LAGraph Algorithms

LAGraph Working Group

November 4, 2020

Abstract

Theoretical documentation for LAGraph.

1 Notation

Table 1 contains the notation of GraphBLAS operations

Additionally, we use $\texttt{D} = diag(\texttt{J}, \texttt{n})$ to construct a diagonal matrix $\texttt{D} \texttt{<-} \{\texttt{J}, \texttt{J}, [1, 1, \ldots, 1]\}$. The elements of the matrix are $\texttt{D}(\texttt{j}, \texttt{j}) = 1$ for $\texttt{j} \in \texttt{J}$.

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

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

2 Algorithms

LAGraph [12] implements graph algorithms using the GraphBLAS C API [13].

An incomplete list of GraphBLAS algorithms:

- MSBFS [8], bidirectional BFS [8], pushpull BFS [19]
- DFS [16]
- weakly connected components [21]
- SCC (LAGraph)
- SSSP, delta-stepping [17]
- triangle count [1, 5], triangle enumeration [1], item local clustering coefficient [4]
- k-truss [5]
- betweenness centrality [12]
- closeness centrality [8]
- DNN algorithm [10, 7]
- PageRank variants (at least 2) [4], IISWC paper, ...
- Louvain [11]
- property graphs: incremental TTC case [9], SIGMOD 2014 Contest [8] Roi Lipman's talk[1]
- CFPQs based on a string of GRADES/ADBIS/other papers [2, 3, 14, 18, 15]
- Implementations: GBTL[2], SuiteSparse:GraphBLAS [6], GraphBLAST [20]

[1] http://wiki.ldbcouncil.org/pages/viewpage.action?pageId=106233859&preview=/106233859/111706128/LDBC-July-2019.pdf

[2] https://github.com/cmu-sei/gbtl

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

Table 1: GraphBLAS operations and methods based on [6]. *Notation:* Matrices and vectors are typeset in bold, starting with uppercase (A) and lowercase (u) letters, respectively. Scalars including indices are lowercase italic (s, i, j) while arrays are uppercase italic (X, I, J). + and × are the addition and multiplication operators forming a semiring and default to conventional arithmetic + and × operators. ⊙ is the apply operator. Masks <M> and <m> are used to selectively write to the result matrix/vector. The complements of masks <M>, <m> can be selected with the negation symbol, denoted with <!M>, <!m>, respectively. Masks with "replace" semantics (annihilating all elements outside the mask) are denoted with <<M>>, <<!M>>, <<m>>, and <<!m>>, The structure of the mask is denoted with <{M}>, <!{M}>, <{m}>, and <!{m}>.

Algorithm 1: Breadth-first search.

```
1 Function BFS
2     frontier(s) = TRUE
3     for level = 1 to n − 1 do
4         seen <frontier>= level
5         frontier <<!seen>>= frontier any.pair A
```

Algorithm 2: Breadth-first search (push/pull).

```
1 Function BFS
2     frontier(s) = TRUE
3     for level = 1 to n − 1 do
4         seen <frontier>= level
5         push = use some heuristics to determine whether to push/pull
6         if push then
7             frontier <<!seen>>= frontier any.pair A
8         else
9             frontier <<!seen>>= A any.pair frontier
```

Algorithm 3: Multi-source breadth-first search.

Data: ...
Result: ...

```
1 Function MSBFS
2     Frontier <- diag(S, n)
3     for level = 1 to n − 1 do
4         Seen <Frontier>= level
5         Frontier <<!Seen>>= Frontier any.pair A
```

Algorithm 4: Betweenness centrality.

```
Function MSBFS
    // The NumSp structure holds the number of shortest paths for each node and starting
       vertex discovered so far.
    // Initialized to source vertices.
    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.
    Frontier < NumSp >= A(s, :)
    d = 0
    // The Sigmas matrices store frontier information for each level of the BFS phase.
    // BFS phase (forward sweep)
    do
        // Sigmas[d](:, s) = d^th level frontier from source vertex s
        Sigmas[d] = bool(n, nsver)
        Sigmas[d](:, :) = Frontier                                  // Convert matrix to Boolean
        NumSp = NumSp + Frontier                                    // Accumulate path counts
        Frontier << NumSp >>= A' +. × Frontier                      // Update frontier
    while nvals(Frontier) > 0
    NumSpInv = fp32(n, nsver)
    NumSpInv = 1.0 DIV NumSp
    BCU = fp32(n, nsver)
    BCU(:) = 1.0                          // Make BCU dense, initialize all elements to 1.0
    W = fp32(n, nsver)
    // Tally phase (backward sweep)
    for i = d − 1 downto 0 do
        W << Sigmas[i] >>= NumSpInv DIV BCU
        W << Sigmas[i − 1] >>= A +. × W // Add contributions by successors and mask with that
         BFS level's frontier.
        BCU += W × NumSp
    // Row reduce BCU and subtract nsver from every entry to account for 1 extra value
       per BCU row element
    delta = [+ BCU]
    delta MINUS = nsver
```

Algorithm 5: PageRank (used in Graphalytics).

```
Data: alpha constant (damping factor)
Result: ...
Function PageRank
    pr(:) = 1/n
    outdegrees = [+_j A(:, j)]
    for k = 1 to numIterations do
        importance = pr DIV outdegrees
        importance = times(importance, alpha)          // apply the times(x, s) = x · s operator
        importance = importance +. × A
        danglingVertexRanks < !outdegrees >= pr(:)
        totalDanglingRank = [+ danglingVertexRanks(i)](alpha)/(n)
        pr(:) = (1 − alpha)/(n) + totalDanglingRank
        pr < importance >= pr + importance
```

Algorithm 6: Algebraic Bellman-Ford for SSSP.

Function *SSSP*
- $\mathtt{d}(\mathtt{s}) = 0$
- **for** $\mathtt{k} = 1$ **to** $\mathtt{n} - 1$ **do**
  - $\mathtt{dtmp} = \mathtt{d}\ \mathtt{min.plus}\ \mathtt{A}$
  - **if** $\mathtt{dtmp} = \mathtt{d}$ **then**
    - **break**
  - $\mathtt{d} \leftarrow \mathtt{dtmp}$

Algorithm 7: Delta-stepping SSSP.

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

Function *DeltaStepping*
- $\mathtt{A_L} = \mathit{select}(0 < \mathtt{A} \leq \Delta)$
- $\mathtt{A_H} = \mathit{select}(\Delta < \mathtt{A})$
- $\mathtt{t}(:) = \infty$
- $\mathtt{t}(\mathtt{s}) = 0$
- **while** $\mathit{nvals}(\mathit{select}(\mathtt{i}\Delta \leq \mathtt{t})) \neq 0$ **do**
  - $\mathtt{s} = 0$
  - $\mathtt{t_{B_i}} = \mathit{select}(\mathtt{i}\Delta \leq \mathtt{t} < (\mathtt{i}+1)\Delta)$
  - **while** $\mathtt{t_{B_i}} \neq 0$ **do**
    - $\mathtt{t_{Req}} = \mathtt{A_L}\text{'} +. \times (\mathtt{t} \times \mathtt{t_{B_i}})$
    - $\mathtt{e} = \mathit{select}(0 < \mathtt{e} + \mathtt{t_{B_i}})$
    - $\mathtt{t_{B_i}} = \mathit{select}(\mathtt{i}\Delta \leq \mathtt{t_{Req}} < (\mathtt{i}+1)\Delta) \times (\mathtt{t_{Req}} \underset{+}{\mathtt{min}}\, \mathtt{t})$
  - $\mathtt{t_{Req}} = \mathtt{A_H}\text{'} +. \times (\mathtt{t} \times \mathtt{e})$
  - $\mathtt{t} = \mathtt{t}\ \mathtt{min}\ \mathtt{t_{Req}}$
  - $\mathtt{i} = \mathtt{i} + 1$

Algorithm 8: All-pairs shortest path (Floyd–Warshall algorithm).

Function *FloydWarshall*
- $\mathtt{D} \leftarrow \mathtt{A}$
- **for** $\mathtt{k} = 1$ **to** $\mathtt{n}$ **do**
  - $\mathtt{D} = \mathtt{D}\,\mathtt{min}[\mathtt{D}(:, \mathtt{k})\ \mathtt{min.plus}\ (\mathtt{D}(\mathtt{k}, :)]$

Algorithm 9: FastSV algorithm.

```
Function FastSV
    n = nrows(A)
    gf = f
    dup = gf
    mngf = gf
    {I, X} <- f
    repeat
        // Step 1: Stochastic hooking
        mngf = mngf min A
        mngf = mngf second.min gf
        f(X) = f min mngf
        // Step 2: Aggressive hooking
        f = f min mngf
        // Step 3: Shortcutting
        f = f min gf
        // Step 4: Calculate grandparents
        {I, X} <- f
        gf = f(X)
        // Step 5: Check termination
        diff = dup ≠ gf
        sum = [+_i diff(i)]
        dup = gf
    until sum == 0
```

Algorithm 10: Triangle count (Cohen's algorithm).

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

Algorithm 11: Triangle count (Sandia).

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

Algorithm 12: Triangle count (FLAME).

```
Function TriangleCountFlame
    for i = 2 to n − 1 do
        A_20 = A(i + 1 : n, 0 : i − 1)
        a_10 = A(0 : i − 1, i)
        a_12 = A(i, i + 1 : n)
        t += a_10 +.× A_20 +.× a_12
```

Algorithm 13: Local clustering coefficient.

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

Algorithm 14: k-truss algorithm.

```
Function KTruss
    C <- A
    nonzeros <- nvals(C)
    for i = 1 to n − 1 do
        C <C>= C +. land C
        C = select(C ≥ k − 2)
        if nonzeros == nvals(C) then
            break
        nonzeros <- nvals(C)
```

Algorithm 15: Louvain algorithm (WIP).

```
Function Louvain
    G += G'
    k = [+ A]
    m = (1)/(2)[+ k]
    S <- I

    vertices_changed <- nvals(k)
    while vertices_changed > 0 do
        for j ∈ range(|V|) do
            v = G(j, :)
            t_q = v any.pair S
            sr = S(j, :)
            S(j, :) = empty

            q <- k
            q <k> × = − k(j)/m
            q += v
            q_1 <t_q>= q +. × S

            t = (q_1 == [max q_1])
            while nvals(t) ≠ 1 do
                p = random() × t
                t = (p == [max p])
            S(j, :) = t

            if nvals(sr × t) == 0 then
                vertices_changed = nvals(k)
            vertices_changed = vertices_changed − 1
```

Algorithm 16: Community detection using label propagation (for undirected graphs).

```
Function CDLP
    L <- diag([0, 1, . . . , n − 1])
    for k = 1 to t do
        F = A any . second L                                  // Frequency matrix
        {I, _, X} <- F
        merge_sort_pairs(I, X)
        labels = for each row in I, select min mode value from X
```

References

[1] A. Azad, A. Buluç, and J. R. Gilbert, "Parallel triangle counting and enumeration using matrix algebra," in *GABB at IPDPS*. IEEE Computer Society, 2015, pp. 804–811. [Online]. Available: https://doi.org/10.1109/IPDPSW.2015.75

[2] R. Azimov and S. Grigorev, "Context-free path querying by matrix multiplication," in *GRADES-NDA at SIGMOD*. ACM, 2018, pp. 5:1–5:10. [Online]. Available: https://doi.org/10.1145/3210259.3210264

[3] ——, "Path querying with conjunctive grammars by matrix multiplication," *Programming and Computer Software*, vol. 45, no. 7, pp. 357–364, 2019. [Online]. Available: https://doi.org/10.1134/S0361768819070041

[4] M. Aznaveh, J. Chen, T. A. Davis, B. Hegyi, S. P. Kolodziej, T. G. Mattson, and G. Szárnyas, "Parallel GraphBLAS with OpenMP," in *CSC*. SIAM, 2020, pp. 138–148. [Online]. Available: https://doi.org/10.1137/1.9781611976229.14

[5] T. A. Davis, "Graph algorithms via SuiteSparse:GraphBLAS: Triangle counting and K-truss," in *HPEC*. IEEE, 2018. [Online]. Available: https://doi.org/10.1109/HPEC.2018.8547538

[6] ——, "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

[7] T. A. Davis, M. Aznaveh, and S. P. Kolodziej, "Write quick, run fast: Sparse deep neural network in 20 minutes of development time via SuiteSparse:GraphBLAS," in *HPEC*. IEEE, 2019, pp. 1–6. [Online]. Available: https://doi.org/10.1109/HPEC.2019.8916550

[8] M. Elekes, A. Nagy, D. Sándor, J. B. Antal, T. A. Davis, and G. Szárnyas, "A GraphBLAS solution to the SIGMOD 2014 programming contest using multi-source BFS," in *HPEC*, 2020.

[9] M. Elekes and G. Szárnyas, "An incremental GraphBLAS solution for the 2018 TTC Social Media case study," in *GrAPL at IPDPS*. IEEE, 2020, pp. 203–206. [Online]. Available: https://doi.org/10.1109/IPDPSW50202.2020.00045

[10] J. Kepner, M. Kumar, J. E. Moreira, P. Pattnaik, M. J. Serrano, and H. M. Tufo, "Enabling massive deep neural networks with the GraphBLAS," in *HPEC*. IEEE, 2017, pp. 1–10. [Online]. Available: https://doi.org/10.1109/HPEC.2017.8091098

[11] T. M. Low, D. G. Spampinato, S. McMillan, and M. Pelletier, "Linear algebraic louvain method in python," in *GrAPL at IPDPS*. IEEE, 2020, pp. 223–226. [Online]. Available: https://doi.org/10.1109/IPDPSW50202.2020.00050

[12] T. Mattson, T. A. Davis, M. Kumar, A. Buluç, S. McMillan, J. E. Moreira, and C. Yang, "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

[13] T. G. Mattson, C. Yang, S. McMillan, A. Buluç, and J. E. Moreira, "GraphBLAS C API: Ideas for future versions of the specification," in *HPEC*. IEEE, 2017. [Online]. Available: https://doi.org/10.1109/HPEC.2017.8091095

[14] N. Mishin, I. Sokolov, E. Spirin, V. Kutuev, E. Nemchinov, S. Gorbatyuk, and S. Grigorev, "Evaluation of the context-free path querying algorithm based on matrix multiplication," in *GRADES-NDA at SIGMOD*. ACM, 2019, pp. 12:1–12:5. [Online]. Available: https://doi.org/10.1145/3327964.3328503

[15] E. Orachev, I. Epelbaum, R. Azimov, and S. Grigorev, "Context-free path querying by Kronecker product," in *ADBIS*, ser. Lecture Notes in Computer Science, vol. 12245. Springer, 2020, pp. 49–59. [Online]. Available: https://doi.org/10.1007/978-3-030-54832-2_6

[16] D. G. Spampinato, U. Sridhar, and T. M. Low, "Linear algebraic depth-first search," in *ARRAY at PLDI*. ACM, 2019, pp. 93–104. [Online]. Available: https://doi.org/10.1145/3315454.3329962

[17] U. Sridhar, M. Blanco, R. Mayuranath, D. G. Spampinato, T. M. Low, and S. McMillan, "Delta-stepping SSSP: From vertices and edges to graphblas implementations," in *GrAPL at IPDPS*. IEEE, 2019, pp. 241–250. [Online]. Available: https://doi.org/10.1109/IPDPSW.2019.00047

[18] A. Terekhov, A. Khoroshev, R. Azimov, and S. Grigorev, "Context-free path querying with single-path semantics by matrix multiplication," in *GRADES-NDA at SIGMOD*. ACM, 2020, pp. 5:1–5:12. [Online]. Available: https://doi.org/10.1145/3398682.3399163

[19] C. Yang, A. Buluç, and J. D. Owens, "Implementing push-pull efficiently in GraphBLAS," in *ICPP*. ACM, 2018, pp. 89:1–89:11. [Online]. Available: https://doi.org/10.1145/3225058.3225122

[20] ——, "GraphBLAST: A high-performance linear algebra-based graph framework on the GPU," *CoRR*, vol. abs/1908.01407, 2019, http://arxiv.org/abs/1908.01407. [Online]. Available: http://arxiv.org/abs/1908.01407

[21] Y. Zhang, A. Azad, and Z. Hu, "FastSV: A distributed-memory connected component algorithm with fast convergence," in *PPSC*. SIAM, 2020, pp. 46–57. [Online]. Available: https://doi.org/10.1137/1.9781611976137.5