The CPS 296.2 Geometric Optimization Class Project: Survey on Routing in Doubling Metric Space

Albert Yu

April 09, 2007

1 Introduction

Finding a shortest path between any two nodes in a network have been studied over the past few decades. A lot of routing algorithms for distributed networks have been introduced. In this paper, we consider routing in a doubling metric space. The doubling dimension is defined as the minimum value of α such that any radius-r ball can be covered by at most 2^α radius-$(r/2)$ balls. Doubling metric space is a metric space which doubling dimension is a constant. If α is $O(\log \log n)$, a network has a low doubling dimension. Consider a large network graph G=(V, E). It is not practical to store all the information of other nodes in every node because of the limitation of space compacity. Therefore, a node must be able to deliver a packet to another node without knowing the whole picture of the graph. The effectiveness of a routing scheme is judged based on space compacity (storage per node) and stretch (the maximum ratio between the route and the shortest path among all n^2 possible source-destination pairs).

There are two routing models. In label routing model, each node is assigned a routing label which provides useful information about the location of the node in the network. When a packet is sent from one node to another one, their corresponding labels are specified instead of their node IDs. One famous example of labeled routing is IP address. In name-independent model, the node ID (original name) is specified as a destination instead of the node label. In order to send a message to the destination, we first have to identify the correspoinding label for the destination node. Such information can be stored in any node in the network. Hence, we need to be able to identify the node which store the label of the destination and route to there in order to retrieve the label. Then the message is directly sent from there to the destination node. This is a two-phase procedure.

Routing in name independent scheme is obviously more difficult, but in many applications including online tracking of mobile users [2], we cannot predefine the routing label in the pre-processing phase. In mobile network setting, users are allowed to move from one node to another and we have to keep track of their current locations so that each user can contact another user. It is not possible to predefine a routing label for each user.

If each node of degree d can stores $O(n \log d)$ bits, we can route to the destination node with a minimum spanning tree rooted at the source. At each intermediate nodes on the path, we use the routing table to identify the next hop. This stretegy can be applied to any graph and we call it a univeral routing scheme. The challenge is whether or not we can find a more compact (defined in section 3) scheme that use less than

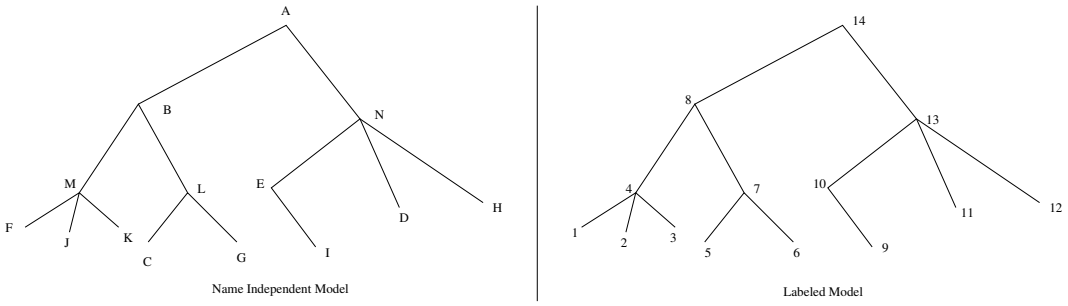

Figure 1: In name independent scheme, if node b wants to send a message to node d, it needs to find out the routing label of node d which is 11, then the message can be sent through the underlying labeled scheme.

$O(n \log d)$ bits per node. $\Omega(n)$ bits per node cannot be avoided if nodes are not allowed to be renamed since we have to store at least 1 bit to indicate the direction of the next hop for each v.

2 Outline

Basic definitions and techniques used in labeled and name independent routing schemes are presented in Sections 3 and 4, respectively. In section 5, we will discuss the existing labeled routing schemes. In section 6, we will present the existing name-independent schemes. In section 7, we will discuss existing scale-free routing schemes for both labeled and name-independent models. Section 8 is the conclusion.

3 Preliminaries

Definition 1 ***Aspect ratio****, $\triangle$, is defined as the ratio between the maximum distance between any source-destination pairs and the minimum distance between any source-destination pairs. If the minimum distance is normalized to 1, then the aspect ratio is simply the weighted diameter of the network. A routing scheme is called scale-free if its storage compacity does not depend on the aspect ratio.*

Definition 2 *A **compact routing scheme** is the scheme that optimizes the stretch but limits the storage per node and the packet header size to polylogarithmic in the number of nodes.*

Definition 3 *A routing scheme is **scale-free** if its routing tables, packet headers, and routing labels require memory which does not depend on $\triangle$.*

Let $d(u, v)$ denote the distance between 2 nodes u and v.

Definition 4 *r-**net** of G is defined as a set of nodes $V' \subseteq V$ such that (1) every pair of nodes in the r-net are seperated by at least r units and (2) for any node $v \in V \backslash V'$, there exists a node $v' \in V'$ such that $d(v, v') < r$.*

r-net can be constructed with a greedy method.

Definition 5 *A ball $B_v(r) = B(v,r)$ is defined as a set of nodes which distance is at most r from node v.*

The following theorem [5] is well-known for a space of bounded doubling dimension:

Theorem 1 *Let $B(v,r)$ to be a set of nodes which distance is at most r from node v. Let Y be an r-net of X and α be the doubling dimension of a metric space (X,d), $|B(x,r') \cap Y| \leq (4r'/r)^{\alpha}$ for any $x \in X$ and any $r' \geq r$.*

Definition 6 (Rings of neighbors) *A set of neighbors of node v is partitioned into rings. The i^{th} ring of v's neighbors is defined as the intersection of the ball $B_v(i)$ centered at v with radius i and $(\triangle/2^i) - net$. For each ball $B_v(i)$, only the i^{th} ring of neighbors will be visited.*

Let $r(u,j)$ denote the minimum radius of a ball centered at u such that the ball contains 2^j nodes.

Definition 7 *Ball packing is a maximal set such that for any $j \in [1,2,\cdots,\log n]$, there exists a ball packing BP_j of G such that*

1) If $B_1 \in BP_j$ and $B_2 \in BP_j$, $B_1 \cap B_2 = \emptyset$.
2) All the balls $B \in BP_j$ contain 2^j elements.
3) For any node u, there exists a node c such that $B_c \in BP_j$ with $r(c,j) \leq r(u,j)$ and $d(u,c) \leq 2r(u,j)$

Such a packing can be constructed in greedy manner.

4 Common techniques

4.1 Hierarchical routing [2, 4]

This strategy relies on generating a sparse cover of clusters. A thumb rule is that the more clusters are involved, the more expensive is the routing cost. We want to have minimum interactions between clusters. However, even if two nodes are close to each other, they may be assigned to different clusters. As a result, we want to have a locality preserving network covers. Clusters may or may not be overlapped. The sparity of a cover can be defined as the number of occurence of a node in the cover. Small clusters require less time complexity because the shortest path trees that span the clusters are small. Low sparsity, on the other hand, requires low memory.

Clustering techniques can be used in conjunction with a hierarchical structure. For metric (X,d), we generate a set of covers of X with geometrially decreasing diameters. Clusters at higher levels are responsible for long distance routing and clusters at lower levels are for short distance routing when the destination node is nearby. Suppose we want to deliver a message to destination node t. The message is first sent at the lowest level. If t is not found in the cluster, the message is then sent at a higher level. Geometric series guarantee that the total

length of a path along the hierarchy tree is bound by $r + r/c + r/c^2 + \cdots + 1 = O(r)$. Hierarchical routing allowes us to organize the communication of an application at the "right" level depending on the locality level of the application. If nodes alway communicate with one another in a small neigborhood, then only clusters with small diameters are involved.

4.2 Interval-based label

We start from the root of a tree and assign a label to all its descendants by traversing the tree in DFS post-order. Each node v stores the range of the label number in the subtree rooted at v. Refer to figure 1. Node B stores the range [1, 8] and node L stores the range [5, 7]. A node v is a descendant of u if $range(v) \subset range(u)$. Each node stores a local routing table and the destination address,$Address(u)$, is defined as $(ID(u), label(u))$.

5 Labeled routing schemes

Stretch is 1 if we are allowed to use labels of $O(n^2 \log n)$ bits since we can simply encode the optimal path into labels. However, we usually only encode labels with $O(\log n)$ bits. For the compact scheme, $1+\epsilon$ stretch is optimal and there are several papers which have achieved this.

Algorithm 1:

Talwar [10] used randomized split-trees to form a hierarchical decomposition with a top down approach. The randomized split trees have some nice properties likes those of randomly shifted grids in Euclidean spaces. They proved that the probability that u and v are assigned to the same cluster at level i is $O(k)d(u, v)/2^i$.

Stretch:	$1 + \epsilon$
Label/header (bits) :	$O(\alpha \log \triangle)$
Storage per node (bits):	$O((1/\alpha\epsilon)^\alpha \log^{2+\alpha} \triangle$

Algorithm 2:

Chan et al. [3] also obtained $1+\epsilon$ stretch based on the hierarchical clustering model. However, they built probabilistic hierarchical decompositions with a bottom up approach that would result in $\tilde{O}(\alpha)$ levels in contrary to $O(\log n)$ levels of a top down approach because their approach could avoid long range dependency. They associated hierarchical decompositions with a stretch-$O(\alpha/\epsilon)$ steiner tree cover and their analysis involved Lovasz Local lemma which states that given a large number of events, if each event is mutually independent of all the others, then the probability that none of the events will occur is very small.

Stretch:	$1 + \epsilon$
Label/header (bits) :	$O(\alpha \log(1/\epsilon) \log \triangle)$
Storage per node (bits):	$(\alpha/\epsilon)^{O(\alpha)} \log \triangle \log n$

Algorithm 3:

Slivkins [9] used rings of neighbors techniques to improve the local storage.

Stretch:	$1+\epsilon$
Label/header (bits) :	$O(\alpha \log(1epsilon) \log \triangle)$
Storage per node (bits):	$(1/\epsilon)^{O(\alpha)} \log \triangle \log n$

Algorithm 4 :

Abraham et al. [1] obtained the following result.

Stretch:	$1+\epsilon$
Label/header (bits) :	$\lceil \log n \rceil$
Storage per node (bits):	$(1/\epsilon)^{O(\alpha)} \log \triangle \log n$

For the rest of this section, we will sketch Algorithm 4.

5.1 Hierarchical net

A hierarchical net is constructed as follow: Let $X_\triangle$ be $\{v\}$ in which v is an arbitrary point in V. X_i is a (2^i)-net of G and it is built by greedily expanding X_{i+1}: All the nodes that are at least 2^i away from any nodes in X_{i+1} are added into the (2^i)-net. By construction, X_0 (the lowest level of the netting tree) contains all the nodes. Then Every node in X_i is connected to the closest node in X_{i+1}.

Construction of a hierarchical net

1: $X_\triangle$ = $\{v\}$, where v is an arbitrary point.
2: **for** ($r = \triangle - 1$; $r \geq 0$; $r--$)
3: $X_r = X_{r+1}$
4: **for all** $u \in V \backslash X_{r+1}$
4: **if** $d(u,x) \geq 2^i$ for all $x \in X_{r+1}$
5: $X_{r+1} = X_{r+1} \cup u$
6: **endif**
7: Connect every node in X_r to the closest node in X_{r+1}
8: **end forall**
9: **end for loop**

By construction, the same node may appear in multiple levels. If x is not in level i of the hierarchical net, it has a parent x' at level i and $d(x,x') \leq 2^{i+1}$.

Next, we applied the interval-based label techniques (introduced in the previous section). Let $x_1, x_2, \cdots, x_m$ be a DFS post-order traversal of the nodes in the hierarchical net. We assign label 1 to the first element, label 2 to the second element, and so on. For any node that appears in multiple levels, only the one in the highest level is assigned a label. Since there are at most n labels, each label takes $O(\log n)$ bits. Each node v at level i maintains $R_i(v)$, the range of the DFS traversal numbering of its subtree. For all $i \in [1, 2, \cdots, \log \triangle]$, define

the i^{th} ring $Y_i(x) = B(x, 2^{i+1}) \cap X_i$. A node x either belongs to $Y_i(x)$ or there exists a node $x' \in Y_i(x)$ such that the range $R_i(x')$ contains $Label(x)$.

5.2 $(1+\epsilon)$ stretch

Each node v maintains $(R_i(x), Label(x))$ pairs for all $x \in B_v(2^{i+1}) \cap X_{i-3-\log(1/\epsilon)}$ and $i \in [1, 2, \cdots, \log \triangle]$.

Assume we want to deliver a message from node s to node t. For each intermediate node v on the path between s and t, when v recieves the message, using the local routing table and the label of t, v is able to find a net point $v' \in X_{i-3-\log(1/\epsilon)}$, where the subtree of v' contains the destination node t. Since v' is at most $2^{i-3-\log(1/\epsilon)+1} = (\epsilon/4)2^i$ away from any node in its subtree, thereby $d(v', t) \leq (\epsilon/4)2^i$. If $2^i < d(s, t) \leq 2^{i+1}$,

$$d(v', t) \leq (\epsilon/4)d(s, t). \tag{1}$$

Therefore, the message is guaranteed moving towards the destination node at each step.

Consider the path from s to t: $s = x_0, x_1, x_2, \cdots, x_p = t$.

$$\begin{aligned}
& d(x_{i-1}, x_i)/(d(x_{i-1}, t) - d(x_i, t)) \\
\leq\ & (d(x_{i-1}, t) + d(x_i, t))/(d(x_{i-1}, t) - d(x_i, t)) && \text{because of triangle inequality} \\
\leq\ & (d(x_{i-1}, t) + (\epsilon/4)d(x_{i-1}, t))/(d(x_{i-1}, t) - (\epsilon/4)d(x_{i-1}, t)) && \text{because of (1)} \\
=\ & (1+\epsilon/4)/(1-\epsilon/4) \leq 1+\epsilon, && \text{where } \epsilon \leq 2.
\end{aligned}$$

Therefore, $\sum_{i=1}^{p} d(x_{i-1}, x_i) \leq (1+\epsilon)\sum_{i=1}^{p}(d(x_{i-1}, t) - d(x_i, t)) \leq (1+\epsilon)d(s, t)$.

Storage: By theorem 1, at most $2^{O}(\alpha \log(1/\epsilon))$ labels need to be maintained at each level. There are $\log \triangle$ levels and each label takes $O(\log n)$ bits, therefore, the total number of memory required at each node is $(1/\epsilon)^{O}(\alpha) \log n \log \triangle$.

6 Name-Independent Scheme

In this section, we will describe two name independent schemes. The first one was provided in [1] which has $O(1)$ stretch. The second one was provided in [6] which has the optimal $9+\epsilon$ stretch.

6.1 Independent scheme 1

We first form a collection of clusters, $C_r(G)$, for each $r = 2^i$ and $i \in [1, 2, \cdots, \log \triangle]$ such that

(1) There always exists a cluster which covers all the nodes in $B_v(r)$ for each $v \in V$, i.e. $\exists C, B(v, r) \subseteq C$;

(2) Each node v appears in at most 4^{α} clusters, where α is the doubling dimension;

(3) No cluster has radius that is greater than $2r$. The radius of a cluster is defined as the minimum radius of the ball that contains all the elements in the cluster.

Abraham at el. proved that such a collection of clusters could be constructed by finding a r-net U and setting the clusters to be the radius-$(2r)$ balls centered at u for all $u \in U$.

In each cluster C, a local tree is constructed as follow: Let U_0 be $\{v\}$ where v is an arbitrary point in C. U_{i+1} is a $(r/2^{i+1})$-net of $C \setminus \cup_{0 \le j \le i} U_j$. Every node in U_{i+1} is connected to the closest node in U_i. Since the tree can contain edges not in G, it may not be a subgraph of G, but we can employ the labeled $(1+\epsilon)$ stretch routing scheme presented in the previous section.

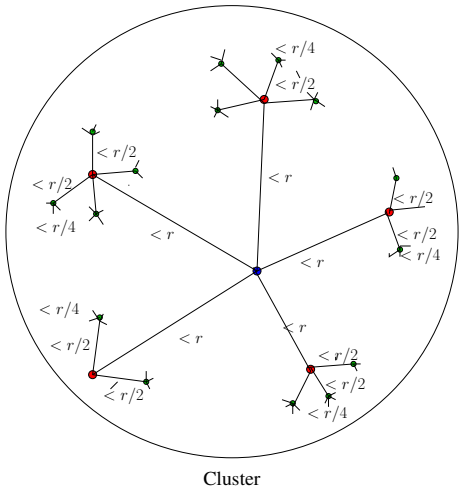

Figure 2: The resulted tree has two properties. First, the diameter of a tree is at most $4r$ because the length of a path from a leaf to the root is bounded by $1+2+4+8+\cdots+r/2+r = 2r-1$. Second, a node cannot have more than 4^{α} children.

Local routing within each cluster:

The underlying labeled scheme is the same as the one described in the previous section. However, in order to use the labeled scheme, we must be able to figure out the routing label of the destination node in the name-independent scheme.

Let $L = u_1, u_2, \cdots, u_n$ be a sorted list of IDs (original names) according to lexicographical order. We store 2 things at each node v:

1) Lets j be the label number of i. We store the label of the j^{th} smallest node at node i. Therefore, each node i stores $label(u_{label(i)})$.

2) Let $Address(u)$ denotes $< u, label(u) >$. Each node v stores $Address(label(v))$ and $Address(label(v'))$, where v' are the children of v.

In order to send a message from u_s to the destination node u_t in the same cluster, we first route to the root and then use $Address(d(\cdot))$ to navigate down to the node which stores the label of u_t. From there we can route to the destination node u_t directly. If the destination node does not exist in the cluster, the message is sent back to the source node s through the root. We start with radius-1 cluster. If radius-$(r/2^i)$ cluster does not contain t, then search into radius-$(r/2^{i-1})$ cluster.

Analysis for storage and stretch:

Storage: There are $\log \triangle$ collections. For each collection, a node can appear in at most 4^α clusters. Each node in a cluster maintains at most 4^α labels by theorem 1. Each label has $\log n$ bits. The memory requirement for each node is, therefore, $(\log \triangle)4^{2\alpha}O(\log n)$ bits. By choosing the labeled scheme in the previous section, we get $2^O(\alpha) \log \triangle \log n$.

Stretch:

The cost of routing to a cluster at level j is bounded by 2^j. The cost of searching the label of the destination node in a cluster at level j is 2^{j+3} because the diameter of the local search tree is bounded by $4 * 2^j$ and the underlying labeled scheme has stretch less than 2. The cost of routing to the destination (when the label is found) in a cluster at level j is also bounded by $4 * 2^j$ for the same reason. Therefore, the cost of searching in a cluster at level j = cost of routing to the cluster + cost of searching the label in the cluster + cost of routing to the destination = $2^j + 2^{j+3} + 2^{j+3} \leq 2^j + 4$.

Suppose $2^{i-1} < d(s,t) < 2^i$. The node that stores the label of t can found in a cluster at level i. The cost of the name-independent scheme is $\sum_{j=0}^{i} 2^{j+4} = 2^{i+5} - 1 < 2^6 d(s,t)$. Therefore, the stretch is $64 = O(1)$.

6.2 Independent scheme 2

Konjevod et al. independently developed another name-independent scheme which improved the result of stretch from $O(1)$ to $9+\epsilon$. They introduced the concept of neighbor sequence in the paper. Still, the schemes of Konjevod et al. and Abraham et al. (previous section) share a lot of similarities.

Their scheme also has $\log(\lceil \triangle \rceil)$ levels in the hierarchy. The neighbors of node u in i-net, $n(u,i)$, are recursively defined such that:

1) $n(u,0) = u$.

2) for $0 < i \leq \log \triangle$, $n(u,i)$ is a node in the i-net such that $d(n(u,i-1), n(u,i))$ is minimized.

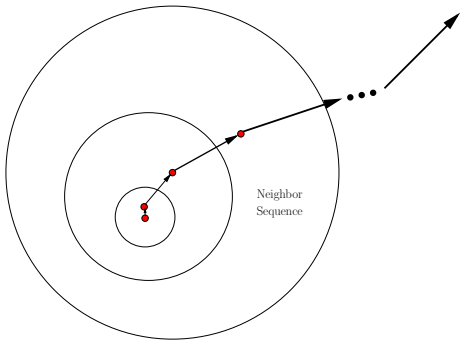

Figure 3: The length of the neighbor sequence is $\sum_{i=1}^{\log \triangle} d(n(u,i-1), n(u,i)) \leq \sum_{i=1}^{\log \triangle} 2^i < 2^{\log(\triangle)+1}$.

The sequence $n(u,0), n(u,1), \cdots, n(u, \log \triangle - 1)$ is called the *neighbor sequence* of node u. The *local area* of node u at level j, $L(u,j)$, is defined as $B(u, 2^j * f(\delta))$, where $f(\delta) = 80/\delta + 2$.

The local search tree is defined differently in this paper. In the previous subsection, all the nodes in the cluster

form a netting tree of size n' where n' is the number of nodes in the cluster and nodes in the cluster may be internal nodes or leaves of the search tree. However, in this paper all the nodes in the cluster are the leaves of the local search tree. For any node $u \in Y_j$ where $0 < j \leq M$, a local search tree $T(u, j)$ is defined as follow:

1) u is the root.

2) Every element $w \in L(u, j)$ is a leaf of $T(u, j)$

3) $\forall w \in L(u, j)$, the path between w and u is the neighbor sequence of w: $w \rightarrow n(w, 1) \rightarrow (w, 2) \rightarrow \cdots \rightarrow n(w, j-1) \rightarrow u$. We use $p(w, u, j)$ to denote the path between w and u.

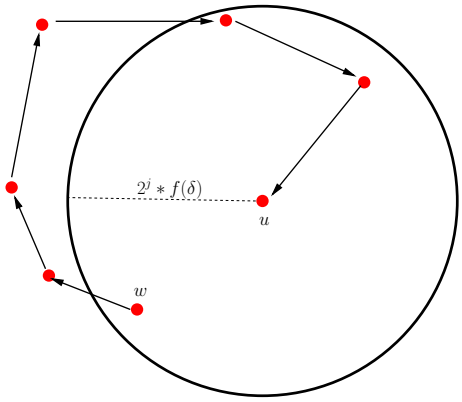

Figure 4: Note that a path of the tree $T(u, j)$ is not necessarily contained within the local region $L(u, j)$. However, even if a node in the local tree is outside the local region, it must still be very close to the local region because the length of any path $p(w, u, j)$ in the tree is bound by $2^j(f(\delta) + 2)$.

The length of any path in the local search tree $T(u, j)$ is:

$$
\begin{aligned}
& |p(w, u, j)| \\
&= \textstyle\sum_{i=1}^{j-1} d(n(w, i-1), n(w, i)) + d(n(w, j-1), u) \\
&\leq A(w, j-1) + d(n(w, j-1), u), && \text{since } A(w, j) = \textstyle\sum_{i=1}^{j} d(n(u, i-1), n(u, i)) \\
&\leq A(w, j-1) + d(n(w, j-1), w) + d(w, u) && \text{because of triangle inequality} \\
&\leq A(w, j-1) + A(w, j-1) + d(w, u) && \text{since } d(n(w, j-1), w) \leq A(w, j-1) \text{ by def.} \\
&\leq 2A(w, j-1) + 2^j f(\delta) && \text{since } w \in B_u(2^j f(\delta)) \\
&\leq 2^j(f(\delta) + 2) && \text{since } A(w, j-1) < 2^j
\end{aligned}
$$

Local search:

The underlying label scheme is a DFS post-order enumberation of the leaves of the search tree. We simply state that the cost of searching in $L(u, j)$ in the worst case is $2*$ length of the longest path $= 2 * 2^i(f(\delta) + 2)$.

Routing algorithm:

Assume we want to send a message from u to v.

Phase 1: Do a local search at its neighbor $w = n(u, i)$. If it is found in the local area $L(w, i)$, go to phase 2. Else, set $i = i + 1$ and repeat phase 1 again.
Phase 2: Go to v from w directly using the underlying labeled routing scheme in section 5.

Stretch and storage analysis:

By choosing the labeled scheme in the previous section, the underlying labeled scheme has $(1 + \delta')$ stretch. We want to send a message from u to v. Assume v is found at level j in phase 1.

The shortest path between u and v is:

$d(u, v) \geq d(n(u, j-1), v) - d(n(u, j-1), u)$ because of triangle inequality

$\geq d(n(u, j-1), v) - A(u, j-1)$

$\geq f(\delta)2^{j-1} - 2^j$ since v cannot be found in $B_{n(u,j-1)}(f(\delta)2^{j-1})$

$= 2^{j-1}(f(\delta) - 2)$

Given the cost of the local search at level i is $2^{i+1}(f(\delta) + 2)(1 + \delta')$, the routing cost is: [routing cost to neighbor $n(u, i)$ for $1 \leq i \leq j$] + [routing cost for searching in the local region $L(n(u, i), i)$ for $1 \leq i \leq j$] + [routing cost from $n(u, j)$ to v]

$\leq (A(u, j) + \sum_{i=0}^{j} 2^{i+1}(f(\delta) + 2) + d(n(u, j), v))(1 + \delta')$

$\leq (A(u, j) + \sum_{i=0}^{j} 2^{i+1}(f(\delta) + 2) + d(n(u, j), u) + d(u, v))(1 + \delta')$ because of triangle inequality

$\leq 2^{j+1} + \sum_{i=0}^{j} 2^{i+1}(f(\delta) + 2) + 2^{j+1} + d(u, v))(1 + \delta')$ since $d(n(u, j), u) \leq A(u, j) < 2^{j+1}$

$\leq (2^{j+2}(f(\delta) + 3) + d(u, v))(1 + \delta')$

$\leq (d(u, v)/2^{j-1}(f(\delta) - 2) * 2^{j+2}(f(\delta) + 3) + d(u, v))(1 + \delta')$ since $d(u, v) \geq 2^{j-1}(f(\delta) - 2)$

After simplifying the inequality, we get the routing cost:

$\leq (9 + 40/(f(\delta) - 2))d(u, v)(1 + \delta')$

$\leq (9 + \delta)d(u, v)$ if we set $f(\delta) = 80/\delta + 2$ and $\delta' = min\delta/20, 1/10 < 1/2$.

7 Scale-free routing

Since the aspect ratio $\triangle$ can be exponential in n, it is desirable to remove the independency on $\triangle$. In this section, we will first discuss a scale-free routing scheme for metric spaces. Then we will discuss a scale-free routing scheme for graphs.

7.1 Scale-free routing for metric space

The scheme in this subsection is based on component trees and rings techniques.

Component trees were introduced in [11] by Thorup. He used component tree data structures to get a linear time algorithm for single source shortest path. The idea is to use hierarchical bucketing structures to avoid comparison sort which has the well known lowerbound $\Omega(n \log n)$. The component tree techniques can also be applied to labeled routing as shown in [1]. We sketch the algorithm below.

A component hierarchy is defined as follow: Denote G_i to be a subgraph of G such that all the edges e of length at least 2^i are removed from G_i. The components at level i are the components in G_i. The root is G and the leaves (at level 0) are the singleton subgraphs. A component C at level i is the parent of component C' at level i' (where $i' < i$) if and only if 1) $C' \subset C$ and 2) there does not exist C'' such that $C' \subset C'' \subset C$. There are at most $2n - 1$ nodes and a component tree can be constructed in linear time and space.

For a component C, let $L(C)$ be the maximum integer such that at least one edge in C has length greater than or equal to $2^{L(C)}$. This definition implies that the maximum length of any edge in C is $2^{L(C)}$ (assuming the length of every edge is an integer). Thus, the length of any simple path in C is less than or equal to $2^{L(C)}(n-1)$.

From now on, we use "node" to mean "a node in the component tree" and "vertex" to mean "a node in the graph". The next step of the construction is to associate each internal node C of the component tree with a vertex v with the following restrictions:

1) $v \in C$ (The singleton component $\{v\}$ is a leaf of C's subtree)
2) Each vertex $v \in V$ is assigned to at most one component node.
3) The assignment is done in the bottom-up fashion: An internal node C remains unassigned until all the internal nodes in C 's subtree are assigned with vertices.

After that, we can construct a routing tree on a subset of vertices $v \in V$ as follow: A vertex v_1 is a parent of another vertex v_2 iff 1) v_1 is assigned to component C_1, 2) v_2 is assigned to component C_2 and 3) C_1 is the parent of C_2 in the component tree.

Define a ring $R(u, i, c)$ as $B_u(2^{i+2}) \cap X_{i-c}$. Each vertex u maintains a set of rings $R(u, i, c)$ for all $L(u) \leq i \leq L(u) + 3\log n + 5 + \log(1/\epsilon)$. If u is assigned to a component node C, then u also maintains another set of rings $R(u, i, c)$ for all $L(C) \leq i \leq L(C) + 3\log n + 5 + \log(1/\epsilon)$

The routing idea is to climb up the component tree until the destination vertex t is found.

Scale-free label routing

```
1: Suppose we want to send a message from s to t.
2: u = s; i = 0;
3: while ( 1)
4:         if ( t is found in the first set of rings R(u, i , c ))
5:                 return  t;
6:         else if ( u is assigned to a component )
7:                 if ( t is found in the second set of rings R(u, i , c ))
8:                         return  t;
9:         else
10:                 u = p(u); i ++;
11:          endif
12: end while
```

Storage: Number of rings $\times$ number of bits per ring $\leq 2(3\log n + 5 + \log(1/\epsilon)) \times (1/\epsilon)^{O(\alpha)} \log n = (1/\epsilon)^{O(\alpha)} \log^2 n$ bits.

$1+\epsilon$ **Stretch**: If the shortest path $d(u,t) \leq n^3 2^{L(C)} 2^5/\epsilon$, then there exists a ring in the first set which contains a net point that is closer to the destination by a factor of ϵ. The proof is similar to the proof in section 5. On the other hand, if $d(u,t) > n^3 2^{L(C)} 2^5/\epsilon$, we have to climb up the component tree, but since the cost of routing from u to its parent $p(u)$ is $n * 2^{L(C)}$, we can repeat this n times to get to the root with the cost:

$n^2 2^{L(C)}$

$= n^3 2^{L(C)} 2^5/\epsilon * \epsilon/(n2^5)$

$< d(s,t)\epsilon/(2^5 n)$

7.2 Scale-free routing for graph

Abraham et al. also provided an optimal-stretch scale-free labeled scheme in [1]. The scheme relies on sparse-dense decomposition: The dense region is handled separately from the sparse region with a different routing scheme.

Labeled scheme:

Stretch:	$1+\epsilon$
Label/header (bits) :	$2^O(\alpha) \log^3 n$
Storage per node (bits):	$2^{O(\alpha)} \log^3 n$

Name-independent scheme:

Stretch:	$O(1)$
Label/header (bits) :	$2^O(\alpha) \log^3 n$
Storage per node (bits):	$2^{O(\alpha)} \log^4 n$

However, the stretch has a very large constant for the name independent scheme and this makes the algorithms impractical, especially for high speed networks. Konjevod et al. [7] simplified the proof with the ball-packing technique. It also requires less memory for routing labels.

Labeled scheme:

Stretch:	$1+\epsilon$
Label/header (bits) :	$O(\log^2 n/\log\log n)$
Storage per node (bits):	$(1/\epsilon)^{O(\alpha)} \log^3 n$

Name-independent scheme:

Stretch:	$9+\epsilon$
Label/header (bits) :	$O(\log^2 n/\log\log n)$
Storage per node (bits):	$(1/\epsilon)^{O(\alpha)}\log^3 n$

For the rest of this section, we will sketch Konjevod's algorithms.

The ball packing technique (defined in section 3) can be used to solve the scale-free name-independent scheme. The main idea is that if the scheme does not depend on $\triangle$, each node cannot store all the information of all $\log\triangle$ layers of a r-net hierarchy. In the paper, only $O(\log n)$ layers' information is maintained and the rests are solved by packing balls. The routing algorithm contains two phases: Phase 1 uses the r-net hierarchy; Phase 2 uses the ball-packing techniques.

Labeled scheme

For each $j \in [1, 2, \cdots, \log N]$, find the ball packing BP_j. Then construct a voronoi diagram, VD, in which the nodes are the centers of the balls in BP_j. After that, for each $VD(c, j)$ (the voronoi region of the ball B_c at level j), construct a shortest path tree rooted at c and every nodes in $VD(c, j)$ store the routing label of c.

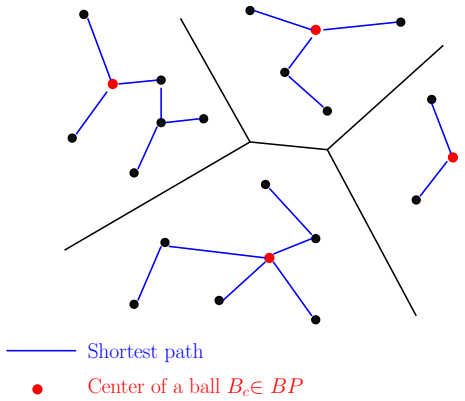

Figure 5: For each ball $B_c \in BP_j$, the corresponding Voronoi region has a shortest path tree rooted at c which spans $V(c, j)$.

The labeled routing scheme is as follow:

Define $Range(u, i)$ to be a set of nodes in the subtree rooted at u and u is in the i-net of the netting tree. If $v \in Range(u_1, i_1)$, $v \in Range(u_2, i_2)$, and $i_2 \leq i_1$, then $d(v, u_2) \leq d(v, u_1)$. Each $u \in V$ stores $Range(u, i)$ for all $i \in [1, 2, \cdots, \log n]$.

Labeled routing scheme

1: Suppose we want to send a message from s to t.
2: $u_0 \leftarrow s$, $i_{-1} \leftarrow \infty$
3: ========== **Phase 1** ==========
4: **for** $k = 0$ to ∞
5: Find a vertex x_k which minimizes i_k subject to $t \in Range(x_k, i_k)$

6: **if** $i_k \leq i_{k-1}$ and u_k and t are not in the same voronoi region $\in BP_k$
7: $u_{k+1} \leftarrow$the next hop along the shortest path between u_k and x_k
8: Go from u_k to u_{k+1}
9: **else**
10: break
11: **end if**
12: **end for loop**
13: ========== **Phase 2** ==========
14: Let j be the current level of the netting tree and we are at node z.
15: Go to the center c of a ball $B_c \in BP_j$ which corresponding Voronoi region $VR(c, j)$ contains z.
16: Route to t using the shortest path tree.

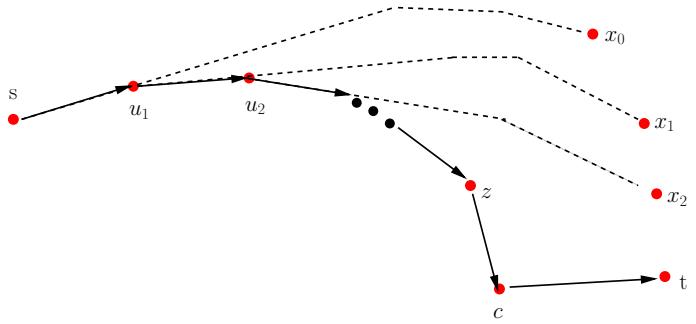

Figure 6: u_{k+1} is the first node along the shortest path from u_k to x_k. When u'_k and t are in the same voronoi region of level j, we can use the shortest path tree within the voronoi region to go the root c and from there we can route to the destination t.

Name-independent scheme

The data structures is the same as those in section 6.2 except we maintain local search trees for two types of balls:

1) Any balls $B_c(r(c, i+2))$ in ball packing BN for all $i \in [1, 2, \cdots, \log(\lceil n \rceil)]$
2) Any balls $B_u(2^i(80/\epsilon + 2))$ for all $i \in [1, 2, \cdots, \log(\lceil \triangle \rceil)]$ and all $u \in i$-net. However, if a ball of type 1 approximately covers the elements in the ball B_c, we simply provide a pointer to node c instead of building a search tree.

Search Algorithm for $B_u(r)$, where radius $r = 2^i(80/\epsilon + 2)$

1: Suppose we want to search for node which ID is w.
2: **if** $B_u(r)$ is in the collection of balls of type 2,
3: search w in the tree $T(u, r)$
4: **else**
5: find a ball of type 1, B_c, from ball packing BN
6: route to node c using the underlying labeled scheme.
7: let r' be the radius of B_c
8: search w in the tree $T(c, r')$

9: route back from c to u
10: **endif**

They proved that each node v is in at most $(1/\epsilon)^{O(\alpha)} \log n$ search trees and the memory requirement per node is $(1/\epsilon)^{O(\alpha)} \log^3 n$ bits.

The stretch is $9 + \epsilon$. The proof is the same as the proof in section 6.2.

8 Conclusion

A lot of research papers have been published for labeled and name-independent compact routing in doubling metrics. Both the optimal-stretch name-independent scheme and the optimal-stretch labeled scheme are known. One future direction is to study a relaxed version of the routing problem. In [8], the authors studied two relaxed version: 1) a small fraction of nodes are allowed to have large storage; 2) A small fraction of source-destination pairs has a larger stretch.

References

[1] I. Abraham, C. Gavoille, A. V. Goldberg, and D. Malkhi. Routing in networks with low doubling dimension. In *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems*, page 75, 2006. 5, 6, 11, 12

[2] B. Awerbuch and D. Peleg. Sparse partitions. In *31th Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society Press*, 1990, pp. 503-513. 1, 3

[3] H. T.-H. Chan, A. Gupta, B. Maggs, and S. Zhou. On hierarchical routing in doubling metrics. In *Proceedings of the 16th ACM-SIAM Symposium on Discrete Algorithms*, pages 762–771, 2005. 1993. 4

[4] C. Gavoille. Routing in distributed networks: Overview and open problems. *ACM SIGACT News - Distributed Computing Column*, vol. 32, no. 1, pages 36–52, 2001. 3

[5] A. Gupta, R. Krauthgamer, and J.R. Lee. Bounded geometries , fractals and low-distortion embeddings. In *Proceedings of teh 44th IEEE Symposium on Foundations of Computer Science*, pages 534-543, 2003. 3

[6] G. Konjevod, A. Richa, and D. Xia. Optimal-stretch name-independent compact routing in doubling metrics. In *Proceedings of the 25th ACM Symposium on Principles of Distributed Computing*, pages 198-207, 2006. 6

[7] G. Konjevod, A. Richa, and D. Xia. Optimal scale-free compact routing schemes in network of low doubling dimension. In *Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms*, 2007. 12

[8] G. Konjevod, A. Richa, D. Xia, and H. Yu. Compact Routing Schemes with Relaxed Guarantees in Networks of low Doubling Dimension. Submitted to PODC 2007. 15

[9] A. Slivkins. Distance estimation and object location via rings of neighbors. In *Proceedings of the 24th ACM Symposium on Principles of Distributed Computing*, pages 41-50, 2005. 5

[10] K. Talwar. Bypassing the embedding: algorithms for low dimenisonal metrics. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, pages 281-290, 2004. 4

[11] M. Thorup. Undirected single-source shortest paths with positive integer weights in linear time. In *Journal of the ACM*, 46:362-394, 1999. 11