Applications of Matrix Structure

D. Bindel

Department of Computer Science
Cornell University

15 Sep 2009

The Computational Science Picture

Ingredients:

- Application modeling
- Mathematical analysis
- Software engineering

Usually requires more than one person!

Outline

1. Linear algebra for network monitoring
2. Low rank structure and fast solvers
3. A mess of microsystems
4. Concluding note

Definitions

- Overlay network of n cooperating hosts
- x_j is the "length" of link j for $j = 1$ to N
  E.g.

$$x_j = -\log(P(\text{successful packet transmit on link } j)).$$

- b_i is "length" of path i, $i = 1$ to $M = n(n-1)$ (or $n(n-1)/2$)
  E.g.

$$b_i = -\log(P(\text{successful packet transmit on path } i))$$

- Have $Gx = b$ where

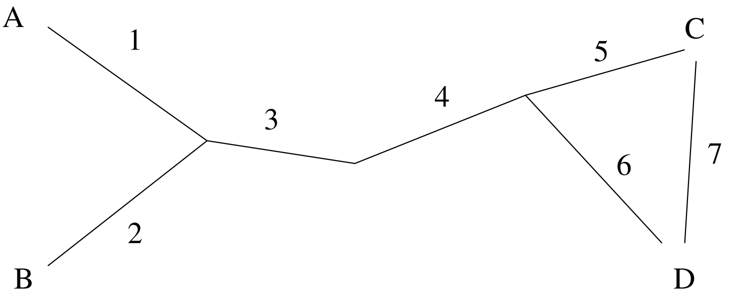
$$G_{ij} = \begin{cases} 1, & \text{path } i \text{ contains link } j \\ 0, & \text{otherwise.} \end{cases}$$

Using low rank

Idea: Use structure of $Gx = b$ to learn about x, b

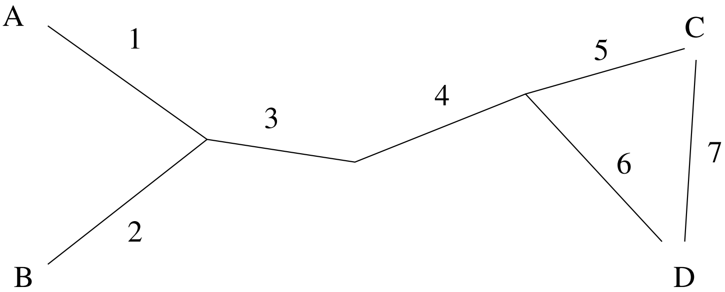
- In general, $\mathrm{rank}(G) < N \ll M$ (for n sufficiently large).
- Can measure $\mathrm{rank}(G)$ paths and infer properties of others.
  (Chen, B., Song, Chavez, Katz —
  ToN 2007, SIGCOMM 2004, IMC 2003)
- In general, can *bound* properties of links.
  (Zhao, Chen, B. — SIGCOMM 2006, ToN (to appear))
- Today: Exposing low rank structure via matrix decomposition.
  (joint with Jiexun Xu)

Example network



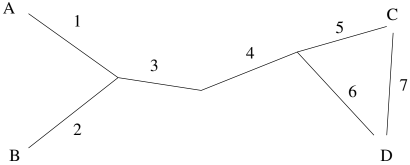
$$G = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

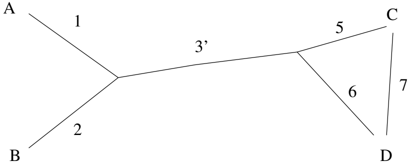
Chains



A *chain* is a sequence of links that occur together on any path where they occur (e.g. 3-4). Appear as identical columns in G. Can think of replacing a chain by a single virtual link — or in terms of a matrix decomposition.

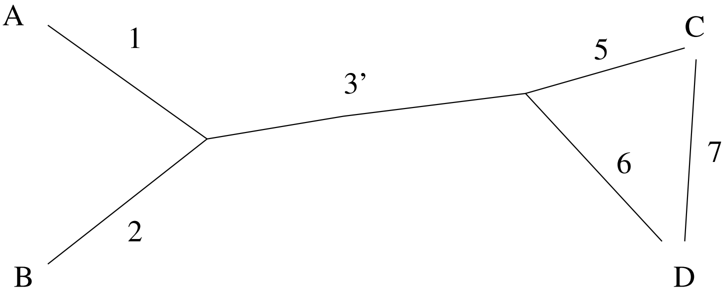
Chain elimination





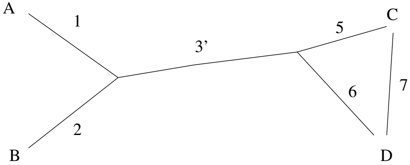
$$G = G_1 T_1 = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

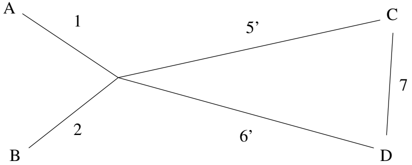
Fans



A *fan* is a link that always appears with exactly one of a set of other links. For example, 3' is a fan since it always occurs with either 5 or 6. Fans generalize chains.

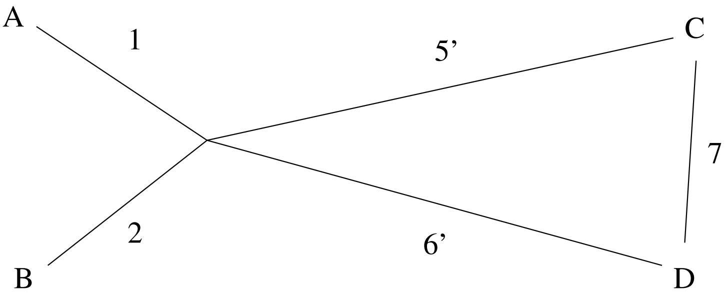
Fan elimination





$$G_1 = G_2 T_2 = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

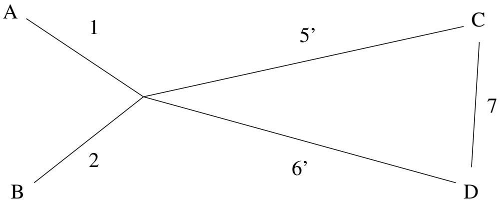
Junctions



A *junction* occurs whenever all paths set S_1 to set S_2 cross at a common point. Implies $|S_1| + |S_2| - 1$ of the $|S_1||S_2|$ paths from S_1 to S_2 are independent. Here, a junction between $\{A, B\}$ and $\{C, D\}$ means

$$(B \to D) = (B \to C) + (A \to D) - (A \to C).$$

Junction elimination



$$(B \to D) = (B \to C) + (A \to D) - (A \to C).$$

$$G_2 = S_3 G_3 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & -1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Virtualization and matrix decomposition

Our network rewriting can be seen as a decomposition of G:

$$G = S\hat{G}T,$$

where $\hat{G}$ is smaller than G, but $\mathrm{rank}(G) = \mathrm{rank}(\hat{G})$.

Idea: solve problems with $\hat{G}$, map back to results with G.

Questions

1. Can we get some theory explaining the scaling of $\mathrm{rank}(G)$ with n?
2. How do we choose $\hat{G}$ (or even smaller basis) to balance load?
3. How do we find good junctions for fast elimination?
4. Are there other common features that can be used for elimination?
5. Can we monitor dependencies to detect routing changes?

Outline

A motivating example

Gravitational potential at mass j from other masses is

$$\phi_j(x) = \sum_{i \neq j} \frac{Gm_i}{|x_i - x_j|}.$$

In cluster A, don't *really* need everything about B. Just summarize.

A motivating example

Gravitational potential is a linear function of masses

$$\begin{bmatrix} \phi_A \\ \phi_B \end{bmatrix} = \begin{bmatrix} P_{AA} & P_{AB} \\ P_{BA} & P_{BB} \end{bmatrix} \begin{bmatrix} m_A \\ m_B \end{bmatrix}.$$

In cluster A, don't *really* need everything about B. Just summarize. That is, represent P_{AB} (and P_{BA}) compactly.

Low-rank interactions

Summarize masses in B with a few variables:

$$z_B = V_B^T m_B, \quad m_B \in \mathbb{R}^{n_B}, z_B \in \mathbb{R}^p.$$

Then contribution to potential in cluster A is $U_A z_B$. Have

$$\phi_A \approx P_{AA} m_A + U_A V_B^T m_B.$$

Do the same with potential in cluster B; get system

$$\begin{bmatrix} \phi_A \\ \phi_B \end{bmatrix} = \begin{bmatrix} P_{AA} & U_A V_B^T \\ U_B V_A^T & P_{BB} \end{bmatrix} \begin{bmatrix} m_A \\ m_B \end{bmatrix}.$$

Idea is the basis of fast *n*-body methods (e.g. fast multipole method).

General picture

- Local details of mass “blur out” far away.
- Same blurring at a distance throughout physics!
- Results in low-rank submatrices summarizing far interactions + *sparse* matrices (few nonzeros) for near interactions
- Lots of work on solving linear systems with low-rank structure... but software lags.
- Idea: Use existing software for solving sparse linear systems.

Sparsification

Want to solve $Ax = b$ where $A = S + UV^T$ is sparse plus low rank.

If we knew x, we could quickly compute b:

$$
\begin{aligned}
z &= V^T x \\
b &= Sx + Uz.
\end{aligned}
$$

Use the same idea to write $Ax = b$ as a bordered system[1]:

$$
\begin{bmatrix} S & U \\ V^T & -I \end{bmatrix} \begin{bmatrix} x \\ v \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}.
$$

Solve this using standard sparse solver package (e.g. UMFPACK).

[1] This is Sherman-Morrison in disguise

Sparsification in gravity example

Suppose we have ϕ, want to compute m in

$$\begin{bmatrix} \phi_A \\ \phi_B \end{bmatrix} = \begin{bmatrix} P_{AA} & U_A V_B^T \\ U_B V_A^T & P_{BB} \end{bmatrix} \begin{bmatrix} m_A \\ m_B \end{bmatrix}.$$

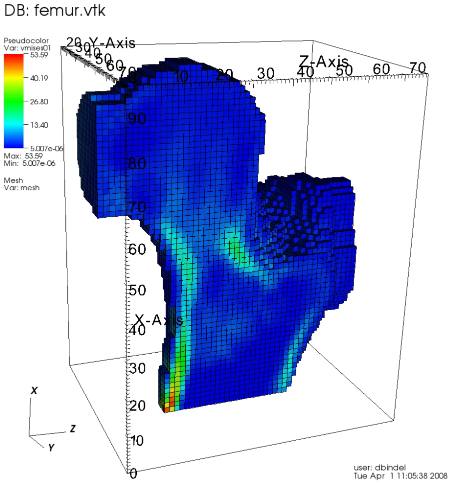
Add auxiliary variables to get

$$\begin{bmatrix} \phi_A \\ \phi_B \\ 0 \\ 0 \end{bmatrix} = \left[\begin{array}{cc|cc} P_{AA} & 0 & 0 & U_A \\ 0 & P_{BB} & U_B & 0 \\ \hline V_A^T & 0 & -I & 0 \\ 0 & V_B^T & 0 & -I \end{array}\right] \begin{bmatrix} m_A \\ m_B \\ z_A \\ z_B \end{bmatrix}.$$

Preliminary work

- Parallel sparsification routine (with Tim Mitchell)
  - User identifies low-rank blocks
  - Code factors the blocks and forms a sparse matrix as above
- Works pretty well on an example problem (charge on a capacitor)
- My goal state: Sparsification of separators for fast PDE solver

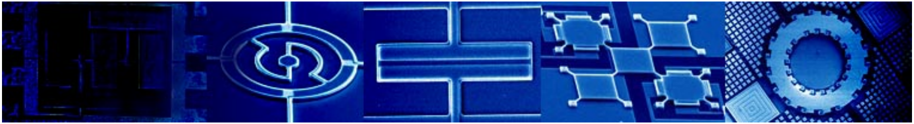
Goal state



I want a direct solver for this!

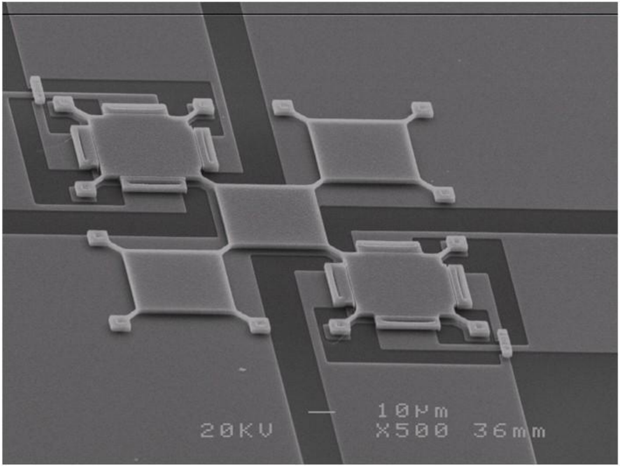
Outline

What are MEMS?



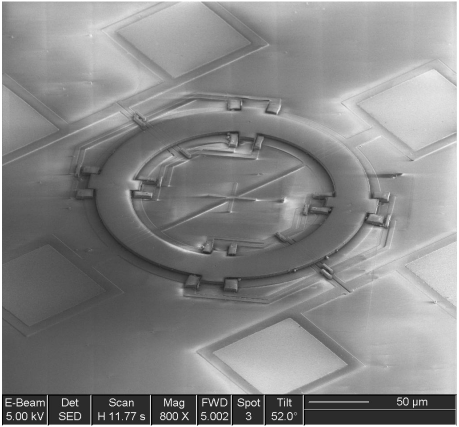
- MEMS = Micro-Electro-Mechanical Systems
- Applications:
  - Sensors (inertial, chemical, pressure)
  - Ink jet printers, biolab chips
  - Radio devices (cell phones, inventory tags, pico radio)
- Ongoing work with Sunil Bhave (ECE) and others on RF MEMS
- Fun source of matrices — and pictures!

Checkerboard resonator and loose coupling

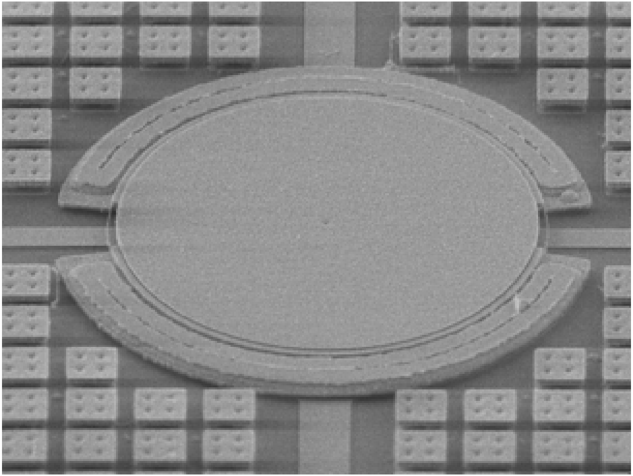


Bhave, Gau, Maboudian, Howe – MEMS 2005;
B., Bai, Demmel – PARA 2004

Shear ring resonator and symmetry

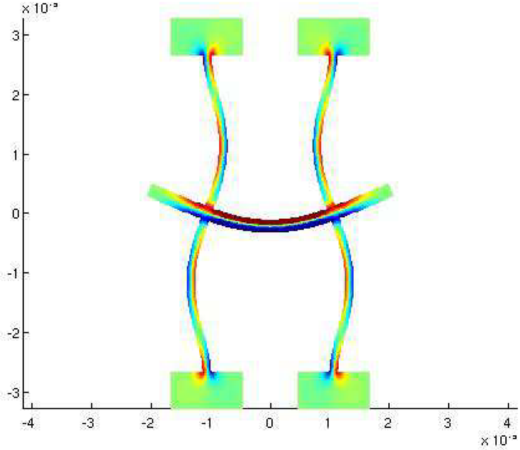


Disk resonator and anchor loss



B., Quevy, Koyama, Govindjee, Demmel, Howe – MEMS 2005;
B., Govindjee – IJNME 2005

Free beam resonator and thermoelastic damping



Koyama, B., He, Quevy, Demmel, Govindjee, Howe – SENSORS 2005

New questions...

- AFM probe tip testing
- Opto-mechanical interactions
- Anchor loss in more complex devices
- Solid dielectric transducers
- Interaction with dielectric liquids

Outline

The Computational Science Picture

Ingredients:

- Application modeling (computer systems, MEMS, ...)
- Mathematical analysis (linear algebra, PDEs, ...)
- Software engineering (finite elements, parallel solvers, ...)

Usually requires more than one person!

$\implies$ I'm usually looking for fun collaborations.

`http://www.cs.cornell.edu/~bindel`