

An Optimal Ancestry Scheme and Small Universal Posets*

Pierre Fraigniaud

CNRS and Univ. Paris Diderot
pierre.fraigniaud@liafa.jussieu.fr

Amos Korman

CNRS and Univ. Paris Diderot
amos.korman@liafa.jussieu.fr

ABSTRACT

In this paper, we solve the *ancestry problem*, which was introduced more than twenty years ago by Kannan et al. [STOC '88], and is among the most well-studied problems in the field of informative labeling schemes. Specifically, we construct an *ancestry labeling scheme* for n -node trees with label size $\log_2 n + O(\log \log n)$ bits, thus matching the $\log_2 n + \Omega(\log \log n)$ bits lower bound given by Alstrup et al. [SODA '03]. Besides its optimal label size, our scheme assigns the labels in linear time, and guarantees that any ancestry query can be answered in constant time.

In addition to its potential impact in terms of improving the performances of XML search engines, our ancestry scheme is also useful in the context of partially ordered sets. Specifically, for any fixed integer k , our scheme enables the construction of a *universal* poset of size $O(n^k \log^{4k} n)$ for the family of n -element posets with tree-dimension at most k . This bound is almost tight thanks to a lower bound of $n^{k-o(1)}$ due to Alon and Scheinerman [Order '88].

Categories and Subject Descriptors

G.2.2 [Discrete Mathematics]: Graph Theory—*Network problems, Graph labeling*; E.1 [Data structures]: [Distributed data structures, Trees]

General Terms

Algorithms, Theory.

Keywords

XML, informative labeling schemes, partially ordered sets.

*This research is supported in part by the ANR projects ALADDIN and PROSE, and by the INRIA project GANG.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

STOC'10, June 5–8, 2010, Cambridge, Massachusetts, USA.
Copyright 2010 ACM 978-1-4503-0050-6/10/06 ...\$10.00.

1. INTRODUCTION

1.1 Background

How to represent a graph in a compact manner is a fundamental data structure question. In most traditional graph representations, the names (or identifiers) given to the nodes serve merely as pointers to entries in a data structure, and thus reveal no information about the graph structure per se. Hence, in a sense, some memory space is wasted for the storage of content-less data. In contrast, Kannan, Naor, and Rudich [23] introduced the notion of *informative labeling schemes*, which involves a mechanism for assigning short, yet informative, labels to nodes. Specifically, the goal of such schemes is to assign labels to nodes in such a way that allows one to infer information regarding any two nodes *directly from their labels*. Hence in essence, this mechanism bases the entire graph representation on the set of labels alone. One important question in the framework of informative labeling schemes is how to efficiently encode the ancestry relation in trees. This was formalized in the seminal work of Kannan et al. [23] as follows.

The ancestry problem.

Given any n -node rooted tree T , label the nodes of T using the shortest possible labels (bit strings) such that, given any pair of nodes u and v in T , one can determine whether u is an ancestor of v in T by merely inspecting the labels of u and v .

The following simple ancestry scheme was suggested in [23]. Given a rooted n -node tree T , perform a DFS traversal in T starting at the root, and provide each node u with a DFS number, $\text{DFS}(u)$, in the range $[0, n - 1]$. (Recall, in a DFS traversal, a node is visited before any of its children, thus the DFS number of a node is smaller than the DFS number of any of its descendants). The label of a node u is simply the interval $I(u) = [\text{DFS}(u), \text{DFS}(v)]$, where v is the descendant of u with the largest DFS number. An ancestry query then amounts to an interval containment query between the corresponding labels: a node u is an ancestor of a node v if and only if $I(v) \subset I(u)$. Clearly, the *label size*, namely, the maximal size of a label assigned by this ancestry scheme to any node in any n -node tree, is bounded by $2 \log n$ bits¹.

The $2 \log n$ bits scheme of [23] initiated an extensive research [1, 3, 7, 13, 21, 22, 26] whose goal was to reduce the label size of ancestry schemes as much as possible. The

¹All logarithms in this paper are taken in base 2.

main motivation behind these works lies in the fact that a small improvement in the label size of ancestry schemes can contribute to a significant improvement in the performances of XML search engines. Indeed, to implement sophisticated queries, XML documents are viewed as labeled trees, and typical queries over the documents amount to testing relationships between document items, which correspond to ancestry queries among the corresponding tree nodes [2, 10, 31, 32]. XML search engines process such queries using an index structure that summarizes this ancestry information. To allow good performances, a large portion of the XML index structure resides in the main memory. A main factor which determines the index size is the length of the labels. Thus, due to the enormous size of the Web data, even a small reduction in the label size can contribute significantly to both memory cost reduction and performance improvement. A more detailed explanation regarding this application can be found in various papers on ancestry labeling schemes, e.g., [1, 22].

In [4], Alstrup et al. constructed a lower bound of $\log n + \Omega(\log \log n)$ bits for the label size of an ancestry scheme. On the other hand, the current state of the art upper bound is $\log n + O(\sqrt{\log n})$ bits [1]. Thus, a large gap is still left between the best known upper and lower bounds on the label size of ancestry labeling schemes. This paper closes this gap, by constructing an ancestry labeling scheme whose label size matches the aforementioned lower bound.

In addition to its potential impact on the performances of XML search engines, our ancestry scheme finds applications also in the context of *partially ordered sets* (posets). To elaborate more on this application, let us first recall several notions from order theory. A poset (X, \leq_X) contains a poset (Y, \leq_Y) as an *induced suborder* if there exists an injective mapping $\phi : Y \rightarrow X$ such that for any two elements $a, b \in Y$: we have $a \leq_Y b \iff \phi(a) \leq_X \phi(b)$. A poset (X, \leq) is called *universal* for a family of posets \mathcal{P} if (X, \leq) contains every poset in \mathcal{P} as an induced suborder. Given a family of posets \mathcal{P} , a natural problem consists of determining what is the smallest size (in term of number of elements) of a universal poset for \mathcal{P} . When considering infinite posets, it is known that a countable universal poset for the family of all countable posets exists. This classical result was proved several times [11, 16, 17] and, in fact, as mentioned in [15], has motivated the whole research area of category theory.

If (X, \leq) and (X, \leq') are orders on the set X , we say that (X, \leq') is an *extension* of (X, \leq) if $x \leq y$ implies $x \leq' y$ for any two elements $x, y \in X$. A common way to characterize a poset (X, \leq) is by its *dimension*, i.e., the smallest number of linear (i.e., total order) extensions of (X, \leq) the intersection of which gives rise to (X, \leq) [27]. It can be easily seen that the smallest size of a universal poset for the family of n -element posets with dimension at most k is at most n^k . Another way of characterizing a poset (X, \leq) is by its *tree-dimension*. A poset (X, \leq) is a *tree*² [9, 29] if, for every pair x and y of incomparable elements in X , there does not exist an element $z \in X$ such that $x \leq z$ and $y \leq z$. (Observe that the Hasse diagram [27] of a tree poset is a forest of rooted trees). The *tree-dimension* [9] of a poset (X, \leq) is the smallest number of tree extensions of (X, \leq) the intersection of which gives rise to (X, \leq) . It is well known that the dimension of a poset is at most twice its tree-dimension.

²Note that the term “tree” for ordered sets is used in various senses in the literature, see e.g., [28].

Thus, the smallest size of a universal poset for the family of all n -element posets with tree-dimension at most k is at most n^{2k} . On the other hand, Alon and Scheinerman [8] showed that for fixed k , the number of n -element posets of dimension at most k is $n^{n(k-o(1))}$. Since the dimension of a poset is at least its tree-dimension, it follows³ that a universal poset for the family of all n -element posets with tree-dimension at most k has number of elements at least $n^{k-o(1)}$. As we show, it turns out that the real bound is much closer to this lower bound than to the n^{2k} upper bound.

1.2 Our results

This paper constructs an ancestry labeling scheme for n -node rooted trees, whose label size is $\log n + O(\log \log n)$ bits. By doing that, we solve the ancestry problem which is among the main open problems in the field of informative labeling schemes. Moreover, our scheme assigns the labels to the nodes of any tree in linear time, and guarantees that any ancestry query is answered in constant time.

In addition, we establish a simple relation between compact ancestry schemes and small universal posets. Specifically, we show that there exists a *consistent* ancestry labeling scheme for n -node forests with label size ℓ (see next section for the definition of consistent ancestry schemes for forests) if and only if, for any integer $k \geq 1$, there exists a universal poset with $2^{k\ell}$ elements for the family of n -element posets with tree-dimension at most k . Using this equivalence, and slightly modifying our ancestry scheme, we prove that for any integer k , there exists a universal poset of size $O(n^k \log^{4k} n)$ for the family of all n -element posets with tree-dimension at most k .

1.3 Related work

As mentioned before, following the $2 \log n$ -bit ancestry scheme in [23], a considerable amount of research has been devoted to improve the upper bound on the label size as much as possible. Specifically, [3] gave a first non-trivial upper bound of $\frac{3}{2} \log n + O(\log \log n)$ bits. In [21], a scheme with label size $\log n + O(d\sqrt{\log n})$ bits was constructed to detect ancestry only between nodes at distance at most d from each other. An ancestry labeling scheme with label size of $\log n + O(\log n / \log \log n)$ bits was given in [26]. The current state of the art upper bound of $\log n + O(\sqrt{\log n})$ bits was given in [7] (that scheme is described in detail in the journal publication [1] joint with [3]).

Following the aforementioned results on ancestry labeling schemes for general rooted trees, two other works were published, which focused on particular types of trees. Specifically, an experimental comparison of different ancestry labeling schemes on XML trees that appear in real life can be found in [22]. More recently, [13] gave an ancestry labeling scheme that is efficient for trees of small depth: for n -node trees with depth d , the scheme uses labels of size $\log n + 2 \log d + O(1)$ bits. This study was motivated by the observation that in real-life data, the typical depth of an XML tree is very small, which may strengthen the assumption that most nodes in the collection of all XML trees

³To see this, observe that the proof of Theorem 1 in [8] implies that the number of non-isomorphic n -element posets with tree-dimension at most k is at least $n^{n(k-o(1))}/n!$. Let M be the size of a universal poset for the family of n -element posets with tree-dimension at most k . We must have $n^{n(k-o(1))}/n! \leq \binom{M}{n}$, and thus $M \geq n^{k-o(1)}$.

belong to trees of small depth. If this is indeed the case, the scheme of [13] would incur a very good *average* label size for XML trees. This would be efficient if one considers a variable-length representation of the labels, but not very useful if the representation is fixed-length (the same amount of space must be used for each label) — see [22] for a more detailed discussion on the label representation.

The following *parenthood* problem can be viewed as a variant of the ancestry problem: given a rooted tree T , label the nodes of T in the most compact way such that one can determine whether u is a *parent* of v in T by merely inspecting the corresponding labels. The parenthood problem was also introduced in [23], and a very simple parenthood scheme was constructed there, using labels of size at most $2 \log n$ bits. (Actually, [23] considered *adjacency* schemes in trees rather than parenthood schemes, however, such schemes are equivalent up to a constant number of bits in the label size⁴). By now, the parenthood problem is almost completely closed thanks to Alstrup and Rauhe [6], who constructed a parenthood scheme for n -node trees with label size $\log n + O(\log^* n)$ bits. In particular, this bound indicates that encoding ancestry in trees is strictly more costly than encoding parenthood.

Informative labeling schemes were also proposed for other decision problems on trees and on graphs in general, including distance [4, 14, 25], routing [12, 26], flow [18, 20], vertex connectivity [18, 19], and nearest common ancestor [5, 24].

The $2 \log n$ -bit ancestry scheme of [23] is consistent, and thus provides yet another evidence for the existence of a universal poset with n^{2k} elements for the family of all n -element posets with tree-dimension at most k . It is not clear whether the ancestry schemes in [3, 7, 21, 26] can be somewhat modified to be consistent and still maintain the same label size. However, even if this is indeed the case, the universal poset for the family of all n -element posets with tree-dimension at most k that would be obtained from those schemes, would be of size $\Omega(n^k 2^{k \sqrt{\log n}})$.

The lower bound of [4] implies a lower bound of $\Omega(n \log n)$ for the number of elements in a universal poset for the family of n -element posets with tree-dimension 1. As mentioned earlier, for fixed $k > 1$, the result of Alon and Scheinerman [8] implies a lower bound of $n^{k-o(1)}$ for the number of elements in a universal poset for the family of n -element posets with tree-dimension at most k .

2. PRELIMINARIES

Let T be a *rooted* tree, i.e., a tree with a designated node r referred as the *root* of T . The *depth* of a node $u \in T$ is defined as the distance from u to the root of T , i.e., the number of edge traversals from u to the root. In particular, the depth of the root is 0. For two nodes u and v in T , we say that u is an *ancestor* of v if $u \neq v$ and u is one of the nodes on the shortest path connecting v and r in T . For every non-root node u , let $\text{parent}(u)$ denote the parent of

⁴To see this equivalence, observe that one can construct a parenthood scheme from an adjacency scheme in trees, as follows. Given a rooted tree T , first label the nodes of T using the adjacency scheme (with ignores the fact that T is rooted). Then, for each node u , in addition to the label given to it by the adjacency scheme, add two more bits, for encoding $d(u)$, the distance from u to the root calculated modulo 3. Now the parenthood scheme follows by observing that for any two nodes u and v in a tree, u is a parent of v iff u and v are adjacent and $d(u) = d(v) - 1$ modulo 3.

u , i.e., the ancestor of u at distance 1 from it. A node v is a *descendant* of u if and only if u is an ancestor of v . The *size* of T , denoted by $|T|$, is the number of nodes in T . The *weight* of a node $u \in T$, denoted by $\text{weight}(u)$, is defined as 1 plus the number of descendants of u , i.e., $\text{weight}(u)$ is the size of the subtree hanging down from u . In particular, the weight of the root is $\text{weight}(r) = |T|$. For every integer n , let $\mathcal{T}(n)$ denote the family of all rooted trees of size at most n . For two nodes u and v in some forest F of rooted trees, we say that u is an *ancestor* of v in F iff u and v belong to the same rooted tree in F and u is an ancestor of v in that tree. For every integer n , let $\mathcal{F}(n)$ denote the family of all forests of rooted trees, where each forest in $\mathcal{F}(n)$ has at most n nodes.

For two nodes u and v in a tree T , let $P[u, v]$ denote the shortest path connecting u and v in T (including u and v), and let $P[u, v] = P[u, v] \setminus \{v\}$. For two integers $a \leq b$, let $[a, b]$ denote the set of integers $\{a, a + 1, \dots, b\}$. We refer to this set as an *interval*. For two intervals $I = [a, b]$ and $I' = [a', b']$, we say that $I \prec I'$ if $b < a'$. The *size* of an interval $I = [a, b]$ is $|I| = b - a + 1$, namely, the number of integers in I .

An *ancestry labeling scheme* $(\mathcal{M}, \mathcal{D})$ for a family \mathcal{F} of forests of rooted trees is composed of the following components:

1. A *marker* algorithm \mathcal{M} that assigns labels (i.e., bit strings) to the nodes of all forests in \mathcal{F} .
2. A *decoder* algorithm \mathcal{D} that given any two labels ℓ_1 and ℓ_2 in the output domain of \mathcal{M} , returns a boolean $\mathcal{D}(\ell_1, \ell_2)$.

These components must satisfy that if $L(u)$ and $L(v)$ denote the labels assigned by the marker algorithm to two nodes u and v in some rooted forest $F \in \mathcal{F}$, then

$$\mathcal{D}(L(u), L(v)) = 1 \iff u \text{ is an ancestor of } v \text{ in } F.$$

It is important to note that the decoder algorithm \mathcal{D} is independent of the forest F . That is, given the labels of two nodes, the decoder algorithm decides the ancestry relationship between the corresponding nodes without knowing to which forest in \mathcal{F} they belong.

The most common complexity measure used for evaluating an ancestry scheme is the *label size*, that is, the maximum number of bits in a label assigned by \mathcal{M} , taken over all nodes in all forests in \mathcal{F} . When considering the *query time* of the decoder algorithm, we use the RAM model of computation, and assume that the length of a computer word is $\Omega(\log n)$ bits. Similarly to previous works on ancestry schemes, our decoder algorithm uses only basic RAM operations (which are assumed to take constant time). Specifically, the basic operations used by our decoder algorithm are the following: addition, subtraction, left/right shifts, less-than comparisons, and extraction of the least significant bit. Our scheme avoids the sometimes more costly operations such as multiplication or division.

Let \mathcal{F} be a family of forests of rooted trees. We say that an ancestry labeling scheme $(\mathcal{M}, \mathcal{D})$ for \mathcal{F} is *consistent* if (1) the marker algorithm \mathcal{M} assigns pairwise distinct labels to the nodes of any forest $F \in \mathcal{F}$, and (2) the decoder algorithm \mathcal{D} satisfies the following conditions, for any three labels ℓ_1, ℓ_2 and ℓ_3 in the output domain of \mathcal{M} :

- *Anti-symmetry*: if $\mathcal{D}(\ell_1, \ell_2) = 1$ then $\mathcal{D}(\ell_2, \ell_1) = 0$, and

- *Transitivity*: if $\mathcal{D}(\ell_1, \ell_2) = 1$ and $\mathcal{D}(\ell_2, \ell_3) = 1$ then $\mathcal{D}(\ell_1, \ell_3) = 1$.

Note that by the definition of an ancestry scheme $(\mathcal{M}, \mathcal{D})$, the decoder algorithm \mathcal{D} trivially satisfies the two conditions above if $\ell_i = L(u_i)$ for $i = 1, 2, 3$, and u_1, u_2 and u_3 belong to the same forest in \mathcal{F} .

3. THE ANCESTRY LABELING SCHEME

This section is dedicated to the proof of the following theorem:

THEOREM 1. *There exists an ancestry labeling scheme for $\mathcal{T}(n)$ with label size $\log n + O(\log \log n)$ bits. Moreover, given a tree $T \in \mathcal{T}(n)$, the labels can be assigned to the nodes of T in $O(n)$ time, and any ancestry query can be answered in constant time.*

A natural approach for constructing an ancestry labeling scheme is to label each node u in the given tree T by a pointer to some interval $I(u)$, in a way such that for any two nodes u and v in T , u is an ancestor of v if and only if $I(v) \subset I(u)$. Thus, to check ancestry, the decoder algorithm can simply apply the interval containment test. This approach relates the partial order defined by ancestry to the natural partial order defined on intervals through containment. It was used, for example, by the original ancestry scheme of [23]. There, the set of intervals U that could potentially be used to label nodes is of size at most n^2 , and the $2 \log n$ bits label size follows. The idea behind the ancestry scheme of [13] was to find a much smaller set of intervals U to which one can map the nodes of trees in such an order preserving manner. Indeed, this method works well when the depths of the given trees are small. Unfortunately, however, the approach is no longer efficient if the input trees can have long paths. Informally, enforcing the decoder algorithm to be merely an interval containment test imposes a strong constraint on the way the intervals must be organized in U . For arbitrary trees, we could not find a way to bypass this constraint while keeping U small. Instead, this paper introduces a decoder algorithm which performs according to a different principle. At a very high level, our new decoder algorithm exploits the fact that intervals can be partially ordered not only by the containment relation, but also by the relation “ \prec ” introduced in the previous section. By combining these two partial orders on intervals, we define a new relation⁵ between *pairs* of intervals. This relation is defined such that the nodes of any tree can be mapped to a limited set of pairs of intervals in an “order preserving manner”, i.e., the ancestry relation in the tree is compatible with the new relation defined on pairs of intervals.

More formally, given any node u in some rooted tree T , we first associate u with a *supervisor* node, denoted by $\text{sup}(u)$, which is either u itself or one of its ancestors. For this purpose, we first mark each node as either *heavy* or *light* as follows. For every non-leaf node u of T , let $H(u)$ be the set of children v of u that satisfy $\text{weight}(v) \geq \text{weight}(w)$ for

⁵As explained in Section 4, this relation is “almost” a partial order on the set of labels (each label corresponds to a pair of intervals) which are assigned by the marker algorithm. Indeed, only a slight modification of the decoder algorithm (for making our ancestry scheme consistent) suffices to induce a partial order on that set.

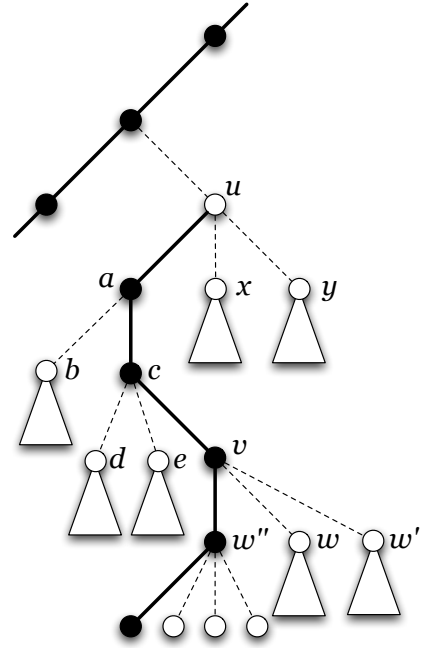


Figure 1: The heavy nodes are depicted in black, while the light nodes are depicted in white. In the figure, $\text{sup}(v) = u$ and $\text{sup}(w) = w$. We have $\text{parent}(w) = v$. Hence, the local quasi ancestors of w are the set of nodes $\text{lqa}(w) = \{x, y, a, b, c, d, e, v, w'\}$ if $\text{DFS}(w) > \text{DFS}(w')$, and $\text{lqa}(w) = \{x, y, a, b, c, d, e, v\}$, otherwise. Similarly, we have $\text{lqa}(x) = \emptyset$ if $\text{DFS}(y) > \text{DFS}(x)$, and $\text{lqa}(x) = \{y\}$, otherwise.

every child w of u . Among the nodes in $H(u)$, select an arbitrary node, and call it *heavy*. A node which is not heavy is called *light*. (In particular, the root is light). For each node $u \in T$, define the *supervisor* of u , denoted by $\text{sup}(u)$, as the light node of largest depth on the path $P[u, r]$ connecting u to the root r . Note that if u is light then $\text{sup}(u)$ is u itself; in particular, $\text{sup}(r) = r$, and $\text{sup}(\text{sup}(u)) = \text{sup}(u)$. Observe also that if u is an ancestor of v , then either $\text{sup}(u) = \text{sup}(v)$, or $\text{sup}(u)$ is an ancestor of $\text{sup}(v)$. See Figure 1.

The marker algorithm \mathcal{M} now assigns an interval $I(u)$ to each node $u \in T$, and associates u with the pair of intervals $I(u)$ and $I(\text{sup}(u))$. More formally, the marker algorithm \mathcal{M} assigns a label $L(u)$ to each node u , in a way such that the intervals $I(u)$ and $I(\text{sup}(u))$ can be extracted from $L(u)$ in constant time. Given the labels $L(u)$ and $L(v)$ assigned by our marker algorithm \mathcal{M} to two nodes u and v in some rooted tree, our boolean decoder algorithm \mathcal{D} outputs 1 if and only if the “decoding conditions” defined below hold at u w.r.t. v .

DEFINITION 1. *Consider a set of pairwise distinct intervals $\{I(u), u \in T\}$ for a tree T (i.e., $I(u) \neq I(v)$ for any two different nodes u and v). We say that the decoding conditions hold at u w.r.t. v if and only if:*

- D1: $I(v) \subset I(\text{sup}(u))$, and
- D2: $I(u) \prec I(v)$ or $I(u) = I(\text{sup}(u))$.

Before dwelling into the details of our construction, we would like to point out that storing both intervals $I(u)$ and $I(\text{sup}(u))$ explicitly in the label of u may consume too many bits. In fact, it will follow from our construction that the encoding of a single interval may already consume $\log n + O(\log \log n)$ bits, in worst case. To overcome this problem, we shall use the fact that for every node u , $I(u) \subseteq I(\text{sup}(u))$. This fact, together with specific constraints which are imposed on the intervals, will allow the decoder algorithm to recover the interval $I(\text{sup}(u))$ from the interval $I(u)$ using only additional $O(\log \log n)$ bits.

3.1 The marker algorithm \mathcal{M}

Assume without loss of generality that n is a power of 2, and fix a tree $T \in \mathcal{T}(n)$. For our scheme to be correct, the marker algorithm \mathcal{M} must assign the interval $I(u)$ to each node u in T in a way such that for every two nodes u and v in T , u is an ancestor of v if and only if the decoding conditions hold at u w.r.t. v . For this purpose, we first show that it is sufficient to provide an assignment of intervals that satisfies more “local” conditions.

The local conditions.

Let us first assign numbers from 0 to $n - 1$ to the nodes according to a DFS traversal that starts at the root, and visits light children first. We denote by $\text{DFS}(u)$ the DFS number of u . Let $P_u = P[\text{parent}(u), \text{sup}(\text{parent}(u))]$. We define the *local quasi-ancestors* of a non-root node u , denoted by $\text{lqa}(u)$, as all nodes in P_u , together with their light children, but removing u , $\text{sup}(\text{parent}(u))$, and all nodes that have DFS numbers higher than u . See Figure 1 for an example. Note that the local quasi-ancestors of a node may not form a connected subtree of T .

DEFINITION 2. *Let us consider a set of pairwise distinct intervals $\{I(u), u \in T\}$ for a tree T . For every non-root node u , we say that u satisfies the local conditions if the two conditions below are satisfied:*

- L1: $I(u) \subseteq I(\text{sup}(u)) \cap I(\text{sup}(\text{parent}(u)))$;
- L2: $I(x) \prec I(u)$ for every $x \in \text{lqa}(u)$.

The following lemma relates the decoding conditions to the local conditions.

LEMMA 1. *If every non-root node u satisfies the local conditions L1 and L2, then for every two different nodes u and v , u is an ancestor of v if and only if the decoding conditions D1 and D2 hold at u w.r.t. v .*

PROOF. Let us assume that every non-root node u satisfies the local conditions L1 and L2. We first prove the following claim:

Claim 1. For any light node u , and any descendent v of u , we have $I(v) \subset I(u)$.

We establish the claim by induction on the distance between u and v . If $\text{dist}(u, v) = 1$ then the claim holds as v satisfies L1 and u is light. Fix $d \geq 1$ and assume that the claim holds for all pairs of nodes u and v such that $\text{dist}(u, v) \leq d$. Consider now a pair of nodes u and v such that $\text{dist}(u, v) = d + 1$. If $u = \text{sup}(\text{parent}(v))$ then the claim follows as v satisfies L1. On the other hand, if $u \neq \text{sup}(\text{parent}(v))$ then there exists a light node $w \notin \{u, v\}$ on the shortest path

connecting u to v , namely $w = \text{sup}(\text{parent}(v))$. The claim then follows by induction.

To establish the lemma, consider first the case where u is an ancestor of v . Since v is a descendent of u , either $\text{sup}(v) = \text{sup}(u)$ or $\text{sup}(v)$ is a descendent of $\text{sup}(u)$. Thus, by Claim 1, $I(v) \subseteq I(\text{sup}(v)) \subseteq I(\text{sup}(u))$. Since $v \neq u$, $I(v) \subset I(\text{sup}(u))$, i.e., D1 follows. If u is light then the fact that u and v satisfy D2 follows trivially from the fact that, in this case, $\text{sup}(u) = u$. So assume now that u is heavy. If $u \in \text{lqa}(v)$, then D2 follows from L2. On the other hand, if $u \notin \text{lqa}(v)$, then there exists a light node w that is an ancestor of v , and such that $u \in \text{lqa}(w)$. D2 follows by combining L2 with Claim 1.

Consider now the case where u is not an ancestor of v . We need to show that either D1 or D2 does not hold. Let w be the light node of largest depth on the path $P[v, r]$, which is an ancestor of u . If w is v itself then $\text{sup}(u)$ is either v or a descendant of v . Therefore, by Claim 1, we have $I(\text{sup}(u)) \subseteq I(v)$, and thus D1 is not satisfied. In the remaining of the proof we thus assume that $w \neq v$. For a node x which is a descendant of w , and which satisfies $\text{sup}(x) \neq w$, let $f(x)$ be the light node of smallest depth on the path $P[x, w]$.

Assume first that $\text{sup}(u) = w$. If also $\text{sup}(v) = w$ then $v \in \text{lqa}(u)$, and thus, by L2, $I(v) \prec I(u)$. Since $u \neq w$, we have $I(u) \neq I(\text{sup}(u))$, and therefore D2 is not satisfied. If, on the other hand, $\text{sup}(v) \neq w$ then we have $f(v) \in \text{lqa}(u)$ and, by Claim 1, $I(v) \subset I(f(v))$. In a way similar to the previous case where $\text{sup}(v) = w$, we then get that D2 is not satisfied.

Assume now that $\text{sup}(u) \neq w$. We have $I(\text{sup}(u)) \subseteq I(f(u))$. If v is an ancestor of $f(u)$ then $v \in \text{lqa}(f(u))$. Consequently, $I(v) \prec I(f(u))$ and thus D1 does not hold. On the other hand, if v is not an ancestor of $f(u)$ then we are left with two cases: $\text{sup}(v) = w$ and $\text{sup}(v) \neq w$. If $\text{sup}(v) = w$ then $I(f(u)) \prec I(v)$ since $f(u) \in \text{lqa}(v)$. Thus D1 does not hold. Finally, if $\text{sup}(v) \neq w$ then either $f(u) \in \text{lqa}(f(v))$ or $f(v) \in \text{lqa}(f(u))$. Since $I(\text{sup}(u)) \subseteq I(f(u))$ and $I(v) \subseteq I(f(v))$, it follows that D1 does not hold. \square

The interval assignment.

By Lemma 1, one of our goals is to let the marker algorithm assign intervals that satisfy the local conditions at each non-root node. For integers a, b and k , let

$$I_{k,a,b} = [2^k a, 2^k(a+b)].$$

For $k \in [1, \log n]$, define the set of intervals:

$$\mathcal{I}_k = \{I_{i,a,b} \mid i \in [1, k], a \in [1, n \log n / 2^{i-2}], \text{ and } b \in [1, 4i]\}$$

Denote $\mathcal{I} = \bigcup_{k=1}^{\log n} \mathcal{I}_k$.

DEFINITION 3. *Let $T \in \mathcal{T}(n)$. A mapping $I : V(T) \rightarrow \mathcal{I}$ is called a legal interval-mapping if I is one-to-one, and $\{I(u), u \in T\}$ satisfies the local conditions at each non-root node of T .*

In order to show that there exists a legal interval-mapping from every tree in $\mathcal{T}(n)$ into \mathcal{I} , we use the following notation. For any interval $J \subseteq [1, 4n \log n]$, and, for any k , $1 \leq k \leq \log n$, let

$$\mathcal{I}_k(J) = \{I_{i,a,b} \in \mathcal{I}_k \mid I_{i,a,b} \subseteq J\}.$$

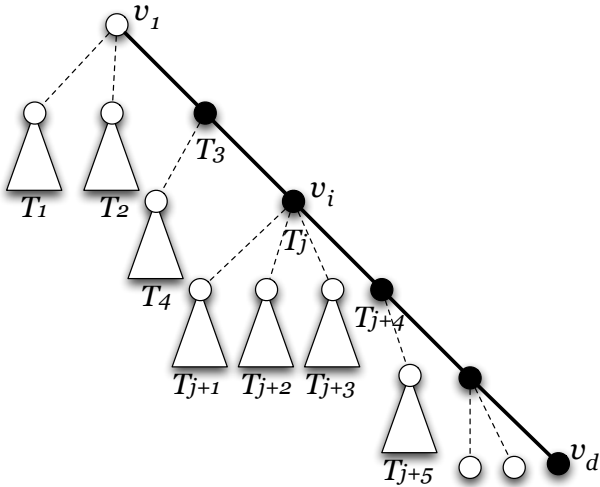


Figure 2: Tree decomposition as in the proof of Lemma 2. In the figure, the DFS traversal that visits light nodes first is supposed to proceed by visiting children from left to right.

LEMMA 2. For every $k \in [1, \log n]$, every tree $T \in \mathcal{T}(2^k)$, and every interval $J \subseteq [1, 4n \log n]$, such that $|J| = 4k|T|$, there exists a legal interval-mapping of T into $\mathcal{I}_k(J)$. Moreover this mapping can be computed in $O(|T|)$ time.

PROOF. We prove the lemma by induction on k . For $k = 1$, the fact that the lemma holds can be easily verified. Assume now that the lemma holds for k with $1 \leq k < \log n$, and let us show that it also holds for $k + 1$. Clearly, if $|T| \leq 2^k$ then we are done by induction. Consider now the case where T is of size $2^k < |T| \leq 2^{k+1}$, and let $J \subseteq [1, 4n \log n]$ be an interval, such that $|J| = 4(k+1)|T|$. Our goal is to construct a legal-interval mapping of T into $\mathcal{I}_{k+1}(J)$.

We make use of the following decomposition of T . Let H be the path from the root of T to a leaf of T such that every non-root node in H is heavy. Let v_1, v_2, \dots, v_d be the nodes of H , ordered top-down, i.e., v_1 is the root of T , v_d is a leaf of T , and for every $1 \leq i < d$, v_i is the parent of v_{i+1} .

For every $1 \leq i \leq d$, let $T_i^1, T_i^2, \dots, T_i^{t_i}$ be the rooted trees hanging down from the light children of v_i . (If v_i does not have any light child, which is the case, for example, for $i = d$, then this set of trees is empty, or, in other words, $t_i = 0$). One important property of these trees is that, for every i and j , $1 \leq j \leq t_i$, we have $|T_i^j| < |T|/2 \leq 2^k$.

Let us now group the nodes in $T \setminus \{r\}$ in disjoint trees T_1, T_2, \dots, T_m , where $m = d - 1 + \sum_{\ell=1}^d t_\ell$ as follows. A tree T_i is either a single heavy node v_j , for $j > 1$, or a subtree hanging down from a light child of some v_j , $j \geq 1$. Moreover, the trees are enumerated according to the DFS numbers of their roots as follows. Recall that $\text{DFS}(u)$ denotes the DFS number of node u in T . The trees are ordered such that if r_j denotes the root of T_j then $\text{DFS}(r_j) < \text{DFS}(r_{j+1})$ for all $j = 1, \dots, m - 1$. See Figure 2.

Consider now the given interval $J \subseteq [1, 4n \log n]$, and express it as $J = [\alpha, \alpha + 4(k+1)|T| - 1]$ for some integer α . Let a be the smallest integer such that $\alpha \leq a2^{k+1}$, and let b be the smallest integer such that $4k|T| \leq b2^{k+1}$. First, we

assign the root r to the interval $J' = [a2^{k+1}, (a+b)2^{k+1}]$. We now show that indeed $J' \in \mathcal{I}_{k+1}(J)$. By definition of a and b , we have

$$(a+b)2^{k+1} = 2^{k+2} + ((a-1) + (b-1))2^{k+1} \leq 2^{k+2} + \alpha + 4k|T| - 2.$$

Since $2^k < |T|$, we get that

$$(a+b)2^{k+1} < \alpha + (4k+4)|T| - 1.$$

Thus

$$J' = [a2^{k+1}, (a+b)2^{k+1}] \subset [\alpha, \alpha + (4k+4)|T| - 1] = J.$$

Therefore, since $a - 1 < \alpha/2^{k+1} < 4n \log n/2^{k+1}$, and since n is a power of 2, we get $1 \leq a \leq 4n \log n/2^{k+1}$. Combining this with the fact that $1 \leq b \leq 4k$, we obtain that $J' \in \mathcal{I}_{k+1}(J)$.

The rest of the nodes in T are mapped as follows. First note that $|J'| > 4k|T|$, and recall that $\sum_{i=1}^m |T_i| = |T| - 1$. Let J'_1, J'_2, \dots, J'_m be m pairwise disjoint consecutive sub-intervals of J' such that, for each $1 \leq i \leq m$, we have $|J'_i| = 4k|T_i|$, and for $1 \leq i < m$, $J'_i \prec J'_{i+1}$. For each $1 \leq i \leq m$, we now use the induction hypothesis to map the nodes in T_i to $\mathcal{I}_k(J'_i)$ via a legal interval-mapping (recall that $|T_i| \leq 2^k$).

The fact that the above recursive mapping can be performed in linear time is obvious. It remains to show that the above mapping of T into $\mathcal{I}_{k+1}(J)$ is indeed a legal interval-mapping. That is, we have to show that the set of intervals satisfies the local conditions L1 and L2 at each non-root node u . The fact that the local conditions hold at every node u in $T_i \setminus r_i$ follows from the induction hypothesis, since $\text{sup}(u)$, $\text{sup}(\text{parent}(u))$, and $\text{lqa}(u)$ are all contained in T_i . Finally, consider the root r_i of T_i . (Note that if r_i is heavy then $T_i = \{r_i\}$). L1 holds trivially for r_i because J' contains J'_i , and, by induction, the interval assigned to r_i is contained in the interval J'_i . To establish that L2 holds, first observe that $\text{lqa}(r_i) = \{r_1, \dots, r_{i-1}\}$. On the other hand, for every $j = 1, \dots, i-1$, $\text{DFS}(r_j) < \text{DFS}(r_i)$, and thus $J'_j \prec J'_i$. Hence L2 holds for r_i as well. Our mapping is thus a legal interval-mapping of T into $\mathcal{I}_{k+1}(J)$. The lemma follows. \square

COROLLARY 1. Let $T \in \mathcal{T}(n)$. There exists a legal interval-mapping of T into \mathcal{I} , which takes $O(|T|)$ time.

The label assignment.

After mapping T into \mathcal{I} using the legal-interval mapping given by Corollary 1, the marker algorithm \mathcal{M} assigns the label $L(u)$ to each node u as follows. Given a node u , the marker algorithm uses the first $\log n + \log \log n + O(1)$ least significant bits of $L(u)$ to encode the interval $I(u)$. This can be done explicitly as $I(u) = I_{k,a,b}$ for some $k \in [1, \log n]$, $a \in [1, n \log n/2^{k-2}]$ and $b \in [1, 4k]$. Specifically, the first $\lceil \log k \rceil$ bits in $L(u)$ are set to 0 and the following bit is set to 1. The next $2 \log k + O(1)$ bits are used for encoding k and b , and the following $\log n + \log \log n - k + O(1)$ bits to encode a .

Finally, the marker algorithm aims at encoding $I(\text{sup}(u))$ in the label of u . However, using the method above to encode $I(\text{sup}(u))$ would consume yet another $\log n + \log \log n + O(1)$ bits, which is obviously undesired. Instead, we use the following trick. Let k', a' , and b' be such that $I(\text{sup}(u)) = I_{k',a',b'}$. Clearly, $2 \log \log n + O(1)$ bits suffice to encode both k' and b' . To encode a' , the marker algorithm acts as

follows. Let $I(u) = [\alpha, \beta]$, and let a'' be the largest integer such that $2^{k'} a'' \leq \alpha$. Recall that by definition $I(\text{sup}(u)) = [2^{k'} a', 2^{k'}(a' + b')]$. We have, $a'' - 4 \log n \leq a'' - b'$ because $b' \leq 4 \log n$. Since $I(u) \subseteq I(\text{sup}(u))$, we also have $2^{k'}(a' + b') \geq \beta \geq \alpha \geq 2^{k'} a''$. Thus $a'' - b' \leq a'$. Finally, again since $I(u) \subseteq I(\text{sup}(u))$, we have $2^{k'} a' \leq \alpha$, and thus $a'' \geq a'$. Combining the above inequalities, we get that $a' \in [a'' - 4 \log n, a'']$. The marker algorithm now encodes the integer $t \in [0, 4 \log n]$ such that $a' = a'' - t$. This is done in consuming another $\log \log n + O(1)$ bits. Hence, we obtain:

LEMMA 3. *Given a tree $T \in \mathcal{T}(n)$, the marker algorithm \mathcal{M} assigns labels to the nodes of T in linear time, and each label is encoded using $\log n + 4 \log \log n + O(1)$ bits.*

3.2 The decoder algorithm \mathcal{D}

Now, we describe our decoder algorithm \mathcal{D} . Given the labels $L(u)$ and $L(v)$ assigned by \mathcal{M} to two different nodes in some tree T , the decoder algorithm \mathcal{D} needs to find whether u is an ancestor of v in T . (Observe that since each node receives a distinct label, the decoder algorithm can easily find out if u and v are in fact the same node, and, in this trivial case, it simply outputs 0.)

The decoder algorithm first extracts $I(u)$ by inspecting the first $\log n + \log \log n + O(1)$ least significant bits of $L(u)$. This is done as follows. Recall that $I(u) = I_{k,a,b}$ for some integers k, a and b . The decoder algorithm first extracts $\lceil \log k \rceil$ by finding the least significant bit that equals to 1. It then inspects the next $2 \log k + O(1)$ bits to obtain k and b , and the next $\log n + \log \log n - k + O(1)$ bits to extract a . Once $I(u) = [\alpha, \beta]$ has been reconstructed from $L(u)$, the decoder algorithm aims at extracting $I(\text{sup}(u))$. For this purpose, it first reconstructs k' and b' that have been explicitly encoded in the next $2 \log \log n + O(1)$ bits of $L(u)$. Then, it computes the largest integer a'' such that $2^{k'} a'' \leq \alpha$. The decoder algorithm then proceeds by extracting t , and computes $a' = a'' - t$. At this point, \mathcal{D} has reconstructed both $I(u)$ and $I(\text{sup}(u))$. Similarly, \mathcal{D} extracts $I(v)$ by inspecting $L(v)$.

Finally, the boolean decoder algorithm \mathcal{D} outputs 1 if and only if the two decoding conditions D1 and D2 hold at u w.r.t. v (see Definition 1).

LEMMA 4. *Let $L(u)$ and $L(v)$ be two labels assigned by \mathcal{M} to two nodes in T . The decoder algorithm $\mathcal{D}(L(u), L(v))$ performs in constant time, and satisfies $\mathcal{D}(L(u), L(v)) = 1$ if and only if u is an ancestor of v in T .*

PROOF. The fact that $\mathcal{D}(L(u), L(v)) = 1$ if and only if u is an ancestor of v in T follows by combining Lemma 1 with the fact that the intervals are assigned by the marker algorithm via a legal-interval mapping (cf. Corollary 1). Since $I(u) = I_{k,a,b} = [2^k a, 2^k(a + b)]$ with all three parameters k, a , and b stored explicitly, computing $I(u)$ from $L(u)$ can be achieved in constant time. (Note that $2^k a$, for example, can be obtained from a by a simple shift of k bits.) Similarly, $I(v)$ can be extracted from $L(v)$ in constant time. Computing $I(\text{sup}(u))$ just needs a simple subtraction and a division by a power of 2, which again amounts to a simple shift operation. The lemma follows. \square

Theorem 1 follows by combining Lemma 3 with Lemma 4. Observe now that a rooted forest $F \in \mathcal{F}(n)$ can be viewed as

a subgraph of T , where T is the rooted tree with $n + 1$ nodes obtained by adding another root node and connecting it to the roots of all trees in F . Thus, by slightly modifying our marker algorithm \mathcal{M} , we obtain a new ancestry scheme $(\mathcal{M}', \mathcal{D})$ that can handle forests as well. Specifically, we have:

COROLLARY 2. *$(\mathcal{M}', \mathcal{D})$ is an ancestry scheme for $\mathcal{F}(n)$ with label size $\log n + 4 \log \log n + O(1)$ bits.*

4. SMALL UNIVERSAL POSETS

For any two positive integers n and k , let $\mathcal{P}(n, k)$ denote the family of all n -element posets with tree-dimension at most k . Recall that an ancestry labeling scheme $(\mathcal{M}, \mathcal{D})$ for $\mathcal{F}(n)$ is consistent if \mathcal{M} assigns distinct labels to the nodes of any $F \in \mathcal{F}(n)$ and \mathcal{D} satisfies the anti-symmetry and transitivity conditions. Our next theorem relates compact consistent ancestry labeling schemes with small universal posets.

THEOREM 2. *There exists a consistent ancestry labeling scheme for $\mathcal{F}(n)$ with label size ℓ if and only if for every integer k , there exists a universal poset of size $2^{k\ell}$ for $\mathcal{P}(n, k)$.*

PROOF. Assume first that for every integer k there exists a universal poset (\mathcal{P}, \leq) of size $2^{k\ell}$ for $\mathcal{P}(n, k)$. In particular, there exists a universal poset (\mathcal{P}, \leq) of size 2^ℓ for $\mathcal{P}(n, 1)$, i.e., for the family of n -element posets with tree-dimension 1. We construct the following ancestry scheme $(\mathcal{M}, \mathcal{D})$ for $\mathcal{F}(n)$. The marker algorithm \mathcal{M} first considers some bijective mapping $\phi : \mathcal{P} \rightarrow [1, |\mathcal{P}|]$. Given a rooted forest $F \in \mathcal{F}(n)$, view F as a poset whose Hasse diagram is F . Obviously, F is a poset with tree-dimension 1. Now, let $\rho : F \rightarrow \mathcal{P}$ be a mapping that preserves the partial order of F . The marker algorithm assigns the label $L(u) = \phi(\rho(u))$ to each node $u \in F$. Given the labels $L(u)$ and $L(v)$ of two nodes u and v in some $F \in \mathcal{F}(n)$, the decoder algorithm \mathcal{D} outputs 1 if and only if $L(u) \neq L(v)$ and $\phi^{-1}(L(u)) \leq \phi^{-1}(L(v))$. Obviously, $(\mathcal{M}, \mathcal{D})$ is a consistent ancestry labeling scheme for $\mathcal{F}(n)$ with label size $\log |\mathcal{P}| = \ell$.

For the other direction, assume that there exists a consistent ancestry labeling scheme for $\mathcal{F}(n)$ with label size ℓ . Let $\mathcal{P} = [1, 2^{\ell k}]$. We define a relation \leq between pairs of elements in \mathcal{P} . For two elements $x, y \in \mathcal{P}$, where $x = (x_1, x_2, \dots, x_k)$ and $y = (y_1, y_2, \dots, y_k)$, we say that $x \leq y$ iff for every i , $1 \leq i \leq k$, either $x_i = y_i$ or $\mathcal{D}(x_i, y_i)$ is defined and equals 1. It can be easily verified that (\mathcal{P}, \leq) is a universal poset for $\mathcal{P}(n, k)$. The theorem follows. \square

For the objective of constructing a small universal poset for $\mathcal{P}(n, k)$, let us consider the ancestry labeling scheme $(\mathcal{M}', \mathcal{D})$ for $\mathcal{F}(n)$ given by Corollary 2. We now define a new decoder algorithm \mathcal{D}' as follows: $\mathcal{D}'(L(u), L(v)) = 1$ if and only if $\mathcal{D}(L(u), L(v)) = 1$ and $I(\text{sup}(u)) \subseteq I(\text{sup}(v))$.

LEMMA 5. *$(\mathcal{M}', \mathcal{D}')$ is a consistent ancestry labeling scheme for $\mathcal{F}(n)$ with label size $\log n + 4 \log \log n + O(1)$ bits.*

PROOF. The fact that $(\mathcal{M}', \mathcal{D}')$ is an ancestry labeling scheme for $\mathcal{F}(n)$ is obvious, since the condition $I(\text{sup}(v)) \subseteq I(\text{sup}(u))$ always holds if u is an ancestor of v in some forest (see Claim 1 in the proof of Lemma 1). Moreover, the required label size of $(\mathcal{M}', \mathcal{D}')$ follows directly from Corollary 2.

We now turn to prove that $(\mathcal{M}', \mathcal{D}')$ is consistent. The fact that \mathcal{M}' assigns disjoint labels to the nodes of any forest $F \in \mathcal{F}(n)$ is obvious from its construction. Consider now any three labels ℓ_1, ℓ_2 and ℓ_3 in the domain of \mathcal{M}' , and let u, v and w be three nodes each belonging to some forest in $\mathcal{F}(n)$ (not necessarily the same forest) where $\ell_1 = L(u)$, $\ell_2 = L(v)$ and $\ell_3 = L(w)$. Assume now that $\mathcal{D}'(L(u), L(v)) = 1$ and let us show that $\mathcal{D}'(L(v), L(u)) = 0$. By definition of \mathcal{D}' , the decoding conditions between u and v must hold, i.e., we have (1) $I(v) \subset I(\text{sup}(u))$, and (2) $I(u) \prec I(v)$ or $I(u) = I(\text{sup}(u))$. (Note, by definition, these conditions must hold also if u and v belong to different forests). If $I(u) = I(\text{sup}(u))$ then $I(v) \subset I(u)$. If also $\mathcal{D}'(L(v), L(u)) = 1$ then we must have $I(v) = I(\text{sup}(v))$ which contradicts the condition $I(u) \subset I(\text{sup}(v))$. If on the other hand $I(u) \neq I(\text{sup}(u))$ then $I(u) \prec I(v)$. If also $\mathcal{D}'(L(v), L(u)) = 1$ then we must have $I(v) = I(\text{sup}(v))$, and therefore since $I(u) \subset I(\text{sup}(v))$, we get $I(u) \subset I(v)$, contradiction.

Let us now show that the transitivity condition holds, i.e., we assume that $\mathcal{D}'(L(u), L(v)) = 1$ and $\mathcal{D}'(L(v), L(w)) = 1$, and our goal is to show that $\mathcal{D}'(L(u), L(w)) = 1$. The fact that $I(\text{sup}(w)) \subseteq I(\text{sup}(u))$ is immediate. Since $I(w) \subset I(\text{sup}(v))$ and $I(\text{sup}(v)) \subseteq I(\text{sup}(u))$, we obtain that $I(w) \subset I(\text{sup}(u))$, i.e., the first decoding condition between u and w holds. We now show that the second decoding condition also holds. If $I(u) = I(\text{sup}(u))$ then we are done, therefore, assume $I(u) \neq I(\text{sup}(u))$. The fact that $\mathcal{D}'(L(u), L(v)) = 1$ now implies that $I(u) \prec I(v)$. Observe, if $I(v) \prec I(w)$ then $I(u) \prec I(w)$ and we are done. Otherwise, the fact that $\mathcal{D}'(L(u), L(w)) = 1$ implies that $I(v) = I(\text{sup}(v))$. Therefore, $I(w) \subset I(\text{sup}(v)) = I(v)$. Combining this with the fact that $I(u) \prec I(v)$, we obtain $I(u) \prec I(w)$. This completes the proof of the lemma. \square

By combining Theorem 2 and Lemma 5, we obtain the following.

THEOREM 3. *For every integer k , there exists a universal poset of size $O(n^k \log^{4k} n)$ for $\mathcal{P}(n, k)$.*

Acknowledgments.

The authors are very thankful to William T. Trotter, Sundar Vishwanathan and Jean-Sebastien Sereni for helpful discussions.

5. REFERENCES

- [1] S. Abiteboul, S. Alstrup, H. Kaplan, T. Milo and T. Rauhe. Compact labeling schemes for ancestor queries. *SIAM Journal on Computing* **35**, (2006), 1295–1309.
- [2] S. Abiteboul, P. Buneman and D. Suciu. Data on the Web: From Relations to Semistructured Data and XML. *Morgan Kaufmann*, (1999).
- [3] Abiteboul, S., Kaplan, H., and Milo, T.: Compact labeling schemes for ancestor queries. In *Proc. 12th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, 2001.
- [4] S. Alstrup, P. Bille and T. Rauhe. Labeling Schemes for Small Distances in Trees. In *Proc. 14th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, 2003.
- [5] S. Alstrup, C. Gavoille, H. Kaplan and T. Rauhe. Nearest Common Ancestors: A Survey and a new Distributed Algorithm. *Theory of Computing Systems* **37**, (2004), 441–456.
- [6] S. Alstrup and T. Rauhe. Small induced-universal graphs and compact implicit graph representations. In *Proc. 43rd IEEE Symp. on Foundations of Computer Science (FOCS)*, 2002.
- [7] S. Alstrup and T. Rauhe. Improved labeling scheme for ancestor queries. In *Proc. 13th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 947–953, 2002.
- [8] N. Alon and E. R. Scheinerman. Degrees of Freedom Versus Dimension for Containment Orders. In *Order* **5** (1988), 11–16.
- [9] G. Behrendt. On trees and tree dimension of ordered sets. *Order* **10(2)**, (1993), 153–160.
- [10] A. Deutsch, M. Fernández, D. Florescu, A. Levy and D. Suciu. A Query Language for XML. *Computer Networks* **31**, (1999), 1155–1169.
- [11] R. Fraisse Theory of relations. *North-Holland*, Amsterdam, 1953.
- [12] P. Fraigniaud and C. Gavoille. Routing in trees. In *Proc. 28th Int. Colloq. on Automata, Languages, and Programing (ICALP)*, LNCS 2076, pages 757–772, Springer, 2001.
- [13] P. Fraigniaud and A. Korman. Compact Ancestry Labeling Schemes for XML Trees. In *Proc. 21st ACM-SIAM Symp. on Discrete Algorithms (SODA)*, 2010.
- [14] C. Gavoille, D. Peleg, S. Pérennes and R. Raz. Distance labeling in graphs. In *Proc. 12th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 210–219, 2001.
- [15] J. Hubicka and J. Nežetřil. Universal partial order represented by means of oriented trees and other simple graphs. *Euro. J. of Combinatorics* **26(5)**, (2005), 765 – 778.
- [16] B. Jonson. Universal relational systems. *Math Scan.* **4**, (1956), 193–208.
- [17] J. B. Johnston. Universal infinite partially ordered sets. *Proc. Amer. Math. Soc.* **7**, (1957), 507–514.
- [18] M. Katz, N.A. Katz, A. Korman, and D. Peleg. Labeling schemes for flow and connectivity. *SIAM Journal on Computing* **34** (2004), 23–40.
- [19] A. Korman. Labeling Schemes for Vertex Connectivity. *ACM Transactions on Algorithms*, to appear.
- [20] A. Korman and S. Kutten. Distributed Verification of Minimum Spanning Trees. *Distributed Computing* **20(4)**: 253–266 (2007).
- [21] H. Kaplan and T. Milo. Short and simple labels for small distances and other functions. In *Workshop on Algorithms and Data Structures*, Aug. 2001.
- [22] H. Kaplan, T. Milo and R. Shabo. A Comparison of Labeling Schemes for Ancestor Queries. In *Proc. 19th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, 2002.
- [23] S. Kannan, M. Naor, and S. Rudich. Implicit representation of graphs. *SIAM J. on Discrete Math* **5**, (1992), 596–603.
- [24] D. Peleg. Informative labeling schemes for graphs.

Theoretical Computer Science **340(3)**, (2005),
577–593.

- [25] M. Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *J. of the ACM* **51**, (2004), 993–1024.
- [26] M. Thorup and U. Zwick. Compact routing schemes. In Proc. 13th ACM Symp. on Parallel Algorithms and Architecture (SPAA), pages 1–10, 2001.
- [27] W. T. Trotter. Combinatorics and partially ordered sets: Dimension theory. *The Johns Hopkins University Press*, (1992).
- [28] W.T. Trotter and J. I. Moore. The dimension of planar posets *J. Comb. Theory B* **22**, (1977), 54–67.
- [29] E.S. Wolk. The comparability graph of a tree. *Proc. Amer. Math. Soc.* **3**, (1962), 789–795.
- [30] W3C. Extensive markup language (XML) 1.0. <http://www.w3.org/TR/REC-xml>.
- [31] W3C. Extensive stylesheet language (xsl) 1.0. <http://www.w3.org/Style/XSL/>.
- [32] W3C. Xsl transformations (xslt) specification. <http://www.w3.org/TR/WD-xslt>