1. Routing with Succinct Tables
2. Constructing Sparse Spanners
3. Routing v.s. Spanners
- Lecture 1 -

Routing with Succinct Tables

An introduction to Compact Routing, and some algorithmic tools for constructing routing schemes
Outline

Context and Motivations

Models, Definitions and Examples

A First Compact Routing Scheme

A First Universal Compact Routing Scheme

Concluding Remarks
Outline

Context and Motivations

Models, Definitions and Examples

A First Compact Routing Scheme

A First Universal Compact Routing Scheme

Concluding Remarks
Message Routing in the Internet

1969. Arpanet connects \( n = 4 \) nodes.


2009. Growth rate for forwarding information base (FIB) size is \( 1.3 \) a year. 175 KB in jan 2006; 300 KB in jan 2009; 1 MB in 2011 (estimate).
1969. Arpanet connects $n = 4$ nodes.
1969. Arpanet connects $n = 4$ nodes.

1969. Arpanet connects $n = 4$ nodes.


2009. Growth rate for *forwarding information base (FIB)* size is 1.3 a year.

- 175 KB in jan 2006;
- 300 KB in jan 2009;
- 1 MB in 2011 (estimate).
Main router builder companies (Alcatel Lucent Bell, Cisco, Nortel) are studying for new routing data organizations to lower the growth rate size of FIBs.

Today the size of FIBs generated by routing protocols (or routing strategy) is linear to $n$, the number of autonomous systems (AS).
Outline

Context and Motivations

Models, Definitions and Examples

A First Compact Routing Scheme

A First Universal Compact Routing Scheme

Concluding Remarks
Statement of the Compact Routing Problem

**Input:** a network $G$ (a weighted connected graph)

**Output:** a **routing scheme** for $G$

A **routing scheme** is a distributed algorithm that allows **any** source node to route messages to **any** destination node, given the destination’s network identifier.
Statement of the Compact Routing Problem

**Input:** a network $G$ (a weighted connected graph)

**Output:** a **routing scheme** for $G$

A **routing scheme** is a distributed algorithm that allows **any** source node to route messages to **any** destination node, given the destination’s network identifier.

**Goal:** to minimize the size of the routing tables
Model: $R_{x_i}(h_i, q_i) = (p_i, h_{i+1})$

- addresses
- headers
- in/outport numbers
Example: Grid with X,Y-coordinates

Routing algorithm: X,Y-routing
(local decision)
Example: Grid with X,Y-coordinates

Routing algorithm: X,Y-routing
(local decision)
Another Example ...
Complexity Measures: Space & Stretch

Space = size of the largest local routing tables
Complexity Measures: Space & Stretch

Space = size of the largest local routing tables
       (more precisely, size of the smallest local routing
        algorithm including all constants and data-structures)

In the grid example: space = $O(\log n)$ bits
Complexity Measures: Space & Stretch

**Space** = size of the largest local routing tables
   (more precisely, size of the smallest local routing algorithm including all constants and data-structures)

In the grid example: \( \text{space} = O(\log n) \) bits

**Stretch** = ratio between length of the route and distance

\[ |\text{route}(x, y)| \leq \text{stretch} \cdot \text{dist}(x, y) \]

In the grid example: \( \text{stretch} = 1 \) (shortest path)
Complexity Measures: Space & Stretch

**Space** = size of the largest local routing tables
(more precisely, size of the smallest local routing algorithm including all constants and data-structures)

In the grid example: \(\text{space} = O(\log n)\) bits

**Stretch** = ratio between length of the route and distance

\[|\text{route}(x, y)| \leq \text{stretch} \cdot \text{dist}(x, y)\]

In the grid example: \(\text{stretch} = 1\) (shortest path)

**Question:** for a given family of graphs, find the best space-stretch trade-off
Routing Tables: A Universal Routing Strategy

Space = $O(n \log d)$ bits, $d = \text{degree}$
Stretch = 1
Outline

Context and Motivations

Models, Definitions and Examples

A First Compact Routing Scheme

A First Universal Compact Routing Scheme

Concluding Remarks
Routing in a Tree (attempt #1)

- DFS numbering
- Interval Routing
- $O(n \log n)$ bits in total!

4: [5,6]:1 [7,9]:2 [10,3]:3
Routing in a Tree (attempt #1)

- DFS numbering
- Interval Routing
- $O(n \log n)$ bits in total!

Space = $O(d \log n)$ bits, $d =$ degree, Stretch = 1

4: [5,6]:1 [7,9]:2 [10,3]:3
Routing in a Tree (attempt #1)

- DFS numbering
- Interval Routing
- $O(n \log n)$ bits in total!

Space = $O(d \log n)$ bits, $d =$ degree, Stretch = 1

**Problem:** Space $\Omega(n)$ whenever $d = \Omega(n/\log n)$
Routing in a Tree (attempt #2)

Idea: enlarge addresses to reduce tables
Routing in a Tree (attempt #2)

**Idea:** enlarge addresses to reduce tables

- $\text{id}(u)$ is a DFS numbering of $u$ in the tree. The *weight* of a node is the number of its descendants.
Routing in a Tree (attempt #2)

Idea: enlarge addresses to reduce tables

- \( \text{id}(u) \) is a DFS numbering of \( u \) in the tree. The \textit{weight} of a node is the number of its descendants.

- \( \text{table}(u) \) is composed of a partial interval routing of \( u \): only for its parent and its \( t \) heaviest children.
Routing in a Tree (attempt #2)

**Idea:** enlarge addresses to reduce tables

- \( \text{id}(u) \) is a DFS numbering of \( u \) in the tree. The *weight* of a node is the number of its descendants.

- \( \text{table}(u) \) is composed of a partial interval routing of \( u \): only for its parent and its \( t \) heaviest children.

- \( \text{address}(u) = (\text{id}(u), \text{hpath}(u)) \), where \( \text{hpath}(u) \) is the sequence of ports encountered when going from the root to \( u \), but we remove ports leading to a child of rank \( < t \) (so with a large weight), a parameter fixed later.
The Routing Algorithm: $x \rightarrow y$

Given $\text{table}(x)$ and $\text{address}(y)$

- If $\text{id}(x) = \text{id}(y)$, then STOP.
- If $\text{id}(y)$ belongs to some intervals of $\text{table}(x)$, then route with this table.
- Otherwise, route to port $\text{hpath}(y)[1 + |\text{hpath}(x)|]$.

Hint: If the two first tests fail, $\text{hpath}(x)$ is a proper prefix of $\text{hpath}(y)$. 
The Routing Algorithm: \( x \rightarrow y \)

Given \( \text{table}(x) \) and \( \text{address}(y) \)

- If \( \text{id}(x) = \text{id}(y) \), then STOP.
- If \( \text{id}(y) \) belongs to some intervals of \( \text{table}(x) \), then route with this table.
- Otherwise, route to port \( \text{hpath}(y)[1 + |\text{hpath}(x)|] \).

**Hint:** If the two first tests fail, \( \text{hpath}(x) \) is a proper prefix of \( \text{hpath}(y) \).
Space Analysis

- \( \text{table}(u) \) is \( O(t \log n) \) bits
- \( \text{address}(u) \) is \( O(|\text{hpath}(u)| \cdot \log n) \) bits

If \( \text{port}(a, b) \) is in \( \text{hpath}(u) \), then \( b \) has rank \( \geq t \) for \( a \).
So, \( w(b) \leq w(a)/t \), and thus \( |\text{hpath}(u)| \leq \log t n \).
Space Analysis

- \( \text{table}(u) \) is \( O(t \log n) \) bits
- \( \text{address}(u) \) is \( O(|\text{hpath}(u)| \cdot \log n) \) bits

If port \((a, b)\) is in \(\text{hpath}(u)\), then \( b \) has rank \( \geq t \) for \( a \).
So, \( w(b) \leq \frac{w(a)}{t} \), and thus \( |\text{hpath}(u)| \leq \log_t n \).

\( \Rightarrow \) addresses and tables have size \( O(\log^2 n / \log \log n) \) with \( t = \log n / \log \log n \).
Space Analysis

- \( \text{table}(u) \) is \( O(t \log n) \) bits
- \( \text{address}(u) \) is \( O(|\text{hpath}(u)| \cdot \log n) \) bits

If port \((a, b)\) is in \( \text{hpath}(u) \), then \( b \) has rank \( \geq t \) for \( a \).
So, \( w(b) \leq w(a)/t \), and thus \( |\text{hpath}(u)| \leq \log_t n \).

\[ \Rightarrow \] addresses and tables have size \( O(\log^2 n / \log \log n) \) with \( t = \log n / \log \log n \).

Note 1: Any stretch-1 routing scheme on trees requires
\( |\text{table}(u)| + |\text{address}(u)| = \Omega(\log^2 n / \log \log n) \) bits for some \( u \) in some tree.
Space Analysis

- \text{table}(u) \text{ is } O(t \log n) \text{ bits}
- \text{address}(u) \text{ is } O(|hpath(u)| \cdot \log n) \text{ bits}

If \text{port}(a, b) \text{ is in } hpath(u), \text{ then } b \text{ has rank } \geq t \text{ for } a.
So, \ w(b) \leq w(a)/t, \text{ and thus } |hpath(u)| \leq \log_t n.

⇒ \text{addresses and tables have size } O(\log^2 n/ \log \log n) \text{ with } 
\ t = \log n/ \log \log n.

Note 1: Any stretch-1 routing scheme on trees requires 
\ |\text{table}(u)| + |\text{address}(u)| = \Omega(\log^2 n/ \log \log n) \text{ bits for some } 
\ u \text{ in some tree.}

Note 2: Any stretch-1 routing scheme on trees requires 
\ |\text{table}(u)| = \Omega(\sqrt{n}) \text{ bits for some } u \text{ in some graph, if } 
\ |\text{address}(u)| = \lceil \log n \rceil.
Outline

Context and Motivations

Models, Definitions and Examples

A First Compact Routing Scheme

A First Universal Compact Routing Scheme

Concluding Remarks
Sub-Linear Routing Table?

Bad news: Any stretch-1 routing scheme on general graphs requires $|\text{address}(u)| + |\text{table}(u)| = \Omega(n)$ bits for some $u$ in some graph.

Idea: Use a stretch $s > 1$ to make wrong the previous statement.

Hints: Use sparse tree covers, and compact routing in trees.
Bad news: Any stretch-1 routing scheme on general graphs requires $|\text{address}(u)| + |\text{table}(u)| = \Omega(n)$ bits for some $u$ in some graph.
Sub-Linear Routing Table?

**Bad news:** Any stretch-1 routing scheme on general graphs requires $|\text{address}(u)| + |\text{table}(u)| = \Omega(n)$ bits for some $u$ in some graph.

**Idea:** Use a stretch $s > 1$ to make wrong the previous statement.
Bad news: Any stretch-1 routing scheme on general graphs requires $|\text{address}(u)| + |\text{table}(u)| = \Omega(n)$ bits for some $u$ in some graph.

Idea: Use a stretch $s > 1$ to make wrong the previous statement.

Hints: Use sparse tree covers, and compact routing in trees.
Step #1: vicinity balls

$x \mapsto B(x)$ be the $t$ closest nodes around $x$, ties are broken consistently.

$n = 16$, $t = 4$
Step #1: vicinity balls

$x \mapsto B(x)$ be the $t$ closest nodes around $x$, ties are broken consistently.

$n = 16$, $t = 4$
Step #1: vicinity balls

\[ x \mapsto B(x) \] be the \( t \) closest nodes around \( x \), ties are broken consistently.

\[ n = 16, \ t = 4 \]
Step #1: vicinity balls

\( x \mapsto B(x) \) be the \( t \) closest nodes around \( x \), ties are broken consistently.

\( n = 16, \ t = 4 \)
Step #1: vicinity balls

\( x \mapsto B(x) \) be the \( t \) closest nodes around \( x \), ties are broken consistently.

\( n = 16, \ t = 4 \)
Step #1: vicinity balls

\[ x \mapsto B(x) \] be the \( t \) closest nodes around \( x \), ties are broken consistently.

\[ n = 16, \ t = 4 \]
Step #1: vicinity balls

$x \mapsto B(x)$ be the $t$ closest nodes around $x$, ties are broken consistently.

$n = 16, \ t = 4$
Step #1: vicinity balls

\[ x \mapsto B(x) \] be the \( t \) closest nodes around \( x \), ties are broken consistently.

\[ n = 16, \; t = 4 \]
Step #1: vicinity balls

\[ x \mapsto B(x) \] be the \( t \) closest nodes around \( x \), ties are broken consistently.

\[ n = 16, \; t = 4 \]
Step #2: Hitting Set

Select a small **hitting set** $P$ for the family $\{B(x)\}_x$ ($P \cap B(x)$ for every $x \in V$). There exists $P$ such that $|P| \leq (n \ln n)/t + 1$. 
Step #2: Hitting Set

Select a small **hitting set** \( P \) for the family \( \{B(x)\}_x \) (\( P \cap B(x) \) for every \( x \in V \)). There exists \( P \) such that \( |P| \leq (n \ln n)/t + 1 \).
Step #2: Hitting Set (cont’d)

Greedy Algorithm:

1. \( P := \emptyset \) and \( \mathcal{F} := \{B(x) \mid x \in V\} \).
   Let \( F(x) := \{B \in \mathcal{F} \mid x \in B\} \)

2. While \( \mathcal{F} \neq \emptyset \) do:
   - pick \( p \in V \) such that \( |F(p)| \) est maximum
   - \( P := P \cup \{p\} \), \( \mathcal{F} := \mathcal{F} \setminus F(p) \)
Step #2: Hitting Set (cont’d)

Greedy Algorithm:

1. \( P := \emptyset \) and \( \mathcal{F} := \{ B(x) \mid x \in V \} \). Let \( F(x) := \{ B \in \mathcal{F} \mid x \in B \} \).

2. While \( \mathcal{F} \neq \emptyset \) do:
   - pick \( p \in V \) such that \( |F(p)| \) est maximum
   - \( P := P \cup \{ p \} \), \( \mathcal{F} := \mathcal{F} \setminus F(p) \)

Analysis: Define \( f_i \) as \( |\mathcal{F}| \) at the end of step \( i \).
There must \( \exists p \in V \) such that \( |F(p)| \geq f_i \cdot t/n \). Indeed,

\[
\sum_{v \in V} |F(v)| = \sum_{B \in \mathcal{F}} |B| \geq f_i \cdot t .
\]

Thus, \( \exists p \in V \) whose \( F(p) \) has size larger than the average, i.e., \( f_i \cdot t/n \).
Step #2: Hitting Set (cont’d)

\[ f_1 = f_0 - f_0 \cdot \frac{t}{n} = f_0 \cdot (1 - \frac{t}{n}) \]
\[ f_2 = f_1 - f_1 \cdot \frac{t}{n} = f_1 \cdot (1 - \frac{t}{n}) = f_0 \cdot (1 - \frac{t}{n})^2 \]
... = ...
\[ f_i = f_0 \cdot (1 - \frac{t}{n})^i \quad \forall i \geq 0 \]

Algorithm stops at the first \( i_0 \) such that \( f_{i_0} < 1 \), and then \(|P| = i_0\). For all \( 0 < \alpha \leq 1/2 \) and \( \beta > 0 \), \( (1 - \alpha)^\beta < e^{-\alpha \beta} \):

\[ (1 - \frac{t}{n})^{i_0} < e^{-t \cdot i_0 / n} \Rightarrow f_0 \cdot (1 - \frac{t}{n})^{i_0} < f_0 / e^{t \cdot i_0 / n} \]

So \( f_{i_0} < 1 \) implies \( f_0 / e^{t \cdot i_0 / n} < 1 \) implies \( i_0 > (n \ln f_0) / t \). Algorithm has finished at the end of step \( i_0 = \lceil (n \ln n) / t \rceil + 1 \).
Step #3: Tree Cover

**Small trees:** Associate with every $x \in V \setminus P$: a shortest-path tree $T_x$ spanning $B(x)$

**Big trees:** Associate with every $p \in P$: a shortest-path tree $T_p$ spanning $G$

Denote by $\mathcal{C}$ the family of these trees.
Step #3: Tree Cover

**Small trees:** Associate with every $x \in V \setminus P$: a shortest-path tree $T_x$ spanning $B(x)$

**Big trees:** Associate with every $p \in P$: a shortest-path tree $T_p$ spanning $G$

Denote by $\mathcal{C}$ the family of these trees.

**Claim:** For every $x, y$, $\exists T \in \mathcal{C}$ such that $d_T(x, y) \leq 3d(x, y)$.
**Step #3: Tree Cover**

**Small trees:** Associate with every $x \in V \setminus P$: a shortest-path tree $T_x$ spanning $B(x)$

**Big trees:** Associate with every $p \in P$: a shortest-path tree $T_p$ spanning $G$

Denote by $\mathcal{C}$ the family of these trees.

**Claim:** For every $x, y$, $\exists T \in \mathcal{C}$ such that $d_T(x, y) \leq 3d(x, y)$. 
Choose \( t := \sqrt{n \ln n} \), so that:
\[
|P| \leq \frac{n \ln n}{t} + 1 = \sqrt{n \ln n} + 1.
\]

**Observation:** The graph \( H := \bigcup_{T \in e} T \) is a spanning subgraph of \( G \) and its **number of edges** is at most:

\[
(n - |P|) \cdot (t - 1) + |P| \cdot (n - 1) < 2n\sqrt{n \ln n} = \tilde{O}(n^{3/2})
\]
Step #3: Tree Cover (cont’d)

Choose \( t \coloneqq \sqrt{n \ln n} \), so that:
\[
|P| \leq \frac{n \ln n}{t} + 1 = \sqrt{n \ln n} + 1.
\]

**Observation:** The graph \( H \coloneqq \bigcup_{T \in \mathcal{C}} T \) is a spanning subgraph of \( G \) and its **number of edges** is at most:

\[
(n - |P|) \cdot (t - 1) + |P| \cdot (n - 1) < 2n \sqrt{n \ln n} = \tilde{O}(n^{3/2})
\]

**Our Goal:** Design a stretch-3 routing scheme with routing tables of \( \tilde{O}(n^{1/2}) \) bits per node and polylog addresses.
How to Route from $x$ to $y$?

**Easy case:** $y \in B(x)$.
Node $x$ can store $B(x)$ and the port to each $v \in B(x)$. This is $\tilde{O}(t) = \tilde{O}(n^{1/2})$ bits of information.

**Claim:** if $w$ is on a shortest-path from $x$ to $v$, then $v \in B(w)$. 
How to Route from $x$ to $y$?

Otherwise: $y \notin B(x)$. Then, route to $p(x)$, and then try to reach $y$ along $T_{p(x)}$.

Each node $z$ can store table $T_{p(z)}$ for each big tree $T_{p}$ (there are only $|P|$ such trees). But, neither $x$ neither $p(x)$ can store address $T_{p(x)}(y)$ for all possible $y$. If we want polylog($n$) addresses, $y$ cannot store address $T_{p(x)}(y)$ in its address.
Another Solution

Hints: if $y \notin B(x)$, route along $T_{p(y)}$ and store $address_{T_{p(y)}}(y)$ in the $y$’s address.
Another Solution

Hints: if $y \notin B(x)$, route along $T_{p(y)}$ and store $\text{address}_{T_{p(y)}}(y)$ in the $y$’s address.
Another Solution

Hints: if \( y \notin B(x) \), route along \( T_{p(y)} \) and store address \( T_{p(y)}(y) \) in the \( y \)'s address.
Outline

Context and Motivations

Models, Definitions and Examples

A First Compact Routing Scheme

A First Universal Compact Routing Scheme

Concluding Remarks
Best Known Universal Routing Schemes

[addresses are polylog, space is up to polylog]

<table>
<thead>
<tr>
<th>stretch (LB)</th>
<th>stretch (UB)</th>
<th>space</th>
<th>scheme</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>$n$</td>
<td>name-indep.</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>$\sqrt{n}$</td>
<td>name-indep.</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>$n^{1/3}$</td>
<td>labeled</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$1.5k - 1$</td>
<td>$4k - 5$</td>
<td>$n^{1/k}$</td>
<td>labeled</td>
</tr>
<tr>
<td>$2k - 1$</td>
<td>$\approx 100k$</td>
<td>$n^{1/k}$</td>
<td>name-indep.</td>
</tr>
</tbody>
</table>
Best Known Universal Routing Schemes

([addresses are polylog, space is up to polylog])

<table>
<thead>
<tr>
<th>stretch (LB)</th>
<th>stretch (UB)</th>
<th>space</th>
<th>scheme</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>$n$</td>
<td>name-indep.</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>$\sqrt{n}$</td>
<td>name-indep.</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>$n^{1/3}$</td>
<td>labeled</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>$n^{1/k}$</td>
<td>...</td>
</tr>
<tr>
<td>$1.5k - 1$</td>
<td>$4k - 5$</td>
<td>$n^{1/k}$</td>
<td>labeled</td>
</tr>
<tr>
<td>$2k - 1$</td>
<td>$\approx 100k$</td>
<td>$n^{1/k}$</td>
<td>name-indep.</td>
</tr>
</tbody>
</table>

**Observations:** Only the two first lines of the table are known to be optimal. For $n^{1/3}$ space, there is a lower bound of 5 on the stretch. For general $k$, the conjecture is stretch $2k - 1$, but neither the current lower or upper bounds reach this bound when $k > 5$. 