

LAGraph Algorithms

LAGraph Working Group

October 14, 2020

Abstract

Theoretical documentation for LAGraph.

1 Notation

Table 1 contains the notation of GraphBLAS operations

Additionally, we use $\mathbf{D} = \text{diag}(J, n)$ to construct a diagonal matrix $\mathbf{D} \leftarrow \{J, J, [1, 1, \dots, 1]\}$. 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 talk¹
- CFPQs based on a string of GRADES/ADBIS/other papers [2, 3, 14, 18, 15]
- Implementations: GBTL², 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 set union of patterns	$\mathbf{C}\langle\mathbf{M}\rangle\odot=\mathbf{A}\oplus\mathbf{B}$ $\mathbf{w}\langle\mathbf{m}\rangle\odot=\mathbf{u}\oplus\mathbf{v}$
eWiseMult	element-wise multiplication set intersection of patterns	$\mathbf{C}\langle\mathbf{M}\rangle\odot=\mathbf{A}\otimes\mathbf{B}$ $\mathbf{w}\langle\mathbf{m}\rangle\odot=\mathbf{u}\otimes\mathbf{v}$
extract	extract submatrix extract column vector extract row vector extract subvector	$\mathbf{C}\langle\mathbf{M}\rangle\odot=\mathbf{A}(I, J)$ $\mathbf{w}\langle\mathbf{m}\rangle\odot=\mathbf{A}(:, j)$ $\mathbf{w}\langle\mathbf{m}\rangle\odot=\mathbf{A}(i, :)$ $\mathbf{w}\langle\mathbf{m}\rangle\odot=\mathbf{u}(I)$
assign	assign matrix to submatrix with mask for \mathbf{C} assign scalar to submatrix with mask for \mathbf{C} assign vector to subvector with mask for \mathbf{w} assign scalar to subvector with mask for \mathbf{w}	$\mathbf{C}\langle\mathbf{M}\rangle(I, J)\odot=\mathbf{A}$ $\mathbf{C}\langle\mathbf{M}\rangle(I, J)\odot=s$ $\mathbf{w}\langle\mathbf{m}\rangle(I)\odot=\mathbf{u}$ $\mathbf{w}\langle\mathbf{m}\rangle(I)\odot=s$
subassign (GxB)	assign matrix to submatrix with submask for $\mathbf{C}(I, J)$ assign scalar to submatrix with submask for $\mathbf{C}(I, J)$ assign vector to subvector with submask for $\mathbf{w}(I)$ assign scalar to subvector with submask for $\mathbf{w}(I)$	$\mathbf{C}(I, J)\langle\mathbf{M}\rangle\odot=\mathbf{A}$ $\mathbf{C}(I, J)\langle\mathbf{M}\rangle\odot=s$ $\mathbf{w}(I)\langle\mathbf{m}\rangle\odot=\mathbf{u}$ $\mathbf{w}(I)\langle\mathbf{m}\rangle\odot=s$
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	$\mathbf{C}\langle\mathbf{M}\rangle\odot=\text{select}(\mathbf{A}, f(k))$ $\mathbf{C}\langle\mathbf{M}\rangle\odot=\text{select}(\text{low} \leq \mathbf{A} \leq \text{up})$ $\mathbf{w}\langle\mathbf{m}\rangle\odot=\text{select}(\mathbf{u}, f(k))$ $\mathbf{w}\langle\mathbf{m}\rangle\odot=\text{select}(\text{low} \leq \mathbf{u} \leq \text{up})$
reduce	reduce matrix to column vector reduce matrix to scalar reduce vector to scalar	$\mathbf{w}\langle\mathbf{m}\rangle\odot=[\oplus_j \mathbf{A}(:, j)]$ $s\odot=[\oplus_{ij} \mathbf{A}(i, j)]$ $s\odot=[\oplus_i \mathbf{u}(i)]$
transpose	transpose	$\mathbf{C}\langle\mathbf{M}\rangle\odot=\mathbf{A}^\top$
kroncker	Kronecker multiplication	$\mathbf{C}\langle\mathbf{M}\rangle\odot=\text{kron}(\mathbf{A}, \mathbf{B})$
new	new matrix new vector	let: $\mathbf{A} \in \text{TYPE}_{\text{PRECISION}}^{n \times m}$ let: $\mathbf{u} \in \text{TYPE}_{\text{PRECISION}}^n$
build	build matrix from index/value arrays build vector from index/value arrays	$\mathbf{C} \leftarrow \{I, J, X\}$ $\mathbf{w} \leftarrow \{I, X\}$
extractTuples	extract index/value arrays	$\{I, J, X\} \leftarrow \mathbf{A}$ $\{I, X\} \leftarrow \mathbf{u}$
dup	duplicate matrix duplicate vector	$\mathbf{C} \leftarrow \mathbf{A}$ $\mathbf{w} \leftarrow \mathbf{u}$
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 $\langle-\mathbf{M}\rangle, \langle-\mathbf{m}\rangle$, respectively. Masks with “replace” semantics (annihilating all elements outside the mask) are denoted with $\llbracket\mathbf{M}\rrbracket, \llbracket-\mathbf{M}\rrbracket, \llbracket\mathbf{m}\rrbracket$, and $\llbracket-\mathbf{m}\rrbracket$. The structure of the mask is denoted with $\langle\{\mathbf{M}\}\rangle, \langle-\{\mathbf{M}\}\rangle, \langle\{\mathbf{m}\}\rangle$, and $\langle-\{\mathbf{m}\}\rangle$.

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  $\leftarrow$  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   NumSp  $\leftarrow \{s, [1, 1, \dots, 1]\}$ 
5   // The Frontier holds the number of shortest paths for each node and starting vertex
   // discovered so far.
6   // Initialized to source vertices.
7   Frontier $\langle \mathbf{NumSp} \rangle = \mathbf{A}(s, :)$ 
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    // Sigmas $[d](:, s) = d^{\text{th}}$  level frontier from source vertex  $s$ 
13    let: Sigmas $[d] \in \mathbb{B}^{n \times nsver}$ 
14    Sigmas $[d](:, :) = \mathbf{Frontier}$  // Convert matrix to Boolean
15    NumSp = NumSp  $\oplus$  Frontier // Accumulate path counts
16    Frontier $\langle \mathbf{NumSp} \rangle = \mathbf{A}^\top \oplus \otimes \mathbf{Frontier}$  // Update frontier
17  while  $nvals(\mathbf{Frontier}) > 0$ 
18  let: NumSpInv  $\in \mathbb{Q}_{32}^{n \times nsver}$ 
19  NumSpInv =  $1.0 \oslash \mathbf{NumSp}$ 
20  let: BCU  $\in \mathbb{Q}_{32}^{n \times nsver}$ 
21  BCU $(:) = 1.0$  // Make BCU dense, initialize all elements to 1.0
22  let: W  $\in \mathbb{Q}_{32}^{n \times nsver}$ 
23  // Tally phase (backward sweep)
24  for  $i = d - 1$  downto 0 do
25    W $\langle \mathbf{Sigmas}[i] \rangle = \mathbf{NumSpInv} \oslash \mathbf{BCU}$ 
26    W $\langle \mathbf{Sigmas}[i - 1] \rangle = \mathbf{A} \oplus \otimes \mathbf{W}$  // Add contributions by successors and mask with that
   // BFS level's frontier.
27    BCU  $\oplus = \mathbf{W} \otimes \mathbf{NumSp}$ 
28  // Row reduce BCU and subtract  $nsver$  from every entry to account for 1 extra value
   // per BCU row element
29  delta =  $[\oplus_j \mathbf{BCU}(:, j)]$ 
30  delta  $\ominus = nsver$ 
```

Algorithm 5: PageRank (used in Graphalytics).

Data: α constant (damping factor)

Result: ...

```
1 Function PageRank
2   pr $(:) = 1/n$ 
3   outdegrees =  $[\oplus_j \mathbf{A}(:, j)]$ 
4   for  $k = 1$  to numIterations do
5     importance = pr  $\oslash$  outdegrees
6     importance = times(importance,  $\alpha$ ) // apply the times( $x, s$ ) =  $x \cdot s$  operator
7     importance = importance  $\oplus \otimes \mathbf{A}$ 
8     danglingVertexRanks $\langle \neg \mathbf{outdegrees} \rangle = \mathbf{pr}(s)$ 
9     totalDanglingRank =  $[\oplus_i \mathbf{danglingVertexRanks}(i)(\otimes)] \frac{\alpha}{n}$ 
10    pr $(s) = \frac{1-\alpha}{n} \oplus \mathbf{totalDanglingRank}$ 
11    pr $\langle \mathbf{importance} \rangle = \mathbf{pr} \oplus \mathbf{importance}$ 
```

Algorithm 6: Algebraic Bellman-Ford for SSSP.

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

Algorithm 7: Delta-stepping SSSP.

Data:

$\mathbf{A}, \mathbf{A}_H, \mathbf{A}_L \in \mathbb{Q}^{|V| \times |V|}$

$s, i \in \mathbb{N}$

$\Delta \in \mathbb{Q}$

$\mathbf{t}, \mathbf{t}_{\text{Req}} \in \mathbb{Q}^{|V|}$

$\mathbf{t}_{B_i}, \mathbf{e} \in \mathbb{N}^{|V|}$

```
1 Function DeltaStepping
2    $\mathbf{A}_L = \text{select}(0 < \mathbf{A} \leq \Delta)$ 
3    $\mathbf{A}_H = \text{select}(\Delta < \mathbf{A})$ 
4    $\mathbf{t}(\cdot) = \infty$ 
5    $\mathbf{t}(s) = 0$ 
6   while  $\text{nvals}(\text{select}(i\Delta \leq \mathbf{t})) \neq 0$  do
7      $s = 0$ 
8      $\mathbf{t}_{B_i} = \text{select}(i\Delta \leq \mathbf{t} < (i+1)\Delta)$ 
9     while  $\mathbf{t}_{B_i} \neq 0$  do
10       $\mathbf{t}_{\text{Req}} = \mathbf{A}_L^\top \oplus. \otimes (\mathbf{t} \otimes \mathbf{t}_{B_i})$ 
11       $\mathbf{e} = \text{select}(0 < \mathbf{e} \oplus \mathbf{t}_{B_i})$ 
12       $\mathbf{t}_{B_i} = \text{select}(i\Delta \leq \mathbf{t}_{\text{Req}} < (i+1)\Delta) \otimes (\mathbf{t}_{\text{Req}} \min_{\oplus} \mathbf{t})$ 
13       $\mathbf{t}_{\text{Req}} = \mathbf{A}_H^\top \oplus. \otimes (\mathbf{t} \otimes \mathbf{e})$ 
14       $\mathbf{t} = \mathbf{t} \text{ min } \mathbf{t}_{\text{Req}}$ 
15       $i = i + 1$ 
```

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

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

Algorithm 9: FastSV algorithm.

```

1 Function FastSV
2    $n = \text{rows}(\mathbf{A})$ 
3    $\mathbf{gf} = \mathbf{f}$ 
4    $\mathbf{dup} = \mathbf{gf}$ 
5    $\mathbf{mngf} = \mathbf{gf}$ 
6    $\{I, X\} \leftarrow \mathbf{f}$ 
7   repeat
8     // Step 1: Stochastic hooking
9      $\mathbf{mngf} = \mathbf{mngf} \min \mathbf{A}$ 
10     $\mathbf{mngf} = \mathbf{mngf} \text{ second.min } \mathbf{gf}$ 
11     $\mathbf{f}(X) = \mathbf{f} \min \mathbf{mngf}$ 
12    // Step 2: Aggressive hooking
13     $\mathbf{f} = \mathbf{f} \min \mathbf{mngf}$ 
14    // Step 3: Shortcutting
15     $\mathbf{f} = \mathbf{f} \min \mathbf{gf}$ 
16    // Step 4: Calculate grandparents
17     $\{I, X\} \leftarrow \mathbf{f}$ 
18     $\mathbf{gf} = \mathbf{f}(X)$ 
19    // Step 5: Check termination
20     $\mathbf{diff} = \mathbf{dup} \neq \mathbf{gf}$ 
21     $\text{sum} = [\oplus_i \mathbf{diff}(i)]$ 
22     $\mathbf{dup} = \mathbf{gf}$ 
23  until  $\text{sum} == 0$ 

```

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

```

1 Function TriangleCount
2    $\mathbf{L} = \text{tril}(\mathbf{A})$ 
3    $\mathbf{U} = \text{triu}(\mathbf{A})$ 
4    $\mathbf{B} = \mathbf{L} \oplus \otimes \mathbf{U}$ 
5    $\mathbf{C} = \mathbf{B} \otimes \mathbf{A}$ 
6    $t = [\oplus_{ij} \mathbf{C}(i, j)]/2$ 

```

Algorithm 11: Triangle count (Sandia).

```

1 Function TriangleCount
2    $\mathbf{L} = \text{tril}(\mathbf{A})$ 
3    $\mathbf{C}(\mathbf{L}) = \mathbf{L} \oplus \otimes \mathbf{L}$ 
4    $t = [\oplus_{ij} \mathbf{C}(i, j)]$ 

```

Algorithm 12: Triangle count (FLAME).

```

1 Function TriangleCountFlame
2   for  $i = 2$  to  $n - 1$  do
3      $\mathbf{A}_{20} = \mathbf{A}(i + 1:n, 0:i - 1)$ 
4      $\mathbf{a}_{10} = \mathbf{A}(0:i - 1, i)$ 
5      $\mathbf{a}_{12} = \mathbf{A}(i, i + 1:n)$ 
6      $\mathbf{t} \oplus= \mathbf{a}_{10} \oplus \otimes \mathbf{A}_{20} \oplus \otimes \mathbf{a}_{12}$ 

```

Algorithm 13: Local clustering coefficient.

```

1 Function PageRank
2    $\mathbf{Tri}(\mathbf{A}) = \mathbf{A} \oplus \otimes \mathbf{A}$  // compute triangle count matrix
3    $\mathbf{tri} = [\oplus_j \mathbf{Tri}(:, j)]$  // reduce to triangle count vector
4    $\mathbf{deg} = [\oplus_j \mathbf{A}(:, j)]$  // reduce to vertex degree vector
5    $\mathbf{wed} = \text{perm2}(\mathbf{deg})$  // apply  $\text{perm2}(x) = x \cdot (x - 1)$  to get wedge count vector
6    $\mathbf{lcc} = \mathbf{tri} \oslash \mathbf{wed}$  // LCC vector

```

Algorithm 14: k -truss algorithm.

```

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

```

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    $\mathbf{S} \leftarrow \mathbf{I}$ 
6
7    $vertices\_changed \leftarrow nvals(\mathbf{k})$ 
8   while  $vertices\_changed > 0$  do
9     for  $j \in range(|V|)$  do
10       $\mathbf{v} = \mathbf{G}(j, :)$ 
11       $\mathbf{t}_q = \mathbf{v}$  any.pair  $\mathbf{S}$ 
12       $\mathbf{sr} = \mathbf{S}(j, :)$ 
13       $\mathbf{S}(j, :) = \text{empty}$ 
14
15       $\mathbf{q} \leftarrow \mathbf{k}$ 
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  $nvals(\mathbf{t}) \neq 1$  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    $\mathbf{L} \leftarrow diag([0, 1, \dots, n - 1])$ 
3   for  $k = 1$  to  $t$  do
4      $\mathbf{F} = \mathbf{A}$  any.second  $\mathbf{L}$                                      // Frequency matrix
5      $\{I, \cup, X\} \leftarrow \mathbf{F}$ 
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>