Graph Exploration: to Linear Algebra (and Beyond?)

Jonathan Berry
Michael Wolf
Dylan Stark

Sandia National Laboratories

GABB Workshop
May 19, 2014

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Talk Objectives

- Motivate the selection of graph primitives via social network analysis
- Argue that there are different levels of graph primitives
 (a better title aside: "To linear algebra and *below*")
- Describe a sincere attempt to implement an analysis using linear algebra from John Gilbert via CombinatorialBLAS
- Discuss lessons learned and implications for GABB design

Example Public Graphs in "Big Data" Analysis

- Twitter-2010 (as obtainable from LAW (http://law.di.unimi.it/))
  - Edges: "A is followed by B"
  - 41M vertices, 1.4B edges
  - Contains *vast* amounts of spam-like behavior (e.g., you can buy 500 "followers" for $5)

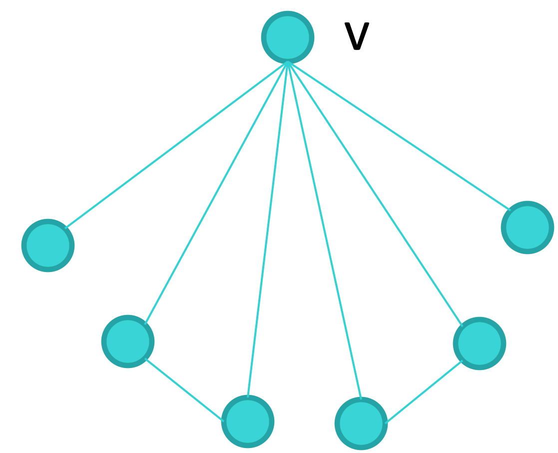
- LiveJournal-2008 (as obtainable from Stanford Network Analysis Project (SNAP: http://snap.stanford.edu))
  - Edges: "A declares friendship with B"
  - 3.9M vertices, 34M edges
  - Also contains strange, non-human-looking behavior (as we'll see)

In addition to social networks like these, there are web graphs, peer-to-peer, etc

Sandia National Laboratories

Per-Degree Clustering Coefficients

- Metrics for characterizing the graph?
- Vertex v has:
  - Degree 6
  - (6 choose 2)=15 wedges (pairs of neighbors)
  - 2 triangles
  - Clustering coefficient 2/15
- Consider the average clustering coefficient of all vertices of degree *d* (for all *d*)
  - Call this the "per-degree clustering coef" (pdcc)
- We'll look at plots of degree vs. pdcc

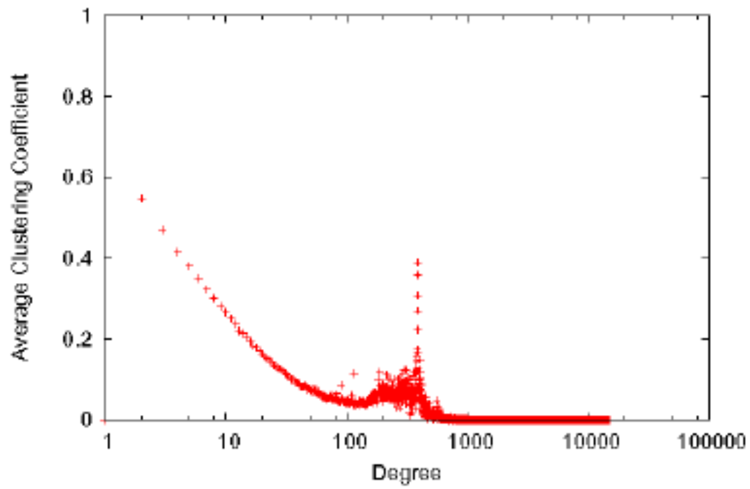


Berry, Jonathan, Aaron Kearns, Cynthia A. Phillips, Jared Saia, "Finding a planted clique in a distributed social network," SIAM Workshop on Network Science, July 2014.

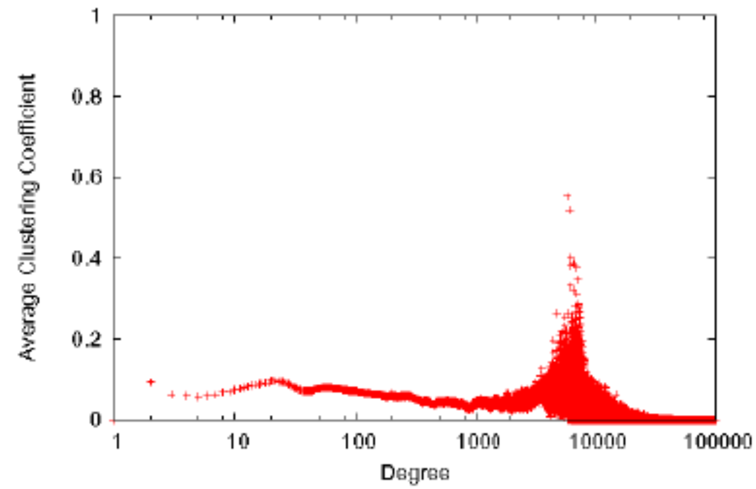
Sandia National Laboratories

PDCC in LiveJournal and Twitter

LiveJournal

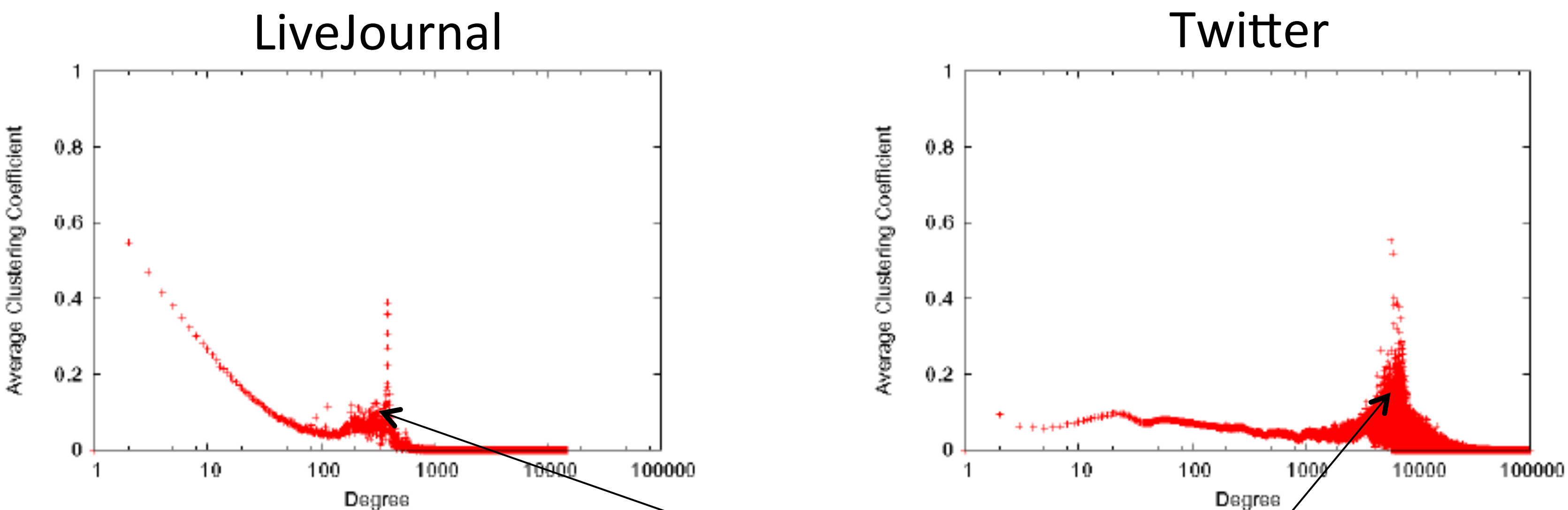


Twitter



Berry, Jonathan, Aaron Kearns, Cynthia A. Phillips, Jared Saia, "Finding a planted clique in a distributed social network," SIAM Workshop on Network Science, July 2014.

PDCC in LiveJournal and Twitter



Example analysis goal: understand these anomalies which are not predicted by social science

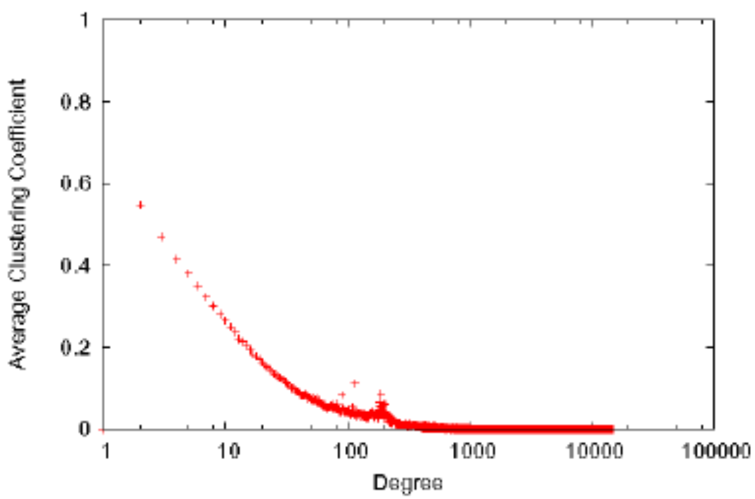
Methodology to Understand Anomalies

- *Enumerate* triangle (3-cycles) in the graphs
  - We don't just want to count triangles
  - We'll examine each triangle and operate on it individually
  - Example:
    - Pass 1: For each triangle T:
      - Increment the triangle degree of each endpoint
    - Pass 2: For each triangle T:
      - If each endpoint has property X: (e.g. triangle degree >= 100)
        - » Call T an "X-triangle"
        - » Increment the "X-triangle degree" of each endpoint
    - Pass 3: etc. – many variations

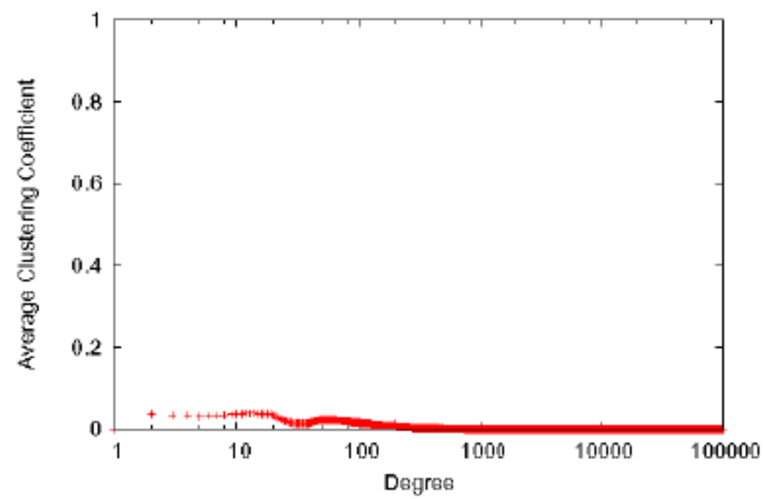
Anomalies Removed

Using a process like this, we can identify and remove such anomalies (details beyond the scope of this talk)

LiveJournal



Twitter



Hence, we believe that enumerating (not just counting) small structures is an important GABB

Berry, Jonathan, Aaron Kearns, Cynthia A. Phillips, Jared Saia, "Finding a planted clique in a distributed social network," SIAM Workshop on Network Science, July 2014.

Sandia National Laboratories

GABB at Different Levels of Granularity

- Low-Level graph (e.g. the PBGL, MTGL)
  - Iteration, vertex & edge properties
  - Maximum flexibility
  - Relatively little help with algorithms

Sizes	
size_type	The type that represents sizes associated with the graph such as the number of edges in the graph or the degree of a vertex.
Descriptors	
vertex_descriptor	The handle used to refer to a vertex.
edge_descriptor	The handle used to refer to a edge.
Vector Iterators	
vertex_iterator	Iterator type that iterates over all the vertices in a graph.
edge_iterator	Iterator type that iterates over all the edges in a graph.
adjacency_iterator	Iterator type that iterates over the adjacent vertices of a given vertex.
in_adjacency_iterator	Iterator type that iterates over the in adjacent vertices of a given vertex.
out_edge_iterator	Iterator type that iterates over the outgoing adjacent edges of a given vertex.
in_edge_iterator	Iterator type that iterates over the incoming adjacent edges of a given vertex.
Thread Iterators	
thread_vertex_iterator	Iterator type that iterates over all the vertices in a graph.
thread_edge_iterator	Iterator type that iterates over all the edges in a graph.
thread_adjacency_iterator	Iterator type that iterates over the adjacent vertices of a given vertex.
thread_in_adjacency_iterator	Iterator type that iterates over the in adjacent vertices of a given vertex.
thread_out_edge_iterator	Iterator type that iterates over the outgoing adjacent edges of a given vertex.
thread_in_edge_iterator	Iterator type that iterates over the incoming adjacent edges of a given vertex.
Categories	
directed_category	Indicates if the graph is undirected, directed, or bidirectional.
iterator_category	Indicates if the graph supports vector iterators, thread iterators, or both.

Edge Functions

The following functions must be defined for all graphs.

vertex_descriptor
source(const edge_descriptor& e, const Graph& g)
Returns the source vertex of e.

vertex_descriptor
target(const edge_descriptor& e, const Graph& g)
Returns the target vertex of e.

Vertex Functions

The following functions must be defined for all graphs.

size_type
out_degree(vertex_descriptor v, const Graph& g)
Returns the number of edges leaving v.

In addition, the following two functions must be defined for a bidirectional graph.

size_type
in_degree(vertex_descriptor v, const Graph& g)
Returns the number of edges entering v.

size_type
degree(vertex_descriptor v, const Graph& g)
Returns the sum of the number of edges leaving v and the number of edges entering v.

Filtering Functions

The following functions must be defined for all graphs. They are used when implementing filtering

vertex_descriptor
null_vertex(const Graph& g)
Returns a special value for a vertex_descriptor that represents an invalid vertex.

edge_descriptor
null_edge(const Graph& g)
Returns a special value for an edge_descriptor that represents an invalid edge.

bool
is_valid(iterator_type& iter, size_type p, const Graph& g)
Returns if the vertex or edge descriptor pointed to by iter[p] is valid.

- High-Level graph (e.g. Combinatorial BLAS)
  - Pre-selected solution method (linear algebra)
  - Customize that pre-selected method

Some GABB for Triangle Enumeration

- MTGL triangle enumeration
  - Bucket data structures
- GraphLab triangle numeration
  - Hashtable operations (variants on "Cuckoo hashing")
  - Set intersection on these objects and arrays
- Combinatorial BLAS triangle enumeration
  - Linear algebra semi-ring operations
  - (the rest of this talk!)

And this just scratches the surface for graph analysis; there are all sorts of approaches for GABB to support such as linear & integer programming, dynamic programming, interaction with machine learning, etc.

J. Berry comment: basic linear algebra support is great, but beware of over-generalizations

Sandia National Laboratories

Gilbert's Algorithm for Triangle Counting (Part 1)

Consider the graph in Figure 1. Its adjacency matrix, with rows and columns sorted by vertex degree is:

$$A = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

Gilbert's algorithm has us split A into lower and upper triangular portions:

$$L = \begin{pmatrix} 0 & & & & \\ 0 & 0 & & & \\ 0 & 0 & 0 & & \\ 0 & 1 & 1 & 0 & \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

A = L + U

$$U = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ & 0 & 0 & 1 & 1 \\ & & 0 & 1 & 1 \\ & & & 0 & 1 \\ & & & & 0 \end{pmatrix}$$

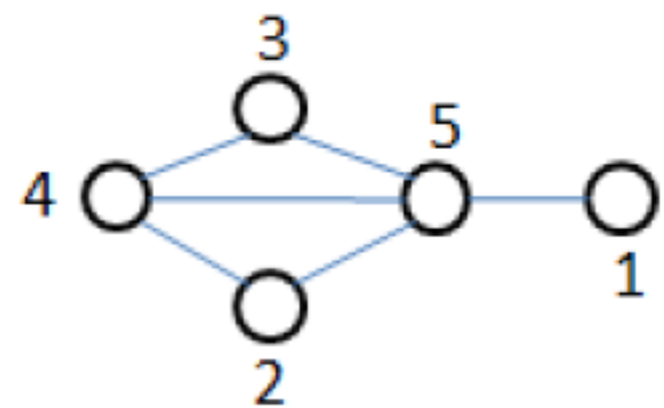
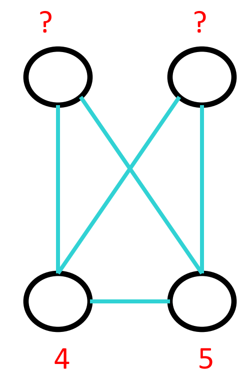


Figure 1

Gilbert's Algorithm for Triangle Counting (Part 2)

The unmodified Gilbert's algorithm then computes $B = L * U$:

$$B = \begin{pmatrix} 0 & & & & \\ 0 & 0 & & & \\ 0 & 0 & 0 & & \\ 0 & 1 & 1 & 0 & \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix} * \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ & 0 & 0 & 1 & 1 \\ & & 0 & 1 & 1 \\ & & & 0 & 1 \\ & & & & 0 \end{pmatrix} = \begin{array}{c} \\ \\ \\ \\ \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} & & & 4 & \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 2 \\ 0 & 0 & 0 & 2 & 4 \end{pmatrix} \end{array}$$

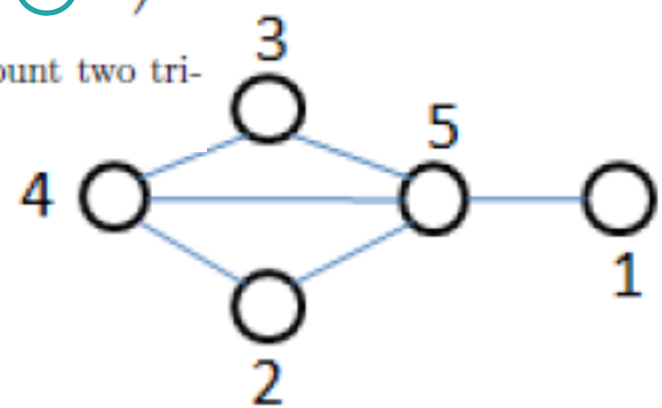


Indicates two unknown wedges on edge (5,4)

Finally, the unmodified algorithm computes $A. * B$:

$$C = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix} .* \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 2 \\ 0 & 0 & 0 & 2 & 4 \end{pmatrix} = \begin{array}{c} \\ \\ \\ \\ \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} & & & 4 & \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 2 & 0 \end{pmatrix} \end{array}$$

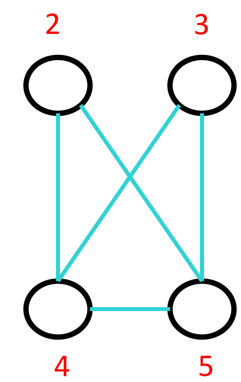
Summing the elements in the lower triangular portion of C, we count two triangles.



CombBLAS supports this algorithm well

Enumeration Version of Gilbert's Algorithm

$$B = L \text{ “enum” } U = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \{2,3\} & \{2,3\} \\ 0 & 0 & 0 & \{2,3\} & \{1,2,3,4\} \end{pmatrix}$$



Ex: builds a list of all wedges on edge (5,4); then A .* B filters to get triangles

Next, we'll consider the GABB implications of this algorithm

GABB for Gilbert Triangle Enumeration 1

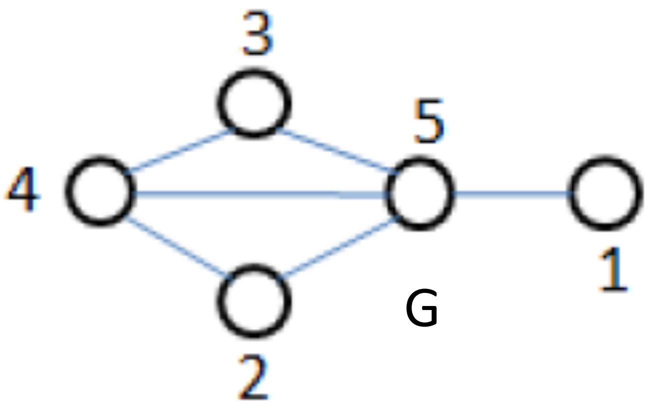
- Problem: We cannot form matrix B; there are too many wedges

- Solution 1: There is no need to compute B; this can be expressed as *fused* operation
  - Use information in A[i,j] to determine whether to calculate B[i,j]

- Solution 2: There is no need to form B; this can be expressed as a *streaming* algorithm
  - As soon as B[i,j] is generated, can (must) immediately begin B[i,j] .* A[i,j], then delete B[i,j]

Algorithm 2 for Triangle Enumeration – UCSB (Part 1)

Consider graph G:



- Each row of L implicitly holds wedges for a given vertex
- Wedges = pairwise combinations of column ids, and row number
- E.g, row 5
  - Wedges: {1,2,5}, {1,3,5}, {1,4,5}, {2,3,5}, {2,4,5}, {3,4,5}

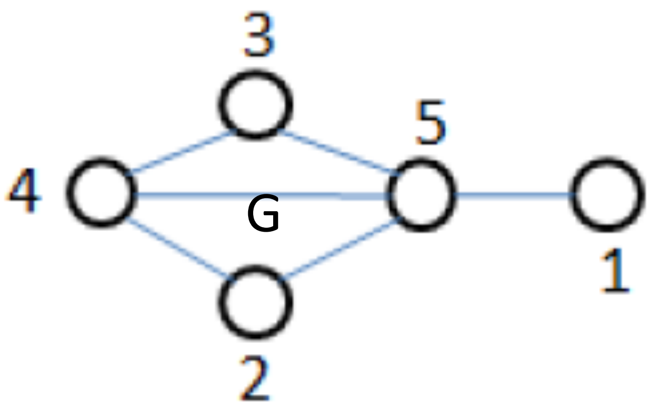
$$A = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

Adjacency matrix of G

$$L = \begin{pmatrix} 0 & & & & \\ 0 & 0 & & & \\ 0 & 0 & 0 & & \\ 0 & 1 & 1 & 0 & \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

Lower triangular portion of A

Algorithm 2 for Triangle Enumeration – UCSB (Part 2)



$$B = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

Incidence matrix of G

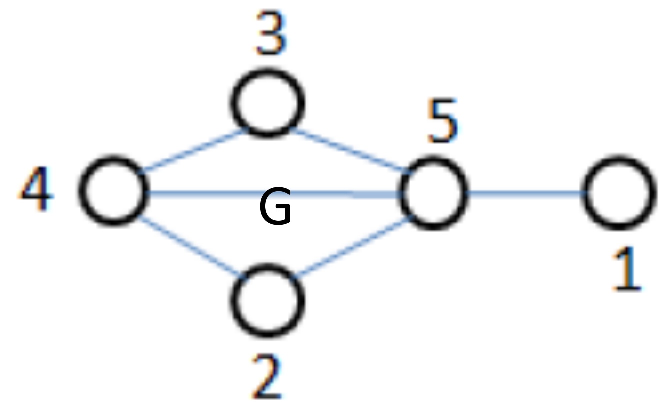
$C = L * B$

$$C = \begin{pmatrix} 0 & & & & \\ 0 & 0 & & & \\ 0 & 0 & 0 & & \\ 0 & 1 & 1 & 0 & \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix} * \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 2 & 2 & 0 \end{pmatrix}$$

- Matrix formed by multiplying L by incidence matrix indicates triangles
- Elements in C of size 2 correspond to triangles

Algorithm 2 for Triangle Enumeration – UCSB (Part 3)

$$B = \begin{array}{c} \\ \\ 3 \\ 4 \\ \\ \end{array} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} \begin{array}{c} \\ 2 \\ \\ 4 \\ \\ \end{array}$$



$$C = \begin{array}{c} \\ \\ \\ \\ 5 \end{array} \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 2 & 2 & 0 \end{pmatrix} \begin{array}{c} \\ \\ \\ \\ 5 \end{array}$$

Triangles:

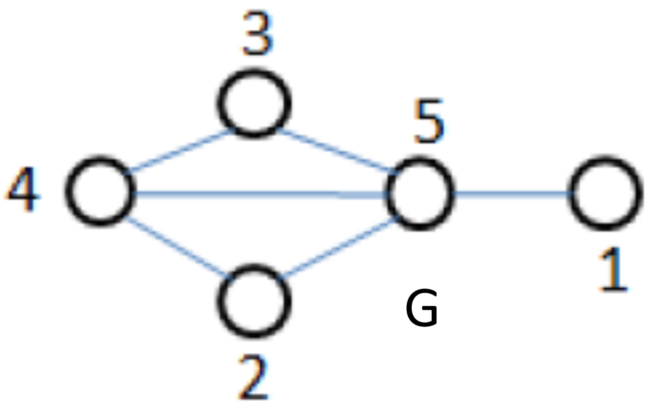
{2, 4, 5}

{3, 4, 5}

- C and incidence matrix can be used to enumerate triangles
- C[i,j]==2 indicates triangle
  - Vertex i is in triangle
  - Two vertices k, where B[k,j]==1 are in triangle

Algorithm 2 with Modification (Part 1)

Consider graph G:



- Each row of L implicitly holds wedges for a given vertex
- Wedges = pairwise combinations of column ids + row number
- E.g, row 5
  - Wedges: {1,5,2}, {1,5,3}, {1,5,4}, {2,5,3}, {2,5,4}, {3,5,4}

$$A = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

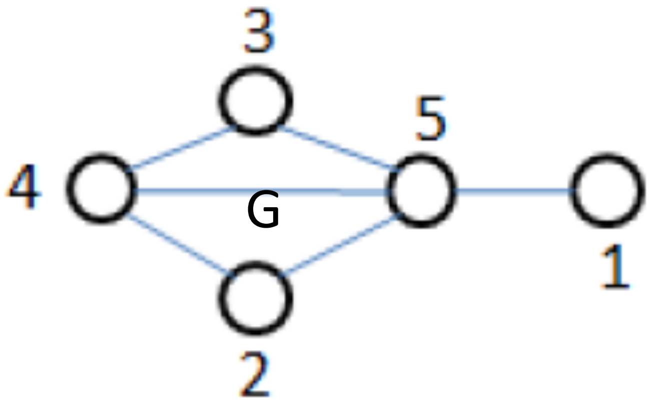
Adjacency matrix of G

$$L = \begin{pmatrix} \emptyset & & & & \\ \emptyset & \emptyset & & & \\ \emptyset & \emptyset & \emptyset & & \\ \emptyset & \{1\} & \{1\} & \emptyset & \\ \{1\} & \{1\} & \{1\} & \{1\} & \emptyset \end{pmatrix}$$

Lower triangular portion of A

Sandia National Laboratories

Algorithm 2 with Modification (Part 2)



$$B = \begin{pmatrix} \{1\} & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \{1\} & \emptyset & \emptyset & \{1\} & \emptyset \\ \emptyset & \emptyset & \{1\} & \{1\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \{1\} & \{1\} & \{1\} \\ \{1\} & \{1\} & \{1\} & \emptyset & \emptyset & \{1\} \end{pmatrix}$$

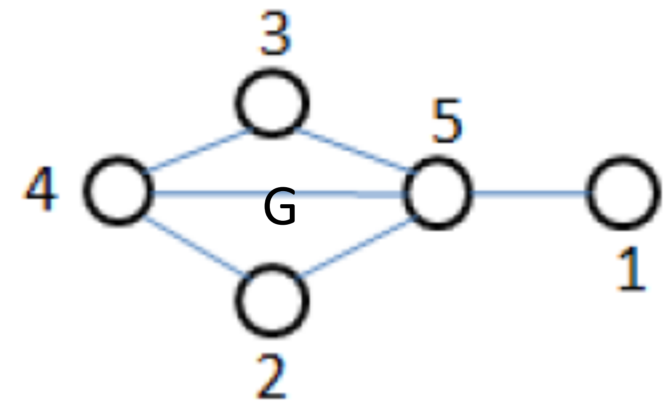
Incidence matrix of G

$$C = L * B$$

$$C = \begin{pmatrix} \emptyset & & & & \\ \emptyset & \emptyset & & & \\ \emptyset & \emptyset & \emptyset & & \\ \emptyset & \{1\} & \{1\} & \emptyset & \\ \{1\} & \{1\} & \{1\} & \{1\} & \emptyset \end{pmatrix} * \begin{pmatrix} \{1\} & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \{1\} & \emptyset & \emptyset & \{1\} & \emptyset \\ \emptyset & \emptyset & \{1\} & \{1\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \{1\} & \{1\} & \{1\} \\ \{1\} & \{1\} & \{1\} & \emptyset & \emptyset & \{1\} \end{pmatrix} = \begin{pmatrix} \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \{2\} & \{3\} & \{3\} & \{2\} & \emptyset \\ \{1\} & \{2\} & \{3\} & \{3,4\} & \{2,4\} & \emptyset \end{pmatrix}$$

- Multiplication by incidence matrix yields triangles
- Elements C[i,j] are formed by appending all k where L[i,k] and B[k,j] are nonzero
- Elements of size 2 correspond to triangles

Algorithm 2 with Modification – (Part 3)



$$D = f(C)$$

$$D = f\begin{pmatrix} \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \{2\} & \{3\} & \{3\} & \{2\} & \emptyset \\ \{1\} & \{2\} & \{3\} & \{3,4\} & \{2,4\} & \emptyset \end{pmatrix} = \begin{pmatrix} \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \{3,4,5\} & \{2,4,5\} & \emptyset \end{pmatrix}$$

Triangles

- Final step
- Remove all elements of size 1
- Append row number to remaining entries
- Triples represent triangles

GABB for Triangle Enumeration Method 2 (Modified)

- Problem: We don’t want to form matrix C explicitly; there are too many nonzeros that will be pruned in next step

- Solution 1: There is no need to store non-triangle indicators; this can be expressed as *fused* operation
  - Fuse function f() with matrix multiply

- Solution 2: There is no need to form C; this can be expressed as a *streaming* algorithm
  - As soon as C[i,j] is generated, we apply f to obtain D[i,j].
  - If C[i,j] is size 1, D[i,j] is a zero
  - If C[i,j] is size 2, D[I,j] contains a triangle

CombBLAS Implementations Status

- CombBLAS implementation of triangle counting working
  - CombBLAS to be powerful and flexible (especially user defined semirings)
  - CombBLAS straight-forward to use

- CombBLAS implementation of triangle enumeration more challenging than counting
  - Complex datatypes (e.g, STL vectors, lists) desirable for this kernel and other graph computation; not well supported in CombBLAS
  - Not sure how fusing and/or streaming fit into CombBLAS
  - Concerned that CombBLAS implementation will not be performance competitive with extensions
  - Future work: compare CombBLAS triangle enumeration implementation to other graph implementations

- CombBLAS-like linear algebra Matevo-style proxy "app" of triangle enumeration
  - Fused linear algebra operations considered crucial to performance
  - "Hand-coded" for our specific need but using a CombBLAS-like API
  - Takeaway: CombBLAS API sufficient to support triangle enumeration algorithm

Sandia National Laboratories