leaf bag. Phase 1b deals with the case in which $K \neq K_r$ and K is not a leaf bag. Phase 1c deals with the case in which $K = K_r$. For Phase 1a and 1b, we recall that by Observation 1 at least one vertex in K stays behind, and at least one vertex is given to its parent bag, which we denote by K'.

Phase 1a. Deal with leaf bags

Suppose $K \neq K_r$ is a leaf bag of T. Let v be a vertex that stays behind. The algorithm distinguishes between the cases $|K| = 2$ and $|K| \geq 3$.

Case 1. $|K| = 2$.
In this case, v must have degree 1 in G. Let w be the other vertex of K. By Observation 1, we find that w is given to K'.

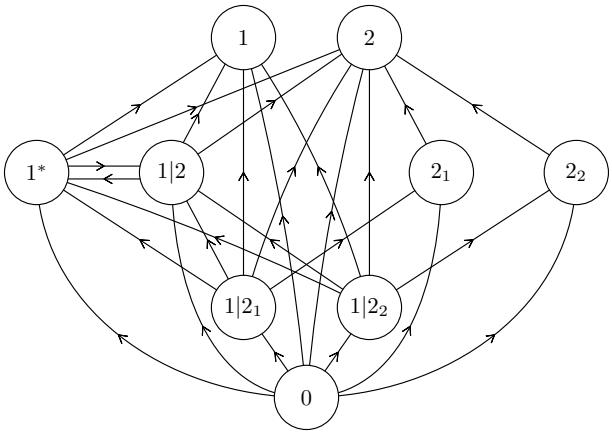

Fig. 4. All possible labels and all possible transitions between them.

Set $L_K(v) := 1$ *and* $L_K(w) := 2_2$.

Reason: Because v has degree 1 in G, v must get role 1. This means w must get role 2, and we must ensure that at least one other neighbor of w gets role 2. Hence, the given label assignments maintain Invariant 1.

Case 2. $|K| \geq 3$.

Set $L_K(u) := 1|2$ *for every vertex* $u \in K$.

Reason: If in Phase 2 all vertices in $K \cap K'$ receive role 2, then we assign role 1 to v and role 2 to all other vertices of $K \setminus K'$. In the other case, when there exists a vertex $x \in K \cap K'$ that receives role 1, we assign role 2 to v and every other vertex in $K \setminus K'$. Hence, Invariant 1 is maintained.

Phase 1b. Deal with non-leaf bags that are not the root bag

Suppose $K \neq K_r$ is not a leaf bag of T. Recall that we process K only after each of its child bags has been processed. Hence, $L_{K_c}(v) \neq 0$ for every vertex $v \in K$ that is given to K from a child bag K_c. Such a vertex v may belong to different child bags, and consequently, it may have received different labels. We show how to combine these multiple labels into a single label $L_K(v)$ in K such that Invariant 1 is maintained. The algorithm distinguish between three cases.

Case 1. K contains a vertex v that has label 1 in a child bag of K.

Our algorithm distinguishes between the following cases.

Case 1.1. Vertex v has received label $2, 2_1$ or 2_2 in another child bag of K.

Output `NO`.

Reason: Invariant 1 forces v to have two different roles. This is not allowed.

Case 1.2. There is a vertex $w \in K \setminus \{v\}$ with label 1 in a child bag of K.

Output `NO`.

Reason: Invariant 1 forces two adjacent vertices, namely v and w, both to have role 1. This is not allowed.

Case 1.3. There is a set $J \subseteq K$ of vertices that received label 1^* in a child bag of K to which v does not belong.

Output `NO`.

Reason: Invariant 1 forces two adjacent vertices, namely v and a vertex from J, both to have role 1. This is not allowed.

Case 1.4. Cases $1.1 - 1.3$ do not occur and $|K| = 2$.
Let w be the other vertex of K. Note that either v or w stays behind in K due to Observation 1.

Case 1.4.1. $L_{K_c}(w) \in \{1^*, 1|2, 1|2_1, 2_1\}$ in some child bag K_c of K.

Set $L_K(v) := 1$ *and* $L_K(w) := 2$.

Reason: Invariant 1 forces v to get role 1, and consequently, w to get role 2. If $L_{K_c}(w) = 1^*$ in some child bag K_c, then by label definition w has a neighbor with label 2_1 and this neighbor will get role 2 in Phase 2. If $L_{K_c}(w) \in \{1|2, 1|2_1, 2_1\}$, then we also apply the label definitions.

Case 1.4.2. $L_{K_c}(w) \in \{1|2_2, 2_2\}$ for every child bag K_c of K that contains w, and $K \backslash K' = \{v\}$.

Set $L_K(v) := 1$ *and* $L_K(w) := 2_2$.

Reason: Invariant 1 forces v to have role 1, and consequently, w must get role 2 and still requires a neighbor with role 2. Note that w may belong to no child bag of K.

Case 1.4.3. $L_{K_c}(w) \in \{1|2_2, 2_2\}$ for every child bag K_c of K that contains w, and $K \backslash K' = \{w\}$.

Output `NO`.

Reason: Invariant 1 forces v to have role 1, and consequently w to have role 2, and then w gets no required neighbor with role 2. Note that w may belong to no child bag of K.

Case 1.5. Cases $1.1 - 1.3$ do not occur and $|K| \geq 3$.

Set $L_K(v) := 1$ *and* $L_K(w) := 2$ *for every* $w \in K \backslash \{v\}$.

Reason: Vertex v must get role 1 by Invariant 1, and in this way each vertex in $K \setminus \{v\}$ will have a neighbor with role 1 (namely v) and a neighbor with role 2.

We conclude that Invariant 1 is maintained in every subcase described above.

Case 2. K contains no vertex that received label 1 in a child bag, but K does contain a vertex that received label 1^* in a child bag.

For some $p \geq 1$, let $K_1, \ldots, K_p$ be the child bags of K that contain vertices with label 1^*. For $i = 1, \ldots, p$, let V_i^* be the set of vertices in K_i that have label 1^* in K_i, while not having label 2, 2_1 or 2_2 in any other child bag of K. So, exactly one vertex in each K_i must get role 1 and this vertex must be chosen from V_i^*. Because such a vertex will be in K and two vertices with role 1 cannot be adjacent, this vertex must be the same vertex for every V_i^*. Hence, it must be taken from the set $V^* = \bigcap_{i=1}^{p} V_i^*$.

The algorithm distinguishes between the following cases.

Case 2.1. $|V^*| = 0$.

Output `NO`.

Reason: See the above argumentation.

Case 2.2. $|V^*| = 1$.

Let $V^* = \{v\}$.

Set $L_K(v) := 1$ *and* $L_K(w) := 2$ *for all* $w \in K \setminus \{v\}$.

Reason: Why the algorithm sets $L_K(v) := 1$ is explained above. The algorithm sets $L_K(u) := 2$ for every $w \in K \setminus \{v\}$ for the following three reasons. Firstly, none of the vertices in $K \setminus \{v\}$ received label 1 in any of the child bags of K, since we assumed that Case 1 does not occur. Secondly, the constraint imposed by the labels 1^* in each K_i will be satisfied by v. Thirdly, every vertex in $K \setminus \{v\}$ has a neighbor that will get role 1 in Phase 2, namely v, and a neighbor that will get role 2. The latter is true because $|K| \geq 3$, which can be seen as follows. Recall that vertices only have label 1^* if they are given to a parent bag in groups of size at least 2. This means K has size at least two. However, if $|K| = 2$ then K is properly contained in one of its child bags, contradicting Observation 1. Hence $|K| \geq 3$ holds indeed.

Case 2.3. $|V^*| \geq 2$ and $V^* \subseteq (K \cap K')$.

Set $L_K(v) := 1^*$ *for all* $v \in V^*$ *and* $L_K(w) := 2$ *for all* $w \in K \setminus V^*$.

Reason: The same arguments as in Case 2.2 apply. The only difference is that there are at least two vertices in V^*. Because these vertices are all given to K', the algorithm later decides which one of them will get role 1.

Case 2.4. $|V^*| \geq 2$ and $V^* \not\subseteq (K \cap K')$.

Set $L_K(v) := 1|2$ *for all* $v \in V^*$ *and* $L_K(w) := 2$ *for all* $w \in K \setminus V^*$.

Reason: The algorithm sets $L_K(v) := 1|2$ for all $v \in V^*$ for the following reason. If a vertex $v \in V^* \cap K'$ receives role 1 in Phase 2, then all neighbors of v will receive role 2. If all vertices in $V^* \cap K'$ receive role 2 (or if $V^* \cap K' = \emptyset$), then the algorithm gives the required role 1 to one of the vertices in $V^* \setminus K'$. The label of all other vertices in K is set to 2 because of the same three reasons as in Case 2.2 and 2.3.

We conclude that Invariant 1 is maintained in every subcase described above.

Case 3. K contains no vertex that received label 1 or 1^* in any of its child bags.

We first update the labels of each vertex $v \in K$ that is given to K from the child bags of K. If v is in only one child bag K_c of K, then set $L_K(v) := L_{K_c}(v)$. Otherwise, if v has labels in two or more different child bags of K, the algorithm acts as follows. It first combines two labels into a new label as prescribed by Table 2, then combines this new label with the next label (if it exists), and continues until a single label remains.

	2	2_1	2_2	$1\|2$	$1\|2_1$	$1\|2_2$
2	2	2	2	2	2	2
2_1	2	2_1	2	2	2_1	2
2_2	2	2	2_2	2	2	2_2
$1\|2$	2	2	2	$1\|2$	$1\|2$	$1\|2$
$1\|2_1$	2	2_1	2	$1\|2$	$1\|2_1$	$1\|2$
$1\|2_2$	2	2	2_2	$1\|2$	$1\|2$	$1\|2_2$

Table 2. Combining two labels from different child bags.

We explain Table 2 by discussing the following two cases. Suppose $v \in K$ has label 2_1 in child bag K_c and label 2_2 in child bag K_d. Label 2_1 means that v is ensured to have a neighbor that will receive role 2 in Phase 2. Label 2_2 means that v is ensured to have a neighbor that will receive role 1 in Phase 2. Hence, the algorithms sets $L_K(v) := 2$. Suppose $v \in K$ has label 2_1 in K_c and label $1|2_2$ in K_d. Then v cannot get role 1 in Phase 2. In that case the algorithm sets $L_K(v) := 2$. Arguments like the above follow directly from the label definitions and can be used for all other combinations. This way Invariant 1 is maintained.

After the algorithm has updated the label of each vertex that was given to K from a child bag, the following holds for each $v \in K$. If v was given to K from a child bag then $L_K(v) \neq 0$; otherwise $L_K(v) = 0$. We write

$$L_K := \{L_K(v) \mid v \in K\},$$

and $L_{K\setminus K'} = \{L_K(v) \mid v \in K\setminus K'\}$ and $L_{K\cap K'} = \{L_K(v) \mid v \in K \cap K'\}$, where we recall that K' is the parent bag of K.

The algorithm distinguishes between the following cases.

Case 3.1. $\{2_2\} \subseteq L_K \subseteq \{0, 1|2, 1|2_1, 1|2_2, 2, 2_1, 2_2\}$.

Case 3.1.1. $|K| \geq 3$ or $|L_K \cap \{2, 2_1, 2_2\}| \geq 2$.

Change every K-label 2_2 into 2 and go to Case 3.2.

Reason: If $|K| \geq 3$, then K will contain at least two vertices with role 2. Hence, any vertex with label 2_2 in K will get its required neighbor with role 2. The same is true if $|K| = 2$ and both vertices of K have a K-label in $\{2, 2_1, 2_2\}$.

Case 3.1.2. $|K| = 2$ and $|L_K \cap \{2, 2_1, 2_2\}| = 1$.

Let $K = \{v, w\}$. Because $2_2 \in L_K$ we may assume that $L_K(v) = 2_2$ and $L_K(w) \notin \{2, 2_1, 2_2\}$, thus $L_K(w) \in \{0, 1|2, 1|2_1, 1|2_2\}$. Note that either v or w stays behind in K by Observation 1.

Case 3.1.2.1. $L_K(w) \in \{0, 1|2_1\}$ and $K \backslash K' = \{v\}$.

Set $L_K(w) := 2_1$.

Reason: Vertex v stays behind and needs a neighbor with role 2. This neighbor can only be w.

Case 3.1.2.2. $L_K(w) \in \{0, 1|2_1\}$, and $K \backslash K' = \{w\}$.

Set $L_K(w) := 1$.

Reason: First suppose $L_K(w) = 0$. Then w is not in a child bag of K. Because w stays behind, this means that w has degree one. Hence w must receive role 1. Now suppose $L_K(w) = 1|2_1$. Because w stays behind, and v will receive role 2, we find that w will not get a neighbor with role 1, which it would need if it gets role 2. Hence w must get role 1.

Case 3.1.2.3. $L_K(w) \in \{1|2, 1|2_2\}$.

Set $L_K(w) := 2$.

Reason: First suppose $K \backslash K' = \{v\}$. In this case w can function as the neighbor with role 2 that v needs. Because w then has a neighbor, namely v, that will receive role 2, we can set $L_K(w) := 2$, even in the case that w had label $1|2_2$ in K. Now suppose $K \backslash K' = \{w\}$. The algorithm lets w be the required role 2 neighbor of v. If $L_K(w) = 1|2_2$, then the algorithm sets $L_K(w) := 2$ instead of $L_K(w) := 2_2$, because v will be a role 2 neighbor of w.

We conclude that Invariant 1 is maintained in every subcase described above.

Case 3.2. $L_K \cap \{2, 2_1\} \neq \emptyset$ and $L_K \subseteq \{0, 1|2, 1|2_1, 1|2_2, 2, 2_1\}$.

The algorithm distinguishes between the following cases.

Case 3.2.1. $L_{K \backslash K'} \cap \{0, 1|2, 1|2_1, 1|2_2\} \neq \emptyset$.

Change every K-label in $\{0, 1|2_1, 1|2_2\}$ into $1|2$, and every K-label 2_1 into 2.

Reason: Let $v \in K \backslash K'$ have $L_K(v) \in \{0, 1|2, 1|2_1, 1|2_2\}$. If none of the vertices in $K \cap K'$ receives role 1 in Phase 2, then Phase 2 assigns role 1 to v; otherwise v gets role 2. The latter is fine, because K contains a vertex with K-label 2 or 2_1 that will receive role 2.

Case 3.2.2. $\{2_1\} \subseteq L_{K \backslash K'} \subseteq \{2_1, 2\}$ and $|L_{K \cap K'} \cap \{0, 1|2, 1|2_1, 1|2_2\}| = 0$.

Output `NO`.

Reason: There is a vertex in $K \backslash K'$ with K-label 2_1, and this vertex will not get its required neighbor with role 1.

Case 3.2.3. $\{2_1\} \subseteq L_{K\setminus K'} \subseteq \{2_1, 2\}$ and $|L_{K\cap K'} \cap \{0, 1|2, 1|2_1, 1|2_2\}| = 1$

Let $v \in L_{K\cap K'}$ be the (unique) vertex in $K \cap K'$ that has $L_K(v) \in \{0, 1|2, 1|2_1\}$.

Set $L_K(v) := 1$ and change the K-label of every vertex in $K\setminus\{v\}$ into 2.

Reason: First suppose $|K| = 2$, say $K = \{v, w\}$. Observe that $w \in K\setminus K'$ due to Observation 1. Because $2_1 \in L_K$, we then find that $L_K(w) = 2_1$. The algorithm changes this label into 2, because v is a neighbor of w that will get role 1 after updating its K-label. Now suppose $|K| \geq 3$. Because v will get role 1, every vertex in $K\setminus\{v\}$ will get role 2. Because $|K| \geq 3$, every vertex in $K\setminus\{v\}$ is guaranteed to have a neighbor with role 2. Hence, the K-label of such a vertex is updated into 2.

Case 3.2.4. $\{2_1\} \subseteq L_{K\setminus K'} \subseteq \{2_1, 2\}$ and $|L_{K\cap K'} \cap \{0, 1|2, 1|2_1, 1|2_2\}| \geq 2$.

Let J consist of all vertices in $K \cap K'$ with K-label in $\{0, 1|2, 1|2_1, 1|2_2\}$, so $|J| \geq 2$.

Change the K-label of every vertex in J into 1^*, and the K-label of every vertex in $K\setminus J$ into 2.

Reason: We use the same arguments as in Case 3.2.2.2 after observing that $|K| \geq 3$ holds.

Case 3.2.5. $L_{K\setminus K'} = \{2\}$.

Change every K-label 0 into $1|2_1$, and every K-label $1|2_2$ into $1|2$.

Reason: Because all vertices in $K\setminus K'$ have K-label 2, they do not need a vertex with role 1 in $K \cap K'$, and every vertex in $K \cap K'$ is guaranteed to have a neighbor with role 2. Therefore, the algorithm does as above.

We conclude that Invariant 1 is maintained in each subcase of Case 3.2.

Case 3.3. $L_K \subseteq \{0, 1|2, 1|2_1, 1|2_2\}$.

Case 3.3.1. $|K| \geq 3$, or $|K| = 2$ with $L_{K\setminus K'} = \{1|2\}$, or $|K| = 2$ with $L_{K\setminus K'} = \{1|2_1\}$ and $L_{K\cap K'} \in \{1|2, 1|2_1\}$.

Change every K-label into $1|2$.

Reason: First suppose $|K| \geq 3$. Then each vertex in K will get a neighbor with role 2. If no vertex of $K \cap K'$ gets role 1, the algorithm gives role 1 to a vertex in $K\setminus K'$. Otherwise every vertex in $K\setminus K'$ gets role 2 and will then have a neighbor with role 1. Hence, the algorithm correctly updates each K-label into $1|2$.

Now suppose $|K| = 2$ with $L_{K\setminus K'} = \{1|2\}$. Let $K = \{v, w\}$ with $v \in K \cap K'$, and $w \in K \cap K'$. Thus, $L_K(v) = 1|2$. If w gets role 1 in Phase 2, then v will get role 2. If w gets role 2, then it already has a neighbor in K' with some role, as $K' \setminus K \neq \emptyset$ due to Observation 1. If w still needs a neighbor of a specific role, then the algorithm sets v to that role; otherwise v is assigned an arbitrary role. Hence, the algorithm correctly sets $L(w) := 1|2$.

Finally suppose $|K| = 2$ with $L_{K\setminus K'} = \{1|2_1\}$ and $L_{K\cap K'} \in \{1|2, 1|2_1\}$. Let $K = \{v, w\}$ with $v \in K\setminus K'$ and $w \in K \cap K'$. Then $L_K(v) = 1|2_1$ and

$L_K(w) \in \{1|2, 1|2_1\}$. If w gets role 2 then v gets role 1 as otherwise, when v gets role 2, v would not have a required neighbor with role 1. Note that w already is guaranteed to have a neighbor with role 2. If w gets role 1 then v gets role 2. Then w will be the required role 1 neighbor of v.

Case 3.3.2 $|K| = 2$ with $L_{K\setminus K'} = \{1|2_1\}$ and $L_{K\cap K'} \in \{0, 1|2_2\}$.

Let $K = \{v, w\}$. Assume $v \in K\setminus K'$, thus $L_K(v) = 1|2_1$. By Observation 1 we find that $w \in K \cap K'$, thus $L_K(w) \in \{0, 1|2_2\}$.

Set $L_K(v) := 1|2$ *and* $L_K(w) := 1|2_2$.

Reason: If w gets role 2 then v gets role 1 as otherwise, when v gets role 2, v would not have a required neighbor with role 1. Then w is still required to get a neighbor with role 2. If w gets role 1 then v gets role 2. Then w will be the required role 1 neighbor of v.

Case 3.3.3. $|K| = 2$ with $L_{K\setminus K'} = \{1|2_2\}$.

Let $K = \{v, w\}$. Assume $v \in K\setminus K'$, thus $L_K(v) = 1|2_2$. By Observation 1 we find that $w \in K \cap K'$. Note that $L_K(w) \in \{0, 1|2, 1|2_1, 1|2_2\}$.

Set $L_K(v) := 1|2$ *and* $L_K(w) := 2$.

Reason: Vertex w cannot get role 1, because then v would get role 2 and miss its required neighbor with role 2. Since K' is maximal, there exists a vertex $w' \in K'\setminus K$. If w' gets role 1, the algorithm assigns role 2 to role v. If w' gets role 2, the algorithm assigns role 1 to v. In this way w will have neighbors of both roles.

Case 3.3.4. $|K| = 2$ with $L_{K\setminus K'} = \{0\}$.

Let $K = \{v, w\}$. Assume $v \in K\setminus K'$, thus $L_K(v) = 0$. This means that v is vertex of degree 1 in G. By Observation 1 we find that $w \in K \cap K'$. Note that $L_K(w) \neq 0$, because then K would be a leaf bag. Hence $L_K(w) \in \{1|2, 1|2_1, 1|2_2\}$.

Case 3.3.4.1. $L_K(w) \in \{1|2, 1|2_1\}$.

Set $L_K(v) := 1$ *and* $L_K(w) := 2$.

Reason: Because v has degree 1 in G, v must get role 1. This means that w must get role 2 and that w has a neighbor with role 1, namely v.

Case 3.3.4.2. $L_K(w) = 1|2_2$.

Set $L_K(v) := 1$ *and* $L_K(w) := 2_2$.

Reason: Because v has degree 1 in G, v must get role 1. This means that w must get role 2 but still needs a neighbor with role 2.

We conclude that Invariant 1 is maintained in each subcase of Case 3.3.

Phase 1c. Deal with the root bag

The root bag K_r is the last bag of T to be processed in Phase 1. Because the root bag is the only bag of T that does not have a parent bag, the case analysis for K_r slightly differs from the case analysis for other bags of T, as we explain below. After processing K_r, the algorithm enters Phase 2 unless it has output `NO`.

Case 1. K_r contains a vertex v that has label 1 in a child bag of K_r.

The algorithm acts as in Case 1 of Phase 1b except when it is in Case 1.4.2, where it does as follows instead.

Output `NO`.

Reason: Vertex w is not able to get a required role 2 neighbor.

Case 2. K_r contains no vertex that received label 1 in a child bag, but K_r does contain a vertex that received label 1^* in a child bag.

The algorithm acts as in Case 2 of Phase 1b except when it is in Case 2.3 or 2.4, where it does as follows instead.

Set $L_{K_r}(v) := 1$ *for some* $v \in V^*$ *and* $L_{K_r}(w) := 2$ *for all* $w \in K_r \backslash \{v\}$.

Reason: K_r does not have a parent bag. Hence, the algorithm must assign one of the vertices role 1.

Case 3. K_r contains no vertex that received label 1 or 1^* in one of its child bags.

The algorithm first updates the K_r-label of every vertex in K_r as in Case 3 of Phase 1b and then distinguishes between the following cases.

Case 3.1. $L_{K_r} \cap \{1|2, 1|2_1, 1|2_2\} \neq \emptyset$.

Because $|K_r| \geq 2$, there exist two different vertices v, w in K_r. Assume $L_{K_r}(v) \in \{1|2, 1|2_1, 1|2_2\}$. Note that $L_{K_r}(w) \in \{0, 1|2, 1|2_1, 1|2_2, 2, 2_1, 2_2\}$.

Case 3.1.1. $|K_r| \geq 3$, or $|K_r| = 2$ with $L_{K_r}(w) \in \{1|2, 1|2_1, 2, 2_1\}$.

Set $L_{K_r}(v) := 1$ *and* $L_{K_r}(u) := 2$ *for all* $u \in K_r \backslash \{v\}$.

Reason: First suppose $|K_r| \geq 3$. This means that $|K_r \backslash \{v\}| \geq 2$. Hence, every vertex in $K_r \backslash \{v\}$ has a neighbor with role 2, while v is the required role 1 neighbor. Now suppose $|K_r| = 2$ with $L_{K_r}(w) \in \{1|2, 1|2_1, 2, 2_1\}$. Then w needs no neighbor of role 2 anymore, and v will be its neighbor of role 1.

Case 3.1.2. $|K_r| = 2$ with $L_{K_r}(v) \in \{1|2, 1|2_1\}$ and $L_{K_r}(w) \in \{0, 1|2_2\}$.

Set $L_{K_r}(v) := 2$ *and* $L_{K_r}(w) := 1$.

Reason: In this way, both v and w have the required roles in their neighborhoods.

Case 3.1.3. $|K_r| = 2$ either with $L_{K_r}(v) = 1|2_2$ and $L_{K_r}(w) = 0$, or with $L_{K_r}(v) = 1|2_1$ and $L_{K_r}(w) = 2_2$.

Output `NO`.

Reason: In the first case, w is in no child bag of K_r. If w gets role 2, then w either has no neighbor with role 1 or no neighbor with role 2. If w gets role 1, then v gets role 2. However, then v has no neighbor with role 2. Hence, the algorithm correctly outputs `NO`. In the second case w will get role 2 and has no neighbor with role 2, unless v gets role 2. However, in that case, v has no neighbor with role 1. Hence, the algorithm correctly outputs `NO`.

Case 3.1.4. $|K_r| = 2$ either with $L_{K_r}(v) = 1|2_2$ and $L_{K_r}(w) \in \{1|2_2, 2_2\}$, or with $L_{K_r}(v) = 1|2$ and $L_{K_r}(w) = 2_2$.

Set $L_{K_r}(v) := 2$ *and* $L_{K_r}(w) := 2$.

Reason: In this way, vertex v and w each get role 2, and both have a neighbor with role 1 from a descendant of K_r, and a neighbor with role 2, namely each other.

Case 3.2. $L_{K_r} \subseteq \{0, 2, 2_1, 2_2\}$.

Case 3.2.1. $\{2_1\} \subseteq L_{K_r} \subseteq \{2, 2_1, 2_2\}$, or $|K_r| = 2$ with $\{0\} \subseteq L_{K_r} \subseteq \{0, 2_2\}$.

Output `NO`.

Reason: Suppose $\{2_1\} \subseteq L_{K_r} \subseteq \{2, 2_1, 2_2\}$. Then the vertex that has K_r-label 2_1 has no neighbor of role 1.

Suppose $|K_r| = 2$ with $\{0\} \subseteq L_{K_r} \subseteq \{0, 2_2\}$. If $L_{K_r} = \{0\}$, then G is a graph on two vertices. Consequently, G has no R_5-role assignment. Suppose $L_{K_r} = \{0, 2_2\}$. Let $K_r = \{v, w\}$ with $L_{K_r}(v) = 0$ and $L_{K_r}(w) = 2_2$. If v gets role 2, then v has no neighbor with role 1. Hence v must get role 1. In that case, however, w has no neighbor with role 2. Thus, the algorithm correctly outputs `NO`.

Case 3.2.2. $L_{K_r} \subseteq \{2, 2_2\}$.

Change the K_r*-label of every vertex in* K_r *into 2.*

Reason: In this way all vertices in K_r will get role 2, while having both a neighbor with role 1 and a neighbor with role 2.

Case 3.2.3. $|K_r| \geq 3$ with $\{0\} \subseteq L_{K_r}$.

Let $v \in K_r$ have $L_{K_r}(v) = 0$.

Set $L_{K_r}(v) := 1$ *and* $L_{K_r}(u) := 2$ *for all* $u \in K_r \backslash \{v\}$.

Reason: Because $|K_r| \geq 3$, we find that at least two vertices in K_r get role 2. Hence, all vertices in K_r get the required roles in their neighborhood.

Case 3.2.4. $|K_r| = 2$ with $\{0\} \subseteq L_{K_r} \not\subseteq \{0, 2_2\}$.

Let $K_r = \{v, w\}$, and let v be a vertex with K_r-label 0. Then $L_{K_r}(w) \in \{2, 2_1\}$, since otherwise we would have $L_{K_r} \subseteq \{0, 2_2\}$.

Set $L_{K_r}(v) := 1$ *and* $L_{K_r}(w) := 2$.

Reason: In this way both v and w get the required roles in their neighborhood.

2.4 Phase 2 in detail

Since the algorithm entered Phase 2, an R_5-role assignment of G exists. Note that at the end of Phase 1 every vertex in K_r either has K_r-label 1 or K_r-label 2. The algorithm will assign role 1 to the vertices in K_r with K_r-label 1 and role 2 to the vertices in K_r with K_r-label 2. It will then construct an R_5-role assignment of G by propagating this partial solution on $G[K_r]$ in a "top-down" matter, processing a bag K only after its parent bag has been processed. Note that in this way a vertex has its most updated label when the algorithm assigns it a role.

Let K' be a child bag of K_r. Any vertex $u \in K'$ that has been assigned a role already must be a vertex of K_r. Because the partial solution on $G[K_r]$ has been chosen such that Invariant 1 is maintained, the role of such a vertex u satisfies the constraints given by $L_{K'}(u)$. Hence u keeps its role.

The algorithm considers all vertices of K' without a role one by one. Let v be such a vertex. If $L_{K'}(v) = 1$ then v gets role 1. If $L_{K'}(v) \in \{2, 2_1, 2_2\}$ then v gets role 2. If $L_{K'}(v) = 1^*$, then v belongs to a set J of vertices that each have K'-label 1^*. We assign role 1 to v unless another vertex from J already got role 1. Otherwise, $L_{K'}(v) \in \{1|2, 1|2_1, 1|2_2\}$ must hold. In that case we follow exactly the same analysis as in the description of Phase 1 by checking if a neighbor w of v in K_r still needs a required neighbor of certain role. If so, we let v be this neighbor of w. In this way the roles of the vertices of K' that are also in child bags of K' satisfy the constraints given by their labels in these child bags. Invariant 1 then again ensures that we can extend our partial solution to every child bag K of K', and we proceed accordingly until all vertices have obtained a role. In this way we obtain an R_5-role assignment of G.

2.5 Running time analysis

Theorem 5. *The R_5-Role Assignment problem can be solved in linear time for the class of chordal graphs.*

Proof. Let $G = (V, E)$ be a connected chordal graph. The algorithm first computes a clique tree $T = (\mathcal{K}, \mathcal{E})$ of G, which can be done in $\mathcal{O}(|V| + |E|)$ time by Theorem 4. The algorithm then acts in the way described in Sections 2.2, 2.3, and 2.4. We already proved correctness of the algorithm in those sections.

In Phase 1, the algorithm computes $|K|$ labels per bag K. Then, by Theorem 3, there are $\mathcal{O}(|V| + |E|)$ labels to compute in total. We first determine, for each vertex $v \in K$, the time required to compute the label $L_K(v)$, given the labels of v in the child bags of K. Next we determine the time required for the remaining part of Phase 1.

The time required to construct a label $L_K(v)$ from a combination of labels from child bags of K to which v belongs is proportional to the number of such child bags. For the entire clique tree, this combining of labels then costs $\mathcal{O}(\sum_{K \in \mathcal{K}} |K|) = \mathcal{O}(|V| + |E|)$ time by Theorem 3.

The time required to update the labels in a bag K as prescribed in Section 2.2 is $\mathcal{O}(|K|)$ by first scanning K to decide what subcase applies and then updating

the labels for K. Again by Theorem 3, this requires $\mathcal{O}(|V| + |E|)$ for the entire clique tree. We conclude that Phase 1 runs in $\mathcal{O}(|V| + |E|)$ time.

Due to our greedy approach in Phase 2, this phase can also be executed in $\mathcal{O}(|V| + |E|)$ time. We conclude that the overall running time of our algorithm is $\mathcal{O}(|V| + |E|)$. This completes the proof of Theorem 5. □

2.6 A remark regarding R_6-role assignments

In our linear time algorithm that solves the 2-Role Assignment problem on chordal graphs we do not have to check if the input graph has an R_6-role assignment (cf. Theorem 2). This is rather "fortunate" as the R_6-Role Assignment problem turns out to be NP-complete even when restricted to split graphs, a subclass of chordal graphs. For showing this we need some extra terminology.

A *split graph* is a graph $G = (V, E)$ whose vertex set V can be partitioned into two disjoint sets I and C, such that I is an independent set in G and C is a clique in G. A *hypergraph* H is a pair $(Q, \mathcal{S})$ consisting of a set $Q = \{q_1, \ldots, q_m\}$, called the *vertices* of H, and a set $\mathcal{S} = \{S_1, \ldots, S_n\}$ of nonempty subsets of Q, called the *hyperedges* of H. With a hypergraph $H = (Q, \mathcal{S})$ we associate its *incidence graph* I, which is a bipartite graph with partition classes Q and $\mathcal{S}$, where for any $q \in Q, S \in \mathcal{S}$ we have $qS \in E_I$ if and only if $q \in S$. A 2-*coloring* of a hypergraph $H = (Q, \mathcal{S})$ is a partition (Q_1, Q_2) of Q such that $Q_1 \cap S_j \neq \emptyset$ and $Q_2 \cap S_j \neq \emptyset$ for $1 \leq j \leq n$. A hypergraph H is called *nontrivial* if Q contains at least three vertices and Q is a member of $\mathcal{S}$. The Hypergraph 2-Colorability problem asks whether a given hypergraph has a 2-coloring. This problem, also known as Set Splitting, is NP-complete (cf. [16]). Obviously, it remains NP-complete when restricted to nontrivial hypergraphs.

Proposition 1. *The R_6-Role Assignment problem is NP-complete for the class of split graphs.*

Proof. Let $(Q, \mathcal{S})$ be a nontrivial hypergraph. In its incidence graph I we add an edge between every pair of vertices in Q. This results in a split graph G. We claim that $(Q, \mathcal{S})$ has a 2-coloring if and only if G has an R_6-role assignment.

Suppose $(Q, \mathcal{S})$ has a 2-coloring (Q_1, Q_2). Since $|Q| \geq 3$, we may without loss of generality assume that $|Q_2| \geq 2$. We give each $q \in Q_1$ role 1 and each $q \in Q_2$ role 2. We assign role 1 to each $S \in \mathcal{S}$. Because (Q_1, Q_2) is a 2-coloring, each vertex in $\mathcal{S}$ has a neighbor with role 1 and a neighbor with role 2 in G. Because Q is a clique in G and $|Q_2| \geq 2$, each vertex in Q_2 has a neighbor with role 1 and a neighbor with role 2. For the same reason, each vertex in Q_1 has a neighbor with role 2. Since $(Q, \mathcal{S})$ is nontrivial, $Q \in \mathcal{S}$. This guarantees that also in case $|Q_1| = 1$, each vertex in Q_1 has a neighbor with role 1. We conclude that G has an R_6-role assignment.

Suppose G has an R_6-role assignment. Then every vertex in $\mathcal{S}$ has a neighbor with role 1 and a neighbor with role 2. By construction, these neighbors are in Q. This immediately gives a 2-coloring of $(Q, \mathcal{S})$. □

3 Complexity of k-Role Assignment for $k \geq 3$

It is known that the k-Role Assignment problem is NP-complete for any fixed $k \geq 2$ [13]. We proved in Section 2 that 2-Role Assignment can be solved in linear time when the input graph is chordal. In this section, we show that the k-Role Assignment problem for chordal graphs is NP-complete for every fixed $k \geq 3$. We use a reduction from the Hypergraph 2-Colorability problem. Our NP-completeness proof is more involved than the one for the general case in [13], as the graph constructed there (also from an instance of Hypergraph 2-Colorability) is not chordal.

Theorem 6. *For $k \geq 3$, the k-Role Assignment problem is NP-complete for the class of chordal graphs.*

Proof. Let $k \geq 3$. We use a reduction from Hypergraph 2-Colorability. Let $(Q, \mathcal{S})$ be a nontrivial hypergraph with incidence graph I.

We modify I as follows. Firstly, we add an edge between any two vertices in Q, so Q becomes a clique. Secondly, for each $S \in \mathcal{S}$ we take a path $P^S = p_1^S \cdots p_{k-2}^S$ and connect it to S by the edge $p_{k-2}^S S$, so these new paths P^S are pendant paths in the resulting graph. Thirdly, we add a copy H^q of a new graph H for each $q \in Q$. Before we explain how to do this, we first define H. Start with a path $u_1 u_2 \cdots u_{2k-4}$. Then take a complete graph on four vertices a, b, c, d, and a complete graph on four vertices w, x, y, z. Add the edges $cu_1, du_1, u_{2k-4}w, u_{2k-4}x$. We then take three paths $S = s_1 \cdots s_{k-2}$, $T = t_1 \cdots t_{k-2}$ and $T' = t'_1 \cdots t'_{k-2}$, and we add the edges $s_{k-2}w, ct_{k-2}, dt'_{k-2}$. This finishes the construction of H. We connect a copy H^q to q via the edge qu_1^q, where u_1^q is the copy of the vertex u_1. We call the resulting graph G; notice that this is a connected chordal graph. See Figure 5 for an example.

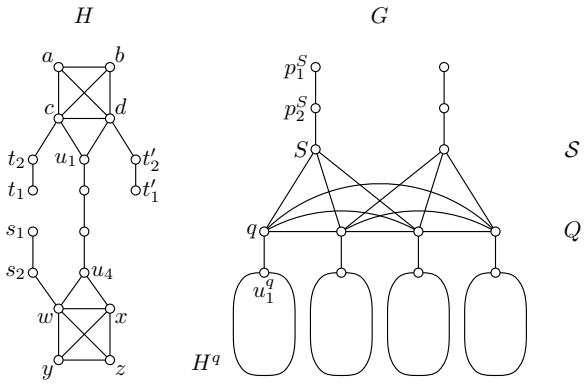

Fig. 5. The graph H (left side) and the graph G (right side) when $k = 4$.

We first show that if G has a k-role assignment $r : V_G \to \{1, \ldots, k\}$, then r is an R^*-role assignment, where R^* denotes the k-vertex path on vertices $1, \ldots, k$

such that there is an edge between vertex i and $i+1$ for $i = 1, \ldots, k-1$, and a loop in vertex $k-1$ and in vertex k. To see this, consider a copy H^q of H in G; we show that we can assign roles to the vertices of H^q in only one way. For convenience, we denote the vertices of H^q without the superscript q.

Let A be an induced path in G on at most k vertices, starting in a vertex of degree 1. We claim that the k-role assignment r must assign exactly $|V_A|$ different roles to the vertices of A, i.e., we have $|r(V_A)| = |V_A|$. This can be seen as follows. Suppose $|r(V_A)| = \alpha < |V_A|$. We may without loss of generality assume that, starting from the vertex of degree 1, r assigns roles $1, \ldots, \alpha$ to the first α vertices of A and role $\alpha - 1$ to the next vertex of A. However, then all neighbors of every role in R are fixed. Then, because G is connected, none of the vertices of G gets assigned role $k \geq |V_A| > \alpha$ by r. This means that r is not a k-role assignment of G, which is a contradiction.

From the above, we find that we may write $r(t_i) = i$ for $i = 1, \ldots, k-2$ and $r(c) = k-1$. This implies that a vertex with role 1 only has vertices with role 2 in its neighborhood and a vertex with role i for $2 \leq i \leq k-2$ only has vertices with role $i-1$ and role $i+1$ as neighbors. Then a vertex with role k can only be adjacent to vertices with role $k-1$ or role k. Hence c must have a neighbor with role k.

Suppose $r(d) = k$. Then $r(t'_{k-2}) \in \{k-1, k\}$ and this eventually leads to $r(t'_1) \geq 2$ without a neighbor of role $r(t'_1) - 1$ for t'_1. This is not possible. Hence $r(d) \neq k$. This means that $k \in r(\{a, b, u_1\})$. Since a, b, u_1 are neighbors of d as well and a vertex with role k can only have neighbors with role $k-1$ and k, we then find that d has role $k-1$.

The above implies that a and b have their role in $\{k-2, k-1, k\}$. Suppose $k = 3$. If $r(a) = 1$, then $r(b) = 2$ implying that r is a 2-role assignment (as $r(c) = r(d) = 2$ and then $r(N_G(b)) = \{1, 2\}$ implying that r cannot use role 3 because G is connected). Suppose $r(a) = 2$. Then a needs a neighbor with role 1. Hence $r(b) = 1$, but then r is a 2-role assignment. Suppose $r(a) = 3$. Then $r(b) \neq 2$, as otherwise b needs a neighbor with role 1. Hence $r(b) = 3$. This means that r is an R^*-role assignment. Suppose $k \geq 4$. If $r(a) = k-2$, then a needs a neighbor with role $k-3$. So, $r(b) = k-3$. However, this is not possible since vertex b with role $k-3$ is adjacent to vertex c with role $k-1$. If $r(a) = k-1$, then $r(b) = k-2$. This is not possible either. Hence $r(a) = k$ and for the same reasons $r(b) = k$. Then r is an R^*-role assignment.

We claim that $(Q, \mathcal{S})$ has a 2-coloring if and only if G has a k-role assignment.

Suppose $(Q, \mathcal{S})$ has a 2-coloring (Q_1, Q_2). We show that G has an R^*-role assignment, which is a k-role assignment. We assign role i to each p_i^S for $i = 1, \ldots, k-2$ and role $k-1$ to each $S \in \mathcal{S}$. As $(Q, \mathcal{S})$ is nontrivial, either Q_1 or Q_2, say Q_2, has size at least two. Then we assign role $k-1$ to each $q \in Q_1$ and role $k-2$ to neighbor u_1^q. We assign role k to each $q \in Q_2$ and $k-1$ to neighbor u_1^q. As $|Q_2| \geq 2$, every vertex in Q has a neighbor with role k. Hence, we can finish off the role assignment by assigning roles to the remaining vertices of each copy H^q of H as follows. For convenience, we remove the superscript q. We map each path S, T, T' to the path $1 \cdots k-2$, where $r(s_i) = r(t_i) = r(t'_i) = i$

for $i = 1, \ldots, k-2$. If u_1 received role $k-2$ we assign u_i role $k-1-i$ for $i = 2, \ldots, k-2$ and we assign u_{k-2+i} role $i+1$ for $i = 1, \ldots, k-2$. Furthermore, we assign role $k-1$ to c, d, w, and role k to a, b, x, y, z. If u_1 received role $k-1$, it already has a neighbor with role k (namely its neighbor in Q). Then we assign u_i role $k-i$ for $i = 2, \ldots, k-1$ and we assign u_{k-1+i} role $i+1$ for $i = 1, \ldots, k-3$. Furthermore, we assign role $k-1$ to c, d, w, x, and role k to a, b, y, z.

To prove the converse statement, suppose G has a k-role assignment r. As we have shown above, by construction, G must have an R^*-role assignment. Then each p_i^S must have role i for $i = 1, \ldots, k-2$. Then $r(S) = k-1$ for each $S \in \mathcal{S}$, and each S must have a neighbor in Q with role $k-1$ and a neighbor in Q with role k. We define $Q_1 = \{q \in Q \mid r(q) = k-1\}$ and $Q_2 = Q \setminus Q_1$. Then we find that (Q_1, Q_2) is a 2-coloring of $(Q, \mathcal{S})$. This completes the proof of Theorem 6. □

4 Conclusions

We have settled an open problem of Sheng [24] by presenting a linear time algorithm that decides whether a chordal graph $G = (V, E)$ has a 2-role assignment. We showed that for any fixed $k \geq 3$ the k-Role Assignment problem stays NP-complete even for the class of chordal graphs.

Role assignments are also studied in topological graph theory. There, a graph G is called an *emulator* of a graph R if G has an R-role assignment. One of the important questions is which graphs allow finite planar emulators; see for example the recent paper of Rieck and Yamashita [22] for nice developments in this area. An interesting question is the computational complexity of the k-Role Assignment problem for planar graphs. The answer to this question is already unknown for $k = 2$.

Acknowledgements. The authors would like to thank Hans Bodlaender and the two anonymous referees for useful comments. The second author also thanks Jiří Fiala for fruitful discussions on the subject.

References

1. Abello, J., Fellows, M. R., and Stillwell, J. C. On the complexity and combinatorics of covering finite complexes. *Australian Journal of Combinatorics 4* (1991) 103–112.
2. Angluin, D. Local and global properties in networks of processors. In *12th ACM Symposium on Theory of Computing* (1980) 82–93.
3. Angluin, D., and Gardiner, A. Finite common coverings of pairs of regular graphs. *Journal of Combinatorial Theory B 30* (1981) 184–187.
4. Bernstein, P. A., and Goodman, N. Power of natural semijoins. *SIAM Journal on Computing 10* (1981) 751–771.
5. Biggs, N. Constructing 5-arc transitive cubic graphs. *Journal of London Mathematical Society II 26* (1982) 193–200.
6. Blair, J. R. S, and Peyton, B. An introduction to chordal graphs and clique trees. *Graph Theory and Sparse Matrix Computation 56* (1993) 1–29.

7. Bodlaender, H. L. The classification of coverings of processor networks. *Journal of Parallel Distributed Computing 6* (1989) 166–182.
8. Chalopin, J., Métivier, Y., and Zielonka, W. Local computations in graphs: the case of cellular edge local computations. *Fundamenta Informaticae 74* (2006) 85–114.
9. Everett, M. G., and Borgatti, S. Role colouring a graph. *Mathematical Social Sciences 21* (1991) 183–188.
10. Fiala, J., and Kratochvíl, J. Complexity of partial covers of graphs. In *12th International Symposium on Algorithms and Computation (ISAAC 2001)* Lecture Notes in Computer Science 2223 (2001) 537–549.
11. Fiala, J., and Kratochvíl, J. Partial covers of graphs. *Discussiones Mathematicae Graph Theory 22* (2002) 89–99.
12. Fiala, J., Kratochvíl, J., and Kloks, T. Fixed-parameter complexity of λ-labelings. *Discrete Applied Mathematics 113* (2001) 59–72.
13. Fiala, J., and Paulusma, D. A complete complexity classification of the role assignment problem. *Theoretical Computer Science 349* (2005) 67–81.
14. Fiala, J., and Paulusma, D. Comparing universal covers in polynomial time. *Theory of Computing Systems 46* (2010) 620–635.
15. Galinier, P., Habib, M., and Paul, P. Chordal Graphs and Their Clique Graphs. In *21st International Workshop on AGraph-Theoretic Concepts in Computer Science (WG 1995)* Lecture Notes in Computer Science 1017 (1995) 358–371.
16. Garey, M. R., and Johnson, D. S. *Computers and Intractability.* W. H. Freeman and Co., New York, 1979.
17. Gavril, F. Algorithms for minimum coloring, maximum clique, minimum covering by cliques and maximum independent set of a chordal graph. *SIAM Journal on Computing 1* (1972) 180–187.
18. Gavril, F. The intersection graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory, Series B 16* (1974) 47–56.
19. Kratochvíl, J., Proskurowski, A., and Telle, J. A. Covering regular graphs. *Journal of Combinatorial Theory B 71* (1997) 1–16.
20. Okamoto, Y., Takeaki, U., and Uehara, R. Counting the number of independent sets in chordal graphs. *Journal of Discrete Algorithms 6* (2008) 229–242.
21. Pekeč, A., and Roberts, F. S. The role assignment model nearly fits most social networks. *Mathematical Social Sciences 41* (2001) 275–293.
22. Rieck, Y., and Yamashita, Y. Finite planar emulators for $K_{4,5} - 4K_2$ and Fellows' conjecture. *European Journal of Combinatorics 31* (2010) 903–907.
23. Roberts, F. S., and Sheng, L. How hard is it to determine if a graph has a 2-role assignment? *Networks 37* (2001) 67–73.
24. Sheng, L. 2-Role assignments on triangulated graphs. *Theoretical Computer Science 304* (2003) 201–214.