Combinatorial Randomized Kaczmarz: An Algebraic Perspective

Sivan Toledo[‡]
in collaboration with
Haim Avron[†]
Haim Kaplan[‡]

[‡]Tel-Aviv University
[†]IBM TJ Watson Research Center

Perspectives on a Randomized Graph Algorithm for Linear Equations

Sivan Toledo‡
in collaboration with
Haim Avron†
Haim Kaplan‡

‡Tel-Aviv University †IBM TJ Watson Research Center

My gift to the Simons Institute for the Theory of Computing is this insight:

Practitioners can take theoretical results and turn them into fantastic software.

My gift to the
Simons Institute for the Theory of Computing
is this insight:

In theory,
Practitioners can take theoretical results and turn them into fantastic software.

My gift to the
Simons Institute for the Theory of Computing
is this insight:

In theory,
Practitioners can take theoretical results and turn them into fantastic software.
In practice, this is often impossible or hard.

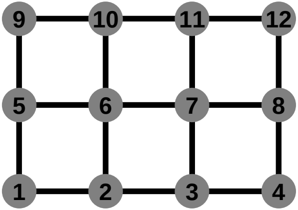
Problem Setup

Solve $Ax = b$ with A symmetric, sparse, large, semidefinite

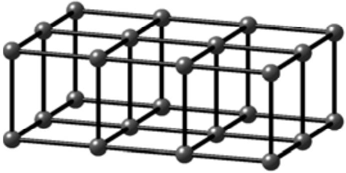
Problem Setup (More Structure)

Solve $Ax = b$ with A symmetric, sparse, large, semidefinite **and diagonally dominant**

Laplacians of Graphs



$$A = \begin{bmatrix}
3 & -1 & & & -1 & & & & & & & \\
-1 & 3 & -1 & & & -1 & & & & & & \\
 & -1 & 3 & -1 & & & -1 & & & & & \\
 & & -1 & 2 & & & & -1 & & & & \\
-1 & & & & 3 & -1 & & & -1 & & & \\
 & -1 & & & -1 & 4 & -1 & & & -1 & & \\
 & & -1 & & & -1 & 4 & -1 & & & -1 & \\
 & & & -1 & & & -1 & 3 & & & & -1 \\
 & & & & -1 & & & & 2 & -1 & & \\
 & & & & & -1 & & & -1 & 3 & -1 & \\
 & & & & & & -1 & & & -1 & 3 & -1 \\
 & & & & & & & & & & -1 & 2
\end{bmatrix}$$



Ancient History: Direct Solvers, Dense and Sparse, (Preconditioned) Conjugate Gradients

Direct Solvers do not Scale Well

Gauss (Cholesky): factor $A = LL^T$, L triangular, $O(n^3)$ arithmetic

Sparse solvers: factor $A = PLL^TP^T$, P permutation

- $O(n^{1.5})$ for 2D meshes,
- $O(n^2)$ for 3D meshes,
- in general depends on size of approximately-balanced vertex separators

Conjugate Gradients (and Variants)

Iteratively improve an approximate solution $x^{(t)} \to x$

Cost of each iteration is $O(\text{nnz}(A)) = O(\#\text{ edges}) = O(m)$

Convergence (number of iterations) depends on the distribution of eigenvalues $\Lambda(A)$

- Faster on 3D meshes than on 2D because eigenvalues in 3D are more clustered
- Fantastic on expanders but too slow on meshes

The Idea of Preconditioning (Still Ancient)

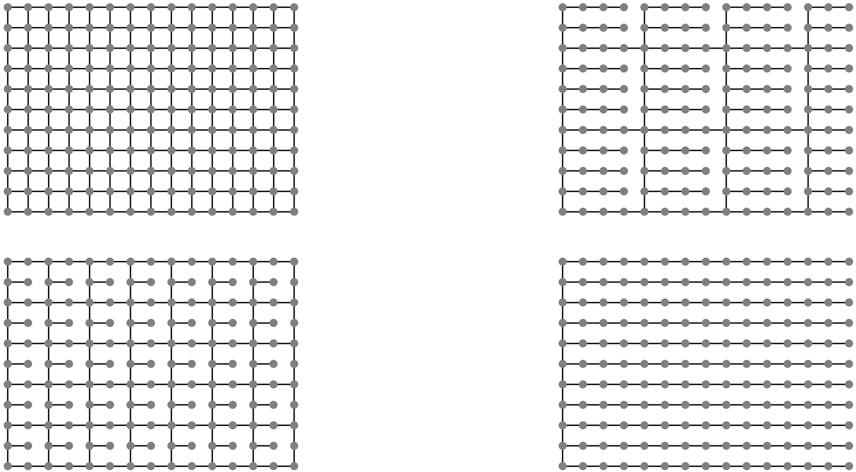
Find an approximation P to A, factor $P = LL^T$ and run CG on

$$\left(L^{-1}AL^{-T}\right)\left(L^T x\right) = L^{-1}b$$

Works well if P is easy to factor and if $\Lambda\left(L^{-1}AL^{-T}\right)$ is distributed better than $\Lambda(A)$

1991-2001: Combinatorial Preconditioners for CG

Combinatorial Subgraph Preconditioners



Reasoning about Convergence Rates: Incidence Factors

Consider the incidence factors $A = UU^T$,

$$A_{ij} = A_{ji} \neq 0 \quad \Longrightarrow \quad U_e = \begin{bmatrix} \vdots \\ \sqrt{A_{ij}} \\ \vdots \\ -\sqrt{A_{ij}} \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ \rho_e \\ \vdots \\ -\rho_e \\ \vdots \end{bmatrix}$$

$U \in \mathbb{R}^{n \times m}$, columns have 2 nonzeros & correspond to edges in the graph of A

Convergence Rates: Combinatorial Spectral Bounds for Subgraphs

Let $A = UU^T$, $U = [U_B \; U_N]$, and $P = U_B U_B^T$

Use a graph embedding to create W such as $U = U_B W$

Can show that $\Lambda\left(L^{-1} A L^{-T}\right) \subseteq [1, \|W\|_2]$

Combinatorial properties of the embeddings (congestion, dilation, stretch) can be used to bound $\|W\|_2$

There's looseness in the combinatorial bounds, but they are useful

[Boman & Hendrickson, 2003]

Constructing the Subgraph

1991: A maximum spanning tree (MST); Augmented MST [Vaidya]

1997: Explicit construction for regular meshes [Joshi]

2001: Low-stretch trees (can't easily augment) [Boman & Hendrickson]

2004+: Super-complicated constructions [Spielman & Teng]

2010+: Simpler but not in this talk [Koutis, Miller, & Peng]

Lots of side shows (Toledo & others)

2013: Low-Stretch Trees without Conjugate Gradients

Kelner, Orecchia, Sidford, Zhu (STOC 2013)

Kelner, Orecchia, Sidford, Zhu (STOC 2013)

New Algorithm

Iterative but does not use Conjugate Gradients or its relatives

Uses low-stretch trees

A new kind of analysis; does not build on earlier results

Analysis based on “electrical” reasoning (was used before as motivation, but not as an analytical tool)

We will show an alternative algebraic analysis

The Big Picture

$Ax = UU^T x = b$

Given U, construct a basis N for its null space $UN^T = 0$, define

$$K = \begin{bmatrix} U \\ N \end{bmatrix} \text{ (square and full rank)}$$

Find a vector f that satisfies

$$Kf = \begin{bmatrix} U \\ N \end{bmatrix} f = \begin{bmatrix} b \\ 0 \end{bmatrix}$$

Because $Nf = 0$, there is an x such that $U^T x = f$

Find that x (easy)

Because $Uf = b$ we have $Uf = UU^T x = Ax = b$

The Big Picture

$Ax = UU^{\mathsf{T}}x = b$

Given U, construct a basis N for its null space $UN^{\mathsf{T}} = 0$, define

$$K = \begin{bmatrix} U \\ N \end{bmatrix} \text{ (square and full rank)}$$

Find a vector f that satisfies

$$Kf = \begin{bmatrix} U \\ N \end{bmatrix} f = \begin{bmatrix} b \\ 0 \end{bmatrix}$$

Because $Nf = 0$, there is an x such that $U^{\mathsf{T}}x = f$

Find that x (easy)

Because $Uf = b$ we have $Uf = UU^{\mathsf{T}}x = Ax = b$

Obviously only makes sense if $Kf = g$ is easy

Constraint-Relation Iterations

Pick a constraint (row) in a linear system $Kf = g$

Modify the approximate solution to satisfy this constraint

Repeat

Jacobi: correct $K_i^T f = g_i$ by modifying f_i

Constraint Relation Iterations

Pick a constraint (row) in a linear system $Kf = g$

Modify the approximate solution to satisfy this constraint

Repeat

Jacobi: correct $K_i^T f = g_i$ by modifying f_i (adding αe_i to f)

Constraint Relation Iterations

Pick a constraint (row) in a linear system $Kf = g$

Modify the approximate solution to satisfy this constraint

Repeat

Kaczmarz: correct by adding αK_i to f (projection)

Jacobi: correct $K_i^T f = g_i$ by modifying f_i (adding αe_i to f)

Kaczmarz and our Augmented System

Find an initial $f^{(0)}$ that

$$Kf^{(0)} = \begin{bmatrix} U \\ N \end{bmatrix} f^{(0)} = \begin{bmatrix} b \\ z \neq 0 \end{bmatrix}$$

(easy); run Kaczmarz; show it maintains the invariant

$$Kf^{(t)} = \begin{bmatrix} U \\ N \end{bmatrix} f^{(t)} = \begin{bmatrix} b \\ \tilde{z} \to 0 \end{bmatrix}$$

Kaczmarz picks a row in K, say i

If the row is in U we have $U_i^T f^{(t)} = b_i$ so $\alpha = 0$; do nothing

If the row is in N, $f^{(t+1)} = f^{(t)} + \alpha N_i^T$ so
$U_i^T f^{(t+1)} = U_i^T (f^{(t)} + \alpha N_i^T) = U_i^T f^{(t)} + 0 = b_i$

$\Rightarrow$ We converge on $\tilde{z}$ without messing up $Uf^{(t)} = b$

Three Hard Interrelated Challenges

1. How to construct N?
2. How many Kaczmarz iterations will the algorithm do?
3. Running Kaczmarz iterations super efficiently

Constructing the Null-Space Basis

Split $U = [U_B \; U_N]$, U_B represents a spanning tree

U_B is a basis for the column space of U, so

$$U = [U_B \; U_N] = U_B W = U_B [I \; W_N]$$

Theorem: $N = [-W_N^T \quad I]$ is a null-space basis

Proof:

$$\begin{aligned}
\begin{bmatrix} U_B & U_N \end{bmatrix} \begin{bmatrix} -W_N \\ I \end{bmatrix} &= -U_B W_N + U_N \\
&= -U_N + U_N \\
&= 0
\end{aligned}$$

But how do you construct W_N?

Constructing W from a Path Embedding

A weighted path in the tree specifies a column of W_N,

$$U_e = \begin{bmatrix} \vdots \\ \rho_e \\ \vdots \\ -\rho_e \\ \vdots \end{bmatrix} = \sum_{e' \in \text{path}} W_{e',e} U_{e'} = \sum_{e' \in G_P} \pm \frac{\rho_e}{\rho_{e'}} U_{e'}$$

How makes a tree “good”? Fast Kaczmarz convergence

Convergence of Randomized Kaczmarz

Randomized Kaczmarz:

- pick row i at random w/probability $\propto \|K_i^T\|^2$

Randomized Kaczmarz converges in $O(\|K\|_F \|K^{-1}\|_2)$ iterations

For us, what matters is only the N part, because Kaczmarz never visits the U rows

We need an N with a small sum-of-squares that cannot make vectors small

The Condition Number of the Null-Space Basis

We need $N = [-W_N^T \quad I]$ with a small sum-of-squares that cannot make vectors small

$\sigma_{\min}(N) \geq 1$ because of the I block

$$\begin{aligned}
\|N\|_F^2 &= \|W_N\|_F^2 + (m-n) = \sum_{e,e' \in \text{embedding}} \left(\frac{\rho_e}{\rho_{e'}}\right)^2 + (m-n) \\
&= \text{stretch of tree} + (m-n)
\end{aligned}$$

A low-stretch tree guarantees $\|N\|_F = \tilde{O}(m)$

A maximum spanning tree guarantees $(\rho_e/\rho_{e'}) \leq 1$; nice but not as good

Super-Efficient Projections (I)

Kaczmarz: pick a non-tree edge i correct by adding αK_i^T to f (projection),

$$f^{(t+1)} = f^{(t)} + \alpha K_i^T$$

with α chosen to ensure $K_i^T f^{(t+1)} = g_i$, or

$$\alpha = \frac{g_i - \sum_j K_{ij} f_j^{(0)} - \sum_j K_{ij} f_j^{(t)}}{\sum_j K_{ij}^2}$$

Only $\sum_j K_{ij} f_j^{(t)}$ changes every iteration; the summation is over a tree path

Super-Efficient Projections (II): The Key Idea

Create a data structure that represents $f^{(t)}$ on tree edges and supports the following operations:

- Evaluate $\sum_j K_{ij} f_j^{(t)}$ (summation over a path)
- Update $f^{(t+1)} = f^{(t)} + \alpha K_i^T$
- Output $f^{(\text{final})}$ explicitly (at teardown)

This is nontrivial because coefficients depend on path (i), not only on edge (j)

Super-Efficient Projections (III): The Data Structure

An algebraic transformation allows us to remove path dependency:

- Evaluate $\sum_j \beta_j f_j^{(t)}$ (summation over a path)
- Update $f^{(t+1)} = f^{(t)} + \alpha K_i^T$
- Output $f^{(\text{final})}$ explicitly (at teardown)

$O(\log n)$ per operation using a hierarchical data structure derived from Sleator and Tarjan's dynamic trees

Summary, Experiences, & Outlook

Not directly related to Vaidya & followups

Low-stretch trees are useful in both frameworks for basically the same reasons

Two key ideas:

- A combinatorial/null-space method to transform an ill-conditioned problem to a larger well conditioned one
- An implicit representation of the iteration vector ensure $O(\log n)$ operations

Convergence appears to be very slow (large constants)

Can this algorithm be parallelized?