Chapter 2

Mathematics

2.1 Introduction: Graphs as Matrices

This chapter describes the mathematics in the GraphBLAS standard. The GraphBLAS define a narrow set of mathematical operations that have been found to be useful for implementing a wide range of graph operations. At the heart of the GraphBLAS are 2D mathematical objects called matrices. The matrices are usually sparse, which implies that the majority of the elements in the matrix are zero and are often not stored to make their implementation more efficient. Sparsity is independent of the GraphBLAS mathematics. All the mathematics defined in the GraphBLAS will work regardless of whether the underlying matrix is sparse or dense.

Graphs represent connections between vertices with edges. Matrices can represent a wide range of graphs using adjacency matrices or incidence matrices. Adjacency matrices are often easier to analyze while incidence matrices are often better for representing data. Fortunately, the two are easily connected by the fundamental mathematical operation of the GraphBLAS: matrix-matrix multiply. One of the great features of the GraphBLAS mathematics is that no matter what kind of graph or matrix is being used, the core operations remain the same. In other words, a very small number of matrix operations can be used to manipulate a very wide range of graphs.

The mathematics of the GraphBLAS will be described using a “center outward” approach. Initially, the most important specific cases will be described that are at the center of GraphBLAS. The conditions on these cases will then be relaxed to arrive at more general definition. This approach has the advantage of being more easily understandable and describing the most important cases first.

2.1.1 Adjacency Matrix: Undirected Graphs, Directed Graphs, Weighted Graphs

Given an adjacency matrix $A$, if $A(v_1, v_2) = 1$, then there exists an edge going from vertex $v_1$ to vertex $v_2$. Likewise, if $A(v_1, v_2) = 0$, then there is no edge from $v_1$ to $v_2$. Adjacency matrices have
direction, which means that $A(v_1, v_2)$ is not the same as $A(v_2, v_1)$. Adjacency matrices can also have edge weights. If $A(v_1, v_2) = w_{12}$ and $w_{12} \neq 0$, then the edge going from $v_1$ to $v_2$ is said to have weight $w_{12}$. Adjacency matrices provide a simple way to represent the connections between vertices in a graph between one set of vertices and another. Adjacency matrices are often square and both out-vertices (rows) and the in-vertices (columns) are the same set of vertices. Adjacency matrices can be rectangular in which case the out-vertices (rows) and the in-vertices (columns) are different sets of vertices. Such graphs are often called bipartite graphs. In summary, adjacency matrices can represent a wide range of graphs, which include any graph with any set of the following properties: directed, weighted, and/or bipartite.

### 2.1.2 Incidence Matrix: Multi-Graphs, Hyper-Graphs, Multipartite Graphs

An incidence, or edge matrix $E$, uses the rows to represent every edge in the graph and the columns represent every vertex. There are a number of conventions for denoting an edge in an incidence matrix. One such convention is to set $E(i, v_1) = -1$ and $E(i, v_2) = 1$ to indicate that edge $i$ is a connection from $v_1$ to $v_2$. Incidence matrices are useful because they can easily represent multi-graphs, hyper-graphs, and multi-partite graphs. These complex graphs are difficult to capture with an adjacency matrix. A multi-graph has multiple edges between the same vertices. If there was another edge, $j$, from $v_1$ to $v_2$, this can be captured in an incidence matrix by setting $E(j, v_1) = -1$ and $E(j, v_2) = 1$. In a hyper-graph, one edge can go between more than two vertices. For example, to denote edge $i$ has a connection from $v_1$ to $v_2$ and $v_3$ can be accomplished by also setting $E(i, v_3) = 1$. Furthermore, $v_1$, $v_2$, and $v_3$ can be drawn from different classes of vertices and so $E$ can be used to represent multi-partite graphs. Thus, an incidence matrix can be used to represent a graph with any set of the following graph properties: directed, weighted, multi-partite, multi-edge, and/or hyper-edge.

### 2.2 Matrix Definition: Starting Vertices, Ending Vertices, Edge Weight Types

The canonical matrix of the GraphBLAS has $N$ rows and $M$ columns of real numbers. Such a matrix can be denoted as

$$A : \mathbb{R}^{N \times M}$$

The canonical row and and column indexes of the matrix $A$ are $i \in I = \{1, \ldots, N\}$ and $j \in J = \{1, \ldots, M\}$, so that any particular value $A$ can be denoted as $A(i, j)$. [Note: a specific
GraphBLAS implementation might use IEEE 64 bit double precision floating point numbers to represent real numbers, 64 bit unsigned integers to represent row and column indices, and the compressed sparse rows (CSR) format or the compressed sparse columns (CSC) format to store the non-zero values inside the matrix.

A matrix of complex numbers is denoted

\[ A : \mathbb{C}^{N \times M} \]

A matrix of integers \( \{ \ldots, -1, 0, 1, \ldots \} \) is denoted

\[ A : \mathbb{Z}^{N \times M} \]

A matrix of natural numbers \( \{1, 2, 3, \ldots \} \) is denoted

\[ A : \mathbb{N}^{N \times M} \]

Canonical row and column indices are natural numbers \( I, J : \mathbb{N} \). In some GraphBLAS implementations these indices could be non-negative integers \( I = \{0, \ldots, N - 1\} \) and \( J = \{0, \ldots, M - 1\} \).

For the GraphBLAS a matrix is defined as the following 2D mapping

\[ A : I \times J \rightarrow \mathbb{S} \]

where the indices \( I, J : \mathbb{Z} \) are finite sets of integers with \( N \) and \( M \) elements respectively, and \( \mathbb{S} \in \{\mathbb{R}, \mathbb{Z}, \mathbb{N}, \ldots\} \) is a set of scalars. Without loss of generality matrices can be denoted

\[ A : \mathbb{S}^{N \times M} \]

If the internal storage format of the matrix needs to be indicated, this can be done by

\[ A : \mathbb{S}_{\text{CSC}}^{N \times M} \quad \text{or} \quad A : \mathbb{S}_{\text{CSR}}^{N \times M} \]

A vector is a matrix where either \( N = 1 \) or \( M = 1 \). A column vector is denoted

\[ v = \mathbb{S}^{N \times 1} \]

A row vector is denoted

\[ v = \mathbb{S}^{1 \times M} \]

A scalar is a single element of a set \( s \in \mathbb{S} \) and has no matrix dimensions.
2.3 Scalar Operations: Combining and Scaling
Graph Edge Weights

The GraphBLAS matrix operations are built on top of scalar operations. The primary scalar operations are standard arithmetic addition (e.g., $1 + 1 = 2$) and multiplication (e.g., $2 \times 2 = 4$). The GraphBLAS also allow these scalar operations of addition and multiplication to be defined by the implementation or the user. To prevent confusion with standard addition and multiplication, $\oplus$ will be used to denote scalar addition and $\otimes$ will be used to denote scalar multiplication. In this notation, standard arithmetic addition and arithmetic multiplication of real numbers $a, b, c \in \mathbb{R}$, where $\oplus \equiv +$ and $\otimes \equiv \times$ results in

\[
c = a \oplus b \quad \Rightarrow \quad c = a + b
\]

\[
c = a \otimes b \quad \Rightarrow \quad c = a \times b
\]

Allowing $\oplus$ and $\otimes$ to be implementation (or user) defined functions enables the GraphBLAS to succinctly implement a wide range of algorithms on scalars of all different types (not just real numbers).

2.4 Scalar Properties: Composable Graph Edge Weight Operations

Certain $\oplus$ and $\otimes$ combinations over certain sets of scalars are particularly useful because they preserve desirable mathematical properties such as associativity

\[
(a \oplus b) \oplus c = a \oplus (b \oplus c) \quad \quad (a \otimes b) \otimes c = a \otimes (b \otimes c)
\]

and distributivity

\[
a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)
\]

Associativity, and distributivity are **extremely** useful properties for building graph applications because they allow the builder to swap operations without changing the result. They also increase opportunities for exploiting parallelism by the runtime.

Example combinations of $\oplus$ and $\otimes$ that preserve scalar associativity and distributivity include (but are not limited to) standard arithmetic

\[
\oplus \equiv + \quad \otimes \equiv \times \quad a, b, c \in \mathbb{R}
\]
max-plus algebras
\[ \oplus \equiv \max \quad \otimes \equiv + \quad a, b, c \in \{-\infty \cup \mathbb{R}\} \]

max-min algebras
\[ \oplus \equiv \max \quad \otimes \equiv \min \quad a, b, c \in [0, \infty] \]

finite (Galois) fields such as GF(2)
\[ \oplus \equiv \text{xor} \quad \otimes \equiv \text{and} \quad a, b, c \in [0, 1] \]

and power set algebras
\[ \oplus \equiv \cup \quad \otimes \equiv \cap \quad a, b, c \subseteq \mathbb{Z} \]

These operations also preserve scalar commutativity. Other functions can also be defined for \( \oplus \) and \( \otimes \) that do not preserve the above properties. For example, it is often useful for \( \oplus \) or \( \otimes \) to pull in other data such as vertex labels of a graph, such as the select2nd operation used in breadth-first search.

### 2.5 Matrix Properties: Composable Operations on Entire Graphs

Associativity, distributivity, and commutativity are very powerful properties of the GraphBLAS and separate it from standard graph libraries because these properties allow the GraphBLAS to be composable (i.e., you can re-order operations and know that you will get the same answer). Composability is what allows the GraphBLAS to implement a wide range of graph algorithms with just a few functions.

Let \( A, B, C \in \mathbb{S}^{N \times M} \), be matrices with elements \( a = A(i, j), b = B(i, j), \) and \( c = C(i, j) \).

Associativity, distributivity, and commutativity of scalar operations translates into similar properties on matrix operations in the following manner.

**Additive Commutativity** Allows graphs to be swapped and combined via matrix element-wise addition without changing the result
\[
a \oplus b = b \oplus a \quad \Rightarrow \quad A \oplus B = B \oplus A
\]

where matrix element-wise addition is given by \( C(i, j) = A(i, j) \oplus B(i, j) \)

**Multiplicative Commutativity** Allows graphs to be swapped, intersected, and scaled via matrix element-wise multiplication without changing the result
\[
a \otimes b = b \otimes a \quad \Rightarrow \quad A \otimes B = B \otimes A
\]
where matrix element-wise (Hadamard) multiplication is given by \( C(i, j) = A(i, j) \odot B(i, j) \)

**Additive Associativity** Allows graphs to be combined via matrix element-wise addition in any grouping without changing the result

\[
(a \oplus b) \oplus c = a \oplus (b \oplus c) \quad \Rightarrow \quad (A \oplus B) \oplus C = A \oplus (B \oplus C)
\]

**Multiplicative Associativity** Allows graphs to be intersected and scaled via matrix element-wise multiplication in any grouping without changing the result

\[
(a \otimes b) \otimes c = a \otimes (b \otimes c) \quad \Rightarrow \quad (A \otimes B) \otimes C = A \otimes (B \otimes C)
\]

**Element-Wise Distributivity** Allows graphs to be intersected and/or scaled and then combined or vice-verse without changing the result

\[
a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c) \quad \Rightarrow \quad A \otimes (B \oplus C) = (A \otimes B) \oplus (A \otimes C)
\]

**Matrix Multiply Distributivity** Allows graphs to be transformed via matrix multiply and then combined or vice-verse without changing the result

\[
a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c) \quad \Rightarrow \quad A(B \oplus C) = (AB) \oplus (AC)
\]

where matrix multiply \( C = AB \) is given by

\[
C(i, j) = \bigoplus_{k=1}^{M} A(i, k) \otimes B(k, j)
\]

for matrices \( A : \mathbb{S}^{N \times M}, B : \mathbb{S}^{M \times L}, \) and \( C : \mathbb{S}^{N \times L} \)

**Matrix Multiply Associativity** is another implication of scalar distributivity and allows graphs to be transformed via matrix multiply in any grouping without changing the result

\[
a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c) \quad \Rightarrow \quad (AB)C = A(BC)
\]

**Matrix Multiply Commutativity** In general, \( AB \neq BA \). Some cases where \( AB = BA \) include when one matrix is all zeros, one matrix is the identity matrix, both matrices are diagonal matrices, or both matrices are rotation matrices.
Sparse matrices play an important role in GraphBLAS. Many implementations of sparse matrices reduce storage by not storing the 0 valued elements in the matrix. In adjacency matrices, the 0 element is equivalent to no edge from the vertex represented by row to the vertex represented by the column. In incidence matrices, the 0 element is equivalent to the edge represented by row not including the vertex represented by the column. In most cases, the 0 element is standard arithmetic 0. The GraphBLAS also allows the 0 element to be defined by the implementation or user. This can be particularly helpful when combined with user defined $\oplus$ and $\otimes$ operations. Specifically, if the 0 element has certain properties with respect scalar $\oplus$ and $\otimes$, then sparsity of matrix operations can be managed efficiently. These properties are the additive identity

$$a \oplus 0 = a$$

and the multiplicative annihilator

$$a \otimes 0 = 0$$

Note: the above behavior of $\oplus$ and $\otimes$ with respect to 0 is a requirement for the GraphBLAS.

Example combinations of $\oplus$ and $\otimes$ that exhibit the additive identity and multiplicative annihilator are:

- Standard arithmetic over the real numbers $a \in \mathbb{R}$, $\oplus \equiv +$, $\otimes \equiv \times$, $0 \equiv 0 \Rightarrow$
  
  additive identity: $a \oplus 0 = a + 0 = a$
  multiplicative annihilator: $a \otimes 0 = a \times 0 = 0$

- Max-plus algebras over $a \in \{-\infty \cup \mathbb{R}\}$, $\oplus \equiv \text{max}$, $\otimes \equiv +$, $0 \equiv -\infty \Rightarrow$
  
  additive identity: $a \oplus 0 = \text{max}(a, -\infty) = a$
  multiplicative annihilator: $a \otimes 0 = a + -\infty = -\infty$

- Min-max algebras over $a \in [0, \infty]$, $\oplus \equiv \text{min}$, $\otimes \equiv \text{max}$, $0 \equiv \infty \Rightarrow$
  
  additive identity: $a \oplus 0 = \text{min}(a, \infty) = a$
  multiplicative annihilator: $a \otimes 0 = \text{max}(a, \infty) = \infty$

- The Galois field GF(2) over $a \in [0, 1]$, $\oplus \equiv \text{xor}$, $\otimes \equiv \text{and}$, $0 \equiv 0 \Rightarrow$
  
  additive identity: $a \oplus 0 = \text{xor}(a, 0) = a$
  multiplicative annihilator: $a \otimes 0 = \text{and}(a, 0) = 0$

- Power set algebras over sets of integers $a \subset \mathbb{Z}$, $\oplus \equiv \cup$, $\otimes \equiv \cap$, $0 \equiv \emptyset \Rightarrow$
  
  additive identity: $a \oplus 0 = a \cup \emptyset = a$
  multiplicative annihilator: $a \otimes 0 = a \cap \emptyset = \emptyset$
2.7 Matrix Graph Operations Overview

The core of the GraphBLAS is the ability to perform a wide range of graph operations on diverse types of graphs with a small number of matrix operations:

**BuildMatrix**  Build a sparse Matrix from row, column, and value triples. Example graph operations include: graph construction from a set of starting vertices, ending vertices, and edge weights.

**ExtractTuples**  Extract the row, column, and value Tuples corresponding to the non-zero elements in a sparse matrix. Example graph operations include: extracting a graph from is matrix represent.

**Transpose**  Flips or Transposes the rows and the columns of a sparse matrix. Implements reversing the direction of the graph. Can be implemented with **ExtractTuples** and **BuildMatrix**.

**MxM, MxV, VxM**  Matrix x (times) Matrix, Matrix x (times) Vector, Vector x (times) Matrix. Example graph operations include: single-source breadth first search, multi-source breadth first search, weighted breadth first search.

**Extract**  Extract sub-matrix from a larger matrix. Example graph operations include: sub-graph selection. Can be implemented with **BuildMatrix** and **MxM**.

**Assign**  Assign matrix to a set of indices in a larger matrix. Example graph operations include: sub-graph assignment. Can be implemented with **BuildMatrix** and **MxM**.

**EwiseAdd, EwiseMult**  Elementwise Addition of matrices, Elementwise Multiplication of matrices (Hadamard product). Example graph operations include: graph union and intersection along with edge weight scale and combine.

**Apply**  Apply unary function to a matrix. Example graph operations include: graph edge weight modification. Can be implemented via **EwiseAdd** or **EwiseMult**.

**Reduce**  Reduce sparse matrix. Implements vertex degree calculations. Can be implemented via **MxM**.

The above set of functions has been shown to be useful for implementing a wide range of graph algorithms. These functions strike a balance between providing enough functions to be useful to an application builders and while being few enough that they can be implemented effectively.

Furthermore, from an implementation perspective, there are only five functions that are truly fundamental: **BuildMatrix, ExtractTuples, MxM, EwiseAdd and EwiseMult**. The other GraphBLAS functions can be implemented from these functions five functions.
2.8 BuildMatrix: Edge List to Graph

The GraphBLAS may use a variety of internal formats for representing sparse matrices. This data can often be imported as triples of vectors $i$, $j$, and $v$ corresponding to the non-zero elements in the sparse matrix. Constructing an $N \times M$ sparse matrix from triples can be denoted

$$C \oplus= S^{N \times M}(i, j, v, \oplus)$$

where $i : I^L$, $j : J^L$, and $v : S^L$, are all $L$ element vectors, and the symbols in blue represent optional operations that can be specified by the user. The optional $\oplus= \oplus$ denotes the option of adding the product to the existing values in $C$. The optional $\oplus$ function defines how multiple entries with the same row and column are handled. If $\oplus$ is undefined then the default is to combine the values using standard arithmetic addition $+$. Other variants include replacing any or all of the vector inputs with single element vectors. For example

$$C = S^{N \times M}(i, j, 1)$$

would use the value of 1 for input values. Likewise, a row vector can be constructed using

$$C = S^{N \times M}(1, j, v)$$

and a column vector can be constructed using

$$C = S^{N \times M}(i, 1, v)$$

The value type of the sparse matrix can be further specified via

$$C : R^{N \times M}(i, j, v)$$

2.9 ExtracTuples: Graph to Vertex List

It is expected the GraphBLAS will need to send results to other software components. Triples are a common interchange format. The GraphBLAS ExtracTuples command performs this operation by extracting the non-zero triples from a sparse matrix and can be denoted mathematically as

$$(i, j, v) = A$$
2.10 Transpose: Swap Start and End Vertices

Swapping the rows and columns of a sparse matrix is a common tool for changing the direction of vertices in a graph. The transpose is denoted as

\[ C \oplus= A^T \]

or more explicitly

\[ C(j, i) = C(j, i) \oplus A(i, j) \]

where \( A : S^{N \times M} \) and \( C : S^{M \times N} \).

Transpose can be implemented using a combination of BuildMatrix and ExtractTuples as follows

\[(i, j, v) = A \]
\[ C = S^{M \times N}(j, i, v) \]

2.11 MxM: Weighted, Multi-Source, Breadth-First-Search

Matrix multiply is the most important operation in the GraphBLAS and can be used to implement a wide range of graph algorithms. In its most common form, MxM performs a matrix multiply using standard arithmetic addition and multiplication

\[ C = AB \]

or more explicitly

\[ C(i, j) = \sum_{k=1}^{M} A(i, k)B(k, j) \]

where \( A : \mathbb{R}^{N \times M} \), \( B : \mathbb{R}^{M \times L} \), and \( C : \mathbb{R}^{N \times L} \). MxM has many important variants that include accumulating results, transposing inputs or outputs, and user defined addition and multiplication. These variants can be used alone or in combination. When these variants are combined with the wide range of graphs that can be represented with sparse matrices, this results in many thousands of distinct graph operations that can be succinctly captured by multiplying two sparse matrices. As will be described subsequently, all of these variants can be represented by the following mathematical statement

\[ C^T \oplus= A^T \oplus \otimes B^T \]
where \( A : \mathbb{S}^{N \times M} \), \( B : \mathbb{S}^{M \times L} \), and \( C : \mathbb{S}^{N \times L} \) denote the option of adding the product to the existing values in \( C \), and \( \oplus \), \( \otimes \) makes explicit that \( \oplus \) and \( \otimes \) can be user defined functions.

Special cases of \( \text{MxM} \) include:

- \( \text{SpMxSpV} \): Sparse matrix times sparse vector
- \( \text{SpMxV} \): Sparse matrix times dense vector
- \( \text{SpMxM} \): Sparse matrix times multiple dense vectors
- \( \text{SpMxM} \): Sparse matrix times dense matrix

### 2.11.1 Accumulation: Summing up Edge Weights

\( \text{MxM} \) can be used to multiply and accumulate values into a matrix. One example is when the result of multiply \( A \) and \( B \) is added to the existing values in \( C \) (instead of replacing \( C \)). This can be written

\[
C += AB
\]

or more explicitly

\[
C(i, j) = C(i, j) + \sum_{k=1}^{M} A(i, k)B(k, j)
\]

### 2.11.2 Transposing Inputs or Outputs: Swapping Start and End Vertices

Another variant is to specify that the matrix multiply should be performed over the transpose of \( A \), \( B \), or \( C \).

Transposing the input matrix \( A \) implies

\[
C = A^TB
\]

or more explicitly

\[
C(i, j) = \sum_{k=1}^{M} A(k, i)B(k, j)
\]

where \( A : \mathbb{R}^{M \times N} \).
Transposing the input matrix $B$ implies

$$C = AB^T$$

or more explicitly

$$C(i, j) = \sum_{k=1}^{M} A(i, k)B(j, k)$$

where $B : \mathbb{R}^{L \times M}$.

Transposing the output matrix $C$ implies

$$C^T = AB$$

or more explicitly

$$C(j, i) = \sum_{k=1}^{M} A(i, k)B(k, j)$$

where $C : \mathbb{R}^{L \times N}$.

Other combinations include transposing both inputs $A$ and $B$

$$C = A^TB^T \quad \Rightarrow \quad C(i, j) = \sum_{k=1}^{M} A(k, i)B(j, k)$$

where $A : \mathbb{R}^{M \times N}$ and $B : \mathbb{R}^{L \times M}$; transposing both input $A$ and output $C$

$$C^T = A^TB \quad \Rightarrow \quad C(j, i) = \sum_{k=1}^{M} A(k, i)B(k, j)$$

where $A : \mathbb{R}^{M \times N}$, $B : \mathbb{R}^{M \times L}$, and $C : \mathbb{R}^{L \times N}$; and transposing both input $B$ and output $C$

$$C^T = AB^T \quad \Rightarrow \quad C(j, i) = \sum_{k=1}^{M} A(i, k)B(j, k)$$

where $A : \mathbb{R}^{N \times M}$, $B : \mathbb{R}^{L \times M}$ and $C : \mathbb{R}^{L \times N}$.

Normally, the transpose operation distributes over matrix multiplication

$$(AB)^T = B^TA^T$$

and so transposing both inputs $A$ and $B$ and the output $C$ is rarely used. Nevertheless, for
completeness, this operation is defined as

\[ \mathbf{C}^T = \mathbf{A}^T \mathbf{B}^T \quad \Rightarrow \quad \mathbf{C}(j, i) = \sum_{k=1}^{M} \mathbf{A}(k, i) \mathbf{B}(j, k) \]

where \( \mathbf{A} : \mathbb{R}^{M \times N}, \mathbf{B} : \mathbb{R}^{L \times M}, \) and \( \mathbf{C} : \mathbb{R}^{L \times N}. \)

### 2.11.3 Addition and Multiplication: Combining and Scaling Edges

Standard matrix multiplication on real numbers first performs scalar arithmetic multiplication on the elements and then performs scalar arithmetic addition on the results. The GraphBLAS allows the scalar operations of addition \( \oplus \) and multiplication \( \otimes \) to be replaced with user defined functions. This can be formally denoted as

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

or more explicitly

\[ \mathbf{C}(i, j) = \bigoplus_{k=1}^{M} \mathbf{A}(i, k) \otimes \mathbf{B}(k, j) \]

where \( \mathbf{A} : \mathbb{S}^{N \times M}, \mathbf{B} : \mathbb{S}^{M \times L}, \) and \( \mathbf{C} : \mathbb{S}^{N \times L}. \) In this notation, standard matrix multiply can be written

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

where \( \mathbb{S} \rightarrow \mathbb{R}. \) Other matrix multiplications of interest include max-plus algebras

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

or more explicitly

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

where \( \mathbb{S} \rightarrow \{ -\infty \cup \mathbb{R} \}; \) min-max algebras

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

or more explicitly

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

where \( \mathbb{S} \rightarrow [0, \infty) \); the Galois field of order 2

\[ \mathbf{C} = \mathbf{A} \text{ xor.and} \mathbf{B} \]
or more explicitly
\[ C(i, j) = \text{xor}_k \{ \text{and}(A(i, k), B(k, j)) \} \]
where \( S \rightarrow [0, 1] \); and power set algebras
\[ C = A \cup \cap B \]
or more explicitly
\[ C(i, j) = \bigcup_{k=1}^{M} A(i, k) \cap B(k, j) \]
where \( S \rightarrow \{ \mathbb{Z} \} \).

Accumulation also works with user defined addition and can be denoted
\[ C \oplus= A \oplus \otimes B \]
or more explicitly
\[ C(i, j) = C(i, j) \oplus \bigoplus_{k=1}^{M} A(i, k) \otimes B(k, j) \]

2.12 Extract: Selecting Sub-Graphs

Selecting sub-graphs is a very common graph operation. The GraphBLAS performs this operation with the Extract function by selecting starting vertices (row) and ending vertices (columns) from a matrix \( A : \mathbb{S}^{N \times M} \)
\[ C^T \oplus= A^T(i, j) \]
or more explicitly
\[ C(i, j) = A(i(i), j(j)) \]
where \( i \in \{1, ..., N_C\} \), \( j \in \{1, ..., M_C\} \), \( i : I^{N_C} \), and \( j : J^{M_C} \) select specific sets of rows and columns in a specific order. The resulting matrix \( C : \mathbb{S}^{N_C \times M_C} \) can be larger or smaller than the input matrix \( A \). Extract can also be used to replicate and/or permute rows and columns in a matrix.

Extract can be implemented using sparse matrix multiply as
\[ C = S(i) A S^T(j) \]
where \( S(i) \) and \( S(j) \) are selection matrices given by
\[ S(i) = \mathbb{S}^{N_C \times N} \{1, ..., N_C\}, i, 1) \]
\[ S(j) = S_{MC \times M}(\{1, ..., MC\}, j, 1) \]

## 2.13 Assign: Modifying Sub-Graphs

Modifying sub-graphs is a very common graph operation. The GraphBLAS performs this operation with the *Assign* function by selecting starting vertices (row) and ending vertices (columns) from a matrix \( A : \mathbb{S}^{N \times M} \) and assigning new values to them from another sparse matrix \( C \):

\[
C^T(i, j) \oplus= A^T
\]

or more explicitly

\[
C(i(i), j(j)) \oplus= A(i, j)
\]

where \( i \in \{1, ..., N_A\}, j \in \{1, ..., M_A\}, i : I_{NA} \) and \( j : J_{MA} \) select specific sets of rows and columns in a specific order and \( \oplus \) optionally allows \( B \) to added to the existing values of \( A \).

The additive form of *Extract* can be implemented using sparse matrix multiply as

\[
C \oplus= S^T(i) A S(j)
\]

where \( S(i) \) and \( S(j) \) are selection matrices given by

\[
S(i) = S_{N \times N}(\{1, ..., N_A\}, i, 1) \\
S(j) = S_{M \times M}(\{1, ..., M_A\}, j, 1)
\]

## 2.14 EwiseAdd, EwiseMult: Combining Graphs, Intersecting Graphs, Scaling Graphs

Combining graphs along with adding their edge weights can be accomplished by adding together their sparse matrix representations. *EwiseAdd* provides this operation

\[
C^T \oplus= A^T \oplus B^T
\]

where \( A, B, C : \mathbb{S}^{N \times M} \) or more explicitly

\[
C(i, j) = C(i, j) \oplus A(i, j) \oplus B(i, j)
\]
where $i \in \{1, ..., N\}$, and $j \in \{1, ..., M\}$ and $\oplus$ is an optional argument.

Intersecting graphs along with scaling their edge weights can be accomplished by element-wise multiplication of their sparse matrix representations. \textbf{EwiseMult} provides this operation

$$C^T \oplus= A^T \otimes B^T$$

where $A, B, C : \mathbb{S}^{N \times M}$ or more explicitly

$$C(i, j) = C(i, j) \oplus A(i, j) \otimes B(i, j)$$

where $i \in \{1, ..., N\}$, and $j \in \{1, ..., M\}$ and $\oplus$ is an optional argument.

### 2.15 Apply: Modify Edge Weights

Modifying edge weights can be done by via the element-wise by unary function $f()$ to the values of a sparse matrix

$$C \oplus= f(A)$$

or more explicitly

$$C(i, j) = C(i, j) \oplus f(A(i, j))$$

where $A, C : \mathbb{S}^{N \times M}$, and $f(0) = 0$.

\textbf{Apply} can be implemented via \textbf{EwiseMult} via

$$C \oplus= A \otimes A$$

where $\otimes \equiv f()$ and $f(a, a) = f(a)$.

### 2.16 Reduce: Compute Vertex Degrees

It is often desired to combine the weights of all the vertices that come out of the same starting vertices. This aggregation can be represented as a matrix product as

$$c \oplus= A \oplus\otimes 1$$
or more explicitly
\[ c(i, 1) = c(i, 1) \oplus M \bigoplus_{j=1}^{A(i,j)} \]
where \( c : S^{N \times 1} \) and \( A : S^{N \times M} \), and \( 1 : S^{M \times 1} \) is a column vector of all ones.

Likewise, combining all the weights of all the vertices that go into the same ending vertices can be represented as matrix product as
\[ c \oplus= 1 \oplus . \times A \]
or more explicitly
\[ c(1, j) = c(1, j) \oplus \bigoplus_{i=1}^{N} A(i,j) \]
where \( c : S^{1 \times M} \) and \( A : S^{N \times M} \), and \( 1 : S^{1 \times N} \) is a row vector of all ones.

3 2.17 Kronecker: Graph Generation (Proposal)

Generating graphs is a common operation in a wide range of graph algorithms. Graph generation is used in testing graphs algorithms, creating graph templates to match against, and to compare real graph data with models. The Kronecker product of two matrices is a convenient and well-defined matrix operation that can be used for generating a wide range of graphs from a few a parameters.

The Kronecker product is defined as follows
\[
C = A \otimes B = \begin{pmatrix}
A(1, 1) \otimes B & A(1, 2) \otimes B & \ldots & A(1, M_A) \otimes B \\
A(2, 1) \otimes B & A(2, 2) \otimes B & \ldots & A(2, M_A) \otimes B \\
\vdots & \vdots & & \vdots \\
A(N_A, 1) \otimes B & A(N_A, 2) \otimes B & \ldots & A(N_A, M_A) \otimes B
\end{pmatrix}
\]
where \( A : S^{N_A \times M_A} \), \( B : S^{N_B \times M_B} \), and \( C : S^{N_A M_B \times M_A M_B} \). More explicitly, the Kronecker product can be written as
\[
C((i_A - 1)N_A + i_B, (j_A - 1)M_A + j_B) = A(i_A, j_A) \otimes B(i_B, j_B)
\]

With the usual accumulation and transpose options, the Kronecker product can be written
\[ C^T \oplus=A^T \otimes B^T \]

The elements-wise multiply operation \( \otimes \) can be user defined so long as the resulting operation obeys the aforementioned rules on elements-wise multiplication such as the multiplicative
annihilator. If elements-wise multiplication and addition obey the conditions specified in section 2.5, then the Kronecker product has many of the same desirable properties, such as associativity

\[(A \otimes B) \otimes C = A \otimes (B \otimes C)\]

and element-wise distributivity over addition

\[A \otimes (B \oplus C) = (A \otimes B) \oplus (A \otimes C)\]

Finally, one unique feature of the Kronecker product is its relation to the matrix product

\[(A \otimes B)(C \otimes D) = (AC) \otimes (BD)\]

2.18 Appendix: Glossary

[Will eventually be moved to front matter.]

2.19 Appendix: Examples

[Will eventually be moved to back matter.]

For my own reference, I’m leaving in a few latex fragments I’d like to remember for when we add our own text.

\[
#pragma omp\textbf\{directive-name \} [clause[ \[ , \]\} clause] \} new-line
\]

As an example of how a construct might look, here is the single construct from OpenMP