Matrix Factorizations for Computer Network Tomography

David Bindel

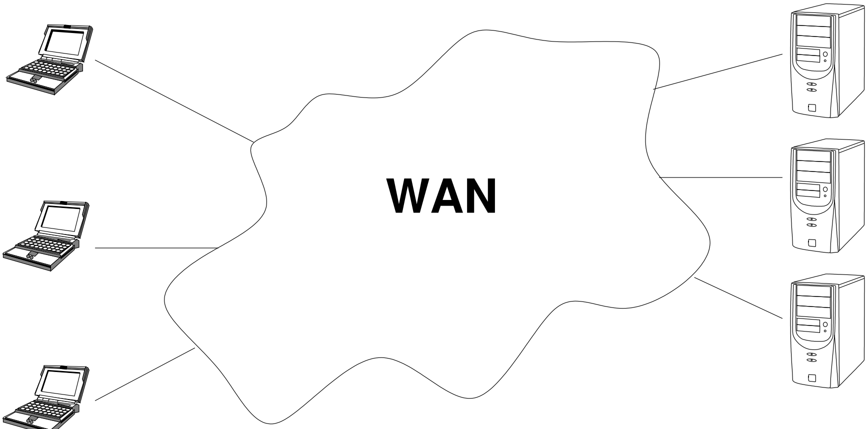
Department of Computer Science
Cornell University

17 Jun 2011

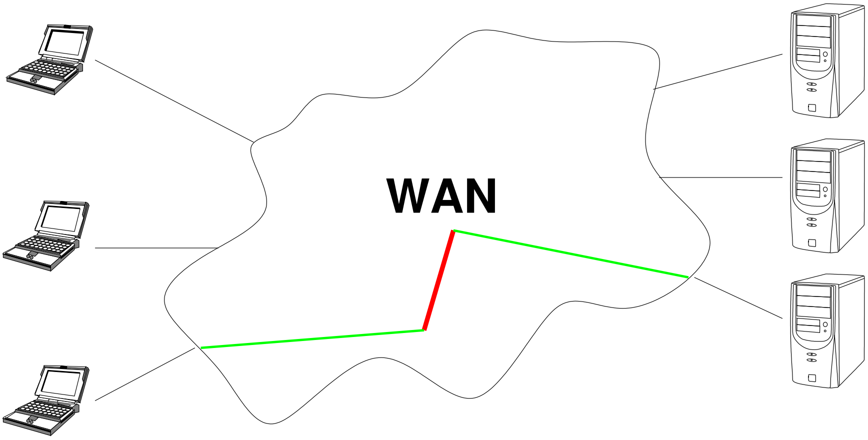
How this started

"I typed in the SVD from Numerical Recipes; it ran for a few days, then told me it wouldn't converge. What can I do?"

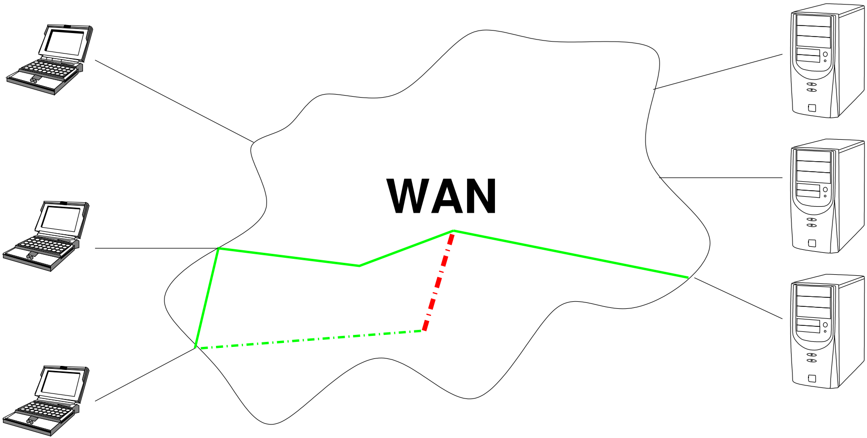
A fuzzy picture



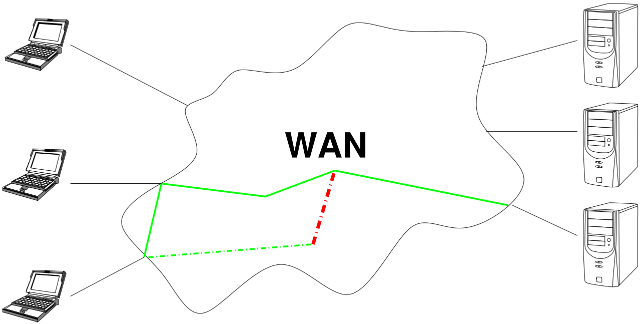
A problem path



Overlays to the rescue?



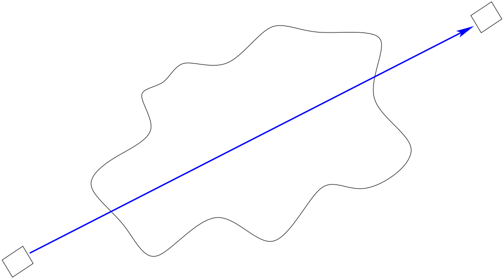
Overlays and measurement



Measure a few paths to infer:

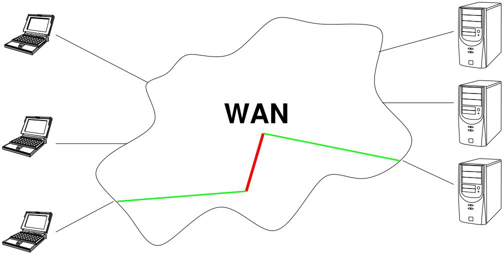
- *Path* properties (Chen, B., Song, Chavez, Katz: 2003, 2004, 2007)
- *Link* properties (Zhao, Chen, B.: 2006, 2009)
- Routing topology? (underway)

Discrete Radon transform



Radon transform:

$$(Ru)(L) = \int_L u(\mathbf{x})\,|d\mathbf{x}|$$



Discrete version:

$$(Gu)_i = \sum_{j \in \text{links}(i)} u_j$$

Additive metrics and path matrices

For additive metrics (log(P(transmission)), latency, jitter, ...):

$$Gu = b$$

where

- b_i = property of ith end-to-end path
- u_j = property of link j
- $G_{ij} = \begin{cases} 1 & \text{if path } i \text{ uses link } j \\ 0 & \text{otherwise} \end{cases}$

Today's goal: A structured rank-revealing decomposition of G

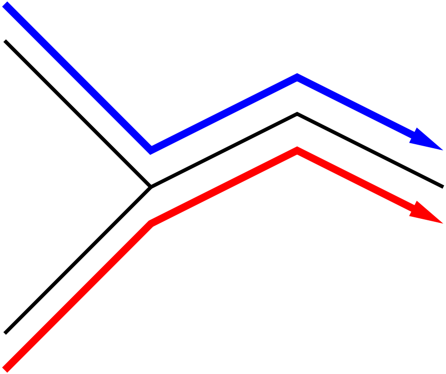
Properties of G

Network	Matrix G
Routing path	Row of G
Short paths	Sparse G
Routing table updates	Low rank updates to G

$k = \mathrm{rank}(G) <$ # links $\ll$ # paths (for n sufficiently large).

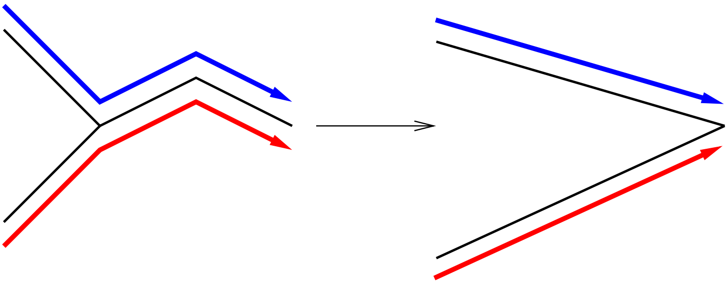
Ex: 500 nodes (of 100K), G is 249500×33237, $k = 9643$.

Identifiability



If both paths are flaky, what link is to blame?

Network virtualization and matrix factorization



Factor out a zero-one "virtualization matrix":

$$G(:, \text{fan}) = \begin{bmatrix} c_1 & c_2 & c_1 + c_2 & c_1 + c_2 \end{bmatrix} = \begin{bmatrix} c_1 & c_2 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

Even virtual links may be "unidentifiable."

Network virtualization and matrix factorization

Write network virtualization as

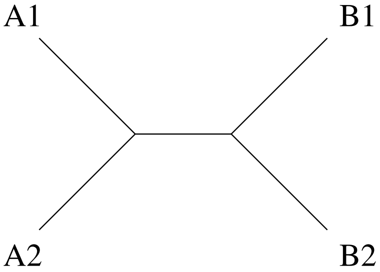
$$G = G^v S$$

where

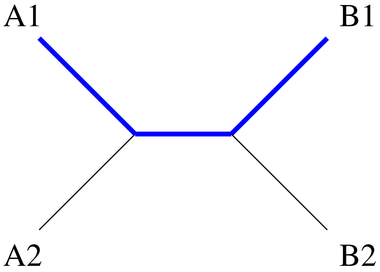
- $G_{ij} = \begin{cases} 1 & \text{if path } i \text{ uses link } j \\ 0 & \text{otherwise} \end{cases}$
- $G^v_{ik} = \begin{cases} 1 & \text{if path } i \text{ uses virtual link } k \\ 0 & \text{otherwise} \end{cases}$
- $S_{kj} = \begin{cases} 1 & \text{if virtual link } k \text{ includes link } i \\ 0 & \text{otherwise} \end{cases}$

This handles (some) column dependencies. What about rows?

The junction pattern

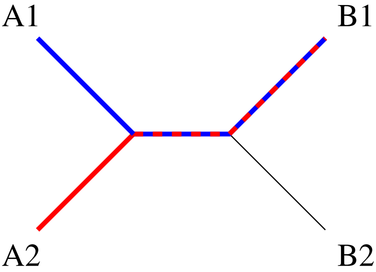


The junction pattern



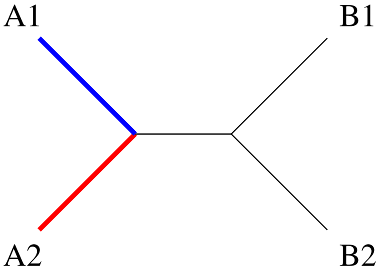
$A_1 B_1$

The junction pattern



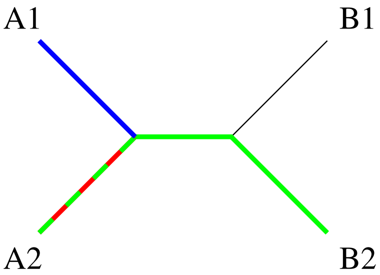
$A_1 B_1 - A_2 B_1$

The junction pattern



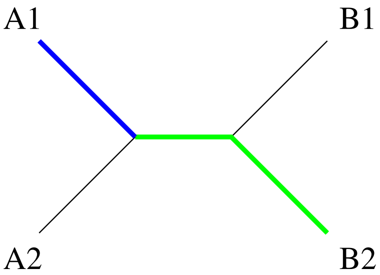
$A_1 B_1 - A_2 B_1$

The junction pattern



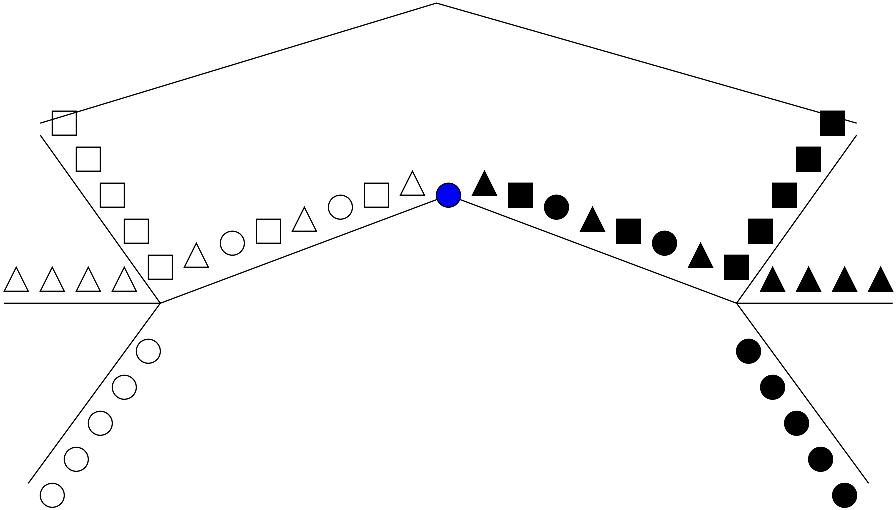
$$A_1 B_1 - A_2 B_1 + A_2 B_2$$

The junction pattern

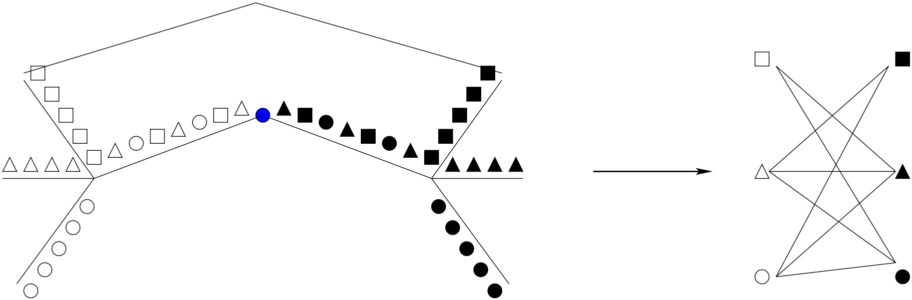


$$A_1 B_1 - A_2 B_1 + A_2 B_2 = A_1 B_2$$

More complicated junctions?



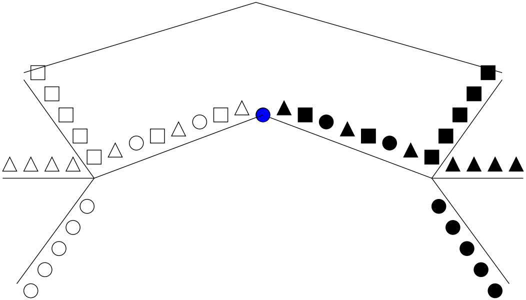
Junctions and bipartite graphs

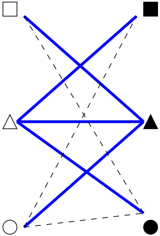


Define a bipartite *router graph* at r:

- Path segments from sources to r are nodes on the left
- Path segments from r to destinations are nodes in the right
- Edges indicate complete paths traversing r

Junctions and bipartite graphs



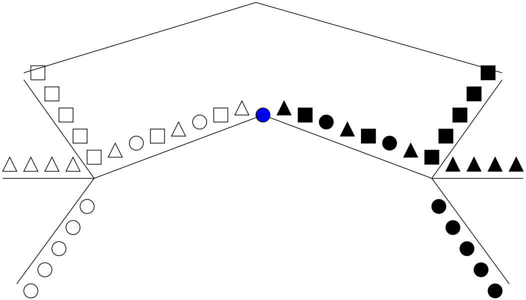


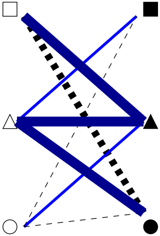
Spanning trees in the router graph

$\Longrightarrow$

spanning sets among path vectors

Junctions and bipartite graphs example





Junction elimination and factorization

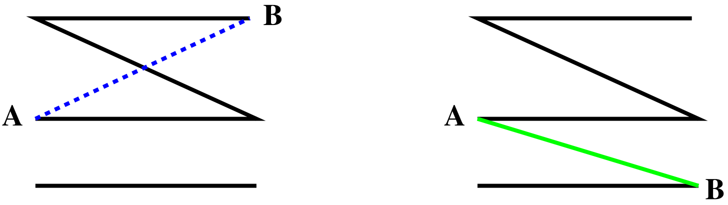
G_r = matrix of path vectors for paths crossing r:

$$G_r = \begin{bmatrix} E_S & E_D \end{bmatrix} \begin{bmatrix} P_S \\ P_D \end{bmatrix} = \begin{bmatrix} I \\ T_r \end{bmatrix} \begin{bmatrix} \bar{E}_S & \bar{E}_D \end{bmatrix} \begin{bmatrix} P_S \\ P_D \end{bmatrix} = \begin{bmatrix} I \\ T_r \end{bmatrix} \begin{bmatrix} \bar{G}_r \end{bmatrix}$$

where

- Rows of P are path segments to/from the router
- Rows of E indicate how path segments sum to form paths
- $\bar{E}$ corresponds to spanning tree edges
- $\bar{G}$ is the corresponding subset of paths
- Rows of T_r consist of ± 1 entries (and zeros) corresponding to paths through the spanning tree

Combining router results



Process each path in turn, build router forests incrementally.
Processing a path from A to B through r, have either

1. $A - r$ and $r - B$ in same component $\implies$ could infer path at r from existing paths
2. $A - r$ and $r - B$ in different components $\implies$ might make new inferences via this path

Elimination algorithm

To process path from *A* to *B*:

```
for each router r on path
  update hash h of route up to r
  if (A,h)-(r,B) in router graph
    mark that path can be inferred at r
  else
    add (A,h)-(r,B) to router graph
    for each edge e from source in [(A,h)]
      if e goes to component for (A,h) and
         edge is not already marked then
        put edge on top of list to be processed

if path was not inferred, mark as measured
```

Matrix factorization perspective

Junction elimination yields

$$PG = \begin{bmatrix} I \\ T \end{bmatrix} \bar{G},$$

where the first factor is a product of matrices of the form

$$\begin{bmatrix} I \\ \tilde{T}_k \end{bmatrix}$$

with rows of $\tilde{T}_k$ representing paths through router graphs.

Matrix factorization perspective

Can combine with topology virtualization:

$$PG = \begin{bmatrix} I \\ T \end{bmatrix} \bar{G}^v S$$

where S is a zero-one virtualization matrix.

Storage

Sufficient to store:

- Each router graph ($\approx$ nnz(G) edges)
- Union-find structures for tracking components
- Markers for which paths are measured
- Router used for inference for each inferred path

Choice of representative paths

Spanning trees are not unique! Want representative paths such that

- There are few redundant measurements
- No host (or router) is overloaded with measurement traffic
- Most inferences involve few paths

Don't know how to do this yet...

Summary

Path matrix G useful for network inference:

- Measure a few paths, infer rest
- Measure a few paths, estimate link behaviors

Cost of factoring G limited scalability.

New factorization for the path matrix G is

- Compact (proportional to G in storage)
- Easy to compute
- Nearly rank-revealing
- Faithful to the underlying problem structure

Questions?

I have lots more questions:

- How should we process paths for load balancing, etc?
- Can these methods be distributed?
- Are there other places where this factorization applies?

Questions from you?

References

- Chen, B., Song, Chavez, Katz. *Algebra-based scalable overlay network monitoring: Algorithms, evaluation, and applications*. ACM ToN, 17(6), 2009.
- Zhao, Chen, B. *Towards unbiased end-to-end network diagnosis*. ACM ToN, 15(5), 2007