LAGraph Algorithms

LAGraph Working Group

July 7, 2021

Abstract

Theoretical documentation for LAGraph.

1 Notation

Table 1 contains the notation of GraphBLAS operations

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

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

- let: $s \in \mathbb{Q}_{64}$

- let: $\mathbf{u} \in \mathbb{Q}_{32}^n$

- let: $\mathbf{A} \in \mathbb{N}_{16}^{m \times n}$

- let: $\mathbf{A} \in \mathbb{Z}_{64}^{k \times m}$

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	$\mathbf{C}\langle\mathbf{M}\rangle\odot=\mathbf{A}\oplus.\otimes\mathbf{B}$
`vxm`	vector-matrix multiplication	$\mathbf{w}\langle\mathbf{m}\rangle\odot=\mathbf{u}\oplus.\otimes\mathbf{A}$
`mxv`	matrix-vector multiplication	$\mathbf{w}\langle\mathbf{m}\rangle\odot=\mathbf{A}\oplus.\otimes\mathbf{u}$
`eWiseAdd`	element-wise addition	$\mathbf{C}\langle\mathbf{M}\rangle\odot=\mathbf{A}\oplus\mathbf{B}$
	set union of patterns	$\mathbf{w}\langle\mathbf{m}\rangle\odot=\mathbf{u}\oplus\mathbf{v}$
`eWiseMult`	element-wise multiplication	$\mathbf{C}\langle\mathbf{M}\rangle\odot=\mathbf{A}\otimes\mathbf{B}$
	set intersection of patterns	$\mathbf{w}\langle\mathbf{m}\rangle\odot=\mathbf{u}\otimes\mathbf{v}$
`extract`	extract submatrix	$\mathbf{C}\langle\mathbf{M}\rangle\odot=\mathbf{A}(I,J)$
	extract column vector	$\mathbf{w}\langle\mathbf{m}\rangle\odot=\mathbf{A}(:,j)$
	extract row vector	$\mathbf{w}\langle\mathbf{m}\rangle\odot=\mathbf{A}(i,:)$
	extract subvector	$\mathbf{w}\langle\mathbf{m}\rangle\odot=\mathbf{u}(I)$
`assign`	assign matrix to submatrix with mask for $\mathbf{C}$	$\mathbf{C}\langle\mathbf{M}\rangle(I,J)\odot=\mathbf{A}$
	assign scalar to submatrix with mask for $\mathbf{C}$	$\mathbf{C}\langle\mathbf{M}\rangle(I,J)\odot=s$
	assign vector to subvector with mask for $\mathbf{w}$	$\mathbf{w}\langle\mathbf{m}\rangle(I)\odot=\mathbf{u}$
	assign scalar to subvector with mask for $\mathbf{w}$	$\mathbf{w}\langle\mathbf{m}\rangle(I)\odot=s$
`subassign (GxB)`	assign matrix to submatrix with submask for $\mathbf{C}(I,J)$	$\mathbf{C}(I,J)\langle\mathbf{M}\rangle\odot=\mathbf{A}$
	assign scalar to submatrix with submask for $\mathbf{C}(I,J)$	$\mathbf{C}(I,J)\langle\mathbf{M}\rangle\odot=s$
	assign vector to subvector with submask for $\mathbf{w}(I)$	$\mathbf{w}(I)\langle\mathbf{m}\rangle\odot=\mathbf{u}$
	assign scalar to subvector with submask for $\mathbf{w}(I)$	$\mathbf{w}(I)\langle\mathbf{m}\rangle\odot=s$
`apply`	apply unary operator	$\mathbf{C}\langle\mathbf{M}\rangle\odot=f(\mathbf{A})$
		$\mathbf{w}\langle\mathbf{m}\rangle\odot=f(\mathbf{u})$
`select (GxB)`	apply select operator	$\mathbf{C}\langle\mathbf{M}\rangle\odot=select(\mathbf{A},f(k))$
		$\mathbf{C}\langle\mathbf{M}\rangle\odot=select(low\leq\mathbf{A}\leq up)$
		$\mathbf{w}\langle\mathbf{m}\rangle\odot=select(\mathbf{u},f(k))$
		$\mathbf{w}\langle\mathbf{m}\rangle\odot=select(low\leq\mathbf{u}\leq up)$
`reduce`	reduce matrix to column vector	$\mathbf{w}\langle\mathbf{m}\rangle\odot=[\oplus_j\mathbf{A}(:,j)]$
	reduce matrix to scalar	$s\odot=[\oplus_{ij}\mathbf{A}(i,j)]$
	reduce vector to scalar	$s\odot=[\oplus_i\mathbf{u}(i)]$
`transpose`	transpose	$\mathbf{C}\langle\mathbf{M}\rangle\odot=\mathbf{A}^\top$
`kronecker`	Kronecker multiplication	$\mathbf{C}\langle\mathbf{M}\rangle\odot=kron(\mathbf{A},\mathbf{B})$
`new`	new matrix	let: $\mathbf{A}\in\mathsf{TYPE}_{\mathrm{PRECISION}}^{n\times m}$
	new vector	let: $\mathbf{u}\in\mathsf{TYPE}_{\mathrm{PRECISION}}^{n}$
`build`	build matrix from index/value arrays	$\mathbf{C}\leftarrow\{I,J,X\}$
	build vector from index/value arrays	$\mathbf{w}\leftarrow\{I,X\}$
`extractTuples`	extract index/value arrays	$\{I,J,X\}\leftarrow\mathbf{A}$
		$\{I,X\}\leftarrow\mathbf{u}$
`dup`	duplicate matrix	$\mathbf{C}\leftarrow\mathbf{A}$
	duplicate vector	$\mathbf{w}\leftarrow\mathbf{u}$
`extractElement`	extract scalar element	$s=\mathbf{A}(i,j)$
		$s=\mathbf{u}(i)$
`setElement`	set element	$\mathbf{C}(i,j)=s$
		$\mathbf{w}(i)=s$

Table 1: GraphBLAS operations and methods based on [6]. *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 (s, i, j) while arrays are uppercase italic (X, I, J). $\oplus$ and $\otimes$ are the addition and multiplication operators forming a semiring and default to conventional arithmetic $+$ and $\times$ operators. $\odot$ is the apply operator. Masks $\langle\mathbf{M}\rangle$ and $\langle\mathbf{m}\rangle$ are used to selectively write to the result matrix/vector. The complements of masks $\langle\mathbf{M}\rangle$, $\langle\mathbf{m}\rangle$ can be selected with the negation symbol, denoted with $\langle\neg\mathbf{M}\rangle$, $\langle\neg\mathbf{m}\rangle$, respectively. Masks with "replace" semantics (annihilating all elements outside the mask) are denoted with $\langle\!\langle\mathbf{M}\rangle\!\rangle$, $\langle\!\langle\neg\mathbf{M}\rangle\!\rangle$, $\langle\!\langle\mathbf{m}\rangle\!\rangle$, and $\langle\!\langle\neg\mathbf{m}\rangle\!\rangle$. The structure of the mask is denoted with $\langle\{\mathbf{M}\}\rangle$, $\langle\neg\{\mathbf{M}\}\rangle$, $\langle\{\mathbf{m}\}\rangle$, and $\langle\neg\{\mathbf{m}\}\rangle$.

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.

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 $\textbf{NumSp} \leftarrowtail \{s, [1, 1, \ldots, 1]\}$

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

6   // Initialized to source vertices.

7 $\textbf{Frontier}\langle\textbf{NumSp}\rangle = \mathbf{A}(s, :)$

8 $d = 0$

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

10   // BFS phase (forward sweep)

11   **do**

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

13   let: $\textbf{Sigmas}[d] \in \mathbb{B}^{n \times nsver}$

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

15 $\textbf{NumSp} = \textbf{NumSp} \oplus \textbf{Frontier}$ // Accumulate path counts

16 $\textbf{Frontier}\langle\!\langle\textbf{NumSp}\rangle\!\rangle = \mathbf{A}^{\top} \oplus.\otimes \textbf{Frontier}$ // Update frontier

17   **while** *nvals*(**Frontier**) > 0

18   let: $\textbf{NumSpInv} \in \mathbb{Q}_{32}^{n \times nsver}$

19 $\textbf{NumSpInv} = 1.0 \oslash \textbf{NumSp}$

20   let: $\textbf{BCU} \in \mathbb{Q}_{32}^{n \times nsver}$

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

22   let: $\mathbf{W} \in \mathbb{Q}_{32}^{n \times nsver}$

23   // Tally phase (backward sweep)

24   **for** $i = d - 1$ **downto** 0 **do**

25 $\mathbf{W}\langle\!\langle\textbf{Sigmas}[i]\rangle\!\rangle = \textbf{NumSpInv} \oslash \textbf{BCU}$

26 $\mathbf{W}\langle\!\langle\textbf{Sigmas}[i-1]\rangle\!\rangle = \mathbf{A} \oplus.\otimes \mathbf{W}$ // Add contributions by successors and mask with that BFS level's frontier.

27 $\textbf{BCU} \oplus= \mathbf{W} \otimes \textbf{NumSp}$

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

29 $\textbf{delta} = [\oplus_j \textbf{BCU}(:, j)]$

30 $\textbf{delta} \ominus= nsver$

Algorithm 5: PageRank (used in Graphalytics).

Data: α constant (damping factor)

Result: ...

1 Function *PageRank*

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

3 $\textbf{outdegrees} = [\oplus_j \mathbf{A}(:, j)]$

4   **for** $k = 1$ **to** *numIterations* **do**

5 $\textbf{importance} = \textbf{pr} \oslash \textbf{outdegrees}$

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

7 $\textbf{importance} = \textbf{importance} \oplus.\otimes \mathbf{A}$

8 $\textbf{danglingVertexRanks}\langle\neg\textbf{outdegrees}\rangle = \textbf{pr}(:)$

9 $totalDanglingRank = [\oplus_i \textbf{danglingVertexRanks}(i)(\otimes)]\frac{\alpha}{n}$

10 $\textbf{pr}(:) = \frac{1-\alpha}{n} \oplus totalDanglingRank$

11 $\textbf{pr}\langle\textbf{importance}\rangle = \textbf{pr} \oplus \textbf{importance}$

Algorithm 6: Algebraic Bellman-Ford for SSSP.
---

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

Algorithm 7: Delta-stepping SSSP.

Data:
$\quad$ $\mathbf{A}, \mathbf{A_H}, \mathbf{A_L} \in \mathbb{Q}^{|V| \times |V|}$
$\quad$ $s, i \in \mathbb{N}$
$\quad$ $\Delta \in \mathbb{Q}$
$\quad$ $\mathbf{t}, \mathbf{t_{Req}} \in \mathbb{Q}^{|V|}$
$\quad$ $\mathbf{t_{B_i}}, \mathbf{e} \in \mathbb{N}^{|V|}$

1 **Function** *DeltaStepping*
2 $\quad$ $\mathbf{A_L} = select(0 < \mathbf{A} \leq \Delta)$
3 $\quad$ $\mathbf{A_H} = select(\Delta < \mathbf{A})$
4 $\quad$ $\mathbf{t}(:) = \infty$
5 $\quad$ $\mathbf{t}(s) = 0$
6 $\quad$ **while** $nvals(select(i\Delta \leq \mathbf{t})) \neq 0$ **do**
7 $\quad\quad$ $s = 0$
8 $\quad\quad$ $\mathbf{t_{B_i}} = select(i\Delta \leq \mathbf{t} < (i+1)\Delta)$
9 $\quad\quad$ **while** $\mathbf{t_{B_i}} \neq 0$ **do**
10 $\quad\quad\quad$ $\mathbf{t_{Req}} = \mathbf{A_L^\top} \oplus.\otimes (\mathbf{t} \otimes \mathbf{t_{B_i}})$
11 $\quad\quad\quad$ $\mathbf{e} = select(0 < \mathbf{e} \oplus \mathbf{t_{B_i}})$
12 $\quad\quad\quad$ $\mathbf{t_{B_i}} = select(i\Delta \leq \mathbf{t_{Req}} < (i+1)\Delta) \otimes (\mathbf{t_{Req}} \underset{\oplus}{\min} \mathbf{t})$
13 $\quad\quad$ $\mathbf{t_{Req}} = \mathbf{A_H^\top} \oplus.\otimes (\mathbf{t} \otimes \mathbf{e})$
14 $\quad\quad$ $\mathbf{t} = \mathbf{t}$ min $\mathbf{t_{Req}}$
15 $\quad\quad$ $i = i + 1$

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

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

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 = [⊕_ij C(i, j)]/2
```

Algorithm 11: Triangle count (Sandia).

```
1  Function TriangleCount
2      L = tril(A)
3      C⟨L⟩ = L ⊕.⊗ L
4      t = [⊕_ij C(i, j)]
```

Algorithm 12: Triangle count (FLAME).

```
1  Function TriangleCountFlame
2      for i = 2 to n − 1 do
3          A_20 = A(i + 1 : n, 0 : i − 1)
4          a_10 = A(0 : i − 1, i)
5          a_12 = A(i, i + 1 : n)
6          t ⊕= a_10 ⊕.⊗ A_20 ⊕.⊗ a_12
```

Algorithm 13: Local clustering coefficient.

```
1  Function PageRank
2      Tri⟨A⟩ = A ⊕.⊗ A                    // compute triangle count matrix
3      tri = [⊕_j Tri(:, j)]              // reduce to triangle count vector
4      deg = [⊕_j A(:, j)]                // reduce to vertex degree vector
5      wed = perm2(deg)          // apply perm2(x) = x · (x − 1) to get wedge count vector
6      lcc = tri ⊘ wed                                              // LCC vector
```

Algorithm 14: *k*-truss algorithm.

1 Function *KTruss*
2 $\mathbf{C} \leftarrowtail \mathbf{A}$
3 $nonzeros \leftarrowtail nvals(C)$
4   **for** $i = 1$ **to** $n - 1$ **do**
5 $\mathbf{C}\langle\mathbf{C}\rangle = \mathbf{C} \oplus.\mathsf{land}\ \mathbf{C}$
6 $\mathbf{C} = select(\mathbf{C} \geq k - 2)$
7   **if** $nonzeros == nvals(C)$ **then**
8   **break**
9 $nonzeros \leftarrowtail nvals(C)$

Algorithm 15: Louvain algorithm (WIP).

1 Function *Louvain*
2 $\mathbf{G} \oplus= \mathbf{G}^{\top}$
3 $\mathbf{k} = [\oplus_j \mathbf{A}(:,j)]$
4 $m = \frac{1}{2}[\oplus_i \mathbf{k}(i)]$
5 $\mathbf{S} \leftarrowtail \mathbf{I}$
6
7 $vertices\_changed \leftarrowtail nvals(\mathbf{k})$
8   **while** $vertices\_changed > 0$ **do**
9   **for** $j \in range(|V|)$ **do**
10 $\mathbf{v} = \mathbf{G}(j,:)$
11 $\mathbf{t_q} = \mathbf{v}\ \mathsf{any.pair}\ \mathbf{S}$
12 $\mathbf{sr} = \mathbf{S}(j,:)$
13 $\mathbf{S}(j,:) = \text{empty}$
14
15 $\mathbf{q} \leftarrowtail \mathbf{k}$
16 $\mathbf{q}\langle\mathbf{k}\rangle \otimes= -\mathbf{k}(j)/m$
17 $\mathbf{q} \oplus= \mathbf{v}$
18 $\mathbf{q_1}\langle\mathbf{t_q}\rangle = \mathbf{q} \oplus.\otimes \mathbf{S}$
19
20 $\mathbf{t} = (\mathbf{q_1} == [\max_i \mathbf{q_1}(i)])$
21   **while** $nvals(t) \neq 1$ **do**
22 $\mathbf{p} = random() \otimes \mathbf{t}$
23 $\mathbf{t} = (\mathbf{p} == [\max_i \mathbf{p}(i)])$
24 $\mathbf{S}(j,:) = \mathbf{t}$
25
26   **if** $nvals(\mathbf{sr} \otimes \mathbf{t}) == 0$ **then**
27 $vertices\_changed = nvals(\mathbf{k})$
28 $vertices\_changed = vertices\_changed - 1$

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

1 Function *CDLP*
2 $\mathbf{L} \leftarrowtail diag([0, 1, \ldots, n - 1])$
3   **for** $k = 1$ **to** t **do**
4 $\mathbf{F} = \mathbf{A}\ \mathsf{any.second}\ \mathbf{L}$ `// Frequency matrix`
5 $\{I, \_, X\} \leftarrowtail \mathbf{F}$
6 $merge\_sort\_pairs(I, X)$
7 $\mathbf{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