

LAGraph Algorithms

PDF build on Thursday 11th February, 2021 at 00:52 UTC

Abstract

Theoretical documentation for LAGraph.

1 Introduction

The goal of this document is to present a notation for GraphBLAS algorithms and showcase it using important GraphBLAS algorithms.

2 The GraphBLAS

Goal The goal of GraphBLAS is to create a layer of abstraction between the graph algorithms and the graph analytics framework, separating the concerns of the algorithm developers from those of the framework developers and hardware designers. To achieve this, it builds on the theoretical framework of matrix operations on arbitrary semirings [3], which allows defining graph algorithms in the language of linear algebra [4]. To ensure portability, the GraphBLAS standard defines a C API that can be implemented on a variety of hardware including GPUs.

Data structures A graph with n vertices can be stored as a square adjacency matrix $\mathbf{A} \in \text{uint}^{n \times n}$, where rows and columns both represent vertices of the graph and element $A(i, j)$ contains the number of edges from vertex i to vertex j . If the graph is undirected, the matrix is symmetric.

Navigation The fundamental step in GraphBLAS is the multiplication of an adjacency matrix with another matrix or vector over a selected semiring. For example, the operation `HasMember lor.land IsLocatedIn` computed over the “*logical or.logical and*” semiring returns a matrix representing the Places where a Forum’s members are located in. Meanwhile, when computed over the conventional arithmetic “*plus.times*” semiring, `HasMember +. × IsLocatedIn` also returns the number of such Persons. A traversal from a certain set of vertices can be expressed by using a boolean vector \mathbf{f} (often referred to as the *frontier*, *wavefront*, or *queue*) and setting `true` values for the elements corresponding to source vertices. For example, for Forums $\mathbf{f} \in \text{bool}^{|\text{forums}|}$, `f lor.land HasMember` returns the Persons who belong to any of the forums in \mathbf{f} . The BFS navigation step can also be captured using other semirings such as `lor.first`, where `first(x, y) = x` ; `lor.second`, where `second(x, y) = y` ; and `any.pair`, where `any(x, y)` returns either x or y , and `pair(x, y) = 1` [2].

2.1 Notation

Table 1 contains the notation of GraphBLAS operations. Additionally, we use $\mathbf{D} = \text{diag}(\mathbf{j}, \mathbf{n})$ to construct a diagonal matrix $\mathbf{D} \leftarrow \{\mathbf{j}, \mathbf{j}, [1, 1, \dots, 1]\}$. The elements of the matrix are $D(\mathbf{j}, \mathbf{j}) = 1$ for $\mathbf{j} \in \mathbf{j}$.

2.1.1 Masks

Masks $\mathbf{C} \leftarrow \mathbf{M}$ and $\mathbf{w} \leftarrow \mathbf{m}$ are used to selectively write to the result matrix/vector. The complements of the masks can be selected with the negation symbol, denoted with: $\mathbf{C} \leftarrow !\mathbf{M}$ and $\mathbf{w} \leftarrow !\mathbf{m}$, respectively.

Masks with “replace” semantics (annihilating all elements outside the mask) are denoted with

- $\mathbf{C} \leftarrow \mathbf{M}$
- $\mathbf{C} \leftarrow !\mathbf{M}$
- $\mathbf{w} \leftarrow \mathbf{m}$
- $\mathbf{w} \leftarrow !\mathbf{m}$

The structure of the mask is denoted with:

op./method	name	notation
mxm	matrix-matrix multiplication	$\mathbf{C} \langle \mathbf{M} \rangle + = \mathbf{A} + . \times \mathbf{B}$
vxm	vector-matrix multiplication	$\mathbf{w} \langle \mathbf{m} \rangle + = \mathbf{u} + . \times \mathbf{A}$
mxv	matrix-vector multiplication	$\mathbf{w} \langle \mathbf{m} \rangle + = \mathbf{A} + . \times \mathbf{u}$
eWiseAdd	element-wise addition	$\mathbf{C} \langle \mathbf{M} \rangle + = \mathbf{A} + \mathbf{B}$
	set union of patterns	$\mathbf{w} \langle \mathbf{m} \rangle + = \mathbf{u} + \mathbf{v}$
eWiseMult	element-wise multiplication	$\mathbf{C} \langle \mathbf{M} \rangle + = \mathbf{A} \times \mathbf{B}$
	set intersection of patterns	$\mathbf{w} \langle \mathbf{m} \rangle + = \mathbf{u} \times \mathbf{v}$
extract	extract submatrix	$\mathbf{C} \langle \mathbf{M} \rangle + = \mathbf{A}(\mathbf{i}, \mathbf{j})$
	extract column vector	$\mathbf{w} \langle \mathbf{m} \rangle + = \mathbf{A}(:, \mathbf{j})$
	extract row vector	$\mathbf{w} \langle \mathbf{m} \rangle + = \mathbf{A}(\mathbf{i}, :)$
	extract subvector	$\mathbf{w} \langle \mathbf{m} \rangle + = \mathbf{u}(\mathbf{i})$
assign	assign matrix to submatrix with mask for \mathbf{C}	$\mathbf{C} \langle \mathbf{M} \rangle (\mathbf{i}, \mathbf{j}) + = \mathbf{A}$
	assign scalar to submatrix with mask for \mathbf{C}	$\mathbf{C} \langle \mathbf{M} \rangle (\mathbf{i}, \mathbf{j}) + = s$
	assign vector to subvector with mask for \mathbf{w}	$\mathbf{w} \langle \mathbf{m} \rangle (\mathbf{i}) + = \mathbf{u}$
	assign scalar to subvector with mask for \mathbf{w}	$\mathbf{w} \langle \mathbf{m} \rangle (\mathbf{i}) + = s$
subassign (GxB)	assign matrix to submatrix with submask for $\mathbf{C}(\mathbf{i}, \mathbf{j})$	$\mathbf{C}(\mathbf{i}, \mathbf{j}) \langle \mathbf{M} \rangle + = \mathbf{A}$
	assign scalar to submatrix with submask for $\mathbf{C}(\mathbf{i}, \mathbf{j})$	$\mathbf{C}(\mathbf{i}, \mathbf{j}) \langle \mathbf{M} \rangle + = s$
	assign vector to subvector with submask for $\mathbf{w}(\mathbf{i})$	$\mathbf{w}(\mathbf{i}) \langle \mathbf{m} \rangle + = \mathbf{u}$
	assign scalar to subvector with submask for $\mathbf{w}(\mathbf{i})$	$\mathbf{w}(\mathbf{i}) \langle \mathbf{m} \rangle + = s$
apply	apply unary operator	$\mathbf{C} \langle \mathbf{M} \rangle + = \mathbf{f}((\mathbf{A}))$
		$\mathbf{w} \langle \mathbf{m} \rangle + = \mathbf{f}((\mathbf{u}))$
select	apply select operator	$\mathbf{C} \langle \mathbf{M} \rangle + = \mathbf{A} \langle \mathbf{f}((\mathbf{A}), \mathbf{k}) \rangle$
		$\mathbf{w} \langle \mathbf{m} \rangle + = \mathbf{u} \langle \mathbf{f}((\mathbf{u}), \mathbf{k}) \rangle$
reduce	reduce matrix to column vector	$\mathbf{w} \langle \mathbf{m} \rangle + = [\mathbf{A}]$
	reduce matrix to scalar	$s + = [\mathbf{A}]$
	reduce vector to scalar	$s + = [\mathbf{u}]$
transpose	transpose	$\mathbf{C} \langle \mathbf{M} \rangle + = \mathbf{A}'$
kroncker	Kronecker multiplication	$\mathbf{C} \langle \mathbf{M} \rangle + = \text{kron}(\mathbf{A}, \mathbf{B})$
new	new matrix	$\mathbf{A} = \text{TYPEPRECISION}(\mathbf{n}, \mathbf{m})$
	new vector	$\mathbf{u} = \text{TYPEPRECISION}(\mathbf{n})$
build	matrix from tuples	$\mathbf{C} \leftarrow \{\mathbf{i}, \mathbf{j}, \mathbf{x}\}$
	vector from tuples	$\mathbf{w} \leftarrow \{\mathbf{i}, \mathbf{x}\}$
extractTuples	extract index/value arrays	$\{\mathbf{i}, \mathbf{j}, \mathbf{x}\} \leftarrow \mathbf{A}$ $\{\mathbf{i}, \mathbf{x}\} \leftarrow \mathbf{u}$
dup	duplicate matrix	$\mathbf{C} \leftarrow \mathbf{A}$
	duplicate vector	$\mathbf{w} \leftarrow \mathbf{u}$
extractElement	extract scalar element	$s = \mathbf{A}(\mathbf{i}, \mathbf{j})$
		$s = \mathbf{u}(\mathbf{i})$
setElement	set element	$\mathbf{C}(\mathbf{i}, \mathbf{j}) = s$ $\mathbf{w}(\mathbf{i}) = s$

Table 1: GraphBLAS operations and methods based on [1]. *Notation:* Matrices and vectors are typeset in bold, starting with uppercase (\mathbf{A}) and lowercase (\mathbf{u}) letters, respectively. Scalars including indices are lowercase italic (k, i, j) while arrays are lowercase bold italic ($\mathbf{x}, \mathbf{i}, \mathbf{j}$). $+$ and \times are the addition and multiplication operators forming a semiring and default to conventional arithmetic $+$ and \times operators. $+$ is the accumulator operator.

- $C < \{M\} >$
- $C < !\{M\} >$
- $w < \{m\} >$
- $w < !\{m\} >$

Combining structure and replace semantics is possible:

- $C << \{M\} >>$
- $C << !\{M\} >>$
- $w << \{m\} >>$
- $w << !\{m\} >>$

Initializing scalars, vectors, and matrices (GraphBLAS methods):

- $s = \text{fp64}()$
- $w = \text{fp32}(n)$
- $A = \text{uint16}(m, n)$
- $A = \text{int64}(k, m)$

3 Algorithms

LAGraph [5] implements graph algorithms using the GraphBLAS C API [6].

Here are a few algorithms that could be included in this document:

Algorithm 1: BFS / Levels variant.

Input: $A, n, \text{startVertex}$

```

1 Function BFS
2    $f(\text{startVertex}) = T$ 
3   for  $\text{level} = 1$  to  $n - 1$  do
4      $s < f > = \text{level}$ 
5      $f << !s >> = fA$ 

```

Algorithm 2: BFS / Parents variant.

Input: $A, n, \text{startVertex}$

```

1 Function ParentsBFS
2    $f(\text{startVertex}) = 0$ 
3   for  $\text{level} = 1$  to  $n - 1$  do
4      $s < f > = f$ 
5      $f << !s >> = f \text{ any.first} j1 \ A$ 

```

Algorithm 3: Direction Optimizing (push/pull) BFS.

Input: $A, A', n, \text{startVertex}$

```
1 Function DirectionOptimizingBFS
2    $f(\text{startVertex}) = T$ 
3   for level = 1 to  $n - 1$  do
4      $s \langle f \rangle = \text{level}$ 
5     if Push( $A, f$ ) then
6        $f \ll !s \gg = fA$ 
7     else
8        $f \ll !s \gg = A'f$ 
```

Algorithm 4: Multi-source breadth-first search.

Data: ...
Result: ...

```
1 Function MSBFS
2   Frontier = diag(sources, n)
3   for level = 1 to  $n - 1$  do
4     Seen  $\langle$  Frontier  $\rangle = \text{level}$ 
5     Frontier  $\ll !\text{Seen} \gg = \text{Frontier any.pair } A$ 
```

Algorithm 5: All-pairs shortest distance (on undirected, unweighted graphs) [7].

Data: ...
Result: ...

```
1 Function APD( $A, n, \text{deg}$ )
2    $Z = A$ 
3    $Z += A + . \times A$ 
4    $B = \{ \langle Z, \text{offdiag} \rangle \}$  // use the pattern as a Boolean matrix
5   if  $A == B$  then
6     return  $A$ 
7    $T = \text{APD}(B, n, \text{deg})$ 
8    $X = T + . \times A$ 
9    $T_{\text{scaled}} = T + . \times \text{diag}((\text{deg}))$ 
10   $X_{\text{filtered}} = \langle X, X \langle T_{\text{scaled}} \rangle$ 
11  return  $(2 \times T) \text{ MINUS } X_{\text{filtered}}$ 
12 Function APD( $A$ )
13    $\text{deg} = [+A]$ 
14    $\text{Distance} = \text{APD}(A, n, \text{deg})$ 
15    $\text{sp} = [+ \text{Distance}]$ 
```

Algorithm 6: Betweenness centrality.

```
1 Function MSBFS
  // The NumSp structure holds the number of shortest paths for each node and starting
  // vertex discovered so far.
  // Initialized to source vertices.
2 NumSp <- {s, [1, 1, ..., 1]}
  // The Frontier holds the number of shortest paths for each node and starting vertex
  // discovered so far.
  // Initialized to source vertices.
3 Frontier <NumSp>= A(s,:)
4 d = 0
  // The Sigmas matrices store frontier information for each level of the BFS phase.
  // BFS phase (forward sweep)
5 do
  // Sigmas[d](:,s) = dth level frontier from source vertex s
6   Sigmas[d] = bool(n, nsver)
7   Sigmas[d](:, :) = Frontier // Convert matrix to Boolean
8   NumSp = NumSp + Frontier // Accumulate path counts
9   Frontier <<NumSp>>= A' + . × Frontier // Update frontier
10 while nvals(Frontier) > 0
11 NumSpInv = fp32(n, nsver)
12 NumSpInv = 1.0 DIV NumSp
13 BCU = fp32(n, nsver)
14 BCU(:) = 1.0 // Make BCU dense, initialize all elements to 1.0
15 W = fp32(n, nsver)
  // Tally phase (backward sweep)
16 for i = d - 1 downto 0 do
17   W << Sigmas[i] >>= NumSpInv DIV BCU
18   W << Sigmas[i - 1] >>= A + . × W // Add contributions by successors and mask with that
   // BFS level's frontier.
19   BCU += W × NumSp
  // Row reduce BCU and subtract nsver from every entry to account for 1 extra value
  // per BCU row element
20 delta = [+ BCU]
21 delta MINUS = nsver
```

Algorithm 7: PageRank (used in Graphalytics).

Data: alpha constant (damping factor)

Result: ...

```
1 Function PageRank
2 pr(:) = 1/n
3 outdegrees = [+j A(:, j)]
4 for k = 1 to numIterations do
5   importance = pr DIV outdegrees
6   importance = times(importance, alpha) // apply the times((x,s) = x · s operator
7   importance = importance + . × A
8   danglingVertexRanks < !outdegrees >= pr(:)
9   totalDanglingRank = (alpha)/(n) × [+ danglingVertexRanks]
10  pr = (1 - alpha)/(n) + totalDanglingRank
11  pr = pr + importance
```

Algorithm 8: Algebraic Bellman-Ford.

```
1 Function SSSP
2    $d(s) = 0$ 
3   for  $k = 1$  to  $n - 1$  do
4      $d' = d \text{ min.plus } A$ 
5     if  $d' == d$  then break
6    $d \leftarrow d'$ 
```

Algorithm 9: Delta-stepping SSSP.

Data:

$A, A_H, A_L \in \text{fp}(|V|, |V|)$
 $s, i \in \text{uint}()$
 $\Delta \in \text{fp}()$
 $t, t_{\text{Req}} \in \text{fp}(|V|)$
 $t_{B_i}, e \in \text{uint}(|V|)$

```
1 Function DeltaStepping
2    $A_L = \langle 0 < A \leq \Delta \rangle$ 
3    $A_H = \langle \Delta < A \rangle$ 
4    $t(:) = \infty$ 
5    $t(s) = 0$ 
6   while  $\text{nvals}(\langle i\Delta \leq t \rangle) \neq 0$  do
7      $s = 0$ 
8      $t_{B_i} = \langle i\Delta \leq t < (i+1)\Delta \rangle$ 
9     while  $t_{B_i} \neq 0$  do
10       $t_{\text{Req}} = A_L' + . \times (t \times t_{B_i})$ 
11       $e = \langle 0 < e + t_{B_i} \rangle$ 
12       $t_{B_i} = \langle i\Delta \leq t_{\text{Req}} < (i+1)\Delta \rangle \times (t_{\text{Req}} \min_+ t)$ 
13       $t_{B_i} = \langle i\Delta \leq t_{\text{Req}} < (i+1)\Delta \rangle \times (t_{\text{Req}} +_{\min} t)$ 
14       $t_{B_i} = \langle i\Delta \leq t_{\text{Req}} < (i+1)\Delta \rangle \times (t_{\text{Req}} \min_+ t)$ 
15       $t = t \min t_{\text{Req}}$ 
16       $t_{\text{Req}} = A_H' + . \times (t \times e)$ 
17       $t = t \min t_{\text{Req}}$ 
18       $i = i + 1$ 
```

Algorithm 10: All-pairs shortest path (Floyd–Warshall algorithm).

```
1 Function FloydWarshall
2    $D \leftarrow A$ 
3   for  $k = 1$  to  $n$  do
4      $D = D \min [D(:, k) \text{ min.plus } (D(k, :))]$ 
```

Algorithm 11: FastSV algorithm.

```
1 Function FastSV
2   n = nrow(A)
3   gf = f
4   dup = gf
5   mngf = gf
6   {i,x} <- f
7   repeat
8     // Step 1: Stochastic hooking
9     mngf = mngf min A
10    mngf = mngf second.min gf
11    f(x) = f min mngf
12    // Step 2: Aggressive hooking
13    f = f min mngf
14    // Step 3: Shortcutting
15    f = f min gf
16    // Step 4: Calculate grandparents
17    {i,x} <- f
18    gf = f(x)
19    // Step 5: Check termination
20    diff = dup ≠ gf
21    sum = [+i diff(i)]
22    dup = gf
23  until sum == 0
```

Algorithm 12: Triangle count (Cohen's algorithm).

```
1 Function TriangleCount
2   L = tril(A)
3   U = triu(A)
4   B = L + . × U
5   C = B × A
6   t = [+ C]/2
```

Algorithm 13: Triangle count (Sandia).

```
1 Function TriangleCount
2   L = tril(A)
3   C <L>= L + . × L
4   t = [+ C]
```

Algorithm 14: Triangle count (FLAME).

```
1 Function TriangleCountFlame
2   for i = 2 to n - 1 do
3     A20 = A(i + 1 : n, 0 : i - 1)
4     a10 = A(0 : i - 1, i)
5     a12 = A(i, i + 1 : n)
6     t += a10 + . × A20 + . × a12
```

Algorithm 15: Local clustering coefficient.

```
1 Function PageRank
2   Tri <A>= A + . × A // compute triangle count matrix
3   tri = [+ Tri] // reduce to triangle count vector
4   deg = [+ A] // reduce to vertex degree vector
5   wed = perm2((deg)) // apply perm2((x)) = x · (x - 1) to get wedge count vector
6   lcc = tri DIV wed // LCC vector
```

Algorithm 16: k -truss algorithm.

```
1 Function KTruss
2   C <- A
3   nonzeros <- nvals(C)
4   for i = 1 to n - 1 do
5     C <C>= C + .1 and C
6     C = <C ≥ k - 2>
7     if nonzeros == nvals(C) then
8       break
9     nonzeros <- nvals(C)
```

Algorithm 17: Louvain algorithm (WIP).

```
1 Function Louvain
2   G + = G'
3   k = [+ A]
4   m = (1)/(2)[+ k]
5   S <- I
6
7   vertices_changed <- nvals(k)
8   while vertices_changed > 0 do
9     for j ∈ range(|V|) do
10      v = G(j, :)
11      tq = v any.pair S
12      sr = S(j, :)
13      S(j, :) = empty
14
15      q <- k
16      q <k> × = - k(j)/m
17      q + = v
18      q1 <tq>= q + . × S
19
20      t = (q1 == [max q1])
21      while nvals(t) ≠ 1 do
22        p = random() × t
23        t = (p == [max p])
24      S(j, :) = t
25
26      if nvals(sr × t) == 0 then
27        vertices_changed = nvals(k)
28      vertices_changed = vertices_changed - 1
```

Algorithm 18: Community detection using label propagation (for undirected graphs).

```
1 Function CDLP
2   L <- diag([0, 1, ..., n - 1])
3   for k = 1 to t do
4     F = A any . second L // Frequency matrix
5     {i, -, x} <- F
6     merge_sort_pairs(i, x)
7     labels = for each row in i, select min mode value from x
```

References

- [1] T. A. Davis, “Algorithm 1000: SuiteSparse:GraphBLAS: Graph algorithms in the language of sparse linear algebra,” *ACM Trans. Math. Softw.*, 2019. [Online]. Available: <https://doi.org/10.1145/3322125>
- [2] ——. (2020) User guide for SuiteSparse:GraphBLAS, version 3.3.3. <https://people.engr.tamu.edu/davis/GraphBLAS.html>.
- [3] J. Kepner *et al.*, “Mathematical foundations of the GraphBLAS,” in *HPEC*. IEEE, 2016. [Online]. Available: <https://doi.org/10.1109/HPEC.2016.7761646>
- [4] J. Kepner and J. R. Gilbert, Eds., *Graph Algorithms in the Language of Linear Algebra*. SIAM, 2011. [Online]. Available: <https://doi.org/10.1137/1.9780898719918>
- [5] T. Mattson *et al.*, “LAGraph: A community effort to collect graph algorithms built on top of the GraphBLAS,” in *GrAPL at IPDPS*, 2019. [Online]. Available: <https://doi.org/10.1109/IPDPSW.2019.00053>
- [6] T. G. Mattson *et al.*, “GraphBLAS C API: Ideas for future versions of the specification,” in *HPEC*. IEEE, 2017. [Online]. Available: <https://doi.org/10.1109/HPEC.2017.8091095>
- [7] R. Seidel, “On the all-pairs-shortest-path problem in unweighted undirected graphs,” *J. Comput. Syst. Sci.*, vol. 51, no. 3, pp. 400–403, 1995. [Online]. Available: <https://doi.org/10.1006/jcss.1995.1078>