Distributed Relationship Schemes for Trees

Cyril Gavoille* and Arnaud Labourel

LaBRI, Université de Bordeaux, France
{gavoille,labourel}@labri.fr

Abstract. We consider a distributed representation scheme for trees, supporting some special relationships between nodes at small distance. For instance, we show that for a tree T and an integer k we can assign local information on nodes such that we can decide for any two nodes u and v if the distance between u and v is at most k and if so, compute it only using the local information assigned. For trees with n nodes, the local information assigned by our scheme are binary labels of $\log n + O(k \log(k \log(n/k)))$ bits, improving the results of Alstrup, Bille, and Rauhe (SODA '03).

Keywords: distributed data-structures, ancestry, tree, distance.

1 Introduction

A *distributed representation scheme* is a scheme maintaining global information on a network using local data structures (or labels) assigned to nodes of the network. Such schemes play an important role in the fields of distributed computing. Their goal is to locally store some useful information about the network and make it conveniently accessible. For instance, *implicit representation* of networks is a distributed representation scheme that supports adjacency queries, i.e., adjacency between two nodes can be determined only by examining the local information stored by the two nodes. So, the network can be manipulated by keeping only its labels in memory, any other global information on the graph (like its matrix) can be removed. The goal is to minimize the maximum length of a label associated with a vertex while keeping fast queries.

Distributed representation is widely used in distributed computing, e.g. in [14,18]. Kannan, Naor and Rudich [17] investigated in particular implicit representation for several families of graphs, including trees with labels of $2 \log n$ bits[1] where n is the number of nodes of the tree. Actually, parent and ancestry queries for rooted trees can be done with $1.5 \log n + O(\log \log n)$ bit labels [2], and this has been improved to $\log n + O(\sqrt{\log n})$ [1]. If we insist only on implicit representation of trees, the label length can be reduced. Chung [9] improved in a non-trivial way to $\log n + \log \log n + O(1)$ bits, and further improved to $\log n + O(\log^* n)$ bits[2] by Alstrup et al. [6].

* Both authors are supported by the project CÉPAGE of INRIA Futur, and the ANR-projects GEOCOMP and GRAAL.

[1] The log function denotes the logarithm in base two.

[2] $\log^* n$ denotes the number of times log should be iterated to get a constant.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 728–738, 2007.

1.1 Related Works

Motivated by applications in XML search engines, and distributed applications as peer-to-peer networks or network routing, several queries on distributed data-structures have been investigated recently. In this framework, distributed data-structure can be seen as a label assigned to each node such that queries can be answered by inspecting the labels only, without any other source of information. For instance, address-based routing in trees [11,19,21] or in a specific class of networks [8,10], distance queries for cycles [20], for interval and permutation graphs [7,13], or for hyperbolic graphs [3,12], ancestry in rooted trees [1], etc. have optimal $O(\log n)$-bit distributed data-structures.

In this paper, we consider labeling scheme for various relationships between nodes of small distance in trees. For instance, we construct a simple distributed representation scheme supporting parent and sibling queries with labels of size $\log n + 2\log\log n + O(1)$, the best bound was $\log n + 5\log\log n + O(1)$ [5]. The current lower bound on the label length for schemes supporting sibling queries in trees is $\log n + \log\log n + O(1)$ [4].

We stress that improving the second order term on the label length complexity is not only of theoretical interest. It does matter in practice because, as mentioned in [2], engines for indexing XML files use such schemes for huge database in which each item is associated with a label. Therefore decreasing by one byte the length of the labels results a save of gigabytes of main memory for large database. Moreover and interestingly, every distributed representation can be interpreted as a universal matrix [20]: row and column indexes of the matrix correspond to all the labels of the scheme, and the value of entry (i, j) in the matrix is the answer of the query applied to two nodes labeled respectively i and j by the scheme. In the case of implicit representation for instance, the universal matrix is Boolean and corresponds to the adjacency matrix of a graph, called *induced-universal graph*, having the property of containing all graphs of the family as induced subgraph. So, proving a complexity of $f(n)$ for label length transfers to the existence of universal matrix of dimension $O(2^{f(n)})$. So improving the second order term in the label length actually improves the complexity of the matrix dimension. For concreteness, the difference between the $2\log n$-bit labeling scheme of [17] and the $\log n + \log\log n + O(1)$ scheme of [9] corresponds to an improvement from $O(n^2)$ to $O(n\log n)$ on the size of the smallest induced-universal graphs containing all n-node trees as induced subgraphs.

1.2 Our Contribution

In a rooted tree, two nodes u and v with nearest common ancestor z are (k_1, k_2)-*related* if the distance from u to z is k_1 and the distance from v to z is k_2. For any integer k, a *k-relationship scheme* is a distributed representation scheme that supports tests for whether u and v are (k_1, k_2)-related for all nodes u and v, and all integers $k_1, k_2 \leqslant k$.

For instance, a 1-relationship scheme supports tests for whether two nodes are $(0, 0)$, $(1, 0)$, $(0, 1)$ or $(1, 1)$-related, and so supporting parent and sibling queries.

Let us observe that a k-relationship scheme supports also distance queries for nodes at distance at most k.

In this paper we propose a k-relationship scheme for trees of n nodes with labels of size $\log n + O(k \log(k \log(n/k)))$ bits. This improves the scheme presented in [4,5] that uses labels of $\log n + O(k^2 \log(k \log n))$ bits. Our scheme is simple and easy to implement, and we show[3] that the time to preprocess the tree and for constructing the n labels is $O(kn + n \log n)$. Our scheme also implies that distance between nodes at distance $o(\log n / \log\log n)$ can be determined with labels of $\log n + o(\log n)$ bits. In contrast, it has been proved in [15] that labels of $\Omega(\log^2 n)$ bits are required for arbitrary distance queries in trees.

A k-relationship scheme has query time complexity t if one can check whether a pair of nodes is (k_1, k_2)-related or not, in time at most t, for any pair of nodes and all integers $k_1, k_2 \leqslant k$.

Our result is summarized in the following statement:

Theorem 1. *The family of n-node rooted trees enjoys a k-relationship scheme with labels of $\log n + O(k \log(k \log(n/k)))$ bits with constant query time.*

For $k = 1$, we show that labels are of length $\log n + 2 \log\log n + O(1)$, improving the $\log n + 5 \log\log n + O(1)$ scheme of [5]. Moreover, for $k = o(\log n / \log\log n)$, we derive directly from Theorem 1 that:

Corollary 1. *The family of n-node trees enjoys a distributed representation supporting distance with labels of $\log n + o(\log n)$ bits for nodes at distance $o(\log n / \log\log n)$.*

In Section 2, we present the k-relationship scheme, and we conclude in Section 3 with some open problems.

2 A Relationship Scheme

2.1 Preliminaries

The basic idea of our scheme is to store into the label of each node u, some identifiers for u and for its k closest ancestors. Indeed, to test if u and v are (k_1, k_2)-related, it suffices to test if the ancestor at distance k_1 of u is equal to the ancestor at distance k_2 of v, and the ancestor at distance $k_1 - 1$ of u differs from the ancestor at distance $k_2 - 1$ of v. A naive implementation would lead to $O(k \log n)$-bit labels for arbitrary identifiers. We can significantly decrease this complexity with a better choice of the identifiers exploiting correlations between nodes.

Let T be a rooted tree of n nodes. We define the *k-ancestry* of a node u as the set of ancestors of u at distance at most k, u included. A *branch* of T is a path leading from the root to a leaf of T. We denote by $G[X]$ the subgraph induced by the set of nodes X.

The main notion we introduce below is illustrated on Fig. 2.1.

[3] Due to space limitation the time complexity of the scheme is detailed in the full version only.

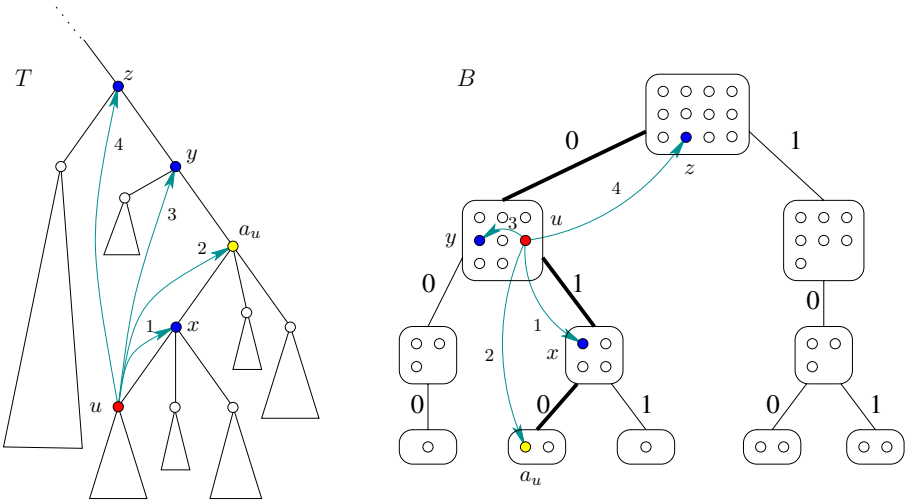

Fig. 1. An example of a k-ancestor decomposition B of an input tree T where the 4-ancestry of u, the set $A_u = \{u, x, a_u, y, z\}$, lies on the branch "010" of B

Definition 1. *A* k-ancestor decomposition *of a rooted tree* T *is a rooted binary tree* B *where nodes, called* parts*, form a partition of* $V(T)$ *such that, for any* k*-ancestry* A *of* T*, the set of parts containing a node of* A *is contained in a branch of* B*.*

2.2 Finding an Nice Ancestor Decomposition

This paragraph is devoted to the proof of the following key lemma for our result.

Lemma 1. *Every* n*-node tree* T *has a* k*-ancestor decomposition such that every part of depth* h *contains at most* $(k+1) \cdot (\log(2n/(k+1)) - h)$ *nodes of* T*.*

Before the formal proof of this result, let us give an overview. The idea is to construct from T a graph G, called hereafter *k-augmentation of T*, obtained by adding an edge between every u and its proper ancestors at distance $\leqslant k$. We then observe that every subgraph H of G has a subset of $k+1$ nodes, called *half-separator*, whose removal leaves H in connected components with less than $|V(H)|/2$ nodes. The root of the willing decomposition B is constructed by finding iteratively $O(\log(n/k))$ half-separators in G, and by grouping all the resulting connected components in two sets V_1, V_2, each with less than $n/2$ nodes. By this way we can guarantee that there are no edges of G between V_1 and V_2 since these two sets are separated by an union of separators in G. The tree B is completed by performing similarly and recursively the subgraphs of $G[V_1]$ and $G[V_2]$. Eventually, B is a k-ancestor decomposition since we observe that:

1. k-ancestries of T induce cliques in G; and
2. all edges of G belong to the same branch in B.

Let us now formalize the proof, and let us first show that:

Property 1. Every induced subgraph of the k-augmentation of T has a half-separator of size at most $k+1$ that is a clique.

Proof. It is known [16] that every *chordal* graph G, that is a graph without any induced cycle of length greater than three, has a half-separator of size at most the maximum clique size of G. Moreover, such a half-separator can be computed in linear time, i.e., $O(|V(G)| + |E(G)|)$ time.

So Property 1 is derived from the fact that every induced subgraph of a chordal graph is a chordal graph as well, and that the k-augmentation of T is a chordal graph of maximum clique size $k+1$.

Let G be the k-augmentation of T. Assume G has a clique K of size $> k+1$. Observe that in G there is no edge between unrelated nodes of T. (Two nodes of a rooted tree are said *related* if one is the ancestor of the other. They are *unrelated* otherwise.) In other words, all the nodes of K belong to the same branch of T. It follows that the distance in T between the closest node (from the root of T), say u, and the farthest node, say v, of K is $\geqslant k+1$. Because u and v are related and are at distance $> k$ in T, they are not adjacent in G: a contradiction with the fact that u and v belong to a clique of G.

Assume now G has a induced cycle C length > 3. Since G has no edges between unrelated nodes, there must exist three consecutive nodes in C, say u, v, w, such that u is ancestor of v, and w is ancestor of v (otherwise C would be a path). Either w is ancestor of u, or u is ancestor of w. In both cases, u and w are related and at distance $\leqslant k$ in T, so there are adjacent in G: a contradiction with the fact that C is an induced cycle of length > 3.

We have therefore proved that G is chordal and of maximum clique size $\leqslant k+1$. □

In the following let us denote by G the k-augmentation of T, and let us denote by Half-Separator(H) a function that returns a half-separator for a subgraph H satisfying Property 1, that is a separator whose size is at most the maximum clique size of H.

We now restrict our attention on specific half-separators. A half-separator S of a graph H is said *binary* if the connected components of $H \setminus S$ can be partitioned in at most two sets, each with a total number of nodes at most $|V(H)|/2$.

We consider the following procedure (Algorithm 1) that given a graph satisfying Property 1 returns a binary half-separator R and the resulting partition (V_1, V_2) for the nodes of $H \setminus R$.

To construct the k-ancestor decomposition B for T we apply Algorithm 1 on G, and we select R as the root of B. The tree B is then completed by applying recursively Algorithm 1 on $G[V_1]$ and $G[V_2]$, and linking to R the resulting decompositions if they are non-empty. Such a recursive approach is possible because Property 1 holds for any induced subgraph of G, so in particular for $G[V_1]$ and $G[V_2]$.

Eventually B is a k-ancestor decomposition of T because every k-ancestry of T induces a clique in G, and no edge connects $G[V_1]$ to $G[V_2]$ since R is a

Algorithm 1: BINARY HALF-SEPARATOR

Input : a subgraph G satisfying Property 1

Output: a binary half-separator R and the associated node partition (V_1, V_2)

$H := G$; $V_1 := V_2 := R := \varnothing$
while $|V(H)| > k+1$ **do**
  $S :=$ HALF-SEPARATOR(H); $H := H \setminus S$; $R := R \cup S$
  forall *connected component* C *of* H *except the largest* **do**
    $H := H \setminus C$;
    if $|V_1| > |V_2|$ **then** $V_2 := V_2 \cup V(C)$ **else** $V_1 := V_1 \cup V(C)$
$R := R \cup H$

half-separator. So, every k-ancestry of T belongs to parts contained in a same branch of B.

It remains to estimate the size of R, V_1, and V_2 returned by Algorithm 1.

At each iteration of the while-main the size of H is divided by at least two since all the resulting components of $H \setminus S$ are removed from H, and the remaining largest component is of size at most $|V(H)|/2$. So, there is at most $\log(n/(k+1))$ iterations where $n = |V(G)|$.

At each step the size of R increases by at most $k+1$ vertices, and the final step add at most $k+1$ vertices to R. Therefore, $|R| \leqslant (k+1) \cdot (\log(n/(k+1)) + 1) = (k+1) \cdot (\log(2n/(k+1)) - h)$ with $h = 0$.

In order to insure the correctness of the decomposition, we need to show the following loop invariant (P).

$$(P): \quad |V_1|, |V_2| \leqslant n/2 \ \text{ and } \ V(H) \cup V_1 \cup V_2 \cup R = V(G)$$

It is straightforward to verify that (P) is true at the beginning of the main loop. Let us show that (P) remains true at the end of the nested loop. The loop invariant clearly remains true after computation of half-separator and statement $H := H \setminus S$ and $R := R \cup S$. Assume w.l.o.g. that $|V_1| \geqslant |V_2|$. We have $V(H) \cup V_1 \cup V_2 \subseteq V(G)$ since (P) is true. We obtain the following relation for the size of the sets.

$$\begin{aligned}
|V(H)| + |V_1| + |V_2| &\leqslant n \\
|V(H)| &\leqslant n - |V_1| - |V_2| \ \leqslant \ n - 2|V_2| \quad (|V_1| \geqslant |V_2|) \\
|V(H)|/2 &\leqslant n/2 - |V_2| \\
|V(C)| &\leqslant n/2 - |V_2| \quad \text{(there is yet another larger component in } H) \\
|V(C)| + |V_2| &\leqslant n/2
\end{aligned}$$

Property (P) remains true at the end of the nested loop. Property (P) is a loop invariant and so is true at the end of the main loop. We have that $|V_1|, |V_2| \leqslant n/2$. Moreover, there is no edge linking vertices of V_1 to vertices of V_2 since what we add to V_1 or V_2 is a full connected component of H.

Since the size of V_1 and V_2 are $\leqslant n/2$, by induction, the size of the root part in the decomposition of $G[V_1]$ and $G[V_2]$ (so parts of depth $h = 1$ in B) would

be at most $(k+1)\cdot(\log(2(n/2)/(k+1))) = (k+1)\cdot(\log(2n/(k+1))-1)$. And more generally, for an arbitrary depth $h \geqslant 0$, the parts of depth h are of size at most $(k+1)\cdot(\log(2(n/2^h)/(k+1)))) = (k+1)\cdot(\log(2n/(k+1))-h)$ as claimed.

This completes the proof of Lemma 1.

2.3 The Labels

Let X be a part of B at depth h. We denote by $\mathrm{path}(X)$ the binary word of length h defining the unique path from the root of T to X. We associated with each $u \in X$ its *rank*, a unique integer $\mathrm{rank}(u) \in [0, |X|)$, and its *position*, defined by the pair $\mathrm{pos}(u) = (h, \mathrm{rank}(u))$.

The *apex* of a k-ancestry A is the node $a \in A$ with the deepest part and, if equality, with the largest rank (see the yellow node of Fig. 2.1). Observe that the positions are relative to a branch of B: every pair of nodes whose parts are on the same branch have distinct positions, and thus the parts of any two nodes having the same positions cannot be related.

Let u be a node of T, let A_u be its k-ancestry, a_u the apex of A_u, and B_{a_u} the part in B that contains a_u. The label of u is defined by the following quadruple:

$$\mathrm{label}(u) = (\mathrm{path}(B_{a_u}), \mathrm{rank}(a_u), d_{a_u}, P_u)$$

where:

- d_{a_u} is the distance in T from u to a_u; and
- $P_u = \{\mathrm{pos}(v) \mid v \in A_u, v \neq a_u\}$.

In order to optimize the query time, we assume that P_u is implemented as an array ordered by distances from u. For concreteness, the label of u in the example of Fig. 2.1 is[4]:

$$\mathrm{label}(u) = (\text{"}010\text{"}, 0, 2, \{(1,5),(2,0),(1,3),(0,9)\}) \ .$$

Let $\mathrm{pos}(A_u) = \{\mathrm{pos}(v) \mid v \in A_u\}$. It is not difficult to see that any k-ancestry A_u is uniquely defined by the pair $(\mathrm{pos}(A_u), \mathrm{path}(B_{a_u}))$, i.e., the set of its positions and the path leading to its apex. Indeed, as said previously, nodes lying on a same branch of B have pairwise distinct positions, and nodes lying on different branches can be identified from the path of their apices (that must therefore differ). The set $\mathrm{pos}(A_u)$ is not a field of $\mathrm{label}(u)$, P_u misses $\mathrm{pos}(a_u)$. However, it can be computed since $\mathrm{pos}(a_u) = (|\mathrm{path}(B_{a_u})|, \mathrm{rank}(a_u))$.

We first bound the length of the labels. Then we will explain how to solve a (k_1, k_2)-relation query, and we will detail an efficient implementation.

Lemma 2. *The label length is* $\log n + O(k \log(k \log(n/k)))$. *For* $k = 1$, *the label length is* $\log n + 2\log\log n + O(1)$.

[4] Ranks of nodes in each part are ordered by rows from left to right and then from top to bottom rows.

Proof. Let $M = \lfloor (k+1)\log(2n/(k+1)) \rfloor$. By Lemma 1, the parts of B have at most M nodes. Consider label(u), and let $h = |\text{path}(B_{a_u})|$ be the depth of B_{a_u}.

The binary word path(B_{a_u}) is of length exactly h. We have rank$(a_u) \in [0, M)$, so $\log M + O(1)$ bits suffice. The value $d_{a_u} \in [0, k]$, so $O(\log k)$ bits suffice. Each position $(h', r') \in P_u$ can be stored with $\log h + \log M + O(1)$ bits since $h' \leqslant h$ and $r' < M$. Moreover, $|P_u| = k$, so $k \cdot (\log h + \log M) + O(k)$ bits suffices for P_u.

Overall, given h, the length of label(u) is at most:

$$|\text{label}(u)| \;\leqslant\; h + \log M + k \cdot (\log h + \log M) + O(k) \;.$$

We will now upper bound the length of each term by a function depending only on the parameters of the problem (here n and k), and not on parameters depending on a tree or a particular node, like h. Therefore, each of the four fields of label(u) can be coded by a binary string of predefined length, and do not require extra field separators.

By Lemma 1, the size of the parts in B that are at depth $\log(n/(k+1))$ are of size at most $(k+1) \cdot (\log(2n/(k+1)) - \log(n/(k+1))) \leqslant k+1$. From the while-condition in Algorithm 1, if $|V(H)| \leqslant k+1$, then the input graph is not separated at all, implying that the part is actually a leaf of B. Therefore B has depth at most $h_0 \leqslant \log(n/(k+1)) < \log(n/k)$. So bounding $h \leqslant h_0$ we obtain:

$$\begin{aligned}
|\text{label}(u)| &\leqslant h_0 + \log M + k \cdot (\log h_0 + \log M) + O(k) \\
&\leqslant h_0 + k \log h_0 + (k+1) \log M + O(k) \\
&\leqslant \log(n/k) + k \log\log(n/k) + (k+1)\log((k+1)\log(2n/(k+1))) + O(k) \\
&\leqslant \log(n/k) + (2k+1)\log\log(n/k) + O(k \log k) \\
&\leqslant \log(n/k) + O(k) \cdot (\log\log(n/k) + \log k) \\
&\leqslant \log n + O(k \log(k \log(n/k))) \;.
\end{aligned}$$

For $k = 1$, the above formula gives $\log n + 3\log\log n + O(1)$ from the above 4th equation. We can slightly improved the above analysis by observing that the two first fields of label(u), namely path(B_{a_u}) and rank(a_u), can be jointly encoded with $\log n + O(1)$ bits instead of $h_0 + \log M \sim \log(n/k) + \log\log(n/k)$ bits.

We remark that for $k = 1$, the k-augmentation G of T is T itself. As a consequence, the size of the half-separator in Property 1 can be reduced since it is well-known that every forest has a single node that halves the forest, rather than a clique separator of size $k+1 = 2$. It follows that in such 1-ancestor decomposition of T the parts of depth h contain at most $\alpha = \log n - h + O(1)$ nodes instead of $(k+1)(\log(2n/(k+1)) - h) = 2(\log n - h)$ as claimed in Lemma 1. A direct consequence is that the two first terms of label(u) can be coded jointly by a string W of $\log n + O(1)$ bits as follows: W is composed of path(B_{a_u}) of length h concatenated to the word $1 \circ 0^{\text{rank}(a_u)}$, i.e., the unary representation of rank(a_u) preceded with a 1. The length of this word is $h + 1 + \alpha = \log n + O(1)$, and clearly the two fields can be extracted by seeking the least significant bit of W. The two remaining fields of label(u) still have a length bounded by

$2k \log\log(n/k) + O(k \log k)$. Overall, the label length of label(u), for $k = 1$, is at most $\log n + 2 \log\log n + O(1)$.

This completes the proof of Lemma 2. □

Actually a finer analysis shows that the label length for $k = 1$ is no more than $\log n + 2 \log\log n + 2$ for every $n \geqslant 16$.

2.4 Relationship Testing

Let u, v be two nodes of T with k-ancestry A_u and A_v respectively. Consider any pair (k_1, k_2) of integers $\leqslant k$. Recall that to check whether (u, v) is (k_1, k_2)-related or not it suffices to check if the nearest common ancestor between u and v (which is a node $z \in A_u \cap A_v$) is at distance k_1 from u and k_2 from v.

Observing that there is exactly one ancestor $z_u \in A_u$ at distance k_1 from u, and one ancestor $z_v \in A_v$ at distance k_2 from v, it suffices to check that $z_u = z_v$, and that z_u is the least common ancestor, that is there is no $z \in A_u \cap A_v$ at distance $k_1 - 1$ from u and $k_2 - 1$ from v.

Let label(u) = $(\text{path}(B_{a_u}), \text{rank}(a_u), d_{a_u}, P_u)$ and label(v)= $(\text{path}(B_{a_v}), \text{rank}(a_v), d_{a_v}, P_v)$. We denote by $\text{path}(X)[0..h]$ the prefix of length h of[5] $\text{path}(X)$.

The following lemma tell us how to check that $z \in A_u \cap A_v$ from the position of z and the labels of u and v. Recall that positions are relative to the branches of B, so a given position (h, r) may correspond to different nodes of T. Unfortunately, we cannot simply check that $\text{pos}(z) \in \text{pos}(A_u) \cap \text{pos}(A_v)$.

Property 2. Assume $z \in A_u$, and $\text{pos}(z) = (h, r)$. Then, $z \in A_v$ if and only if $\text{pos}(z) \in \text{pos}(A_v)$ and $\text{path}(B_{a_u})[0..h] = \text{path}(B_{a_v})[0..h]$.

Proof. Let $z \in A_u$, and let B_z be its part in the decomposition B. We have that $\text{path}(B_z)$ is a prefix of $\text{path}(B_{a_u})$ since $z \in A_u$. If the depth of z is h, then $\text{path}(B_z) = \text{path}(B_{a_u})[0..h]$. Similarly, $z \in A_v$ implies that $\text{path}(B_z) = \text{path}(B_{a_v})[0..h]$. Obviously, $z \in A_v$ implies $\text{pos}(z) \in \text{pos}(A_v)$. Therefore, we have shown that $z \in A_v$ implies $\text{pos}(z) \in \text{pos}(A_v)$ and $\text{path}(B_{a_u})[0..h] = \text{path}(B_{a_v})[0..h] = \text{path}(B_z)$.

Conversely, if $\text{pos}(z) = (h, r) \in \text{pos}(A_v)$ and $\text{path}(B_{a_u})[0..h] = \text{path}(B_{a_v})[0..h]$, then, since $z \in A_u$, the part of z, B_z is given by $\text{path}(B_z) = \text{path}(B_{a_u})[0..h] = \text{path}(B_{a_v})[0..h]$. It follows that the position (h, r) corresponds to a node of B_z which is moreover in A_v. In B_z, there is a unique node whose rank is r: node z. So $z \in A_v$. □

For every distance $i \in \{0, \ldots, k\}$, let us denote by $\text{pos}(A_u)[i]$ the position of the ith ancestor of u, i.e., at distance i from u. Note that $\text{pos}(A_u)[i]$ can be extracted from label(u) in constant time as follows (assuming that P_u is ordered according to the distance from u):

Extract $\text{pos}(A_u)[i]$ (given label(u)):

[5] Such prefix can be extracted (shift) in constant time in the word RAM model, because $\text{path}(X)$ is a binary word of length $\leqslant \log n$.

1. If $i = d_{a_u}$, then return $\mathrm{pos}(a_u)$, i.e., $(|\mathrm{path}(B_{a_u})|, \mathrm{rank}(a_u))$.
2. If $i > d_{a_u}$, then $i = i - 1$.
3. Return $P_u[i]$.

According to Lemma 2, to check whether $z \in A_u \cap A_v$ we have to check that $\mathrm{pos}(z) = (h, r) \in \mathrm{pos}(A_u) \cap \mathrm{pos}(A_v)$ and that the two prefixes of length h corresponds. This leads to the following test procedure:

FINAL TEST (whether (u, v) is (k_1, k_2)-related given label(u) and label(v)):

1. Extract $(h_u, r_u) = \mathrm{pos}(A_u)[k_1]$ and $(h_v, r_v) = \mathrm{pos}(A_v)[k_2]$.
2. If $(h_u, r_u) \neq (h_v, r_v)$, then return FALSE.
3. If $\mathrm{path}(B_{a_u})[0..h_u] \neq \mathrm{path}(B_{a_v})[0..h_v]$, then return FALSE.
4. If $k_1 = 0$ or $k_2 = 0$, then return TRUE.
5. Extract $(h'_u, r'_u) = \mathrm{pos}(A_u)[k_1 - 1]$ and $(h'_v, r'_v) = \mathrm{pos}(A_v)[k_2 - 1]$.
6. If $(h'_u, r'_u) \neq (h'_v, r'_v)$, then return TRUE.
7. If $\mathrm{path}(B_{a_u})[0..h'_u] \neq \mathrm{path}(B_{a_v})[0..h'_v]$, then return TRUE.
8. Return FALSE.

Lemma 3. *Any k-relationship can be tested in constant time.*

Proof. The above procedure clearly takes a constant time, and its validity is derived from Property 2. □

Combining Lemma 1, Lemma 2, and Lemma 3, we have proved Theorem 1.

3 Conclusion and Further Works

We have constructed in this paper a k-relationship scheme for n-node trees with $\log n + O(k \log(k \log(n/k)))$ bit labels. This scheme implies that distances in trees can be computed as well with labels of $\log n + o(\log n)$ bits if the distance is $o(\log n / \log\log n)$. We leave open the following two questions:

- Design a distance labeling scheme for trees with $\log n + o(\log n)$ bit labels and for larger distances, say for distances up to $\log n$.
- Design a distance labeling scheme for small distances for bounded treewidth graphs.

References

1. Abiteboul, S., Alstrup, S., Kaplan, H., Milo, T., Rauhe, T.: Compact labeling schemes for ancestor queries. SIAM Journal on Computing 35, 1295–1309 (2006)
2. Abiteboul, S., Kaplan, H., Milo, T.: Compact labeling schemes for ancestor queries. In: 12^{th} Symposium on Discrete Algorithms (SODA), pp. 547–556 (January 2001)
3. Abraham, I., Balakrishnan, M., Kuhn, F., Malkhi, D., Talwar, K., Ramasubramanian, V.: Reconstructing approximate tree metrics. In: 26^{th} Annual ACM Symposium on Principles of Distributed Computing (PODC), pp. 43–52. ACM Press, New York (2007)

4. Alstrup, S., Bille, P., Rauhe, T.: Labeling schemes for small distances in trees. In: 14^{th} Symposium on Discrete Algorithms (SODA), ACM-SIAM, pp. 689–698 (2003)
5. Alstrup, S., Bille, P., Rauhe, T.: Labeling schemes for small distances in trees. SIAM Journal on Discrete Mathematics 19, 448–462 (2005)
6. Alstrup, S., Rauhe, T.: Small induced-universal graphs and compact implicit graph representations. In: 43^{rd} Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 53–62. IEEE Computer Society Press, Los Alamitos (2002)
7. Bazzaro, F., Gavoille, C.: Localized and compact data-structure for comparability graphs. In: Deng, X., Du, D.-Z. (eds.) ISAAC 2005. LNCS, vol. 3827, pp. 1122–1131. Springer, Heidelberg (2005)
8. Chepoi, V.D., Dragan, F.F., Vaxes, Y.: Distance and routing labeling schemes for non-positively curved plane graphs. Journal of Algorithms 61(2), 60–88 (2006)
9. Chung, F.R.K.: Universal graphs and induced-universal graphs. Journal of Graph Theory 14, 443–454 (1990)
10. Dragan, F.F., Lomonosov, I.: On compact and efficient routing in certain graph classes. In: Fleischer, R., Trippen, G. (eds.) ISAAC 2004. LNCS, vol. 3341, pp. 402–414. Springer, Heidelberg (2004)
11. Fraigniaud, P., Gavoille, C.: Routing in trees. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, pp. 757–772. Springer, Heidelberg (2001)
12. Gavoille, C., Ly, O.: Distance labeling in hyperbolic graphs. In: Deng, X., Du, D.-Z. (eds.) ISAAC 2005. LNCS, vol. 3827, pp. 1071–1079. Springer, Heidelberg (2005)
13. Gavoille, C., Paul, C.: Optimal distance labeling schemes for interval and circular-arc graphs. In: Di Battista, G., Zwick, U. (eds.) ESA 2003. LNCS, vol. 2832, pp. 254–265. Springer, Heidelberg (2003)
14. Gavoille, C., Peleg, D.: Compact and localized distributed data structures. Distributed Computing 16, 111–120 (2003) PODC 20-Year Special Issue
15. Gavoille, C., Peleg, D., Pérennès, S., Raz, R.: Distance labeling in graphs. Journal of Algorithms 53, 85–112 (2004)
16. Gilbert, J.R., Rose, D.J., Edenbrandt, A.: A separator theorem for chordal graphs. SIAM Journal on Algebraic and Discrete Methods 5, 306–313 (1984)
17. Kannan, S., Naor, M., Rudich, S.: Implicit representation of graphs. In: 20^{th} Annual ACM Symposium on Theory of Computing (STOC), pp. 334–343. ACM Press, New York (1988)
18. Korman, A., Kutten, S., Peleg, D.: Proof labeling system. In: 24^{th} Annual ACM Symposium on Principles of Distributed Computing (PODC), pp. 9–18. ACM Press, New York (2005)
19. Korman, A., Peleg, D.: Compact separator decompositions in dynamic trees and applications to labeling schemes. In: 21^{st} International Symposium on Distributed Computing (DISC). LNCS, vol. 4731, Springer, Heidelberg (2007)
20. Korman, A., Peleg, D., Rodeh, Y.: Constructing labeling schemes through universal matrices. In: Asano, T. (ed.) ISAAC 2006. LNCS, vol. 4288, pp. 409–418. Springer, Heidelberg (2006)
21. Thorup, M., Zwick, U.: Compact routing schemes. In: 13^{th} Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA), pp. 1–10. ACM Press, New York (2001)