A note on models for graph representations

Amos Korman, Shay Kutten

PII: S0304-3975(08)00787-1
DOI: 10.1016/j.tcs.2008.10.036
Reference: TCS 7198

To appear in: *Theoretical Computer Science*

Please cite this article as: A. Korman, S. Kutten, A note on models for graph representations, *Theoretical Computer Science* (2008), doi:10.1016/j.tcs.2008.10.036

A Note on Models for Graph Representations*

Amos Korman † Shay Kutten ‡

Abstract

This paper is intended more to ask questions than to give answers. Specifically, we consider models for labeling schemes, and discuss issues regarding the number of labels consulted vs. the sizes of the labels.

Recently, quite a few papers studied methods for representing network properties by assigning *informative labels* to the vertices of a network. Consider some graph function f on pairs of vertices (for example, f can be the distance function). In an *f-labeling scheme*, the labels are constructed in such a way so that given the labels of any two vertices u and v, one can compute the function $f(u, v)$ (e.g. the graph distance between u and v) just by looking at these two labels. Some very involved lower bounds for the sizes of the labels were proven. Also, some highly sophisticated labeling schemes were developed to ensure short labels.

In this paper, we demonstrate that such lower bounds are very sensitive to the number of vertices consulted. That is, we show several constructions of such labeling schemes that beat the lower bounds by large margins. Moreover, as opposed to the strong technical skills that were needed to develop the traditional labeling schemes, most of our schemes are almost trivial. The catch is that in our model, one needs to consult the labels of *three* vertices instead of two. That is, a query about vertices u and v can access also the label of some third vertex w (w is determined by the labels of u and v). More generally, we address the model in which a query about vertices u and v can access also the labels of c other vertices. We term our generalized model *labeling schemes with queries*.

The main importance of this model is theoretical. Specifically, this paper may serve as a first step towards investigating different tradeoffs between the amount of labels consulted and the amount of information stored at each vertex. As we show, if all vertices can be consulted then the problem almost reduces to the corresponding sequential problem. On the other hand, consulting just the labels of u and v (or even just the label of u) reduces the problem to a purely distributed one. Therefore, in a sense, our model spans a range of intermediate notions between the sequential and the distributed settings.

In addition to the theoretical interest, we also show cases that schemes constructed for our model can be translated to the traditional model or to the sequential model, thus, simplifying the construction for those models as well. For implementing query labeling schemes in a distributed environment directly, we point at a potential usage for some new paradigms that became common recently, such as P2P and overlay networks.

*A preliminary version of this paper was presented in SIROCCO'2007

†Information Systems Group, IE&M, The Technion, Haifa, 32000 Israel. E-mail: pandit@tx.technion.ac.il. Supported in part at the Technion by an Aly Kaufman fellowship.

‡Contact person. Information Systems Group, Faculty of IE&M, The Technion, Haifa, 32000 Israel. E-mail: kutten@ie.technion.ac.il. Fax: +972 (4) 829 5688. Phone: +972 (4) 829 4505. Supported in part by a grant from the Israel Science Foundation.

1 Introduction

Background: Network representations play a major role in many domains of computer science, ranging from data structures, graph algorithms, and combinatorial optimization to databases, distributed computing, and communication networks. In many traditional network representations, the names or identifiers given to the vertices betray no useful information, and they serve only as pointers to entries in the data structure, which form a *global* representation of the network. In some cases, the identifiers do convey additional information, for example, one can often deduce the geographical location of phone subscribers from their phone numbers; sometimes, one can deduce the organization to which a vertex belongs from the vertex's IP number[1]. Recently, quite a few papers studied methods for representing network properties by assigning *informative labels* to the vertices of the network (see e.g., [5, 7, 16, 18, 24]).

Informally, *informative labeling schemes* are schemes in which every vertex is assigned some label that stores some information. Later, the labels of some vertices are used to compute some function f of these vertices (e.g. the graph distance between two such vertices). The idea is to compute the function $f(u, v)$ without consulting the rest of the graph (e.g., in the distance case, there exists an algorithm that gets as an input only the labels of u and v and outputs the distance of u and v). Some very involved lower bounds for the sizes of the labels were proved. (Of course, it is trivial to compute any graph function using labels that are large enough, for example, if every label includes the description of the whole graph it is easy to compute from even one label the distance of every two vertices). Also, some highly sophisticated labeling schemes were developed to ensure small labels.

In this paper, we introduce a natural generalization of the above model. The idea is to distribute the global information (relevant to f) to the vertices, in such a way that $f(u, v)$ can be inferred by inspecting not only the labels of u and v but possibly the labels of c other vertices. That is, given the labels of u and v, we first find c other vertices and then consult their labels to derive $f(u, v)$. We term our generalized model *c-query labeling schemes*. As in the traditional labeling schemes model, we evaluate c-query labeling schemes by the maximum number of bits stored in a label.

Motivations: The main point of this model is theoretical. In particular, this paper may serve as a first step towards investigating different tradeoffs between the number c of labels consulted and the amount of information stored at each vertex. As we show later, if all the vertices can be consulted then the problem reduces to a sequential (non-distributed) one. On the other hand, consulting just the labels of u and v (or even just the label of u) yields a distributed algorithm. Therefore, in a sense, our model spans a range of intermediate notions between the sequential and the distributed settings, one for each value of c.

We note that most of the constructions given in this paper, calculate $f(u, v)$ by inspecting the labels of u and v, and only one additional vertex w (i.e., 1-query labeling schemes). As we show, increasing the storage accessed by the query by a constant factor (that is, by vertex w above) turns out to reduce the label size significantly. As opposed to the strong technical skills that were needed to

[1] We use the term *vertex* to represent a network node, for consistency with the rest of the paper where we use graph theoretic terms.

develop the traditional labeling schemes, most of our schemes are almost trivial. Still, these 1-query labeling schemes beat the lower bounds of the corresponding traditional labeling scheme by large margins. This demonstrates that the very involved lower bounds in the traditional setting of labeling schemes, are very sensitive to the number of vertices consulted.

The fact that constructing query labeling schemes is sometimes easier, can help to identify the source of the difficulty in constructing labels in the traditional model (with no queries). Moreover, the simplicity of the design of labeling schemes with queries can sometimes help to construct labeling schemes for the traditional model. This can be done in two steps: first design the simple scheme with queries, and then "simulate" this scheme in the traditional model. This method is demonstrated in this paper, by considering the routing function in trees. (In our example, the "simulation" had some associated cost, which made the resulting scheme work for an approximation routing scheme, rather than for the exact routing scheme we would have liked).

The field of informative labeling schemes has applications also in the sequential setting (e.g., [7, 24]). That is, the short labels are used in order to save on the memory space and the time used for representing the graph and answering different inquiries regarding it. Labeling schemes with queries can be used for the same purpose too. Note, that in the sequential setting, if the labels of u and v dictate that the label of w must be consulted, then it is easy to access w. Therefore, the notion of labeling schemes with queries may be found particularly useful for the sequential setting.

In a distributed implementation, however, one needs to address the question of how to access the labels of the additional consulted vertices (e.g., of w mentioned above). We point at some feasible directions for doing this in certain cases. For example, in the case of the routing scheme we use, each consulted vertex w is a neighbor of the consulted vertex u, and therefore, w can be accessed relatively easy. In fact, it may be easier for u to access w than to access its own secondary memory, see, for example, [25].

Distributed implementations may also be feasible in the context of *overlay networks* (including many *Peer to Peer* networks), see, e.g. [17]. A component of an overlay network, e.g. a link, may be mapped into parts of an underlying network, e.g. into a longer route. For example, in the application layer network of [17], a link is mapped into a TCP connection of the Internet. There, it is often the case that knowing the identifier of a vertex is enough to establish a TCP connection to it, and then, to access it.

Note, that we do not require that u and w interact. All u needs is the label of w, which may be more easily accessible than w itself. For example, the label may be available in some cache. Systems that follow a similar approach are used in practice. For example, in the Domain Name Server system of the Internet, a vertex is given some designer name of an address (e.g., ie.technion.ac.il, is the designer name of the vertex whose fixed name is 132.68.160.4) [11]. A vertex u who wishes to route to ie.technion.ac.il first consults a third vertex w that is one of its domain name servers. Vertex u may then get as an answer the fixed name 132.68.160.4. Alternatively, vertex u may then receive an answer that refers it to a forth vertex w' that is another domain name server. Vertex u can then consult w'.

Various papers suggested generalizations of this DNS approach. For example, there are many discussions of networks in which each vertex belongs to several clusters, each with its cluster leader. Furthermore, the routing schemes do force a vertex to consult its cluster leaders. The query algorithm

in this case, will tell v which cluster leader w to consult. Such schemes are even hierarchical- the cluster head may belong to several super clusters, and so forth (see, e.g. [8]).

Putting this even more generally, it makes sense that there exists a restricted subset of the network vertices such that the query algorithm knows how to consult a vertex in this subset (in the DNS example, the DNS servers were in this set, and in the clustering schemes, the cluster leaders). See an additional discussion of this issue in Section 7.

Related work: In this subsection, we mostly survey results concerning labeling schemes (with no queries). However, let us, first, mention some studies concerning overlay and Peer to Peer networks that may serve as a practical motivation for our work, and for some other studies concerning labeling schemes in a distributed network environment.

In the model for the complexity of algorithms in fast networks of [1], as well in many later studies of overlay networks, it is argued that the main overhead in accessing a remote vertex w is finding w (which can be done in our constructions by u using v's label) and creating the connection to w. Such models are presented explicitly, e.g. in [1, 4, 17, 28], where such remote accesses are used to construct and use overlay data structures. In overlay networks, a vertex w is addressed by its contents [15, 28]. This may motivate common labeling schemes that assume content addressability. It also motivates the model used here that assumes it is easy to access the label of a third vertex w as explained above.

Let $\mathcal{F}$ be some graph family and let f be a function on pairs of vertices in a graph. In an *f-labeling scheme* for the graph family $\mathcal{F}$, it is required to label all vertices in all graphs in $\mathcal{F}$, such that given the labels of two vertices u and v in some graph $G \in \mathcal{F}$, one can deduce $f(u, v)$.

The concept of f-labeling schemes was implicitly introduced in [9] and then formalized in [24, 27]. Labeling schemes supporting the adjacency and ancestry functions on trees were investigated in [6, 7, 24].

Distance labeling schemes were studied in [16, 26, 30]. In particular, [26] showed that the family of n vertex weighted trees with integer edge capacity of at most W, enjoys a scheme using $O(\log^2 n + \log n \log W)$-bit labels. This bound was proven in [16] to be asymptotically optimal.

Labeling schemes for routing on trees were investigated in a number of papers (e.g., [12, 29, 32]), until finally optimized in [13, 14, 31]. For the *designer-port* model, in which the designer of the scheme can enumerate the port numbers of the vertices freely, [13] shows how to construct a routing scheme using labels of $O(\log n)$ bits on n-vertex trees. In the *fixed-port* model, in which the port numbers are fixed by an adversary, they show how to construct a routing scheme using labels of $O(\log^2 n / \log \log n)$ bits on n-vertex trees. In [14], they show that both of these label sizes are asymptotically optimal. Independently, a routing scheme for trees using $(1 + o(1)) \log n$-bit labels was introduced in [31] for the designer port model.

Two variants of labeling schemes supporting the nearest common ancestor (NCA) function in trees, appear in the literature. In an id-NCA labeling scheme, the vertices of the input graph are assumed to have disjoint identifiers (using $O(\log n)$ bits) given by an adversary. The goal of an id-NCA labeling scheme is to label the vertices such that given the labels of any two vertices u and v, one can find the identifier of the NCA of u and v. Static labeling schemes on trees supporting the separation level and

id-NCA functions were given in [27] using $\Theta(\log^2 n)$-bit labels. The second variant considered is the label-NCA labeling scheme, whose goal is to label the vertices such that given the labels of any two vertices u and v, one can find the label (and not the pre-given identifier) of the NCA of u and v. In [5], they present a label-NCA labeling scheme on trees enjoying $\Theta(\log n)$-bit labels.

In [18], they give a labeling scheme supporting the flow function on n-vertex general graphs using $\Theta(\log^2 n + \log n \log W)$-bit labels, where W is the maximum capacity of an edge. In the restricted case of weighted trees, [21] gives a labeling scheme supporting the flow function with label size $\Theta(\log n \log W)$. In [18], the authors also show a labeling scheme supporting the k-vertex-connectivity function on general graphs using $O(2^k \log n)$-bit labels. This upper bound on the label size was recently improved in [20] to $k^2 \log n$.

Most of the research concerning labeling schemes in the dynamic settings, considered the following two dynamic models on tree topologies. In the *leaf-dynamic* tree model, the topological event that may occur is that a leaf is either added to or removed from the tree. In the *leaf-incremental* tree model, the only topological event that may occur is that a leaf joins the tree.

The study of dynamic distributed labeling schemes was initiated in [23, 22]. In [23], a dynamic labeling scheme is presented for distances in the leaf-dynamic tree model, with $O(\log^2 n)$ label size and $O(\log^2 n)$ amortized message complexity (the number of messages, divided by the number of changes in the network), where n is the current tree size. β-approximate distance labeling schemes (in which, given two labels, one can infer a β-approximation to the distance between the corresponding vertices) are presented [22]. Their schemes apply for dynamic models in which the tree topology is fixed but the edge weights may change.

Two general translation methods for extending static labeling schemes on trees to the dynamic setting are considered in the literature. Both approaches fit a number of natural functions on trees, such as ancestry, routing, label-NCA, id-NCA etc. Given a static labeling scheme on trees, in the leaf-incremental tree model, the resulting dynamic scheme in [23], incurs multiplicative overheads of $O(\log n)$ in both the label size and the communication complexity. Moreover, if an upper bound n_f on the final number of vertices in the tree is known in advance, the resulting dynamic scheme in [23], incurs multiplicative overheads (over the static scheme) of $O(\log^2 n_f / \log\log n_f)$ in the label size and only $O(\log n / \log\log n)$ in the communication complexity. In the leaf-dynamic tree model, there is an extra additive factor of $O(\log^2 n)$ to the amortized message complexity of the resulted schemes.

In [19], it is shown how to construct for any "reasonable" function $k(x)$, a dynamic labeling scheme in the leaf-incremental tree model, extending a given static scheme, such that the resulting scheme incurs multiplicative overheads (over the static scheme) of $O(\log_{k(n)} n)$ in the label size and $O(k(n) \log_{k(n)} n)$ in the communication complexity. As in [23], in the leaf-dynamic tree model, there is an extra additive factor of $O(\log^2 n)$ to the amortized message complexity of the resulted schemes. In particular, by setting $k(n) = n^\epsilon$, dynamic labeling schemes are obtained with the same asymptotic label size as the corresponding static schemes and sublinear amortized message, namely, $O(n^\epsilon)$.

In [10], a labeling scheme is given for a subgraph distance between two vertices. Given a graph G, labels of the form $L(.)$ are computed, such that given $L(u)$, $L(v)$, and $L(X)$, $X \subset G$, one can compute the distance $d_{G-X}(u, v)$. In a sense, this is an $|X|$-query scheme for that function, in our terminology.

1.1 Our contribution

We introduce the notion of f-labeling schemes with queries that is a natural generalization of the notion of f-labeling schemes.

Using this notion, we demonstrate a source of the difficulty in schemes in the traditional model. That is, we demonstrate that by increasing slightly (from two to three) the number of vertices whose labels are inspected, the size of the labels decreases considerably. As examples, we show that there exist simple labeling schemes with a single *query*, supporting the distance function on n-vertex trees as well as the flow function on n-vertex general graphs, with label size $O(\log n + \log W)$, where W is the maximum (integral) capacity of an edge. Recall, that the lower bound for labeling schemes without queries, for each of these problems, is $\Omega(\log^2 n + \log n \log W)$ [16, 18]. We also show that there exists a labeling scheme with one query supporting the id-NCA function on n-vertex trees with label size $O(\log n)$. The lower bound for schemes without queries is $\Omega(\log^2 n)$ [27]. In addition, we show a routing labeling scheme with one query in the fixed-port model using $O(\log n)$-bit labels, while the lower bound (see [14]) for the case of no queries is $\Omega(\frac{\log^2 n}{\log \log n})$. We note that all the schemes we introduce have asymptotically optimal label size for schemes with one query. The matching lower bound proofs are straightforward in most of the cases, and we present the proofs in the remaining cases.

The above examples concern consulting three vertices instead of two. On the other end of the spectrum, if all the vertices can be consulted, we show that in a sense, the problem reduces to a sequential one. That is, essentially, the scheme is designed so that the whole graph structure can be reconstructed by consulting the labels of all vertices, and then any query is answered in a sequential manner. Specifically, we show that for any graph family $\mathcal{F}$ and any (computable) function f, there exists an $(n-2)$-query f-labeling scheme with $\frac{\log |\mathcal{F}|}{n} + \Theta(\log n)$-bit labels (though the decoder may not be polynomial). On the other hand, for any graph family $\mathcal{F}$ and many functions f (for example, f can be the distance or adjacency functions), any $(n-2)$-query f-labeling scheme must have label size at least $\frac{\log |\mathcal{F}|}{n}$.

Most of the results are obtained by simple constructions, which strengthens the motivation for this model. Somewhat more involved are the dynamic schemes. We use model translation methods based on those of [19, 23]. However, in order to save on the message complexity, we needed to make some adaptations to those methods, as well as to one of the static routing schemes of [13].

As an additional motivation for the model, we show that the study of the queries model can help with the traditional model too. That is, using ideas from our routing labeling scheme with one query, we show how to construct a 3-approximation routing scheme *without queries* for unweighted trees in the fixed-port model with $\Theta(\log n)$-bit labels. Finally, we demonstrate the usefulness of the model for a non-distributed environment by showing that one can preprocess a general weighted graph using almost linear space so that flow queries can be answered in almost constant time.

2 Preliminaries

Let $G = \langle V, E \rangle$ be a graph, where V is the set of vertices in G and E is the set of edges. We assume that each edge $e \in E$ is associated with a positive integer *weight*, denoted by $\omega(e)$. A graph is called *unweighted* if all its edges have weight 1.

Let T be a tree with a designated vertex called the *root*. For a vertex v in T, let $deg(v)$ denote the degree of v. For a non-root vertex $v \in T$, let $p(v)$ denote the parent of v in T. The *depth* of a vertex is defined as its weighted distance to the root. The *nearest common ancestor* of u and w, $NCA(u, w)$, is the common ancestor of both u and w of maximum depth. Let $\mathcal{T}(n)$ denote the family of all n-vertex unweighted trees. Let $\mathcal{T}(n, W)$ (respectively, $\mathcal{G}(n, W)$) denote the family of all n-vertex trees (resp., connected graphs) with (integer) edge weights bounded from above by W.

Incoming and outgoing links from every vertex are identified by, so called, *port-numbers*. When considering routing schemes, we distinguish between the following two variants of port models. In the *designer port* model, the designer of the scheme can assign the port numbers of each vertex freely (as long as these port numbers are unique), and in the *fixed-port* model, the port numbers at each vertex are assigned by an adversary. We assume that each port number is encoded using $O(\log n)$ bits.

2.1 The functions

We consider the following functions which are applied on pairs of vertices u and v in a graph $G = \langle V, E \rangle$. (When the underlying graph is clear from the context we omit the corresponding subscript).

- Flow: denote by G' the multigraph obtained by replacing each edge e in G with $\omega(e)$ parallel edges of capacity 1. A set of paths P in G' is *edge-disjoint* if each edge appears in no more than one path $p \in P$. Let $\mathcal{P}_{u,v}$ be the collection of all sets P of edge-disjoint paths in G' between u and v. Then, the maximum flow between u and v is defined as $flow_G(u, v) = \max_{P \in \mathcal{P}_{u,v}}\{|P|\}$, where $|P|$ is the number of paths in P. See [18] for additional details.
- Distance: $d_G(u, v)$ is the weighted distance between u and v in G.
- Routing: function: $rout_G(u, v)$ is the port number at u leading to the next vertex on the shortest path connecting u to v.

If the graph is a tree T then we consider also the following functions:

- Separation level: $Sep - level_T(u, v)$ is the depth of $NCA(u, v)$.
- We distinguish between the following two variants for the NCA function.
  - Id-NCA: assume that identifiers containing $O(\log n)$ bits are assigned to the vertices by an adversary. Then, $Id - NCA_T(u, v)$ is the identifier of $NCA(u, v)$.

– Label-NCA: assuming each vertex can select its own identifier freely (as long as all identifiers remain unique), $Label - NCA_T(u, v)$ is the identifier of $NCA(u, v)$. (In this case, the identifiers may also be referred to as labels.)

2.2 Labeling schemes and c-query labeling schemes

Let f be a function defined on pairs of vertices of a graph. An *f-labeling scheme* $\pi = \langle \mathcal{M}, \mathcal{D} \rangle$ for a family of graphs $\mathcal{F}$ is composed of the following components:

1. A *marker* algorithm $\mathcal{M}$ that given a graph $G \in \mathcal{F}$, assigns a label $\mathcal{M}(v)$ to each vertex $v \in G$.
2. A (polynomial time)*decoder* algorithm $\mathcal{D}$ that given the labels $\mathcal{M}(u)$ and $\mathcal{M}(v)$ of two vertices u and v in some graph $G \in \mathcal{F}$, outputs $f(u, v)$.

The most common measure used to evaluate a labeling scheme $\pi = \langle \mathcal{M}, \mathcal{D} \rangle$, is the *label size*, i.e., the maximum number of bits used in a label $\mathcal{M}(v)$ over all vertices v in all graphs $G \in \mathcal{F}$.

Let c be some non-negative integer. Informally, in contrast to an f-labeling scheme, in a c-query f-labeling scheme, given the labels of two vertices u and v, the decoder may also consult the labels of c other vertices (for some given positive integer c). More formally, a *c-query f-labeling scheme* $\varphi = \langle \mathcal{M}, Q, \mathcal{D} \rangle$ is composed of the following components:

1. A *marker* algorithm $\mathcal{M}$ that, given a graph $G \in \mathcal{F}$, assigns a label $\mathcal{M}(v)$ to each vertex $v \in G$. This label is composed of two sublabels, namely, $\mathcal{M}^{index}(v)$ and $\mathcal{M}^{data}(v)$, where it is required that the index sublabels are unique, i.e., for every two vertices v and u, $\mathcal{M}^{index}(v) \neq \mathcal{M}^{index}(u)$. (In other words, the index sublabels can serve as identifiers.)
2. A (polynomial time) *query* algorithm Q that given the labels $\mathcal{M}(u)$ and $\mathcal{M}(v)$ of two vertices u and v in some graph $G \in \mathcal{F}$, outputs $Q(\mathcal{M}(u), \mathcal{M}(v))$ which is a set containing the indices (i.e., the first sublabels) of at most c vertices in G.
3. A (polynomial time) *decoder* algorithm $\mathcal{D}$ that given the labels $\mathcal{M}(u)$ and $\mathcal{M}(v)$ of two vertices u and v and the labels of the vertices specified by $Q(\mathcal{M}(u), \mathcal{M}(v))$, outputs $f(u, v)$.

As in the case of f-labeling schemes, we evaluate a c-query f-labeling scheme $\varphi = \langle \mathcal{M}, Q, \mathcal{D} \rangle$ by its *label size*, i.e, the maximum number of bits used in a label $\mathcal{M}(v)$ over all vertices v in all graphs $G \in \mathcal{F}$. Note, that since the index sublabels must be disjoint, any c-query f-labeling scheme on any family of n-vertex graphs must have label size $\Omega(\log n)$.

See Section 7 for an alternative definition for query labeling schemes.

2.3 Routing schemes and β-approximation routing schemes

A *routing scheme* is composed of (1) a *marker algorithm* $\mathcal{M}$ for assigning each vertex v of a graph G with a label $\mathcal{M}(v)$, coupled with (2) a *router* algorithm $\mathcal{R}$ whose inputs are the header of a message, $\mathcal{M}(v)$, and the label $\mathcal{M}(y)$ of a destination vertex y. If a vertex x wishes to send a message to vertex

y, it first prepares and attaches a header to the message. Then, the router algorithm x outputs a port of x on which the message is delivered to the next vertex. This is repeated in every vertex, until the message reaches the destination vertex y. Each intermediate vertex u on the route may replace the header of the message with a new header and may perform a local computation. The requirement is that the length of the resulting path connecting x and y is the same as the distance between x and y in G.

For a constant β, a β-approximation routing scheme is the same as a routing scheme except for the following change: The requirement now is that the length of resulting route connecting x and y is a β-approximation of the distance between x and y in G.

In addition to the label size, we also evaluate a routing scheme (and a β-approximation routing scheme) by the *header size*, i.e., the maximum number of bits used in a header of a message.

Note, that in a routing scheme, the router algorithm is the equivalent of the decoder algorithm in general labeling schemes. A c-query routing scheme has a query algorithm too, beside the marker and the router algorithms. Similarly to the case of the general c-query labeling scheme (where the query algorithm generates an input for the decoder algorithm), in a c-query routing scheme, the output of the query algorithm is an additional input for the router algorithm.

In the routing example given in this paper, the routing information of each vertex is distributed among its neighbors. When a vertex, v, wishes to send a message to another vertex, u, its query algorithm tells v which neighbor w to consult. The neighbor is then consulted to find the next vertex on the route to u. In this case, accessing w in order to consult it is easy since w is near to v. As mentioned under "motivation" in the introduction, it would be interesting to find useful schemes where w is far.

3 Labeling schemes with one query

In this section, we demonstrate that the query model allows for significantly shorter labels. In particular, we describe simple 1-query labeling schemes with labels that beat the lower bounds in the following well studied cases: for the family of n-vertex trees, schemes supporting the routing (in the fixed-port model), distance, separation level, and the id-NCA functions; for the family of n-vertex general graphs, a scheme supporting the flow function. We note that all the schemes we present use asymptotically optimal labels.

Most of the 1-query labeling schemes obtained in this section use the label-NCA labeling scheme $\pi_{NCA} = \langle \mathcal{M}_{NCA}, \mathcal{D}_{NCA} \rangle$ described in [5] for the traditional model. There, the marker algorithm $\mathcal{M}_{NCA}$ assigns each vertex v a distinct label $\mathcal{M}_{NCA}(v)$ using $O(\log n)$ bits. Given the labels $\mathcal{M}_{NCA}(v)$ and $\mathcal{M}_{NCA}(u)$ of two vertices v and u in the tree, the decoder $\mathcal{D}_{NCA}$ (of the scheme of [5]) outputs the label of $NCA(u, v)$.

3.1 Id-NCA function in trees

We describe first a trivial 1-query labeling scheme $\varphi_{id-NCA} = \langle \mathcal{M}_{id-NCA}, Q_{id-NCA}, \mathcal{D}_{id-NCA} \rangle$ that demonstrates how easy it is to support the id-NCA function on $\mathcal{T}(n)$ using one query and $O(\log n)$-bit labels. Recall that the lower bound on schemes without queries is $\Omega(\log^2 n)$ [27].

The idea is to use, in the *query model* the known fact that solving label-NCA in the *traditional model* requires only $O(logn)$ bits. We use that to reduce the label size of *id-NCA* in the query model. Informally, the idea behind φ_{id-NCA} is to have the labels of u and v (specifically, their first sublabels) include the labels given by the label-NCA labeling scheme $\pi_{NCA}(v)$ of [5]. Hence, their labels are enough for the query algorithm to find the π_{NCA} label of their nearest common ancestor w using the scheme of [5]. Then, the decoder algorithm (of the new scheme) finds w's identifier simply in the second sublabel of w.

To demonstrate the formalism, we now describe the 1-query labeling scheme φ_{id-NCA} more formally. Given a tree T, recall, that it is assumed that each vertex v is assigned a unique identifier $id(v)$ by an adversary and that each such identifier is composed of $O(\log n)$ bits. The marker algorithm $\mathcal{M}_{id-NCA}$ labels each vertex v with the label $\mathcal{M}_{id-NCA}(v) = \langle \mathcal{M}^{index}(v), \mathcal{M}^{data}(v) \rangle = \langle \mathcal{M}_{NCA}(v), id(v) \rangle$[2]. ($\mathcal{M}_{NCA}$ is computed by using the method of the scheme of [5].) Given the labels $\mathcal{M}_{id-NCA}(v)$ and $\mathcal{M}_{id-NCA}(u)$ of two vertices v and u in the tree, the query algorithm Q_{id-NCA} uses the (scheme of [5]) decoder $\mathcal{D}_{NCA}$ applied on the corresponding first sublabels to output the sublabel $\mathcal{M}^{index}_{id-NCA}(w) = \mathcal{M}_{NCA}(w)$, where w is the NCA of v and u. Given the labels $\mathcal{M}_{id-NCA}(v)$, $\mathcal{M}_{id-NCA}(u)$ and $\mathcal{M}_{id-NCA}(w)$ where w is the NCA of v and u, the decoder $\mathcal{D}_{id-NCA}$ simply outputs the second sublabel of w, i.e., $\mathcal{M}^{data}_{id-NCA}(w) = id(w)$. The fact that φ_{id-NCA} is a correct 1-query labeling scheme for the id-NCA function on $\mathcal{T}(n)$ follows from the correctness of the label-NCA labeling scheme π_{NCA}. Since the label size of $\pi_{NCA}(v)$ is $O(\log n)$ and since the identifier of each vertex v is encoded using $O(\log n)$ bits, we obtain that the label size of φ_{id-NCA} is $O(\log n)$. As mentioned before, since the index sublabels must be disjoint, any query labeling scheme on $\mathcal{T}(n)$ must have label size $\Omega(\log n)$. The following lemma follows.

Lemma 3.1 *The label size of a 1-query id-NCA labeling scheme on $\mathcal{T}(n)$ is $\Theta(\log n)$.*

3.2 Distance and separation level in trees

The above method can be applied for other functions. For example, let us now describe 1-query labeling schemes φ_{dist} and $\varphi_{sep-level}$ supporting the distance and separation level functions respectively on $\mathcal{T}(n, W)$. Both of our schemes have label sizes $\Theta(\log n + \log W)$. Recall, that any labeling scheme (without queries) supporting either the distance function or the separation level function on $\mathcal{T}(n, W)$ must have label size $\Omega(\log^2 n + \log n \log W)$ [16, 27]. We first show the following lemma.

Lemma 3.2 *Let c be a positive integer constant. Any c-query labeling scheme supporting either the separation level function or the distance function on $\mathcal{T}(n, W)$ must have label size $\Omega(\log W + \log n)$.*

[2]To simlify the notations, we omit the $id - NCA$ subscript when it is clear from the context. For example, we use $\mathcal{M}^{index}(v)$ instead of $\mathcal{M}^{index}_{id-NCA}(v)$.

Proof: As mentioned above, any query labeling scheme on $\mathcal{T}(n)$ must have label size $\Omega(\log n)$. We prove the $\Omega(\log W)$ part of the bound for the distance function. The proof for the separation level function is similar.

First note, that we may assume that $W \geq ((c+2)\cdot n^c)^2$, otherwise, the lower bound follows trivially, since $\Omega(\log n) = \Omega(\log n + \log W)$. Let $\varphi = \langle \mathcal{M}, Q, \mathcal{D} \rangle$ be any c-query labeling scheme supporting the distance function. Let P be an n-vertex path rooted at one of its end-vertices r. Let v be the (only) child of r. For every $1 \leq i \leq W$, let P_i be the path P such that the edge (r, v) has weight i and all the other edges have weight 1. For every $1 \leq i \leq W$ and every vertex $u \in P_i$, let $L_i(u)$ denote the label given to u by the marker algorithm $\mathcal{M}$ applied on P_i. For every $1 \leq i \leq W$, let S_i be the set of c vertices given by the query algorithm applied on the labels of r and v in P_i, i.e., $S_i = Q(L_i(r), L_i(v))$. Since there are at most n^c sets of c vertices, there exists a set $X \subset \{1, 2, \cdots W\}$ such that $|X| \geq W/n^c$ and for every $i, j \in X$, $S_i = S_j$. Let S denote the set of c vertices such that for every $i \in X$, $S_i = S$. For each $i \in X$, given the labels $L_i(r)$, $L_i(v)$ and the labels of the vertices in S assigned by $\mathcal{M}$ applied on P_i, the decoder outputs i, which is the distance between r and v in P_i. Since $|X| \geq W/((c+2)\cdot n^c)$, there must exist a vertex in $u \in \{r, v\} \cup S$ such that the set $\{L_i(u) \mid i \in X\}$ contains $\frac{W}{(c+2)\cdot n^c}$ values. Therefore, there must exist an $i \in X$ such that $L_i(u)$ contains $\log W - \log((c+2)\cdot n^c) > \frac{1}{2}\log W$ bits. The lemma follows. ∎

We now show how to construct 1-query labeling schemes supporting the separation level and distance functions using a method that is very similar to the one described in Subsection 3.1. Recall, that in Subsection 3.1, the data sublabel of a vertex contained the vertex's *id*. Instead of that *id*, here, the second sublabel will contain the depth of the vertex in the tree. Recall, that,

$$d(v, u) = depth(v) + depth(u) - 2 \cdot depth(NCA(v, u)). \tag{1}$$

Based on the above equation, the reader can now construct, as an exercise, a 1-query labeling scheme for the distance function. This scheme will use the depth in the second sublabel of the Nearest Common Ancestor (and will find the nearest common ancestor using the first sublabels and the scheme of [5]), in a way that is very similar to the scheme of Subsection 3.1. A similar exercise is constructing a scheme for the separation level function.

Lemma 3.3 *The label size of a 1-query labeling scheme supporting either the separation-level or the distance function on $\mathcal{T}(n, W)$ is $\Theta(\log n + \log W)$.*

3.3 An example in general graphs (Flow)

We now consider the family $\mathcal{G}(n, W)$ of connected n-vertex weighted graphs with maximum edge capacities W, and present a 1-query flow labeling scheme φ_{flow} for this family using $O(\log n + \log W)$-bit labels. Recall, that any labeling scheme (without queries) supporting the flow function on $\mathcal{G}(n, W)$ must have size $\Omega(\log^2 n + \log n \log W)$ [18].

Lemma 3.4 *The label size of a 1-query flow labeling scheme on $\mathcal{G}(n, W)$ is $\Theta(\log n + \log W)$.*

Sketch of the proof: The fact that any 1-query flow labeling scheme on $\mathcal{G}(n, W)$ must have label size $\Omega(\log n + \log W)$ follows using similar arguments as in the proof of Lemma 3.2. We now show how to construct a 1-query flow labeling scheme on $\mathcal{G}(n, W)$ with label size $O(\log n + \log W)$. As shown in [18], given a graph $G \in \mathcal{G}(n, W)$ with vertices $u_1, u_2, \cdots, u_n$, one can construct a weighted tree $T \in \mathcal{T}(m, Wn)$ such that $m = O(n)$, T has n leaves $v_1, v_2, \cdots, v_n$ and $flow_G(u_i, u_j) = sep - level_T(v_i, v_j)$. Therefore, using our 1-query separation-level labeling scheme $\varphi_{sep-level}$ on T, we obtain a 1-query flow labeling scheme φ_{flow} with size $O(\log m + \log Wn) = O(\log n + \log W)$. ∎

3.4 Routing in trees using one query

In this subsection, we demonstrate that the use of queries can decrease the label size for routing schemes too. That is, we demonstrate a 1-query routing labeling scheme φ_{fix} in the fixed-port model, using $O(\log n)$-bit labels. (As mentioned above, any 1-query routing labeling scheme on $\mathcal{T}(n)$ must have label size $\Omega(\log n)$.) The query at a vertex v in this scheme is made to some w that is a neighbor of v (that is, the network graph contains the edge (w, v)). Hence, this scheme even looks feasible for a distributed implementation.

In [13], they give a routing scheme $\pi_{des} = \langle \mathcal{M}_{des}, \mathcal{D}_{des} \rangle$ for the designer port model in $\mathcal{T}(n)$. Given a tree $T \in \mathcal{T}(n)$, for every vertex $v \in T$ and every neighbor u of v, let $port_{des}(v, u)$ denote the port number (assigned by the designer of routing scheme π_{des} above) leading from v to u. In particular, the port number leading from each non-root vertex v to its parent $p(v)$ is assigned the number 1, i.e., $port_{des}(v, p(v)) = 1$. Given the labels $\mathcal{M}_{des}(v)$ and $\mathcal{M}_{des}(u)$ of two vertices v and u in T, the decoder $\mathcal{D}_{des}$ outputs the port number $port_{des}(v, w)$ at v leading from v to the next vertex w on the shortest path connecting v and u.

Let T be an n-vertex tree. We refer to a port number assigned by the designer of the routing scheme π_{des} as a *designer port number* and to a port number assigned by the adversary as a *fixed-port number*. Let *port* be some port of a vertex in the fixed-port model. Besides having a fixed-port number assigned by the adversary, we may also consider *port* as having a designer port number, the number that would have been assigned to it had we been in the designer port model. For a port leading from vertex v to vertex u, let $port_{fix}(v, u)$ denote its fixed-port number and let $port_{des}(v, u)$ denote the designer port number we would have liked it to have.

We now describe a 1-query routing labeling scheme $\varphi_{fix} = \langle \mathcal{M}_{fix}, Q_{fix}, \mathcal{D}_{fix} \rangle$ which operates in the fixed-port model. Informally, we construct a translation from the known scheme of [13] for designer routing, to the desired fixed routing scheme. In this translation, a message forwarding from a vertex to its parent (in the scheme of [13]) is translated to a message forwarding to the parent here too. The harder case is when the next vertex on the route (from v to u) is a child z of v. In that case, the scheme must encode enough information for the router algorithm to be able to choose between the children. The designer scheme of [13] yields a pointer X to some child of v. That is, had the ports of v been numbered by the designer, the desired child z would have been in port number X of v. Unfortunately, in the fixed-port model, the port number of v leading to the desired child z is some Y that our translation needs to find out.

Informally, the marker algorithm prepares a translation table, where each entry gives the fixed

port number Y of some designer port number X. Each entry is given to one child w. To enable the query algorithm to find the right entry, Y is made a part of the index sublabel of the child w holding the entry. This is enough to identify w among the children of v. However, the definition of the query algorithm requires us to identify w among all the vertices (not just the children of v), so, the unique label of v (in the scheme of [13]) is also added to the index sublabel of every child of v. The entry of the translation table is put in the data sublabel of the child.

Now, given the labels of above mentioned vertices v and u, the query algorithm finds w, the child of v whose index sublabel contains X. Then, the decoder algorithm consults w's data sublabel to find Y. (For the sake of simplicity, in the formal description, we chose w to be the vertex to which fixed port Y of v leads, that is, we identified w above with z above). We now give a formal description of the scheme.

Formal description of the routing scheme: Given a a tree $T \in \mathcal{T}(n)$ and a vertex $v \in T$, the index sublabel of v is composed of two fields, namely, $\mathcal{M}^{index}(v) = \langle \mathcal{M}_1^{index}(v), \mathcal{M}_2^{index}(v) \rangle$ and the data sublabel of v is composed of three fields, namely, $\mathcal{M}^{data}(v) = \langle \mathcal{M}_1^{data}(v), \mathcal{M}_2^{data}(v), \mathcal{M}_3^{data}(v) \rangle$.

The index sublabel of the root r of T is $\langle 0, 0 \rangle$ and the data sublabel of r is $\mathcal{M}^{data}(r) = \langle \mathcal{M}_{des}(r), 0, 0 \rangle$. If v is not the root then the index and data sublabels of v are,

$$\mathcal{M}^{index}(v) = \langle \mathcal{M}_{des}(p(v)) \, , \, port_{des}(p(v), v) \rangle \quad , \quad \mathcal{M}^{data}(v) = \langle \mathcal{M}_{des}(v) \, , \, port_{fix}(p(v), v) \, , \, port_{fix}(v, p(v)) \rangle.$$

Note, that we use the designer port number as a part of the label in the fixed-port model. Moreover, the designer port number at the parent is used as a part of the label of the child in the fixed-port model.

Note also, that $\mathcal{M}_{des}(x)$ must be unique for π_{des} to be a correct routing scheme. Therefore, if two vertices u and v share the same first field in the index sublabel then they also share the same parent. Is this case, the second field in the index sublabels of u and v must be different. It therefore follows that index sublabels are unique.

Given the labels $\mathcal{M}(v)$ and $\mathcal{M}(u)$ of two vertices v and u, the decoder $\mathcal{D}$ first checks whether $\mathcal{D}_{des}(\mathcal{M}_1^{data}(v), \mathcal{M}_1^{data}(u)) = 1$, i.e., whether the next vertex on the shortest path leading from v to u is v's parent. In this case, the query algorithm is ignored and the decoder $\mathcal{D}_{fix}$ simply outputs $\mathcal{M}_3^{data}(v)$ which is the (fixed) port number at v leading to its parent.

Otherwise, if $\mathcal{D}_{des}(\mathcal{M}_1^{data}(v), \mathcal{M}_1^{data}(u)) \neq 1$, then the query algorithm Q_{fix} outputs

$$\langle \mathcal{M}_1^{data}(v), \mathcal{D}_{des}(\mathcal{M}_1^{data}(v), \mathcal{M}_1^{data}(u)) \rangle \quad = \quad \langle \mathcal{M}_1^{data}(v), \mathcal{D}_{des}(\mathcal{M}_{des}(v), \mathcal{M}_{des}(u)) \rangle$$

which is precisely the index sublabel $\langle \mathcal{M}_1^{data}(v), port_{des}(v, w) \rangle$ of w, where w is the next vertex on the shortest path leading from v to u (and a child of v). Therefore, given labels $\mathcal{M}_{fix}(v)$, $\mathcal{M}_{fix}(u)$ and label $\mathcal{M}_{fix}(w)$, the decoder $\mathcal{D}_{fix}$ outputs $\mathcal{M}_2^{data}(w)$ which is the desired port number $port_{fix}(v, w)$. Since the label size of π_{des} is $O(\log n)$ and since each port number is encoded using $O(\log n)$ bits, we obtain the following lemma.

Lemma 3.5 *In the fixed-port model, the label size of a 1-query routing labeling scheme on $\mathcal{T}(n)$ is $\Theta(\log n)$.*

4 Examples for applications to traditional models

4.1 An application to traditional routing schemes

By applying the method described above to the traditional model, we now show how to construct a 3-approximation routing scheme (without queries) on $\mathcal{T}(n)$. Our 3-approximation routing labeling scheme π_{approx} operates in the fixed-port model, and it has label size and header size which are $O(\log n)$. Recall, that any (precise) routing scheme on $\mathcal{T}(n)$ must have label size $\Omega(\log^2 n/\log\log n)$ [14]. We note that our ideas for translating routing schemes from the designer port model to the fixed-port model appear in [3] implicitly. (Still, a 3-approximation routing scheme (without queries) on $\mathcal{T}(n)$ is not constructed there explicitly.)

Informally, the scheme operates as follows. Similarly to the query case, we use the known scheme of [13] for designer routing, and forward a message to the parent if that message would have been forwarded to the parent in the scheme of [13]. Consider the case where the next vertex on the route (from v to u) is some child s of v. The designer scheme of [13] yields a pointer X to s. That is, had the ports of v been numbered by the designer, the desired child s would have been pointed by port number X. Let Y be the corresponding real port number, i.e., the port number leading from v to s. We note that the port numbers assigned to v by the designer scheme of [13] are numbered from 1 to $deg(v)$, where $deg(v)$ is the degree of v. Moreover, if v is not the root, then the assigned designer port numbers leading to its children, are numbered from 2 to $deg(v)$. We store the value Y in the child w of v whose (fixed) port number is the $X-1$'st smallest among the (fixed) port numbers leading from v to its children. (In the case where v is the root, we store the value Y in the child w of v whose port number is the X'th smallest.) Therefore, the message from v can now go to w to obtain Y, go back to v and continue to s via port number Y. The formal description of the 3-approximation routing labeling scheme π_{approx} as well as the proof of the following lemma should be clear now from the informal description. For completeness, they are given in [2].

Lemma 4.1 *π_{approx} is a correct 3-approximation routing scheme on $\mathcal{T}(n)$ operating in the fixed port model. Moreover, its label size and header size are $\Theta(\log n)$.*

4.2 An application for flow queries in a non-distributed environment

As mentioned above, the query model may be found particularly useful for the sequential setting, since there, the issue of physically finding the consulted vertices does not arise. We now show that general graphs can be preprocessed efficiently to support flow queries in the sequential setting. (This is not very surprising, since we use NCA, which is known as a strong tool in sequential processing.)

The label-NCA labeling scheme $\pi_{NCA} = \langle \mathcal{M}_{NCA}, \mathcal{D}_{NCA} \rangle$ described in [5], answers NCA queries in constant time. By adding a pointer from each vertex to its weighted depth and using Equation 1, their method can be modified slightly to support also separation level (or distance) queries in a weighted tree. The space used for storing these pointers is $O(n \cdot \max\{1, \frac{\log W}{\log n}\})$ and the query time will now be $O(\max\{1, \frac{\log W}{\log n}\})$, if W is the maximum weight in the tree. Applying this on the weighted tree $\tilde{T}_G$ (of [18]) mentioned above (Section 3.3) yields the following.

Lemma 4.2 *Any graph $G \in \mathcal{G}(n, W)$ can be preprocessed using $O(n \cdot \max\{1, \frac{\log W}{\log n}\})$ space such that flow queries can be answered in $O(\max\{1, \frac{\log W}{\log n}\})$ time.*

5 Labeling schemes with $n-2$ queries

Inspecting two labels, and inspecting three, are approaches that lie on one end of a spectrum. On the other end of the spectrum would be a representation for which the decoder inspects the labels of all the vertices before answering ($(n-2)$-query labeling schemes). In this section, we give very simple examples showing that in many case, inspecting all vertices reduces the problem, in a sense, to a sequential one.

Lemma 5.1 *Let $\mathcal{F}_k(n)$ be any family of n-vertex graph with degrees at most k. Then, for any (polynomially computable) function f, there exist an $(n-2)$-query labeling scheme on $\mathcal{F}_k(n)$ with label size $O(k \log n)$.*

Proof: Given a graph $G \in \mathcal{F}_k(n)$, the marker algorithm first assigns each vertex v a unique identifier $id(v)$ in the range $1, 2, \cdots n$, and then labels each vertex by its own identifier and by the identifiers of all its neighbors. Clearly, the label size of such a scheme is $O(k \log n)$. Given a query about two vertices u and v, the decoder is allowed to consult the labels of all the vertices, and therefore, can reconstruct in polynomial time, the whole graph G. Then, the decoder simply outputs the value $f(u, v)$ by calculating it on G. ∎

The above lemma may clarify why we say that $n-2$ queries reduce a problem into a sequential one. This is because the computation performed by the decoder algorithm is a traditional sequential computation, using the whole graph as an input.

Lemma 5.2 *Let $\mathcal{F}(n)$ be any family of n vertex graphs. For any (computable) function f, there exists an $(n-2)$-query labeling scheme on $\mathcal{F}(n)$ with label size $\frac{\log |\mathcal{F}(n)|}{n} + O(\log n)$ (though the decoder may not be polynomial).*

Proof: Let ρ be the number of graphs in $\mathcal{F}(n)$, i.e., $\rho = |\mathcal{F}(n)|$. First, enumerate the graphs in $\mathcal{F}(n)$ by some arbitrary method, known to both the marker and the decoder algorithms. Given a graph $G \in \mathcal{F}(n)$, let i be its index in the above enumeration. Note, that i can be encoded using $\lceil \log |\mathcal{F}(n)| \rceil$ bits.

The marker algorithm distributes the value i among the vertices of G as follows. First, the marker assigns each vertex v a unique identifier $id(v)$ in the range $1, 2, \cdots n$. The marker algorithm now splits the encoding of i into at most n portions, each of size at most $\lceil \frac{\log |\mathcal{F}(n)|}{n} \rceil$. Let u be the j'th vertex in G, i.e., $id(u) = j$. The label given to u consists of two fields. The first field contain j and the second field contains the j'th portion in the encoding of i. We let the first field in each label consist of precisely $\lceil \log n \rceil$ bits, padding enough zero's to the left of the encoding if necessary. It follows that the label size of the scheme is $\frac{\log |\mathcal{F}(n)|}{n} + O(\log n)$.

Given the labels of all vertices, the decoder can easily reconstruct the index i, and therefore can reconstruct G, since it knows the enumeration of the graphs in $\mathcal{F}(n)$. (Note however, that for this reason, the decoder may not be polynomial.) Once it has G, the decoder can now answer the desired

query. ∎

The following lemma shows that the above lemma is in a sense, tight.

Lemma 5.3 *Let $\mathcal{F}(n)$ be any family of graphs on the set of n vertices $v_1, v_2, \cdots v_n$. Let f any function such that for any two different graphs G_1 and G_2 in $\mathcal{F}(n)$, there exists two vertices v_i and v_j such that $f_{G_1}(v_i, v_j) \neq f_{G_2}(v_i, v_j)$. (For example, f can be the distance function or the adjacency function). Then, the label size of any $(n-2)$-query f-labeling scheme on $\mathcal{F}(n)$ must be at least $\frac{\log|\mathcal{F}(n)|}{n}$.*

Proof: For any graph $G \in \mathcal{F}(n)$, let $L(G)$ denote the concatenation of all the labels given to the vertices in G, i.e., $L(G) = L(v_1) \circ L(v_2) \circ \cdots \circ L(v_n)$. By the assumption of f, given any two graphs G_1 and G_2 in $\mathcal{F}(n)$, the decoder, given the labels of all vertices, must be able make a distinction between the two graphs. Therefore, for any two graphs G_1 and G_2 in $\mathcal{F}(n)$, $L(G_1) \neq L(G_2)$. It follows, that there exists a graph $G \in \mathcal{F}(n)$ such that $|L(G)| \geq \log|\mathcal{F}(n)|$. Therefore, there exists a vertex $v \in G$ whose label consists of at least $\frac{\log|\mathcal{F}(n)|}{n}$ bits. ∎

6 Dynamic labeling schemes with one query

In this section, we demonstrate that the reduction in the label sizes obtained by introducing a single query in the dynamic scenario can be similar to the reduction in the static case. The specific dynamic schemes shown here are translations of the 1-query schemes of the previous sections to the limited dynamic models of *leaf incremental* and *leaf dynamic* [19, 23].

The initial idea is to apply the methods introduced in [19, 23], to convert labeling schemes for static networks to work on dynamic networks too. Unfortunately, we cannot do this directly, since these methods were designed for traditional labeling schemes and not for 1-query labeling schemes.

The next idea is to perform the conversion indirectly. For that, recall (Section 3), that our 1-query labeling schemes utilize known components. Those are schemes in the traditional model (with no queries). That is, some utilize π_{NCA}, the label-NCA labeling scheme of [5] and some utilize π_{des}, the routing scheme of [13]. Hence, one can first convert these components to the dynamic setting. Second, one can attempt to use the resulted dynamic components in a similar way that we used the static components in Section 3. This turns out to be simple in the cases of the distance, separation level, and id-NCA functions, and somewhat more involved in the case of the routing function.

6.1 Dynamic 1-query distance, separation-level and id-NCA labeling schemes

Recall, that in our (static) 1-query labeling schemes supporting the distance, separation level, and id-NCA functions, the index sublabel of each vertex v is simply the label assigned to v by π_{NCA}, the label-NCA labeling scheme of [5], and the data sublabel of v is either v's depth or v's identifier . Therefore, we can maintain the index sublabel of each vertex dynamically, using $\hat{\pi}_{NCA}$, the dynamic label-NCA labeling scheme resulted by applying either Theorem 4.16 of [23] or Theorems 1 and 2 of [19] on π_{NCA}. In addition, the data sublabel of each vertex v can be maintained easily as follows. In case we consider the id-NCA function, whenever a new vertex v joins the tree and an identifier $id(v)$ is given to v by the adversary, this identifier is stored at the data sublabel of v. If we consider,

instead, either the distance or the separation level functions, the data sublabel of the root is set to be 0, and whenever a new vertex v joins the tree, it communicates with its parent $p(v)$ and sets its data sublabel to be $\hat{\mathcal{M}}_{data}(v) = 1 + \hat{\mathcal{M}}_{data}(p(v))$, where $\hat{\mathcal{M}}_{data}(p(v))$ is the data sublabel given to $p(v)$ by our dynamic 1-query scheme. Therefore, each vertex v maintains its depth (or identifier) in the its data sublabel, with an extra constant additive cost to the amortized message complexity of $\hat{\pi}_{NCA}$.

The query and decoder algorithms of the resulted dynamic 1-query labeling scheme relate to $\hat{\pi}_{NCA}$ similarly to the way the query and decoder algorithms of the corresponding static 1-query scheme relate to π_{NCA}. It follows, that Theorem 4.16 of [23] and Theorems 1 and 2 of [19] imply the claims of Theorem 6.1 regarding the dynamic 1-query labeling schemes supporting the distance, separation level and id-NCA functions.

6.2 Dynamic 1-query routing labeling schemes for the fixed port model

When trying to translate our 1-query routing labeling scheme to the dynamic settings, we again translate using updates. We use two translations. Let us, first, explain the use of the translation of [23] in order to prove the second part of Theorem 6.1 below. The use of this translation is rather immediate. After it,we explain the use of the translation of [19]. This somewhat more involved translation application is used to prove the first and third part of Theorem 6.1 below.

Recall, that in φ_{fix}, our static 1-query routing labeling scheme, each non-root vertex 'knows' the label assigned to its parent by another static routing scheme π_{des} (see Section 3). The natural translation is to enable the child to continue to 'know' the label of its parent as follows.
Update: whenever the label of a non-leaf vertex changes, it notifies the new label to all its children. When a child receives such a notification message, it updates its φ_{fix} label accordingly (see Section 3 for the way the labels of φ_{fix} are computed).
The main technical issue is to show that the number of such updates is small.

Let us account for the messages used for the above update. This turns out to be relatively cheap when using the translation method of [23]. The relevant facts about [23] are the following

1. Theorem 4.16 of [23]: Consider the leaf-incremental model. If the final number of vertices n_f is known in advance, then there exists a dynamic routing labeling scheme $\hat{\pi}_{des}$ for the designer port model, with label size $O(\frac{\log^3 n_f}{\log\log n_f})$ and message complexity $O(\frac{\log n_f}{\log\log n_f})$.
2. In the above dynamic routing labeling scheme $\hat{\pi}_{des}$, the label of a vertex v changes only as a part of changes in all the vertices of some subtree T'.
3. The changes in T' mentioned in Item 2 above involve $\Omega(|T'|)$ messages anyhow (not including the new update messages we are introducing now from v to its children).
4. whenever a vertex v is included in T', all its children are included in T' as well.

By the above items, the cost of the new update messages can be amortized over the cost of the messages sent anyhow in T' to perform the changes. Hence, we can (1) use the transformation of [23] to obtain the above dynamic routing scheme $\hat{\pi}_{des}$ for the designer port model, and then (2) use the

new update messages to implement φ_{fix} using $\hat{\pi}_{des}$. This yields a dynamic 1- query routing scheme, enjoying the complexities promised in [23]. This discussion is summarized in part 2 in Theorem 6.1 below, for routing. (Recall, that we already proved above the theorem for the other functions).

Other dynamic 1-query routing schemes are claimed in parts 1 and 3 of Theorem 6.1 below. For these other schemes we use the translation method (from static to dynamic) of [19]. The adaptation of that translation to the query model turns out to be somewhat less easy.

We now show how to adapt the schemes of [19] to obtain parts 1 and 3 of Theorem 6.1. For this, we need to know the list of the cases where messages are sent in the schemes of [19]. This is because we would like to show that we can amortize the cost of the new messages we now send, on the messages sent anyhow by the method of [19].

Like in the case of [23], messages in [19] are sent whenever the main protocol applies some *reset* protocol on some subtree T'. Moreover, each such application involves sending at least $|T'|$ messages. The labels of vertices may only change due to such applications. However, unlike the case of [23], here, a subtree T' may include some vertex v, but not all of its children. Recall, that our modifications includes the update: sending messages from v, whose label was changed, to all of its children. When v has children outside of T' (unlike the case of using [23] above), we cannot amortize the cost of the notification messages on the cost of the messages sent anyhow by the method of [19].

Fortunately, the only vertex in T' such that not all of its children are necessarily in T' is the root r' of T'. Therefore, the update messages sent by a vertex $v \in T'$, where $v \neq r'$, do not increase the cost of the messages sent anyhow by the method of [19].

The only thing left to consider is the case where the root r' changes its label in the dynamic labeling scheme given by the method of [19] applied on π_{des}, the static routing scheme of [13]. Informally, to bound the number of notification messages sent by r', we make sure that the label of r' does not change as a result of applying the reset protocol on T' (This is done by modifying the static routing scheme π_{des} such that the label of the root is always $\langle 1 \rangle$, no matter which tree it belongs to.)

It, therefore, follows, that the cost of the remaining notification messages can be amortized on the cost of the messages sent anyhow by [19]. The necessary modifications of the schemes, as well as a more formal proof of the claims in the following theorem regarding routing, are full of details, but are not difficult. For completeness, they appear in [2].

Theorem 6.1 *Consider the fixed-port model and let $k(x)$ be any function satisfying that $k(x)$, $\log_{k(x)} x$ and $\frac{k(x)}{\log k(x)}$ are nondecreasing functions and that $k(\Theta(x)) = \Theta(k(x))$.[3] There exist dynamic 1-query labeling schemes on trees supporting the distance, separation level, id-NCA and routing functions, with the following complexities.*

1. *In the leaf-incremental tree model, with label size $O(\log_{k(n)} n \cdot \log n)$ and amortized message complexity $O(k(n) \cdot \log_{k(n)} n)$.*
2. *In the leaf-incremental tree model, if an upper bound n_f on the number vertices in the dynamically growing tree is known in advance, with label size $O(\frac{\log^3 n_f}{\log \log n_f})$ and amortized message complexity*

[3] The above requirements are satisfied by most natural sublinear functions such as $\alpha x^{\epsilon} \log^{\beta} x$, $\alpha \log^{\beta} \log x$ etc..

$O(\frac{\log n_f}{\log\log n_f})$.

3. *In the leaf-dynamic tree model, with label size* $O(\log_{k(n)} n \cdot \log n)$ *and amortized message complexity* $O\left(\sum_i k(n_i) \cdot \log_{k(n_i)} n \cdot \frac{\mathcal{MC}(\pi,n_i)}{n_i}\right) + O(\sum_i \log^2 n_i)$.

7 Conclusion and open problems

In this paper, we raise the question of what happens if one changes the model for implicit labeling schemes somewhat. In particular, we discuss the use of three labels instead of two. We also demonstrated a more drastic change of considering all n labels.

A natural question is to examine other points in this spectrum, i.e, examine c-query labeling schemes for $1 < c < n$. In particular, it would be interesting to find non-trivial c-query labeling schemes as variants of more complex labeling schemes, such as the ones in [16, 20, 30].

There are other dimensions to the above question. For example, by our definition, given the labels of two vertices, the (at most) c vertices that are chosen by the query algorithm Q are chosen simultaneously. Alternatively, one may define a possibly stronger model in which these vertices are chosen one by one, i.e., the next vertex is determined using the knowledge obtained from the labels of previous vertices.

In addition, as mentioned in Subsection 2.3, it would be interesting to consider query labeling schemes in which the queried vertices come from a restricted set of vertices. This may be found especially useful in hierarchical routing schemes.

References

[1] I. Cidon, I. S. Gopal, and S. Kutten. New models and algorithms for future networks. *IEEE Transactions on Information Theory* 41(3): 769-780 (1995).

[2] A. Korman and S. Kutten. A Note on Models for Graph Representations. http://ie.technion.ac.il/kutten/SIROCCO2007.pdf

[3] I. Abraham, C. Gavoille, and D. Malkhi. Routing with improved communication-space trade-off. *Proc. 18th Int. Symp. on Distributed Computing (DISC)*, Oct. 2004.

[4] D. Angluin, J. Aspnes, J. Chen, Y Wu, Y. Yin. Fast construction of overlay networks. *Proc. SPAA 2005*, pp. 145-154.

[5] S. Alstrup, C. Gavoille, H. Kaplan and T. Rauhe. Nearest Common Ancestors: A Survey and a new Distributed Algorithm. *Theory of Computing Systems* **37**, (2004), 441–456.

[6] S. Abiteboul, S. Alstrup, H. Kaplan, T. Milo and T. Rauhe. Compact labeling schemes for ancestor queries. *SIAM Journal on Computing* **35**, (2006), 1295–1309.

[7] S. Alstrup and T. Rauhe. Small induced-universal graphs and compact implicit graph representations. In *Proc. 43'rd annual IEEE Symp. on Foundations of Computer Science*, Nov. 2002.

[8] B. Awerbuch and D. Peleg. Sparse Partitions. *FOCS* 1990: pp. 503-513.

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

[10] Bruno Courcelle and Andrew Twigg. Compact Forbidden-Set Routing, Proceedings of STACS 2007, pp. 37-48.

[11] Domain name system. Wikipedia, http://en.wikipedia.org/wiki/Domain_name_system.

[12] T. Eilam C. Gavoille and D. Peleg. Compact Routing Schemes with Low Stretch Factor. In *Proc. 17th Annual ACM Symp. on Principles of Distributed Computing* , ACM Press, may 1996, 11–20.

[13] P. Fraigniaud and C. Gavoille. Routing in trees. In *Proc. 28th Int. Colloq. on Automata, Languages & Prog.*, LNCS 2076, pages 757–772, July 2001.

[14] P. Fraigniaud and C. Gavoille. A space lower bound for routing in trees. In *Proc. 19th Int. Symp. on Theoretical Aspects of Computer Science*, March 2002, 65-75.

[15] P. Fraigniaud and P. Gauron. D2B: A de Bruijn based content-addressable network. Theor. Comput. Sci. 355(1): 65-79 (2006).

[16] C. Gavoille, D. Peleg, S. Pérennes, and R. Raz. Distance labeling in graphs. *J. of Algorithms* **53(1)** (2004), 85–112.

[17] M. Harchol-Balter, F. T. Leighton, and D. Lewin. Resource Discovery in Distributed Networks. *Proc. PODC* 1999, pp. 229-237.

[18] M. Katz, N.A. Katz, A. Korman and D. Peleg. Labeling schemes for flow and connectivity. *SIAM Journal on Computing* **34** (2004),23–40.

[19] A. Korman. General Compact Labeling Schemes for Dynamic Trees. In *J. Distributed Computing*, to appear. (Conference version: DISC 2005).

[20] A. Korman. Labeling Schemes for Vertex Connectivity. In *Proc. 34th Int. Colloq. on Automata, Languages, and Programming*, 2007, pp. 102-109.

[21] A. Korman and S. Kutten. Distributed Verification of Minimum Spanning Trees. In *J. Distributed Computing*, to appear. (Conference version: PODC 2006).

[22] A. Korman and D. Peleg. Labeling Schemes for Weighted Dynamic Trees. In *Information & Computation*, to appear. (Conference version: ICALP 2003).

[23] A. Korman, D. Peleg, and Y. Rodeh. Labeling schemes for dynamic tree networks. *Theory of Computing Systems* **37**, (2004), 49–75.

[24] S. Kannan, M. Naor, and S. Rudich. Implicit representation of graphs. In *SIAM J. on Discrete Math* **5**, (1992), 596–603.

[25] E. P. Markatos and G. Dramitinos. Remote Memory to Avoid Disk Thrashing: A Simulation Study. *Proc. MASCOTS* 1996: 69-73.

[26] D. Peleg. Proximity-preserving labeling schemes and their applications. In *Proc. 25th Int. Workshop on Graph-Theoretic Concepts in Computer Science*, pages 30–41, June 1999.

[27] D. Peleg. Informative labeling schemes for graphs. In *Proc. 25th Symp. on Mathematical Foundations of Computer Science*, volume LNCS-1893, pages 579–588. SV, Aug. 2000.

[28] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. *Proc. ACM SIGCOMM* 2001, pp. 161-172, August 2001.

[29] N. Santoro and R. Khatib. Labeling and implicit routing in networks. *The Computer Journal* **28**, (1985), 5–8.

[30] M. Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *J. of the ACM* **51**, (2004), 993–1024.

[31] M. Thorup and U. Zwick. Compact routing schemes. In *Proc. 13th ACM Symp. on Parallel Algorithms and Architecture*, pages 1–10, Hersonissos, Crete, Greece, July 2001.

[32] J. Van Leeuwen and R. B. Tan. Interval routing. *The Computer Journal* **30**, (1987), 298–307.