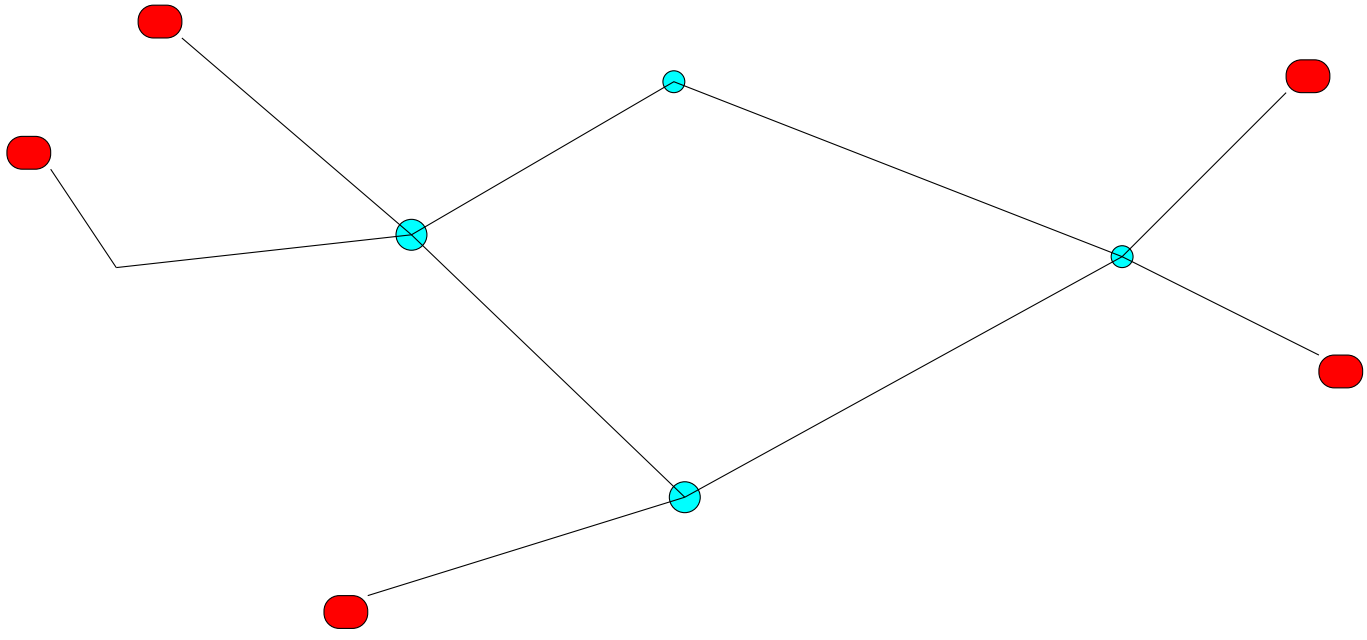
Linear Algebra for Network Loss Characterization

David Bindel

UC Berkeley, CS Division

Application: Network monitoring



- Set of n hosts in a large network
- $n(n-1)/2$ (undirected) paths between them
- Want latency and packet loss rates for each path
- Use info to choose servers, route around faults

Network “distance” metrics

Linear relation between path length y_{ij} and link length x_l:

$$y_{ij} = \sum_{l \in \text{path}} x_l = \sum_l g_{ijl} x_l$$

where g_{ijl} indicates if link l is used on path $i \to j$.
Write in matrix form (one path per row): $y = Gx$.

Examples:

- Latency
- Log probability of transmission success
- Jitter (variance in latency)

Path distance inference

Given network topology:

- Choose and instrument a basis of paths
- From those measurements, infer other paths lengths
- Quickly update when topology changes

Given G:

- Choose a basis $\bar{G}$ from rows of G
- Given measured rows $\bar{y}$, compute y
- Updates may affect a few rows/columns of G

Unidentifiable paths

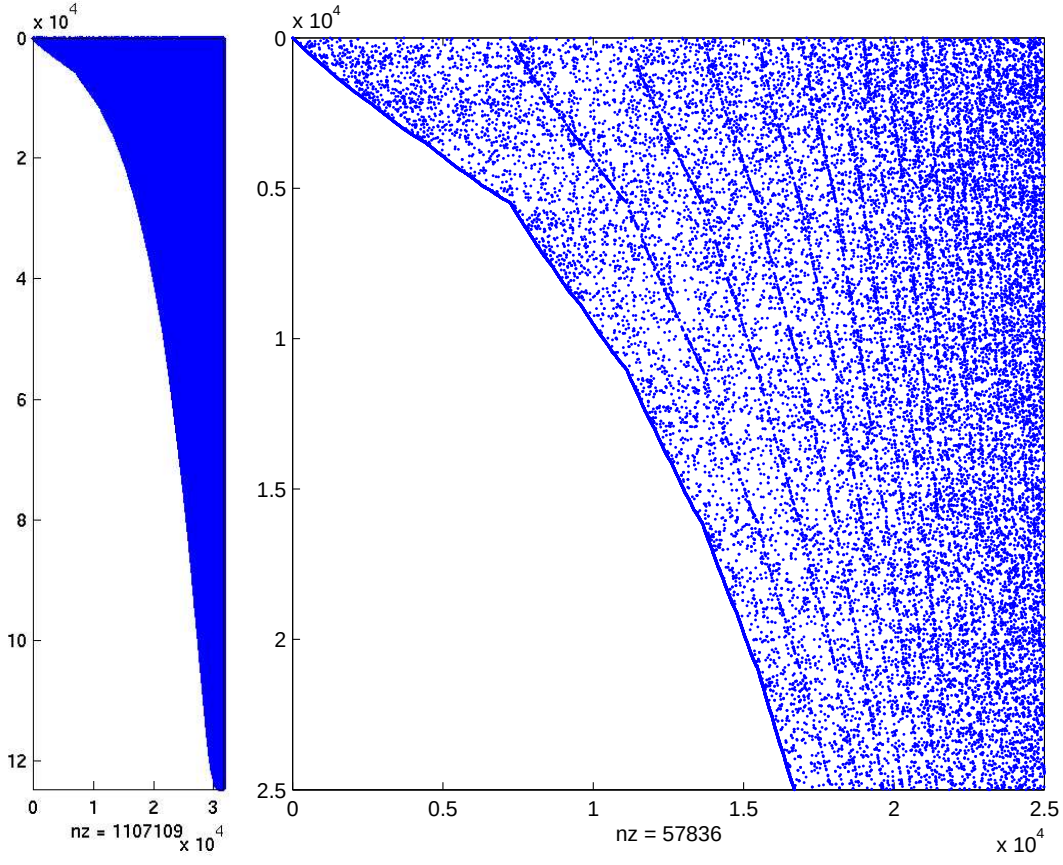
Unidentifiable paths are outside row space of G:

- Paths from host affected by a topology change
- Individual links on measured paths

If row vector v represents an unidentifiable path, can use non-negativity of distance to bound length η:

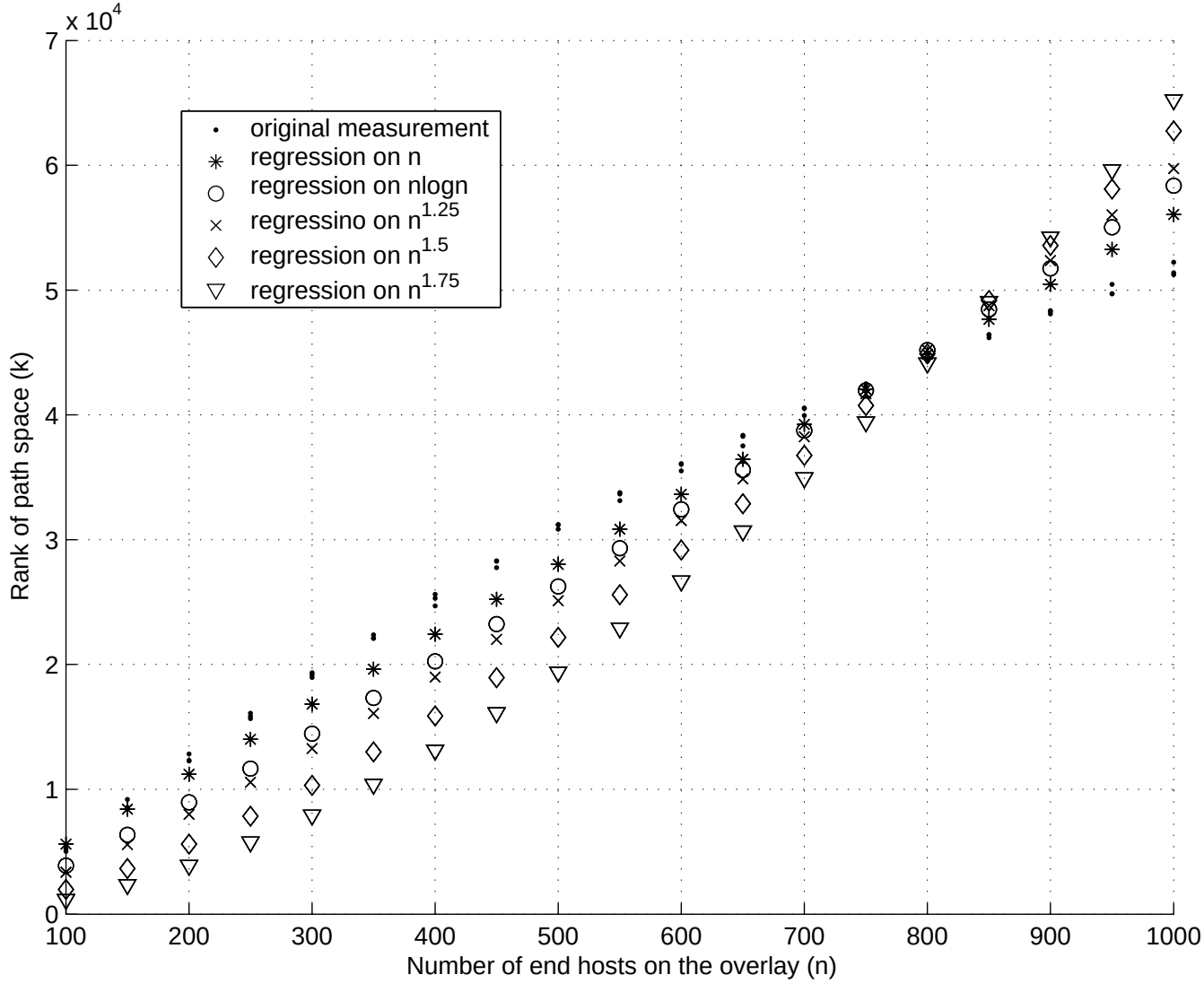
$$\begin{aligned} \eta &\leq \min_{zG \geq v} zy \\ \eta &\geq \max_{zG \leq v} zy \end{aligned}$$

Structure of G

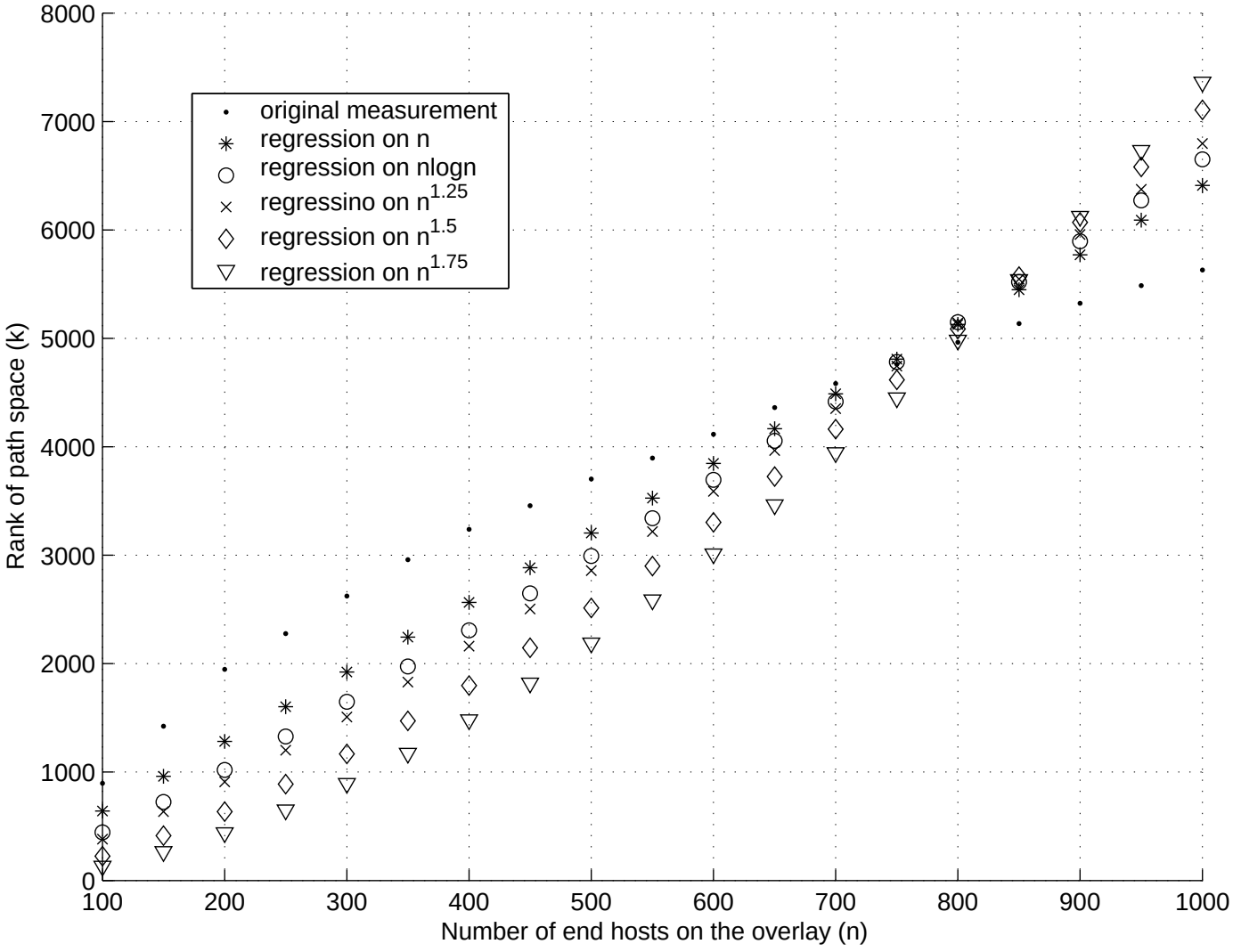


- Network has hierarchy $\Longrightarrow$ paths overlap
- $k := \mathrm{rank}(G) \leq$ links used $=$ nonzero columns of G
- links used $< O(n^2)$ – grows line $O(n)$?

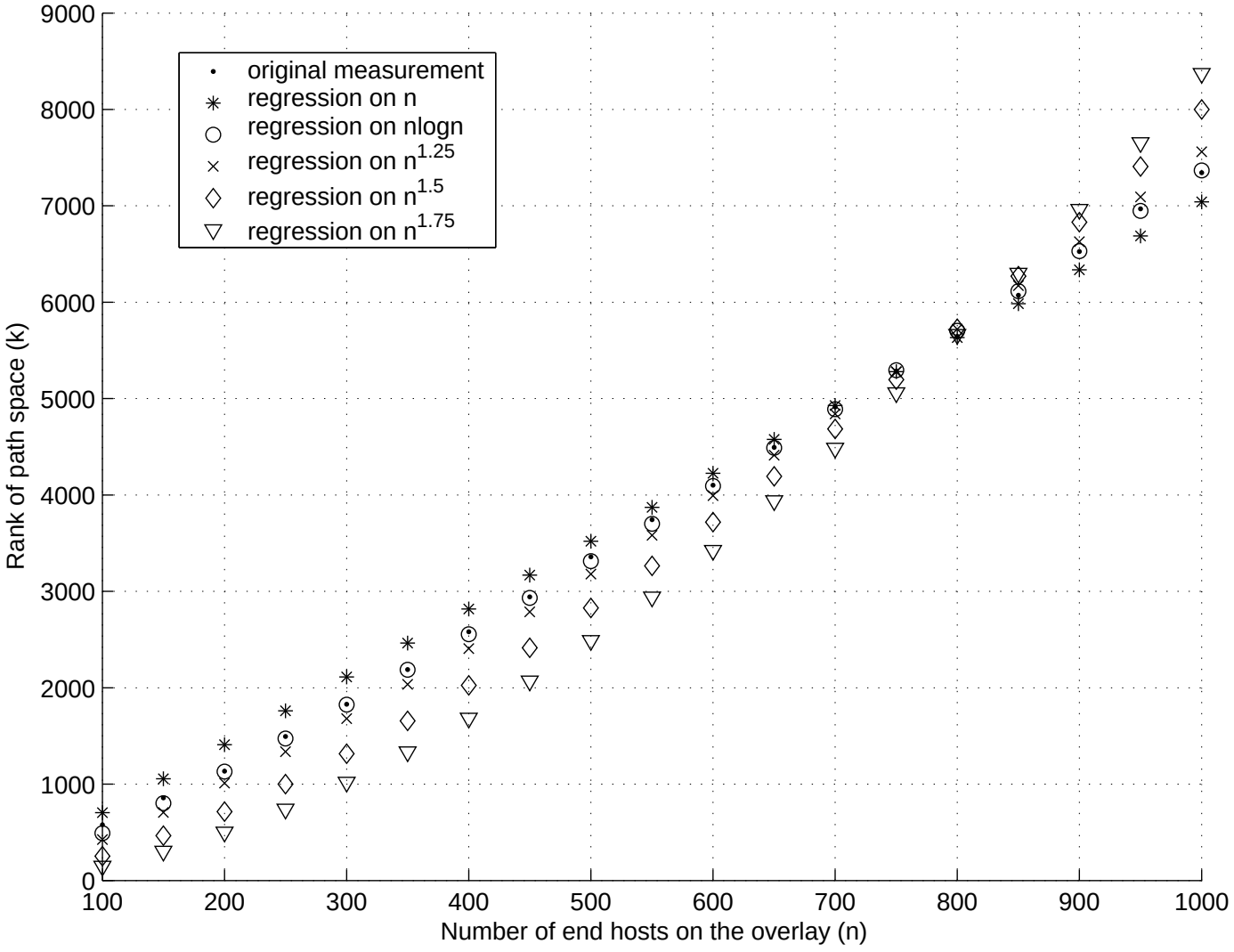
Rank of G: Lucent scan (bound)



Rank of G: AS-level Albert-Barabasi



Rank of G: AS Barabasi + RT Waxman



Algorithm tasks

- Choose basis $\bar{G}$ for row space of G
- Solve linear systems involving $\bar{G}$
- Quickly update basis choice / factorizations on:
  - Addition of new nodes / paths
  - Deletion of nodes / paths
  - Localized changes to network topology

Key ingredient is quickly solving linear systems with $\bar{G}$.

Dense direct algorithm

- QR factorization of G^T
- Only keep part of R for $\bar{G}^T$
- Basically block CGS with iterative refinement
- Store R densely, but in (block) packed single precision
- Use $Q := G^T R^{-1}$ if needed

Row selection and factorization

Input: Current $\bar{G}$, path vectors V, current R
Output: Updated $\bar{G}$, R

```
R12      = R' \ (Gbar' * V);
R22      = V' * V - R12' * R12;
[q,r,e] = qr(R22);
k        = sum(abs(diag(r)) > tol);
R        = [R, R22(:,       e(1:k));
             0, R12(e(1:k), e(1:k));
Gbar     = [Gbar; V(e(1:k),:)'];
```

Computing y

- Compute minimum norm solution to $\bar{G}x = \bar{y}$:
  - $\bar{x} = (\bar{G}^T R^{-1}) R^{-T} \bar{b}$ plus iterative refinement
- Compute $y = \bar{G}\bar{x}$

Alternative:

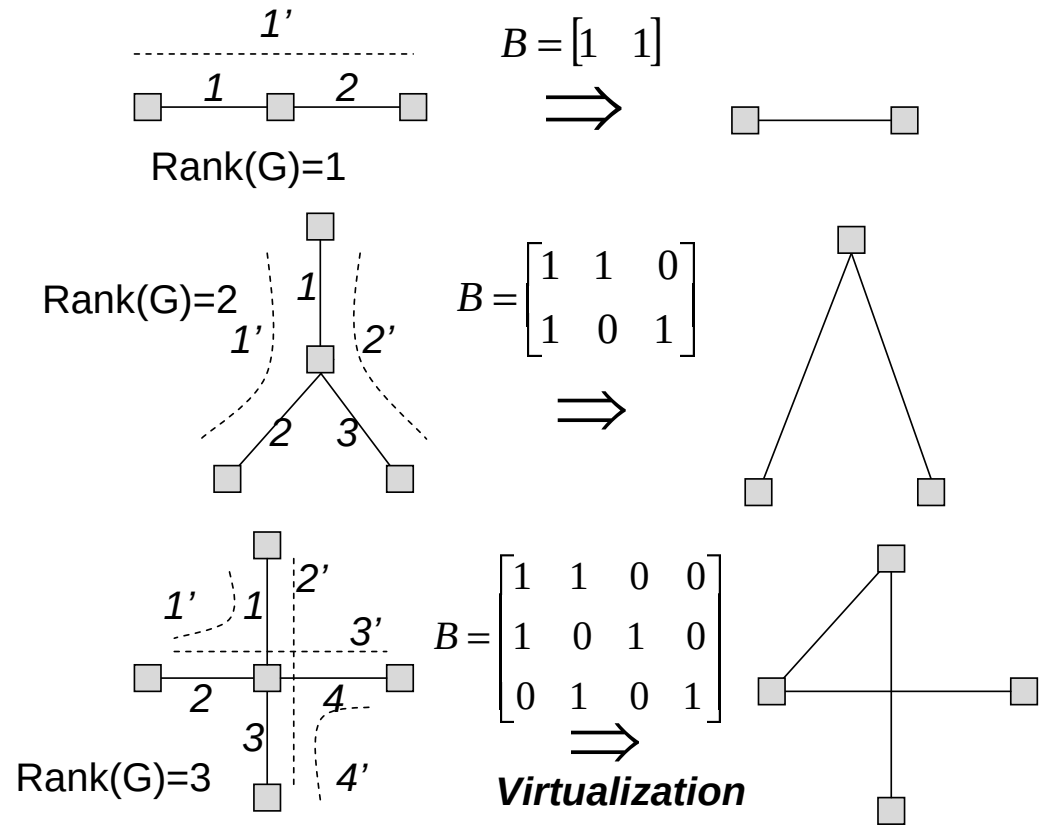
- Directly write paths as combinations of measured paths
- $W\bar{G} = G \implies W\bar{y} = y$
- Average nonzero count per row seems small

Removing paths

To remove row i of $\bar{G}$:

- Compute a vector in the null space of $\bar{G}$ minus row i: $\bar{G}_{\text{orig}} z = e_i$
- Compute $r := Gz$
- If $r \neq 0$ then add row j such that $r_j \neq 0$ to $\bar{G}$
- If $r = 0$ then k decreased by one, no replacement
- Can update R in $O(k^2)$ time in standard way

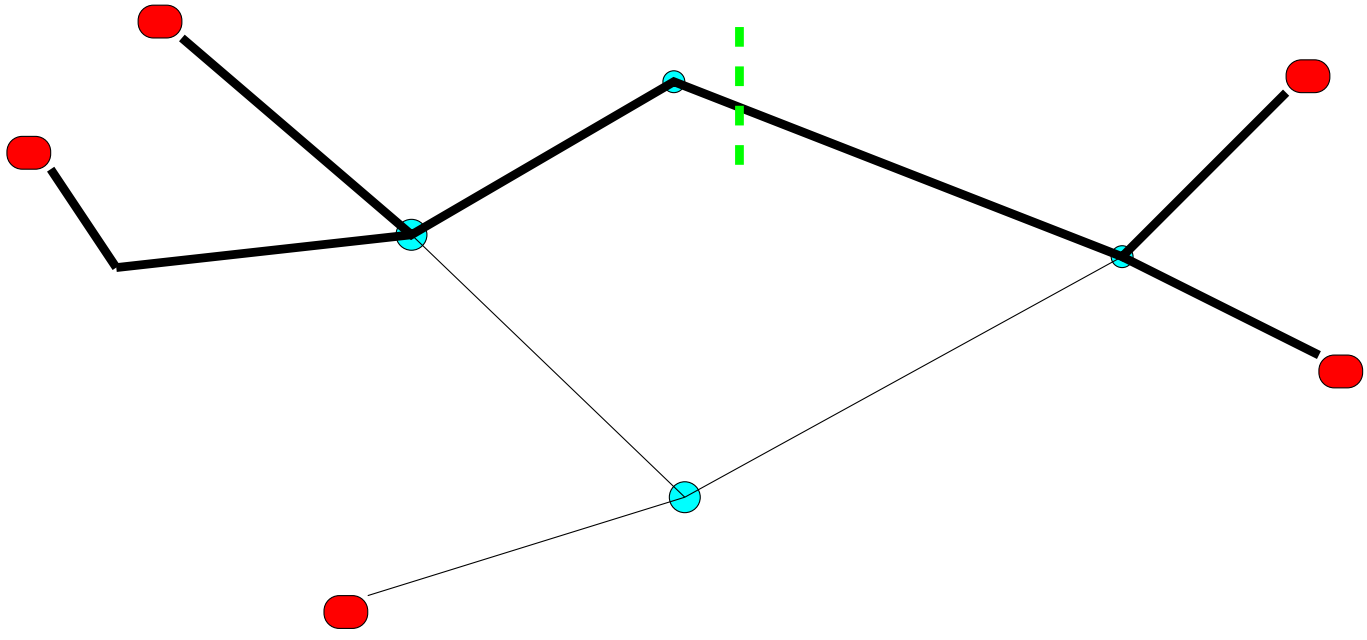
Virtualization and local elimination



Real links (solid) and overlay paths (dotted) going through them

Virtual links

Network partitioning



Suppose nodes $1, \ldots, k_1$ talk to $1', \ldots k_2'$ through a single link. Then

$$g_{ij'l} = g_{i1'l} + g_{1j'l} - g_{11'l}$$

Only $k_1 + k_2$ of $k_1 k_2/2$ paths may be independent.

Idea extends to case when a few links are used.

Current direct scheme

- Find heaviest columns (index set J)
- Run pivoted QR on $G(:,J)^T G(:,J)$ to find redundancy cols
- For a few columns $j \in J$
  - Identify rows I that use column j
  - Run pivoted QR on $G(I,:)G(I,:)^T$ to find redundant rows
- Remove heaviest rows and columns
- Run UMFPACK and hope

Iterative scheme

Apply CG to normal equations

- Convergence of basic iteration is mediocre
- Should use a preconditioner (what?)
- Want fast multiply (how?)

Fast multiply

- Partition G by source node

$$G^T G = \sum_i G_i^T G_i$$

  where G_i is paths from i.

- Paths in G_i come from a tree rooted at i
- Fast multiply by $G_i G_i^T$ uses tree structure and intuition from interpretation of the multiply:

  $y = Gx$ Path length = sum of edge lengths

  $z = G^T y$ Edge load = sum of traversing path loads

Fast multiply by $G_i^T G_i$

nsubtree(j) : Number of nodes beneath j

fromcost(j) : Cost of path from root to j

treecost(j) : Total cost of paths from j

$x(j,k)$ and $z(j,k)$: Input and output for edge $j \to k$

$$\text{fromcost}(j) = \begin{cases} \text{fromcost}(\text{parent}(j)) + x(\text{parent}(j), j) \\ 0 \text{ at root} \end{cases}$$

$$\text{treecost}(j) = \sum_{k \in \text{children}(j)} \text{nsubtree}(k) \times x(j,k) + \text{treecost}(k)$$

$$z(j,k) = \text{nsubtree}(k) \times \text{fromcost}(k) + \text{treecost}(k)$$

Some system issues

- Measurement load balancing
- Construction and updates to G are nontrivial
  - Incorrect mapping due to aliasing is okay.
  - Can still do useful work with incomplete info.
- How do applications actually use the info?
  - Set up notifications when a path becomes lossy
  - Query server for loss rates when choosing paths

Status

On the numerical side:

- Direct solver works (but gets expensive)
- Topology update algorithms work fine
- CGNE works, no fast multiply or preconditioner yet
- Sparse solver works for small networks

On the systems side:

- Have an experimental setup on PlanetLab testbed
- Streaming media application is running
  - Detect congestion: 1.5 s
  - Find alternative with low latency/loss rate: 0.66 s
  - Reconnect and concatenate stream: 0.73 s

Future work

On the numerical side:

- Better understanding of the structure of G
- Some combination of sparse and iterative solvers
- Linear-program based bounds from incomplete info

Project page:
www.cs.northwestern.edu/$\sim$ychen/research/wnmms/