Introduction to GraphBLAS-based Graph Analytics

Gábor Szárnyas
GRAPH PROCESSING

- **Graph use cases:** Focus on the connections between entities rather than attribute values.
  - Social network: friend recommendation
  - Financial transactions: fraud detection
  - Logistics: routing, shortest paths
  - HW/SW models: validation
  - Runtime models: verification
  - Ontologies: inferencing

- Graph processing covers a **broad set of problems with a rich family of algorithms.** Is there a generic framework?
GRAPH PROCESSING CHALLENGES

connectedness
the “curse of connectedness”

computer architectures
contemporary computer architectures are good at processing linear and hierarchical data structures, such as lists, stacks, or trees

caching and parallelization
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 Bulletin 2017
Graph processing challenges

Graph algorithms have “a high communication to computation ratio” [ICCS’15]. Computing is cheap, finding what to compute is expensive.

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

What can we do about this?
HETEROGENEOUS COMPUTER ARCHITECTURE
DATA PROCESSING PIPELINES

Recurring problems in CS often form a “cluster” that can be tackled efficiently using the right tools.

**Tabular data processing**  relational databases  **relational algebra**
**Machine learning**  ML frameworks  **linear algebra**
**String manipulation**  Unix tools & pipeline  **finite automata algebras**

These algebras have proven to be composable, portable, and expressive. They have well-optimized implementations.

**Catch:** the underlying algebras are always the most intuitive ones.
ABSTRACTION LAYERS

Application
  - Dataframe lib
    - SQL
      - Logical plan
      - Physical plan

Numerical applications
  - LINPACK/LAPACK
    - BLAS
      - Dense linear algebra
      - Dense LA algorithms

Graph analytical apps
  - LAGraph
    - GraphBLAS
      - Sparse linear algebra
      - Sparse LA algorithms

Hardware architecture
The GraphBLAS standard
Graphs are encoded as **sparse adjacency matrices**.

Use **vector/matrix operations** to express graph algorithms.
# 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>Problem</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></td>
<td></td>
<td></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>$mn \log n$</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
KEY FEATURES OF GRAPHBLAS

- **Portable:** supported on x86, Arm (WIP), GPUs (WIP)
- **Efficient:** within one order-of-magnitude compared to hand-tuned code
- **Concise:** most textbook algorithms can be expressed with a few operations.
- **Composable:** the output of an algorithm can be used as an input of a subsequent algorithm
- **Flexible:** can express algorithms on typed graphs and property graphs
Theoretical foundations of the GraphBLAS
DENSE MATRIX MULTIPLICATION

Definition:

\[ \mathbf{C} = \mathbf{A}\mathbf{B} \]

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

Example:

\[ C(2,3) = A(2,1) \times B(1,3) + A(2,2) \times B(2,3) + A(2,3) \times B(3,3) \]

\[ = 2 \times 5 + 3 \times 0 + 6 \times 4 = 34 \]
**SPARSE MATRIX MULTIPLICATION**

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

Only evaluate the multiplication operator \( \odot \) for positions where there is a non-zero element in both \( A(i, k) \) and \( B(k, j) \).

Example:
\[ C(2,3) = A(2,1) \times B(1,3) + A(2,3) \times B(3,3) \]
\[ = 2 \times 5 + 6 \times 4 = 34 \]
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

\[ A_{ij} = \begin{cases} 
1 & \text{if } (v_i, v_j) \in E \\
0 & \text{if } (v_i, v_j) \notin E 
\end{cases} \]
$$A^T_{ij} = \begin{cases} 1 & \text{if } (v_j, v_i) \in E \\ 0 & \text{if } (v_j, v_i) \not\in E \end{cases}$$
ADJACENCY MATRIX TRANSPOSED

$$A^T_{ij} = \begin{cases} 
1 & \text{if } (v_j, v_i) \in E \\
0 & \text{if } (v_j, v_i) \notin E 
\end{cases}$$
GRAPH TRAVERSAL WITH MATRIX MULTIPLICATION

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

One hop: \( \mathbf{fA} \)
GRAPH TRAVERSAL WITH MATRIX MULTIPLICATION

\[ \mathbf{f} \mathbf{A}^k \] means \( k \) hops in the graph

One hop: \( \mathbf{fA} \)

Two hops: \( \mathbf{fA}^2 \)
MATRIX MULTIPLICATION $\mathbf{C} = \mathbf{A} \oplus \mathbf{B}$

Multiplication on dense matrices

\[ \mathbf{C}(i, j) = \bigoplus_j \mathbf{A}(i, k) \otimes \mathbf{B}(k, j) \]

Multiplication on sparse matrices

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

Example: $\mathbf{C} = \mathbf{A} + .\times \mathbf{B}$
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>⊕</th>
<th>⊗</th>
<th>0</th>
<th>graph semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>any-pair</td>
<td>arbitrary</td>
<td>any</td>
<td>pair</td>
<td>F</td>
<td>connectivity</td>
</tr>
<tr>
<td>integer arithmetic</td>
<td>$a \in \mathbb{N}$</td>
<td>+</td>
<td>$\times$</td>
<td>0</td>
<td>number of 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>
</tbody>
</table>

The default semiring is the conventional one:

- Operator $\otimes$ defaults to floating point multiplication.
- Operator $\oplus$ defaults to floating point addition.
## Matrix-Vector Multiplication Semantics

### Semantics: number of paths

<table>
<thead>
<tr>
<th>Semiring</th>
<th>Set</th>
<th>$\oplus$</th>
<th>$\otimes$</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer arithmetic</td>
<td>$a \in \mathbb{N}$</td>
<td>+</td>
<td>$\times$</td>
<td>0</td>
</tr>
</tbody>
</table>

### Example

Given a matrix $A$ and a vector $f$, the product $f + \times A$ represents the number of paths from the starting node to each node in the path.

- $1 \times 1 = 1$
- $1 + 1 = 2$

**Matrix $A$:**

```
\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 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\end{pmatrix}
```

**Vector $f$:**

```
\begin{pmatrix}
1 \\
1 \\
1 \\
1 \\
1 \\
1 \\
1 \\
1 \\
\end{pmatrix}
```

**Result $f + \times A$:**

```
\begin{pmatrix}
1 \\
2 \\
1 \\
1 \\
1 \\
1 \\
1 \\
1 \\
\end{pmatrix}
```
MATRIX-VECTOR MULTIPLICATION SEMANTICS

Semantics: reachability

Semiring | Set | ⊕ | ⊗ | 0
---|---|---|---|---
any-pair | arbitrary | any pair | F

Semantics: reachability
Semantics: shortest path

\[
\text{Semiring: } \minplus \quad a \in \mathbb{R} \cup \{+\infty\} \quad \min \quad + \quad +\infty
\]

\[
\begin{array}{c|c|c|c|c}
\text{semiring} & \text{set} & \ominus & \otimes & 0 \\
\hline
\minplus & \mathbb{R} \cup \{+\infty\} & \min & + & +\infty
\end{array}
\]

\[
f_{\min} + \mathbf{A}
\]

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

\[
\min(0.9,1.1)=0.9 \\
f \min \quad + \quad \mathbf{A}
\]
ELEMEENT-WISE MULTIPLICATION: $A \land B$
ELEMENT-WISE ADDITION: $A \lor B$

**A**  1  2  3  4  5  6  7
1  
2  
3  
4  
5  
6  
7  

**B**  1  2  3  4  5  6  7
1  
2  
3  
4  
5  
6  
7  

$A \lor B$  1  2  3  4  5  6  7
1  
2  
3  
4  
5  
6  
7  

$A \lor B$ =

$A \lor B$  1  2  3  4  5  6  7
1  
2  
3  
4  
5  
6  
7  

$A \lor B$ =

$A \lor B$  1  2  3  4  5  6  7
1  
2  
3  
4  
5  
6  
7  

$A \lor B$ =
TURNING A GRAPH INTO UNDIRECTED: $A \lor A^T$
MASKING

Prevent redundant computations by reducing the scope of an operation.
Operations can be executed:
- without a mask
- with a regular mask
- with a complemented mask

\[ w = f \oplus \cdot \odot A \]
Prevent redundant computations by reducing the scope of an operation.

Operations can be executed:
- without a mask
- with a regular mask
- with a complemented mask

$$w(m) = f \oplus \odot A$$
MASKING

Prevent redundant computations by reducing the scope of an operation.

Operations can be executed:

- without a mask
- with a regular mask
- with a complemented mask

\[ w(\overline{m}) = f \oplus. \otimes A \]
### 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.

<table>
<thead>
<tr>
<th>symbol</th>
<th>operation</th>
<th>notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \oplus \otimes )</td>
<td>matrix-matrix multiplication</td>
<td>( \mathbf{C} \langle \mathbf{M} \rangle = \mathbf{A} \oplus \otimes \mathbf{B} )</td>
</tr>
<tr>
<td>( \otimes )</td>
<td>vector-matrix multiplication</td>
<td>( \mathbf{w} \langle \mathbf{m} \rangle = \mathbf{v} \oplus \otimes \mathbf{A} )</td>
</tr>
<tr>
<td></td>
<td>matrix-vector multiplication</td>
<td>( \mathbf{w} \langle \mathbf{m} \rangle = \mathbf{A} \oplus \otimes \mathbf{v} )</td>
</tr>
<tr>
<td>( \otimes )</td>
<td>element-wise multiplication (set intersection of patterns)</td>
<td>( \mathbf{C} \langle \mathbf{M} \rangle = \mathbf{A} \otimes \mathbf{B} ) ( \mathbf{w} \langle \mathbf{m} \rangle = \mathbf{u} \otimes \mathbf{v} )</td>
</tr>
<tr>
<td>( \oplus )</td>
<td>element-wise addition (set union of patterns)</td>
<td>( \mathbf{C} \langle \mathbf{M} \rangle = \mathbf{A} \oplus \mathbf{B} ) ( \mathbf{w} \langle \mathbf{m} \rangle = \mathbf{u} \oplus \mathbf{v} )</td>
</tr>
<tr>
<td>( f )</td>
<td>apply unary operator</td>
<td>( \mathbf{C} \langle \mathbf{M} \rangle = f(\mathbf{A}) ) ( \mathbf{w} \langle \mathbf{m} \rangle = f(\mathbf{v}) )</td>
</tr>
<tr>
<td>( \oplus \ldots )</td>
<td>reduce to vector</td>
<td>( \mathbf{w} \langle \mathbf{m} \rangle = \left[ \oplus_j \mathbf{A}(:,j) \right] )</td>
</tr>
<tr>
<td>( \oplus \ldots )</td>
<td>reduce to scalar</td>
<td>( s = \left[ \oplus_{ij} \mathbf{A}(i,j) \right] )</td>
</tr>
<tr>
<td>( \mathbf{A}^\top )</td>
<td>transpose matrix</td>
<td>( \mathbf{C} \langle \mathbf{M} \rangle = \mathbf{A}^\top )</td>
</tr>
</tbody>
</table>
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>any</td>
<td>pair</td>
<td>F</td>
</tr>
</tbody>
</table>

$level = 1$

Any pair $\mathbf{A}$
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>any</td>
<td>pair</td>
<td>F</td>
</tr>
</tbody>
</table>

level = 2

$\mathbf{f}$

$v$

$f(\mathbf{v}) = f$ any. pair $\mathbf{A}$
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>any pair</td>
<td>F</td>
<td></td>
</tr>
</tbody>
</table>

level = 3

\[
\mathbf{f}\langle \mathbf{v} \rangle = \mathbf{f} \text{ any pair } \mathbf{A}
\]
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>any pair F</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

level = 4

\(f\) 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. $f(\overline{v}) = f$ any. pair $A$  
   use the any-pair semiring to advance the frontier
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>any-pair</td>
<td>$a \in {F, T}$</td>
<td>any pair</td>
<td>$F$</td>
<td></td>
</tr>
</tbody>
</table>
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
  - Unlike in BFS, there is no masking here, as revisiting edges that have been visited previously can be useful.
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>

A

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

\[\begin{align*}
\mathbf{d} & = \begin{bmatrix} 0 \\
\end{bmatrix} \\
\mathbf{d} \min+\mathbf{A} & = \\
\end{align*}\]
**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>

**Matrix \( A \):**

\[
\begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
1 & 0.3 & 0.8 & & & & & \\
2 & 0 & 0.1 & 0.7 & & & & & \\
3 & & 0 & 0.5 & & & & & \\
4 & & & 0.2 & 0.4 & 0 & & & \\
5 & & & & 0 & 0.1 & & & \\
6 & & & & & 0.5 & 0 & & \\
7 & & & & & & 0.1 & 0.5 & 0.9 & 0 \\
\end{array}
\]

**Matrix \( d \):**

\[
\begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
1 & 0 & & & & & & \\
2 & 0.3 & & & & & & \\
3 & & & & & & & & \\
4 & & & & & & & & \\
5 & & & & & & & & \\
6 & & & & & & & & \\
7 & & & & & & & & \\
\end{array}
\]

\[ \text{d min.} + \text{A} \]
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>+∞</td>
</tr>
</tbody>
</table>

Diagram of a directed graph with edge weights.

Matrix $A$ and vector $d$: $d \min.+ A$.
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>

\[ d_0 = 0.3 \]

\[ A \]

\[ d \]

\[ d_{\text{min.} \, + \, A} \]
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>+∞</td>
</tr>
</tbody>
</table>

\[
\begin{array}{ccccccccc}
\text{1} & \text{2} & \text{3} & \text{4} & \text{5} & \text{6} & \text{7} \\
\begin{bmatrix}
0 & .3 & .8 & 0 & .1 & .7 & 0 \\
0 & 0 & .1 & .5 & 0 & .1 & .5 \\
0 & 0 & 0 & 0 & .5 & 0 & .1 \\
0 & 0 & 0 & 0 & .5 & 0 & .1 \\
0 & 0 & 0 & 0 & .5 & 0 & .1 \\
0 & 0 & 0 & 0 & .5 & 0 & .1 \\
0 & 0 & 0 & 0 & .5 & 0 & .1 \\
\end{bmatrix}
\end{array}
\]

\[
\begin{array}{ccccccccc}
\text{1} & \text{2} & \text{3} & \text{4} & \text{5} & \text{6} & \text{7} \\
\begin{bmatrix}
0 & .3 & 1 & .8 & .4 & .5 & 1 \\
0 & 0 & .3 & .1 & .8 & .4 & .5 & 1 \\
\end{bmatrix}
\end{array}
\]
SSSP – ALGEBRAIC BELLMAN-FORD ALGO.

Input: adjacency matrix \( A \), source vertex \( s \), \#vertices \( n \)

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

Output: distance vector \( d \) (real)

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

Triangle count / Definition
TRIANGLE COUNT

- 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

![Diagram of a graph with labeled edges and vertices]

Number of unique triangles: \( \frac{30}{6} \)

*GraphChallenge.org: Raising the Bar on Graph Analytic Performance, HPEC 2018*
Graph algorithms in GraphBLAS

Triangle count / Naïve algorithm
TC EXAMPLE: NAÏVE APPROACH

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

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

\[ \text{TRI} = A + A \times A \times A \]

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

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

\[
\text{TRI}(A) = A.\times A
\]

\[
\text{tri2} = [+j \text{TRI}(\cdot,j)]
\]

\[
\begin{array}{cccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 2 & 1 & 1 \\
2 & 1 & 1 & 1 \\
1 & 2 & 2 & 1 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
2 & 2 & 2 & 2 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
2 & 2 & 2 & 2 \\
4 & 4 & 4 & 4 \\
8 & 8 & 8 & 8 \\
6 & 6 & 6 & 6 \\
4 & 4 & 4 & 4 \\
2 & 2 & 2 & 2 \\
2 & 2 & 2 & 2 \\
2 & 2 & 2 & 2 \\
\end{array}
\]

\[
[+j \cdots]
\]
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, $A \times A$ 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

Graph algorithms & GraphBLAS primitives
Based on the figure in A. Buluç: *Graph algorithms, computational motifs, and GraphBLAS*, ECP Meeting 2018
Complex case study
SIGMOD 2014 PROGRAMMING CONTEST

Annual contest
- Teams compete on database-related programming tasks
- Highly-optimized C++ implementations

2014 event
- Tasks on the LDBC social network graph
  - Benchmark data set for property graphs
  - People, forums, comments, hashtags, etc.
- 4 queries
  - Mix of filtering operations and graph algorithms
I. Compute an induced subgraph over Person-knows-Person

II. Run the graph algorithm on the subgraph

I.

Create induced subgraph from (pA)-[:knows]-{pB}.
hasTag

\( t: \text{Tag} \)

name = \$t$

Forum

hasMember

pA: Person

knows

Forum

hasMember

pB: Person

II.

exact closeness centrality

In the subgraph, compute the closeness centrality value for each Person \( p \), then return top-\( k \) Persons with the highest values.

p: Person

knows*

Person

0.67

0.80

0.80

0.67

key kernel: all-source BFS
Q4: CLOSINESS CENTRALITY VALUES

Q4 computes the top-\(k\) Person vertices based on their exact closeness centrality values:

\[
CCV(p) = \frac{(C(p) - 1)^2}{(n-1) \cdot s(p)}
\]

where

- \(C(p)\) is the size of the connected component of vertex \(p\),
- \(n\) is the number of vertices in the induced graph,
- \(s(p)\) is the sum of geodesic distances to all other reachable persons from \(p\).

\(s(p)\) is challenging: needs unweighted all-pairs shortest paths.
BOOLEAN ALL-SOURCE BFS ALGORITHM

Seen

\[\begin{array}{c|c|c|c|c|c}
1 & t1 & 2 & t2 & 3 & t3 & 4 & t4 & 5 & t5 \\
\hline
1 & \text{red} & 2 & \text{red} & 3 & \text{red} & 4 & \text{red} & 5 & \text{red} \\
2 & \text{red} & 1 & \text{red} & 3 & \text{red} & 4 & \text{red} & 5 & \text{red} \\
3 & \text{red} & 1 & \text{red} & 2 & \text{red} & 4 & \text{red} & 5 & \text{red} \\
4 & \text{red} & 1 & 2 & \text{red} & 3 & \text{red} & 5 & \text{red} & \text{red} \\
5 & \text{red} & 1 & 2 & 3 & \text{red} & 4 & \text{red} & \text{red} & \text{red} \\
\end{array}\]

\(\text{Frontier}\)

\[\begin{array}{c|c|c|c|c}
1 & \text{red} & 2 & \text{red} & 3 & \text{red} \\
2 & \text{red} & 1 & \text{red} & 3 & \text{red} \\
3 & \text{red} & 1 & \text{red} & 2 & \text{red} \\
4 & \text{red} & 1 & 2 & \text{red} & \text{red} \\
5 & \text{red} & 1 & 2 & 3 & \text{red} \\
\end{array}\]

\(\text{Next}(\neg \text{Seen}) = \quad \text{A any . pair Frontier}\)

\(\text{Seen}' = \quad \text{Seen any Next}\)
**BOOLEAN ALL-SOURCE BFS ALGORITHM**

- **Frontier**
  - `t1, t2, t3, t4, t5`

- **Seen**
  - `t1, t2, t3, t4, t5`

- **A**
  - `1, 2, 3, 4, 5`

- **Next(¬Seen)**
  - `A any pair Frontier`

- **Seen'**
  - `t1, t2, t3, t4, t5`

- **Seen'**
  - `t1, t2, t3, t4, t5`
BITWISE ALL-SOURCE BFS ALGORITHM

- For large graphs, the all-source BFS algorithm might need to run 500k+ traversals.

- Two top-ranking teams used bitwise operations to process traversals in batches of 64 [Then et al., VLDB’15].

- This idea can be adopted in the GraphBLAS algorithm by
  - using UINT64 values
  - performing the multiplication on the BOR. SECOND semiring, where BOR is “bitwise or” and SECOND(x, y) = y

- 5-10x speedup compared to the Boolean all-source BFS.
BITWISE ALL-SOURCE BFS ALGORITHM

Using UINT4s here

**Frontier**
- t1-t4: 1000 0000
- t5: 0000 1000

**Seen**
<table>
<thead>
<tr>
<th>t1-t4</th>
<th>t5</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>1000 0000</td>
</tr>
<tr>
<td>t2</td>
<td>0100 0000</td>
</tr>
<tr>
<td>t3</td>
<td>0010 0000</td>
</tr>
<tr>
<td>t4</td>
<td>0001 0000</td>
</tr>
<tr>
<td>t5</td>
<td>0000 1000</td>
</tr>
</tbody>
</table>

**A**
- t1-t4
- t5

**Next**
- A bor . second Frontier

**Seen’**
<table>
<thead>
<tr>
<th>t1-t4</th>
<th>t5</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>0101 0000</td>
</tr>
<tr>
<td>t2</td>
<td>1010 1000</td>
</tr>
<tr>
<td>t3</td>
<td>0101 1000</td>
</tr>
<tr>
<td>t4</td>
<td>1010 0000</td>
</tr>
<tr>
<td>t5</td>
<td>0110 0000</td>
</tr>
</tbody>
</table>

**Seen’** = Seen bor Next
BITWISE ALL-SOURCE BFS ALGORITHM

Frontier

<table>
<thead>
<tr>
<th></th>
<th>t1-t4</th>
<th>t5</th>
</tr>
</thead>
<tbody>
<tr>
<td>①</td>
<td>0101</td>
<td>0000</td>
</tr>
<tr>
<td>②</td>
<td>1010</td>
<td>1000</td>
</tr>
<tr>
<td>③</td>
<td>0101</td>
<td>1000</td>
</tr>
<tr>
<td>④</td>
<td>1010</td>
<td>0000</td>
</tr>
<tr>
<td>⑤</td>
<td>0110</td>
<td>0000</td>
</tr>
</tbody>
</table>

See

<table>
<thead>
<tr>
<th></th>
<th>t1-t4</th>
<th>t5</th>
</tr>
</thead>
<tbody>
<tr>
<td>①</td>
<td>0101</td>
<td>0000</td>
</tr>
<tr>
<td>②</td>
<td>1010</td>
<td>1000</td>
</tr>
<tr>
<td>③</td>
<td>0101</td>
<td>1000</td>
</tr>
<tr>
<td>④</td>
<td>1010</td>
<td>0000</td>
</tr>
<tr>
<td>⑤</td>
<td>0110</td>
<td>0000</td>
</tr>
</tbody>
</table>

A

<table>
<thead>
<tr>
<th></th>
<th>①</th>
<th>②</th>
<th>③</th>
<th>④</th>
<th>⑤</th>
</tr>
</thead>
<tbody>
<tr>
<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>
</tr>
<tr>
<td>③</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>
</tr>
<tr>
<td>⑤</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Next = A bor . second Frontier

Seen’

<table>
<thead>
<tr>
<th></th>
<th>t1-t4</th>
<th>t5</th>
</tr>
</thead>
<tbody>
<tr>
<td>①</td>
<td>1111</td>
<td>1000</td>
</tr>
<tr>
<td>②</td>
<td>1111</td>
<td>1000</td>
</tr>
<tr>
<td>③</td>
<td>1111</td>
<td>1000</td>
</tr>
<tr>
<td>④</td>
<td>1111</td>
<td>1000</td>
</tr>
<tr>
<td>⑤</td>
<td>1111</td>
<td>1000</td>
</tr>
</tbody>
</table>

Full VLDB paper on this algorithm vs. 9 GrB operations
GraphBLAS and SuiteSparse internals
GRAPHBLAS C API

“"A crucial piece of the GraphBLAS effort is to translate the mathematical specification to an API that
  o is faithful to the mathematics as much as possible, and
  o enables efficient implementations on modern hardware."

\[ \mathbf{C} \left( \mathbf{M} \right) \odot = \oplus \otimes \left( \mathbf{A}^T, \mathbf{B}^T \right) \]

\[
\text{mxm} (\text{Matrix } *\mathbf{C}, \text{Matrix } \mathbf{M}, \text{BinaryOp } \text{accum}, \text{Semiring } \text{op}, \text{Matrix } \mathbf{A}, \text{Matrix } \mathbf{B}, \text{Descriptor } \text{desc})
\]

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)

```
A
1  2  3  4  5  6  7
1  .3  .8  
2  .1  .7  
3  .5  
4  .2  .4
5  
6  .5
7  .1  .5  .9
```
SUITEPARSE: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.
- A 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
SUITESPARSE:GRAPHBLAS / LDBC GRAPHALYTICS

- **Twitter**: 50M nodes, 2B edges
- **d-8.8-zf**: 168M nodes, 413M edges
- **d-8.6-fb**: 6M nodes, 422M edges

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

Results from late 2019, new version is even faster
Summary
SUMMARY

- Linear algebra is a powerful abstraction
- GraphBLAS has good expressive power, allows concise formulation of algorithms and offers good performance
- Lots of ongoing research on the underpinning & usage
- Overall: sparse linear algebra as an abstraction layer for graph algorithms fits the age of heterogeneous hardware
Performance engineering case study: Grakn
Knowledge graph:
- is a highly heterogeneous graph built from multiple sources
- models an open domain

Basically: a graph with a non-trivial schema

Storage/processing without a schema is slow.

Grakn offers
- A rich type system for defining a schema
- The Graql query language
- Automated reasoning ("knowledge discovery")
Rules

Grakn allows you to define rules in your knowledge schema, which extends the expressivity of your model as it enables the system to derive new conclusions when a certain logical form in your dataset is satisfied.

Like functions in programming, that rules can chain itself to another, creating abstractions of behaviour at the data level.

```gql
# Schema Rules

define

transitive-location sub rule,
when {
    (located: $x, locating: $y);
    (located: $y, locating: $z);
},
then {
    (located: $x, locating: $z);
};

commit
```
1/ 2020 has been one helluva year for #engineering at @GraknLabs: we rewrote #grakn and #graql. We took the best of our #design, #math and #compsci from our work over the past 5 years that the #community loved, and we #rebuilt it with the best #software #architecture and #mindset
11/ Right now the team is in the midst of completing the new #reasoning engine, designed ground up using our very own #multithreaded #eventloop and #concurrent #actormodel. The new #reasoner #computation will be natively #parallel! @GraknLabs