LAGraph Algorithms

LAGraph Working Group

September 16, 2020

Abstract

Theoretical documentation for LAGraph.

1 Notation

Table 1 contains the notation of GraphBLAS operations

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

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

- $s = $ `fp64()`

- $u = $ `fp32(n)`

- $A = $ `uint16(m, n)`

- $A = $ `int64(k, 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	$C <M>+ =A +. \times B$
`vxm`	vector-matrix multiplication	$w <m>+ =u +. \times A$
`mxv`	matrix-vector multiplication	$w <m>+ =A +. \times u$
`eWiseAdd`	element-wise addition	$C <M>+ =A + B$
	set union of patterns	$w <m>+ =u + v$
`eWiseMult`	element-wise multiplication	$C <M>+ =A \times B$
	set intersection of patterns	$w <m>+ =u \times v$
`extract`	extract submatrix	$C <M>+ =A(I, J)$
	extract column vector	$w <m>+ =A(:, j)$
	extract row vector	$w <m>+ =A(i, :)$
	extract subvector	$w <m>+ =u(I)$
`assign`	assign matrix to submatrix with mask for `C`	$C <M> (I, J)+ =A$
	assign scalar to submatrix with mask for `C`	$C <M> (I, J)+ =s$
	assign vector to subvector with mask for `w`	$w <m> (I)+ =u$
	assign scalar to subvector with mask for `w`	$w <m> (I)+ =s$
`subassign (GxB)`	assign matrix to submatrix with submask for `C(I,J)`	$C(I, J) <M>+ =A$
	assign scalar to submatrix with submask for `C(I,J)`	$C(I, J) <M>+ =s$
	assign vector to subvector with submask for `w(I)`	$w(I) <m>+ =u$
	assign scalar to subvector with submask for `w(I)`	$w(I) <m>+ =s$
`apply`	apply unary operator	$C <M>+ =\mathbf{f}(A)$
		$w <m>+ =\mathbf{f}(u)$
`select (GxB)`	apply select operator	$C <M>+ =select(A, \mathbf{f}(k))$
		$C <M>+ =select(\text{low} \leq A \leq \text{up})$
		$w <m>+ =select(u, \mathbf{f}(k))$
		$w <m>+ =select(\text{low} \leq u \leq \text{up})$
`reduce`	reduce matrix to column vector	$w <m>+ =[+ A]$
	reduce matrix to scalar	$s+ =[+ A]$
	reduce vector to scalar	$s+ =[+ u]$
`transpose`	transpose	$C <M>+ =A'$
`kronecker`	Kronecker multiplication	$C <M>+ =kron(A, B)$
`new`	new matrix	$A = \text{TYPEPRECISION}(n, m)$
	new vector	$u = \text{TYPEPRECISION}(n)$
`build`	build matrix from index/value arrays	$C <- \{I, J, X\}$
	build vector from index/value arrays	$w <- \{I, X\}$
`extractTuples`	extract index/value arrays	$\{I, J, X\} <- A$
		$\{I, X\} <- u$
`dup`	duplicate matrix	$C <- A$
	duplicate vector	$w <- u$
`extractElement`	extract scalar element	$s = A(i, j)$
		$s = u(i)$
`setElement`	set element	$C(i, j) = s$
		$w(i) = s$

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 $\times$ are the addition and multiplication operators forming a semiring and default to conventional arithmetic $+$ and $\times$ operators. $\odot$ 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 $\ll M \gg$, $\ll !M \gg$, $\ll m \gg$, and $\ll !m \gg$, 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 $\ll$! seen $\gg=$ 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 $\ll$! seen $\gg=$ frontier any.pair A
8   **else**
9   frontier $\ll$! seen $\gg=$ A any.pair frontier

Algorithm 3: Multi-source breadth-first search.

    Data: ...
    Result: ...
1 Function *MSBFS*
2   Frontier <- diag$(\mathbb{S}, n)$
3   **for** level = 1 *to* n − 1 **do**
4   Seen $<$ Frontier $>=$ level
5   Frontier $\ll$! Seen $\gg=$ Frontier any.pair A

Algorithm 4: Betweenness centrality.
---

1 **Function** *MSBFS*

2 // The NumSp structure holds the number of shortest paths for each node and starting vertex discovered so far.

3 // Initialized to source vertices.

4 $\text{NumSp} <\text{-} \{\text{s}, [1, 1, \ldots, 1]\}$

5 // The Frontier holds the number of shortest paths for each node and starting vertex discovered so far.

6 // Initialized to source vertices.

7 $\text{Frontier} < \text{NumSp} >= \text{A}(\text{s}, :)$

8 $\text{d} = 0$

9 // The Sigmas matrices store frontier information for each level of the BFS phase.

10 // BFS phase (forward sweep)

11 **do**

12   // $\text{Sigmas}[\text{d}](:, \text{s}) = \text{d}^{\text{th}}$ level frontier from source vertex s

13 $\text{Sigmas}[\text{d}] = \text{bool}(\text{n}, \text{nsver})$

14 $\text{Sigmas}[\text{d}](:, :) = \text{Frontier}$ // Convert matrix to Boolean

15 $\text{NumSp} = \text{NumSp} + \text{Frontier}$ // Accumulate path counts

16 $\text{Frontier} \ll \text{NumSp} \gg= \text{A}' + . \times \text{Frontier}$ // Update frontier

17 **while** *nvals*$(\text{Frontier}) > 0$

18 $\text{NumSpInv} = \text{fp32}(\text{n}, \text{nsver})$

19 $\text{NumSpInv} = 1.0 \, \text{DIV} \, \text{NumSp}$

20 $\text{BCU} = \text{fp32}(\text{n}, \text{nsver})$

21 $\text{BCU}(:) = 1.0$ // Make BCU dense, initialize all elements to 1.0

22 $\text{W} = \text{fp32}(\text{n}, \text{nsver})$

23 // Tally phase (backward sweep)

24 **for** $\text{i} = \text{d} - 1$ **downto** 0 **do**

25 $\text{W} \ll \text{Sigmas}[\text{i}] \gg= \text{NumSpInv} \, \text{DIV} \, \text{BCU}$

26 $\text{W} \ll \text{Sigmas}[\text{i} - 1] \gg= \text{A} + . \times \text{W}$ // Add contributions by successors and mask with that BFS level's frontier.

27 $\text{BCU} + = \text{W} \times \text{NumSp}$

28 // Row reduce BCU and subtract nsver from every entry to account for 1 extra value per BCU row element

29 $\text{delta} = [+ \, \text{BCU}]$

30 $\text{delta} \, \text{MINUS} = \text{nsver}$

Algorithm 5: PageRank (used in Graphalytics).

Data: alpha constant (damping factor)

Result: ...

1 **Function** *PageRank*

2 $\text{pr}(:) = 1/n$

3 $\text{outdegrees} = [+_{\text{j}} \, \text{A}(:, \text{j})]$

4 **for** $\text{k} = 1$ **to** numIterations **do**

5 $\text{importance} = \text{pr} \, \text{DIV} \, \text{outdegrees}$

6 $\text{importance} = \text{times}(\text{importance}, \text{alpha})$ // apply the times$(x, s) = x \cdot s$ operator

7 $\text{importance} = \text{importance} + . \times \text{A}$

8 $\text{danglingVertexRanks} < ! \, \text{outdegrees} >= \text{pr}(:)$

9 $\text{totalDanglingRank} = [+ \, \text{danglingVertexRanks}(\text{i})](\text{alpha})/(n)$

10 $\text{pr}(:) = (1 - \text{alpha})/(n) + \text{totalDanglingRank}$

11 $\text{pr} < \text{importance} >= \text{pr} + \text{importance}$

Algorithm 6: Algebraic Bellman-Ford for SSSP.

1 Function *SSSP*
2 $\mathrm{d(s)} = 0$
3   **for** $\mathrm{k} = 1$ **to** $\mathrm{n} - 1$ **do**
4 $\mathrm{dtmp} = \mathrm{d\,min.plus\,A}$
5   **if** $\mathrm{dtmp} = \mathrm{d}$ **then**
6   **break**
7 $\mathrm{d\,\texttt{<-}\,dtmp}$

Algorithm 7: Delta-stepping SSSP.

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

1 Function *DeltaStepping*
2 $\mathrm{A_L} = select(0 < \mathrm{A} \leq \Delta)$
3 $\mathrm{A_H} = select(\Delta < \mathrm{A})$
4 $\mathrm{t}(:) = \infty$
5 $\mathrm{t}(\mathrm{s}) = 0$
6   **while** $nvals(select(\mathrm{i}\Delta \leq \mathrm{t})) \neq 0$ **do**
7 $\mathrm{s} = 0$
8 $\mathrm{t_{B_i}} = select(\mathrm{i}\Delta \leq \mathrm{t} < (\mathrm{i}+1)\Delta)$
9   **while** $\mathrm{t_{B_i}} \neq 0$ **do**
10 $\mathrm{t_{Req}} = \mathrm{A_L}\text{'} + . \times (\mathrm{t} \times \mathrm{t_{B_i}})$
11 $\mathrm{e} = select(0 < \mathrm{e} + \mathrm{t_{B_i}})$
12 $\mathrm{t_{B_i}} = select(\mathrm{i}\Delta \leq \mathrm{t_{Req}} < (\mathrm{i}+1)\Delta) \times (\mathrm{t_{Req}} \underset{+}{\min} \mathrm{t})$
13 $\mathrm{t_{Req}} = \mathrm{A_H}\text{'} + . \times (\mathrm{t} \times \mathrm{e})$
14 $\mathrm{t} = \mathrm{t\,min\,t_{Req}}$
15 $\mathrm{i} = \mathrm{i} + 1$

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

1 Function *FloydWarshall*
2 $\mathrm{D\,\texttt{<-}\,A}$
3   **for** $\mathrm{k} = 1$ **to** n **do**
4 $\mathrm{D} = \mathrm{D\,min}[\mathrm{D}(:, \mathrm{k})\,\mathrm{min.plus}\,(\mathrm{D}(\mathrm{k}, :)]$

Algorithm 9: FastSV algorithm.

---
1 Function *FastSV*
```
2      n = nrows(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 10: 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 11: Triangle count (Sandia).

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

Algorithm 12: 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 13: 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 14: *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 + .land C
6   C = *select*(C ≥ k − 2)
7   **if** nonzeros == *nvals*(C) **then**
8   **break**
9   nonzeros <- *nvals*(C)

Algorithm 15: 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 t_q = 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 q_1 < t_q >= q + .× S
19
20   t = (q_1 == [max q_1])
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 16: 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] 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