Laboratoire Bordelais de Recherche en Informatique
UMR 5800 - Université Bordeaux 1, 351, cours de la Libération,
33405 Talence CEDEX, France

Research Report `RR-1389-06`

Short Labels by Traversal and Jumping

by Nicolas Bonichon, Cyril Gavoille, and Arnaud Labourel

March, 2006

Short Labels by Traversal and Jumping

Nicolas Bonichon* Cyril Gavoille* Arnaud Labourel*

Abstract

In this paper, we propose an efficient implicit representation of caterpillars and binary trees with n vertices. Our schemes, called *Traversal & Jumping*, assign to vertices of the tree distinct labels of $\log_2 n + O(1)$ bits, and support constant time adjacency queries between any two vertices by using only their labels. Moreover, all the labels can be constructed in $O(n)$ time.

Keywords: labeling schemes, binary trees, caterpillars

1 Introduction

The two basic ways of representing a graph are adjacency matrices and adjacency lists. The latter representation is space efficient for sparse graphs, but adjacency queries require searching in the list, whereas matrices allow fast queries to the price of a super-linear space. Another technique, called *implicit representation* or *adjacency labeling scheme*, consists in assigning labels to each vertex such that adjacency queries can be computed alone from the labels of the two involved vertices without any extra information source. The goal is to minimize the maximum length of a label associated with a vertex while keeping fast adjacency queries.

Adjacency labeling schemes, introduced by [Bre66, BF67], have been investigated by [KNR88, KNR92]. They construct for several families of graphs adjacency labeling schemes with $O(\log n)$-bit labels. In particular, for trees the scheme consists in: 1) choosing an arbitrary prelabeling of the n vertices, a permutation of $\{1, \ldots, n\}$; 2) choosing a root; and 3) setting the label of a vertex to be the pair formed by its prelabel and the prelabel of its parent. The adjacency test checks whether the prelabel for one vertex equals the parent prelabel of the other vertex. Such labels are of $2\lceil \log n \rceil$ bits[1], whereas $\lceil \log n \rceil$ bits are clearly necessary since labels must be different.

*Laboratoire Bordelais de Recherche en Informatique, University of Bordeaux, Bordeaux, France. {bonichon,gavoille,labourel}@labri.fr. The three authors are supported by the project "GeoComp" of the ACI Masses de Données.

[1] All the logarithms are in base two.

Improving the label length of this straightforward scheme is not an easy task. It has been however improved in a non trivial way by [AKM01] to $1.5 \log n + O(\log \log n)$ bits, and more recently to $\log n + O(\log^* n)$ bits[2] [AR02], leaving open the question of whether trees enjoy a labeling scheme with $\log n + O(1)$ bit labels.

1.1 Related work

Motivated by applications in XML search engines, and distributed applications as peer-to-peer networks or network routing, several other distributed data-structures with optimal $O(\log n)$-bit labels, have been developed.

For instance, routing in trees [FG01, TZ01], near-shortest path routing in specific networks [BG05, DL02, DL04], distance queries for interval, circular-arc, and permutation graphs [BG05, GP03a], etc. have $O(\log n)$-bit distributed data-structures. And, specifically for several queries on trees, we have: nearest common ancestor [AGKR04] with $O(\log n)$-bit labels, ancestry [AAK+05] with $\log n + O(\sqrt{\log n})$ bit labels, and small distance queries and other related functions with $\log n + \Theta(\log \log n)$ bit labels [KM01, ABR05]. Interestingly, it is shown in [ABR05] that for sibling queries in trees of maximum degree Δ, $\log n + \Theta(\log \log \Delta)$ bit labels are necessary and sufficient. A survey on labeling schemes can be founded in [GP03b]. All these schemes achieve labeling of length[3] $\log n + \omega(1)$.

To our best knowledge, for reasonably large families of graphs, no distributed data-structure is known to have an optimal label size up to an additive constant. In particular, for adjacency queries in trees, the current lower bound is $\log n$ and the upper bound is $\log n + O(\log^* n)$ [AR02]. This latter scheme, based on a recursive decomposition of the tree in $\Theta(\log^* n)$ levels, has adjacency query time of $\Omega(\log^* n)$.

1.2 Our contributions

In this paper we present adjacency labeling schemes for caterpillars (i.e., a tree whose nonleaf vertices induce a path), and binary trees with n vertices. Both schemes assign distinct labels of $\log n + O(1)$ bits, and support constant time adjacency queries. Moreover, all the labels can be constructed in $O(n)$ time. We observe that the recursive scheme of [AR02] for general trees does not simplify for caterpillars or binary trees. The worst-case label length remains $\log n + O(\log^* n)$ and the adjacency query time $\Omega(\log^* n)$.

As far as we know, this is the first $\log n + O(1)$ bit adjacency labeling supporting constant query time for a family of trees including trees with an arbitrary numbers of arbitrary degree vertices (caterpillars). The technique, called *Traversal & Jumping*, is interesting on its own, and we believe that it might be extended to larger families of graphs, and to other queries.

[2] $\log^* n$ denotes the number of times log should be iterated to get a constant.
[3] $f(n) = \omega(g(n))$ if and only if $g(n) = o(f(n))$.

1.3 Outline of techniques

To introduce our labeling technique, let us consider an n-vertex caterpillar whose path is $x_1, \ldots, x_k$ and where the j-th leaf of x_i is denoted by $y_{i,j}$.

The first naive approach consists in labeling each vertex x_i with the pair $(i, 0)$ and $y_{i,j}$ with (i, j). The adjacency test is then trivial. This labeling, which is a variant of the tree labeling scheme presented above, is not efficient since every pair of nonnegative integers (i, j) with $i + j \leqslant n$ will be assigned by the scheme to some caterpillars. There are at least $(n/2)^2$ such pairs, yielding some labels of at least $2 \log n - O(1)$ bits.

A second less trivial labeling (with distinct labels) assigns to each vertex $y_{i,j}$ the pair (r_i, j) where r_i is the rank of the number of leaves of x_i, so that less bits are used for r_i if x_i has many leaves, leaving room for the index j. Because $j \leqslant n/r_i$, the label length reduced now to $\lceil \log r_i \rceil + \lceil \log(n/r_i) \rceil \leqslant \log n + O(1)$. However, the fields of the pair (r_i, j) have variable length, so $\log \log \min \{r_i, j\} + O(1)$ bits are required to code the position of the two values of the pair. Moreover, this scheme does not give adjacency for two nodes in the path. Anyway, as all possible pairs (r_i, j) can occur, this required an extra information of $\log \log \sqrt{n}$ bits (in the worst-case $\min \{r_i, j\} \geqslant \sqrt{n}$), yielding labels of length at least $\log n + \log \log n - O(1)$.

A third solution is to apply some recursive decomposition, as in [AR02]. However, any decomposition in a non-constant levels produces labels with a non-constant number of fields, yielding a label length of $\log n + \omega(1)$ bits, furthermore with $\omega(1)$ adjacency query time. Labelings with $\log n + O(1)$ bits require new ideas.

Roughly speaking, the Traversal & Jumping technique consists in:

1. Selecting a suitable traversal of the tree (or of the graph);
2. Associating with each vertex x some information $C(x)$;
3. Performing the traversal and assign the labels with increasing but non necessarily consecutive numbers to the vertices.

Intuitively, the adjacency test between x and y is done on the basis of $C(x)$ and $C(y)$. Actually, the jumps achieved in Step 3 are done by selecting an interval associated with each vertex in which its label must be. It is important to note that the intervals are ordered in the same way as the corresponding vertices in the traversal. Moreover, all vertex intervals must be disjoint. The position of the label of x in its interval is tuned in order to encode $C(x)$ in the label in a self-extracting way. In general, the information $C(x)$ determines the intervals length of all the neighbours of x which are after in the traversal.

The main difficulty is to design the minimal information $C(x)$ and to tune the jumps, i.e., the interval length. The maximum label length is simply determined by the value of the last label assigned during the traversal.

This technique fundamentally differs from previous schemes, in which a label is essentially viewed as a unique prelabel of $\lceil \log n \rceil$ bits plus some small extra fields, inevitably leading to labels of $\log n + \omega(1)$ bits. On the contrary, Traversal & Jumping abandons this representation, and uses the full range of values $[0, O(n)]$ to get labels of length $\log n + O(1)$.

Section 2 presents the scheme for caterpillars, and Section 3 for binary trees. We propose further works in Section 4.

1.4 Preliminaries

We assume a RAM model of computation with $\Omega(\log n)$-bit words. In this model, standard arithmetic operations on words of $O(\log n)$ bits can be done in constant time. These include additions, comparisons, binary masks, shifting, MSB and LSB (returning respectively the position of the most and least significant bit of a word).

Given a binary string A, we denote by $|A|$ its length, and for a binary string B, $A \circ B$ denotes the concatenation of A followed by B.

Given an $x \in \mathbb{N}$, we denote by $\lg x = \log \max\{x, 1\}$, and by $\mathrm{bin}(x)$ its standard binary representation. We have $|\mathrm{bin}(x)| = \lfloor \lg x \rfloor + 1$. We denote by $\mathrm{val}(w)$ the integer x such that $w = \mathrm{bin}(x)$. When it is clear from the context, we confuse w and $\mathrm{val}(w)$. We also denote by $\lceil x \rceil_2 = 2^{\lceil \lg x \rceil}$.

A *code* is a set of words, and a code is *suffix-free* if no words of the code is the ending of another one. A basic property of suffix-free codes it that they can be composed, by the concatenation of a fixed number of fields, to form new suffix-free codes. A simple suffix-free code is defined by $\mathrm{code}_0(x) = 1 \circ 0^x$, where 0^x is the binary string composed of x zeros. This code extends to more succinct codes defined recursively by $\mathrm{code}_{i+1}(x) = \mathrm{bin}(x) \circ \mathrm{code}_i(|\mathrm{bin}(x)| - 1)$ for every $i \geqslant 0$. It is easy to check that, for every $i \geqslant 0$, code_i is suffix-free. E.g., $\mathrm{code}_0(5) = 100000$, $\mathrm{code}_1(5) = 101\,100$, and $\mathrm{code}_2(5) = 101\,10\,10$.

If a word w has $\mathrm{code}_i(x)$ as suffix, then x can be extracted from w in $O(i)$ time (in particular with the use of LSB to extract the length of code_0). In the sequel, any integer sequence $x_1, \ldots, x_k$ can be stored as a suffix $\mathrm{code}_i(x_1) \circ \cdots \circ \mathrm{code}_i(x_k)$, and can be extracted in $O(ik)$ time.

In this paper, we will essentially use code_i for $i \in \{0, 1, 2\}$. We check that for every $x \in \mathbb{N}$, $|\mathrm{code}_0(x)| = x + 1$, $|\mathrm{code}_1(x)| = 2\lfloor \lg x \rfloor + 2$, and $|\mathrm{code}_2(x)| = \lfloor \lg x \rfloor + 2\lfloor \lg \lfloor \lg x \rfloor \rfloor + 3$.

Claim 1 *Let w be a word, and z an integer. One can compute in constant time an integer $x \in [z, z + 2^{|w|})$ such that w is a suffix of* $\mathrm{bin}(x)$.

Proof. Observe that, for all strings A and B, $\mathrm{val}(A \circ B) = \mathrm{val}(A) \cdot 2^{|B|} + \mathrm{val}(B)$, and that $\mathrm{val}(A) < 2^{|A|}$.

Let $u = \lfloor z/2^{|w|} \rfloor$ and $v = z \bmod 2^{|w|}$, so that $z = u \cdot 2^{|w|} + v$. Set $b = 0$ if $\mathrm{val}(w) \geqslant v$, and set $b = 1$ otherwise. The integer x is defined by $\mathrm{bin}(x) = \mathrm{bin}(u + b) \circ w$, that clearly contains w as suffix. Note that x can be computed in constant time using shifts, masks and MSB (in particular MSB is used to compute $|w|$ from w).

It remains to check that $x \in [z, z + 2^{|w|})$. We have $x = (u + b) \cdot 2^{|w|} + \mathrm{val}(w) = z - v + b \cdot 2^{|w|} + \mathrm{val}(w)$.

If $b = 0$, then $x = z - v + \mathrm{val}(w) \leqslant z + \mathrm{val}(w) < z + 2^{|w|}$. For $b = 0$, $\mathrm{val}(w) \geqslant v$, thus $x \geqslant z$.

If $b = 1$, then $x = z - v + 2^{|w|} + \mathrm{val}(w) > z + \mathrm{val}(w) \geqslant z$ since $v < 2^{|w|}$. For $b = 1$, $\mathrm{val}(w) < v$, thus $z - v + 2^{|w|} + \mathrm{val}(w) < z + 2^{|w|}$. □

2 Caterpillars

A leaf is a vertex of degree one, and an inner vertex is a nonleaf vertex. A tree is a *caterpillar* if the subgraph induced by its inner vertices is a path.

Theorem 1 *The family of caterpillars with n vertices enjoys an adjacency labeling scheme with labels of length at most $\lceil \log n \rceil + 6$ bits, supporting constant time adjacency query. Moreover, all the labels can be constructed in $O(n)$ time.*

2.1 Description of the labeling scheme

Consider a caterpillar G of n vertices. We denote by $X = \{x_1, \ldots, x_k\}$ the inner vertices of G (ordered along the path). For every i, let $Y_i = \{y_{i,1}, \ldots, y_{i,d_i}\}$ be the set of leaves attached to x_i, with $d_i = 0$ if $Y_i = \varnothing$.

The traversal used in our scheme is a prefix traversal of the caterpillar rooted at x_1 where the vertices of Y_i are traversed before the vertex x_{i+1}. According to this traversal, the inner vertex x_i stores necessary information to determine the adjacency with the vertices of $Y_i \cup \{x_{i+1}\}$. The leaves do not store any specific information in their label.

With each inner vertex x_i, we associate an interval of length p_i, for some suitable integer p_i, in which its label $\ell(x_i)$ must be. For some technical reasons, impose that $p_i = 2^{t_i+3}$ with t_i is an integer $\geqslant 0$. With the set of the labels of Y_i we associate an interval of same length: $(\ell(x_i), \ell(x_i) + p_i]$. In this interval $\ell(y_{i,j}) = \ell(x_i) + j$. Finally, the interval associated with vertex x_{i+1} is $(\ell(x_i) + p_i, \ell(x_i) + p_i + p_{i+1}]$.

The information encoded by x_i is the ordered pair (t_i, t_{i+1}). To encode this information, we propose the following suffix-free code:

$$C(x_i) \;=\; \mathrm{code}_0(t_i + 3 - |\mathrm{code}_1(t_{i+1})|) \circ \mathrm{code}_1(t_{i+1}) \;.$$

Three conditions on p_i (and so on t_i) have to be satisfied to ensure that the code is valid. The value p_i must be large enough to encode the information, large enough so that all the labels of the vertices of Y_i can be placed in the interval $(\ell(x_i), \ell(x_i) + p_i]$, and $p_i \geqslant 8$. The following relation ensures such conditions:

$$t_i \;=\; \max\left\{|\mathrm{code}_1(t_{i+1})| - 3, \lceil \lg d_i \rceil - 3, 0\right\}, \text{ with } t_{k+1} = 0 \;.$$

So, given $\ell(x_i)$, $\ell(x_{i+1})$ is computed applying Claim 1 with $w = C(x_{i+1})$ and $z = \ell(x_i) + p_i$.

One can remark that the value of t_i depends on the value of t_{i+1}. The computation of the labels can be done with two traversals of the caterpillar. The value of the t_i is computed from a traversal of the path from x_k to x_1. A second traversal (a prefix one starting from x_1) computes the labels of the vertices. Each traversal takes $O(n)$ time. Finally, an additional bit is added to the labels to determine if the vertex is an inner vertex: $\ell'(x_i) = 1 \circ \ell(x_i)$ and $\ell'(y_{i,j}) = 0 \circ \ell(y_{i,j})$.

2.2 Adjacency test

Lemma 2 *For every pair of vertices u and v, the adjacency between u and v can be computed in constant time from $\ell'(u)$ and $\ell'(v)$.*

Proof. Looking at the first bit of $\ell'(u)$ and $\ell'(v)$ one can check whether u and v belongs to X or to $Y = \bigcup_{i=1}^{i=k} Y_i$. Because two leaves cannot be adjacent, let us assume that $u \in X$ with $u = x_i$.

In constant time, we can compute $\ell(x_i)$, t_i and t_{i+1} from $\ell'(x_i)$, decoding $C(x_i)$. Recall that p_i and p_{i+1} can be directly deduced from t_i and t_{i+1}. There are two cases to consider:

- Case 1: $v \in Y$ (the first bit of the label is 0). By construction, the labels of vertices of Y_i and only these belong to the interval $(\ell(x_i), \ell(x_i) + p_i]$. Since the length of the labels is $O(\log n)$ (see Lemma 3), this test can be performed in constant time.

- Case 2: $v \in X$ (the first bit of the label is 1). Let $v = x_j$, and w.l.o.g. assume that $j > i$ (we simply check whether $\ell'(v) > \ell'(u)$).

  By construction if $j = i+1$, then $\ell(x_i) + p_i < \ell(x_j) \leqslant \ell(x_i) + p_i + p_{i+1}$. This interval may contain other labels (labels of vertices of Y_{i+1}), but the only label of inner vertex if $\ell(x_{i+1})$. This test can also be performed in constant time.

$\square$

2.3 Label length

Lemma 3 *The length of the labels is at most $\lceil \log n \rceil + 6$.*

Proof. First, let us show by induction the following property (P_m):

$$(P_m): \quad \sum_{i=m}^{k} p_i \ \leqslant\ 8\left(\sum_{i=m}^{k} \lceil d_i + 1 \rceil_2\right) - p_m \ .$$

(P_k) is true since $d_k > 0$ and $\sum_{i=k}^{i=k} p_i = p_k = \max\{8, \lceil d_k + 1 \rceil_2\} \leqslant 8 \sum_{i=k}^{k} \lceil d_i + 1 \rceil_2 - p_k$. Assume that (P_m) is true for some $m \in [2, k]$, and let us show (P_{m-1}):

Applying the induction hypothesis:

$$\sum_{i=m-1}^{k} p_i \;\leqslant\; 8 \left(\sum_{i=m}^{k} \lceil d_i + 1 \rceil_2 \right) - p_m + p_{m-1} \;.$$

There are three cases to consider:

- Case 1: $\lceil d_{m-1} \rceil_2 \leqslant 8$ and $2^{|\mathrm{code}_1(t_m)|} \leqslant 8 \Rightarrow p_{m-1} = 8$.

$$\sum_{i=m-1}^{k} p_i \;\leqslant\; 8 \left(\sum_{i=m}^{k} \lceil d_i + 1 \rceil_2 \right) - p_m + 8$$

  Since $\lceil d_{m-1} + 1 \rceil_2 \geqslant 1$:

$$\sum_{i=m-1}^{k} p_i \;\leqslant\; 8 \left(\sum_{i=m-1}^{k} \lceil d_i + 1 \rceil_2 \right) - p_m$$

  Since $p_m \geqslant p_{m-1}$:

$$\sum_{i=m-1}^{k} p_i \;\leqslant\; 8 \left(\sum_{i=m-1}^{k} \lceil d_i + 1 \rceil_2 \right) - p_{m-1}$$

- Case 2: $\lceil d_{m-1} \rceil_2 \geqslant 2^{|\mathrm{code}_1(t_m)|} \Rightarrow p_{m-1} = \lceil d_{m-1} \rceil_2$.

$$\begin{aligned}
\sum_{i=m-1}^{k} p_i \;&\leqslant\; 8 \left(\sum_{i=m}^{k} \lceil d_i + 1 \rceil_2 \right) - p_k + \lceil d_{m-1} \rceil_2 \\
&\leqslant\; 8 \left(\sum_{i=m-1}^{k} \lceil d_i + 1 \rceil_2 \right) - p_k - 7 \lceil d_{m-1} + 1 \rceil_2 \\
&\leqslant\; 8 \left(\sum_{i=m-1}^{k} \lceil d_i + 1 \rceil_2 \right) - p_{m-1}
\end{aligned}$$

- Case 3: $\lceil d_{m-1} \rceil_2 < 2^{|\mathrm{code}_1(t_m)|}$ and $8 < 2^{|\mathrm{code}_1(t_m)|} \Rightarrow p_{m-1} = 2^{|\mathrm{code}_1(t_m)|}$.

$$\sum_{i=m-1}^{k} p_i \;\leqslant\; 8\left(\sum_{i=m}^{k} \lceil d_i+1 \rceil_2\right) - p_m + 2^{|\mathrm{code}_1(t_m)|} \;.$$

Since $p_{m-1} = 2^{|\mathrm{code}_1(t_m)|} = 2^{2\lceil \log((\log(p_m)-3)+1)\rceil} \leqslant \frac{1}{2}p_m$:

$$\begin{aligned}
\sum_{i=m-1}^{k} p_i \;&\leqslant\; 8\left(\sum_{i=m-1}^{k} \lceil d_i+1 \rceil_2\right) - \frac{1}{2}p_m \\
&\leqslant\; 8\left(\sum_{i=m-1}^{k} \lceil d_i+1 \rceil_2\right) - p_{m-1} \;.
\end{aligned}$$

So (P_m) is true for any positive $m \leqslant k$. Hence:

$$\sum_{i=1}^{k} p_i \;\leqslant\; 8\left(\sum_{i=1}^{k} \lceil d_i+1 \rceil_2\right) \;.$$

The maximum label length is determined by the label of the last leaf of x_k, say $y_{k,j}$. We can bound $\ell(y_{k,j})$ by:

$$\ell(y_{k,j}) \;\leqslant\; 2\sum_{i=1}^{k} p_i \;\leqslant\; 2^4 \sum_{i=1}^{k} \lceil d_i+1 \rceil_2 \;\leqslant\; 2^5 \sum_{i=1}^{k} (d_i+1) \;\leqslant\; 2^5 n \;.$$

The length labels $\ell(y_{k,j})$ is at most $\lceil \log n \rceil + 5$. The effective labels, $\ell'(v)$, use one more bit. So the label length is at most $\lceil \log n \rceil + 6$. □

3 Binary trees

Theorem 4 *The family of binary trees with n vertices enjoys an adjacency labeling scheme with labels of length at most $\log n + O(1)$ bits, supporting constant time adjacency query. Moreover, all the labels can be constructed in $O(n)$ time.*

3.1 Description of the labeling scheme

Let T be a rooted binary tree with n vertices. For any vertex v, let T_v denote the subtree of T rooted at v. Let r_T be the root of T. We denote by v^- and v^+ respectively the left and the right child of v (if exist). We assume that the children of every inner vertex v are ordered such that the weight of v^- is at most the weight of v^+, i.e., $|V(T_{v^-})| \leqslant |V(T_{v^+})|$. For the shake of the proof, we assume that v^- exists for every inner vertex v, possibly by completing the tree with some extra vertices. Note that this at most double the size of the tree.

The traversal considered in our scheme is a prefix traversal of T in which v^- is visited before v^+, for every inner vertex v. Let $s(v)$ be the length of the interval assigned to v, and let $p(v)$ be the length of the interval of values assigned to the labels of vertices of T_v (see Fig. 1). In the scheme, the interval associate with v is at the beginning of the interval of T_v (on Fig. 2 arrows $s(v)$ and $p(v)$ are aligned on the left). The interval of T_{v^-} is at distance $s(v)$ from $\ell(v)$ (i.e., the difference of the left boundaries of the intervals is $s(v)$). The interval of T_{v^+} begins at distance $s(v) + q(v^-)$ from $\ell(v)$ (cf. Fig. 2), for a suitable length $q(v^-)$.

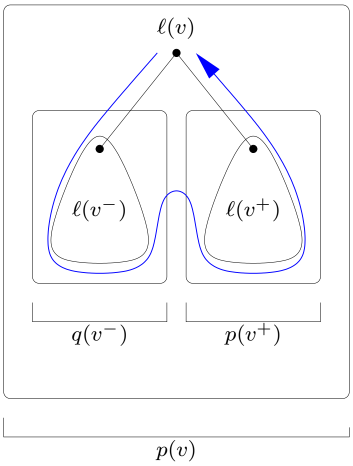

Figure 1: Traversal of the tree.

In addition, our scheme imposes that $s(v)$ is a power of 2, and that $q(v^-)$ is a square. More precisely, $s(v) = 2^{m(v)}$ for some integer $m(v) \geqslant 0$, and $q(v^-) = \lceil \sqrt{p(v^-)}\, \rceil^2$. Observe that for every v, $q(v) = p(v) + O(\sqrt{p(v)}\,)$, and that $q(v)$ can be encoded with half many bits than for $p(v)$.

To compute the adjacency with its children, vertex v stores a single bit 0 if it is a leaf, and the quadruple $(s(v^-), s(v^+), q(v^-), 1)$ if it is inner. To encode this information, we propose the following suffix-free code:

$$C(v) \;=\; \begin{cases} 0 & \text{if } v \text{ is a leaf} \\ \mathrm{code}_1(m(v^-)) \circ \mathrm{code}_1(m(v^+)) \circ \mathrm{code}_2(\sqrt{q(v^-)}\,) \circ 1 & \text{otherwise} \end{cases}$$

We set $s(v) = 2^{|C(v)|-1}$, i.e., $m(v) = |C(v)| - 1$. To compute the labels we need first to compute $s(v)$ and $p(v)$ for each vertex v of T. This is done in linear time with a postfix traversal considering the recursive relation:

$$p(v) \;=\; \begin{cases} 2 & \text{if } v \text{ is a leaf} \\ q(v^-) + p(v^+) + 2s(v) & \text{otherwise} \end{cases}$$

Then, the labels can be computed in linear time with a traversal of T, and applying Claim 1 with $w = C(v)$.

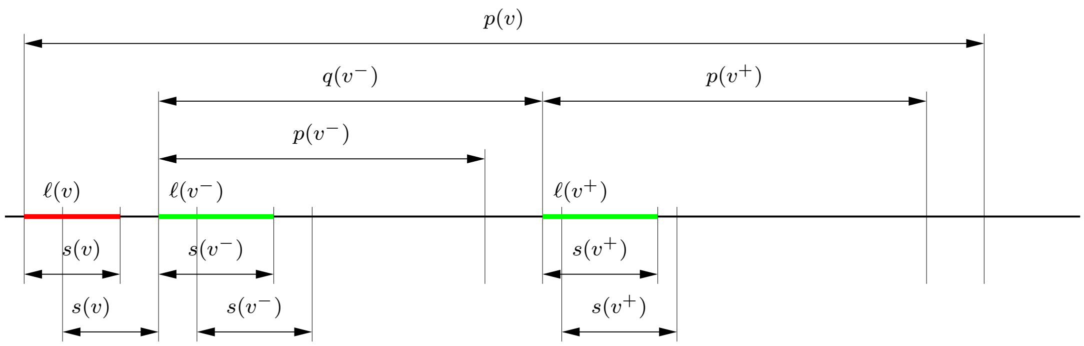

Figure 2: Description of the labeling.

3.2 Adjacency test

Lemma 5 *Let any pair of vertices v and u, the adjacency of v and u can be computed in constant time from $\ell(v)$ and $\ell(u)$.*

Proof. W.l.o.g., we can consider that $\ell(v) < \ell(u)$. To test adjacency between v and u, we use the following conditions:

v and u are adjacent if and only if y is inner and:

- either $\ell(u) \in [\ell(v) + s(v), \ell(v) + s(v) + s(v^-))$ (in this case $u = v^-$);
- or $\ell(u) \in [\ell(v) + s(v) + q(v^-), \ell(v) + s(v) + q(v^-) + s(v^+))$ (in this case $u = v^+$)

We can remark that this test can be computed in constant time from the labels of the vertices. Indeed, $s(v)$, $s(v^-)$, $s(v^+)$, and $q(v^-)$ can be extracted in constant time from $\ell(u)$.

It remains to prove the validity of this test. If v is a leaf (and $\ell(v) < \ell(u)$), v and u cannot be adjacent. To check it, it suffices to extract the last bit of $\ell(v)$. Now assume v is inner. By construction, if $u = v^-$ then $\ell(u) \in [\ell(v)+s(v), \ell(v)+s(v)+s(v^-))$. In the same way, if $u = v^+$ then $\ell(u) \in [\ell(v) + s(v) + q(v^-), \ell(v) + s(v) + q(v^-) + s(v^+))$. Moreover, $\ell(v^-)$ is the only label in $[\ell(v) + s(v), \ell(v) + s(v) + s(v^-))$ because, in the construction we make a jump from $\ell(v)$ to $\ell(v^-)$ of length $s(v)$ and we make another jump from $\ell(v^-)$ to $\ell(v^{--})$ (if exists) of length at least $s(v^-)$.

With the same argument, we prove that $\ell(v^+)$ is the only label in $[\ell(v) + s(v) + q(v^-), \ell(v) + s(v) + q(v^-) + s(v^+))$. □

3.3 Label length

Lemma 6 *The length of the labels is at most* $\log n + O(1)$.

Proof. Let B_n be the family of all binary trees of at most n vertices such that every leaf has a sibling (i.e., v^- and v^+ exist for every inner vertex v).

For every tree $T \in B_n$ with $n \geqslant 3$:

$$\begin{aligned} p(r_T) &= q(r_T{}^-) + p(r_T{}^+) + 2s(r_T) \quad \text{where} \\ \log s(r_T) &= |C(r_T)| - 1 \;=\; |\mathrm{code}_1(m(r_T{}^-))| + |\mathrm{code}_1(m(r_T{}^+))| + |\mathrm{code}_2(\lceil \sqrt{p(r_T{}^-)} \rceil)| \\ &= |\mathrm{code}_1(\log s(r_T{}^-))| + |\mathrm{code}_1(\log s(r_T{}^+))| + |\mathrm{code}_2(\lceil \sqrt{p(r_T{}^-)} \rceil)| \end{aligned}$$

Let $P(n) = \max_{T \in B_n} p(r_T)$, and $S(n) = \max_{T \in B_n} s(r_T)$. Because every label assigned to T ranges in $[0, P(n))$, our goal is to upper bound $P(n)$ by $O(n)$.

Let $i = |V(T_{r_T{}^-})|$ be the weight of the left subtree. We have $p(r_T{}^-) \leqslant P(i)$ and $p(r_T{}^+) \leqslant P(n-1-i)$. Similarly, $s(r_T{}^-) \leqslant S(i)$ and $s(r_T{}^+) \leqslant S(n-1-i)$. By the ordering of the children, observe that $i \leqslant n-1-i$, i.e., i can only range in $I = \{1, \ldots, \lfloor (n-1)/2 \rfloor\}$.

From previous equations we derive (recall that $q(v^-) = \lceil \sqrt{p(v^-)} \rceil^2$):

$$\begin{aligned} P(n) &= \max_{i \in I} \left\{ \left\lceil \sqrt{P(i)} \right\rceil^2 + P(n-i-1) + 2S \right\} \quad \text{where} \\ \log S &= |\mathrm{code}_1(\log S(i))| + |\mathrm{code}_1(\log S(n-i-1))| + |\mathrm{code}_2(\lceil \sqrt{P(i)} \rceil)| \end{aligned}$$

with $P(1) = 2$ and $S(1) = 1$. Note that for $x \in \mathbb{N}$, $2^{\lg x} \leqslant x + 1$. The two first terms of $\log S$ can be bounded by:

$$\begin{aligned} |\mathrm{code}_1(\log S(i))| &= 2 \lfloor \lg \log S(i) \rfloor + 2, \text{ and thus} \\ 2^{|\mathrm{code}_1(\log S(i))|} &\leqslant 4 \log^2 S(i) + 4, \text{ and similarly} \\ 2^{|\mathrm{code}_1(\log S(n-i-1))|} &\leqslant 4 \log^2 S(n-i-1) + 4 \ . \end{aligned}$$

By construction, $p(v) \geqslant 2s(v)$ for every v, thus $S(n) \leqslant P(n)/2$ for every n. So, bounding $(x-1)^2 \leqslant x^2 - 1$, we obtain:

$$2^{|\mathrm{code}_1(\log S(i))|} \leqslant 4 \log^2 P(i) \quad \text{and} \quad 2^{|\mathrm{code}_1(\log S(n-i-1))|} \leqslant 4 \log^2 P(n-i-1) \ .$$

Let $u = \sqrt{P(i)}$. We have $u \geqslant 1$. The third term of $\log S$ can be bounded by:

$$\begin{aligned} |\mathrm{code}_2(\lceil u \rceil)| &= \lfloor \lg \lceil u \rceil \rfloor + 2 \lfloor \lg \lfloor \lg \lceil u \rceil \rfloor \rfloor + 3, \text{ and thus} \\ 2^{|\mathrm{code}_2(\lceil u \rceil)|} &\leqslant 8 \cdot (\lceil u \rceil + 1) \cdot (\lfloor \lg \lceil u \rceil \rfloor^2 + 1) \;\leqslant\; 16u(\log^2 u + 2) \ . \end{aligned}$$

Therefore,

$$\begin{aligned} S &\leqslant 4 \cdot \log^2 P(i) \cdot \log^2 P(n-i-1)) \cdot 16\sqrt{P(i)} \cdot (\log^2 \sqrt{P(i)} + 2) \\ &\leqslant 16\sqrt{P(i)} \cdot (\log^2 P(i) + 8) \cdot \log^2 P(i) \cdot \log^2 P(n-i-1) \end{aligned}$$

One can check that for $x \geqslant 1$, $\lceil \sqrt{x} \rceil^2 \leqslant x + 2\sqrt{x}$. Hence:

$$\begin{aligned} P(n) &\leqslant \max_{i \in I} \left\{ P(i) + 2\sqrt{P(i)} + P(n-i-1) + 2S \right\} \\ &\leqslant \max_{i \in I} \left\{ P(i) + P(n-i-1) + 34\sqrt{P(i)} \log^4 P(i) \log^2 P(n-i-1) \right\} \end{aligned}$$

In particular, we deduce that $\exists \alpha, \beta, \gamma, \delta \in \mathbb{R}^+, \delta < 1/2 < \gamma < 1$ and $\delta + \gamma < 1$ such that:

$$P(n) \leqslant \max_{i \in I} \left\{ P(i) + P(n-i-1) + \alpha P(i)^\gamma P(n-i-1)^\delta + \beta \right\} .$$

The following claim shows that $P(n) = O(n)$, and so the label length is $\log n + O(1)$.

Claim 2 *Let $P(n)$ be a sequence. If there are $\alpha, \beta, \gamma, \delta \in \mathbb{R}^+, \delta < \gamma < 1, \delta + \gamma < 1$ such that $P(n) \leqslant \max_{i \in I} \left\{ P(i) + P(n-i-1) + \alpha P(i)^\gamma P(n-i-1)^\delta + \beta \right\}$ and $P(1) > 0$, then $P(n) = O(n)$.*

Proof. Let a and b be two positive constants we will determine later. Let us prove by induction the property (Q_n):

$$(Q_n): \quad P(n) \leqslant an - bn^{\gamma+\delta} .$$

Q_1 is true if and only if a and b satisfy $P(1) \leqslant a - b$. Assume that Q_i is true for $i < n$.

$$\begin{aligned} P(n) &\leqslant \max_{i \in I} \left\{ an - bi^{\gamma+\delta} - b(n-i-1)^{\gamma+\delta} + \alpha(an)^\gamma (a(n-i-1))^\delta - (a - \beta) \right\} \\ &\leqslant an - (a-\beta) + \max_{i \in I} \{ h(n,i) + f(n,i) \} \end{aligned}$$

with $h(n,i) = -bi^{\gamma+\delta} - b(n-i-1)^{\gamma+\delta}$ and $f(n,i) = a^{\gamma+\delta} n^\gamma (n-i-1)^\delta$.

In order to bound $\max_{i \in I} h(n,i)$, we compute:

$$\frac{\partial}{\partial i} h(n,i) = bi^{\gamma+\delta}(n-i-1)^{\gamma+\delta} \left(\frac{i^{1-\gamma-\delta} - (n-i-1)^{1-\gamma-\delta}}{i(n-i-1)} \right)$$

For $i \in I$, $\frac{\partial}{\partial i} h(n,i) \leqslant 0$ because $i \leqslant n-i-1$ and $\gamma > \delta$. So, we obtain:

$$\max_{i \in I} h(n,i) \leqslant -bn^{\gamma+\delta} \leqslant -2^{\gamma+\delta} b \left(\frac{n}{2} \right)^{\gamma+\delta}$$

In order to bound $\max_{i \in I} f(n,i)$, we compute:

$$\frac{\partial}{\partial i} f(n,i) = a^{\gamma+\delta} i^\gamma (n-i-1)^\delta \left(\frac{\gamma(n-i-1) - \delta i}{i(n-i-1)} \right) .$$

For $i \in I$, $\frac{\partial}{\partial i} f(n,i) \geqslant 0$ because $i \leqslant n-i-1$ and $\gamma > \delta$. So:

$$\max_{i \in I} f(n,i) \leqslant a^{\gamma+\delta} \left(\frac{n}{2} \right)^{\gamma+\delta}$$

and thus,

$$P(n) \leqslant an - \left(2^{\gamma+\delta}b - a^{\gamma+\delta}\right) n^{\gamma+\delta} - (a-\beta) \ .$$

The two constants must fulfill the following equalities:

$$\begin{cases} P(1) \leqslant a - b \\ 2^{\gamma+\delta}b - a^{\gamma+\delta}\alpha \geqslant b \\ a - \beta \geqslant 0 \end{cases}$$

For instance, it suffices to choose b such that:

$$a - P(1) \ \geqslant \ b \ \geqslant \ a^{\gamma+\delta}\alpha \quad \text{with } a \geqslant \beta$$

which is possible for a large enough since $\gamma + \delta < 1$. □

This completes the proof of Lemma 6. □

4 Conclusion

The unsolved *implicit graph representation conjecture* of [KNR88, KNR92] asks whether every hereditary[4] family of graphs with $2^{O(n \log n)}$ labeled graphs of n vertices enjoys a $O(\log n)$-bit adjacency labeling scheme. This is motivated by the fact that every family with at least $2^{cn \log n}$ labeled graphs of n vertices requires adjacency labels of at least $c \log n$ bits.

Our schemes suggest that, at least for trees, labels of $\log n + O(1)$ bits may be possible. Therefore, we propose to prove or to disprove the following:

Every hereditary family of graphs with at most $n!2^{O(n)} = 2^{n \log n + O(n)}$ *labeled graphs of* n *vertices enjoys an adjacency labeling scheme with labels of* $\log n + O(1)$ *bits.*

We observe that several well-known families of graphs are concerned by this proposition: trees, planar graphs, bounded treewidth graphs, graphs of bounded genus, graphs excluding a fixed minor (cf. [NRTW05] for counting such graphs). Proving the latter conjecture appears to be hard, e.g., the best upper bound for planar graphs is only $3 \log n + O(\log^* n)$.

[4]That is a family of graphs closed under induced subgraph taking.

References

[AAK$^+$05] Serge Abiteboul, Stephen Alstrup, Haim Kaplan, Tova Milo, and Theis Rauhe. Compact labeling schemes for ancestor queries. *SIAM Journal on Computing*, 2005.

[ABR05] Stephen Alstrup, Philip Bille, and Theis Rauhe. Labeling schemes for small distances in trees. *SIAM Journal on Discrete Mathematics*, 19(2):448–462, 2005.

[AGKR04] Stephen Alstrup, Cyril Gavoille, Haim Kaplan, and Theis Rauhe. Nearest common ancestors: A survey and a new algorithm for a distributed environment. *Theory of Computing Systems*, 37:441–456, 2004.

[AKM01] Serge Abiteboul, Haim Kaplan, and Tova Milo. Compact labeling schemes for ancestor queries. In 12^{th} *Symposium on Discrete Algorithms (SODA)*, pages 547–556. ACM-SIAM, January 2001.

[AR02] Stephen Alstrup and Theis Rauhe. Small induced-universal graphs and compact implicit graph representations. In 43^{rd} *Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 53–62. IEEE Computer Society Press, November 2002.

[BF67] Melvin A. Breuer and Jon Folkman. An unexpected result on coding the vertices of a graph. *Journal of Mathematical Analysis and Applications*, 20:583–600, 1967.

[BG05] Fabrice Bazzaro and Cyril Gavoille. Localized and compact data-structure for comparability graphs. In 16^{th} *Annual International Symposium on Algorithms and Computation (ISAAC)*, volume 3827 of Lecture Notes in Computer Science, pages 1122–1131. Springer, December 2005.

[Bre66] Melvin A. Breuer. Coding the vertexes of a graph. *IEEE Transactions on Information Theory*, IT-12:148–153, 1966.

[DL02] Feodor F. Dragan and Irina Lomonosov. New routing schemes for interval graphs, circular-arc graphs, and permutation graphs. In 14^{th} *IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS)*, pages 78–83, November 2002.

[DL04] Feodor F. Dragan and Irina Lomonosov. On compact and efficient routing in certain graph classes. In 15^{th} *Annual International Symposium on Algorithms and Computation (ISAAC)*, volume 3341 of Lecture Notes in Computer Science, pages 402–414. Springer, December 2004.

[FG01] Pierre Fraigniaud and Cyril Gavoille. Routing in trees. In Fernando Orejas, Paul G. Spirakis, and Jan van Leeuwen, editors, 28^{th} *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 2076 of Lecture Notes in Computer Science, pages 757–772. Springer, July 2001.

[GP03a] Cyril Gavoille and Christophe Paul. Optimal distance labeling schemes for interval and circular-arc graphs. In G. Di Battista and U. Zwick, editors, 11^{th} *Annual European Symposium on Algorithms (ESA)*, volume 2832 of Lecture Notes in Computer Science, pages 254–265. Springer, September 2003.

[GP03b] Cyril Gavoille and David Peleg. Compact and localized distributed data structures. *Journal of Distributed Computing*, 16:111–120, May 2003. PODC 20-Year Special Issue.

[KM01] Haim Kaplan and Tova Milo. Short and simple labels for small distances and other functions. In 7^{th} *International Workshop on Algorithms and Data Structures (WADS)*, volume 2125 of Lecture Notes in Computer Science, pages 32–40. Springer, August 2001.

[KNR88] Sampath Kannan, Moni Naor, and Steven Rudich. Implicit representation of graphs. In 20^{th} *Annual ACM Symposium on Theory of Computing (STOC)*, pages 334–343. ACM Press, May 1988.

[KNR92] Sampath Kannan, Moni Naor, and Steven Rudich. Implicit representation of graphs. *SIAM Journal on Discrete Mathematics*, 5:596–603, 1992.

[NRTW05] Serguei Norine, Neil Robertson, Robin Thomas, and Paul Wollan. Proper minor-closed families are small. *Journal of Combinatorial Theory, Series B*, 2005. To appear.

[TZ01] Mikkel Thorup and Uri Zwick. Compact routing schemes. In 13^{th} *Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 1–10. ACM Press, July 2001.