

Small Induced-Universal Graphs and Compact Implicit Graph Representations

Stephen Alstrup*

Theis Rauhe*

Abstract

We show that there exists a graph G with $n \cdot 2^{O(\log^* n)}$ nodes, where any forest with n nodes is a node-induced subgraph of G . Furthermore, the result implies existence of a graph with $n^k 2^{O(\log^* n)}$ nodes that contains all n -node graphs of fixed arboricity k as node-induced subgraphs. We provide a lower bound of $\Omega(n^k)$ for the size of such a graph. The upper bound is obtained through a simple labeling scheme for parent queries in rooted trees.

1. Introduction

Two basic ways of representing a graph are adjacency matrices and adjacency lists [32, 26, 54]. Adjacency matrices allow fast adjacency queries, but the space required is super-linear for sparse graphs. Adjacency lists reduce the space needed, but then adjacency queries involve searching in neighbour lists.

There is a number of papers on representing a sparse graph in a compact way while allowing fast adjacency queries, among these [46, 18, 34, 42, 51, 53]. One research direction is the implicit representation of graphs. In an implicit representation of a graph, all nodes are assigned a label. Given the label of two nodes, adjacency between the two nodes can be computed alone from the labels of the two nodes without access to global information. We denote such a labeling of the nodes an adjacency labeling scheme, and the objective is to minimize the maximum number of bits needed to represent a label associated to a node.

For trees we have the following simple algorithm [33, 34]. Root the tree. Prelabel each node with a preorder number. Let the label of each node be its prelabel appended with the prelabel of its parent. A test for adjacency is then simply to test whether the prelabel for one of the nodes equals the stored parent prelabel of the

other node. If the tree has size n , the labels assigned to the nodes have length $2\lceil \log n \rceil$ bits¹. The same labeling scheme can be used for forests. Before labeling the nodes in a forest F , an additional node is added as the parent to the root nodes in F . This will only increase the label length with 1.

Kannan, Naor and Rudich [34], established a connection from adjacency labeling schemes to induced-universal graphs. A graph G is said to be \mathcal{F} -induced-universal if it contains all graphs in \mathcal{F} as node-induced subgraphs. That is, a graph $H = (V', E')$ is contained in $G = (V, E)$ as a node-induced subgraph if $V' \subseteq V$ and $E' = \{(v, w) \mid v, w \in V' \wedge (v, w) \in E\}$. For undefined terminology, the reader is referred to [10]. Let $g_v(\mathcal{F})$ denote the minimum number of nodes in an \mathcal{F} -induced-universal graph. In [34] they show that

Theorem 1 (Kannan, Naor and Rudich [34]) *If a family \mathcal{F} has an adjacency labeling scheme with unique labels of length at most $k \log n$ then $g_v(\mathcal{F}) \leq n^k$.*

Let \mathcal{A} denote the family of forests with n nodes. Now the mentioned $2\lceil \log n \rceil + 1$ result for an adjacency labeling scheme for forest combined with Theorem 1 gives $g_v(\mathcal{A}) = O(n^2)$ [34]. Chung [19, Theorem 5] subsequently shows that $g_v(\mathcal{A}) = O(n \log n)$. She does this by establishing a reduction from universal graphs. We say a graph $H = (V', E')$ is contained in $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. A graph G is said to be \mathcal{F} -universal if G contains all graphs in \mathcal{F} as subgraphs. Let $f_e(\mathcal{F})$ denote the minimum number of edges in an \mathcal{F} -universal graph. In a series of papers [8, 21, 20, 22, 24, 43, 19] it was established that:

$$f_e(\mathcal{A}) = \Theta(n \log n). \quad (1)$$

Let $G = (V, E)$ be a universal graph for a family \mathcal{H} of acyclic graphs. In [19, Theorem 1 and Corollary 1.1.] Chung shows:

$$g_v(\mathcal{H}) \leq 2|E| + |V|. \quad (2)$$

*The IT University of Copenhagen, Glentevej 67, DK-2400 Copenhagen NV, Denmark. E-mail: {stephen,theis}@it-c.dk.

¹ \log is \log_2 , and \log^* is the number of times \log should be iterated to get a constant.

Combining (1) and (2) she concludes $g_v(\mathcal{A}) = O(n \log n)$. This bound can only be improved using techniques different from [19], because of the lower bound in (1). More generally, Chung considers the family \mathcal{A}_k of graphs with arboricity k and n nodes. A graph has arboricity k if the edges from the graph can be partitioned to at most k forests. Chung [19, Theorem 9] shows that $g_v(\mathcal{A}_k) = O((n \log n)^k)$.

Addressing a number of open problems in [34, 19, 20, 21], e.g. the problems stated in [19, emphasized on page 452], and in [34, page 603], we show that for fixed arboricity k :

Theorem 2 *There exists an adjacency labeling scheme for the family \mathcal{A}_k with n nodes, with label length bounded by $k \log n + O(\log^* n)$ bits.*

Theorem 3 *An adjacency labeling scheme for the family \mathcal{A}_k with n nodes, uses labels of length l , $l \geq k \log n$ bits.*

Theorem 4 $g_v(\mathcal{A}_k) \leq n^k 2^{O(\log^* n)}$

Theorem 5 $g_v(\mathcal{A}_k) = \Omega(n^k)$

As a special case of Theorem 4 we have $g_v(\mathcal{A}) \leq n 2^{O(\log^* n)}$, and as a special case of Theorem 2 we have a labeling scheme for planar graphs (which have arboricity 3) using labels of length $3 \log n + O(\log^* n)$. Perhaps, of independent interest, we show that if the nodes have already been assigned a label as the preorder number, we can extend the label with additional $O(\log \log n)$ bits such that adjacency can be tested alone from the labels.

To the best of our knowledge, Theorem 5 is the first lower bound given for this problem. Theorem 3 follows by combining Theorem 1 with Theorem 5. Theorem 4 follows by combining Theorem 2 with Theorem 1. Theorem 2 follows directly from Theorem 7 which is stated and proved in Section 4. In Theorem 7 we give a labeling scheme that can be used to determine if one node is a parent to another node in a tree, and as shown above, it can be used to test for adjacency in forests. For a graph G with arboricity k we can partition the edges to k forests, and finally use Theorem 7 to label each forest.

If we restrict the family of graphs to forests with bounded degree, denoted \mathcal{A}_B , there is a universal graph with n nodes and $O(n)$ edges for this family [11, 12]. Combining this result with (2), Chung [19, Theorem 4] establishes that $g_v(\mathcal{A}_B) = O(n)$. Hence, on contrary to induced-universal graphs, there is a provable gap of a $\log n$ factor between universal graphs for the family of trees and the family of bounded degree trees.

Related work In the early 1960s, induced-universal graphs were studied in [45]. Since then many results have been obtained for the related concept of universal graphs, e.g. for families of graphs such as cycles [13], trees [20, 24, 22, 21, 23, 27], bounded degree trees [11, 12, 29], caterpillars [25, 38], graphs of n edges [8], planar graphs [8], bounded degree graphs [11, 12, 2], and graphs with bounded path-width [50]. Induced-universal graphs for the family of all graphs on n nodes were studied in [40], and for trees, planar graphs, and graphs with bounded arboricity in [34, 19], tournaments [9], and in [39] the family of hereditary graphs is studied. The above list of references is not complete.

In [14, 15] an adjacency labeling scheme was given for general graphs. Efficient adjacency labeling schemes were introduced in [33, 41]. Since then, adjacency labeling schemes producing small labels have been given for different kinds of sparse graphs, see e.g. [17, 51, 48, 35, 46, 16]. A book on implicit graph representation can be found in [49], and an extensive survey on labeling schemes in [30].

To test various kinds of relationships, in addition to adjacency, between two nodes alone from the labels, has been studied in a number of papers: Labeling schemes are given for ancestor in [34, 1, 52, 7, 35, 3], sibling and parent in [35], and distances in [31, 36, 44]. Efficient labeling schemes are essential to a number of distributed computation problems, as e.g. routing technology [47, 52]. Recently, papers on labeling schemes with applications to XML-Based Search Engines have appeared, see e.g. [1].

In [35] an adjacency labeling scheme for trees is given, which assign labels of size $\log n + O(\sqrt{\log n})$ bits. The labels assigned in [35] can, beside adjacency, be used to test for other properties as e.g. ancestor relationship. If adjacency labeling schemes are defined without addressing their computability, Chung's [19, Theorem 5] result can be viewed as an adjacency labeling scheme for trees using labels of size $\log n + O(\log \log n)$. In [6] it is shown that a labeling scheme for rooted trees supporting both parent and sibling queries requires labels of size $\log n + \Omega(\log \log n)$.

2. Preliminaries

Let $T = (V, E)$ be a rooted tree with n nodes. We denote the set of nodes and edges in T as $V(T)$ and $E(T)$ respectively. We let $T(u)$ denote the subtree of T rooted at node $u \in V(T)$. If $w \in V(T(u))$ then u is an ancestor to w , and we write $u \prec w$. If $w \in V(T(u)) \setminus \{u\}$ then u is a proper ancestor to w . If u is a (proper) ancestor to w , then w is a (proper) descendant to u . For nodes u and v in T , we denote the path between

u and v including u and v by $u \rightsquigarrow v$, and the *distance* between u and v is the number of edges on this path. For nodes v different from the root, we denote the parent by $p(v)$.

In the rest of this paper we order the trees such that any node's children root subtrees with sizes in non-decreasing order from left to right. Hence the rightmost child w of a node v roots a largest subtree among the children. We say that the edge (v, w) is a *heavy* edge, and the remaining edges (v, u) , u a child of v , are *light*. A node v , where edge $(p(v), v)$ is light, is called an *apex* node. Let $\text{heavy}(v)$ denote the heavy child of internal node v . The *light depth* of a node v is the number of light edges on the path $r \rightsquigarrow v$. Let the family of all trees of size n be denoted \mathcal{T}_n .

Lemma 1 *For any tree $T \in \mathcal{T}_n$ and node $v \in V(T)$, the light depth of v is bounded by $\log n$.*

We let $[k]$ denote the set $\{1, 2, \dots, k\}$. A bitstring of length n is a sequence $a = a_0 a_1 \dots a_{n-1}$, where $a_i \in \{0, 1\}$. For $0 \leq j \leq n-1$, $a_0 \dots a_j$ and $a_j \dots a_{n-1}$ are a *prefix* and a *suffix* of bitstring a respectively. For positive integer k , the *standard binary representation* of k is the unique bitstring $a_0 \dots a_{r-1}$, where $r = \lceil \log(k+1) \rceil$ and $k = \sum_{0 \leq j \leq r-1} a_j 2^{r-j-1}$. For two bitstrings $a = a_0 \dots a_k$ and $b = b_0 \dots b_k$, we denote their *concatenation* $a_0 \dots a_k b_0 \dots b_k$ by $a \circ b$.

A *parent labeling scheme* for trees of size n is a pair (e, d) , where e and d are mappings called the *encoder* and the *decoder* respectively. The encoder e defines a *label assignment* e_T for all trees $T \in \mathcal{T}_n$ which is a mapping of the nodes $V(T)$ into bitstrings called *labels*. The decoder d maps pairs of labels into $\{0, 1\}$, i.e., $d : \{0, 1\}^* \times \{0, 1\}^* \rightarrow \{0, 1\}$. The encoder and the decoder satisfy $\forall T \in \mathcal{T}_n : \forall v, w \in V(T) : d(e_T(w), e_T(v)) = 1$ iff $p(v) = w$. Furthermore if the label assignment e_T is an injective mapping for all tree $T \in \mathcal{T}$, we say that the labeling scheme assigns *unique labels* to the nodes. The labeling schemes we construct in this paper have unique labels. A function $\text{id} : \{0, 1\}^* \rightarrow \{0, 1\}^m$ is a *node identity function* for (e, d) if $\text{id} \circ e_T : V(T) \rightarrow \{0, 1\}^m$ is injective for all trees $T \in \mathcal{T}_n$. When such a labeling id is defined, we call the bitstring $\text{id}(e_T(v))$ for the *node identity label* of v .

A parent labeling scheme has *label length* bounded by m if the maximal length of the labels assigned to any node in any tree $T \in \mathcal{T}_n$ is bounded by m , i.e., $m \geq \max_{T \in \mathcal{T}_n, v \in V(T)} \{|e_T(v)|\}$. Sometimes we restrict this notion to the maximal label length of leaf nodes or internal nodes.

When we speak of the value $d(x, y) \in \{0, 1\}$ we will sometimes identify 0 and 1 with the Boolean values false

and true respectively and vice versa.

Sometimes various computability requirements are imposed on e and d , i.e., in [37] d should be computable in polynomial time, in [1] in constant time on the RAM, and polynomial time Turing machine computable in [34]. Furthermore, in [34] they restrict attention to labeling schemes where any label assignment labels the graph with unique labels, since Theorem 1 needs this property, whereas the time complexity of d is not important. Theorem 1 obtained is as follows: for each of the at most n^k labels, we have a node in the induced-universal graph U for the family \mathcal{F} , and two nodes are adjacent in U iff the corresponding labels l_1, l_2 satisfy $d(l_1, l_2) = 1$.

3. A simple parent labeling scheme based on preorder numbers

In this section we provide a simple construction which leads to a parent labeling scheme that will serve as basis of our main result in next section. This simple labeling scheme will have internal nodes with label length bounded by $\log n + O(\log \log n)$, and leafs with label length bounded by $\log n + O(1)$. Furthermore it will support a node identity function mapping to bitstrings bounded by $\log n + O(1)$. A key ingredient of this labeling scheme is *preorder numbers*. Consider a *preorder depth-first traversal* of a tree T , i.e., start by visiting the root, and then recursively visit the subtrees rooted at the children of the root, in order from left to right. The *preorder number* $\text{pre}(v)$ is the number of nodes visited before v in this traversal, i.e., the root has preorder number 0 and the rightmost leaf has preorder number $n-1$. Define $\text{dist_parent}(v) = \text{pre}(v) - \text{pre}(p(v))$, and for the root r we let $\text{dist_parent}(r) = n$. For internal nodes, we define $\text{dist_heavy}(v) = \text{pre}(\text{heavy}(v)) - \text{pre}(v)$ and leave leafs undefined. Let $\text{lightdepth}(v)$ denote the light depth of v .

Let T be a tree of size n . If we assign labels to the nodes of T which encode both $\text{pre}(v)$ and $\text{dist_parent}(v)$ in the label for node $v \in V(T)$, then there is a simple test for parentship. Node w is parent of v if and only if $\text{pre}(w) = \text{pre}(v) - \text{dist_parent}(v)$. Each label uses $2 \lceil \log n \rceil$ to store the two integer values bounded by n .

The main idea behind our labeling scheme is to store $\text{lightdepth}(v)$, and a cheaper factor 2 approximation of $\text{dist_parent}(v)$ and $\text{dist_heavy}(v)$ in the label of v . That is, we only store the discrete logarithm of $\text{dist_parent}(v)$ and $\text{dist_heavy}(v)$, and hence limit the bit cost of storing these $\log n$ bounded integer values to $O(\log \log n)$ bits in the label of v . Similarly, only $\lceil \log \log n \rceil$ bits are needed to store $\text{lightdepth}(v)$, since it is also bounded by $\log n$ by Lemma 1. We still store

the exact value of $\text{pre}(v)$ using $\lceil \log n \rceil$ bits. Proposition 1 and 2 below give necessary and sufficient conditions for a parentship test based on these values.

Proposition 1 *For any apex node v and internal node w , w is parent of v iff $\text{lightdepth}(v) = \text{lightdepth}(w) + 1$, $\text{pre}(w) < \text{pre}(v)$, $\lfloor \log \text{dist_heavy}(w) \rfloor \geq \lfloor \log \text{dist_parent}(v) \rfloor$ and $\lfloor \log(\text{pre}(v) - \text{pre}(w)) \rfloor = \lfloor \log \text{dist_parent}(v) \rfloor$.*

Proof. For $w = p(v)$ it is straightforward to verify that the conditions are satisfied. For the other direction, assume on the contrary that node $w \neq p(v)$ satisfies the conditions. Since $\text{pre}(w) < \text{pre}(v)$, and $\text{lightdepth}(w) = \text{lightdepth}(v) - 1 = \text{lightdepth}(p(v))$, w cannot be a descendant of $p(v)$, and thus $p(p(v))$ is on the path $w \rightsquigarrow v$. Hence, the nodes visited by the preorder traversal of T from node w to v contain both the nodes visited between w and $\text{heavy}(w)$ and the nodes visited between $p(v)$ and v . That is $\text{pre}(v) - \text{pre}(w) \geq \text{dist_heavy}(w) + \text{dist_parent}(v)$. Using inequalities $\lfloor \log \text{dist_heavy}(w) \rfloor \geq \lfloor \log \text{dist_parent}(v) \rfloor$ and $\lfloor \log(\text{pre}(v) - \text{pre}(w)) \rfloor = \lfloor \log \text{dist_parent}(v) \rfloor$, this leads to the contradiction $\text{pre}(v) - \text{pre}(w) \geq \text{dist_heavy}(w) + \text{dist_parent}(v) \geq 2 \cdot 2^{\lfloor \log \text{dist_parent}(v) \rfloor} = 2 \cdot 2^{\lfloor \log(\text{pre}(v) - \text{pre}(w)) \rfloor} > \text{pre}(v) - \text{pre}(w)$. ■

Proposition 2 *For any non-apex node v , and internal node w , w is parent of v iff $\text{lightdepth}(v) = \text{lightdepth}(w)$, $\text{pre}(w) < \text{pre}(v)$ and $\lfloor \log(\text{pre}(v) - \text{pre}(w)) \rfloor = \lfloor \log \text{dist_heavy}(w) \rfloor = \lfloor \log \text{dist_parent}(v) \rfloor$.*

Proof. Similar to the proof of Proposition 1. ■

Based on the above propositions we can define a parent labeling scheme (e, d) as follows. For $T \in \mathcal{T}_n$, let the label $e_T(v)$, $v \in V(T)$ encode the values $\text{pre}(v)$, $\lfloor \log \text{dist_parent}(v) \rfloor$, and $\lfloor \log \text{dist_heavy}(v) \rfloor$ together with a type bit indicating whether v is apex or not. Clearly, using standard binary representation of these integers, such a label has length bounded by $\log n + 3 \log \log n + O(1)$. Furthermore, it is straightforward to define the decoder such that $d(e_T(w), e_T(v)) = 1$ iff $w = p(v)$ by performing the tests suggested by the conditions of Proposition 1 or 2. In summary, this proves the lemma below.

Lemma 2 *There exists a parent labeling scheme for trees of size n with label length bounded by $\log n + 3 \log \log n + O(1)$.*

Reducing length of labels for leafs We will now describe how we can modify the above labeling scheme such that the length of leaf labels is reduced to $\log n + O(1)$ bits, accepting an additional $\lceil \log \log n \rceil$ bit cost in internal nodes. Reducing the bit costs of labels to near optimal $\log n + O(1)$ plays an essential role in the construction of the main result provided in Section 4.

We begin with a simple extension of the labeling scheme from Lemma 2, where we extend the labels of *internal* nodes v to store also the number $\#\text{leafs}(v)$ of *leaf* children, i.e. $\#\text{leafs}(v) = |\{q \mid q \text{ is a leaf with } p(q) = v\}|$.

Clearly, storing $\#\text{leafs}(v)$ would increase the bit cost with unacceptable $\log n$ bits, but the positive aspect is that it is now easy to limit the number of bits in leafs. That is, we can, based solely on the preorder number of a leaf q , determine whether any node w is parent expressed in the following lemma.

Lemma 3 *For any leaf q and $w \in V(T)$, $w = p(q)$ iff $\text{pre}(w) < \text{pre}(q) \leq \text{pre}(w) + \#\text{leafs}(w)$.*

Proof. Follows by the ordering of children to a node. ■

As mentioned, the problem of this approach is that we use too many bits in the internal nodes v in order to store the exact value of $\#\text{leafs}(v)$. We thus use the previous approach of limiting the number of used bits to $\lceil \log \log n \rceil$ by only storing an approximate value of $\#\text{leafs}(v)$. That is we store the integer $\lfloor \log \#\text{leafs}(v) \rfloor \leq \log n$. Then for leafs q , where $\text{pre}(q) \leq \text{pre}(p(q)) + 2^{\lfloor \log \#\text{leafs}(p(q)) \rfloor}$,

$$\text{pre}(w) < \text{pre}(q) \leq \text{pre}(w) + 2^{\lfloor \log \#\text{leafs}(p(q)) \rfloor} \quad (3)$$

is true iff w is parent of q by Lemma 3. But for leafs q , where $\text{pre}(q) > \text{pre}(p(q)) + 2^{\lfloor \log \#\text{leafs}(p(q)) \rfloor}$, (3) obviously fails for $w = p(q)$. To solve this problem, we use a trick of “virtually moving” the preorder numbers for these leafs into the preorder range that satisfy (3). Formally, for all leafs in T define the value $\text{pre}(q)$ by $\text{pre}(q) = \text{pre}(q)$ if $\text{pre}(q) \leq \text{pre}(p(q)) + 2^{\lfloor \log \#\text{leafs}(p(q)) \rfloor}$ and $\text{pre}(q) = \text{pre}(q) - 2^{\lfloor \log \#\text{leafs}(p(q)) \rfloor}$ otherwise.

With $\text{pre}(q)$ encoded in the labels for leafs instead of $\text{pre}(q)$, we can omit the value $\text{lightdepth}(q)$ and $\lfloor \log \text{dist_parent} \rfloor$ in leafs and still test for $w = p(q)$ for any w as given by:

Lemma 4 *For any leaf q and $w \in V(T)$, $w = p(q)$ iff $\text{pre}(w) < \text{pre}(q) \leq \text{pre}(w) + 2^{\lfloor \log \#\text{leafs}(p(q)) \rfloor}$.*

By the above lemma, we can replace the labels of leafs such that they only store the integer $0 \leq \text{pre}(q) \leq n - 1$, and hence have bit cost bounded by $\log n + O(1)$.

For leafs q , the decoder then answers the parent query according to the above Lemma 4 using the additional information of integer value $\lceil \log \#\text{leafs}(v) \rceil$ in internal nodes.

The main problem left with the above modification of the labeling scheme is that the label of the leafs is not necessarily unique anymore which we need in next section. More precisely, for any pair of leafs $q \neq q'$, $\text{pre}(q) = \text{pre}(q')$ if and only if $|\text{pre}(q) - \text{pre}(q')| = 2^{\lceil \log \#\text{leafs}(w) \rceil}$ for a common parent w . It is straightforward to modify the labeling of each such pair with a single bit which differs for q and q' and hence reestablish uniqueness of the labels.

Furthermore, if we encode $\text{pre}(w)$ or $\text{pre}(v)$ together with the distinguishing bit for leafs in the prefix of the label, then this prefix is unique for each node w and bounded by $\lceil \log n \rceil + 1$. Hence, the function that returns this unique prefix of length $\lceil \log n \rceil + 1$ is a node identity function for the labeling scheme. In summary, we can conclude with the following theorem.

Theorem 6 *There exists a parent labeling scheme for trees of size n with label length of internal nodes bounded by $\log n + 4 \log \log n + O(1)$ and label length of leaf nodes bounded by $\log n + O(1)$. Furthermore, the function $\text{id} : \{0, 1\}^* \rightarrow \{0, 1\}^*$ that returns the prefix of length $\lceil \log n \rceil + 1$ of bitstring x is a node identity function for this labeling scheme.*

4. Main result

This section is devoted a proof of the following main result of this paper.

Theorem 7 *There exists a parent labeling scheme for trees of size n with label length bounded by $\log n + O(\log^* n)$.*

We obtain this main result by refining the labeling schemes from last section, and start by giving and overview of the techniques introduced to do this.

Overview Our general approach for the construction of the compact parent labeling scheme is based on a recursively iterated combination of two given parent labeling schemes, say A and B. The scheme A uses “many” bits in internal nodes, but has near optimal bit cost in leafs. The scheme B has a better worst case label length than A, but it uses more bits in the leafs than A. The combination of A and B balances this to obtain a better scheme than B, and hence we can recursively reapply this procedure for B. The combination of A and B is defined in terms of a so-called cluster partition of a tree.

Consider a tree T of size n . The combined label for a node in T consists, in addition to a constant number of type bits, of the concatenation of the labels of two nodes, one in a certain tree labeled by scheme A and the other in another certain tree labeled by B. These two trees are defined according to the *cluster partition* of T . This is basically a partition of T into small subtrees with size bounded by some threshold, say $O(t)$. Each such subtree, called a micro tree, is uniquely identified with a node in the so-called macro tree of the cluster partition. This macro tree has at most n/t nodes. The macro tree is labeled by scheme A and the micro trees are labeled by scheme B. A label of a node is composed of either

- a) a label from an internal node in the macro tree or
- b) the label of a leaf node in the macro tree concatenated with the label of a node in a micro tree or
- c) the node identity label for a node in the macro tree concatenated with the label from a node in a micro tree.

With this, the decoder can test for the parent query. In very general terms, this test is divided into two cases. Both nodes may belong to the same micro tree, which we can test for using the part of the labels storing the node identity for the associated node in the macro tree. If these parts equal, the parent test reduces to parentship in the common micro tree, for which both nodes in this case have a label part from. If the two nodes are not in the same micro tree, the parentship in a certain case reduces to parentship of the corresponding macro nodes in the macro tree. If this is the case, both nodes have a label part being the full labels of the corresponding macro tree labels, which then suffices for the parentship on this macro tree.

Analysing the label length, assume that we use Theorem 6 for scheme A and any parent labeling scheme for B with label length by $\log n + f(n)$ for some function $f(n)$. With threshold $t \geq \log^5 n$, the label of the macro tree is bounded by $\log(n/t) + 4 \log \log n + O(1) \leq \log n$. Furthermore, the length of the node identities and leaf labels is bounded by $\log n/t + O(1) \leq \log n - \log t + O(1)$. The labels of the micro trees are bounded by $\log t + O(1) + f(t)$. Hence in case a) the labels are bounded by $\log n$ and in case b) and c) they are bounded by $(\log n - \log t + O(1)) + (\log t + f(t) + O(1)) \leq \log n + f(\log^5 n) + c$ for some constant c .

Hence, if $\lceil f(\log^5 n) + c \rceil < f(n)$, this combined scheme has shorter labels than scheme B. Hence, we can recursively reapply the procedure until $f(n) \leq \lceil f(\log^5 n) + c \rceil$. This recurrence has a solution with $f(n) = O(\log^* n)$, implying our result.

The rest of this section is devoted to a formal proof of the following lemma that formalises this overview. Theorem 7 is a corollary of this lemma.

Lemma 5 *Suppose there exists a parent labeling scheme for trees of size n with label size bounded by $\log n + f(n)$ for a monotonically increasing function $f \in O(\log^5 n)$. Then there exists a parent labeling scheme with labels of size $\log n + f(\log^5 n) + O(1)$.*

Proof of Theorem 7. By Lemma 2 there exists a parent labeling for trees of size n with the labels bounded by $\log n + f(n)$ for $f(n) = 3 \log \log n + O(1)$. Applying Lemma 5 with this labeling and function $f(n)$ shows that we can obtain a labeling scheme with labels bounded by $\log n + f'(n)$ where $f'(n) = f(\log^5 n) + O(1)$. We can repeat this process setting $f(n) = f'(n)$ and reapply Lemma 5 and obtain improved label bounds until $f'(\log^5 n) + O(1) \geq f'(n)$ in which case $f'(n) = O(\log^* n)$ which concludes the proof. ■

Clustering a tree Let T be a tree of size $n = |V(T)| > 1$. For a connected subgraph C of T , we call a node in $V(C)$ incident with a node in $V(T) \setminus V(C)$ a *boundary node*. The boundary nodes of C are denoted as δC . A *cluster* is a connected subgraph of T where $|\delta C| \leq 2$. A set of clusters CS is a *cluster partition* of a tree T with root r iff $V(T) = \cup_{C \in CS} V(C)$, $E(T) = \cup_{C \in CS} E(C)$, and for any $C_1, C_2 \in CS$, $E(C_1) \cap E(C_2) = \emptyset$, $|E(C_1)| \geq 1$, $r \in \delta C$ if $r \in V(C)$. From [5, 4, 28] it follows that:

Lemma 6 *Given a rooted tree T , $n > 1$, and a parameter $1 \leq x \leq n$, there exists a cluster partition CS where $|CS| \leq n/x$, and for a fixed constant c , $|V(C)| \leq cx$ for all $C \in CS$.*

To each cluster $C \in CS$ with only one boundary node, we associate the rightmost leaf in C as the second boundary node. For two boundary nodes u, v in a cluster, where $\text{depth}(u) < \text{depth}(v)$, we use the notation $C(u, v)$ to denote this unique cluster. We call u and v the *upper* and the *lower* boundary node of this cluster $C(u, v)$ respectively.

A cluster partition has a *macro tree* T_m defined as follows. The nodes $V(T_m)$ are the set of boundary nodes of cluster partitions CS , including the additional ones defined above. The edges $E(T_m)$ are the set of pairs (u, v) , where u, v belongs to the same cluster $C(u, v)$. We root T_m at the root r from T which by definition is a boundary node and hence also a node of T_m . Clusters $C(u, v)$ where v is a leaf in T_m are called *leaf clusters*.

To simplify some of the case analyse later, and for some other technical reasons, we impose a restriction

on the cluster partitions that we consider in the remaining of the paper. We say a cluster partition is a *single child cluster partition* if the following holds. Any *non-leaf* cluster $C(u, v)$, either contains at most two nodes or $u \rightsquigarrow v$ contains at least 5 nodes. Furthermore, v has no children in $C(u, v)$, and u has only one, i.e., the node on the path to v .

Lemma 7 *Given a rooted tree T , $n > 1$, and a parameter $1 \leq x \leq n$, there exists a cluster partition CS satisfying the single child property, where $|CS| \leq n/x$, and for a fixed constant c , $|V(C)| \leq cx$ for all $C \in CS$.*

Proof omitted.

For a single child cluster partition, we call non-leaf clusters with only two nodes for *small internal clusters* and the other internal clusters are called *big internal clusters*.

The labeling scheme Let $n \geq 2$ be an integer and fix $t = \lfloor \log^5 n \rfloor$. We will define a parent labeling scheme (e, d) for trees of size n , where the label size is bounded by $\log n + f(t) + O(1)$. As described in the overview, (e, d) will consist of the combination of two schemes A and B. The scheme B is the assumed scheme from the premises of the lemma, and we denote it by (e', d') for trees of size $n' \leq n$, where the assumed label length is bounded by $\log n' + f(n')$. The scheme A will as explained in the overview be the preorder based labeling scheme from Theorem 6, and will be denoted as (e^p, d^p) for trees of size n'' . The parameters n' and n'' will be defined later.

The encoder e Let $T \in \mathcal{T}_n$ be any rooted tree of size n and let r denote its root. In what follows we will define the label assignment e_T , and denote the label assigned a node $v \in V(T)$ by $l(v) = e_T(v)$. Let CS be a *single child* cluster partition of T with at most n/t clusters, where each simple cluster has size bounded by $c_1 t$ for a constant c_1 as provided in Lemma 7. Let T_m denote the corresponding macro tree for this cluster partition CS . Next, let l_m denote the preorder based label assignment of this macro tree, i.e. $l_m(v) = e_{T_m}^p(v)$ for $v \in V(T_m)$. Furthermore, for the node identity function id^p for (e^p, d^p) , we let the node identity label $\text{id}(e_{T_m}^p(v))$ for a node $v \in V(T_m)$ be denoted by $\text{id}_m(v)$.

Since $|T_m| \leq n/t$, all internal nodes v in T_m satisfy $|l_m(v)| \leq \log(n/t) + 4 \log \log(n/t) + O(1)$ by Theorem 6 and leafs q in T_m satisfy $|l_m(q)| \leq \log n - \log t + O(1)$. Furthermore, the node identity labels (the prefix of $l_m(v)$ returned by id^p) are bounded by $\lceil \log n/t \rceil + 1$. As explained in the overview, the label we define for $w \in V(T)$ will have bitstrings $l_m(v)$ or $\text{id}_m(v)$ as components for some $v \in V(T_m)$. A technical point, when

defining such a label, is to enable the decoder to compute and isolate the exact location of these labels parts. To this end we need to know the exact length of $\text{id}_m(v)$ or $l_m(v)$, and these must be fixed relative to n and independent of T . Since the size of the macro tree T_m may depend on the topology of T , we need to be a little careful, since the length of $\text{id}_m(v)$ and $l_m(v)$ is expressed in terms of $|T_m|$ and not n . Instead we modify $l_m(v)$ and $\text{id}_m(v)$ such that their lengths are fixed to the size of the upper bounds given above. That is, we assume $|l_m(v)| = \lceil \log(n/t) + 4 \log \log(n/t) \rceil + c_2$ and $|\text{id}_m(v)| = \lceil \log n/t \rceil + c_3$ for some constants c_2 and c_3 for any $v \in V(T_m)$ and tree T . It is straightforward to ensure this independence of $|T_m|$, since we can pad each label with the prefix $0^k 1$, where k is chosen such that the fixed length is reached. The decoder d^p is then modified such that the initial prefix $0^* 1$ is removed before it makes the original decoding on the remaining suffix.

Next we will define the label for each node w in T by considering a number of possible cases for w . For a node which is not a boundary node, it is either contained in a unique big internal cluster or in a unique leaf cluster. With the exception of the root, if w is a boundary node, it is a *lower boundary node* for a *unique* cluster. We distinguish between cases for w according to the type of this unique cluster.

We start by defining $l(w)$ for each node w that is contained in a *big internal cluster*. Let $w \neq r$ be such node, and let $C(u, v)$ be the *unique* big internal cluster containing w where $u \neq w$. Let u' be the unique child of u in $C(u, v)$ and let v' be the parent of v . Note that u' and v' are distinct, non-adjacent nodes by the assumption of cluster $C(u, v)$ to be a big internal cluster. Let $T(u, v)$ denote the subtree of T consisting of the nodes in $C(u, v)$ excluding u and v . That is $T(u, v)$ has u' as root. Note that $|T(u, v)| = |C(u, v)| - 2 \leq c_1 t$. Let $l_{uv} = e'(T(u, v))$ be the label function associated this tree by the assumed labeling scheme (e', d') for trees of size less than or equal to $c_1 t$. Hence,

$$\begin{aligned} |l_{uv}(w)| &\leq \log(c_1 t) + f(c_1 t) \\ &\leq \log t + f(t) + O(1). \end{aligned} \quad (4)$$

The label $l(w)$ is defined according to the following possible cases for node w . Each label has the last four bits reserved to indicate its *type* encoded by different bitstrings we denote by $\text{child_of_upper}, \text{in_between}, \dots \in \{0, 1\}^4$.

I.1 $w = u'$: Then $l(w) := l_m(v) \circ \text{child_of_upper}$.

This label has size bounded by

$$\begin{aligned} |l(w)| &= |l_m(v)| + O(1) \\ &\leq \log(n/t) + 4 \log \log(n/t) + O(1) \\ &\leq \log n + O(1). \end{aligned}$$

I.2 w has u' as parent: Then

$$l(w) := \text{id}_m(v) \circ l_{uv}(w) \circ \text{grand_child_of_upper}.$$

This label has size bounded by

$$\begin{aligned} |l(w)| &= |\text{id}_m(v)| + |l_{uv}(w)| + O(1) \\ &\leq (\lceil \log n/t \rceil + O(1)) + \log t + f(t) + O(1) \\ &\leq \log n + f(t) + O(1). \end{aligned}$$

I.3 $w = v'$: Then

$$l(w) := \text{id}_m(v) \circ l_{uv}(w) \circ \text{parent_of_lower}.$$

The label size is as above bounded by $\log n + f(t) + O(1)$.

I.4 $w = v$: Then

$$l(w) := l_m(v) \circ \text{lower_in_big}.$$

With label size bounded by

$$|l(w)| \leq |l_m(v)| + O(1) \leq \log n + O(1).$$

I.5 w is none of the above. Then

$$l(w) := \text{id}_m(v) \circ l_{uv}(w) \circ \text{in_between}.$$

As in previous cases

$$\begin{aligned} |l(w)| &= |\text{id}_m(v)| + |l_{uv}(w)| + O(1) \\ &\leq \log n + f(t) + O(1). \end{aligned}$$

If w is the lower boundary node in a small internal cluster $C(u, w)$, then we define its label by $l(w) := l_m(w) \circ \text{lower_in_small}$. We refer to this case for w as I.1. If w is the root r of T , we define $l(w) := l_m(r) \circ \text{the_root}$, and refer to this case as R.1. In both of these cases, the label length is bounded $|l_m(r)| + O(1) \leq \log n + O(1)$.

The remaining cases of nodes are nodes located in leaf clusters, and are not upper boundary nodes in these. Let w be such a node, and let $C(u, v)$, $u \neq w$ denote the unique leaf cluster that contains w . Let $T(u, v)$ denote the subtree of T within cluster $C(u, v)$.

Let $l_{uv} = e'(T(u, v))$ be the label function associated this tree by the assumed labeling scheme (e', d') for trees of size less than or equal to $c_1 t$. As in (4) we have $|l_{uv}(w)| \leq \log t + f(t) + O(1)$. The label $l(w)$ is defined according to the following two cases for w .

L.1 w has u as parent. Then

$$l(w) := l_m(v) \circ l_{uv}(w) \circ \text{child_of_upper_in_leaf}.$$

The label length is bounded by

$$\begin{aligned} |l(w)| &\leq |l_m(v)| + |l_{uv}(w)| + O(1) \\ &\leq \log(n/t) + O(1) + \log t + f(t) + O(1) \\ &\leq \log n + f(t) + O(1). \end{aligned}$$

L.2 w does not have u as parent. Then

$l(w) := l_m(v) \circ l_{uv}(w) \circ \text{in_leaf}$. The label length is bounded by

$$|l_m(v)| + |l_{uv}(v)| + O(1) \leq \log n + f(t) + O(1).$$

With the above definitions we have defined the labels of all nodes in T . To summarize, each node is one of the cases I.1-I.5, I'.1, R.1 and L.1-L.2. Furthermore, the labels are unique, *i.e.*, any pair of different nodes have different labels.

The decoder d Before describing the decoder, we provide some general properties and terminology for the labels assigned nodes in T . Let w be any node in T , and let x denote the label provided by the above labeling scheme. The last four bits of x describe which of the above conclusive cases the node belongs to. We denote this type by $\text{typeof}(x)$, *i.e.*, $\text{typeof}(x) \in \{\text{child_of_upper}, \text{grand_child_of_upper}, \text{parent_of_lower}, \text{lower_in_big}, \text{in_between}, \text{lower_in_small}, \text{the_root}, \text{child_of_upper_in_leaf}, \text{in_leaf}\}$ which indicates that w is in case I.1-I.5, I'.1, R.1, L.1 and L.2 respectively.

For each of these cases, the label of the node will consist of one or more of the following parts; the whole label of a node in the macro tree T_m , the unique id of a label from macro tree T_m and the whole label of a labeling of a subtree of T . We refer to these three parts of a label as the *macro part*, the *id part* and the *micro part* of the label respectively. The following lists possible cases of the type of label x , and which of these label parts are defined for each case of label x .

- For $\text{typeof}(x) \in \{\text{child_of_upper}, \text{lower_in_big}, \text{lower_in_small}, \text{the_root}, \text{child_of_upper_in_leaf}, \text{in_leaf}\}$, w is in one of the cases I.1, I.4, I'.1, R.1, L.1 or L.2. In each of these cases, x has a prefix that is the label of a node in T_m . This prefix has fixed length dependent on n but independent of T , and we denote this prefix by $\text{macropart}(x)$. Furthermore, we say that x has a *macro part* if and only if it satisfies one of these cases.
- The label x has a prefix which is the node identity label for a node in T_m . This prefix has fixed length relative to n and we denote this prefix of x by $\text{idpart}(x)$.
- For $\text{typeof}(x) \in \{\text{parent_of_lower}, \text{in_between}, \text{grand_child_of_upper}, \text{in_leaf}, \text{child_of_upper_in_leaf}\}$, w is

one of the cases I.2, I.3, I.5, or L.2. In any of these, the label also contains the label of the subtree of T of fixed length ending before the 4 last type bits. We let $\text{micropart}(x)$ denote this infix for the labels of these cases.

With the above notation for these parts of the label of the nodes in T , the decoder d for the parent labeling scheme is given in Figure 1. Note that the decoder only refers to the macro parts and micro parts for labels x and x' if these parts are defined for x according to the above definition. Correctness of the labeling scheme follows by the following lemma.

Lemma 8 *For all pairs of nodes w and w' in T with labels x and x' respectively, w' is parent of w iff $d(x', x)$ return true.*

Proof omitted.

This then concludes the construction of the parent labeling scheme proving Lemma 5.

```

proc d(x', x)
  case typeof(x) of
    child_of_upper:
      return typeof(x') ∈ {lower_in_big,
        lower_in_small, the_root} ∧
        dp(macropart(x'), macropart(x)) = 1
    grand_child_of_upper:
      return typeof(x') = child_of_upper ∧
        idpart(x') = idpart(x)
    parent_of_lower:
      return typeof(x') = in_between ∧
        idpart(x') = idpart(x) ∧
        d'(micropart(x'), micropart(x)) = 1
    lower_in_big:
      return typeof(x') = parent_of_lower ∧
        idpart(x') = idpart(x)
    in_between:
      return typeof(x') ∈ {in_between,
        grand_child_of_upper,
        parent_of_lower} ∧
        idpart(x') = idpart(x) ∧
        d'(micropart(x'), micropart(x)) = 1
    lower_in_small:
      return typeof(x') ∈ {lower_in_small,
        lower_in_big, the_root} ∧
        dp(macropart(x'), macropart(x)) = 1
    child_of_upper_in_leaf:
      return typeof(x') ∈ {lower_in_small,
        lower_in_big, the_root} ∧
        dp(macropart(x'), macropart(x)) = 1
    in_leaf:
      return typeof(x') ∈ {in_leaf,
        child_of_upper_in_leaf} ∧
        macropart(x') = macropart(x) ∧
        d'(micropart(x'), micropart(x)) = 1
    the_root:
      return false

```

Figure 1. The decoder d

5. A lower bound

Lemma 9 For $k \geq 2$, $|\mathcal{A}_k| \geq \frac{n^{(n-2)(k-1)}}{n2^{k^2(n-1)}}$.

Proof sketch. A labeled tree is a tree where each node is labeled uniquely with an integer from 1 to n . It is well-known that the number of labeled trees is n^{n-2} , see e.g. [55]. For graphs in \mathcal{A}_k we consider the set L of labelings that labels all nodes with 0 except a single node labeled one, and which labels the edges with subsets of $\{0, 1, \dots, k-1\}$. For a graph $G \in \mathcal{A}_k$ there is at most $k(n-1)$ edges, so $|L| \leq n2^{k^2(n-1)}$. The above bound is then obtained by showing there is an injective mapping $\mathcal{G} : \mathcal{L}^{k-1} \rightarrow \mathcal{A}_k \times L$. ■

Theorem 8 For a family \mathcal{F} : $g_v(\mathcal{F}) \geq \frac{n}{e} |\mathcal{F}|^{\frac{1}{n}}$.

Proof. The number of possible node-induced subgraphs of size n in a graph with $N \geq n$ nodes, is clearly bounded by $\binom{N}{n}$. Hence the minimal induced-universal graph for a family \mathcal{F} , $g_v(\mathcal{F})$, satisfy $|\mathcal{F}| \leq \binom{g_v(\mathcal{F})}{n} \leq \left(\frac{eg_v(\mathcal{F})}{n}\right)^n$ implying $g_v(\mathcal{F}) \geq \frac{n}{e} |\mathcal{F}|^{\frac{1}{n}}$ as desired. ■

Proof of Theorem 5. Let $k \geq 2$ be fixed, and let N be a sufficiently large constant such that for $n \geq N$ we have $k^2(n-1) + \log n < (n-2)(k-1)(k+3)$. Using Lemma 9 and Theorem 8 we can bound $g_v(\mathcal{A}_k)$ assuming $n \geq N$ as follows,

$$\begin{aligned} g_v(\mathcal{A}_k) &\geq \frac{n}{e} |\mathcal{A}_k|^{\frac{1}{n}} \\ &> \frac{n}{e} \left(\frac{2^{(n-2)(k-1)\log n}}{2^{(n-2)(k-1)(k+3)}} \right)^{\frac{1}{n}} \\ &= \frac{n}{e} \left(\frac{n}{2^{k+5}} \right)^{k-1} = \Omega(n^k). \end{aligned}$$

References

- [1] S. Abiteboul, H. Kaplan, and T. Milo. Compact labeling schemes for ancestor queries. In *Proceedings of the twelfth annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 547–556, 2001.
- [2] N. Alon, M. Capalbo, Y. Kohayakawa, V. Rodl, A. Rucinski, and E. Szemerédi. Universality and tolerance. In *41st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 14–21, 2000.
- [3] S. Alstrup, C. Gavoille, H. Kaplan, and T. Rauhe. Nearest common ancestors: A survey and a new distributed algorithm. In *The Fourteenth Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, 2002.
- [4] S. Alstrup, J. Holm, K. de Lichtenberg, and M. Thorup. Minimizing diameters of dynamic trees. In *Automata, Languages and Programming, 24th International Colloquium (ICALP)*, pages 270–280, 1997.
- [5] S. Alstrup, J. Holm, and M. Thorup. Maintaining median and center in dynamic trees. In *7th Scandinavian Workshop on Algorithm Theory (SWAT)*, pages 46–56, 2000.
- [6] S. Alstrup and T. Rauhe. Labeling scheme lower bounds for ancestor, sibling and connectivity. Technical Report TR-2001-10, IT University of Copenhagen, 2001.
- [7] S. Alstrup and T. Rauhe. Improved labeling schemes for ancestor queries. In *Proceeding of the thirteen annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2002.
- [8] L. Babai, F. R. K. Chung, P. Erdős R. L. Graham, and J. Spencer. On graphs which contain all sparse graphs. *Ann. discrete Math.*, 12:21–26, 1982.
- [9] L. W. Beineke and R. T. Wilson. A survey on recent results on tournaments. *Recent advances in Graph Theory. Proc. Prague Symp.*, 1975.
- [10] C. Berge. *Graphs and Hypergraphs*, chapter 5. North Holland Publ. Comp., Amsterdam, 1973.
- [11] S. N. Bhatt, F. R. K. Chung, F. T. Leighton, and A. L. Rosenberg. Optimal simulations of tree machines. In *27th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 274–282, 1986.
- [12] S. N. Bhatt, F. R. K. Chung, F. T. Leighton, and A. L. Rosenberg. Universal graphs for bounded-degree trees and planar graphs. *SIAM Journal on Discrete Mathematics*, 2(2):145–155, 1989.
- [13] J. A. Bondy. Pancyclic graphs. *J. Combinat. theory B*, 11:80–84, 1971.
- [14] M. A. Breuer. Coding vertexes of a graph. *IEEE Trans. on Information Theory*, IT-12:148–153, 1966.
- [15] M. A. Breuer and J. Folkman. An unexpected result on coding vertices of a graph. *J. of Mathematical analysis and applications*, 20:583–600, 1967.
- [16] G. S. Brodal and R. Fagerberg. Dynamic representations of sparse graphs. In *Proc. 6th International Workshop on Algorithms and Data Structures (WADS)*, pages 342–351. 1999.
- [17] M. Chrobak and D. Eppstein. Planar orientations with low out-degree and compaction of adjacency matrices. *Theoretical Computer Science*, 86:243–266, 1991.
- [18] R. C. Chuang, A. Garg, X. He, M. Kao, and H. Lu. Compact encoding of planar graphs via canonical orderings and multiple parentheses. In *International Colloquium on Automata, Languages and programming (ICALP)*, 1998.
- [19] F. R. K. Chung. Universal graphs and induced-universal graphs. *Journal of Graph Theory*, 14(4):443–454, 1990.
- [20] F. R. K. Chung and R. L. Graham. On graphs which contain all small trees. *Journal of combinatorial theory, Series B*, 24(1):14–23, 1978.

- [21] F. R. K. Chung and R. L. Graham. On universal graphs. *Ann. Acad. Sci.*, 319:136–140, 1979.
- [22] F. R. K. Chung and R. L. Graham. On universal graphs for spanning trees. *J. London Math. Soc.*, 27:203–211, 1983.
- [23] F. R. K. Chung, R. L. Graham, and D. Coppersmith. On trees which contain all small trees. In *The theory and applications of graphs*, pages 265–272. John Wiley and Sons, New York, 1981.
- [24] F. R. K. Chung, R. L. Graham, and N. Pippenger. On graphs which contain all small trees ii. *Colloquia Mathematica*, pages 213–223, 1976.
- [25] F. R. K. Chung, R. L. Graham, and J. Shearer. Universal caterpillars. *J. Combinat. Theory B*, 31:348–355, 1981.
- [26] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to algorithms*. MIT Press, Cambridge, Mass., 1990.
- [27] P. C. Fishburn. Minimum graphs that contain all small trees. *Ars Combinat.*, 25:133–165, 1985.
- [28] G.N. Frederickson. Ambivalent data structures for dynamic 2–edge–connectivity and k smallest spanning tree. *SIAM J. Computing*, 26(2):484–538, 1997.
- [29] J. Friedman and N. Pippenger. Expanding graphs contain all small trees. *Combinatorica*, 7:71–76, 1987.
- [30] C. Gavoille and D. Peleg. Compact and localized distributed data structures. Technical Report RR-1261-01, Lab. Bordelais de Recherche en Informatique, 2001.
- [31] C. Gavoille, D. Peleg, S. Perennes, and R. Raz. Distance labeling in graphs. In *Proceeding of the 12th annual ACM-SIAM Symposium On Discrete Algorithms (SODA)*, 2001.
- [32] John E. Hopcroft and Robert E. Tarjan. Efficient algorithms for graph manipulation. *Communications of the ACM*, 16(6):372–378, 1973.
- [33] S. Kannan, M. Naor, and S. Rudich. Implicit representation of graphs. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing (STOC)*, pages 334–343, 1988.
- [34] S. Kannan, M. Naor, and S. Rudich. Implicit representation of graphs. *SIAM J. Disc. Math.*, 1992.
- [35] H. Kaplan and T. Milo. Short and simple labels for small distances and other functions. In *7nd Work. on Algo. and Data Struc. (WADS)*, pages 32–40, 2001.
- [36] M. Katz, N. Katz, and D. Peleg. Distance labeling schemes for well-separated graph classes. In *17th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*. Springer Verlag, 2000.
- [37] M. Katz, N. A. Katz, A. Korman, and D. Peleg. Labeling schemes for flow and connectivity. In *Proceedings of the thirteen annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2002.
- [38] R. J. Kimble and A. J. Scwenk. On universal caterpillars. In *The theory and applications of graphs*, pages 437–447. John Wiley and Sons, New York, 1981.
- [39] V. V. Lozin. On minimal universal graphs for hereditary classes. *Discrete Math. Appl.*, 7(3):295–304, 1997.
- [40] J. W. Moon. On minimal n-universal graphs. In *Proc. Galasgow Math. Soc.*, volume 7, pages 32–33, 1965.
- [41] J. H. Muller. *Local Structure in Graphs*. PhD thesis, School of Information and Computer Science, Georgia Inst. of Technology, 1988.
- [42] J. I. Munro and V. Raman. Succinct representation of balanced parentheses, static trees and planar graphs. In *38th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 118–126, 1997.
- [43] L. Nebesky. On tree-complete graphs. *Casopis Pest. Mat.*, 100:334–338, 1975.
- [44] D. Peleg. Proximity-preserving labeling schemes and their applications. In *Graph-Theoretic concepts in computer science, 25th international workshop (WG)*, pages 30–41, 1999.
- [45] R. Rado. Universal graphs and universal functions. *Acta Arith.*, pages 331–340, 1964.
- [46] C. D. Zaroliagis S. R. Arikati, A. Maheshwari. Efficient computation of implicit representations of sparse graphs. *Discrete Applied Mathematics*, 78:1–16, 1997.
- [47] N. Santoro and R. Khatib. Labeling and implicit routing in networks. *The computer J.*, 28:5–8, 1985.
- [48] E. R. Scheinerman. Local representations using very short labels. *Discrete Mathematics*, 203:287–290, 1999.
- [49] J. Spinrad. *Implicit graph representation*. draft version, 1997. <http://www.vuse.vanderbilt.edu/spin/book.ps>.
- [50] A. Takahashi and S. Ueno Y. Kajitani. Universal graphs for graphs with bounded path-width. *IEICE Trans. Fundamentals*, E78–A(4):458–462, 1995.
- [51] M. Talamo and P. Vocca. Compact implicit representation of graphs. In *Graph-Theoretic concepts in Computer Science, 24th international workshop (WG)*, pages 164–176, 1998.
- [52] M. Thorup and U. Zwick. Compact routing schemes. In *The Thirteen Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, 2001.
- [53] G. Turan. Succinct representation of graphs. *Discrete Applied Math.*, 8:289–294, 1984.
- [54] J. van Leeuwen. *Handbook of Theoretical Computer Science*, volume vol. A: Algorithms and Complexity, chapter Graph algorithms, pages 525–631. North-Holland Publ. Comp., 1990.
- [55] D. B. West. *Introduction to Graph Theory*. Prentice Hall, 2001.