Solving Laplacian Systems: Some Contributions from Theoretical Computer Science

Nick Harvey
UBC Department of Computer Science
Today’s Problem

• Solve the system of linear equations

\[ L x = b \]

where \( L \) is the **Laplacian** matrix of a graph.

*Think:* Symmetric, Diagonally Dominant

• **Assumptions:**
  – Want fast algorithms for sparse matrices: running time \( \approx \) # non-zero entries of \( L \)
  – Want provable, asymptotic running time bounds
  – Computational model: single CPU, random access memory, infinite-precision arithmetic
  – Ignore numerical stability, etc.
What is the Laplacian of a Graph?

\[ L = \begin{array}{cccc}
  2 & -1 & -1 & -1 \\
-1 & 2 & -1 & -1 \\
-1 & -1 & 3 & -1 \\
-1 & 1 & -1 & 1 \\
\end{array} \]

-1 indicates an edge from a to c

degree of node c = # edges that hit c

• \( L \) is symmetric, diagonally dominant
  ⇒ positive semidefinite
  ⇒ all eigenvalues real, non-negative

• \( \text{Kernel}(L) = \text{span}([1,1,...,1]) \) (assuming G connected)
Why is $L$ useful?

- What is effective resistance between $a$ and $d$?
  - Highschool Calculation: $1 + 1/(1+1/(1+1)) = 5/3$
  - Linear Algebra: $x^T L^+ x$, where $x = [1, 0, 0, -1]$

\[
L = \begin{bmatrix}
  2 & -1 & -1 \\
  -1 & 2 & -1 \\
  -1 & -1 & 3 \\
  -1 & 1 \\
\end{bmatrix}
\]

\[
L^+ = \begin{bmatrix}
  17 & 1 & -3 & -15 \\
  1 & 17 & -3 & -15 \\
  -3 & -3 & 9 & -3 \\
  -15 & -15 & -3 & 33 \\
\end{bmatrix} / 48
\]

Replace edges with 1-ohm resistors
Laplace’s Equation
Given a function $f : \Gamma \to \mathbb{R}$, find a function $g : \Omega \to \mathbb{R}$ such that

$$\nabla^2 g := \frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2} = 0$$
Discretization

\[ q^2 \nabla^2 g(e) \approx (g(f) - g(e)) - (g(e) - g(d)) + (g(b) - g(e)) - (g(e) - g(h)) \]
\[ = g(b) + g(d) + g(f) + g(h) - 4g(e) \]

So \( q^2 \nabla^2 g(v) \approx -L g(v) \) for all vertices \( v \) in grid interior

**Conclusion:** can compute approximate solution to Laplace’s equation by solving a linear system involving \( L \).
Aren’t existing algorithms good enough?

• Preconditioned Conjugate Gradient
  – Exact solution in $O(nm)$ time (L has size $n \times n$, $m$ non-zeros)
    • **Caveat**: this bound fails with inexact arithmetic
  – Better bounds with a good preconditioner?

• Multigrid
  – Provable performance for specific types of PDEs in low-dimensional spaces
  – Not intended for general “$Lx = b$” Laplacian systems

• Algebraic Multigrid
  – This is close to what we want
  – I am unaware of an existing theorem of the form:
    For any matrix from class $X$ we can efficiently compute coarsenings such that convergence rate is $Y$
Some Contributions from Theoretical Computer Science

- Let $L$ be Laplacian of a graph with $n$ vertices, max degree $\Delta$
  - **Key Idea:** Use Graph Theory to design preconditioners for $L$
  - Can find a preconditioner $P$ in $O((\Delta n)^{1.5})$ time such that:
    - relative condition number $\kappa(L, P) = O((\Delta n)^{1.5})$
    - solving systems in $P$ takes $O(\Delta n)$ time
  - Using conjugate gradient method, get a solution for “$Lx=b$” with relative error $\varepsilon$ in $O((\Delta n)^{1.75} \log(1/\varepsilon))$ time
  - i.e., let $x := L^+ b$ and let $y$ be the algorithm’s output
    Then $\left\|y-x\right\|_L \leq \varepsilon \left\|x\right\|_L$
    $$(y-x)^T L (y-x) \leq \varepsilon x^T L x$$
Some Contributions from Theoretical Computer Science

• Let $L$ be the Laplacian of a graph with $n$ vertices, $m$ edges

• Spielman-Teng [STOC ‘04]:
  – Can find a solution for “$Lx = b$” with relative error $\varepsilon$ in $O(m^{1+o(1)} \log(1/\varepsilon))$ time.
  – Can view as a rigorous implementation of Algebraic Multigrid
  – Highly technical: journal version is 116 pages

• Koutis-Miller-Peng [FOCS ‘11]:
  – Can find a solution for “$Lx = b$” with relative error $\varepsilon$ in $O(m \log n (\log \log n)^2 \log(1/\varepsilon))$ time.
  – Significantly simpler: only 16 pages

• **Ingredients:** low-stretch trees, random matrix theory
Iterative methods & Preconditioning

• Suppose you want to solve $Lx = b$ (m non-zeros)
• Instead choose a \textit{preconditioner} matrix $P$ and solve
  \[ P^{-1/2} LP^{-1/2} y = P^{-1/2} b \]
• Setting $x = P^{-1/2} y$ gives a solution to $Lx = b$
• The \textit{relative condition number} is
  \[ \kappa(L, P) = \frac{\lambda_{\text{max}}(P^{-1/2} LP^{-1/2})}{\lambda_{\text{min}}(P^{-1/2} LP^{-1/2})} = \frac{\lambda_{\text{max}}(P^{-1} L)}{\lambda_{\text{min}}(P^{-1} L)} \]
• Iterative methods (e.g., conjugate gradient) give
  – a solution with relative error $\epsilon$
  – after $O(\sqrt{\kappa(L, P) \log(1/\epsilon)})$ iterations
  – each iteration takes time $O(m + (\text{time to solve a linear system in } P))$
Tool #1: Low-Stretch Trees

Laplacians of Subgraphs

• Suppose you want to solve $Lx = b$, where $L$ is the Laplacian of graph $G=(V,E)$

• For any $F \subseteq E$, we can write $L = L_F + L_{E \setminus F}$, where
  - $L_F$ is the Laplacian of $(V,F)$
  - $L_{E \setminus F}$ is the Laplacian of $(V,E \setminus F)$

• Easy Fact: $\lambda_{\min}(L_F^+ L) \geq 1 \implies \kappa(L,L_F) \leq \lambda_{\max}(L_F^+ L)$

$$L = \begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 3 & -1 \\ -1 & 1 \end{bmatrix}$$
$$L_F = \begin{bmatrix} 2 & -1 & -1 \\ -1 & 1 \end{bmatrix}$$
$$L_{E \setminus F} = \begin{bmatrix} 1 & -1 \\ -1 & 2 & -1 \\ -1 & 1 \end{bmatrix}$$
Subtree Preconditioners

• Let $L$ be the Laplacian of $G=(V,E)$ \((n = \#\text{vertices}, m = \#\text{edges})\)
• Let $T=(V,F)$ be a sub\textbf{tree} of $G$
• Consider using $P=L_F$ as a preconditioner
• **Useful Property:** Solving linear systems in $P$ takes $O(n)$ time, essentially by back-substitution
Subtree Preconditioners

- Let $L$ be the Laplacian of $G=(V,E)$ \hspace{1cm} (n = #vertices, m = #edges)
- Let $T=(V,F)$ be a subtree of $G$
- Consider using $P=L_F$ as a preconditioner
- **Useful Property:** Solving linear systems in $P$ takes $O(n)$ time, essentially by back-substitution
- **Easy Fact:** $\kappa(L,P) \leq \lambda_{\max}(P^+L)$
- CG gives a solution with relative error $\epsilon$ in time
  \[
  O(m \sqrt{\lambda_{\max}(P^{-1}L) \log(1/\epsilon)})
  \]
Low-Stretch Trees

- Let $L$ be the Laplacian of $G=(V,E)$ ($n = \#\text{vertices}, m = \#\text{edges}$)
- Let $T=(V,F)$ be a subtree of $G$ and $P=L_F$
- **Lemma** [Boman-Hendrickson ’01]: $\lambda_{\max}(P^{-1} L) \leq \text{stretch}(G,T)$, where $\text{stretch}(G,T) = \sum_{e=(u,v) \in E} \text{distance between } u \text{ and } v \text{ in } T$
- **Theorem** [Alon-Karp-Peleg-West ’91]: Every graph $G$ has a tree $T$ with $\text{stretch}(G,T) = m^{1+o(1)}$. 

![Diagram of graphs G and T with distance from u to v highlighted]
Low-Stretch Trees

• Let $L$ be the Laplacian of $G=(V,E)$ \( (n = \#\text{vertices}, m = \#\text{edges}) \)

• Let $T=(V,F)$ be a subtree of $G$ and $P=L_F$

• **Lemma** [Boman-Hendrickson ‘01]: $\lambda_{\max}(P^{-1} L) \leq \text{stretch}(G,T)$, where $\text{stretch}(G, T) = \sum_{e=(u,v) \in E} \text{distance between } u \text{ and } v \text{ in } T$

• **Theorem** [Alon-Karp-Peleg-West ‘91]: Every graph $G$ has a tree $T$ with $\text{stretch}(G,T) = m^{1+o(1)}$.

• **Theorem** [Abraham-Bartal-Neiman ‘08]: Every $G$ has a tree $T$ with $\text{stretch}(G,T) \leq m \log n (\log \log n)^2$. Moreover, $T$ can be found in $O(m \log^2 n)$ time.
Low-Stretch Trees

- Let $L$ be the Laplacian of $G=(V,E)$ \hspace{1cm} (n = #vertices, m = #edges)
- Let $T=(V,F)$ be a subtree of $G$ and $P=L_F$
- **Lemma** [Boman-Hendrickson ‘01]: $\lambda_{\text{max}}(P^{-1} L) \leq \text{stretch}(G,T)$, where
  \[
  \text{stretch}(G,T) = \sum_{e=(u,v) \in E} \text{distance between } u \text{ and } v \text{ in } T
  \]
- **Theorem** [Alon-Karp-Peleg-West ‘91]:
  Every graph $G$ has a tree $T$ with $\text{stretch}(G,T) = m^{1+o(1)}$
  and $T$ can be found in $O(m \log^2 n)$ time.
- **Corollary**: Every Laplacian $L$ has a preconditioner $P$ s.t.
  - $\kappa(L,P) \leq \lambda_{\text{max}}(P^{-1} L) \leq m^{1+o(1)}$
  - CG gives $\epsilon$-approx solution in time
    \[
    O(m \sqrt{\kappa(L,P)} \log(1/\epsilon)) = O(m^{3/2+o(1)} \log(1/\epsilon))
    \]
  - Tighter analysis gives $O(m^{4/3+o(1)} \log(1/\epsilon))$ \hspace{1cm} [Spielman-Woo ‘09]
Tool #2: Random Sampling

Spectral Sparsifiers

- Let $L_G$ be the Laplacian of $G=(V,E)$ ($n =$ #vertices, $m =$ #edges)
- A spectral sparsifier of $G$ is a (weighted) graph $H=(V,F)$ s.t.
  - $|F|$ is small
  - $1-\epsilon \leq \lambda_{\text{min}}(L_H^+ L)$ and $\lambda_{\text{max}}(L_H^+ L) \leq 1+\epsilon$
- Useful notation: $(1-\epsilon) L_G \preceq L_H \preceq (1+\epsilon) L_G$
- Consider the linear system $L_G x = b$.
  Actual solution is $x := L_G^+ b$.
  Instead, compute $y := L_H^+ b$.
- Then $y$ has low multiplicative error: $\|y-x\|_{L_G} \leq 2\epsilon \|x\|_{L_G}$
- Computing $y$ is fast since $H$ is sparse: conjugate gradient method takes $O(n|F|)$ time
Spectral Sparsifiers

• Let $L_G$ be the Laplacian of $G=(V,E)$ \hspace{1cm} (n = \#vertices, m = \#edges)

• A spectral sparsifier of $G$ is a (weighted) graph $H=(V,F)$ s.t.
  • $|F|$ is small
  • $(1-\epsilon)L_G \preceq L_H \preceq (1+\epsilon)L_G$

• Theorem [Spielman-Srivastava ‘08]: Every $G$ has a spectral sparsifier $H$ with $|F| = O(n \log n / \epsilon^2)$. Moreover, $H$ can be constructed in $O(m \log^3 n)$ time.

• Algorithm: Using CG, we get an $\epsilon$-approx solution to “$Lx = b$” in $O(m \log^3 n + n^2 \log n / \epsilon^2)$ time
  – Caveat: this algorithm uses circular logic. To construct the sparsifier $H$, we need to solve several linear systems “$Lx = b$".
Decomposing Laplacian into Edges

- Let $L_G$ be the Laplacian of $G=(V,E)$
- For every $e \in E$, let $L_e$ be the Laplacian of $(V,\{e\})$
- Then $L_G = \sum_e L_e$

![Graph with nodes a, b, c, d]

$$L_G = \begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 3 & -1 & -1 & 1 \end{bmatrix}$$

$L_a b$

$$\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

$L_a c$

$$\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

$L_b c$

$$\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

$L_c d$

$$\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$
Decomposing Laplacian into Edges

• Let $L_G$ be the Laplacian of $G=(V,E)$
• For every $e \in E$, let $L_e$ be the Laplacian of $(V,\{e\})$
• Then $L_G = \sum_e L_e$
• **Sparsification:**
  – Find coefficients $w_e$ for every $e \in E$
  – Let $H$ be the weighted graph where $e$ has weight $w_e$
  – So $L_H = \sum_e w_e L_e$
• **Goals:**
  • Most of the $w_e$ are zero
  • $(1-\epsilon) L_G \leq L_H \leq (1+\epsilon) L_G$
The General Problem: Sparsifying Sums of PSD Matrices

• **General Problem:** Given PSD matrices $L_e$ s.t. $\sum_e L_e = L$, find coefficients $w_e$, mostly zero, such that

$$(1-\epsilon) L \preceq \sum_e w_e L_e \preceq (1+\epsilon) L$$

• **Theorem:** [Ahlswede-Winter ’02]
  Random sampling gives $w$ with $O( n \log n/\epsilon^2 )$ non-zeros.

• **Theorem:** [de Carli Silva-Harvey-Sato ‘11], building on [Batson-Spielman-Srivastava ‘09]
  There exists $w$ with $O( n/\epsilon^2 )$ non-zeros.
Concentration Inequalities

• **Theorem:** [Chernoff ‘52, Hoeffding ‘63]
  Let $Y_1, \ldots, Y_k$ be i.i.d. random non-negative real numbers s.t. $E[Y_i] = Z$ and $Y_i \leq uZ$. Then
  \[
  \Pr \left[ (1 - \epsilon)Z \leq \sum_{i=1}^{k} \frac{Y_i}{k} \leq (1 + \epsilon)Z \right] \geq 1 - 2 \exp \left( - \Omega(\epsilon^2 k / u) \right)
  \]

• **Theorem:** [Ahlswede-Winter ‘02]
  Let $Y_1, \ldots, Y_k$ be i.i.d. random PSD $n \times n$ matrices s.t. $E[Y_i] = Z$ and $Y_i \leq uZ$. Then
  \[
  \Pr \left[ (1 - \epsilon)Z \leq \sum_{i=1}^{k} \frac{Y_i}{k} \leq (1 + \epsilon)Z \right] \geq 1 - 2n \exp \left( - \Omega(\epsilon^2 k / u) \right)
  \]

The only difference
Solving the General Problem

- **General Problem:** Given PSD matrices $L_e$ s.t. $\sum_e L_e = L$, find coefficients $w_e$, mostly zero, such that
  $$(1-\epsilon) L \preceq \sum_e w_e L_e \preceq (1+\epsilon) L$$

- **AW Theorem:** Let $Y_1, \ldots, Y_k$ be i.i.d. random PSD matrices such that $E[Y_i] = Z$ and $Y_i \preceq uZ$. Then
  $$\Pr\left[ (1 - \epsilon) Z \preceq \sum_{i=1}^k \frac{Y_i}{k} \preceq (1 + \epsilon) Z \right] \geq 1 - 2n \exp\left( - \Omega(\epsilon^2 k/u) \right)$$

- To solve General Problem with $O(n \log n / \epsilon^2)$ non-zeros
  - Repeat $k := \Theta(n \log n / \epsilon^2)$ times
  - Pick an edge $e$ with probability $p_e := \frac{\text{Tr}(L_e L^+)}{n}$
  - Increment $w_e$ by $1/k \cdot p_e$

**Main Caveat:** Sampling probabilities are hard to compute.
Low-Stretch Trees
+ Random Sampling
+ Recursion

= Nearly-Optimal Algorithm
Obstacles encountered so far

1. Low-stretch trees are easy to compute but only give preconditioners with $\kappa \approx m$
   - **Low-stretch tree:** fast, low-quality preconditioner

2. Sparsifiers give preconditioners with $\kappa \approx 1+\epsilon$, but they are harder to compute
   - **Sparsifier:** slow, high-quality preconditioner

3. Using a sparsifier $H$ as a preconditioner is not very efficient because solving $L_H^+ x = b$ is slow
   - **Sparsifier:** slow to construct *and* slow to use
Bypassing the Obstacles

- **Idea #1**: Get a “medium-quality” preconditioner by combining the low- and high-quality preconditioners.
- Specifically, do random sampling according to stretch
- **Intuition**: The path is a bad approximation to the cycle

\[ K(L_G, L_T) = n \]

High condition number caused by missing edge, which has high stretch.
Bypassing the Obstacles

- **Idea #1:** Get a “medium-quality” preconditioner by combining the low- and high-quality preconditioners.

- Specifically, do random sampling according to stretch

- Compute a low-stretch tree $T$. For every edge $uv$, set

$$p_{uv} = \frac{\text{distance between } u \text{ and } v \text{ in } T}{\text{stretch}(G, T)}$$

- Construct sparsifier $H$ by making $O(m / \log^2 n)$ samples, where $e$ is sampled with probability $p_e$. Then add $\log^4 n$ copies of $T$ to $H$.

- **Theorem** [Koutis, Miller, Peng FOCS’10]:
  - $H$ has at most $O(m / \log^2 n)$ edges
  - $L_G \leq L_H \leq \log^4 n \ L_G$
• **Idea #2 [Joshi ‘97, Spielman-Teng ‘04]:** To solve linear systems in the sparsifier “$L_H x = b$”, use recursion.

Solving $L_G x = b$
- Construct sparsifier $H_1$
- Recursively solve $L_{H_1} x = b$
- Use Chebyshev iterations to improve $x$
- Output $\epsilon$-approx solution to $L_G x = b$

Solving $L_{H_1} x = b$
- Construct sparsifier $H_2$
- Recursively solve $L_{H_2} x = b$
- Use Chebyshev iterations to improve $x$
- Output $\epsilon$-approx solution to $L_{H_1} x = b$

Solving $L_{H_2} x = b$
- Output $\epsilon$-approx solution to $L_{H_2} x = b$

$L_G \preceq L_{H_1} \preceq \log^4 n L_G$

$L_{H_1} \preceq L_{H_2} \preceq \log^4 n L_{H_1}$

$L_{H_2} \preceq L_{H_3} \preceq \log^4 n L_{H_2}$

...
Idea #2 [Joshi ‘97, Spielman-Teng ‘04]: To solve linear systems in the sparsifier “$L_H x = b$”, use recursion.

**Sketch of Analysis:**
- Few Chebyshev iterations because $H_{i+1}$ is a good approximation of $H_i$.
- Few levels of recursion because $H_{i+1}$ is a constant factor smaller than $H_i$.

### Solving $L_G x = b$
- Construct sparsifier $H_1$
- Recursively solve $L_{H_1} x = b$
- Use Chebyshev iterations to improve $x$
- Output $\epsilon$-approx solution to $L_G x = b$

### Solving $L_{H_1} x = b$
- Construct sparsifier $H_2$
- Recursively solve $L_{H_2} x = b$
- Use Chebyshev iterations to improve $x$
- Output $\epsilon$-approx solution to $L_{H_1} x = b$
Conclusion

• Let A be a symmetric, diagonally dominant matrix of size nxn with m non-zero entries.
• There is an algorithm to solve “Ax = b” with relative error $\epsilon$ in $O(m \log n (\log \log n)^2 \log(1/\epsilon))$ time.
• Ingredients: Low-stretch trees, concentration of random matrices

Open Questions

• Parallelization?
• Practical implementation?
• Numerical stability?
• Arbitrary PSD matrices?