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 $\mathbf{D} = diag(J, n)$ 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`
`vxm`
`mxv`
	matrix-matrix multiplication
	vector-matrix multiplication
	matrix-vector multiplication
		$\mathbf{C}\langle\mathbf{M}\rangle\odot{=}\mathbf{A} \oplus.\otimes \mathbf{B}$
		$\mathbf{w}\langle\mathbf{m}\rangle\odot{=}\mathbf{u} \oplus.\otimes \mathbf{A}$
		$\mathbf{w}\langle\mathbf{m}\rangle\odot{=}\mathbf{A} \oplus.\otimes \mathbf{u}$

`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	$\mathbf{C}\langle\mathbf{M}\rangle\odot{=}\mathit{select}(\mathbf{A}, f(k))$ $\mathbf{C}\langle\mathbf{M}\rangle\odot{=}\mathit{select}(\mathit{low} \leq \mathbf{A} \leq \mathit{up})$ $\mathbf{w}\langle\mathbf{m}\rangle\odot{=}\mathit{select}(\mathbf{u}, f(k))$ $\mathbf{w}\langle\mathbf{m}\rangle\odot{=}\mathit{select}(\mathit{low} \leq \mathbf{u} \leq \mathit{up})$
`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	$\mathbf{C}\langle\mathbf{M}\rangle\odot{=}\mathbf{A}^\top$
`kronecker`	Kronecker multiplication	$\mathbf{C}\langle\mathbf{M}\rangle\odot{=}\mathit{kron}(\mathbf{A},\mathbf{B})$
`new`	new matrix new vector	let: $\mathbf{A} \in \mathsf{TYPE}^{n \times m}_{\mathsf{PRECISION}}$ let: $\mathbf{u} \in \mathsf{TYPE}^{n}_{\mathsf{PRECISION}}$
`build`	build matrix from index/value arrays build vector from index/value arrays	$\mathbf{C} \mapsfrom \{I, J, X\}$ $\mathbf{w} \mapsfrom \{I, X\}$
`extractTuples`	extract index/value arrays	$\{I, J, X\} \mapsfrom \mathbf{A}$ $\{I, X\} \mapsfrom \mathbf{u}$
`dup`	duplicate matrix duplicate vector	$\mathbf{C} \mapsfrom \mathbf{A}$ $\mathbf{w} \mapsfrom \mathbf{u}$
`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.

Function *BFS*
  frontier(s) = TRUE
  for $level = 1$ **to** $n - 1$ **do**
    seen⟨frontier⟩ = $level$
    frontier⟨⟨¬seen⟩⟩ = frontier any.pair **A**

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

Function *BFS*
  frontier(s) = TRUE
  for $level = 1$ **to** $n - 1$ **do**
    seen⟨frontier⟩ = $level$
    $push$ = use some heuristics to determine whether to push/pull
    if $push$ **then**
      frontier⟨⟨¬seen⟩⟩ = frontier any.pair **A**
    else
      frontier⟨⟨¬seen⟩⟩ = **A** any.pair frontier

Algorithm 3: Multi-source breadth-first search.

Data: ...
Result: ...
Function *MSBFS*
  Frontier ↤ diag(S, n)
  for $level = 1$ ***to*** $n - 1$ **do**
    Seen⟨Frontier⟩ = $level$
    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 $\hookleftarrow \{s, [1, 1, \ldots, 1]\}$

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

// Initialized to source vertices.

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

$d = 0$

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

// BFS phase (forward sweep)

do

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

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

    Sigmas$[d](:, :) =$ **Frontier**   // Convert matrix to Boolean

    NumSp $=$ **NumSp** $\oplus$ **Frontier**   // Accumulate path counts

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

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

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

NumSpInv $= 1.0 \oslash$ **NumSp**

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

BCU$(:) = 1.0$ // Make **BCU** dense, initialize all elements to 1.0

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

// Tally phase (backward sweep)

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

    W$\langle\!\langle$**Sigmas**$[i]\rangle\!\rangle =$ **NumSpInv** $\oslash$ **BCU**

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

    BCU $\oplus=$ **W** $\otimes$ **NumSp**

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

delta $= [\oplus_j$ **BCU**$(:, j)]$

delta $\ominus= nsver$

Algorithm 5: PageRank (used in Graphalytics).

Data: α constant (damping factor)

Result: ...

Function *PageRank*

pr$(:) = 1/n$

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

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

    importance $=$ **pr** $\oslash$ **outdegrees**

    importance $= times($**importance**$, \alpha)$ // apply the $times(x, s) = x \cdot s$ operator

    importance $=$ **importance** $\oplus.\otimes$ **A**

    danglingVertexRanks$\langle\neg$**outdegrees**$\rangle =$ **pr**$(:)$

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

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

    pr$\langle$**importance**$\rangle =$ **pr** $\oplus$ **importance**

Algorithm 6: Algebraic Bellman-Ford for SSSP.

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

Algorithm 7: Delta-stepping SSSP.

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

Function *DeltaStepping*
  $\mathbf{A_L} = \mathit{select}(0 < \mathbf{A} \leq \Delta)$
  $\mathbf{A_H} = \mathit{select}(\Delta < \mathbf{A})$
  $\mathbf{t}(:) = \infty$
  $\mathbf{t}(s) = 0$
  while $\mathit{nvals}(\mathit{select}(i\Delta \leq \mathbf{t})) \neq 0$ **do**
    $s = 0$
    $\mathbf{t_{B_i}} = \mathit{select}(i\Delta \leq \mathbf{t} < (i+1)\Delta)$
    while $\mathbf{t_{B_i}} \neq 0$ **do**
      $\mathbf{t_{Req}} = \mathbf{A_L}^\top \oplus.\otimes (\mathbf{t} \otimes \mathbf{t_{B_i}})$
      $\mathbf{e} = \mathit{select}(0 < \mathbf{e} \oplus \mathbf{t_{B_i}})$
      $\mathbf{t_{B_i}} = \mathit{select}(i\Delta \leq \mathbf{t_{Req}} < (i+1)\Delta) \otimes (\mathbf{t_{Req}} \underset{\oplus}{\mathsf{min}}\ \mathbf{t})$
    $\mathbf{t_{Req}} = \mathbf{A_H}^\top \oplus.\otimes (\mathbf{t} \otimes \mathbf{e})$
    $\mathbf{t} = \mathbf{t}\ \mathsf{min}\ \mathbf{t_{Req}}$
    $i = i + 1$

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

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

Algorithm 9: FastSV algorithm.

Function *FastSV*
  $n = nrows(\mathbf{A})$
  $\mathbf{gf} = \mathbf{f}$
  $\mathbf{dup} = \mathbf{gf}$
  $\mathbf{mngf} = \mathbf{gf}$
  $\{I, X\} \hookleftarrow \mathbf{f}$
  repeat
    `// Step 1: Stochastic hooking`
    $\mathbf{mngf} = \mathbf{mngf}$ min $\mathbf{A}$
    $\mathbf{mngf} = \mathbf{mngf}$ second.min $\mathbf{gf}$
    $\mathbf{f}(X) = \mathbf{f}$ min $\mathbf{mngf}$
    `// Step 2: Aggressive hooking`
    $\mathbf{f} = \mathbf{f}$ min $\mathbf{mngf}$
    `// Step 3: Shortcutting`
    $\mathbf{f} = \mathbf{f}$ min $\mathbf{gf}$
    `// Step 4: Calculate grandparents`
    $\{I, X\} \hookleftarrow \mathbf{f}$
    $\mathbf{gf} = \mathbf{f}(X)$
    `// Step 5: Check termination`
    $\mathbf{diff} = \mathbf{dup} \neq \mathbf{gf}$
    $sum = [\oplus_i \mathbf{diff}(i)]$
    $\mathbf{dup} = \mathbf{gf}$
  until $sum == 0$

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

Function *TriangleCount*
  $\mathbf{L} = tril(\mathbf{A})$
  $\mathbf{U} = triu(\mathbf{A})$
  $\mathbf{B} = \mathbf{L} \oplus.\otimes \mathbf{U}$
  $\mathbf{C} = \mathbf{B} \otimes \mathbf{A}$
  $t = [\oplus_{ij} \mathbf{C}(i, j)]/2$

Algorithm 11: Triangle count (Sandia).

Function *TriangleCount*
  $\mathbf{L} = tril(\mathbf{A})$
  $\mathbf{C}\langle\mathbf{L}\rangle = \mathbf{L} \oplus.\otimes \mathbf{L}$
  $t = [\oplus_{ij} \mathbf{C}(i, j)]$

Algorithm 12: Triangle count (FLAME).

Function *TriangleCountFlame*
  for $i = 2$ **to** $n - 1$ **do**
    $\mathbf{A_{20}} = \mathbf{A}(i + 1 : n, 0 : i - 1)$
    $\mathbf{a_{10}} = \mathbf{A}(0 : i - 1, i)$
    $\mathbf{a_{12}} = \mathbf{A}(i, i + 1 : n)$
    $\mathbf{t} \oplus= \mathbf{a_{10}} \oplus.\otimes \mathbf{A_{20}} \oplus.\otimes \mathbf{a_{12}}$

Algorithm 13: Local clustering coefficient.

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

Algorithm 14: k-truss algorithm.

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

Algorithm 15: Louvain algorithm (WIP).

Function *Louvain*
  $\mathbf{G} \oplus= \mathbf{G}^\top$
  $\mathbf{k} = [\bigoplus_j \mathbf{A}(:, j)]$
  $m = \frac{1}{2}[\bigoplus_i \mathbf{k}(i)]$
  $\mathbf{S} \leftarrowtail \mathbf{I}$

  $vertices\_changed \leftarrowtail nvals(\mathbf{k})$
  while $vertices\_changed > 0$ **do**
    for $j \in range(|V|)$ **do**
      $\mathbf{v} = \mathbf{G}(j, :)$
      $\mathbf{t_q} = \mathbf{v}\ \mathsf{any.pair}\ \mathbf{S}$
      $\mathbf{sr} = \mathbf{S}(j, :)$
      $\mathbf{S}(j, :) = \text{empty}$

      $\mathbf{q} \leftarrowtail \mathbf{k}$
      $\mathbf{q}\langle\mathbf{k}\rangle \otimes= -\mathbf{k}(j)/m$
      $\mathbf{q} \oplus= \mathbf{v}$
      $\mathbf{q_1}\langle\mathbf{t_q}\rangle = \mathbf{q} \oplus.\otimes\ \mathbf{S}$

      $\mathbf{t} = (\mathbf{q_1} == [\mathsf{max}_i\ \mathbf{q_1}(i)])$
      while $nvals(t) \neq 1$ **do**
        $\mathbf{p} = random() \otimes \mathbf{t}$
        $\mathbf{t} = (\mathbf{p} == [\mathsf{max}_i\ \mathbf{p}(i)])$
      $\mathbf{S}(j, :) = \mathbf{t}$

      if $nvals(\mathbf{sr} \otimes \mathbf{t}) == 0$ **then**
        $vertices\_changed = nvals(\mathbf{k})$
      $vertices\_changed = vertices\_changed - 1$

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

Function *CDLP*
  $\mathbf{L} \leftarrowtail diag([0, 1, \ldots, n - 1])$
  for $k = 1$ **to** t **do**
    $\mathbf{F} = \mathbf{A}\ \mathsf{any.second}\ \mathbf{L}$ // Frequency matrix
    $\{I, \llcorner\!\lrcorner, X\} \leftarrowtail \mathbf{F}$
    $merge\_sort\_pairs(I, X)$
    $\mathbf{labels} = \text{for each row in } I\text{, select min mode value from } X$

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