Graph Algorithms in the Language of Linear Algebra: How did we get here, where do we go next?

John R. Gilbert
University of California, Santa Barbara

IPDPS Graph Algorithms Building Blocks
May 21, 2018

Support: Intel, Microsoft, DOE Office of Science, NSF
George Pólya on how to give a mathematical talk

(as described by John Todd)

“Pólya’s recipe was as follows: The first quarter should be understandable to absolutely everyone, the second quarter should include kind words about your friends (especially those in the audience), and then it doesn’t matter what you say in the last half hour.”
George Pólya on how to give a mathematical talk

(as described by John Todd)

“Pólya’s recipe was as follows: The first quarter should be understandable to absolutely everyone, the second quarter should include kind words about your friends (especially those in the audience), and then it doesn’t matter what you say in the last half hour.”

“I [Todd] adjust this by adding, sit down after a quarter hour.”
In the year 1961 ...
Prehistory: A 1-person game on graphs

[S. Parter 1961]
Prehistory: A 1-person game on graphs

- Mark a vertex.
Prehistory: A 1-person game on graphs

- Mark a vertex.
- Connect its unmarked neighbors.
Prehistory: A 1-person game on graphs

- Mark a vertex.
- Connect its unmarked neighbors.
- Repeat.
Prehistory: A 1-person game on graphs

- Mark a vertex.
- Connect its unmarked neighbors.
- Repeat.
Prehistory: A 1-person game on graphs

• Mark a vertex.
• Connect its unmarked neighbors.
• Repeat.
Prehistory: A 1-person game on graphs

- Mark a vertex.
- Connect its unmarked neighbors.
- Repeat.
Prehistory: A 1-person game on graphs

- Mark a vertex.
- Connect its unmarked neighbors.
- Repeat.
Prehistory: A 1-person game on graphs

• Mark a vertex.
• Connect its unmarked neighbors.
• Repeat.
Prehistory: A 1-person game on graphs

Mark a vertex.
Connect its unmarked neighbors.
Repeat.

Goal: End up with as few edges as possible.
Prehistory: A 1-person game on graphs

- Mark a vertex.
- Connect its unmarked neighbors.
- Repeat.

Goal: End up with as few edges as possible.
Prehistory: A 1-person game on graphs

- Mark a vertex.
- Connect its unmarked neighbors.
- Repeat.

Goal: End up with as few edges as possible.
Prehistory: A 1-person game on graphs

- Mark a vertex.
- Connect its unmarked neighbors.
- Repeat.

Goal: End up with as few edges as possible.
Prehistory: A 1-person game on graphs

- Mark a vertex.
- Connect its unmarked neighbors.
- Repeat.

Goal: End up with as few edges as possible.
Prehistory: A 1-person game on graphs

• Mark a vertex.
• Connect its unmarked neighbors.
• Repeat.

Goal: End up with as few edges as possible.
Prehistory: A 1-person game on graphs

- Mark a vertex.
- Connect its unmarked neighbors.
- Repeat.

Goal: End up with as few edges as possible.
Vertex elimination game (or chordal completion)  
[Parter, Rose]

Repeat:
- Choose a vertex \(v\) and mark it;
- Add edges between unmarked neighbors of \(v\);

Until: Every vertex is marked

Goal: End up with as few edges as possible.

- Best play is NP-complete [Yannakakis 1981]
- The final graph is always chordal (every cycle has a shortcut edge).
- Perfect play is possible iff the initial graph is chordal.
- Changing “fewest edges” to “smallest complete subgraph” gives the graph’s treewidth, which shows up in lots of graph algorithms.
“I observed that most of the coefficients in our matrices were zero; i.e., the nonzeros were ‘sparse’ in the matrix, and that typically the triangular matrices associated with the forward and back solution provided by Gaussian elimination would remain sparse if pivot elements were chosen with care.”

- Harry Markowitz, describing the 1950s work on portfolio theory that won the 1990 Nobel Prize for Economics
Cholesky factorization: $A = LL^T$

[Parter, Rose]

Symmetric Gaussian elimination:
for $j = 1$ to $n$
add edges between $j$'s higher-numbered neighbors

Fill: new nonzeros in factor

$G(A)$

$G^+(A)$ [chordal]
Complexity measures for chordal completion

\[ G^+(A) \]

**Nonzeros** = **edges** = \( \sum_j d_j \) (moment 1)

**Work** = **flops** = \( \sum_j (d_j)^2 \) (moment 2)

**Front size ~ fast memory** = \( \max_j d_j \) (moment \( \infty \))

Elimination degree:

\[ d_j = \# \text{ higher neighbors of } j \text{ in } G^+ \]

\[ d = (2, 2, 2, 2, 2, 2, 1, 2, 1, 0) \]

(minimum possible front size is the same as treewidth)
Aside: Matrix structure prediction

- Computing the nonzero structure of Cholesky factor $L$ is much cheaper than computing $L$ itself.
- Cost to compute $\text{nnz}(L)$ is almost linear in $\text{nnz}(A)$. [G, Ng, Peyton]

Lots of cool recent work on sampling algorithms to estimate statistics of matrix functions.
Aside: Matrix structure prediction

- Computing the nonzero structure of Cholesky factor \( L \) is much cheaper than computing \( L \) itself.
- Cost to compute \( \text{nnz}(L) \) is almost linear in \( \text{nnz}(A) \). \([G, Ng, Peyton]\)
- Not so for sparse matrix product (SpGEMM); computing \( \text{nnz}(B*C) \) seems to be as hard as computing \( B*C \).
Aside: Matrix structure prediction

- Computing the nonzero structure of Cholesky factor $L$ is much cheaper than computing $L$ itself.

- Cost to compute $\text{nnz}(L)$ is almost linear in $\text{nnz}(A)$. [G, Ng, Peyton]

- Not so for sparse matrix product (SpGEMM); computing $\text{nnz}(B*C)$ seems to be as hard as computing $B*C$.

- Can estimate $\text{nnz}(B*C)$ accurately in time linear in $\text{nnz}(B, C)!$ [E. Cohen 1998]

- Lots of cool recent work on sampling algorithms to estimate statistics of matrix functions.
Orderings for sparse Gaussian elimination

\[
A x = b
\]

\[
A = L_1 L_1^T
\]

\[
(PAP^T) (Px) = (Pb)
\]

\[
PAP^T = L_2 L_2^T
\]
Nested dissection and graph partitioning

[George 1973, many extensions]

- Heuristic: Find small vertex separator, put it last, recurse on subgraphs
- Theory: Approx optimal separators $\Rightarrow$ approx optimal fill
- Practice: Lots of work on heuristics for graph partitioning!
Many, many graph algorithms have been used, invented, implemented at large scale for sparse matrix computation:

- **Symmetric problems:** elimination tree, nonzero structure prediction, sparse triangular solve, sparse matrix-matrix multiplication, min-height etree, …

- **Nonsymmetric problems:** sparse triangular solve, bipartite matching (weighted and unweighted), Dulmage-Mendelsohn decomposition / strong components, …

- **Iterative methods:** graph partitioning again, independent set, low-stretch spanning trees, …
In the year 1992 ...
History: Mesh partitioning for scientific computing, circa 1992

- Both for nested dissection and for parallel sparse matvec
- Spectral partitioning: Laplacian eigenvectors
- Recursive coarsening: Chaco \cite{HendricksonLeland}, Metis \cite{KarypisKumar}
- ...
History: Mesh partitioning for scientific computing, circa 1992

- Both for nested dissection and for parallel sparse matvec
- Spectral partitioning: Laplacian eigenvectors
- Recursive coarsening: Chaco \cite{HendricksonLeland}, Metis \cite{KarypisKumar}
- Geometric partitioning: Shang-Hua Teng’s PhD thesis ...
History: Mesh partitioning for scientific computing, circa 1992

- Both for nested dissection and for parallel sparse matvec
- Spectral partitioning: Laplacian eigenvectors
- Recursive coarsening: Chaco \cite{HendricksonLeland, Metis} [Karypis/Kumar]
- Geometric partitioning: Shang-Hua Teng’s PhD thesis ...
- ... and sparse matrices had just been added to Matlab ...
Geometric partitioning in Matlab \cite{G, Miller, Teng}

1. Original Mesh

2. Mesh Points

3. Stereographic Projection

4. Conformal Mapping

5. Projected Back Down

6. Partitioned Mesh
In the year 2002
(and soon after) ...
In the year 2002
(and soon after) ... 

(In 2002, JRG shared an office with Jeremy Kepner at MIT.)
Many tight clusters, loosely interconnected
Input data is edge triples \(<i, j, a>\)
Vertices and edges permuted randomly
Greedy clustering by breadth-first search

• Grow local clusters from many seeds in parallel
• Breadth-first search by sparse matrix * matrix
• Cluster vertices connected by many short paths

% Grow each seed to vertices
% reached by at least k paths of length 1 or 2

C = sparse(seeds, 1:ns, 1, n, ns);
C = A * C;
C = C + A * C;
C = C >= k;
Multiple-source breadth-first search
Multiple-source breadth-first search

- Sparse array representation => space efficient
- Sparse matrix-matrix multiplication => work efficient
- Three possible levels of parallelism: searches, vertices, edges
The final HPCS graph analysis benchmark (SSCA2) was betweenness centrality, not clustering -- but the main primitive was still multiple-source breadth-first search!
In the year 2010

(and soon after) …
Matrix-based graph processor design at MIT-LL

[Song, Kepner, et al. 2010]

3-D Graph Processor


Lincoln Laboratory, Massachusetts Institute Technology, Lexington, MA 02420

**Figure 1:** Computational Throughput Differences between Conventional and Graph Processing.

**Figure 2:** Sparse Matrix Representation of Graph.

**Figure 3:** 3-D Graph Processor with Electromagnetic Coupling Communications between Processor Boards.
An extensible distributed-memory library offering a small but powerful set of linear algebraic operations specifically targeting graph analytics.

- Aimed at graph algorithm designers/programmers who are not expert in mapping algorithms to parallel hardware.
- Flexible templated C++ interface.
- Scalable performance from laptop to 100,000-processor HPC.
- Open source software.
- Version 1.6.2 released April 2018.

Combinatorial BLAS [2010]

gauss.cs.ucsb.edu/~aydin/CombBLAS

[Azad, Buluc, G, Lugowski, ...]
Sparse matrix-sparse matrix multiplication

Element-wise operations

Sparse matrix-sparse vector multiplication

Sparse matrix indexing

Matrices over various semirings: (+, ·), (and, or), (min, +), ...
Examples of semirings in graph algorithms

<table>
<thead>
<tr>
<th>“values”: edge/vertex attributes, “add”: vertex data aggregation, “multiply”: edge data processing</th>
<th>General schema for user-specified computation at vertices and edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real field: ((\mathbb{R}, +, \times))</td>
<td>Numerical linear algebra</td>
</tr>
<tr>
<td>Boolean algebra: (({0, 1},</td>
<td>, &amp;))</td>
</tr>
<tr>
<td>Tropical semiring: ((\mathbb{R} \cup {\infty}, \min, +))</td>
<td>Shortest paths</td>
</tr>
<tr>
<td>((S, \text{select, select}))</td>
<td>Select subgraph, or contract nodes to form quotient graph</td>
</tr>
</tbody>
</table>
Graph algorithms in the language of linear algebra

- Kepner et al. study [2006]: fundamental graph algorithms including min spanning tree, shortest paths, independent set, max flow, clustering, ...
- SSCA#2 / centrality [2008]
- Basic breadth-first search / Graph500 [2010]
- Combinatorial BLAS [2010]
History: D4M and Graphulo

[D4M and Graphulo [Kepner et al., MIT & UW 2011 - 2015]

D4M: “Databases For Matlab”

- D4M binds Associative Arrays to Triple Store, enabling rapid prototyping of data-intensive cloud analytics and visualization

**Triple Store**
- Distributed Database

**D4M**
- Dynamic
- Distributed
- Dimensional
- Data
- Model

**Associative Arrays**
- Numerical Computing Environment

A D4M query returns a sparse matrix or graph from Accumulo ...

...for statistical signal processing or graph analysis in MATLAB

- Linear algebra on associative arrays for heterogeneous distributed databases & graphs

Computer Networks

**Network Events Table:**

<table>
<thead>
<tr>
<th>Row</th>
<th>Key (time)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2001-10-01 01 01 00</td>
</tr>
<tr>
<td>2</td>
<td>2001-10-01 01 01 00 00</td>
</tr>
<tr>
<td>3</td>
<td>2001-10-01 01 01 01 00</td>
</tr>
<tr>
<td>4</td>
<td>2001-10-01 01 01 01 04 00</td>
</tr>
<tr>
<td>5</td>
<td>2001-10-01 01 01 05 00</td>
</tr>
<tr>
<td>6</td>
<td>2001-10-01 01 01 06 00</td>
</tr>
</tbody>
</table>

- Define ranges of rows and columns
- $\mathcal{T} = \{2001-10-01 01 01 00, 2001-10-01 01 01 04 00\}$
- Query table and find popular pairs
- $A = \mathcal{T}(x,0)$
- $A^{' \times}A > 200$

**How to do matrix math in Accumulo?**
- The TwoTable iterator pipeline
- Jaccard & k-Truss

> **When to do matrix math in Accumulo?**
- Memory requirements
- Compare in-database I/O vs. alternatives

> **Future Work:** Multi-Node, expand to Relational Algebra, use an Optimizer to choose the best plan

Contact: Dylan Hutchison
dhutchis@cs.washington.edu

http://graphulo.mit.edu
• Clustering coefficient (triangle counting)
• Connected components (bully algorithm)
• Maximum independent set (NP-hard)
• Maximal independent set (Luby algorithm)
• Single-source shortest paths
• Special betweenness (for subgraph isomorphism)
Counting triangles (clustering coefficient)

**Clustering coefficient:**
- Pr (wedge i-j-k makes a triangle with edge i-k)
- \( 3 \times \frac{\# \text{ triangles}}{\# \text{ wedges}} \)
- \( 3 \times \frac{4}{19} = 0.63 \) in example
- may want to compute for each vertex j
Counting triangles (clustering coefficient)

Clustering coefficient:

- $\operatorname{Pr}(\text{wedge } i-j-k \text{ makes a triangle with edge } i-k)$
- $3 \times \frac{\# \text{ triangles}}{\# \text{ wedges}}$
- $3 \times \frac{4}{19} = 0.63$ in example
- may want to compute for each vertex $j$

“Cohen’s” algorithm to count triangles:

- Count triangles by lowest-degree vertex.
- Enumerate “low-hinged” wedges.
- Keep wedges that close.
Counting triangles (clustering coefficient)

\[ A = L + U \] (hi→lo + lo→hi)
\[ L \times U = B \] (wedge, low hinge)
\[ A \wedge B = C \] (closed wedge)

\[ \text{sum}(C)/2 = 4 \text{ triangles} \]
A = L + U  \ (\text{hi}\rightarrow\text{lo} + \text{lo} \rightarrow \text{hi})

L \times U = B \ (\text{wedge, low hinge})

A \wedge B = C \ (\text{closed wedge})

\text{sum}(C)/2 = \text{4 triangles}

\text{Spoiler: } (L \times L) \wedge L \text{ works better in practice} \ [\text{Wolf et al. 2017}]
History: The Graph BLAS Forum

http://graphblas.org

Standards for Graph Algorithm Primitives

Tim Mattson (Intel Corporation), David Bader (Georgia Institute of Technology), Jon Berry (Sandia National Laboratory), Aydın Buluç (Lawrence Berkeley National Laboratory), Jack Dongarra (University of Tennessee), Christos Faloutsos (Carnegie Mellon University), John Feo (Pacific Northwest National Laboratory), John Gilbert (University of California at Santa Barbara), Joseph Gonzalez (University of California at Berkeley), Bruce Hendrickson (Sandia National Laboratory), Jeremy Kepner (Massachusetts Institute of Technology), Charles Leiserson (Massachusetts Institute of Technology), Andrew Lumsdaine (Indiana University), David Padua (University of Illinois at Urbana-Champaign), Stephen Poole (Oak Ridge National Laboratory), Steve Reinhardt (Cray Corporation), Mike Stonebraker (Massachusetts Institute of Technology), Steve Wallach (Convey Corporation), Andrew Yoo (Lawrence Livermore National Laboratory)

Abstract-- It is our view that the state of the art in constructing a large collection of graph algorithms in terms of linear algebraic operations is mature enough to support the emergence of a standard set of primitive building blocks. This paper is a position paper defining the problem and announcing our intention to launch an open effort to define this standard.

Mathematical Foundations of the GraphBLAS

Jeremy Kepner (MIT Lincoln Laboratory Supercomputing Center), Peter Aaltonen (Indiana University), David Bader (Georgia Institute of Technology), Aydın Buluç (Lawrence Berkeley National Laboratory), Franz Franchetti (Carnegie Mellon University), John Gilbert (University of California, Santa Barbara), Dylan Hutchison (University of Washington), Manoj Kumar (IBM), Andrew Lumsdaine (Indiana University), Henning Meyerhenke (Karlsruhe Institute of Technology), Scott McMillan (CMU Software Engineering Institute), Jose Moreira (IBM), John D. Owens (University of California, Davis), Carl Yang (University of California, Davis), Marcin Zalewski (Indiana University), Timothy Mattson (Intel)
In the year 2018 ....
Fast Linear Algebra-Based Triangle Counting with KokkosKernels

Michael M. Wolf, Mehmet Deveci, Jonathan W. Berry, Simon D. Hammond, Sivasankaran Rajamanickam
Center for Computing Research, Sandia National Laboratories
Albuquerque, NM 87185
{mwolf, mehmet, jerry, ohannes, sandia}@sandia.gov

About—Triangle counting serves as a key building block for a set of important graph algorithms in network science. In this paper, we address the IEEE HPDC-StatGraph Challenge problem of triangle counting, focusing on obtaining the best parallel performance on a single multicore node. Our implementation uses a linear algebra-based approach to triangle counting that has grown out of work related to our minTIG data analytics mini-applications [1] and our efforts to pose graph algorithms in the language of linear algebra. We leverage KokkosKernels to implement this approach efficiently on multicore architectures. Our performance results are competitive with the fastest known graph traversal-based approaches and are significantly faster than the Graph Challenge reference implementations, up to 5x faster for large, dense graphs of similar size. Our performance results are also significantly faster than the Python reference on a single core of a Haswell node.

B. Linear Algebra Primitives for Graph Algorithms
The GraphBLAS [13, 14] community has been working to standardize a set of building blocks to solve graph problems in the language of sparse linear algebra. Many graph computations can be efficiently written in terms of linear algebra [17], including breadth-first search, betweenness centrality, and triangle counting/minimization [18, 19] (discussed further to pose graph algorithms in the language of linear algebra. We focus on triangle counting on a single compute node, leveraging KokkosKernels [14] to implement this approach efficiently). We obtain results that are competitive with the fastest known graph traversal-based approaches.

User Guide for SuiteSparse:GraphBLAS
Timothy A. Davis
davis@tamu.edu, Texas A&M University.
http://www.suitesparse.com and http://aldemnth.com
VERSION 2.0.1, Mar 15, 2018

Abstract
SuiteSparse:GraphBLAS is a full implementation of the GraphBLAS standard, which defines a set of sparse matrix operations on an extended algebra of semirings using an almost unlimited variety of operators and types. When applied to sparse adjacency matrices, these algebraic operations are equivalent to computations on graphs. GraphBLAS provides a powerful and expressive framework for creating graph algorithms based on the elegant mathematics of sparse matrix operations on a semiring.

Graph Challenge Champions
2017 Analysis of all Triangle Counting Submissions
Champions
• Fast Linear Algebra-Based Triangle Counting with Ko Hammond, Sivasankaran Rajamanickam (Sandia)

Motivation Challenges Data Sets Scenarios Submit

Home

The Present

The GraphBLAS C API Specification:\nVersion 1.2.0

Aydin Buluç, Timothy Mattson, Scott McMillan, José Moreira, Carl Yang

Operation Name Mathematical Notation
mxm C(M, z) C ⊗ A ⊗ B
mvx w(m, z) w ⊗ A ⊗ u
vxm wT(mT, z) wT ⊗ uT ⊗ A
ewiseMult C(M, z) C ⊗ A ⊗ B
ewiseAdd C(M, z) C ⊗ A ⊗ B
reduce (row) w(m, z) w ⊗ [bij A(i, j)]
reduce (scalar) s s ⊗ [bij A(i, j)]
apply C(M, z) C ⊗ f(u)
transpose C(M, z) C ⊗ AT
extract C(M, z) C ⊗ A(i, j)
assign w(m, z) w ⊗ u(i)

Spectral Graph Drawing: Building Blocks and Performance Analysis
Shad Kirmani and Kamesh Madduri

Parallel generation of large-scale random graphs
Anil Vullikanti

Design, Generation, and Validation of Extreme Scale Power-Law Graphs
Jeremy Keppeler and Sid Samii

On Large-Scale Graph Generation with Validation of Diverse Triangle Statistics at Edges and Vertices
Geoffrey Sanders, Roger Pearce, Timothy La Fond and Jeremy Keppeler

Enabling Massive Deep Neural Networks with the GraphBLAS
Jeremy Keppeler¹, Manoj Kumar², José Moreira¹, Pratap Pattnak¹, Mauricio Serrano¹, Henry Tufo²

Input Features

Y1 Y2 Y3

Hidden Layers

Y0

Output Features

Y0 Y1 Y2 Y3

Graph Building Blocks in Graph Challenge

Graph Challenge Champions

GABB 2018 Talks

(8:30am-10am) Spectral Graph Drawing: Building Blocks and Performance Analysis
Shad Kirmani and Kamesh Madduri

Morning Break (10-10:30am)

1 (10:30am-12) Parallel generation of large-scale random graphs
Anil Vullikanti

Generating Graphs with known properties

Semantic Graphs of Vectors, Graphs, and Matrices

Oracle: How to Pose Graph Algorithms in the Language of Linear Algebra

GABB 2018 Talks

Lunch (12-1:30pm)

2 (1:30pm-3pm) Patterns of GraphBLAS Algorithms: Tales from the Trenches
Scott McMillian

Implementing the GraphBLAS C API
Jose Moreira, Manoj Kumar and William Horn

PyGB: GraphBLAS DSL in Python with Dynamic Compilation into Efficient C++
Jesse Chamberlin, Marcin Zalewski, Scott McMillan and Andre Lumsdaine

3 (3:30pm-5pm) A Survey of Modern Analysis on Graphs
Chris Long

Panel: Graph Building Blocks in Graph Challenge
Panelists: Jose Moreira, Chris Long and Marcin Zalewski.

(user guide)
In the years 2019 —
What do we hope for in the future?

- More basic capabilities
  - Streaming and dynamic-graph algorithms
  - “Priority queue” algorithms: strong components, top k vertices, etc.
  - Not materializing intermediate results (eg, incidence matrix methods)
  - Laplacian paradigm for graph algorithms
In sparse Gaussian elimination, for nonsymmetric $A$, one can find . . .

- column nested dissection or min degree permutation
- column elimination tree $T(A^TA)$
- row and column counts for $G^+(A^TA)$
- supernodes of $G^+(A^TA)$
- nonzero structure of $G^+(A^TA)$

. . . efficiently, without ever forming $A^TA$ explicitly.

- How generally can we do graph algorithms in linear algebra without storing intermediate results?
- Can we do fine-grained scheduling of vertex and edge operations to break out of bulk synchronous execution?
- Can we reason directly about products of sparse matrices?
Storing $A$, operating implicitly on $A^T A$

- CombBLAS represents graphs as adjacency matrices.
- D4M represents graphs as incidence matrices; matrix $A$ represents $G(A^T A)$:

  - **Column** = vertex
  - **Row** = hyperedge
Storing A, operating implicitly on $A^TA$

- Many other cases:
  - Optimization: KKT systems, interior point methods.
  - Automatic differentiation: distance-2 coloring.
  - Linear equations: QR factorization, structure prediction for LU factorization with partial pivoting.

- **Question**: What can you do fast on $G(A^TA)$ just from $G(A)$?
Statistics for $A^T A$ itself are harder!

- $\text{nnz}(A^T A)$ seems to be as hard as computing $A^T A$.
  - but randomized estimate is possible [Cohen 1998]

- Sampling algorithms are possible too, e.g. diamond sampling for $k$ largest elements of $A^T A$ (or $B^*C$ in general) [Ballard/Kolda/Pinar/Seshadri 2015]

\[ \text{Ballard et al. ICDM 2015} \]
What do we hope for in the future?

- More basic capabilities
  - Streaming and dynamic algorithms
  - “Priority queue” algorithms: strong components, top k vertices, etc.
  - Not materializing intermediate results (e.g., incidence matrix methods)
  - Laplacian paradigm for graph algorithms
Laplacian matrix of a graph

- **Graph Laplacian:** Symmetric, positive semidefinite, weighted.
- **Laplacian paradigm:** Use $Ax = b$ as a subroutine in graph algorithms
  
  \[ \text{[Kelner, Teng, many others]} \]

- **Laplacian eigenvectors** for partitioning, embedding, and clustering
  
  \[ \text{[Fiedler, Pothen/Simon, Spielman/Teng, many others]} \]

- Interesting new ideas coming from theoretical computer science.
What do we hope for in the future?

• More basic capabilities
  – Streaming and dynamic algorithms
  – “Priority queue” algorithms: strong components, top k vertices, etc.
  – Not materializing intermediate results (eg, incidence matrix methods)
  – Laplacian paradigm for graph algorithms
What do we hope for in the future?

- **More basic capabilities**
  - Streaming and dynamic algorithms
  - “Priority queue” algorithms: strong components, top k vertices, etc.
  - Not materializing intermediate results (e.g., incidence matrix methods)
  - Laplacian paradigm for graph algorithms

- **More directions**
  - Integration with numerical matrix libraries
  - Statistical perspective: random objects, stochastic graphs, etc.
  - Deep neural networks (more)
  - Signal processing on graphs
What do we hope for in the future?

- More basic capabilities
  - Streaming and dynamic algorithms
  - “Priority queue” algorithms: strong components, top k vertices, etc.
  - Not materializing intermediate results (e.g., incidence matrix methods)
  - Laplacian paradigm for graph algorithms
- More directions
  - Integration with numerical matrix libraries
  - Statistical perspective: random objects, stochastic graphs, etc.
  - Deep neural networks (more)
  - Signal processing on graphs
- More uptake
  - By hardware vendors
  - By software vendors
Summary: Past 60 Years

As the “middleware” of scientific computing, linear algebra has given us:

- Mathematical tools
- High-level primitives
- High-quality software libraries
- High-performance kernels for computer architectures
- Interactive environments
Today

Continuous Physical Modeling

Linear Algebra

Computers

Discrete Structure Analysis

Graph Theory

Computers
Today

Continuous Physical Modeling

Discrete Structure Analysis

Linear Algebra & Graph Theory

Computers

Computers
Tomorrow

- Continuous Physical Modeling
  - Linear Algebra
    - Computers

- Discrete Structure Analysis
  - Graph Theory
    - Computers

- Extracting Sense from Data
  - Computers

UCSB
Tomorrow

- Continuous Physical Modeling
  - Linear Algebra
    - Computers

- Discrete Structure Analysis
  - Graph Theory
    - Computers

- Extracting Sense from Data
  - Statistics ?
    - Computers
Tomorrow

- Continuous Physical Modeling
  - Linear Algebra
  - Computers

- Discrete Structure Analysis
  - Graph Theory
  - Computers

- Extracting Sense from Data
  - Deep Learning?
  - Computers
Continuous Physical Modeling
  ↓
  Linear Algebra
  ↓
  Computers

Discrete Structure Analysis
  ↓
  Graph Theory
  ↓
  Computers

Extracting Sense from Data
  ↓
  Neuromorphics?
  ↓
  Computers
Tomorrow

Continuous Physical Modeling

Discrete Structure Analysis

Extracting Sense from Data

Linear Algebra

Graph Theory

Computers

Computers

Computers
Continuous Physical Modeling

Discrete Structure Analysis

Extracting Sense from Data

Linear Algebra & Graph Theory & ???

Computers

Computers

Computers
Thanks …