Labeling Schemes with Queries

Amos Korman * and Shay Kutten **

Information Systems Group, Faculty of IE&M, The Technion, Haifa, 32000 Israel.
pandit@tx.technion.ac.il, kutten@ie.technion.ac.il

Abstract. Recently, quite a few papers studied methods for representing network properties by assigning *informative labels* to the vertices of a network. Consulting the labels given to any two vertices u and v for some function f (e.g. "distance(u, v)") one can compute the function (e.g. the graph distance between u and v). Some very involved lower bounds for the sizes of the labels were proven.

In this paper, we demonstrate that such lower bounds are very sensitive to the number of vertices consulted. That is, we show several almost trivial constructions of such labeling schemes that beat the lower bounds by large margins. The catch is that one needs to consult the labels of *three* vertices instead of two. We term our generalized model *labeling schemes with queries*.

Additional contributions are several extensions. In particular, we show that it is easy to extend our schemes for tree to work also in the dynamic scenario. We also demonstrate that the study of the queries model can help in designing a scheme for the traditional model too. Finally, we demonstrate extensions to the non-distributed environment. In particular, we show that one can preprocess a general weighted graph using almost linear space so that flow queries can be answered in almost constant time.

Keywords: Labeling schemes, routing schemes, distance queries.

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 most 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 forms a *global* representation of the network. Recently, quite a few papers studied methods for representing network properties by assigning *informative labels* to the vertices of the network (see e.g., [40, 9, 29, 36, 45]).

* Supported in part at the Technion by an Aly Kaufman fellowship.
** Supported in part by a grant from the Israel Science Foundation.

Let f be a function on pairs of vertices (e.g., distance). Informally, the goal of an f-labeling scheme is to label the vertices of a graph G in such a way that for every two vertices $u, v \in G$, the value $f(u, v)$ (e.g., the distance between u and v) can be inferred by merely inspecting the labels of u and v. Of course, this can be done trivially using labels that are large enough (e.g., every label includes the description of the whole graph). Therefore, the main focus of the research concerning labeling schemes is to minimize the amount of information (the sizes of the labels) required. Informally, an f-labeling scheme can be viewed as a way of distributing the graph structure information concerning f to the vertices of the graph, using small chunks of information per vertex.

Rather involved proofs were introduced to lower bound the sizes of such labeling schemes. The main contribution of this paper is the demonstration that these lower bounds are very sensitive to the model used. Intuitively, if the labels of three vertices can be consulted (rather than two, such as u and v above), it is very easy to reduce the sizes significantly, much below the previous lower bounds. Moreover, our query labeling schemes can be obtained using very simple methods, sometimes trivial.

Elaborating somewhat more (formal definitions appear in Section 2), this paper introduces the notion of *f-labeling schemes with queries* which generalizes the notion of f-labeling schemes. 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 additional vertices. We note that all the constructions given in this paper calculate $f(u, v)$ by inspecting the labels of three vertices (u and v above, and some w). That is, given the labels of u and v, we first find a vertex w and then consult its label to derive $f(u, v)$. However, in the concluding section we discuss generalizations to inspecting additional labels.

We also show several additional extensions. One extension is meant to demonstrate an advantage of the simplicity of the design of labeling schemes with queries. It helped us to simplify the design of a labeling scheme for the traditional model (with no queries). We did that in two steps: first we designed the simple scheme with queries, and then "simulated" this scheme in the old model (with no queries). (The "simulation" had some associated cost, which made the resulting scheme work for an approximation function f, rather than for the exact function we would have liked).

A second extension of the result is to dynamic scenarios. We note that most previous research concerning distributed network representations considered the *static* scenario, in which the topology of the underlying network is fixed. This is probably due to the fact that designing for the dynamic scenario is more complex. Some recent papers did tackle task of labeling dynamic networks in a distributed fashion. Such labeling methods should, of course, be dynamic too. Indeed, the designs in these recent paper tend to be harder and more complex. We show that the effect of introducing a query to the dynamic case is similar to the effect on the static case. That is- the sizes can be reduced considerably, and the construction of the schemes is rather easy (though somewhat more complex

than in the static case). To do that, we modify the model translation methods of [35] and [38], and then use them to extend our static labeling schemes with queries on trees to the dynamic scenario. We then show that the sizes of the resulted schemes are smaller than those of the schemes for the older model. The reduction in the label size is similar to the reduction in the static case.

In a final extension, we show that our methods are also useful in the non-distributed environment.

Related work: In this subsection we mostly survey results concerning labeling schemes (with no queries). However, let us first mention an area of research (namely, overlay and Peer to Peer networks) that may serve as a practical motivation for our work, and for some other studies concerning labeling schemes. We stress, though, that the main motivation for this paper is theoretical.

When the third vertex w (mentioned above) is near by to u, it may be quite cheap for u to access the main memory at w, sometimes even cheaper than consulting the disk at u itself. See, for example [41, 34]. Indeed, some of our schemes below are based on such a "near by" w. Even when w is remote, accessing it may be cheap in some overlay networks. The main overhead there is finding w (which can be done in our constructions by u using v's label) and creating the connection to it. Such models are presented explicitly, e.g. in [31, 15, 2, 48], where such remote accesses are used to construct and to use overlay data structures. Famous overlay data structures that can fit such models appear for example in [49, 55, 42, 23, 43]. In some of these overlay networks, a vertex w is addressed by its contents. This may motivate common labeling schemes that assume content addressability.

Implicit labeling schemes were first introduced in [12, 40]. Labeling schemes supporting the adjacency and ancestry functions on trees were investigated in [40, 9, 8].

Distance labeling schemes were studied in [44, 29, 52, 17, 4, 39]. In particular, [44] 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 [29] to be asymptotically optimal.

Labeling schemes for routing on trees were investigated in a number of papers until finally optimized in [21, 22, 54]. For the *designer port* model, in which the designer of the scheme can freely enumerate the port numbers of the nodes, [21] shows how to construct a routing scheme using labels of $O(\log n)$ bits on n-node trees. In the *adversary 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-node trees. In [22] they show that both label sizes are asymptotically optimal. Independently, a routing scheme for trees using $(1 + o(1)) \log n$-bit labels was introduced in [54] 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 schemes, 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 [45] 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 [36] they give a labeling scheme supporting the flow function on n-node general graphs using $\Theta(\log^2 n + \log n \log W)$-bit labels, where W is the maximum capacity of an edge. They also show a labeling scheme supporting the k-vertex-connectivity function on general graphs using $O(2^k \log n)$-bit labels. See [27] for a survey on (static) labeling schemes.

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-increasing* 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 [38, 37]. In [38], 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, 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 nodes) are presented [37]. 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-increasing tree model, the resulting dynamic scheme in [38] incurs overheads (over the static scheme) 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 [38] incurs 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 [35], it is shown how to construct for many functions $k(x)$, a dynamic labeling scheme in the leaf-increasing tree model extending a given static scheme, such that the resulting scheme incurs 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 [38], 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, $O(n^\epsilon)$.

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 that by increasing slightly the number of vertices whose labels are inspected, the size of the labels decreases considerably. Specifically, we inspect the labels of 3 vertices instead of 2, that is, we use a single *query*. In particular, we show that there exist simple labeling schemes with one query supporting the distance function on n-node trees as well as the flow function on n-node general graphs with label size $O(\log n + \log W)$, where W is the maximum (integral) capacity of an edge. (We note that the lower bound for labeling schemes without queries for each of these problems is $\Omega(\log^2 n + \log n \log W)$ [29, 36].) We also show that there exists a labeling scheme with one query supporting the id-NCA function on n-node trees with label size $O(\log n)$. (The lower bound for schemes without queries is $\Omega(\log^2 n)$ [45].) In addition, we show a routing labeling scheme with one query in the fixed-port model using $O(\log n)$-bit labels. (The lower bound (see [22]) 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.) Moreover, most of the results are obtained by simple constructions, which strengthens the motivation for this model.

We then show several extensions that are somewhat more involved. In particular, we show that our labeling schemes with queries on trees can be extended to the dynamic scenario using model translation methods based on those of [38, 35]. In order to save in the message complexity, we needed to make some adaptations to those methods, as well as to one of the static routing schemes of [21]. Second, 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 turn to a non-distributed environment and demonstrate similar constructions. That is, first, we show a simple method to transform previous results on NCA queries on static and dynamic trees in order to support also distance queries. Then, we show 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 T be a tree and let v be a vertex 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. In the case where the tree T is weighted (respectively, unweighted), the *depth* of a vertex is defined as its weighted (resp., unweighted) 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-node unweighted trees. Let

$\mathcal{T}(n, W)$ (respectively, $\mathcal{G}(n, W)$) denote the family of all n-node weighted trees (resp., connected graphs) with (integral) edge weights bounded above by W.

Incoming and outgoing links from every node 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 freely assign the port numbers of each vertex (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.

We consider the following functions which are applied on pairs of vertices u and v in a graph $G = \langle V, E \rangle$. (1) **flow** (maximum legal flow between u and v), (2) **distance** (either weighted, or unweighted), (3) **routing** (the port in u to the next vertex towards v). If the graph is a tree T then we consider also the following functions: (4) **separation level** (depth of $NCA(u, v)$), (5) **id-NCA**, (6) **label-NCA**. In (5) above, it is assumed that identities containing $O(\log n)$ bits are assigned to the vertices by an adversary, and $id - NCA(u, v)$ is the identity of $NCA(u, v)$. In (6) above, it is assumed that each vertex can freely select its own identity (as long as all identities remain unique). In this case, the identities may also be referred to as labels.

Labeling schemes and c-query labeling schemes: Let f be a function defined on pairs of vertices. 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 constant 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. 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 identities.)
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 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 in $Q(\mathcal{M}(u), \mathcal{M}(v))$, 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 $G \in \mathcal{F}$. We note that all the schemes in this paper use $c = 1$. Let us comment also that clearly, since the index sublabels must be disjoint, any c-query f-labeling scheme on any family of n-node graphs must have label size $\Omega(\log n)$. See Section 7 for alternative definition for query labeling schemes.

2.1 Routing schemes and β-approximation routing schemes

A *routing scheme* is composed of a *marker algorithm* $\mathcal{M}$ for assigning each vertex v of a graph G with a label $\mathcal{M}(v)$, coupled with 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 weighted length of 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 that the requirement is that the length of resulting route connecting x and y is a β-approximation for the distance between x and y in G.

In addition to the label size, we also measure 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.

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-node trees, schemes supporting the routing (in the fixed-port model), distance, separation level, and the id-NCA functions; for the family of n-node 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]. Given an n-node, 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 $\mathcal{D}_{NCA}$ outputs the label $\mathcal{M}_{NCA}(w)$.

3.1 Id-NCA function in trees

We first describe a 1-query 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 $\Omega(\log^2 n)$ [45].)

Informally, the idea behind φ_{id-NCA} is to have the labels of u and v (their first sublabels) be the labels given by the label-NCA labeling scheme $\pi_{NCA}(v)$. Hence, they are enough for the query algorithm to find the π_{NCA} label of their nearest common ancestor w. Then, the decoder algorithm finds w's identity simply in the second sublabel of w.

Let us 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 identity $id(v)$ by an adversary and that each such identity is composed of $O(\log n)$ bits. The marker algorithm $\mathcal{M}_{id-NCA}$ labels each vertex v with the label $\mathcal{M}_{id-NCA}(v) = \langle \mathcal{M}^{index}_{id-NCA}(v), \mathcal{M}^{data}_{id-NCA}(v) \rangle = \langle \mathcal{M}_{NCA}(v), id(v) \rangle$. 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 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 identity 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 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 $\varphi_{sep-level}$ and φ_{dist} supporting the distance and separation level functions respectively on $\mathcal{T}(n, W)$. Both our scheme have label size $\Theta(\log n + \log W)$. Recall that any labeling scheme (without queries) supporting either the distance function or the separation level function on $\mathcal{T}(n, W)$ must have size $\Omega(\log^2 n + \log n \log W)$, [29, 45]. The proof of the following lower bound claim is deferred to the full paper.

Claim. Let c be a 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 $\Omega(\log W + \log n)$.

The construction of our 1-query labeling schemes supporting the separation level and distance functions uses a similar method to the one described in Subsection 3.1. Both schemes are based on keeping the depth of a vertex in its data sublabel (instead of its identity). The correctness of the 1-query labeling scheme supporting the distance function is based on the following equation.

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

The description of these schemes as well as the proof of the following lemma is deferred to the full paper.

Lemma 2. *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 Routing in trees using one query

As mentioned before, any 1-query routing labeling scheme on $\mathcal{T}(n)$ must have label size $\Omega(\log n)$. In this subsection, we establish a 1-query routing labeling scheme φ_{fix} in the fixed-port model using $O(\log n)$-bit labels.

In [21], 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 the routing scheme π_{des}) 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 $\mathcal{M}_{des}(w)$ of two vertices v and w in T, the decoder $\mathcal{D}_{des}$ outputs the port number $port_{des}(v,u)$ at v leading from v to the next vertex u on the shortest path connecting v and w.

Let T be an n-node 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 an *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 its designer port number.

We now describe our 1-query routing scheme $\varphi_{fix} = \langle \mathcal{M}_{fix}, Q_{fix}, \mathcal{D}_{fix} \rangle$ which operates in the fixed-port model. 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, $\mathcal{M}^{data}(v) = \langle \mathcal{M}_1^{data}(v), \mathcal{M}_2^{data}(v), \mathcal{M}_3^{data}(v) \rangle$. If v is not the root then the index and data sublabels of v are $\mathcal{M}^{index}(v) = \langle \mathcal{M}_{des}(p(v)) \ , \ port_{des}(p(v),v) \rangle$ and $\mathcal{M}^{data}(v) = \langle \mathcal{M}_{des}(v) \ , \ port_{fix}(p(v),v) \ , \ port_{fix}(v,p(v)) \rangle$. 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 to label the child in the fixed-port model. Also note that the index sublabel is unique, since $\mathcal{M}_{des}(x)$ must be unique for π_{des} to be a correct routing scheme.

The index sublabel of the root r of T is $\langle 0, 0\rangle$ and the data sublabel of r is $\mathcal{M}^{data}(r) = \langle \mathcal{M}_{des}(r), 0, 0\rangle$. Note that since the labels given by the marker algorithm $\mathcal{M}_{des}$ are unique, the index sublabels of the vertices are unique.

Given the labels $\mathcal{M}(v)$ and $\mathcal{M}(w)$ of two vertices v and w, the decoder $\mathcal{D}$ first checks whether $\mathcal{D}_{des}(\mathcal{M}_1^{data}(v), \mathcal{M}_1^{data}(w)) = 1$, i.e., whether the next vertex on the shortest path leading from v to w 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, the query algorithm Q_{fix} outputs
$\langle \mathcal{M}_1^{data}(v), \mathcal{D}_{des}(\mathcal{M}_1^{data}(v), \mathcal{M}_1^{data}(w))\rangle = \langle \mathcal{M}_1^{data}(v), \mathcal{D}_{des}(\mathcal{M}_{des}(v), \mathcal{M}_{des}(w))\rangle$
which is precisely the index sublabel of u, the next vertex on the shortest path leading from v to w (and a child of v), i.e., $\langle \mathcal{M}_1^{data}(v), port_{des}(v, u)\rangle$. Therefore, given labels $\mathcal{M}_{fix}(v)$, $\mathcal{M}_{fix}(w)$ and label $\mathcal{M}_{fix}(u)$, the decoder $\mathcal{D}_{fix}$ outputs $\mathcal{M}_2^{data}(u)$ which is the desired port number $port_{fix}(v, u)$. 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. *In the fixed-port model, the label size of a 1-query routing scheme on $\mathcal{T}(n)$ is $\Theta(\log n)$.*

3.4 Flow in general graphs

We now consider the family $\mathcal{G}(n, W)$ of connected n-node 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 $\Omega(\log^2 n + \log n \log W)$ [36]. The proof of the following lemma is deferred to the full paper.

Lemma 4. *The label size of a 1-query flow labeling scheme on $\mathcal{G}(n, W)$ is $\Theta(\log n + \log W)$.*

4 A 3-approximation routing scheme in the fixed-port model

We construct a 3-approximation routing scheme (without queries) on $\mathcal{T}(n)$ by applying the method described in Subsection 3.3 to the traditional model. Our 3-approximation routing labeling scheme π_{approx} operates in the fixed-port model and has label size and header size $O(\log n)$. Recall that any (precise) routing scheme on $\mathcal{T}(n)$ must have label size $\Omega(\log^2 n / \log\log n)$ [22]. We note that our ideas for translating routing schemes from the designer port model to the fixed-port model implicitly appear in [1], however, a 3-approximation routing scheme (without queries) on $\mathcal{T}(n)$ is not explicitly constructed there. The description of the 3-approximation routing labeling scheme π_{approx} as well as the proof of the following lemma is deferred to the full parer.

Lemma 5. *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 $\Theta(\log n)$.*

5 Adapting the 1-query schemes on trees to the dynamic setting

In this section we show how to translate our 1-query labeling schemes on trees to the dynamic settings, i.e, to the leaf-increasing and leaf-dynamic tree models, [35, 38] (see also "Related work" in Section 1). In a dynamic scheme, the marker protocol updates the labels after every topological change. We show that the reduction in the label sizes obtained by introducing a single query in the dynamic scenario is similar to the reduction in the static case.

To describe the adaptation fully, we need to give many details about the methods of [38, 35, 21]. Unfortunately, this is not possible in this extended abstract.

The initial idea is to apply the methods introduced in [38, 35] 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. That is, recall (Section 3) that our 1-query labeling schemes utilize components that 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 [21]. 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, but more involved in the case of the routing function. The necessary modifications of the schemes, as well as the proof of the following theorem, appear in the full paper.

Theorem 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 $k(\Theta(x)) = \Theta(k(x))$.*[1] *There exist dynamic 1-query labeling schemes supporting the distance, separation level, id-NCA and routing functions on trees with the following complexities.*

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

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

6 Applications in a non-distributed environments

Distance queries in trees: Harel and Tarjan [33] describe a linear time algorithm to preprocess a tree and build a data structure allowing NCA queries to be answered in constant time on a RAM. Subsequently, simpler algorithms with better constant factors have been proposed in [47, 13, 25, 51, 14]. On a pointer machine, [33] show a lower bound of $\Omega(\log\log n)$ on the query time, which matches the upper bound of [53]. In the leaf-increasing tree model, [20, 10] show how to make updates in amortized constant time while keeping the constant worst-case query time on a RAM, or the $O(\log\log n)$ worst-case query time on a pointer machine. In [16] they show how to maintain the above mentioned results on a RAM in the leaf-dynamic tree model with worst case constant update. See [5] for a survey.

Simply by adding a pointer from each vertex to its depth and using Equation 1, we obtain the following lemma.

Lemma 6. *The results of [33, 47, 13, 25, 51, 14, 33, 53, 20, 10, 16] can be translated to support either distance queries or separation level queries (instead of NCA queries).*

We note that other types of dynamic models were studied in the non-distributed environment regarding NCA queries (e.g. [16, 10]). However, in these types of topological changes, our transformation to distance queries is not efficient since any such changes may effect the depth of too many vertices.

Flow queries in general graphs: Let $G \in \mathcal{G}(n, W)$ and let $u_1, u_2, \cdots, u_n$ be the set of vertices of G. Recall that in in [36], they show how to construct a weighted tree $\tilde{T}_G \in \mathcal{T}(O(n), W \cdot n)$ with n leaves $v_1, v_2, \cdots, v_n$ such that $flow_G(u_i, u_j) = sep - level_{\tilde{T}_G}(v_i, v_j)$. Using Lemma 6 applied on the results of [33], we can preprocess $\tilde{T}_G$ with $O(n \cdot \max\{1, \frac{\log W}{\log n}\})$ space such that separation level queries can be answered in $O(\max\{1, \frac{\log W}{\log n}\})$ time. The exact model needed to prove the following lemma formally is deferred to the full paper.

Lemma 7. *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.*

7 Conclusion and open problems

In this paper we demonstrate that considering the labels of three vertices, instead of two, can lead to a significant reduction in the sizes of the labels. 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 nodes before answering (n-query labeling schemes). It is not hard to show that for any graph family $\mathcal{F}$ on n node graphs, and for many functions (for example, distance or adjacency), one can construct an n-query labeling scheme on $\mathcal{F}$ using asymptotically optimal labels, i.e, $\log|\mathcal{F}|/n + \Theta(\log n)$-bit labels (though the decoder may not be polynomial). The idea behind

such a scheme is to enumerate the graphs in $\mathcal{F}$ arbitrarily. Then, given some $G \in \mathcal{F}$ whose number in this enumeration is i, distribute the binary representation of i among the vertices of G. In this way, given the labels of all nodes, the decoder can reconstruct the graph and answer the desired query. Therefore, a natural question is to examine other points in this spectrum, i.e, examine c-query labeling schemes for $1 < c < n$.

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

References

1. I. Abraham, C. Gavoille, and D. Malkhi. Routing with improved communication-space trade-off. *Proc. 18th Int. Symp. on Distributed Computing*, Oct. 2004.
2. D. Angluin, J. Aspnes, J. Chen, Y Wu, Y. Yin. Fast construction of overlay networks. *Proc. SPAA 2005*, pp. 145-154.
3. Y. Afek, B. Awerbuch, S.A. Plotkin and M. Saks. Local management of a global resource in a communication. *J. of the ACM* **43**, (1996), 1–19.
4. S. Alstrup, P. Bille and T. Rauhe. Labeling schemes for small distances in trees. In *Proc. 14th ACM-SIAM Symp. on Discrete Algorithms*, Jan. 2003.
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. Alstrup, J. Holm and M. Thorup. Maintaining Center and Median in Dynamic Trees. In *Proc. 7th Scandinavian Workshop on Algorithm Theory*, July. 2000.
7. S. Abiteboul, H. Kaplan and T. Milo. Compact labeling schemes for ancestor queries. In *Proc. 12th ACM-SIAM Symp. on Discrete Algorithms*, Jan. 2001.
8. S. Alstrup and T. Rauhe. Improved Labeling Scheme for Ancestor Queries. In *Proc. 19th ACM-SIAM Symp. on Discrete Algorithms*, Jan. 2002.
9. 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.
10. S. Alstrup and M. Thorup. Optimal pointer algorithms for finding nearest common ancestors in dynamic trees. *J. of Algorithms*, **35(2)**, (2000), 169–188.
11. M.A. Breuer. Coding the vertexes of a graph. *IEEE Trans. on Information Theory*, IT-12:148–153, 1966.
12. M.A. Breuer and J. Folkman. An unexpected result on coding the vertices of a graph. *J. of Mathematical Analysis and Applications* **20**, (1967), 583–600.
13. M.A. Bender and M. Farach-Colton. The LCA problem revised. In *4th LATIN*, 88–94, 2000.
14. O. Berkman and U. Vishkin. Recursive star-tree parallel data structure. *SIAM J. on Computing* **22(2)**, (1993), 221–242.
15. 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).
16. R. Cole and R. Hariharan. Dynamic LCA Queries on Trees. *SIAM J. on Computing* **34(4)**, (2005), 894-923.

17. E. Cohen, E. Halperin, H. Kaplan and U. Zwick. Reachability and Distance Queries via 2-hop Labels. In *Proc. 13th ACM-SIAM Symp. on Discrete Algorithms*, Jan. 2002.
18. E. Cohen, H. Kaplan and T. Milo. Labeling dynamic XML trees. In *Proc. 21st ACM Symp. on Principles of Database Systems*, June 2002.
19. D. Eppstein, Z. Galil and G. F. Italiano. Dynamic Graph Algorithms. In *Algorithms and Theoretical Computing Handbook* , M.J. Atallah, Ed., CRC Press, 1999.
20. H. N. Gabow. Data Structure for Weighted Matching and Nearest Common Ancestor with Linking. In *Proc. 1st Annual ACM Symp. on Discrete Algorithms* , Jan. 1990, 434–443.
21. P. Fraigniaud and C. Gavoille. Routing in trees. In *Proc. 28th Int. Colloq. on Automata, Languages & Prog.*, LNCS 2076, pages 757–772, July 2001.
22. 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.
23. P. Fraigniaud and P. Gauron. D2B: A de Bruijn based content-addressable network. Theor. Comput. Sci. 355(1): 65-79 (2006).
24. J. Feigenbaum and S. Kannan. Dynamic Graph Algorithms. In *Handbook of Discrete and Combinatorial Mathematics*, CRC Press, 2000.
25. H. N. Gabow, J. L. Bentley and R. E. Tarjan. Scaling and related techniques for geometry problems. In *Proc. 16th Annual ACM Symp. on Theory of Computing*, May 1984.
26. C. Gavoille and C. Paul. Split decomposition and distance labeling: an optimal scheme for distance hereditary graphs. In *Proc. European Conf. on Combinatorics, Graph Theory and Applications*, Sept. 2001.
27. C. Gavoille and D. Peleg. Compact and Localized Distributed Data Structures. *J. of Distributed Computing* **16**, (2003), 111–120.
28. C. Gavoille, M. Katz, N.A. Katz, C. Paul and D. Peleg. Approximate Distance Labeling Schemes. In *9th European Symp. on Algorithms*, Aug. 2001, Aarhus, Denmark, SV-LNCS 2161, 476–488.
29. C. Gavoille, D. Peleg, S. Pérennes and R. Raz. Distance labeling in graphs. *J. of Algorithms* **53(1)** (2004), 85–112.
30. H. Harel. A linear time algorithm for lowest common ancestors problem. In *21st Annual IEEE Symp. on Foundation of Computer Science*, Nov. 1980.
31. M. Harchol-Balter, F. T. Leighton, and D. Lewin. Resource Discovery in Distributed Networks. *Proc. PODC* 1999, pp. 229-237.
32. J. Holm, K. Lichtenberg and M. Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. of the ACM* **48(4)**, (2001), 723–760.
33. H. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. on Computing*, **13(2)**, 1984, 338–355.
34. H. A. Jamrozik, M. J. Feeley, G. M. Voelker, J. Evans, A. R. Karlin, H. M. Levy, and M. K. Vernon. Reducing network latency using subpages in a global memory environment. *Proc. the 7th ACM Conference on Architectural Support for Programming Languages and Operating Systems*, 1996.
35. A. Korman. General Compact Labeling Schemes for Dynamic Trees. In *Proc. 19th Int. Symp. on Distributed Computing*, Sep. 2005.
36. M. Katz, N.A. Katz, A. Korman and D. Peleg. Labeling schemes for flow and connectivity. *SIAM Journal on Computing* **34** (2004),23–40.
37. A. Korman and D. Peleg. Labeling Schemes for Weighted Dynamic Trees. In *Proc. 30th Int. Colloq. on Automata, Languages & Prog.*, July 2003, SV LNCS.

38. A. Korman, D. Peleg and Y. Rodeh. Labeling schemes for dynamic tree networks. *Theory of Computing Systems* **37**, (2004), 49–75.
39. A. Korman, D. Peleg and Y. Rodeh. Constructing Labeling schemes through Universal Matrices. In *Proc. 17th Int. Symp. on Algorithms and Computation*, Dec. 2006.
40. S. Kannan, M. Naor, and S. Rudich. Implicit representation of graphs. In *SIAM J. on Discrete Math* **5**, (1992), 596–603.
41. E. P. Markatos and G. Dramitinos. Remote Memory to Avoid Disk Thrashing: A Simulation Study. *Proc. MASCOTS* 1996: 69-73.
42. D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. *Proc. 21st annual ACM symposium on Principles of distributed computing*, 2002.
43. Moni Naor and Udi Wieder. Novel architectures for P2P applications: the continuous-discrete approach. *Proc. SPAA* 2003, pp. 50-59.
44. D. Peleg. Proximity-preserving labeling schemes and their applications. In *Proc. 25th Int. Workshop on Graph-Theoretic Concepts in Computer Science*, June 1999.
45. 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.
46. D. Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM, 2000.
47. P. Powell. A further improved LCA algorithm. Technical Report TR90-01, University of Minneapolis, 1990.
48. 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.
49. I. Stocia, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. *Proc. ACM SIGCOMM* 2001, San Diego, CA, Aug. 2001.
50. D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences* **26(1)**, (1983), 362–391.
51. B. Schieber and U. Vishkin. On finding lowest common ancestors: Simplification and parallelization. *SIAM J. on Computing*, **17**, 1988, 1253–1262.
52. M. Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *J. of the ACM* **51**, (2004), 993–1024.
53. A. K. Tsakalides and J. van Leeuwen. An optimal pointer machine algorithm for finding nearest common ancestors. Technical Report RUU-CS-88-17, Department of CS, University of Utrecht, 1988.
54. M. Thorup and U. Zwick. Compact routing schemes. In *Proc. 13th ACM Symp. on Parallel Algorithms and Architecture*, pages 1–10, July 2001.
55. B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. Tech. rep., Univ. of California, Berkeley, 2001.