GraphBLAS Mathematics
- Provisional Release 1.0 -

Jeremy Kepner

Generated on April 26, 2017

Contents

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.

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

Given an adjacency matrix $\mathbf{A}$, if $\mathbf{A}(v_1, v_2) = 1$, then there exists an edge going from vertex v_1 to vertex v_2 (see Figure 1.1). Likewise, if $\mathbf{A}(v_1, v_2) = 0$, then there is no edge from v_1 to v_2. Adjacency matrices have direction, which means that $\mathbf{A}(v_1, v_2)$ is not the same as $\mathbf{A}(v_2, v_1)$. Adjacency matrices can also have edge weights. If $\mathbf{A}(v_1, v_2) = w_{12}$, 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.

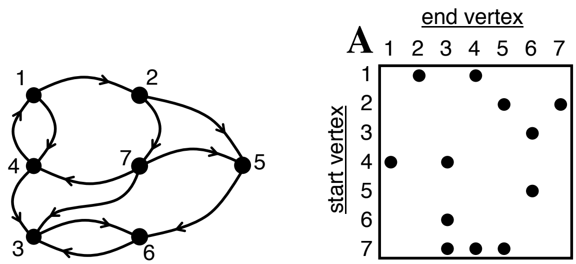

Figure 1.1: (left) Seven vertex graph with 12 edges. Each vertex is labeled with an integer. (right) 7×7 adjacency matrix $\mathbf{A}$ representation of the graph. $\mathbf{A}$ has 12 non-zero entries corresponding to the edges in the graph.

1.2 Incidence Matrix: Multi-Graphs, Hyper-Graphs, Multipartite Graphs

An incidence, or edge matrix $\mathbf{E}$, 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 $\mathbf{E}_{\text{start}}(i, v_1) = 1$ and $\mathbf{E}_{\text{end}}(i, v_2) = 1$ to indicate that edge i is a connection from v_1 to v_2 (see Figure 1.2). 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 $\mathbf{E}_{\text{start}}(j, v_1) = 1$ and $\mathbf{E}_{\text{end}}(j, v_2) = 1$ (see Figure 1.3). 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 $\mathbf{E}_{\text{end}}(i, v_3) = 1$ (see Figure 1.3). Furthermore, v_1, v_2, and v_3 can be drawn from different classes of vertices and so $\mathbf{E}$ 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 Matrix Definition: Starting Vertices, Ending Vertices, Edge Weight Types

The canonical matrix of the GraphBLAS has m rows and n columns of real numbers. Such a matrix can be denoted as

$$\mathbf{A} : \mathbb{R}^{m \times n}$$

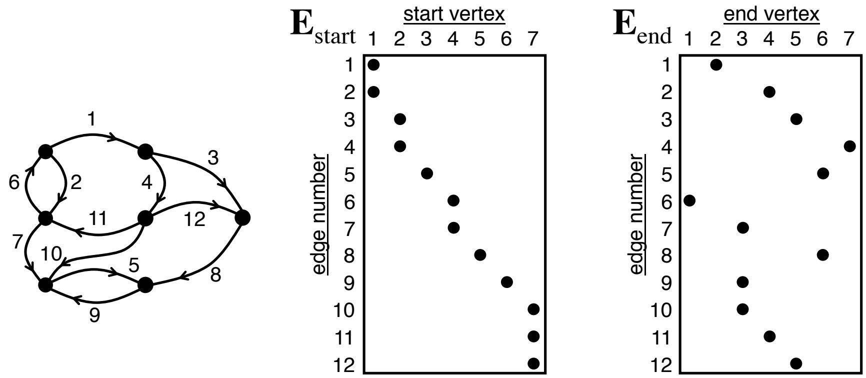

Figure 1.2: (left) Seven vertex graph with 12 edges. Each edge is labeled with an integer; the vertex labels are the same as in Figure 1.1. (middle) 12×7 incidence matrix $\mathbf{E}_{\text{start}}$ representing the starting vertices of the graph edges. (right) 12×7 incidence matrix $\mathbf{E}_{\text{end}}$ representing of the ending vertices of the graph edges. Both $\mathbf{E}_{start}$ and $\mathbf{E}_{\text{end}}$ have 12 non-zero entries corresponding to the edges in the graph.

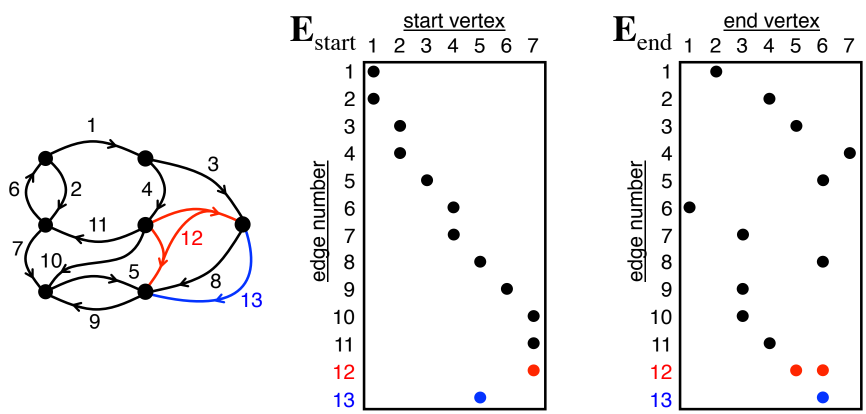

Figure 1.3: Graph and incidence matrices from Figure 1.2 with a hyper-edge (edge 12) and a multi-edge (edge 13). The graph is a hyper-graph because edge 12 has more than one end vertex. The graph is a multi-graph because edge 8 and edge 13 have the same start and end vertex.

The canonical row and and column indexes of the matrix $\mathbf{A}$ are $i \in I = \{1, \ldots, m\}$ and $j \in J = \{1, \ldots, n\}$, so that any particular value $\mathbf{A}$ can be denoted as $\mathbf{A}(i, j)$. [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

$$\mathbf{A} : \mathbb{C}^{m \times n}$$

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

$$\mathbf{A} : \mathbb{Z}^{m \times n}$$

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

$$\mathbf{A} : \mathbb{N}^{m \times n}$$

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

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

$$\mathbf{A} : I \times J \rightarrow \mathbb{S}$$

where the indices $I, J : \mathbb{Z}$ are finite sets of integers with m and n 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

$$\mathbf{A} : \mathbb{S}^{m \times n}$$

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

$$\mathbf{A} : \mathbb{S}^{m \times n}_{\text{CSC}} \qquad \text{or} \qquad \mathbf{A} : \mathbb{S}^{m \times n}_{\text{CSR}}$$

A matrix where $m = 1$ is a column vector and is denoted

$$\mathbf{v} = \mathbb{S}^{m \times 1}$$

A matrix where $n = 1$ is a row vector and is denoted

$$\mathbf{v} = \mathbb{S}^{1 \times n}$$

A pure vector is simply denoted

$$\mathbf{v} = \mathbb{S}^{m}$$

whether pure vector it is treated as a column vector or a row vector is determined by its context.

A scalar is a single element of a set $s \in \mathbb{S}$ and has no matrix dimensions.

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

$$c = a \oplus b \qquad \Rightarrow \qquad c = a + b$$

and

$$c = a \otimes b \qquad \Rightarrow \qquad c = a \times b$$

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).

4 Scalar Properties: Composable Graph Edge Weight Operations

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

$$(a \oplus b) \oplus c = a \oplus (b \oplus c) \qquad\qquad (a \otimes b) \otimes c = a \otimes (b \otimes c)$$

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

$$\oplus \equiv + \qquad \otimes \equiv \times \qquad a, b, c \in \mathbb{R}$$

max-plus algebras

$$\oplus \equiv \max \qquad \otimes \equiv + \qquad a, b, c \in \{-\infty \cup \mathbb{R}\}$$

max-min algebras

$$\oplus \equiv \max \qquad \otimes \equiv \min \qquad a, b, c \in [0, \infty]$$

finite (Galois) fields such as GF(2)

$$\oplus \equiv \text{xor} \qquad \otimes \equiv \text{and} \qquad a, b, c \in [0, 1]$$

and power set algebras

$$\oplus \equiv \cup \qquad \otimes \equiv \cap \qquad a, b, c \subset \mathbb{Z}$$

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.

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 $\mathbf{A}, \mathbf{B}, \mathbf{C} \in \mathbb{S}^{m \times n}$, be matrices with elements $a = \mathbf{A}(i, j)$, $b = \mathbf{B}(i, j)$, and $c = \mathbf{C}(i, j)$. 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 (see Figure 1.4) without changing the result

$$a \oplus b = b \oplus a \qquad \Rightarrow \qquad \mathbf{A} \oplus \mathbf{B} = \mathbf{B} \oplus \mathbf{A}$$

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

Multiplicative Commutativity Allows graphs to be swapped, intersected, and scaled via matrix element-wise multiplication (see Figure **??**) without changing the result

$$a \otimes b = b \otimes a \qquad \Rightarrow \qquad \mathbf{A} \otimes \mathbf{B} = \mathbf{B} \otimes \mathbf{A}$$

where matrix element-wise (Hadamard) multiplication is given by $\mathbf{C}(i,j) = \mathbf{A}(i,j) \otimes \mathbf{B}(i,j)$

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

$$(a \oplus b) \oplus c = a \oplus (b \oplus c) \qquad \Rightarrow \qquad (\mathbf{A} \oplus \mathbf{B}) \oplus \mathbf{C} = \mathbf{A} \oplus (\mathbf{B} \oplus \mathbf{C})$$

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

$$(a \otimes b) \otimes c = a \otimes (b \otimes c) \qquad \Rightarrow \qquad (\mathbf{A} \otimes \mathbf{B}) \otimes \mathbf{C} = \mathbf{A} \otimes (\mathbf{B} \otimes \mathbf{C})$$

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

$$a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c) \qquad \Rightarrow \qquad \mathbf{A} \otimes (\mathbf{B} \oplus \mathbf{C}) = (\mathbf{A} \otimes \mathbf{B}) \oplus (\mathbf{A} \otimes \mathbf{C})$$

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

$$a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c) \qquad \Rightarrow \qquad \mathbf{A}(\mathbf{B} \oplus \mathbf{C}) = (\mathbf{A}\mathbf{B}) \oplus (\mathbf{A}\mathbf{C})$$

where matrix multiply $\mathbf{C} = \mathbf{A}\mathbf{B}$ is given by

$$\mathbf{C}(i,j) = \bigoplus_{k=1}^{l} \mathbf{A}(i,k) \otimes \mathbf{B}(k,j)$$

for matrices $\mathbf{A} : \mathbb{S}^{m \times l}$, $\mathbf{B} : \mathbb{S}^{l \times n}$, and $\mathbf{C} : \mathbb{S}^{m \times n}$

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

$$a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c) \qquad \Rightarrow \qquad (\mathbf{A}\mathbf{B})\mathbf{C} = \mathbf{A}(\mathbf{B}\mathbf{C})$$

Matrix Multiply Commutativity In general, $\mathbf{A}\mathbf{B} \neq \mathbf{B}\mathbf{A}$. Some cases where $\mathbf{A}\mathbf{B} = \mathbf{B}\mathbf{A}$ include when one matrix is all zeros, one matrix is the identity matrix, both matrices are diagonal matrices, or both matrices are rotation matrices.

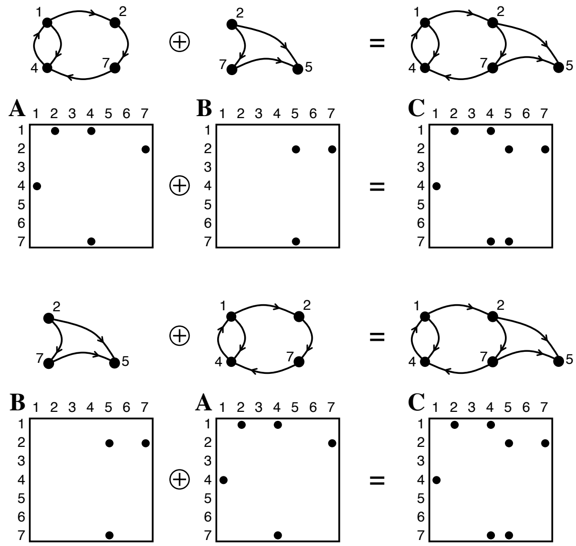

Figure 1.4: Illustration of the commutative property of the element-wise addition of two graphs and their corresponding adjacency matrix representations.

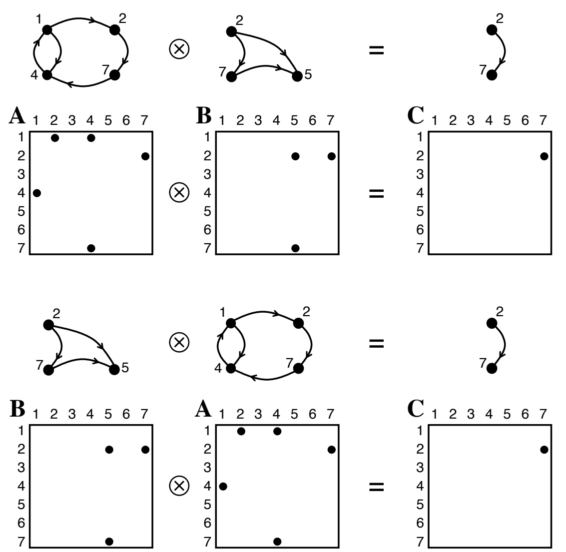

Figure 1.5: Illustration of the commutative property of the element-wise multiplication of two graphs and their corresponding adjacency matrix representations.

6 0-Element: No Graph Edge

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:

Arithmetic on Real Numbers ($+.\times$) Given standard arithmetic over the real numbers

$$a \in \mathbb{R}$$

where addition is

$$\oplus \equiv +$$

multiplication is

$$\otimes \equiv \times$$

and zero is

$$0 \equiv 0$$

which results in additive identity

$$a \oplus 0 = a + 0 = a$$

and multiplicative annihilator

$$a \otimes 0 = a \times 0 = 0$$

Max-Plus Algebra ($\max.+$) Given real numbers with a minimal element

$$a \in \{\text{-}\infty \cup \mathbb{R}\}$$

where addition is

$$\oplus \equiv \max$$

multiplication is

$$\otimes \equiv +$$

and zero is

$$0 \equiv \text{-}\infty$$

which results in additive identity

$$a \oplus 0 = \max(a, \text{-}\infty) = a$$

and multiplicative annihilator

$$a \otimes 0 = a + \text{-}\infty = \text{-}\infty$$

Min-Plus Algebra ($\min.+$) Given real numbers with a maximal element

$$a \in \{\mathbb{R} \cup \infty\}$$

where addition is

$$\oplus \equiv \min$$

multiplication is

$$\otimes \equiv +$$

and zero is

$$0 \equiv \infty$$

which results in additive identity

$$a \oplus 0 = \min(a, \infty) = a$$

and multiplicative annihilator

$$a \otimes 0 = a + \infty = \infty$$

Max-Min Algebra ($\max.\min$) Given non-negative real numbers

$$\mathbb{R}_{\geq 0} = \{a : 0 \leq a < \infty\}$$

where addition is

$$\oplus \equiv \max$$

multiplication is

$$\otimes \equiv \min$$

and zero is

$$0 \equiv 0$$

which results in additive identity

$$a \oplus 0 = \max(a, 0) = a$$

and multiplicative annihilator

$$a \otimes 0 = \min(a, 0) = 0$$

Min-Max Algebra ($\min.\max$) Given non-positive real numbers

$$\mathbb{R}_{\leq 0} = \{a : \text{-}\infty < a \leq 0\}$$

where addition is

$$\oplus \equiv \min$$

multiplication is

$$\otimes \equiv \max$$

and zero is

$$0 \equiv 0$$

which results in additive identity

$$a \oplus 0 = \min(a, 0) = a$$

and multiplicative annihilator

$$a \otimes 0 = \max(a, 0) = 0$$

Galois Field ($\text{xor}.\text{and}$) Given a set of two numbers

$$a \in \{0, 1\}$$

where addition is

$$\oplus \equiv \text{xor}$$

multiplication is

$$\otimes \equiv \text{and}$$

and zero is

$$0 \equiv 0$$

which results in additive identity

$$a \oplus 0 = \text{xor}(a, 0) = a$$

and multiplicative annihilator

$$a \otimes 0 = \text{and}(a, 0) = 0$$

Power Set ($\cup.\cap$) Given any subset of integers

$$a \subset \mathbb{Z}$$

where addition is

$$\oplus \equiv \cup$$

multiplication is

$$\otimes \equiv \cap$$

and zero is

$$0 \equiv \emptyset$$

which results in additive identity

$$a \oplus 0 = a \cup \emptyset = a$$

and multiplicative annihilator

$$a \otimes 0 = a \cap \emptyset = \emptyset$$

The above examples are a small selection of the operators and sets that are useful for building graph algorithms. Many more are possible. The ability to change the scalar values and operators while preserving the overall behavior of the graph operations is one of the principal benefits of using matrices for graph algorithms. For example, relaxing the requirement that the multiplicative annihilator be the additive identity, as in the above examples, yields additional operations, such as

Max-Max Algebra ($\max.\max$**)** Given non-positive real numbers with a minimal element

$$a \in \{\text{-}\infty \cup \mathbb{R}_{\leq 0}\}$$

where addition is

$$\oplus \equiv \max$$

multiplication is (also)

$$\otimes \equiv \max$$

and zero is

$$0 \equiv \text{-}\infty$$

which results in additive identity

$$a \oplus 0 = \max(a, \text{-}\infty) = a$$

Min-Min Algebra ($\min.\max$**)** Given non-negative real numbers with a maximal element

$$a \in \{\mathbb{R}_{\geq 0} \cup \infty\}$$

where addition is

$$\oplus \equiv \min$$

multiplication is (also)

$$\otimes \equiv \min$$

and zero is

$$0 \equiv \infty$$

which results in additive identity

$$a \oplus 0 = \min(a, \infty) = a$$

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:

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

Vector_build **build** a sparse **Vector** from index value tuples.

Matrix_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.

Vector_extractTuples **extract** the index and value **Tuples** corresponding to the non-zero elements in a sparse **vector**.

transpose Flips or **transpose**s the rows and the columns of a sparse matrix. Implements reversing the direction of the graph. Can be implemented with **ExtracTuples** and **BuildMatrix**.

mxm, mxv, vxm **m**atrix **x** (times) **m**atrix, **m**atrix **x** (times) **v**ector, **v**ector **x** (times) **m**atrix. 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 **Matrix_build** and **mxm** or **Vector_build** and **mxv** or **vxm**.

assign **assign** matrix or vector to a set of indices in a larger matrix or vector. Example graph operations include: sub-graph assignment. Can be implemented with **Matrix_build** and **mxm** or **Vector_build** and **mxv** or **vxm**.

eWiseAdd, eWiseMult **e**lement**Wise** **Add**ition of matrices or vectors, **e**lement**Wise** **Mult**iplication (Hadamard product) of matrices or vectors. Example graph operations include: graph union and intersection along with edge weight scale and combine.

apply **apply** unary function to a matrix or a vector. 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**, **mxv**, or **vxm**.

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 six functions that are truly fundamental: **Matrix_build**, **Matrix_extractTuples**, **transpose**, **mxm**, **eWiseAdd** and **eWiseMult**. The other GraphBLAS functions can be implemented from these six functions.

8 Matrix_build: 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 $\mathbf{i}$, $\mathbf{j}$, and $\mathbf{v}$ corresponding to the non-zero elements in the sparse matrix. Constructing an $m \times n$ sparse matrix from triples can be denoted

$$\mathbf{C} \;\oplus\!= \;\mathbb{S}^{m \times n}(\mathbf{i}, \mathbf{j}, \mathbf{v}, \oplus)$$

where $\mathbf{i} : I^L$, $\mathbf{j} : J^L$, $\mathbf{i}, \mathbf{v} : \mathbb{S}^L$, are all L element vectors, and the symbols in blue represent optional operations that can be specified by the user. The optional $\oplus\!=$ denotes the option of adding the product to the existing values in $\mathbf{C}$. 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

$$\mathbf{C} = \mathbb{S}^{m \times n}(\mathbf{i}, \mathbf{j}, 1)$$

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

$$\mathbf{C} = \mathbb{S}^{m \times n}(1, \mathbf{j}, \mathbf{v})$$

and a column vector can be constructed using

$$\mathbf{C} = \mathbb{S}^{m \times n}(\mathbf{i}, 1, \mathbf{v})$$

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

$$\mathbf{C} : \mathbb{R}^{m \times n}(\mathbf{i}, \mathbf{j}, \mathbf{v})$$

9 Vector_build

Using notation similar to **Matrix_build**, constructing an m element abstract sparse vector can be denoted

$$\mathbf{c} \ \oplus\!= \ \mathbb{S}^m(\mathbf{i}, \mathbf{v}, \oplus)$$

10 Matrix_extractTuples: 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 **ExtractTuples** command performs this operation by extracting the non-zero triples from a sparse matrix and can be denoted mathematically as

$$(\mathbf{i}, \mathbf{j}, \mathbf{v}) = \mathbf{A}$$

11 Vector_extractTuples

Using notation similar to **Matrix_extractTuples**, extracting the non-zero elements from a sparse vector and can be denoted mathematically as

$$(\mathbf{i}, \mathbf{v}) = \mathbf{c}$$

12 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 (see Figure 1.6). The transpose is denoted as

$$\mathbf{C} \ \oplus\!= \ \mathbf{A}^\mathsf{T}$$

or more explicitly

$$\mathbf{C}(j, i) = \mathbf{C}(j, i) \ \oplus \ \mathbf{A}(i, j)$$

where $\mathbf{A} : \mathbb{S}^{m \times n}$ and $\mathbf{C} : \mathbb{S}^{n \times m}$

Transpose can be implemented using a combination of **Matrix_build** and **Matrix_extractTuples** as follows

$$(\mathbf{i}, \mathbf{j}, \mathbf{v}) = \mathbf{A}$$

$$\mathbf{C} = \mathbb{S}^{n \times m}(\mathbf{j}, \mathbf{i}, \mathbf{v})$$

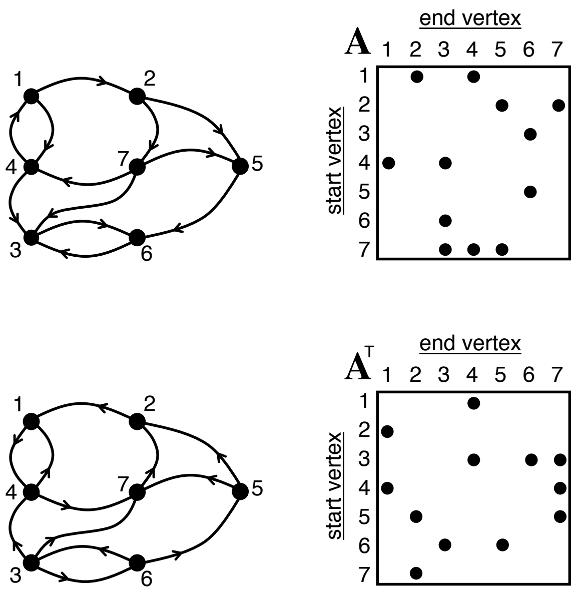

Figure 1.6: Transposing the adjacency matrix of a graph switches the directions of its edges.

13 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. Examples include finding the nearest neighbors of a vertex (see Figure 1.7) and constructing an adjacency matrix from an incidence matrix (see Figure 1.8). In its most common form, **mxm** performs a matrix multiply using standard arithmetic addition and multiplication

$$\mathbf{C} = \mathbf{AB}$$

or more explicitly

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

where $\mathbf{A} : \mathbb{R}^{m \times l}$, $\mathbf{B} : \mathbb{R}^{l \times n}$, and $\mathbf{C} : \mathbb{R}^{m \times n}$. **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

$$\mathbf{C}^{\mathsf{T}} \;\oplus\!= \; \mathbf{A}^{\mathsf{T}} \oplus.\otimes \mathbf{B}^{\mathsf{T}}$$

where $\mathbf{A} : \mathbb{S}^{m \times l}$, $\mathbf{B} : \mathbb{S}^{l \times m}$, and $\mathbf{C} : \mathbb{S}^{m \times n}$, $\oplus\!=$ denotes the option of adding the product to the existing values in $\mathbf{C}$, and $\oplus.\otimes$ makes explicit that $\oplus$ and $\otimes$ can be user defined functions.

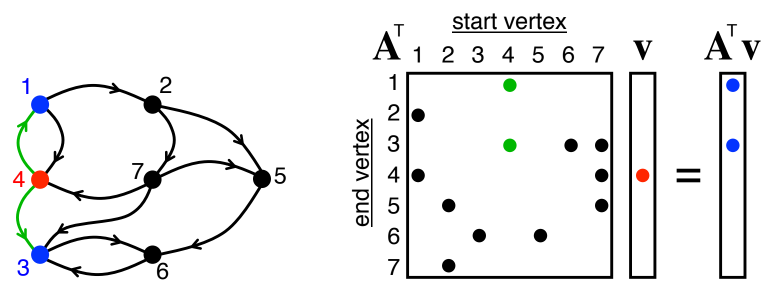

Figure 1.7: (left) Breadth-first-search of a graph starting at vertex 4 and traversing out to vertices 1 and 3. (right) Matrix-vector multiplication of the adjacency matrix of a graph performs the equivalent operation.

13.1 accumulation: Summing up Edge Weights

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

$$\mathbf{C} \;\mathrel{+}= \; \mathbf{A}\mathbf{B}$$

or more explicitly

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

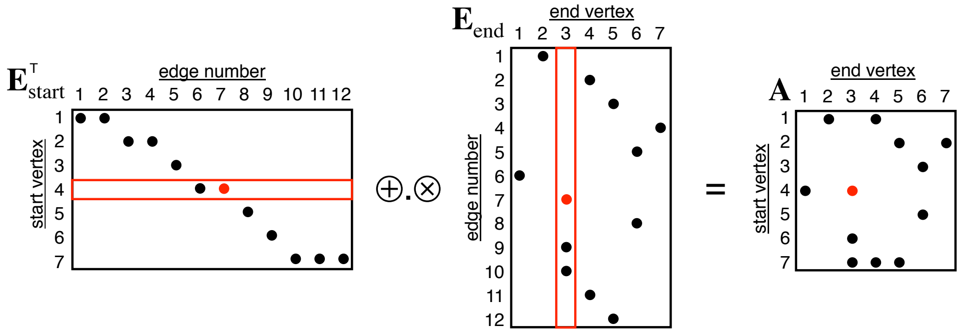

Figure 1.8: Construction of an adjacency matrix of a graph from its incidence matrices via matrix-matrix multiply. The entry $\mathbf{A}(4,3)$ is obtained by combining the row vector $\mathbf{E}^{\mathsf{T}}_{\text{start}}(4,k)$ with the column vector $\mathbf{E}_{\text{end}}(k,3)$ via matrix-matrix product $\mathbf{A}(4,3) = \bigoplus_{k=1}^{13} \mathbf{E}^{\mathsf{T}}_{\text{start}}(4,k) \otimes \mathbf{E}_{\text{end}}(k,3)$

13.2 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 $\mathbf{A}$, $\mathbf{B}$, or $\mathbf{C}$.

Transposing the input matrix $\mathbf{A}$ implies

$$\mathbf{C} = \mathbf{A}^{\mathsf{T}}\mathbf{B}$$

or more explicitly

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

where $\mathbf{A} : \mathbb{R}^{n \times m}$.

Transposing the input matrix $\mathbf{B}$ implies

$$\mathbf{C} = \mathbf{A}\mathbf{B}^{\mathsf{T}}$$

or more explicitly

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

where $\mathbf{B} : \mathbb{R}^{l \times n}$.

Transposing the output matrix $\mathbf{C}$ implies

$$\mathbf{C}^{\mathsf{T}} = \mathbf{A}\mathbf{B}$$

or more explicitly

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

where $\mathbf{C} : \mathbb{R}^{m\times n}$.

Other combinations include transposing both inputs $\mathbf{A}$ and $\mathbf{B}$

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

where $\mathbf{A} : \mathbb{R}^{l\times m}$ and $\mathbf{B} : \mathbb{R}^{n\times l}$; transposing both input $\mathbf{A}$ and output $\mathbf{C}$

$$\mathbf{C}^{\mathsf{T}} = \mathbf{A}^{\mathsf{T}}\mathbf{B} \qquad \Rightarrow \qquad \mathbf{C}(j,i) = \sum_{k=1}^{l} \mathbf{A}(k,i)\mathbf{B}(k,j)$$

where $\mathbf{A} : \mathbb{R}^{l\times m}$, $\mathbf{B} : \mathbb{R}^{l\times n}$, and $\mathbf{C} : \mathbb{R}^{m\times n}$; and transposing both input $\mathbf{B}$ and output $\mathbf{C}$

$$\mathbf{C}^{\mathsf{T}} = \mathbf{A}\mathbf{B}^{\mathsf{T}} \qquad \Rightarrow \qquad \mathbf{C}(j,i) = \sum_{k=1}^{l} \mathbf{A}(i,k)\mathbf{B}(j,k)$$

where $\mathbf{A} : \mathbb{R}^{m\times l}$, $\mathbf{B} : \mathbb{R}^{n\times l}$ and $\mathbf{C} : \mathbb{R}^{n\times m}$.

Normally, the transpose operation distributes over matrix multiplication

$$(\mathbf{A}\mathbf{B})^{\mathsf{T}} = \mathbf{B}^{\mathsf{T}}\mathbf{A}^{\mathsf{T}}$$

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

$$\mathbf{C}^{\mathsf{T}} = \mathbf{A}^{\mathsf{T}}\mathbf{B}^{\mathsf{T}} \qquad \Rightarrow \qquad \mathbf{C}(j,i) = \sum_{k=1}^{l} \mathbf{A}(k,i)\mathbf{B}(j,k)$$

where $\mathbf{A} : \mathbb{R}^{l\times m}$, $\mathbf{B} : \mathbb{R}^{n\times l}$, and $\mathbf{C} : \mathbb{R}^{n\times m}$.

13.3 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

$$\mathbf{C} = \mathbf{A} \oplus.\otimes \mathbf{B}$$

or more explicitly

$$\mathbf{C}(i,j) = \bigoplus_{k=1}^{l} \mathbf{A}(i,k) \otimes \mathbf{B}(k,j)$$

where $\mathbf{A} : \mathbb{S}^{m \times l}$, $\mathbf{B} : \mathbb{S}^{m \times l}$, and $\mathbf{C} : \mathbb{S}^{m \times n}$. In this notation, standard matrix multiply can be written

$$\mathbf{C} = \mathbf{A} +.\times \mathbf{B}$$

where $\mathbb{S} \to \mathbb{R}$. Other matrix multiplications of interest include max-plus algebras

$$\mathbf{C} = \mathbf{A} \text{ max.+ } \mathbf{B}$$

or more explicitly

$$\mathbf{C}(i,j) = \max_k \{\mathbf{A}(i,k) + \mathbf{B}(k,j)\}$$

where $\mathbb{S} \to \{-\infty \cup \mathbb{R}\}$; min-max algebras

$$\mathbf{C} = \mathbf{A} \text{ min.max } \mathbf{B}$$

or more explicitly

$$\mathbf{C}(i,j) = \min_k \{\max(\mathbf{A}(i,k), \mathbf{B}(k,j))\}$$

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

$$\mathbf{C} = \mathbf{A} \text{ xor.and } \mathbf{B}$$

or more explicitly

$$\mathbf{C}(i,j) = \text{xor}_k \{\text{and}(\mathbf{A}(\text{i},\text{k}), \mathbf{B}(\text{k},\text{j}))\}$$

where $\mathbb{S} \to [0, 1]$; and power set algebras

$$\mathbf{C} = \mathbf{A} \cup.\cap \mathbf{B}$$

or more explicitly

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

where $\mathbb{S} \rightarrow \{\mathbb{Z}\}$.

Accumulation also works with user defined addition and can be denoted

$$\mathbf{C} \ \oplus\!= \ \mathbf{A} \oplus.\!\otimes \mathbf{B}$$

or more explicitly

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

14 mxv

Using notation similar to **mxm**, matrix vector multiply can be represented by the following mathematical statement

$$\mathbf{c} \ \oplus\!= \ \mathbf{A}^{\mathsf{T}} \oplus.\!\otimes \mathbf{b}$$

where $\mathbf{A} : \mathbb{S}^{m \times n}$, $\mathbf{b} : \mathbb{S}^{n}$, and $\mathbf{c} : \mathbb{S}^{m}$

15 vxm

Using notation similar to **mxm**, vector matrix multiply can be represented by the following mathematical statement

$$\mathbf{c} \ \oplus\!= \ \mathbf{a} \oplus.\!\otimes \mathbf{B}^{\mathsf{T}}$$

where $\mathbf{a} : \mathbb{S}^{m}$, $\mathbf{B} : \mathbb{S}^{m \times n}$, and $\mathbf{c} : \mathbb{S}^{n}$

16 extract: Selecting Sub-Graphs

Selecting sub-graphs is a very common graph operation (see Figure 1.9). The GraphBLAS performs this operation with the **Extract** function by selecting starting vertices (row) and ending vertices (columns) from a matrix $\mathbf{A} : \mathbb{S}^{m \times n}$

$$\mathbf{C}^{\mathsf{T}} \ \oplus\!= \ \mathbf{A}^{\mathsf{T}}(\mathbf{i},\mathbf{j})$$

or more explicitly

$$\mathbf{C}(i,j) = \mathbf{A}(\mathbf{i}(i),\mathbf{j}(j))$$

where $i \in \{1,...,m_C\}$, $j \in \{1,...,n_C\}$, $\mathbf{i} : I^{m_C}$, and $\mathbf{j} : J^{n_C}$ select specific sets of rows and columns in a specific order. The resulting matrix $\mathbf{C} : \mathbb{S}^{m_C \times n_C}$ can be larger or smaller than the input matrix $\mathbf{A}$. **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

$$\mathbf{C} = \mathbf{S}(\mathbf{i})\ \mathbf{A}\ \mathbf{S}^{\mathsf{T}}(\mathbf{j})$$

where $\mathbf{S}(\mathbf{i})$ and $\mathbf{S}(\mathbf{j})$ are selection matrices given by

$$\mathbf{S}(\mathbf{i}) = \mathbb{S}^{m_C \times m}(\{1,...,m_C\},\mathbf{i},1)$$

$$\mathbf{S}(\mathbf{j}) = \mathbb{S}^{n_C \times n}(\{1,...,n_C\},\mathbf{j},1)$$

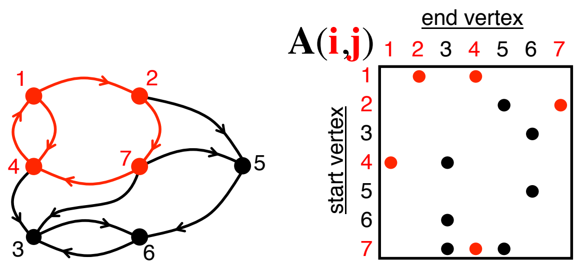

Figure 1.9: Selection of a 4 vertex sub-graph from the adjacency matrix via selecting sub-sets of rows and columns $\mathbf{i} = \mathbf{j} = \{1,2,4,7\}$.

17 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 $\mathbf{C} : \mathbb{S}^{m \times n}$ and assigning new values to them from another sparse matrix, $\mathbf{A} : \mathbb{S}^{m_A \times n_A}$

$$\mathbf{C}^{\mathsf{T}}(\mathbf{i},\mathbf{j})\ \oplus\!=\ \mathbf{A}^{\mathsf{T}}$$

or more explicitly

$$\mathbf{C}(\mathbf{i}(i),\mathbf{j}(j))\ \oplus\!=\ \mathbf{A}(i,j)$$

where $i \in \{1, ..., m_A\}$, $j \in \{1, ..., n_A\}$, $\mathbf{i} : I^{m_A}$ and $\mathbf{j} : J^{n_A}$ select specific sets of rows and columns in a specific order and $\oplus$ optionally allows $\mathbf{A}$ to added to the existing values of $\mathbf{C}$.

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

$$\mathbf{C} \ \oplus\!= \ \mathbf{S}^{\mathsf{T}}(\mathbf{i}) \ \mathbf{A} \ \mathbf{S}(\mathbf{j})$$

where $\mathbf{S}(\mathbf{i})$ and $\mathbf{S}(\mathbf{j})$ are selection matrices given by

$$\mathbf{S}(\mathbf{i}) = \mathbb{S}^{m_A \times m}(\{1, ..., m_A\}, \mathbf{i}, 1)$$

$$\mathbf{S}(\mathbf{j}) = \mathbb{S}^{n_A \times n}(\{1, ..., n_A\}, \mathbf{j}, 1)$$

18 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

$$\mathbf{C}^{\mathsf{T}} \ \oplus\!= \ \mathbf{A}^{\mathsf{T}} \ \oplus \ \mathbf{B}^{\mathsf{T}}$$

where $\mathbf{A}, \mathbf{B}, \mathbf{C} : \mathbb{S}^{m \times n}$ or more explicitly

$$\mathbf{C}(i,j) \ = \ \mathbf{C}(i,j) \ \oplus \ \mathbf{A}(i,j) \ \oplus \ \mathbf{B}(i,j)$$

where $i \in \{1, ..., m\}$, and $j \in \{1, ..., n\}$ 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. **EwiseMult** provides this operation

$$\mathbf{C}^{\mathsf{T}} \ \oplus\!= \ \mathbf{A}^{\mathsf{T}} \ \otimes \ \mathbf{B}^{\mathsf{T}}$$

where $\mathbf{A}, \mathbf{B}, \mathbf{C} : \mathbb{S}^{m \times n}$ or more explicitly

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

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

19 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

$$\mathbf{C} \ \oplus\!= \ f(\mathbf{A})$$

or more explicitly

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

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

Apply can be implemented via **EwiseMult** via

$$\mathbf{C} \ \oplus\!= \ \mathbf{A} \otimes \mathbf{A}$$

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

20 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

$$\mathbf{c} \ \oplus\!= \mathbf{A} \ \oplus.\otimes \ \mathbf{1}$$

or more explicitly

$$\mathbf{c}(i,1) = \mathbf{c}(i,1) \ \oplus \ \bigoplus_{j=1}^{M} \mathbf{A}(i,j)$$

where $\mathbf{c} : \mathbb{S}^{m \times 1}$ and $\mathbf{A} : \mathbb{S}^{m \times n}$, and $\mathbf{1} : \mathbb{S}^{n \times 1}$ 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

$$\mathbf{c} \ \oplus\!= \mathbf{1} \ \oplus.\otimes \ \mathbf{A}$$

or more explicitly

$$\mathbf{c}(1,j) = \mathbf{c}(1,j) \ \oplus \ \bigoplus_{i=1}^{m} \mathbf{A}(i,j)$$

where $\mathbf{c} : \mathbb{S}^{1 \times n}$ and $\mathbf{A} : \mathbb{S}^{m \times n}$, and $\mathbf{1} : \mathbb{S}^{1 \times m}$ is a row vector of all ones.

21 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

$$\mathbf{C} = \mathbf{A} \circledast \mathbf{B} = \begin{pmatrix} \mathbf{A}(1,1) \otimes \mathbf{B} & \mathbf{A}(1,2) \otimes \mathbf{B} & \ldots & \mathbf{A}(1,n_A) \otimes \mathbf{B} \\ \mathbf{A}(2,1) \otimes \mathbf{B} & \mathbf{A}(2,2) \otimes \mathbf{B} & \ldots & \mathbf{A}(2,n_A) \otimes \mathbf{B} \\ \vdots & \vdots & & \vdots \\ \mathbf{A}(m_A,1) \otimes \mathbf{B} & \mathbf{A}(m_A,2) \otimes \mathbf{B} & \ldots & \mathbf{A}(m_A,n_A) \otimes \mathbf{B} \end{pmatrix}$$

where $\mathbf{A} : \mathbb{S}^{m_A \times n_A}$, $\mathbf{B} : \mathbb{S}^{m_B \times n_B}$, and $\mathbf{C} : \mathbb{S}^{m_A m_B \times n_A n_B}$. More explicitly, the Kronecker product can be written as

$$\mathbf{C}((i_A - 1)m_A + i_B, (j_A - 1)n_A + j_B) = \mathbf{A}(i_A, j_A) \otimes \mathbf{B}(i_B, j_B)$$

With the usual accumulation and transpose options, the Kronecker product can be written

$$\mathbf{C}^{\mathsf{T}} \;\; \oplus\!= \mathbf{A}^{\mathsf{T}} \circledast \mathbf{B}^{\mathsf{T}}$$

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

$$(\mathbf{A} \circledast \mathbf{B}) \circledast \mathbf{C} = \mathbf{A} \circledast (\mathbf{B} \circledast \mathbf{C})$$

and element-wise distributivity over addition

$$\mathbf{A} \circledast (\mathbf{B} \oplus \mathbf{C}) = (\mathbf{A} \circledast \mathbf{B}) \oplus (\mathbf{A} \circledast \mathbf{C})$$

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

$$(\mathbf{A} \circledast \mathbf{B})(\mathbf{C} \circledast \mathbf{D}) = (\mathbf{AC}) \circledast (\mathbf{BD})$$

22 Graph Algorithms and Diverse Semirings

The ability to change $\oplus$ and $\otimes$ operations allows different graph algorithms to be implemented using the same element-wise addition, element-wise multiplication, and matrix multiplication operations. Different semirings are best suited for certain classes of graph algorithms. The pattern of non-zero entries resulting from breadth-first-search illustrated in Figure 1.7 is generally preserved for various semirings. However, the non-zero values assigned to the edges and vertices can be very different and enable different graph algorithms.

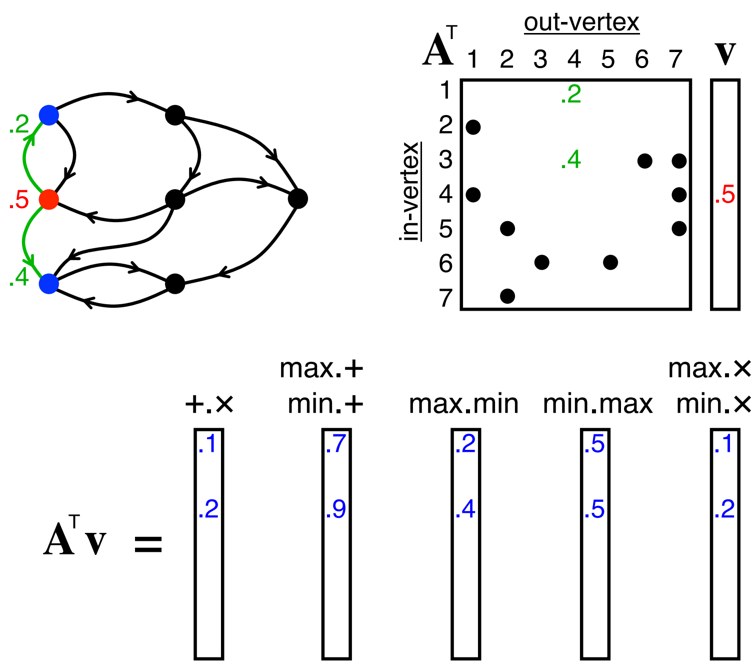

Figure 1.10: (top left) One-hop breadth-first-search of a weighted graph starting at vertex 4 and traversing to vertices 1 and 3. (top right) Matrix representation of the weighted graph and vector representation of the starting vertex. (bottom) Matrix-vector multiplication of the adjacency matrix of a graph performs the equivalent operation. Different pairs of operations $\oplus$ and $\otimes$ produce different results. For display convenience, operator pairs that produce the same values *in this specific example* are stacked.

Figure 1.10 illustrates performing a single-hop breadth-first-search using seven semirings ($+.\times$, $\max.+$, $\min.+$, $\max.\min$, $\min.\max$, $\max.\times$, and $\min.\times$). For display convenience, operator pairs that produce the same result *in this specific example* are stacked. In Figure 1.10 the starting vertex 4 is assigned a value of .5 and the edges to its vertex neighbors 1 and 3 are assigned values of .2 and .4. Empty values are assumed to be the corresponding 0 of the operator pair. In all cases, the pattern of non-zero entries of the results are the same. In each case, because there is only one path

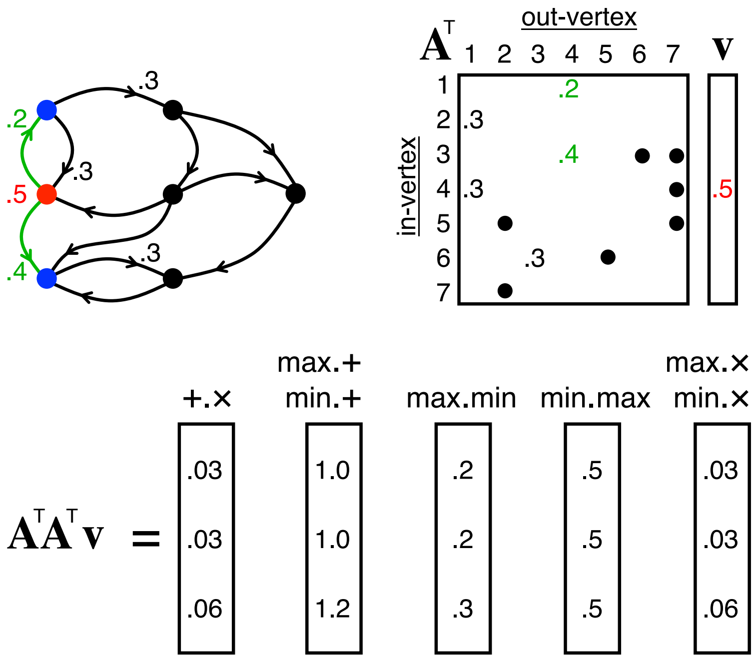

Figure 1.11: (top left) Two-hop breadth-first-search of a weighted graph starting at vertex 4 and traversing to vertices 1 and 3 and continues on to vertices 2, 4, and 6. (top right) Matrix representation of the weighted graph and vector representation of the starting vertex. (bottom) Matrix-vector multiplication of the adjacency matrix of a graph performs the equivalent operation. Different pairs of operations $\oplus$ and $\otimes$ produce different results. For display convenience, operator pairs that produce the same result *in this specific example* are stacked.

from vertex 4 to vertex 1 and from vertex 4 to vertex 3, the only effect of the $\otimes$ of operator is to compare the non-zero output the $\otimes$ operator with the 0. Thus, the differences between the $\oplus$ operators have no impact in this specific example because for any values of a

$$a \oplus 0 = 0 \oplus a = a$$

The graph algorithm implications of different $\oplus.\otimes$ operator pairs is more clearly seen in the two-hop breadth-first-search. Figure 1.11 illustrates graph traversal that starts at vertex 4, goes to vertices 1 and 3, and then continues on to vertices 2, 4, and 6. For simplicity, the additional edge weights are assigned values of .3. The first operator pair $+.\times$ provides the product of all the weights of all paths from the starting vertex to each ending vertex. The $+.\times$ semiring is valuable for determining the strengths of all the paths between the starting and ending vertices. In this example, there is only one path between the starting vertex and the ending vertices and so $+.\times$ and

$\max.\times$ and $\min.\times$ all produce the same results. If there were multiple paths between the start and end vertices then $\oplus$ would operate on more than one non-zero value and the differences would be apparent. Specifically, $+.\times$ combines all paths while $\max.\times$ and $\min.\times$ selects either the minimum or the maximum path. Thus, these different operator pairs represent different graph algorithms. One algorithm produces a value that combines all paths while the other algorithm produces a value that is derived from a single path.

A similar pattern can be seen among the other operator pairs. $\max.+$ and $\min.+$ compute the sum of the weights along each path from the starting vertex to each ending vertex and then selects the largest (or smallest) weighted path. $\max.\min$ and $\min.\max$ compute the minimum (or maximum) of the weights along each path from the starting vertex to each end vertex and then selects the largest (or smallest) weighted path.

A wide range of breadth-first-search weight computations can be performed via matrix multiplication with different operator pairs. A synopsis of the types of calculations illustrated in Figures 1.10 and 1.11 is as follows

$+.\times$ sum of products of weights along each path; computes the strength of all connections between the starting vertex and the ending vertices.

$\max.\times$ maximum of products of weights along each path; computes the longest product of all of the connections between the starting vertex and the ending vertices.

$\min.\times$ minimum of products of weights along each path; computes the shortest product of all of the connections between the starting vertex and the ending vertices.

$\max.+$ maximum of sum of weights along each path; computes the longest sum of all of the connections between the starting vertex and the ending vertices.

$\min.+$ minimum of sum of weights along each path; computes the shortest sum of all of the connections between the starting vertex and the ending vertices.

$\max.\min$ maximum of minimum of weight along each path; computes the longest of all the shortest connections between the starting vertex and the ending vertices.

$\min.\max$ minimum of maximum of weight along each path; computes the shortest of all the longest connections between the starting vertex and the ending vertices.