Introduction to GraphBLAS Algorithms

Gábor Szárnyas
szarnyas@mit.bme.hu
Graph algorithms in GraphBLAS

PageRank
We use one of the simpler definitions. For $k = 1$ to $t$ iterations:

$$
PR_0(v) = \frac{1}{n}
$$

$$
PR_k(v) = \frac{1 - \alpha}{n} + \alpha \cdot \sum_{u \in N_{\text{in}}(v)} \frac{PR_{k-1}(u)}{|N_{\text{out}}(u)|} + \alpha \cdot \sum_{w \in dng} PR_{k-1}(w)
$$

$\alpha$: damping factor

dng: dangling vertices, $dng = \{w \in V \mid |N_{\text{out}}(w)| = 0\}$

There are dozens of PR definitions, some treat dangling vertices differently.
PAGERANK – IN LINEAR ALGEBRA

Initially:
\[ \mathbf{pr}_0 = [1 \ 1 \ \ldots \ 1] \odot n, \quad \mathbf{outd} = [\bigoplus_j \mathbf{A}(:,j)] \]

In each iteration:
\[
\begin{align*}
\mathbf{PR}_k(v) &= \frac{1 - \alpha}{n} + \alpha \cdot \sum_{u \in N_{in}(v)} \frac{\mathbf{PR}_{k-1}(u)}{|N_{out}(u)|} + \frac{\alpha}{n} \cdot \sum_{w \in dng} \mathbf{PR}_{k-1}(w) \\
\mathbf{pr}_k &= \frac{1 - \alpha}{n} \bigoplus \alpha \otimes \left( \frac{\mathbf{pr}_{k-1}}{\mathbf{outd}} \right) \bigotimes \mathbf{A} \bigoplus \frac{\alpha}{n} \otimes \left[ \bigoplus_i (\mathbf{pr}_k \otimes \overline{\mathbf{outd}})_i \right]
\end{align*}
\]

constant \hspace{2cm} \text{SpMV} \hspace{2cm} \text{element-wise sparse vector-dense vector multiplication}
PAGERANK – ALGORITHM

Input: adjacency matrix $A$, damping factor $\alpha$, #iterations $t$, #vertices $n$

Output: PageRank $pr$ (real)

Workspace: outdegrees $outd$

1. $pr = [1\ 1\ \ldots\ 1] \otimes n$
2. $outd = [\oplus_j A(:, j)]$
3. for $k = 1$ to $t$

4. $pr = \frac{1 - \alpha}{n} \oplus \alpha \times \left(\frac{pr}{outd}\right) \otimes \otimes A \oplus \frac{\alpha}{n} \otimes \left[\oplus_i (pr \otimes outd)_i\right]$

PageRank variants using Markov models are more difficult to express.
Graph algorithms in GraphBLAS

Local clustering coefficient (transitivity)
**LCC: LOCAL CLUSTERING COEFFICIENT**

\[
LCC(v) = \frac{\#\text{edges between neighbours of } v}{\#\text{possible edges between neighbours of } v} = \frac{\text{tri}(v)}{\text{wed}(v)}
\]

If \(|N(v)| \leq 1\), \(LCC(v) = 0\)

Important in social network analysis (known as *transitivity*).

The numerator is the number of *triangles* in \(v\), \(\text{tri}(v)\).

The denominator is the number of *wedges* in \(v\), \(\text{wed}(v)\).

The difficult part is \(\text{tri}(v)\).
LCC: NUMBER OF WEDGES IN EACH VERTEX

\[ LCC(v) = \frac{\text{tri}(v)}{\text{wed}(v)} \]

- For \( \text{wed}(v) \), we determine the number of wedges for each vertex as the 2-permutation of its degree:
  \[ \text{perm2}(x) = x \cdot (x - 1) \]

- Given the degrees \( \text{deg} = [\bigoplus_j A(:, j)] \), we compute \( \text{wed} \) by applying a unary function on the elements of the vector:
  \[ \text{wed} = \text{perm2}(|\text{deg}|) \]
LCC EXAMPLE: NUMBER OF WEDGES

\[
\begin{align*}
A & : j, \\
\text{deg} & : 2, 4, 3, 5, 3, 3, 4 \\
\text{wed} & : 2, 12, 6, 6, 6, 12
\end{align*}
\]
LCC EXAMPLE: COMPLETE ALGORITHM

\[
\text{TRI}(A) = A \oplus \otimes A
\]

\[
deg \begin{bmatrix}
2 \\
4 \\
3 \\
5 \\
3 \\
3 \\
4
\end{bmatrix}
\]

\[
\text{perm}_2(\ldots)
\]

\[
\text{tri}_2 \begin{bmatrix}
2 \\
6 \\
4 \\
8 \\
2 \\
2 \\
6
\end{bmatrix}
\]

\[
\text{wed} \begin{bmatrix}
2 \\
12 \\
6 \\
20 \\
6 \\
6 \\
12
\end{bmatrix}
\]

\[
\text{lcc} \begin{bmatrix}
1.00 \\
0.50 \\
0.67 \\
0.40 \\
0.33 \\
0.33 \\
0.50
\end{bmatrix}
\]
LCC: ALGORITHM

Input: adjacency matrix $A$
Output: vector $lcc$
Workspace: matrix $TRI$, vectors $tri2$, $deg$, $wed$, and $lcc$

1. $TRI(A) = A \oplus A \otimes A$ compute triangle count matrix
2. $tri2 = \bigoplus_j TRI(:, j)$ reduce to triangle count vector
3. $deg = \bigoplus_j A(:, j)$ reduce to vertex degree vector
4. $wed = \text{perm2}(deg)$ apply perm2 to get wedge count vector
5. $lcc = tri2 \ominus wed$ LCC vector

LCC: FURTHER OPTIMIZATIONS

Further optimization: use $L$, the lower triangular part of $A$.

$$\text{TRI}(A) = A \bigoplus \bigotimes L$$

The number of wedges is now the 2-combination of $\text{deg}$.

$$\text{comb2}(x) = \frac{x \cdot (x - 1)}{2}$$

Permuting the adjacency matrix allows further optimizations.
LCC EXAMPLE: LOWER TRIANGULAR PART OF MX.

\[ \text{TRI}(A) = A \oplus L \]

\[ \text{deg} = \begin{bmatrix} 2 \\ 4 \\ 3 \\ 5 \\ 3 \\ 3 \\ 4 \end{bmatrix} \]

\[ \text{comb2}(\ldots) \]

\[ \text{tri} = \begin{bmatrix} 1 \\ 3 \\ 2 \\ 4 \end{bmatrix} \]

\[ \text{wed} = \begin{bmatrix} 1 \\ 6 \\ 3 \\ 10 \end{bmatrix} \]

\[ \text{lcc} = \begin{bmatrix} 1.00 \\ 0.50 \\ 0.67 \\ 0.40 \\ 0.33 \end{bmatrix} \]
Graph algorithms in GraphBLAS

Triangle count / Cohen’s algorithm
COHEN’S ALGORITHM: PSEUDOCODE

Input: adjacency matrix $A$
Output: triangle count $t$
Workspace: matrices $L$, $U$, $B$, $C$

1. $L = \text{tril}(A)$  
   extract the lower triangle from $A$
2. $U = \text{triu}(A)$  
   extract the upper triangle from $A$
3. $B = L \odot \cdot \otimes U$  
   multiply matrices $L$ and $U$
4. $C = B \otimes A$  
   element-wise multiplication
5. $t = \sum C / 2$  
   sum the values in $C$ and divide by 2

COHEN’S ALGORITHM

\[ t = \sum C / 2 \]
COHEN’S ALGORITHM: MASKING

\[ C(A) = L \oplus \otimes U \]
\[ t = \sum C / 2 \]
Graph algorithms in GraphBLAS

Triangle count / Sandia algorithm
SANDIA ALGORITHM

Input: adjacency matrix $A$
Output: triangle count $t$
Workspace: matrices $L, U, B, C$

1. $L = \text{tril}(A)$
   extract the lower triangle from $A$
2. $C(L) = L \oplus \odot L$
   multiply matrices $L$ and $L$ using mask $L$
3. $t = \sum C$
   sum the values in $C$

M.M. Wolf et al. (Sandia National Laboratories):
Fast linear algebra-based triangle counting with KokkosKernels, HPEC 2017
SANDIA ALGORITHM

\[ C(L) = L \oplus \bigotimes L \]
\[ t = \sum C \]
Graph algorithms in GraphBLAS

Triangle count / CMU algorithm
CMU ALGORITHM

- Iterates on the vertices of the graph, extracts corresponding submatrices and computes $t = t + a_{10}^T \odot \otimes A_{20} \odot \otimes a_{12}$

- Tradeoffs:
  - does not require $mxm$, only $vxm$ and $mxv$
  - slower than $mxm$-based algorithms

- The formula is derived using the matrix trace $\text{tr}(A) = \sum_{i=0}^{n-1} A_{ii}$ and its invariant property under cyclic permutation, e.g. $\text{tr}(ABC) = \text{tr}(BCA) = \text{tr}(CAB)$. See the paper for details.

---

T.M. Low et al. (Carnegie Mellon University):
*First look: linear algebra-based triangle counting without matrix multiplication*, HPEC 2017
CMU ALGORITHM: PSEUDOCODE

Input: adjacency matrix $A$
Output: triangle count $t$
Workspace: matrix $A_{20}$, $a_{10}$, $a_{12}^\top$, $C$

1. for $i = 2$ to $n - 1$
2. $A_{20} = A[i + 1:n, 0:i - 1]$
3. $a_{10} = A[0:i - 1, i]$
4. $a_{12} = A[i, i + 1:n]$
5. $t = t + a_{10}^\top \oplus \otimes A_{20} \oplus \otimes a_{12}$
The “CMU algorithm” belongs to a family of algorithms which can be derived using the “FLAME approach”. There are 8 similar algorithms in total, the one presented here is Algorithm 2.

**Algorithm 2**

\[
\begin{align*}
\Delta &= \Delta + \frac{1}{2} \alpha_{01}^T A_{00} a_{01} \\
\Delta &= \Delta + a_{01}^T A_{02} a_{21}
\end{align*}
\]
Graph algorithms in GraphBLAS

Vertex-wise triangle count
VERTEX-WISE TRIANGLE COUNT

Triangle – Def 1: a set of three mutually adjacent vertices.

Def 2: a three-length closed path.

Usages:
- Global clustering coefficient
- Local clustering coefficient
- Finding communities

GraphChallenge.org: Raising the Bar on Graph Analytic Performance, HPEC 2018
TC: ELEMENT-WISE MULTIPLICATION

\[ \mathbf{TRI} = \mathbf{A} \oplus \mathbf{\otimes A} \mathbf{\otimes A} \]

\[ \mathbf{tri} = \left[ \oplus_j \mathbf{TRI}(i,j) \right]/2 \]

\( \mathbf{A} \oplus \mathbf{\otimes A} \) is still very dense.
TC: ELEMENT-WISE MULTIPLICATION

Masking limits where the operation is computed. Here, we use $A$ as a mask for $A \oplus \otimes A$.

$$\text{TRI}(A) = A \oplus \otimes A$$

$$\text{tri} = \left[ \oplus_j \text{TRI}(:,j) \right]/2$$
TC: ALGORITHM

Input: adjacency matrix $A$
Output: vector $\text{tri}$
Workspace: matrix $\text{TRI}$

1. $\text{TRI}(A) = A \oplus \otimes A$  
   compute the triangle count matrix
2. $\text{tri} = \left[ \oplus_j \text{TRI}(:,j) \right] / 2$  
   compute the triangle count vector

Optimization: use $L$, the lower triangular part of $A$ to avoid duplicates.

$\text{TRI}(A) = A \oplus \otimes L$

Worst-case optimal joins: There are deep theoretical connections between masked matrix multiplication and relational joins. It has been proven in 2013 that for the triangle query, binary joins always provide suboptimal runtime, which gave rise to new research on the family of worst-case optimal multi-way joins algorithms.
Graph algorithms in GraphBLAS

$k$-truss
K-TRUSS

- **Definition:** the $k$-truss is a subset of the graph with the same number of vertices, where each edge appears in at least $k - 2$ triangles in the original graph.
K-TRUSS ALGORITHM

- **Input:** adjacency matrix $\mathbf{A}$, scalar $k$
- **Output:** $k$-truss adjacency matrix $\mathbf{C}$
- **Helper:** $f(x, \text{support}) = x \geq \text{support}$

1. $\mathbf{C} = \mathbf{A}$
2. for $i = 1$ to $n - 1$
3. $\mathbf{C} \langle \mathbf{C} \rangle = \mathbf{C} \oplus \mathbf{C}$ \text{ use the “plus-and” semiringing}
4. $\mathbf{C} = f(\mathbf{C}, k - 2)$ \text{ drop entries in } \mathbf{C} \text{ less than } k - 2$
5. terminate if the number of non-zero values in $\mathbf{C}$ did not change

T.A. Davis: *Graph algorithms via SuiteSparse:GraphBLAS: triangle counting and k-truss*, HPEC 2018
Graph algorithms in GraphBLAS

Community detection using label propagation
Goal: assign a label to each vertex representing the community it belongs to. The algorithm (originally published in network science) is slightly altered to ensure deterministic execution. Initially:

\[ L_0(v) = v \]

In the \( k \)th iteration:

\[ L_k(v) = \min(\arg\max_l |u \in N(v) \mid L_{k-1}(u) = l|), \]

where \( N(v) \) is the set of neighbours of \( v \).

Run for \( t \) iterations or until reaching a fixed point.

IDEA: CAPTURE CDLP IN PURE GRAPHBLAS

- Define a semiring that operates over occurrence vectors
- $\oplus$ operator: combines two occurrence vectors
  - $\{6 \rightarrow 1, 9 \rightarrow 1\} \oplus \{6 \rightarrow 1, 7 \rightarrow 2\} = \{6 \rightarrow 2, 7 \rightarrow 2, 9 \rightarrow 1\}$
- Convert each element in a row to an occurrence vector
  - $\{6 \rightarrow 1\}, \{6 \rightarrow 1\}, \{7 \rightarrow 1\}, \{7 \rightarrow 1\}, \{9 \rightarrow 1\}$
- Reduce each row into a single occurrence vector:
  - $\{6 \rightarrow 2, 7 \rightarrow 2, 9 \rightarrow 1\}$
- Select the min. mode element from the occurrence vector
  - $6$
- Works on paper, but occurrence vectors need dynamic memory allocation, which leads to very poor performance
CDLP IN LINEAR ALGEBRA

- Extract each row from $F$
  - Easy if the matrix is stored in CSR format
- Select the minimum mode value in each row
  - Sort elements using parallel merge sort
  - Pick the min value that has the longest run (done in a single pass)
- Sort each row $r$
- Use the sorted list to compute $\text{mode}(r)$
Initially, $\text{lab} = [1, 2, \ldots, n]$

- Propagate labels to create a “frequency matrix”:
  $F = A \text{any.sel2nd diag(\text{lab})}$
CDLP EXAMPLE

step: 1
CDLP EXAMPLE

step: 2
CDLP EXAMPLE

step: 3
CDLP EXAMPLE

step: 4
same result as in step 2
CDLP: ALGORITHM

Input: adjacency matrix $A$, #vertices $n$, #iterations $t$
Output: vector $\text{lab}$
Workspace: matrix $F$, vector $r$

1. $\text{lab} = [1 \ 2 \ ... \ n]$
2. for $k = 1$ to $t$
3. \hspace{1em} $F = A \ \text{any.sel2nd diag(\text{lab})}$
4. for $i = 1$ to $n$
5. \hspace{1em} $r = F(i,:)$
6. \hspace{3em} sort($r$)
7. \hspace{1em} $\text{lab}(i) = \text{select\_min\_mode}(r)$

Can be batched and parallelized
CDLP: ALGORITHM

Input: adjacency matrix $A$, #vertices $n$, #iterations $t$
Output: vector $\text{lab}$
Workspace: matrix $F$, vector $r$, array of row indices $I$, array of values $X$

1. $\text{lab} = [1 \ 2 \ldots \ n]$
2. for $k = 1$ to $t$
3. $F = A \ \text{any.sel2nd diag}(\text{lab})$
4. $\langle I, _, X \rangle = \text{extract_tuples}(F)$
5. $\text{merge_sort_pairs}(\langle I, X \rangle)$
6. $\text{lab} = \text{for each row in } I, \text{ select min mode value from } X$
For directed graphs, we compute the labels $L_k(v)$ as:

$$\min(\arg\max_l [\{u \in N_{in}(v) | L_{k-1}(u) = l\} + \{u \in N_{out}(v) | L_{k-1}(u) = l\}])$$

- In linear algebra, this can be expressed with two matrices:
  - $F_{in} = A \text{ any. sel2nd diag(lab)}$
  - $F_{out} = A^T \text{ any. sel2nd diag(lab)}$

- Simultaneously iterate over rows $r_{in}$ of $F_{in}$ and $r_{out}$ of $F_{out}$
- For each row pair, sort $r_{in} \cup r_{out}$ and select the minimum mode value
- Batching also works:
  - $\langle I_{in}, X_{in} \rangle = \text{extract} \_\text{tuples}(F_{in})$
  - $\langle I_{out}, X_{out} \rangle = \text{extract} \_\text{tuples}(F_{out})$
  - merge_sort_pairs($\langle I_{in} \cup I_{out}, X_{in} \cup X_{out} \rangle$)
Graph algorithms in GraphBLAS

Other algorithms
OTHER ALGORITHMS IN GRAPHBLAS

Betweenness centrality: Brandes’ algorithm

A. Buluç et al.: Design of the GraphBLAS C API, GABB@IPDPS 2017

\(k\text{-}\text{truss}\): a subset of the graph with the same number of vertices, where each edge appears in at least \(k - 2\) triangles in the original graph.

T.A. Davis: Graph algorithms via SuiteSparse:GraphBLAS: triangle counting and \(k\)-truss, HPEC 2018

Maximal independent set: Luby’s randomized algorithm

T.A. Davis: Algorithm 1000: SuiteSparse:GraphBLAS: graph algorithms in the language of sparse linear algebra, ACM TOMS, 2019

Sparse DNNs: represent sparse deep neural networks as graphs

J. Kepner et al.: Enabling Massive Deep Neural Networks with the GraphBLAS, HPEC 2017

T.A. Davis et al.: Write Quick, Run Fast: Sparse Deep Neural Network in 20 Minutes of Development Time via SuiteSparse:GraphBLAS, HPEC 2019
Graph algorithms in GraphBLAS

Weakly connected components
WCC: WEAKLY CONNECTED COMPONENTS (SKETCH)

\[ C = I \lor A \lor A^2 \lor A^3 \lor A^4 \lor \ldots \]
\[ D = I + A + A^2 + A^3 + A^4 \lor \ldots \lor A^K \]
\[ E \equiv (I - A)D \]
\[ E = D - AD \]
\[ = I + A + A^2 + A^3 + A^4 + \ldots - A - A^2 - A^3 - A^4 - \ldots \]
\[ = I \]
\[ = (I - A)D \]
\[ D = (I - A)^{-1} \]

J. Kepner, J.R. Gilbert:
*Graph Algorithms in the Language of Linear Algebra*, Chapter 3, SIAM, 2011
WCC: WEAKLY CONNECTED COMPONENTS

- The series for $D$ does usually not converge. However, by replacing $A$ with $\alpha A$, where $\alpha$ is a sufficiently small positive number, the series converges. The derivation presented in the previous slide is still valid, resulting in $D = (I - \alpha A)^{-1}$.

- While this trick ensures that the presented algorithm works correctly, this approach is still impractical as it computes the inverse of a sparse matrix which is dense in most cases.
WCC: WEAKLY CONNECTED COMPONENTS

- Linear Algebraic Connected Components algorithm (LACC)
  o Recent result, based on the Awerbuch-Shiloach algorithm.
  o Designed for supercomputers, scales to 250,000+ CPU cores.
  o “Distributed-memory LACC code that is used in our experiments is publicly available as part of the CombBLAS library. A simplified unoptimized serial GraphBLAS implementation is also committed to the LAGraph Library for educational purposes.”

- See also FastSV.

A. Azad, A. Buluç: LACC: A Linear-Algebraic Algorithm for Finding Connected Components in Distributed Memory, IPDPS 2019
Graph algorithms in GraphBLAS

Maximal independent set: Luby’s algorithm
LUBY’S ALGORITHM

To be done later.
Push/pull BFS in GraphBLAS
This optimization is subject to future work.

For low diameter graphs, it is worth using push/pull phases.

Push/pull is simple to express in GraphBLAS.

- See [Yang et al., ICPP’18], [Yang et al., preprint’19]

But deciding when to change is non-trivial.