Improved ancestry labeling scheme for trees

Søren Dahlgaard*, Mathias Bæk Tejs Knudsen*** and Noy Rotbart

Department of Computer Science, University of Copenhagen
Universitetsparken 5, 2100 Copenhagen
{soerend,knudsen,noyro}@di.ku.dk

Abstract. We consider ancestry labeling schemes: Given a rooted tree $T$, assign a binary string (label) to each node, such that given the labels of any two nodes, one can determine whether the first is an ancestor of the second in $T$.

Recently, Fräniiaud and Korman [STOC’10] showed that such labels can be assigned using $\log n + O(\log \log n)$ bits per label, solving a long standing open question and matching a lower bound of $\log n + \Omega(\log \log n)$ due to Alstrup et al. [SODA’03]. In this paper we present an alternative ancestry labeling scheme using $\log n + 2 \log \log n + O(1)$ bits. Similar to the previously known schemes, our scheme relies on intervals and can encode any tree in $O(n)$ time. Rather than using complicated decompositions, our scheme uses approximate interval sizes, which gives a clean algorithm that is simple enough to be taught and implemented.

1 Introduction

A labeling scheme is a method of distributing the information about the structure of a graph among its vertices by assigning short labels, such that a selected function on pairs of vertices can be computed using only their labels. The concept was introduced in a restricted manner by Breuer and Folkman [1], revisited by Kannan, Naor and Rudich [2], and explored by a wealth of subsequent work [3–8].

An ancestry labeling scheme labels the nodes of a tree $T$, such that for any pair of nodes $u$ and $v$ in $T$, one can determine whether $u$ is an ancestor of $v$ solely from the labels of $u$ and $v$. The following classic ancestry scheme was introduced in [2]: Given a rooted tree $T$ of $n$ nodes, perform a DFS traversal of $T$ and assign to each node $u$ its DFS number $\text{DFS}(u)$. The label of a node $u$ is then given as the pair $(\text{DFS}(u), S(u) - 1)$, where $S(u)$ denotes the size of the subtree rooted at $u$. The label corresponds to an

* Research partly supported by Mikkel Thorup’s Advanced Grant from the Danish Council for Independent Research under the Sapere Aude research carrier programme.

** Research partly supported by the FNU project AlgoDisc - Discrete Mathematics, Algorithms, and Data Structures.
interval $I(u) = [\text{DFS}(u), \text{DFS}(u) + S(u) - 1]$, and ancestry queries can be answered by looking at interval containment: $u$ is an ancestor of $v$ if and only if $I(v) \subset I(u)$. Clearly the maximal label size assigned by this scheme on any $n$-node tree is at most $2 \lceil \log n \rceil$. Recently it was shown by Fraigniaud and Korman \cite{7} how to construct ancestry labels of size $\log n + O(\log \log n)$ solving the ancestry problem, which was introduced more than twenty years ago by matching a lower bound of $\log n + \Omega(\log \log n)$ due to Alstrup et al. \cite{6}. Their labeling scheme also reduces to testing interval containment $I(v) \subset I(u)$ to answer queries and assigns labels to any $n$-node tree in $O(n)$ time.

Ancestry labeling schemes for trees have been studied extensively in the literature \cite{2, 6, 7, 9–14}. The main motivation is their practical applications in improving the performance of XML search engines. XML documents are often viewed as trees, and queries over such documents amount to testing ancestry relationships between the nodes. Search engines process queries using an index structure summarizing the ancestor relations. In the index, each document (tree node) is represented using some label, such that given two labels, the engine can determine if one is ancestor of the other. The label size plays a major role in the index size and reducing the label size is thus a critical issue. A more detailed explanation can be found in \cite{15}.

1.1 Our result

In this paper we present a labeling scheme for answering ancestry queries with maximum label size $\log n + 2 \log \log n + O(1)$. Similar to the labeling schemes of \cite{2, 7}, our scheme relies on creating intervals. Our labeling scheme is almost identical to the classic scheme described above, but relies on storing a $(1 + \varepsilon)$-approximation of $S(u)$ in order to save space. To make this work we also need to “reserve” certain DFS numbers. Contrary to previous labeling schemes, the interval induced by the label of a node is not guaranteed to be fully contained in the intervals of its ancestors. However, the DFS number of the node is. The main technical part of the paper is to show that the number of “reserved” DFS numbers does not become too big, and thus we can keep the label size small. Our algorithm is simple enough that it can be explained to undergraduate students and takes only six lines of pseudocode to present. Additionally it uses slightly less space than the scheme of \cite{7} ($\log n + 2 \log \log n + O(1)$ compared to $\log n + 4 \log \log n + O(1)$).

\footnote{Throughout this paper we let $\log$ denote the base-2 logarithm.}
2 Preliminaries

The size of a binary string \( x = x_1 \cdots x_\ell \) is the number of bits \( \ell \) and is denoted by \( |x| \).

Let \( T \) be a rooted tree with root \( r \). For any pair of nodes \( u, v \in T \) we say that \( u \) is an ancestor of \( v \) if it lies on the unique path from \( r \) to \( v \). A node \( v \) is a descendant of a node \( u \) if and only if \( u \) is an ancestor of \( v \). We denote by \( S(u) \) the size of the subtree rooted at \( u \), i.e. the number of descendants of \( u \), where a node is a descendant of itself.

Let \( F \) be a family of rooted trees. An ancestry labeling scheme for \( F \) consists of an encoder and decoder. The encoder is an algorithm that receives any tree \( T \in F \) and produces a binary string called a label \( L(v) \) for each node \( v \in T \). The decoder is an algorithm that receives any two labels \( L(u), L(v) \) and answers TRUE if and only if \( u \) is an ancestor of \( v \) in \( T \). We denote by the label size of a labeling scheme the maximum size of any label assigned by the encoder taken over all nodes in all trees in \( F \).

3 The labeling scheme

In this section we will prove the following theorem:

**Theorem 1.** There exists an ancestry labeling scheme for trees of size \( n \) with label size \( \lceil \log n \rceil + 2 \lceil \log \log n \rceil + 3 \). Moreover, given a tree \( T \), the labels can be assigned in \( O(n) \) time, and any query can be answered in constant time.

Recall the classic ancestry labeling scheme described in the introduction: Each node gets a label \( (x, y) \) and given two nodes \( u, v \) with labels \( (x, y) \) and \( (z, w) \) respectively, \( u \) is an ancestor of \( v \) iff \( x \leq z \leq x + y \). Say that \( (x, y) \) is a label corresponding to a tree with \( n \) nodes. The encoding of \( (x, y) \) is done with exactly \( 2 \lceil \log n \rceil \) bits: The concatenation of the binary encoding of \( x \) and \( y \) (each padded with 0's such that they are exactly \( \lceil \log n \rceil \) bits each). The decoder will then extract \( x \) and \( y \) as the left and the right half of the binary string respectively.

Our labeling scheme is similar and each node still gets a label \( (x, y) \), and the decoder will do the same test: Say \( u, v \) have labels \( (x, y) \) and \( (z, w) \) respectively, then \( u \) is an ancestor of \( v \) iff \( x \leq z \leq x + y \). However, note that the test \( x \leq z \leq z + w \leq x + y \) (corresponding to interval containment) will not necessarily give the same answer even though it would in the setting of the classic ancestry labeling scheme.
The difference between our labeling scheme and the classic labeling scheme is that the labels \((x, y)\) are restricted such that they are cheaper to encode. Any label given to a node in a tree of size \(n\) satisfies:

1. \(x \in [2n] = \{0, 1, \ldots, 2n - 1\}\)
2. \(y = \lceil (1 + \varepsilon)^k \rceil - 1\) for some integer \(k \in \lfloor 4([\log n])^2 \rfloor\) where \(\varepsilon\) is chosen such that \(\log(1 + \varepsilon) = (\lfloor \log n \rfloor)^{-1}\).

Such a label can be encoded using \([\log n] + 2 [\log \log n] + 3\) bits in the following way: First \(x\) is encoded using \([\log n] + 1\) bits (padding with 0’s if necessary). Then \(k\) is encoded using \(2 + 2 [\log [\log n]]\) bits (padding with 0’s if necessary). The label is then the concatenation of these two bit strings. From the size of the label the decoder will be able to deduce \([\log n]\), and given \([\log n]\) it is easy to extract \(x\) and \(k\), and thus reconstruct \((x, y)\).

1: function classic(node \(v\), int \(Id\))
2: \(a \leftarrow \text{Id}\)
3: for all \(w \in \text{children}(v)\) do
4: \(a \leftarrow \text{classic}(w, a + 1)\)
5: \(v.\text{label} \leftarrow (\text{Id}, (a - \text{Id}))\)
6: return \(a\)

1: function improved(node \(v\), int \(Id\))
2: \((a, b) \leftarrow (\text{Id}, \text{Id})\)
3: for all \(w \in \text{sorted-children}(v)\) do
4: \((a, b) \leftarrow \text{improved}(w, b + 1)\)
5: \(v.\text{label} \leftarrow (\text{Id}, \text{Approx}(a - \text{Id}))\)
6: return \((a, \max\{b, \text{Approx}(a - \text{Id}) + \text{Id}\})\)

Fig. 1. Our algorithm (right) and the classic algorithm of [2] (left). Here \(\text{sorted-children}(v)\) denotes the children of \(v\) ordered increasingly by the size of their corresponding subtrees and \(\text{Approx}(x)\) denotes a \((1 + \varepsilon)\)-approximation of \(x\).

Fig. 2. Label assignment of our algorithm (right) using \(\varepsilon = 1\) and the classic algorithm (left).
The assignment of the labels is given as depicted in the pseudo-code of Fig. 1. An example labeling can be seen in Fig. 2. Note that this example uses \( \varepsilon = 1 \) for instructive purposes. The function \( \text{Approx}(x) \) returns the smallest possible value of \( \left\lceil (1 + \varepsilon)^k \right\rceil - 1 \) that is larger than \( x \), where \( \varepsilon \) is chosen such that \( \log(1 + \varepsilon) = (\lceil \log n \rceil)^{-1} \). Note that \( k \) can be computed efficiently since \( k = \lceil \log(x + 1) \cdot \lceil \log n \rceil \rceil \).

The algorithm satisfies the following invariants:

**Lemma 1.** Let \( (a, b) \) be the values returned by \( \text{IMPROVED}(v, \text{Id}) \) and consider the labels \( (x_0, y_0), \ldots, (x_{r-1}, y_{r-1}) \) assigned to all the nodes in \( v \)'s subtree (including \( v \)'s label). Then:

\[
a = \max_i \{ x_i \}, \quad b = \max_i \{ x_i + y_i \}
\]

*Proof.* The proof is a routine induction argument. \( \square \)

The invariant of Lem. 1 implies that the decoder correctly answers TRUE given the labels \( L(u) \) and \( L(v) \) if \( u \) is an ancestor of \( v \). Furthermore, lines 4 and 6 of Fig. 1 implies that no id in the interval induced by \( L(u) \) is assigned outside the subtree rooted in \( u \) implying that the decoder answers TRUE if and only if \( u \) is an ancestor of \( v \).

Now we only need to prove that the labels can be encoded efficiently, i.e. we need to show that each label \( (x, \left\lfloor (1 + \varepsilon)^k \right\rfloor - 1) \) satisfies \( x < 2n \) and \( k < 4 \lceil \log n \rceil^2 \). In order to justify this claim we will need Lem. 2.

**Lemma 2.** Let \( v \) be a node in a tree \( T \) and let \( S = S(v) \), \( x \) an integer and \( l = \lceil \log S \rceil \). If \( (a, b) = \text{IMPROVED}(v, x) \) then:

1. \( a \leq x + S \cdot (1 + \varepsilon)^l - 1 \),
2. \( b \leq x + S \cdot (1 + \varepsilon)^{l+1} - 1 \).

*Proof.* Let the children of \( v \) be denoted by \( w_1, \ldots, w_k \) with \( S(w_1) \leq \ldots \leq S(w_k) \). Denote by \( a_i, b_i \) the \( a \) and \( b \) values returned by \( \text{IMPROVED}(w_i, b_{i-1} + 1) \) with \( a_0 = b_0 = x \).

We will prove the claims by induction on \( S \). When \( S = 1 \) we have \( a = b = x \), so the claim holds trivially.

Since \( w_k \) is the heaviest node we have \( S(w_1), \ldots, S(w_{k-1}) \leq S/2 \), so \( \lceil \log S(w_i) \rceil \leq l - 1 \) for \( i < k \). Thus it holds by our second induction hypothesis that for each \( 1 \leq i < k \):

\[
b_i \leq (b_{i-1} + 1) + S(w_i) \cdot (1 + \varepsilon)^l - 1 = b_{i-1} + S(w_i) \cdot (1 + \varepsilon)^l
\]
In particular:

\[ b_{k-1} \leq b_0 + (S(w_1) + \ldots + S(w_{k-1})) \cdot (1+\varepsilon)^l = x + (S - S(w_k) - 1) \cdot (1+\varepsilon)^l. \]

Since \( b_k \leq b_{k-1} + S(w_k) \cdot (1+\varepsilon)^{l+1} \), we get:

\[ b_k \leq x + (S - S(w_k) - 1) \cdot (1+\varepsilon)^l + S(w_k) \cdot (1+\varepsilon)^{l+1} \leq x + S \cdot (1+\varepsilon)^{l+1} - 1 \]

In the same manner we obtain:

\[ a_k \leq x + S \cdot (1+\varepsilon)^l - 1 \]

Since \( a = a_k \) the first part is proved. The \( b \) that is returned is the maximum of \( b_k \) and \( x + \text{Approx}(a_k - x) \). Since we already proved that \( b_k \) is at most \( x + S \cdot (1+\varepsilon)^{l+1} - 1 \), we just need to prove that:

\[ \text{Approx}(a_k - x) \leq S \cdot (1+\varepsilon)^{l+1} - 1 \]

Recall that \( \text{Approx}(a_k - x) = \lceil (1+\varepsilon)^i \rceil - 1 \) for the smallest possible value of \( i \) such that \( \text{Approx}(a_k - x) \geq a_k - x \). In particular:

\[ \lceil (1+\varepsilon)^{i-1} \rceil - 1 < a_k - x \]

This trivially implies that

\[ (1+\varepsilon)^{i-1} < a_k - x + 1 \leq S \cdot (1+\varepsilon)^l \]

Therefore,

\[ \text{Approx}(a_k - x) \leq (1+\varepsilon)^{i-1} - 1 < (1+\varepsilon)(a_k - x + 1) - 1 \leq S \cdot (1+\varepsilon)^{l+1} - 1 \]

which finishes the proof.

Let \((x, y)\) be a label assigned to a node in a tree with size \(n\). By Lem.\(^1\) and \(^2\) we see that:

\[ x \leq n(1+\varepsilon)^{|\log n|} - 1, \quad y \leq x + y \leq n \cdot (1+\varepsilon)^{|\log n|+1} - 1. \]

By the definition of \( \varepsilon \) we have \( (1+\varepsilon)^{|\log n|} = (2^{1/|\log n|})^{|\log n|} = 2 \) and hence \( x \leq 2n - 1 \). For \( y \), we see that

\[ n = \left( (1+\varepsilon)^{|\log n|} \right)^{\log n} \leq (1+\varepsilon)^{(\log n)^2}, \]

and thus

\[ y \leq (1+\varepsilon)^{(\log n)^2+\log n+1} - 1. \]

Letting \( y = \lceil (1+\varepsilon)^k \rceil - 1 \), we get \( k \leq (\log n)^2 + \log n + 1 \).

We have now shown that \( x < 2n \) and \( k < 4 |\log n|^2 \). It follows from the previous discussion that we can always encode the labels using no more than \( |\log n| + 2 |\log \log n| + 3 \) bits proving Thm.\(^1\).
Bibliography

[1] M. A. Breuer, J. Folkman, An unexpected result in coding the vertices of a graph, J. Mathematical Analysis and Applications 20 (1967) 583–600.
[2] S. Kannan, M. Naor, S. Rudich, Implicit representation of graphs, in: SIAM Journal On Discrete Mathematics, 1992, pp. 334–343.
[3] D. Peleg, Proximity-preserving labeling schemes, Journal of Graph Theory 33 (3) (2000) 167–176.
[4] M. Thorup, U. Zwick, Compact routing schemes, in: SPAA ’01, 2001, pp. 1–10.
[5] S. Alstrup, T. Rauhe, Small induced-universal graphs and compact implicit graph representations, in: FOCS ’02, 2002, pp. 53–62.
[6] S. Alstrup, P. Bille, T. Rauhe, Labeling schemes for small distances in trees, SIAM J. Discret. Math. 19 (2) (2005) 448–462.
[7] P. Fraigniaud, A. Korman, An optimal ancestry scheme and small universal posets, in: STOC ’10, STOC ’10, 2010, pp. 611–620.
[8] S. Alstrup, E. B. Halvorsen, K. G. Larsen, Near-optimal labeling schemes for nearest common ancestors, in: SODA, 2014, pp. 972–982.
[9] E. Cohen, H. Kaplan, T. Milo, Labeling dynamic xml trees, in: PODS ’02, ACM, New York, NY, USA, 2002, pp. 271–281.
[10] S. Alstrup, P. Bille, T. Rauhe, Labeling schemes for small distances in trees, in: SODA ’03, SODA ’03, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2003, pp. 689–698.
[11] X. Wu, M. L. Lee, W. Hsu, A prime number labeling scheme for dynamic ordered xml trees, in: Data Engineering, 2004. Proceedings. 20th International Conference on, IEEE, 2004, pp. 66–78.
[12] T. Härdler, M. Haustein, C. Mathis, M. Wagner, Node labeling schemes for dynamic xml documents reconsidered, Data & Knowledge Engineering 60 (1) (2007) 126–149.
[13] E. Cohen, H. Kaplan, T. Milo, Labeling dynamic xml trees, SIAM Journal on Computing 39 (5) (2010) 2048–2074.
[14] P. Fraigniaud, A. Korman, Compact ancestry labeling schemes for xml trees, in: In Proc. 21st ACM-SIAM Symp. on Discrete Algorithms (SODA, 2010.
[15] S. Abiteboul, S. Alstrup, H. Kaplan, T. Milo, T. Rauhe, Compact labeling scheme for ancestor queries, SIAM J. Comput. 35 (6) (2006) 1295–1309.