GraphBLAS: A linear algebraic approach for high-performance graph algorithms

Gábor Szárnyas
szarnyas@mit.bme.hu
What makes graph computations difficult?
The GraphBLAS standard
Theoretical foundations of GraphBLAS
Graph algorithms in GraphBLAS
Graph processing in relational algebra
GraphBLAS internals and API
Further reading and libraries
Summary
What makes graph computations difficult?
GRAPH PROCESSING CHALLENGES

the “curse of connectedness”

contemporary computer architectures are good at processing linear and hierarchical data structures, such as Lists, Stacks, or Trees

a massive amount of random data access is required, CPU has frequent cache misses, and implementing parallelism is difficult

B. Shao, Y. Li, H. Wang, H. Xia (Microsoft Research), *Trinity Graph Engine and its Applications*, IEEE Data Engineering Bullelting 2017
GRAPH PROCESSING CHALLENGES

What does it mean that graph algorithms have “a high communication to computation ratio”? [ICCS’15]

Most of the time is spent on chasing pointers.

Speedup with a CPU that has better arithmetic performance:
- machine learning → a lot
- relational query → some
- graph processing → very little

Standard latency hiding techniques break down:
- pre-fetching and branch prediction provide little benefit
VERTEX-CENTRIC PROGRAMMING MODEL

- Pregel paper
- SIGMOD 2020 “Test of time” award
- Enables distributed and fault-tolerant processing

G. Malewicz et al. (Google),
_Pregel: A System for Large-Scale Graph Processing_,
SIGMOD 2010
DISTRIBUTED GRAPH PROCESSING

- Many recent works focused on computation models for distributed execution allowing systems to scale out.
  - BSP (Bulk Synchronous Parallel) model by Leslie Valiant
  - MapReduce and Pregel models by Google
  - Vertex-centric, Scatter-Gather, Gather-Apply-Scatter models
  - Apache projects: Giraph, Spark GraphX, Flink Gelly, Hama
- Lots of research, summarized in survey/experiments papers
  
  V. Kalavri et al.: High-Level Programming Abstractions for Distributed Graph Processing, TKDE 2018
  
  K. Ammar, M.T. Özsu: Experimental Analysis of Distributed Graph Systems, VLDB 2018
  
  O. Batarfi et al.: Large scale graph processing systems: survey and an experimental evaluation, Cluster 2015
  
  M.T. Özsu: Graph Processing: A Panaromic View and Some Open Problems, VLDB 2019
SCALING OUT VS. SCALING UP

- Distributed approaches are **scalable but comparatively slow**
  - large communication overhead
  - load balancing issues due to irregular distributions

- Many systems **struggle to outperform a single-threaded setup**
  - COST = Configuration that Outperforms a Single Thread

- Alternatives:
  - Partition-centric programming model (Blogel, etc.)
  - Linear algebra-based programming model

---

F. McSherry et al.: *Scalability! But at what COST?* HotOS 2015

N. Satish et al.: *Navigating the Maze of Graph Analytics Frameworks using Massive Graph Datasets*, SIGMOD 2014
LINEAR ALGEBRA-BASED GRAPH PROCESSING

- Graphs are encoded as **sparse adjacency matrices**.
- Use **vector/matrix operations** to express graph algorithms.

\[
\begin{pmatrix}
1 & 1 \\
1 & 1 \\
1 & 1 \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
1 & 1 \\
1 & 1 \\
1 & 1 \\
\end{pmatrix}
\]
TEXTBOOKS ON SEMIRING-BASED GRAPH PROCESSING

  - *The Design and Analysis of Computer Algorithms*
    - Sec. 5.6: Path-finding problems
    - Sec. 5.9: Path problems and matrix multiplication

- 1990: Cormen-Leiserson-Rivest book
  - *Introduction to Algorithms*
    - Sec. 26.4: A general framework for solving path problems in directed graphs

- 2011: GALLA book (edited by Kepner and Gilbert)
  - *Graph Algorithms in the Language of Linear Algebra*

A lot of literature but few practical implementations.
The GraphBLAS standard
THE GRAPHBLAS STANDARD

**Goal:** separate the concerns of the hardware/library/application designers.

- **1979:** BLAS  
  Basic Linear Algebra Subprograms
- **2001:** Sparse BLAS  
  an extension to BLAS (little uptake)
- **2013:** GraphBLAS  
  an effort to define standard building blocks for graph algorithms in the language of linear algebra

---

**Figure:**

- Numerical applications: LINPACK/LAPACK
- Graph analytical apps: LAGraph
- BLAS
- Hardware architecture
- Separation of concerns
- GraphBLAS
- Hardware architecture

---

S. McMillan @ SEI Research Review (Carnegie Mellon University, 2015):  
*Graph algorithms on future architectures*
### Graph Algorithms in Linear Algebra

**Notation:** \( n = |V|, m = |E| \). The complexity cells contain asymptotic bounds.

**Takeaway:** The majority of common graph algorithms can be expressed efficiently in LA.

<table>
<thead>
<tr>
<th></th>
<th>Algorithm</th>
<th>Canonical Complexity ( \Theta )</th>
<th>LA-based Complexity ( \Theta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>breadth-first search</td>
<td>Dijkstra</td>
<td>( m )</td>
<td>( m )</td>
</tr>
<tr>
<td>single-source shortest paths</td>
<td>Dijkstra</td>
<td>( m + n \log n )</td>
<td>( n^2 )</td>
</tr>
<tr>
<td></td>
<td>Bellman-Ford</td>
<td>( mn )</td>
<td>( mn )</td>
</tr>
<tr>
<td>all-pairs shortest paths</td>
<td>Floyd-Warshall</td>
<td>( n^3 )</td>
<td>( n^3 )</td>
</tr>
<tr>
<td>minimum spanning tree</td>
<td>Prim</td>
<td>( m + n \log n )</td>
<td>( n^2 )</td>
</tr>
<tr>
<td></td>
<td>Borůvka</td>
<td>( m \log n )</td>
<td>( m \log n )</td>
</tr>
<tr>
<td>maximum flow</td>
<td>Edmonds-Karp</td>
<td>( m^2 n )</td>
<td>( m^2 n )</td>
</tr>
<tr>
<td>maximal independent set</td>
<td>greedy</td>
<td>( m + n \log n )</td>
<td>( mn + n^2 )</td>
</tr>
<tr>
<td></td>
<td>Luby</td>
<td>( m + n \log n )</td>
<td>( m \log n )</td>
</tr>
</tbody>
</table>


See also L. Dhulipala, G.E. Blelloch, J. Shun: *Theoretically Efficient Parallel Graph Algorithms Can Be Fast and Scalable*, SPAA 2018
Theoretical foundations of GraphBLAS
**Matrix Multiplication**

**Definition:**

\[
C = AB \\
C(i, j) = \sum_k A(i, k) \cdot B(k, j)
\]

**Example:**

\[
C(2,3) = A(2,1) \cdot B(1,3) + A(2,2) \cdot B(2,3) \\
= 2 \cdot 5 + 3 \cdot 4 = 22
\]

\[
C = A \cdot B
\]
ADJACENCY MATRIX

\[ A_{ij} = \begin{cases} 
1 & \text{if } (v_i, v_j) \in E \\
0 & \text{if } (v_i, v_j) \notin E 
\end{cases} \]
ADJACENCY MATRIX

\[ A_{ij} = \begin{cases} 
1 & \text{if } (v_i, v_j) \in E \\
0 & \text{if } (v_i, v_j) \notin E 
\end{cases} \]
**ADJACENCY MATRIX**

\[ A_{ij} = \begin{cases} 
1 & \text{if } (v_i, v_j) \in E \\
0 & \text{if } (v_i, v_j) \notin E 
\end{cases} \]

\[
\begin{array}{cccccccc}
\hline
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
2 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
3 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
4 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
5 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
6 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
7 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\hline
\end{array}
\]
ADJACENCY MATRIX TRANSPOSED

\[ A_{ij}^T = \begin{cases} 
1 & \text{if } (v_j, v_i) \in E \\
0 & \text{if } (v_j, v_i) \notin E 
\end{cases} \]
ADJACENCY MATRIX TRANSPOSED

\[
A_{ij}^T = \begin{cases} 
1 & \text{if } (v_j, v_i) \in E \\
0 & \text{if } (v_j, v_i) \notin E
\end{cases}
\]
GRAPH TRAVERSAL WITH MATRIX MULTIPLICATION

\[ f \times A^k \] means \( k \) hops in the graph

One-hop: \( fA \)
GRAPH TRAVERSAL WITH MATRIX MULTIPLICATION

\[ fA^k \] means \( k \) hops in the graph

one-hop: \( fA \)

two-hop: \( fA^2 \)
MATRIX MULTIPLICATION ON SEMIRINGS

- Using the conventional semiring
  \[ C = AB \]
  \[ C(i, j) = \sum_k A(i, k) \cdot B(k, j) \]

- Generalized formula*
  \[ C = A \oplus \odot B \]
  \[ C(i, j) = \bigoplus_k A(i, k) \odot B(k, j) \]

- A cornerstone of GraphBLAS: Use arbitrary semirings that override the \(\oplus\) addition and \(\odot\) multiplication operators.

* Remark: Some definitions in the presentation are simplified. The full definitions are given at the end of the slideshow.
GRAPHBLAS SEMIRINGS*

The \( \langle D, \oplus, \otimes, 0 \rangle \) algebraic structure is a GraphBLAS semiring if

- \( \langle D, \oplus, 0 \rangle \) is a commutative monoid using the addition operation \( \oplus : D \times D \rightarrow D \), where \( \forall a, b, c \in D \):
  - Commutative \( a \oplus b = b \oplus a \)
  - Associative \( (a \oplus b) \oplus c = a \oplus (b \oplus c) \)
  - Identity \( a \oplus 0 = a \)

- The multiplication operator is a closed binary operator \( \otimes : D \times D \rightarrow D \).

The mathematical definition of a semiring requires that \( \otimes \) is a monoid and distributes over \( \oplus \). GraphBLAS omits these requirements.
## Semirings

<table>
<thead>
<tr>
<th>Semiring</th>
<th>Set</th>
<th>$\oplus$</th>
<th>$\otimes$</th>
<th>0</th>
<th>Graph semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>lor-land</td>
<td>$a \in {F, T}$</td>
<td>$\lor$</td>
<td>$\land$</td>
<td>$F$</td>
<td>connectivity</td>
</tr>
<tr>
<td>Integer arithmetic</td>
<td>$a \in \mathbb{N}$</td>
<td>$+$</td>
<td>$\cdot$</td>
<td>$0$</td>
<td>number of paths</td>
</tr>
<tr>
<td>Real arithmetic</td>
<td>$a \in \mathbb{R}$</td>
<td>$+$</td>
<td>$\cdot$</td>
<td>$0$</td>
<td>strength of all paths</td>
</tr>
<tr>
<td>Min-plus</td>
<td>$a \in \mathbb{R} \cup {+\infty}$</td>
<td>$\min$</td>
<td>$+$</td>
<td>$+\infty$</td>
<td>shortest path</td>
</tr>
<tr>
<td>Max-plus</td>
<td>$a \in \mathbb{R} \cup {-\infty}$</td>
<td>$\max$</td>
<td>$+$</td>
<td>$-\infty$</td>
<td>graph matching</td>
</tr>
</tbody>
</table>

The default semiring is the conventional one:
- Operator $\otimes$ defaults to floating point multiplication.
- Operator $\oplus$ defaults to floating point addition.
Semantics: number of paths

Semiring: integer arithmetic

Set: $a \in \mathbb{N}$

Operations: $+$, $\cdot$, $0$

Matrix $A$ and vector $f$ showing the multiplication and addition operations.

$1 \cdot 1 = 1$

$1 + 1 = 2$
MATRIX-VECTOR MULTIPLICATION SEMANTICS

Semiring  set  ⊕  ⊗  0
lor-land  a ∈ {F, T}  ∨  ∧  F

Semantics: reachability

```
A
1 2 3 4 5 6 7
1 T T T
2 T T T
3 T T T
4 T T T
5 T T T
6 T T T
7 T T T
```

```
f
1 2 3 4 5 6 7
1 T T T
2 T T T
3 T T T
4 T T T
5 T T T
6 T T T
7 T T T
```

T ∨ T = T
T ∧ T = T

f ∨ ∧ A
MATRIX-VECTOR MULTIPLICATION SEMANTICS

<table>
<thead>
<tr>
<th>Semiring</th>
<th>Set</th>
<th>⊕</th>
<th>⊗</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>real arithmetic</td>
<td>( a \in \mathbb{R} )</td>
<td>+</td>
<td>⋅</td>
<td>0</td>
</tr>
</tbody>
</table>

Semantics: strength of all paths

\[
\begin{align*}
A &= \begin{pmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\end{pmatrix}
\end{align*}
\]

\[
\begin{align*}
\mathbf{f} &= \begin{pmatrix}
0.5 \cdot 0.4 = 0.2 \\
0.6 \cdot 0.5 = 0.3 \\
0.2 + 0.3 = 0.5 \\
\end{pmatrix}
\end{align*}
\]

\[
\mathbf{f} \oplus \mathbf{A} 
\]
**Matrix-Vector Multiplication Semantics**

<table>
<thead>
<tr>
<th>semiring</th>
<th>set</th>
<th>( \oplus )</th>
<th>( \otimes )</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>min-plus</td>
<td>( a \in \mathbb{R} \cup {+\infty} )</td>
<td>min</td>
<td>+</td>
<td>+( \infty )</td>
</tr>
</tbody>
</table>

Semantics: shortest path

### Example

- **Matrix** \( A \):
  
<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Vector** \( f \):
  
  \[
  f = \begin{pmatrix}
  0.5 \\
  0.4 \\
  0.6 \\
  0.5 \\
  0.6 \\
  0.7 \\
  0.9 \\
  \infty
  \end{pmatrix}
  \]

- **Result** \( f \min + A \):
  
  \[
  f \min + A = \begin{pmatrix}
  0.9 \\
  1.1 \\
  0.9 \\
  0.9 \\
  1.1 \\
  0.7 \\
  0.9 \\
  \infty
  \end{pmatrix}
  \]
MATRIX-VECTOR MULTIPLICATION SEMANTICS

<table>
<thead>
<tr>
<th>semiring</th>
<th>set</th>
<th>(\oplus)</th>
<th>(\otimes)</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>max-plus</td>
<td>(a \in \mathbb{R} \cup {-\infty})</td>
<td>max</td>
<td>+</td>
<td>(-\infty)</td>
</tr>
</tbody>
</table>

Semantics: matching (independent edge set)

\[
\begin{align*}
A &= \begin{pmatrix}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
\end{pmatrix}
\end{align*}
\]

\[
\begin{align*}
0.5 + 0.4 &= 0.9 \\
0.6 + 0.5 &= 1.1 \\
\end{align*}
\]

\[
\begin{align*}
\text{max}(0.9, 1.1) &= 1.1 \\
f \max + A &= \begin{pmatrix}
\end{align*}
\]
**Matrix-Vector Multiplication Semantics**

- **Semiring**: set
- **Set**: \( a \in \mathbb{R} \cup \{+\infty\} \)
- **Semantics**: shortest product of connections

**Example**:

Given a matrix \( A \) and a vector \( f \), the output is calculated as follows:

\[
\begin{align*}
A & = \begin{pmatrix}
1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\text{\(1\)} & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\
\text{\(2\)} & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\
\text{\(3\)} & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\
\text{\(4\)} & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\
\text{\(5\)} & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\
\text{\(6\)} & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\
\text{\(7\)} & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\
\end{pmatrix} \\
\end{align*}
\]

- \( f = \begin{pmatrix} .5 \ 0.6 \end{pmatrix} \)
- \( f \min \otimes A = \begin{pmatrix} .5 \ .6 \ \infty \end{pmatrix} \)

- Calculations:
  - \( 0.5 \cdot 0.4 = 0.2 \)
  - \( 0.6 \cdot 0.5 = 0.3 \)
- \( \text{min}(0.2, 0.3) = 0.2 \)
MATRIX-VECTOR MULTIPLICATION SEMANTICS

Semantics: longest of all shortest connections

Semiring | Set | + | × | 0
---|---|---|---|---
max-min | $a \in \{0, +\infty\}$ | max | min | 0

min(0.5, 0.4) = 0.4
min(0.6, 0.5) = 0.5
max(0.4, 0.5) = 0.5

$f \max \cdot \min A$
ELEMENT-WISE MULTIPLICATION: \( A \land B \)
ELEMENT-WISE ADDITION: $A \lor B$
TURNING A GRAPH INTO UNDIRECTED: $A \lor A^T$
NOTATION*

- **Symbols:**
  - $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{M}$ – matrices
  - $\mathbf{u}, \mathbf{v}, \mathbf{w}, \mathbf{m}$ – vectors
  - $s$ – scalar
  - $i, j$ – indices
  - $\langle \mathbf{M} \rangle, \langle \mathbf{m} \rangle$ – masks

- **Operators:**
  - $\oplus$ – addition
  - $\otimes$ – multiplication
  - $\top$ – transpose
  - $\oslash$ – element-wise division

Vectors can act as both column and row vectors.

(Notation omitted for accumulator, selection, extraction, assignment...)
LINEAR ALGEBRAIC PRIMITIVES FOR GRAPHS #1

Element-wise addition: union of non-zero elements

Element-wise multiplication: intersection of non-zero elements

Sparse matrix times sparse vector: process incoming edges

Sparse vector times sparse matrix: process outgoing edges
Sparse matrix times sparse matrix:
process connecting outgoing edges

Matrix transpose:
reverse edges

Reduction:
aggregate values in each row

Apply:
apply unary operator on all values
Graph algorithms in GraphBLAS
ALGORITHMS

- Breadth-first search
  - Levels
  - Parents
  - Multi-source
- Bellman-Ford
- PageRank
- Triangle count
  - Naïve
  - Masked
  - Cohen’s algorithm
  - Sandia
- CMU
- Local clustering coefficient
- \( k \)-truss
- CDLP (Community Detection using Label Propagation)
Graph algorithms in GraphBLAS

Breadth-first search
BFS: BREADTH-FIRST SEARCH

- **Algorithm:**
  - Start from a given vertex
  - “Explore all neighbour vertices at the present level prior to moving on to the vertices at the next level” [Wikipedia]

- **Variants:**
  - **Levels** compute traversal level for each vertex
  - **Parents** compute parent for each vertex
  - **MSBFS** start traversal from multiple source vertices
Graph algorithms in GraphBLAS

Breadth-first search / Levels
BFS – LEVELS

<table>
<thead>
<tr>
<th>semiring</th>
<th>set</th>
<th>$\oplus$</th>
<th>$\otimes$</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>lor-land</td>
<td>$a \in {F, T}$</td>
<td>$\lor$</td>
<td>$\land$</td>
<td>$F$</td>
</tr>
</tbody>
</table>

level = 1

$f(\vec{v}) = f \lor \land A$
BFS – LEVELS

<table>
<thead>
<tr>
<th>semiring</th>
<th>set</th>
<th>⨁</th>
<th>⨂</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>lor-land</td>
<td>$a \in {F,T}$</td>
<td>$\lor$</td>
<td>$\land$</td>
<td>$F$</td>
</tr>
</tbody>
</table>

$level = 2$

$\vec{v}_1 \rightarrow 2 \rightarrow 7 \rightarrow 5 \rightarrow 4 \rightarrow 3 \rightarrow 6 \rightarrow 1$

$q$:

$$f(\vec{v}) = f \lor \land A$$
BFS – LEVELS

<table>
<thead>
<tr>
<th>semiring</th>
<th>set</th>
<th>$\oplus$</th>
<th>$\otimes$</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>lor-land</td>
<td>$a \in {F, T}$</td>
<td>$\lor$</td>
<td>$\land$</td>
<td>$F$</td>
</tr>
</tbody>
</table>

level = 3

$q$

$v$

$f\langle \vec{v} \rangle = f \lor \land A$
BFS – LEVELS

<table>
<thead>
<tr>
<th>semiring</th>
<th>set</th>
<th>$\oplus$</th>
<th>$\otimes$</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>lor-land</td>
<td>$a \in {F, T}$</td>
<td>$\lor$</td>
<td>$\land$</td>
<td>$F$</td>
</tr>
</tbody>
</table>

$level = 4$

$f(\vec{v}) = f \lor \land A$

$f$ is empty $\rightarrow$ terminate
### BFS – LEVELS: ALGORITHM

- **Input:** adjacency matrix $A$, source vertex $s$, #vertices $n$
- **Output:** vector of visited vertices $v$ (integer)
- **Workspace:** frontier vector $f$ (Boolean)

1. $f(s) = T$
2. for $level = 1$ to $n - 1$  
   *terminate earlier if $f$ is empty*
3. $v(f) = level$  
   assign level value to the vertices in the frontier
4. clear($f$)  
   clear the frontier $f$ (all elements set to FALSE)
5. $f(\overline{v}) = f \lor .\land A$  
   use the lor-land semiring to advance the frontier
BFS – LEVELS: OPTIMIZATIONS

- Instead of the $\lor \land$ semiring, one can use the any. pair semiring where
  - any($x, y$) randomly returns $x$ or $y$. This operator is commutative and associative but does not have an identity element.
  - pair($x, y$) = 1. This is a closed binary operator.

- These are available built-in in SuiteSparse as GxB_ANY and GxB_PAIR. The documentation says:
  - The result of the ANY monoid is non-deterministic, unless it is coupled with the PAIR multiplicative operator. In this case, the ANY_PAIR semiring will return a deterministic result, since $f(1, 1)$ is always 1, for the ANY operator $f(x, y)$. 
Graph algorithms in GraphBLAS

Breadth-first search / Parents
BFS – PARENTS

<table>
<thead>
<tr>
<th>semiring</th>
<th>set</th>
<th>$\oplus$</th>
<th>$\otimes$</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>min-select 1st</td>
<td>$a \in \mathbb{N} \cup {+\infty}$</td>
<td>min</td>
<td>sel 1st</td>
<td>$+\infty$</td>
</tr>
</tbody>
</table>

$\text{sel 1st}(x, y) = x$

$f_{\langle \mathbf{p} \rangle} = f_{\min \text{sel 1st } \mathbf{A}}$

$p_{\langle f \rangle} = f$

$f(f) = \text{id}$
BFS – PARENTS

<table>
<thead>
<tr>
<th>semiring</th>
<th>set</th>
<th>⊕</th>
<th>⊗</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>min-select 1st</td>
<td>a ∈ N U {+∞}</td>
<td>min</td>
<td>sel1st</td>
<td>+∞</td>
</tr>
</tbody>
</table>

sel1st(x, y) = x

\[ f(p) = f \text{ min. sel1st } A \]

\[ f^{-1}(f) = f \]

\[ f(f) = \text{id} \]
BFS – PARENTS

<table>
<thead>
<tr>
<th>semiring</th>
<th>set</th>
<th>$\oplus$</th>
<th>$\otimes$</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>min-select 1st</td>
<td>$a \in \mathbb{N} \cup {+\infty}$</td>
<td>min</td>
<td>sel1st</td>
<td>$+\infty$</td>
</tr>
</tbody>
</table>

$\text{sel1st}(x, y) = x$

$\mathbf{f} \mathbf{p} = \mathbf{f}_{\text{min sel1st } \mathbf{A}}$

$\mathbf{p} \mathbf{f} = \mathbf{f}$

$\mathbf{f}(\mathbf{p}) = \text{id}$
**BFS – PARENTS**

<table>
<thead>
<tr>
<th>semiring</th>
<th>set</th>
<th>⊕</th>
<th>⊗</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>min-select 1st</td>
<td>$a \in \mathbb{N} \cup {+\infty}$</td>
<td>min</td>
<td>sel1st</td>
<td>$+\infty$</td>
</tr>
</tbody>
</table>

$$\text{sel1st}(x, y) = x$$

$f(p) = f \text{ min. sel1st } A$

$f$ is empty $\rightarrow$ terminate
BFS – PARENTS: ALGORITHM

- **Input:** adjacency matrix \( A \), source vertex \( s \), #vertices \( n \)
- **Output:** parent vertices vector \( p \) (integer)
- **Workspace:** vertex id vector \( i_d \) (integer), frontier vector \( f \) (integer)

1. \( i_d = [1 \ 2 \ \ldots \ n] \)  
   this talk uses 1-based indexing (unlike GrB)
2. \( f(s) = s \)
3. \( p(s) = -1 \)
4. for \( l = 1 \) to \( n - 1 \)  
   *terminate earlier if we reach a fixed point*
5. \( f\langle\overline{p}\rangle = f \min. \text{sel1st} A \)  
   advance the frontier
6. \( p\langle f\rangle = f \)  
   assign parent ids to the frontier’s vertices
7. \( f\langle f\rangle = i_d \)  
   assign vertex ids \( f(i) = i \)
BFS – PARENTS: OPTIMIZATIONS

- If getting deterministic results is not a requirement (i.e. any parent vertex can be returned), instead of min. sel1st, one can use the any. sel1st semiring.
- This optimization is allowed by the GAP Benchmark Suite.
- Direction-optimizing traversal (push/pull) can be exploited.

[TODO] Details are discussed in the IISWC’20 GAP paper.
Graph algorithms in GraphBLAS

Multi-source BFS
MULTI-SOURCE BFS – LEVELS

<table>
<thead>
<tr>
<th>semiring</th>
<th>set</th>
<th>(\oplus)</th>
<th>(\otimes)</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>lor-land</td>
<td>(a \in {F, T})</td>
<td>(\lor)</td>
<td>(\land)</td>
<td>(F)</td>
</tr>
</tbody>
</table>

F

\(F(\overline{V}) = F \lor \land A\)
MULTI-SOURCE BFS – PARENTS

<table>
<thead>
<tr>
<th>semiring</th>
<th>set</th>
<th>⊕</th>
<th>⊗</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>min-select 1st</td>
<td>$a \in \mathbb{N} \cup {+\infty}$</td>
<td>min</td>
<td>sel1st</td>
<td>$+\infty$</td>
</tr>
</tbody>
</table>

$F(P) = F \text{ min. sel1st } A$
Various GraphBLAS-based MSBFS algorithms are discussed in our paper tackling the SIGMOD 2014 programming contest’s queries.

[TODO] Details to follow later.
BFS – PERFORMANCE

- Naïve BFS impls are sometimes slow on real graphs with skewed distributions – further optimizations are needed.

- Direction-optimizing BFS was published in 2012.
  - Switches between push ($vA$) and pull ($A^Tv$) during execution:
    - Use the push direction when the frontier is small
    - Use the pull direction when the frontier becomes large
  - Adopted to GraphBLAS in 2018 (Yang et al.’s ICPP paper)
  - [TODO] Explain this in more details

---

C. Yang, A. Buluç, J.D. Owens: *Implementing Push-Pull Efficiently in GraphBLAS*, ICPP 2018

A. Buluç: *GraphBLAS: Concepts, algorithms, and applications*, Scheduling Workshop, 2019
Graph algorithms in GraphBLAS

Single-source shortest path
SSSP – SINGLE-SOURCE SHORTEST PATHS

- Problem:
  - From a given start vertex $s$, find the shortest paths to every other (reachable) vertex in the graph

- Bellman-Ford algorithm:
  - Relaxes all edges in each step
  - Guaranteed to find the shortest path using at most $n - 1$ steps

- Observation:
  - The relaxation step can be captured using a VM multiplication
### SSSP – ALGEBRAIC BELLMAN-FORD

<table>
<thead>
<tr>
<th>semiring</th>
<th>set</th>
<th>$\oplus$</th>
<th>$\otimes$</th>
<th>$0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>min-plus</td>
<td>$a \in \mathbb{R} \cup {+\infty}$</td>
<td>min</td>
<td>$+$</td>
<td>$+\infty$</td>
</tr>
</tbody>
</table>

#### Example

**Graph:**
- Node 1 to 2: $0.3$
- Node 2 to 3: $0.5$
- Node 2 to 4: $0.8$
- Node 2 to 5: $0.7$
- Node 2 to 6: $0.1$
- Node 3 to 4: $0.4$
- Node 3 to 6: $0.1$
- Node 4 to 7: $0.8$
- Node 4 to 5: $0.5$
- Node 5 to 6: $0.1$
- Node 6 to 7: $0.5$

**Matrix $A$:**

$$
\begin{pmatrix}
0 & 0.3 & 0.8 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0.1 & 0.2 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.9 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
$$

**Matrix $D$:**

$$
\begin{pmatrix}
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0
\end{pmatrix}
$$

**$d$ min. + $A$:**

$$
\begin{pmatrix}
1 & 2 & 3 & 4 & 5 & 6 & 7
\end{pmatrix}
\begin{pmatrix}
0.3 \\
0.1 \\
0.4 \\
0.8 \\
0.5 \\
0.1 \\
0.5 \\
0.9
\end{pmatrix}
$$
SSSP – ALGEBRAIC BELLMAN-FORD

<table>
<thead>
<tr>
<th>semiring</th>
<th>set</th>
<th>⊕</th>
<th>⊗</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>min-plus</td>
<td>$a \in \mathbb{R} \cup {+\infty}$</td>
<td>min</td>
<td>+</td>
<td>$+\infty$</td>
</tr>
</tbody>
</table>

\[ d \]

\[
\begin{array}{ccccccc}
\text{min-plus} & \min-\text{plus} & \infty & \min & + & \infty \\
\text{set} & a & \in & \mathbb{R} & \cup & \{+\infty\} & \text{min} & + & \infty \\
\end{array}
\]

\[
\begin{array}{ccccccc}
\text{A} & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
1 & 0 & .3 & .8 & & & & \\
2 & 0 & 0 & .1 & .7 & & & \\
3 & 0 & 0 & 0 & .5 & & & \\
4 & .2 & .4 & 0 & & & & \\
5 & & & & & & & \\
6 & .5 & 0 & & & & & \\
7 & .1 & .5 & .9 & 0 & & & \\
\end{array}
\]

\[
\begin{array}{ccccccc}
d & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
1 & 0 & & & & & & \\
2 & .3 & .8 & & & & & \\
3 & & & & & & & \\
4 & & & & & & & \\
5 & & & & & & & \\
6 & & & & & & & \\
7 & & & & & & & \\
\end{array}
\]

\[
d \quad \min.+ \quad A
\]
SSSP – ALGEBRAIC BELLMAN-FORD

**Semiring and Set**

<table>
<thead>
<tr>
<th>Semiring</th>
<th>Set</th>
<th>⊕</th>
<th>⊗</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>min-plus</td>
<td>$a \in \mathbb{R} \cup {+\infty}$</td>
<td>min</td>
<td>+</td>
<td>+∞</td>
</tr>
</tbody>
</table>

**Graph and Matrix**

- **Matrix A**
  
<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>.3</td>
<td>0</td>
<td>.4</td>
<td>0</td>
<td>.2</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>.1</td>
<td>0</td>
<td>.5</td>
<td>0</td>
<td>.1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>.5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>.8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>.1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>.7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>.8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- **Distance d**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>.3</td>
<td>0</td>
<td>.8</td>
<td>0</td>
<td>.5</td>
<td>0</td>
</tr>
</tbody>
</table>

- **Result of d min. + A**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>.3</td>
<td>0</td>
<td>.8</td>
<td>0</td>
<td>.5</td>
<td>0</td>
</tr>
</tbody>
</table>
SSSP – ALGEBRAIC BELLMAN-FORD

<table>
<thead>
<tr>
<th>semiring</th>
<th>set</th>
<th>⊕</th>
<th>⊗</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>min-plus</td>
<td>$a \in \mathbb{R} \cup {+\infty}$</td>
<td>min</td>
<td>+</td>
<td>+∞</td>
</tr>
</tbody>
</table>

**Diagram:**

- Nodes: 1, 2, 3, 4, 5, 6, 7
  - **1**: .3
  - **2**: .8
  - **3**: .2
  - **4**: .5
  - **5**: .1
  - **6**: .4
  - **7**: .8

**Matrix A:**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>.3</td>
<td>.8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>.8</td>
<td>0</td>
<td>.1</td>
<td>.7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>.2</td>
<td>.4</td>
<td>0</td>
<td>.5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>.5</td>
<td>.9</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>.1</td>
<td>.5</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>.1</td>
<td>.5</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>.1</td>
<td>.5</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Matrix d:**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>.3</td>
<td>1.2</td>
<td>.8</td>
<td>.4</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>.5</td>
<td>.1</td>
<td>.8</td>
<td></td>
<td>+∞</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>.1</td>
<td>.5</td>
<td>.9</td>
<td></td>
<td>+∞</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>.1</td>
<td>.5</td>
<td>.9</td>
<td></td>
<td>+∞</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>.1</td>
<td>.5</td>
<td>.9</td>
<td></td>
<td>+∞</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>.1</td>
<td>.5</td>
<td>.9</td>
<td></td>
<td>+∞</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>.1</td>
<td>.5</td>
<td>.9</td>
<td></td>
<td>+∞</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$d_{\text{min.}+A}$

$$d_{\text{min.}+A}$$
## SSSP – ALGEBRAIC BELLMAN-FORD

<table>
<thead>
<tr>
<th>semiring</th>
<th>set</th>
<th>⊕</th>
<th>⊗</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>min-plus</td>
<td>$a \in \mathbb{R} \cup {+\infty}$</td>
<td>min</td>
<td>+</td>
<td>$+\infty$</td>
</tr>
</tbody>
</table>

### Example Graph

- **Adjacency Matrix** $A$
  - $A_{ij}$ represents the weight of the edge from vertex $i$ to vertex $j$.
  - For example, $A_{12} = 0.3$.

### Example Table

- **Distance Matrix** $d$
  - $d_{ij}$ represents the shortest path weight from vertex $i$ to vertex $j$.
  - For example, $d_{12} = 0.3$.

- **Augmented Matrix** $A$
  - $A_{ij}$ includes the weight of the edge from vertex $i$ to vertex $j$.
  - For example, $A_{12} = 0.3$.

### Example Diagram

- A directed graph with vertices 1, 2, 3, 4, 5, 6, 7.
- Edges with weights: $0.3$, $0.8$, $0.7$, $0.1$, $0.5$, $0.8$, $0.1$.
- The shortest path from vertex 1 to vertex 2 is via vertex 3 with a total weight of $0.3 + 0.1 = 0.4$. 

---

**min-plus semiring** is a semiring where the operation $\oplus$ is the minimum function, and the operation $\otimes$ is the plus function. This is often used in graph theory to find the shortest path in a weighted graph.

**Adjacency Matrix** $A$ is a matrix that represents a graph, where $A_{ij}$ is the weight of the edge between vertices $i$ and $j$.

**Distance Matrix** $d$ is a matrix that represents the shortest path weights from each vertex to every other vertex.

**Augmented Matrix** $A$ includes both the adjacency matrix and the distance matrix, which is used in the Bellman-Ford algorithm to find the shortest paths.
SSSP – ALGEBRAIC BELLMAN-FORD ALGO.

**Input:** adjacency matrix $A$, source vertex $s$, #vertices $n$

$$A_{ij} = \begin{cases} 
0 & \text{if } i = j \\
\omega(e_{ij}) & \text{if } e_{ij} \in E \\
\infty & \text{if } e_{ij} \notin E
\end{cases}$$

**Output:** distance vector $d$ (real)

1. $d = [\infty \infty \ldots \infty]$
2. $d(s) = 0$
3. for $k = 1$ to $n - 1$  \quad *terminate earlier if we reach a fixed point*
4. $d = d \min + A$

Optimizations for BFS (push/pull) also work here.
Graph algorithms in GraphBLAS

PageRank
PAGERANK – DEFINITION (LDBC GRAPHALYTICS)

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

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

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

$\alpha$: damping factor

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

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

Initially:

\[
\mathbf{pr}_0 = [1 1 \ldots 1] \oslash n, \quad \text{outd} = [\bigoplus_j \mathbf{A}(\cdot, j)]
\]

In each iteration:

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

constant \hspace{1cm} \text{SpMV} \hspace{1cm} \text{element-wise sparse vector-dense vector multiplication}
PAGERANK – ALGORITHM

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

Output: PageRank $\mathbf{pr}$ (real)

Workspace: outdegrees $\mathbf{outd}$

1. $\mathbf{pr} = [1\ 1\ \ldots\ 1] \odot n$
2. $\mathbf{outd} = [\bigoplus_j \mathbf{A}(,j)]$
3. for $k = 1$ to $t$
4. $\mathbf{pr} = \frac{1 - \alpha}{n} \bigoplus \alpha \otimes \left( \frac{\mathbf{pr}}{\mathbf{outd}} \right) \bigoplus \mathbf{A} \bigoplus \frac{\alpha}{n} \otimes \left[ \bigoplus_i (\mathbf{pr} \otimes \mathbf{outd})_i \right]$

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

Local clustering coefficient
**LCC: LOCAL CLUSTERING COEFFICIENT**

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

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

Important metric in social network analysis.

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

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

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

The difficult part is \(\text{tri}(v)\).
LCC EXAMPLE: NAÏVE APPROACH

\[
\text{tr} = \text{diag}^{-1}(A \oplus \otimes A \oplus \otimes A)
\]
LCC: NUMBER OF TRIANGLES IN EACH VERTEX

Observation: Matrix $A \oplus \otimes A \oplus \otimes A$ is not sparse.

Optimization: Use element-wise multiplication $\otimes$ to close wedges into triangles:

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

Then, perform a row-wise summation to get the number of triangles (counted twice) in each vertex:

$$\text{tri2} = \left[ \oplus_j \text{TRI}(:,j) \right]$$
LCC EXAMPLE: ELEMENT-WISE MULTIPLICATION

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

\[ \text{tri2} = [\oplus_j \text{TRI}(::, j)] \]
LCC EXAMPLE: ELEMENT-WISE MULTIPLICATION

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

\[ \text{TRI}(A) = A \oplus \odot A \]
\[ \text{tri2} = [\oplus_j \text{TRI}(:, j)] \]
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:
  \[
  \text{perm}_2(x) = x \cdot (x - 1)
  \]

- Given the degrees \( \text{deg} = [\bigoplus_j A(:, j)] \), we compute \( \text{wed} \) by applying a unary function on the elements of the vector:
  \[
  \text{wed} = \text{perm}_2(\text{deg})
  \]
LCC EXAMPLE: NUMBER OF WEDGES

\[
\begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 \\
1 & 1 & & & & & \\
2 & 1 & 1 & 1 & & & \\
3 & & 1 & 1 & 1 & & \\
4 & 1 & 1 & 1 & 1 & 1 & \\
5 & & & & 1 & 1 & 1 \\
6 & 1 & 1 & 1 & & & \\
7 & 1 & 1 & 1 & 1 & & \\
\end{array}
\]
LCC EXAMPLE: COMPLETE ALGORITHM

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

\[ \text{deg} \]

\[ \text{perm2}(\cdots) \]

\[ \text{tri2} \]

\[ \text{wed} \]

\[ \text{lcc} \]

\[ = \]

\[ \begin{array}{cccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\end{array} \]
LCC: ALGORITHM

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

1. $TRI(A) = A \oplus A \circ A$ compute triangle count matrix
2. $trii = \bigoplus_j TRI(:, j)$ reduce to triangle count vector
3. $deg = \bigoplus_j A(:, j)$ reduce to vertex degree vector
4. $wed = perm2(deg)$ apply perm2 to get wedge count vector
5. $lcc = trii \odot wed$ LCC vector

Further optimization: use $\mathbf{L}$, the lower triangular part of $\mathbf{A}$.

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

The number of wedges is now the 2-combination of $\text{deg}$. \[
\text{comb}_2(x) = \frac{x \cdot (x - 1)}{2}
\]
Permuting the adjacency matrix allows further optimizations.
LCC EXAMPLE: LOWER TRIANGULAR PART OF MX.

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

\[
\begin{array}{c|c|c|c|c|c|c|c}
\hline
& 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
1 & 1 & 1 & 1 & & & & \\
2 & 1 & 1 & 1 & 1 & & & \\
3 & 1 & 1 & 1 & & & & \\
4 & 1 & 1 & 1 & 1 & & & \\
5 & 1 & 1 & 1 & & & & \\
6 & 1 & 1 & 1 & & & & \\
7 & 1 & 1 & 1 & 1 & 1 & & \\
\hline
\end{array}
\]
Graph algorithms in GraphBLAS

Triangle count / Definition
IEEE GraphChallenge: an annual competition at the HPEC conference

The task of the 2017 GraphChallenge was **triangle count**: given a graph G, count the number of triangles.

**Triangle** = “set of three mutually adjacent vertices in a graph”

Many solutions employed a linear algebraic computation model

Number of unique triangles:

\[
\frac{30}{6}
\]
Graph algorithms in GraphBLAS

Triangle count / Naïve algorithm
**TC EXAMPLE: NAÏVE APPROACH**

\[
\text{tri2} = \text{diag}^{-1}(A \oplus \times A \oplus \times A)
\]
Graph algorithms in GraphBLAS

Triangle count / Masked algorithm
TC EXAMPLE: ELEMENT-WISE MULTIPLICATION

\[ \text{TRI} = A \bigoplus \cdot \otimes A \bigotimes A \]

\[ \text{tri2} = [\bigoplus_j \text{TRI}(:,j)] \]
TC EXAMPLE: ELEMENT-WISE MULTIPLICATION

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

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

$$\text{tri2} = [\oplus_j \text{TRI}(:, j)]$$
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. $B = L \odot U$ 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

U

A

B

C

t = \sum \frac{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. $C \langle L \rangle = L \oplus \otimes L$ 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

\[ \mathbf{C} = \mathbf{L} \oplus \mathbf{L} \]

\[ t = \sum \mathbf{C} \]
Graph algorithms in GraphBLAS

Triangle count / CMU algorithm
CMU ALGORITHM

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

- 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. $t = t + a_{10}^\top \oplus \otimes A_{20} \oplus \otimes a_{12}$

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

M. Lee, T.M. Low (Carnegie Mellon University): *A family of provably correct algorithms for exact triangle counting*, CORRECTNESS @ SC 2017
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

\[
\begin{array}{c|c|c|c|c|c|c|c|}
& 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
A & \begin{array}{cccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\end{array} \\
\hline
\end{array}
\]

\[
\begin{array}{c|c|c|c|c|c|c|c|}
& 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
\text{TRI} & \begin{array}{cccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\
1 & 2 & 3 & 3 & 3 & 3 & 3 & 3 \\
1 & 2 & 2 & 5 & 5 & 5 & 5 & 5 \\
1 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\
1 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\
1 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\
1 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\
\end{array} \\
\hline
\end{array}
\]

\[
\begin{array}{c|c|c|c|c|c|c|c|}
& 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
\text{tri} & \begin{array}{cccc}
1 & 1 & 1 & 1 \\
1 & 2 & 2 & 2 \\
1 & 2 & 2 & 2 \\
1 & 2 & 2 & 2 \\
\end{array} \\
\hline
\end{array}
\]

\[
\text{TRI} = A \oplus A \otimes A \\
\text{tri} = [\oplus_j \text{TRI}(i,j)]/2
\]

A \oplus A \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 $A \oplus \otimes A$.

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

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

Input: adjacency matrix $A$
Output: vector $tri$
Workspace: matrix $TRI$

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

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

$$TRI(A) = A \oplus \odot L$$

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.

- [TODO]: example – the example graph does not have any $4$-trusses, another example is required.
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. $\mathbf{C} \langle \mathbf{C} \rangle = \mathbf{C} \oplus \land \mathbf{C}$ \hspace{1em} use the “plus-and” semiringing
4. $\mathbf{C} = f(\mathbf{C}, k - 2)$ \hspace{1em} drop entries in $\mathbf{C}$ 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
CDLP: 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:

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

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
  - $\{6 \to 1, 9 \to 1\} \oplus \{6 \to 1, 7 \to 2\} = \{6 \to 2, 7 \to 2, 9 \to 1\}$
- Convert each element in a row to an occurrence vector
  - $\{6 \to 1\}, \{6 \to 1\}, \{7 \to 1\}, \{7 \to 1\}, \{9 \to 1\}$
- Reduce each row into a single occurrence vector:
  - $\{6 \to 2, 7 \to 2, 9 \to 1\}$
- 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 $\mathbf{F}$
  - 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 $\mathbf{r}$
- Use the sorted list to compute $\text{mode}(\mathbf{r})$
Initially, \( \text{lab} = [1 \ 2 \ \ldots \ n] \)

- Propagate labels to create a “frequency matrix”:
  \[
  F = A \text{any.sel2nd} \text{diag(\text{lab})}
  \]
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 $lab$
Workspace: matrix $F$, vector $r$

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

Can be batched and parallelized
CDLP: ALGORITHM

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

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

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

- 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:
  - $\langle I_{in}, \_X_{in} \rangle = \text{extractuples}(F_{in})$
  - $\langle I_{out}, \_X_{out} \rangle = \text{extractuples}(F_{out})$
  - $\{\text{merge_sort_pairs}((I_{in} \cup I_{out}, X_{in} \cup X_{out}))\}$
Graph algorithms in GraphBLAS

The importance of masking
THE IMPORTANCE OF MASKING

**Q:** Is masking absolutely necessary?

**A:** Yes, it can reduce the complexity of some algorithms. We demonstrate this with two examples.
A simple corner case is the star graph: there are \((n - 1)^2\) wedges but none of them close into triangles.

We do quadratic work while it’s clear that there are no triangles in the graph (it’s a tree).
A full bipartite graph $K_{2,3}$ with the vertices in the top partition connected.

A bipartite graph only has cycles of even length, so it’s easy to see that all triangles will contain the two vertices in the top partition. Still, $A \oplus \times A$ enumerates all wedges starting and ending in the bottom partition, thus performing a lot of unnecessary work.
Masking avoids the materialization of large interim data sets by ensuring that we only enumerate wedges whose endpoints are already connected.
Graph algorithms in GraphBLAS

Summary and other algorithms
## SUMMARY OF THE ALGORITHMS DISCUSSED

<table>
<thead>
<tr>
<th>Algo.</th>
<th>Features demonstrated</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFS</td>
<td>lor-land and min-sel1st semirings, masked SpMV</td>
</tr>
<tr>
<td>SSSP</td>
<td>min-plus semiring</td>
</tr>
<tr>
<td>PR</td>
<td>real semiring, row-wise reduce</td>
</tr>
<tr>
<td>LCC</td>
<td>apply operation, masked matrix-market multiplication</td>
</tr>
</tbody>
</table>
GraphBLAS primitives in increasing arithmetic intensity

- **Misc.**
  - Connectivity, subgraph matching, traversal (BFS), max. independent set

- **Centrality**
  - PageRank, betweenness centrality

- **Graph clustering**
  - Markov cluster, peer pressure, spectral, local

- **Shortest paths**
  - All-pairs shortest, single-source, temporal

- **SpMSpV**
  - Sparse Matrix Sparse Vector
- **SpMV**
  - Sparse Matrix Dense Vector
- **SpMM**
  - Sparse Matrix Multiple Dense Vectors
- **SpGEMM**
  - Sparse Matrix Sparse Matrix
- **SpDM^3**
  - Sparse-Dense Matrix-Matrix M.

Based on the figure in A. Buluç: *Graph algorithms, computational motifs, and GraphBLAS*, ECP Meeting 2018
OTHER ALGORITHMS IN GRAPHBLAS

**Betweenness centrality:** Brandes’ algorithm

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

**k-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 processing in relational algebra
RELATIONAL ALGEBRA ON GRAPHS

Natural join: \( A \bowtie B \)

A natural join connects edges from \( A \) with an edge in \( B \).

Semijoin: \( A \bowtie B \)

A semijoin keeps edges from \( A \) with a connecting edge in \( B \).

Antijoin: \( A \bar{\bowtie} B \)

An antijoin keeps edges from \( A \) without a connecting edge in \( B \).

Left outer join: \( A \bowtie B \)

A left outer join keeps all edges from \( A \) with or without a connecting edge in \( B \).
Given relations $A(i, j), B(j, k), C(i, k)$ triangles can be enumerated with:

$$A \bowtie B \bowtie C$$

Wedges that do not close into a triangle

*Edges are stored in one direction: $R[1] < R[2]"
For $m$ edges, the AGM-bound of the triangle query is $O(m^{3/2})$

**Theorem #1:**
Any binary join plan for the triangle query takes $\Omega(m^2)$ time.

To mitigate this, worst-case optimal join (WCOJ) algorithm use multi-way joins such as $\bowtie (A, B, C)$.

**Theorem #2:**
The worst-case complexity of the generic join algorithm (a WCOJ algorithm) $\leq$ AGM bound.


We only use the lower triangular part.
$\text{TRI}(\mathbf{C}) = \mathbf{A} \oplus \mathbf{B}$

*We only use the lower triangular part.*
Vertex-wise triangle count in linear algebra:

\[
\text{TRI}(C) = A \bigoplus \bigotimes B
\]
\[
\text{tri} = \left[ \bigoplus_j \text{TRI}(, j) \right]
\]

Notice that in relational algebra, given \( A(i, j), B(j, k), C(i, k) \):

\[
A \bowtie B \bowtie C \equiv (A \bowtie B) \bowtie C
\]
\[
A \bowtie B \bowtie C \equiv (B \bowtie C) \bowtie A \equiv (B \bowtie C) \bowtie A
\]
\[
A \bowtie B \bowtie C \equiv (A \bowtie C) \bowtie B \equiv (A \bowtie C) \bowtie B
\]

Where the semijoin operator \( \bowtie \) is defined as:

\[
R \bowtie S = \pi_{\text{schema}(R)}(R \bowtie S) = R \bowtie \pi_{\text{schema}(R) \cap \text{schema}(S)}(S)
\]
Given relational schemas $A(i, j), B(j, k), C(i, k)$ and expression:

$$A \bowtie B \bowtie C \equiv (A \bowtie B) \bowtie C$$

The “masking” technique cannot be translated to relational algebra as a single operation – we have to use an outside loop to express it. Using relation $T(i, k)$:

1. for all $c \in C$ do
2. $\text{TRI} = \text{TRI} \cup \pi_{i,k}((A \bowtie \{c\}) \bowtie (B \bowtie \{c\}))$
3. $\tri = \gamma_{i,\text{count}(k)\rightarrow \text{triCount}}(\text{TRI})$

Cf. $\text{TRI}(C) = A \ominus \otimes B$, $\tri = [\oplus_j \text{TRI}(:, j)]$

\[
\begin{array}{c|c}
  i & \text{triCount} \\
  \hline
  1 & 5 \\
  2 & 1 \\
  3 & 1 \\
\end{array}
\]
Relational algebra also allows triangle enumeration.

Given $A(i,j), B(j,k), C(i,k)$, instead of the previous expression:

1. for all $c \in C$ do
2. $\tri = \tri \cup \pi_{i,k}(A \bowtie \{c\} \bowtie (B \bowtie \{c\}))$
3. $\tri = \gamma_{i,\text{count}(k)\rightarrow\text{triCount}}(\tri)$

Compute:

1. for all $c \in C$ do
2. $\tri = \tri \cup ((A \bowtie \{c\}) \bowtie (B \bowtie \{c\}))$
TRIANGLE ENUMERATION IN RELATIONAL ALG.

1. for all $c \in C$ do

2. $TRI = TRI \cup ((A \bowtie \{c\}) \bowtie (B \bowtie \{c\}))$

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$C$</td>
<td>$i$</td>
<td>$k$</td>
<td>$A$</td>
<td>$i$</td>
<td>$j$</td>
<td>$B$</td>
<td>$j$</td>
<td>$k$</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td></td>
<td>1</td>
<td>4</td>
<td></td>
<td>4</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td></td>
<td>2</td>
<td>4</td>
<td></td>
<td>4</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td></td>
<td>3</td>
<td>4</td>
<td></td>
<td>4</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td></td>
<td>1</td>
<td>5</td>
<td></td>
<td>5</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td></td>
<td>1</td>
<td>6</td>
<td></td>
<td>6</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

$A \bowtie \{c\}$  $B \bowtie \{c\}$

$A \bowtie B \bowtie C$
TRIANGLE ENUMERATION IN RELATIONAL ALG.

1. for all \( c \in C \) do

2. \( TRI = TRI \cup ((A \bowtie \{c\}) \bowtie (B \bowtie \{c\})) \)

\[
\begin{array}{c}
\begin{array}{c|c}
C & \begin{array}{c|c|c}
i & k \\
1 & 7 \\
1 & 8 \\
1 & 9 \\
2 & 7 \\
3 & 7 \\
\end{array}
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c|c|c}
A & \begin{array}{c|c|c}
i & j \\
1 & 4 \\
2 & 4 \\
3 & 4 \\
1 & 5 \\
1 & 6 \\
\end{array}
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c|c|c}
B & \begin{array}{c|c|c}
i & j \\
4 & 7 \\
4 & 8 \\
4 & 9 \\
5 & 7 \\
6 & 7 \\
\end{array}
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c|c|c|c}
A \bowtie B \bowtie C & \begin{array}{c|c|c|c}
i & j & k \\
1 & 4 & 7 \\
1 & 5 & 7 \\
1 & 6 & 7 \\
1 & 4 & 8 \\
1 & 4 & 9 \\
2 & 4 & 7 \\
3 & 4 & 7 \\
\end{array}
\end{array}
\end{array}
\end{array}
\]

\( A \bowtie \{c\} \)  \( B \bowtie \{c\} \)
1. for all $c \in C$ do

2. $TRI = TRI \cup ((A \bowtie \{c\}) \bowtie (B \bowtie \{c\}))$
TRIANGLE ENUMERATION IN RELATIONAL ALG.

1. for all $c \in C$ do

2. $TRI = TRI \cup \left( (A \bowtie \{c\}) \bowtie (B \bowtie \{c\}) \right)$
1. for all $c \in C$ do

2. $\text{TRI} = \text{TRI} \cup ((A \bowtie \{c\}) \bowtie (B \bowtie \{c\}))$
TRIANGLE ENUMERATION IN RELATIONAL ALG.

1. for all $c \in C$ do

2. $TRI = TRI \cup ((A \bowtie \{c\}) \bowtie (B \bowtie \{c\}))$

Using semijoin pushdown on each tuple, we computed $A \bowtie B \bowtie C$ without enumerating a quadratic-size interim relation. However, for efficient implementations, relations should also be sorted (see ref. for details).

Triangle enumeration is possible by using triples as matrix elements and overloading “⊕, ⊗” so that multiplying the adjacency matrix $A$ with the incidence matrix $B$ results in $C$:

$$C_{ij} = (i, x, y) \text{ iff } A_{ix} = A_{iy} = 1 \text{ and } B_{xj} = B_{yj} = 1$$

$$A = \begin{pmatrix}
0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 & 1 \\
1 & 1 & 1 & 0 & 0
\end{pmatrix}, \quad B = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 \\
0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 \\
1 & 1 & 1 & 0 & 0
\end{pmatrix}, \quad C = \begin{pmatrix}
\emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\
\emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\
\emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\
\emptyset & (4, 2, 5) & (4, 3, 5) & \emptyset & \emptyset \\
\emptyset & \emptyset & \emptyset & (5, 3, 4) & (5, 2, 4)
\end{pmatrix}$$
MATRICES MULTIPLICATION IN RELATIONAL ALG.

<table>
<thead>
<tr>
<th>set</th>
<th>⊕</th>
<th>⊗</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>set relations</td>
<td>$R \subseteq D_1 \times \cdots \times D_k$</td>
<td>∪</td>
<td>⊗</td>
</tr>
<tr>
<td>matrix relations</td>
<td>$R \subseteq D_1 \times D_2$</td>
<td>∪</td>
<td>⊗</td>
</tr>
</tbody>
</table>

Observations:
- a binary relation is analogous to a Boolean sparse matrix stored in coordinate list (COO) format
- a matrix multiplication is analogous to an equijoin on a binary relation that uses a single join attribute, and discards it after performing the join.

**Definition:** Given two binary relations $R, S$ with a common attribute $x$, we define “multjoin” as:

$$R \Join S = \pi_{R \cup S \setminus \{x\}}\left(\left[R \Join_{R.x=S.x} S\right]\right)$$

**Example:**

<table>
<thead>
<tr>
<th>$i$</th>
<th>$j$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>$i$</td>
<td>$k$</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
</tr>
</tbody>
</table>
**MM VS. MULTJOIN: REACHABILITY**

**Matrices:**
\[ A \in \mathbb{B}^{m \times n}, \quad B \in \mathbb{B}^{n \times o}, \quad C \in \mathbb{B}^{o \times p} \]

For \( A \lor \land B \):
\[ \langle AB \rangle_{ik} = \lor_k (A_{ij} \land B_{jk}) \]

For \( A \lor \land B \lor \land C \):
\[ \langle ABC \rangle_{il} = \]
\[ = \lor_k (\langle AB \rangle_{ik} \land C_{kl}) \]
\[ = \lor_k \left( \lor_j \left( A_{ij} \land B_{jk} \right) \right) \land C_{kl} \]
\[ = \lor_k \left( \lor_j \left( A_{ij} \land B_{jk} \land C_{kl} \right) \right) \]

**Relations:**
\[ A(i, j), \ B(j, k), \ C(k, l) \]
\[ A \Join B \quad \text{sch: } (i, k) \]
\[ A \Join B \Join C \quad \text{sch: } (i, l) \]
MM VS. JOIN: COUNTING PATHS

Matrices:
\[ A \in \mathbb{N}^{m \times n}, \quad B \in \mathbb{N}^{n \times o}, \quad C \in \mathbb{N}^{o \times p} \]

\[
\langle AB \rangle_{ik} = \sum_j A_{ij} \otimes B_{jk}
\]

\[
\langle ABC \rangle_{il} = \sum_k \left( \sum_j A_{ij} \otimes B_{jk} \right) \otimes C_{kl}
\]

Relations:
\[ A(i, j, va), \quad B(j, k, vb), \quad C(k, l, vc) \]

\[
\gamma^{A.i,B.k}_{\text{sum}(A_{va} \cdot B_{vb})}(A \bowtie B)
\]

\[
\gamma^{A.i,C.l}_{\text{sum}(A_{va} \cdot B_{vb} \cdot C_{vc})}(A \bowtie B \bowtie C)
\]

Enumerating triangles requires applying \( \sigma_{A.i=C.l}(\ldots) \)
MATRIX MULTIPLICATION IN SQL: PATHS

$$\langle \mathbf{AB} \rangle_{ik} = \sum_j A_{ij} \otimes B_{jk} \Rightarrow \gamma_{\Sigma(A_{va} \cdot B_{vb})}^{A.i,B.k}(A \Join B)$$

```
select A.i, B.k, sum(A.va * B.vb)
from A, B
where A.j = B.j
group by A.i, B.k;
```

$$\langle \mathbf{ABC} \rangle_{il} = \sum_k (\sum_j A_{ij} \otimes B_{jk} \otimes C_{kl}) \Rightarrow \gamma_{\Sigma(A_{va} \cdot B_{vb} \cdot C_{vc})}^{A.i,C.l}(A \Join B \Join C)$$

```
select A.i, C.l, sum(A.va * B.vb * C.vc)
from A, B, C
where A.j = B.j
    and B.k = C.k
group by A.i, C.l;
```
MATRIX MULTIPLICATION IN SQL: TRIANGLES

Relations \(A(i, j, va), B(j, k, vb), C(k, l, vc)\)

\[
\langle \text{ABC} \rangle_{ii} = \sum_k \left( \sum_j A_{ij} \otimes B_{jk} \otimes C_{kl} \right) \Rightarrow \gamma_{A.i,C.l} \left( \sigma_{A.i=C.l} (A \bowtie B \bowtie C) \right)
\]

select A.i, C.l, sum(A.va * B.vb * C.vc)
from A, B, C
where A.j = B.j
  and B.k = C.k
  and A.i = C.l
group by A.i, C.l;

Given relations \(A(i, j, va), B(j, k, vb), C(i, k, vc)\), we can simply use two natural joins: \(A \bowtie B \bowtie C\).
GraphBLAS and SuiteSparse internals
“A crucial piece of the GraphBLAS effort is to translate the mathematical specification to an API that
- is faithful to the mathematics as much as possible, and
- enables efficient implementations on modern hardware.”

\[ C \leftarrow M \odot = \oplus \cdot \otimes (A^T, B^T) \]

mxm(Matrix *C, Matrix M, BinaryOp accum, Semiring op, Matrix A, Matrix B, Descriptor desc)

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

- GraphBLAS objects are opaque: the matrix representation can be adjusted to suit the data distribution, hardware, etc.
- The typical representations are compressed formats are:
  - CSR: Compressed Sparse Row (also known as CRS)
  - CSC: Compressed Sparse Column (also known as CCS)

**CSR representation of \( \mathbf{A} \):**

<table>
<thead>
<tr>
<th>Value</th>
<th>Row ptr</th>
<th>Col index</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0.3, 0.8]</td>
<td>1, 3, 5, 6, 8, 9, 10, 13</td>
<td>2, 4, 5, 7, 6, 1, 3, 6, 3, 3, 4, 5</td>
<td>[0.3, 0.8, 0.1, 0.7, 0.5, 0.2, 0.4, 0.1, 0.5, 0.1, 0.5, 0.9]</td>
</tr>
</tbody>
</table>
SUITESPARSE:GRAPHBLAS INTERNALS

- Authored by Prof. Tim Davis at Texas A&M University, based on his SuiteSparse library (used in MATLAB).
- Design decisions, algorithms and data structures are discussed in the TOMS paper and in the User Guide.
- Extensions: methods and types prefixed with GxB.
- Sophisticated load balancer for multi-threaded execution.
- GPU implementation is a work-in-progress.

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

T.A. Davis: SuiteSparse:GraphBLAS: graph algorithms via sparse matrix operations on semirings, Sparse Days 2017
Graph benchmarks
Standard specifications for benchmarking certain aspects of relational DBs

TPC price/perf trend 1990-2005: improved 58% per year, prices have declined 37%/y

TPC-A
price/perf trend 37% per year

TPC-C
price/perf trend 37% per year

TPC-C price/perf trend 37% per year
LDBC: LINKED DATA BENCHMARK COUNCIL

- A non-profit academic & industrial organization
- Dedicated to define standard benchmarks for *graph databases*

**EU FP7 project — Community meetings — Benchmark papers**


Social Network Benchmark
Interactive workload
SIGMOD 2015
Erling et al.

Graphalytics
VLDB 2016, Iosup et al.

SNB
Business Intelligence
GRADES@SIGMOD 2018
Szárnyas et al.
GRAPH PROCESSING CATEGORIES IN LDBC

- **OLAP: SNB BI**
  - Graph analytics: structure only
  - \( \approx O(n + e) \)

- **OLTP: SNB Int.**
  - Graph queries: structure, types, and properties
  - \( \approx O(d \cdot \log n) \)

- **Graphalytics**
  - \( \approx O(n \log n) \)
PRELIMINARY RESULTS WITH SUITEVERSE:GB

Ubuntu Server, 512 GB RAM, 64 CPU cores, 128 threads

Twitter 50M vertices, 2B edges

graph500-26 33M vertices, 1.1B edges

d-8.8-zf 168M vertices, 413M edges

d-8.6-fb 6M vertices, 422M edges

loading often takes longer than processing
THE GAP BENCHMARK SUITE

- Part of the Berkeley Graph Algorithm Platform project
- Same authors as the direction-optimizing BFS algorithm
- Six algorithms: BFS, SSSP, PageRank, connected components, betweenness centrality, triangle count
- Five data sets: Twitter, Web, Road, Kron, Urand
  - 24M to 134M vertices, 68M to 2.1B edges
  - Weighted/unweighted, directed/undirected
- Very efficient baseline implementation in C++


gap.cs.berkeley.edu/benchmark.html
Further reading and libraries
RESOURCES

Presentations and tutorials for learning GraphBLAS along with a link to a collection of pointers.

J.R. Gilbert:
*GraphBLAS: Graph Algorithms in the Language of Linear Algebra,* Seminar talk since 2014

S. McMillan and T.G. Mattson:
*A Hands-On Introduction to the GraphBLAS,* Tutorial at HPEC since 2018

A. Buluç:
*GraphBLAS: Concepts, algorithms, and applications,* Scheduling Workshop, 2019

M. Kumar, J.E. Moreira, P. Pattnaik:
*GraphBLAS: Handling performance concerns in large graph analytics,* Computing Frontiers 2018

List of GraphBLAS-related books, papers, presentations, posters, software, etc. [szarnyasg/graphblas-pointers](https://szarnyasg/graphblas-pointers)
THE LAGRAPH LIBRARY

- Similar to the LAPACK library for BLAS
- Uses SuiteSparse:GraphBLAS
- Implementations of common algorithms
  - BFS, SSSP, LCC, PageRank, Boruvka
  - Triangle count, $k$-truss
  - CDLP (community detection using label propagation)
  - Weakly connected components, Strongly Conn. Comps (draft)
  - Betweenness centrality
  - Deep neural network

T.G. Mattson et al.: LAGraph: A Community Effort to Collect Graph Algorithms Built on Top of the GraphBLAS, GrAPL@IPDPS 2019
REQUIREMENTS BY GRAPH COMPUTATIONS

Libraries for linear-algebra based graph processing support the following features (prioritized):

1. Sparse matrices  For reasonable performance
2. Arbitrary semirings  For expressive power
3. Masking  A big reduction in complexity for some algos
4. Parallel execution  Constant speedup, ideally by #threads

Most matrix libraries only satisfy requirement #1:

- Intel MKL, Eigen, Boost uBLAS, MTL4, Armadillo, NIST Sparse BLAS, GMM++, CUSP (Thrust), Numpy, Efficient Java Matrix Library (EJML)
- The only exception is Julia’s SparseArrays library, where arbitrary semirings can be used (#2) due to Julia’s sophisticated type system
GRAPHBLAS PAPERS AND BOOKS

- **Standards for Graph Algorithm Primitives**
  - Position paper by 19 authors @ IEEE HPEC 2013

- **Novel Algebras for Advanced Analytics in Julia**
  - Technical paper on semirings in Julia @ IEEE HPEC 2013

- **Mathematical Foundations of the GraphBLAS**
  - Theory paper by 16 authors @ IEEE HPEC 2016

- **Design of the GraphBLAS C API**
  - Design decisions and overview of the C API @ GABB@IPDPS 2017

- **Algorithm 1000: SuiteSparse:GraphBLAS: graph algorithms in the language of sparse linear algebra**
  - Algorithms in the SuiteSparse implementation @ ACM TOMS 2019
**BOOKS**

- **Graph Algorithms in the Language of Linear Algebra**
  - Algorithms for connected components, shortest paths, max-flow, betweenness centrality, spanning tree, graph generation, etc.
  - Algorithms and data structure for fast matrix multiplication
  - Predates GraphBLAS: preliminary notation, no API usage

- **Mathematics of Big Data**
  - Authored by Jananthan & Kepner, published by MIT Press in 2018
  - Generalizes the semiring-based approach for associative arrays
  - Contains reprints of papers, including the HPEC’16 paper above
  - Discusses D4M (Dynamic Distributed Dimensional Data Model)
GRAPHBLAS COMMUNITY

Wiki: graphblas.org | Communication: mailing list, monthly telcos

Annual events:

▪ May: IEEE IPDPS conference
  o GrAPL workshop (Graphs, Architectures, Programming and Learning), a merger of
    • GABB (Graph Algorithms Building Blocks)
    • GraML (Graph Algorithms and Machine Learning)
  o See graphanalysis.org for previous editions

▪ Sep: IEEE HPEC conference
  o GraphBLAS BoF meeting

▪ Nov: IEEE/ACM Supercomputing conference
  o GraphBLAS Working Group
  o IA³ workshop (Workshop on Irregular Applications: Architectures and Algorithms)

Blog: AldenMath by Timothy Alden Davis
REDISGRAPH

- Graph database built on top of Redis with partial (but extending) support for the Cypher language
- Uses SuiteSparse:GraphBLAS for graph operations
- Preliminary benchmark results show good performance on traversal-heavy workloads

R. Lipman, T.A. Davis: 
*Graph Algebra – Graph operations in the language of linear algebra* 
RedisConf 2018

R. Lipman: 
*RedisGraph internals* 
RedisConf 2019
GRAPHBLAS IMPLEMENTATIONS

- SuiteSparse:GraphBLAS
  - v1.0.0: Nov 2017 – sequential
  - v3.0.1: July 2019 – parallel
  - v3.1.0: Oct 2019 – supports MATLAB bindings

- IBM GraphBLAS
  - Complete implementation in C++, released in May 2018
  - Concise but sequential

- GBTL (GraphBLAS Template Library): C++
  - v1.0: parallel but no longer maintained
  - v2.0: sequential

- GraphBLAST: GPU implementation, based on GBTL
GRAPHULO

- Build on top of the Accumulo distributed key-value store
- Written in Java
- Focus on scalability

V. Gadepally et al.: Graphulo: Linear Algebra Graph Kernels for NoSQL Databases, GABB@IPDPS 2015
COMBBLAS: COMBINATORIAL BLAS

- “an extensible distributed memory parallel graph library offering a small but powerful set of linear algebra primitives”
- Not a GraphBLAS implementation but serves as an incubator for new ideas that may later find their way into GraphBLAS
- Scales to 250k+ CPU cores
  - Used on supercomputers such as Cray

Learning GraphBLAS
LEARNING GRAPHBLAS

- Challenging to get started – need to have an understanding of:
  - Linear algebra (semirings, matrix multiplication, etc.)
  - Connection between linear algebra and graph operations (intuition)
  - Sizeable C API

- 200+ slides presenting algorithms in GraphBLAS
  - BFS variants, PageRank, Triangle count, \( k \)-truss
  - Community detection using label propagation
  - Luby’s maximal independent set algorithm
  - computing connected components on an overlay graph
  - etc.
PYGRAPHBLAS: PYTHON WRAPPER #1

- A Python wrapper
- See example code for SSSP and triangle count
- Close to pseudo-code
- Jupyter support

Added benefit: both Python and GraphBLAS use 0-based indexing

```python
def sssp(matrix, start):
    v = Vector.from_type(matrix.gb_type, matrix.nrows)
    v[start] = 0

    with min_plus, Accum(min_int64):
        for _ in range(matrix.nrows):
            w = Vector.dup(v)
            v @= matrix
            if w == v:
                break
    return v

def sandia(A, L):
    return L.mxm(L, mask=L).reduce_int()

sandia(M, M.tril())
```

michelp/pygraphblas
GRBLAS: PYTHON WRAPPER #2

- Goal: one-to-one mapping of the GraphBLAS API
  - less Pythonic
  - comes with a Conda package
  - compiles user-defined functions to C

jim22k/grblas
TRADEOFFS OF GRAPHBLAS

- Updating CSR/CSC matrices is slow
  - SuiteSparse mitigates this to some extent by allowing incremental build before using the matrix but it still needs to rebuild the matrix before major operations such as mxv, mxm
  - But: GraphBLAS objects are opaque and alternative formats can be used
    - Hornet [HPEC’18]
    - famGraph [SC’18]
    - STINGER [HPEC’12]
Summary
VENUES ON GRAPH PROCESSING TECHNIQUES

- Database systems: SIGMOD, VLDB, ICDE, EDBT, VLDB J., TKDE
- Database theory: PODS, ICDT, TODS
- Information retrieval: KDD, CIKM, WSDM, TKDD, DMKD J., & Big Data
- Operating systems: OSDI, SOSP, NSDI, ICPE, EuroSys
- High-perf. computing: SC, HPEC, HiPC, HPCC, HPDC, IPDPS
- Distributed computing: PODC, DISC, ICDCS
- Parallel programming: SPAA, ICPP, PLDI, PPoPP, TOPC, Euro-Par
- Software engineering: ICSE, MODELS, FSE, ASE, FASE, OOPSLA, SoSyM
- Semantic web: WWW (TheWebConf), ISWC, WWW J., SWJ
- Graph transformation: ICMT/ICGT, CAV, FMCAD, AGTIVE, ENTCS

...and many other mathematics and computer science venues.
SUMMARY

- Linear algebra is a powerful abstraction
  - Good expressive power
  - Concise formulation of most graph algorithms
  - Very good performance
  - Still lots of ongoing research

- Trade-offs:
  - Learning curve (maths, C programming, GraphBLAS API)
  - Some algorithms are difficult to formulate in linear algebra
  - Only a few GraphBLAS implementations (yet)

- Overall: a very promising programming model for graph algorithms suited to the age of heterogeneous hardware
SOME OPEN QUESTIONS

- What primitives should be added to GraphBLAS?
  - Sorting (CDLP)
  - Submatrix selection ($k$-truss)
  - Permutations (DFS)
- How to efficiently express SCC?
- How to implement GraphBLAS in a distributed system?
- Incremental evaluation of SpMM/SpMV operations
  - Use an updatable matrix representation
  - Semirings $\rightarrow$ rings $\rightarrow$ additive inverse $\rightarrow$ addition and removal
  - Incremental view maintenance techniques need rings (?)

D.G. Spampinato et al.: *Linear Algebraic Depth-First Search*, ARRAYS@PLDI 2019

Y. Ahmad et al.: *LA3: A Scalable Link- and Locality-Aware Linear Algebra-Based Graph Analytics System*, VLDB 2018
ACKNOWLEDGEMENTS

- Tim Davis and Tim Mattson for helpful discussions, members of GraphBLAS mailing list for their detailed feedback.
- The LDBC Graphalytics task force for creating the benchmark and assisting in the measurements.
- Master’s students at BME for developing GraphBLAS-based algorithms: Bálint Hegyi, Márton Elekes, Petra Várhegyi, Lehel Boér
“Nuances” – Some important adjustments to the definitions
The GraphBLAS specification defines semirings as follows:

\[ \langle D_{\text{out}}, D_{\text{in}_1}, D_{\text{in}_2}, \oplus, \otimes, 0 \rangle \]

structure is a GraphBLAS semiring defined by

- \( D_{\text{out}}, D_{\text{in}_1}, \) and \( D_{\text{in}_2} \) three domains
- \( \oplus : D_{\text{out}} \times D_{\text{out}} \rightarrow D_{\text{out}} \) an associative and commutative addition operation
- \( \otimes : D_{\text{in}_1} \times D_{\text{in}_2} \rightarrow D_{\text{out}} \) a multiplicative operation
- \( 0 \in D_{\text{out}} \) an identity element for \( \oplus \)

\( A = \langle D_{\text{out}}, \oplus, 0 \rangle \) is a commutative monoid.

\( F = \langle D_{\text{out}}, D_{\text{in}_1}, D_{\text{in}_2}, \otimes \rangle \) is a closed binary operator.

"It is expected that implementations will utilize IEEE-754 floating point arithmetic, which is not strictly associative." (C API specification)
**MATRIX MULTIPLICATION**

The simple formula of matrix multiplication

\[
C = A \oplus \times B
\]

\[
C(i, j) = \bigoplus_j A(i, k) \times B(k, j)
\]

can lead to more complicated definitions. For example, what should the binary operator \( \text{sel1st}(x, y) = x \) return if \( y = 0 \)?

\[
\text{sel1st}(x, y) = \begin{cases} 
0 & \text{if } y = 0 \\
 x & \text{otherwise}
\end{cases}
\]

To simplify definitions, we use a generalized form of multiplication:

\[
C(i, j) = \bigoplus_{k \in \text{ind}(A(i,:)) \cap \text{ind}(B(:,j))} A(i, k) \times B(k, j)
\]

where we only perform the multiplication in the intersection of the non-zero indices of \( A \)'s \( i \)th row and \( B \)'s \( j \)th column.
NOTATION*

Symbols:
- A, B, C, M – matrices
- u, v, w, m – vectors
- s, k – scalar
- i, j – indices
- <m>, <M> – masks

Operators:
- ⊕ – addition
- ⊗ – multiplication
- ⊤ – transpose
- ⊙ – accumulator
- ⊞ – extract submatrix
- ⊘ – assign submatrix with submask for C
- ⊙ – assign submatrix with mask for C
- ⊗ – apply select operator (GxB)
- ⊖ – Kronecker product

This table contains all GrB and GxB (SuiteSparse-specific) operations.
Sparse matrix extraction: induced subgraph

Sparse submatrix assignment: replace subgraph

Sparse matrix selection (SuiteSparse): filtering edges

Kronecker product: graph generation
MATRIX-VECTOR MULTIPLICATION*

The operation $v \oplus.\otimes A$ gives the vertices reachable from the ones in $v$. However, GraphBLAS publications and implementations often use $A^T \oplus.\otimes v$ instead. The difference between these is that the former produces a row vector, while the latter produces a column vector:

$$v \oplus.\otimes A \equiv (A^T \oplus.\otimes v^T)^T$$

GraphBLAS does not distinguish row/column vectors, therefore the notations are (formally) equivalent:

$$v \oplus.\otimes A \equiv A^T \oplus.\otimes v$$

In practice, the order of the operands can have a significant impact on performance. For example, if matrices are stored in CSR format, computing $v \oplus.\otimes A$ is expensive compared to $A^T \oplus.\otimes v$. 
Element-wise subtraction can be defined as an element-wise addition on the INT64_MINUS monoid. It has the following semantics $C = A \ominus B$ is computed on the union of the patterns of the input matrices $A$ and $B$.

For cells where only one input matrix has a non-zero value but the other does not (e.g. $B[0,0] = 1$ but $A[0,0]$ is empty), the result is the non-zero value: $C[0,0] = 1$.

\[
\begin{array}{c}
A & \ominus & B \\
\begin{array}{ccc}
1 & & \\
& & \\
1 & 1 & \\
1 & 1 & \\
\end{array} & = & \\
\begin{array}{ccc}
1 & 0 & \\
1 & 1 & \\
\end{array}
\end{array}
\]

This might come across as counter-intuitive first but it confirms the specification:

The intermediate matrix $\widetilde{T} = \langle D_{out}(\text{op}), \text{nrows}(\tilde{A}), \text{ncols}(\tilde{A}), \{(i, j, T_{ij}) : \text{ind}(\tilde{A}) \cap \text{ind}(\tilde{B}) \neq \emptyset\} \rangle$ is created. The value of each of its elements is computed by

\[
T_{ij} = (\tilde{A}(i, j) \oplus \tilde{B}(i, j)), \forall (i, j) \in \text{ind}(\tilde{A}) \cap \text{ind}(\tilde{B})
\]

\[
T_{ij} = \tilde{A}(i, j), \forall (i, j) \in (\text{ind}(\tilde{A}) - (\text{ind}(\tilde{B}) \cap \text{ind}(\tilde{A})))
\]

\[
T_{ij} = \tilde{B}(i, j), \forall (i, j) \in (\text{ind}(\tilde{B}) - (\text{ind}(\tilde{B}) \cap \text{ind}(\tilde{A})))
\]

where the difference operator in the previous expressions refers to set difference.
Extra slides
GRAPHBLAS TIMELINE

Book — Papers — GraphBLAS standards — SuiteSparse:GraphBLAS releases

2011

Graph Algorithms in the Language of Linear Algebra

Mathematical foundations, HPEC

2016

0.9

C API, GABB@IPDPS

2017

1.0

LAGraph, GrAPL@IPDPS

2018

1.2

2.2

SuiteSparse: GraphBLAS, TOMS

2019

1.3

2020
Many irregular applications contain coarse-grained parallelism that can be exploited by abstractions at the proper level.

<table>
<thead>
<tr>
<th>Traditional graph computation</th>
<th>Graphs in the language of linear algebra</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data-driven, unpredictable communication</td>
<td>Fixed communication patterns</td>
</tr>
<tr>
<td>Irregular and unstructured, poor locality of reference</td>
<td>Operations on matrix blocks exploit memory hierarchy</td>
</tr>
<tr>
<td>Fine-grained data accesses, dominated by latency</td>
<td>Coarse-grained parallelism, bandwidth-limited</td>
</tr>
</tbody>
</table>

D. Bader et al., *The GraphBLAS effort and its implications for Exascale*, SIAM Workshop on Exascale Applied Mathematics Challenges and Opportunities, 2014
DIRECTED HYPERGRAPH EXAMPLE
Graph algorithms in GraphBLAS

Weakly connected components
WCC: WEAKLY CONNECTED COMPONENTS (SKETCH)

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

J. Kepner, J.R. Gilbert: 
*Graph Algorithms in the Language of Linear Algebra*, Chapter 3, SIAM, 2011
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)
  - Recent result, based on the Awerbuch-Shiloach algorithm.
  - Designed for supercomputers, scales to 250,000+ CPU cores.
  - “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.
ABOUT THIS PRESENTATION

- This presentation is intended to serve as an introduction to semiring-based graph processing and the GraphBLAS.
- Common graph algorithms (BFS, shortest path, PageRank, etc.) are used to demonstrate the features of GraphBLAS. Many of the algorithms presented are part of the LAGraph library.
- The presentation complements existing technical talks on GraphBLAS and can form a basis of anything from a short 20min overview to 2×90min lectures on the GraphBLAS.
- The slides contain numerous references to papers and talks.
- There are detailed textual explanations on some slides to make them usable as a learning material.
TECHNICAL DETAILS

- The slides were created with PowerPoint 2016 using the Open Sans and DejaVu Sans Mono font families (embedded in the presentation).
- The mathematical expressions are typeset with PowerPoint’s built-in Equation Editor.
- The circled numbers (denoting graph vertices) are rendered using the standard Wingdings font.
- The text is written in Oxford English.
- The icons for referenced papers and talks are clickable and will lead to an open-access / preprint / author’s copy version of the referred work (if such copy exists). The icons depict the first page of the cited document.
FUTURE WORK #1

- Add a historical overview of research on sparse matrix multiplication.
- Add bidirectional BFS algorithm.
- Discuss design patterns for handling graph queries
  - e.g. multiplying by a diagonal matrix from the left selects source vertices and is a “sticky” operation
- How to represent a typed graph as a matrix?
  - A square matrix with labels in it (using the LOR.ISEQ operator)
  - Multiple square matrices
  - Multiple rectangular (not necessarily square) matrices that capture each type as a bipartite graph
FUTURE WORK #2

- **Matrix representations**
  - The ‘typical’ sparse matrix data structures: CSR, CRS
  - Basic updatable data structures: COO, LIL, DOK
  - Advanced updatable data structures: Hornet, faim

- **Important algorithms**
  - Matrix multiplication (dot, hash, heap, Gustavson)

- **Advanced GraphBLAS features**
  - the accumulator
  - merge/replace
  - submatrix extraction, assignment, Kronecker product