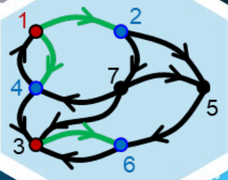
LAGraph:

A collection of graph algorithms built on top of GraphBLAS

Scott McMillan, Carnegie Mellon University / Software Engineering Institute
Tim Mattson, Intel Corporation

With contributions from:

Gabor Szarnyas, CWI Amsterdam
Tim Davis, Texas A&M University
Erick Welch , Anaconda
James Kitchen, Anaconda
David Bader, NJIT
Roi Lipman, RedisLabs



GraphBLAS Minisymposium @ SIAM CSE21
1 March 2021

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM21-0172

A Brief History…

- Sep. 2013: GraphBLAS “position paper” at IEEE HPEC
- Jun. 2015: GraphBLAS Forum kickoff

Standards for Graph Algorithm Primitives

Tim Mattson (Intel Corporation), David Bader (Georgia Institute of Technology), Jon Berry (Sandia National Laboratory), Aydin Buluc (Lawrence Berkeley National Laboratory), Jack Dongarra (University of Tennessee), Christos Faloutsos (Carnegie Melon University), John Feo (Pacific Northwest National Laboratory), John Gilbert (University of California at Santa Barbara), Joseph Gonzalez (University of California at Berkeley), Bruce Hendrickson (Sandia National Laboratory), Jeremy Kepner (Massachusetts Institute of Technology), Charles Leiserson (Massachusetts Institute of Technology), Andrew Lumsdaine (Indiana University), David Padua (University of Illinois at Urbana-Champaign), Stephen Poole (Oak Ridge National Laboratory), Steve Reinhardt (Cray Corporation), Mike Stonebraker (Massachusetts Institute of Technology), Steve Wallach (Convey Corporation), Andrew Yoo (Lawrence Livermore National Laboratory)

A Brief History…

- Sep. 2013: GraphBLAS “position paper” at IEEE HPEC
- Jun. 2015: GraphBLAS Forum kickoff

Standards for Graph Algorithm Primitives

Tim Mattson (Intel Corporation), David Bader (Georgia Institute of Technology), Jon Berry (Sandia National Laboratory), Aydin Buluc (Lawrence Berkeley National Laboratory), Jack Dongarra (University of Tennessee), Christos Faloutsos (Carnegie Melon University), John Feo (Pacific Northwest National Laboratory), John Gilbert (University of California at Santa Barbara), Joseph Gonzalez (University of California at Berkeley), Bruce Hendrickson (Sandia National Laboratory), Jeremy Kepner (Massachusetts Institute of Technology), Charles Leiserson (Massachusetts Institute of Technology), Andrew Lumsdaine (Indiana University), David Padua (University of Illinois at Urbana-Champaign), Stephen Poole (Oak Ridge National Laboratory), Steve Reinhardt (Cray Corporation), Mike Stonebraker (Massachusetts Institute of Technology), Steve Wallach (Convey Corporation), Andrew Yoo (Lawrence Livermore National Laboratory)

- Dec. 2015: Formation of API Working Group
- May 2017: GraphBLAS C API Specification v1.0 released (“provisional”)
- Nov. 2017: SuiteSparse GraphBLAS v1.0 released
- May 2018: IBM GraphBLAS released, **“provisional” removed from C API spec.**

A Brief History…

- Sep. 2013: GraphBLAS "position paper" at IEEE HPEC
- Jun. 2015: GraphBLAS Forum kickoff

Standards for Graph Algorithm Primitives

Tim Mattson (Intel Corporation), David Bader (Georgia Institute of Technology), Jon Berry (Sandia National Laboratory), Aydin Buluc (Lawrence Berkeley National Laboratory), Jack Dongarra (University of Tennessee), Christos Faloutsos (Carnegie Melon University), John Feo (Pacific Northwest National Laboratory), John Gilbert (University of California at Santa Barbara), Joseph Gonzalez (University of California at Berkeley), Bruce Hendrickson (Sandia National Laboratory), Jeremy Kepner (Massachusetts Institute of Technology), Charles Leiserson (Massachusetts Institute of Technology), Andrew Lumsdaine (Indiana University), David Padua (University of Illinois at Urbana-Champaign), Stephen Poole (Oak Ridge National Laboratory), Steve Reinhardt (Cray Corporation), Mike Stonebraker (Massachusetts Institute of Technology), Steve Wallach (Convey Corporation), Andrew Yoo (Lawrence Livermore National Laboratory)

- Dec. 2015: Formation of API Working Group
- May 2017: GraphBLAS C API Specification v1.0 released ("provisional")
- Nov. 2017: SuiteSparse GraphBLAS v1.0 released
- May 2018: IBM GraphBLAS released, **"provisional" removed from C API spec.**

… But the GraphBLAS are really low level. They are for algorithm developers and researchers, not algorithm users.

What can we do to reach graph algorithm users?

A Brief History…

- Sep. 2013: GraphBLAS “position paper” at IEEE HPEC
- Jun. 2015: GraphBLAS Forum kickoff

Standards for Graph Algorithm Primitives

Tim Mattson (Intel Corporation), David Bader (Georgia Institute of Technology), Jon Berry (Sandia National Laboratory), Aydin Buluc (Lawrence Berkeley National Laboratory), Jack Dongarra (University of Tennessee), Christos Faloutsos (Carnegie Melon University), John Feo (Pacific Northwest National Laboratory), John Gilbert (University of California at Santa Barbara), Joseph Gonzalez (University of California at Berkeley), Bruce Hendrickson (Sandia National Laboratory), Jeremy Kepner (Massachusetts Institute of Technology), Charles Leiserson (Massachusetts Institute of Technology), Andrew Lumsdaine (Indiana University), David Padua (University of Illinois at Urbana-Champaign), Stephen Poole (Oak Ridge National Laboratory), Steve Reinhardt (Cray Corporation), Mike Stonebraker (Massachusetts Institute of Technology), Steve Wallach (Convey Corporation), Andrew Yoo (Lawrence Livermore National Laboratory)

- Dec. 2015: Formation of API Working Group
- May 2017: GraphBLAS C API Specification v1.0 released (“provisional”)
- Nov. 2017: SuiteSparse GraphBLAS v1.0 released
- May 2018: IBM GraphBLAS released, **“provisional” removed from C API spec.**

- May 2019: LAGraph “position paper” at IEEE IPDPSW
- Mar 2020: Formation of LAGraph Working Group
- May 2021: Release of LAGraph v1.0 (planned)

LAGraph: A Community Effort to Collect Graph Algorithms Built on Top of the GraphBLAS

Tim Mattson[‡], Timothy A. Davis[◇], Manoj Kumar[¶], Aydın Buluç[†], Scott McMillan[§], José Moreira[¶], Carl Yang[*,†]

[‡]*Intel Corporation* [†]*Computational Research Division, Lawrence Berkeley National Laboratory*
[◇]*Texas A&M University* [¶]*IBM Corporation* [§]*Software Engineering Institute, Carnegie Mellon University*
[*]*Electrical and Computer Engineering Department, University of California, Davis*

A Brief History…

- Sep. 2013: GraphBLAS “position paper” at IEEE HPEC
- Jun. 2015: GraphBLAS Forum kickoff
- Dec. 2015: Formation of APLW
- May 2017: Gr
- Nov. 2017: S
- May 2018: IB C API spec.
- May 2019: LAG at IEEE IPDPSW
- Mar 2020: Formation of LAGraph Working Group
- **May 2021**: Release of LAGraph v1.0 (planned)

THIS IS A WORK IN PROGESS
(some details could change)

Standards for Graph Algorithm Primitives

Tim Mattson (Intel Corporation), David Bader (Georgia Institute of Technology), Jon Berry (Sandia National Laboratory), Aydin Buluc (Lawrence Berkeley National Laboratory), Jack Dongarra (University of Tennessee), Christos Faloutsos (Carnegie Melon University), John Feo (Pacific Northwest National Laboratory), John Gilbert (University of California at Santa Barbara), Joseph Gonzalez (University of California at Berkeley), Bruce Hendrickson (Sandia National Laboratory), Jeremy Kepner tts Institute of Technology), Charles Leiserson (Massachusetts Institute of Technology) na University), David Padua (University of Illinois at Urbana-Champaign) aboratory), Steve Reinhardt (Cray Corporation), Mike Stonebr teve Wallach (Convey Corporation), oratory)

LAGraph: A Community Effort to Collect Graph Algorithms Built on Top of the GraphBLAS

Tim Mattson‡, Timothy A. Davis◦, Manoj Kumar¶, Aydın Buluç†, Scott McMillan§, José Moreira¶, Carl Yang*,†

‡Intel Corporation †Computational Research Division, Lawrence Berkeley National Laboratory
◦Texas A&M University ¶IBM Corporation §Software Engineering Institute, Carnegie Mellon University
*Electrical and Computer Engineering Department, University of California, Davis

LAGraph Working Group

- Members (meeting ~ weekly since March 2020):
  - Tim Davis, Gabor Szarnyas, Tim Mattson, Scott McMillan, Jim Kitchen, Erik Welch, David Bader, Roi Lipman, and others

- The purpose of the LAGraph effort:
  - Provide a repository for researchers to share graph algorithms based on GraphBLAS.
  - Create a library of “commercial-grade” algorithms for data scientists and other users.
- Supporting goals
  - Portability across different implementations of GraphBLAS and different platforms
  - Test GraphBLAS implementations, expose weaknesses, and push new features
  - Create a transparent process to new algorithms/utilities to the LAGraph user library
  - Support development of other language bindings for LAGraph+GraphBLAS

LAGraph Target Audience

- We serve two communities
  - **Developers** of graph algorithms (new and/or improved implementations)
  - **Users** of graph algorithms (application writers, data scientists, etc)

- Our users come in two flavors:
  - Basic users:
    - Want things to “just work”
    - Modest graphs, exploratory exercises
    - Ease of use over performance
  - Advanced users:
    - HUGE graphs
    - Parallel applications
    - Need more control on memory use, etc…

Elements of the LAGraph library Design

- Graph data structure
- Basic vs. Advanced API
- Signature conventions
- Error handling

The LAGraph "graph" data structure: LAGraph_Graph

- Transparent struct
- Contains two types of data:
  - Primary components
  - "Cached" properties
- New members added as need arises

```
struct LAGraph_Graph
{
    // Primary components (REQUIRED)
    GrB_Matrix   A;          // adjacency matrix
    LAGraph_Kind kind;       // directed, undirected, etc

    // Cached properties (OPTIONAL)
    GrB_Matrix   AT          // transposed matrix
    GrB_Vector   row_degree;
    GrB_Vector   col_degree;
    LAGraph_BooleanProperty
                 A_pattern_is_symmetric; // T/F/Unk
    // etc...
};
```

- Construction "moves" matrix into LAGraph object

```
// Matrix ownership transfers on construction
GrB_Matrix M;    //...construction of M omitted
LAGraph_Graph G;
LAGraph_New(&G, &M, LAGRAPH_DIRECTED, msg);
```

- Utilities provided for explicit computation of cached properties

```
// explicit computation of cached properties
LAGraph_Property_AT(G, msg);
```

Basic vs. Advanced Interfaces

Basic Interfaces

- Limited options
    - Likely only one function for a given algorithm
- May inspect input and compute expensive cached properties as needed
    - E.g. compute vertex degrees and sort prior to computation

Advanced Interfaces

- Multiple implementations (algorithms) for the same computation
    - E.g., push vs. pull, or batched modes
- Stricter requirements on inputs
    - E.g., will return an error before computing needed cached properties
- May include lower level entry points into existing algorithms (e.g. single-hop BFS)

Signature Conventions

- Naming conventions
  - **LAGraph_** "namespace"
  - Category of algorithm or utility
  - Algorithm name

```
int LAGraph_<Category>_<Algorithm>
(
    // outputs
    Type1   *out1,    // allocated internally
    Type2   *out2,
    ...
    // input/output
    Type3    inout,   // allocated by user
    ...
    // inputs
    Type4    input1,
    Type5    input2,
    ...
    char *msg
);
```

- Examples
  - `LAGraph_Community_Louvain`
  - `LAGraph_Community_Markov`
  - `LAGraph_Community_LabelPropagation`
  - `LAGraph_Property_AT`
  - `LAGraph_Property_RowDegree`
  - `LAGraph_Property_ColDegree`

Signature Conventions

- Outputs
  - Pass by reference, pointer created by user
  - Allocated by algorithm
  - Pass **`NULL`** for optional outputs
- Input/Outputs
  - Pass by value
  - Allocated before call (by user or other functions)
  - Pass graph object here if properties are computed
- Inputs
  - Pass by value (not modified)

```
int LAGraph_<Category>_<Algorithm>
(
   // outputs
   Type1   *out1,   // allocated internally
   Type2   *out2,
   ...
   // input/output
   Type3    inout,  // allocated by user
   ...
   // inputs
   Type4    input1,
   Type5    input2,
   ...
   char *msg
);
```

Signature Conventions

- Return Value, signed integer
  - = 0 → success
  - > 0 → algorithm specific warning code
  - < 0 → algorithm specific error code

- Error messages, **`msg`**
  - User-allocated char buffer of size LAGRAPH_MSG_LEN
  - Holds algorithm-specific error or warning messages

```
int LAGraph_<Category>_<Algorithm>
(
    // outputs
    Type1   *out1,    // allocated internally
    Type2   *out2,
    ...
    // input/output
    Type3    inout,   // allocated by user
    ...
    // inputs
    Type4    input1,
    Type5    input2,
    ...
    char *msg
);
```

Error Handling

- All GraphBLAS and LAGraph methods can return an error.
- Errors should be checked before proceeding.
- Resources should be released after an unrecoverable error.
- Exception handling in C:

```
#define LAGraph_TRY(LAGraph_method)
{
   int LAGraph_status = LAGraph_method;
   if (LAGraph_status < 0) {
      LAGraph_CATCH(LAGraph_status);
   }
}
```

- Developers define LAGraph_CATCH before a function using LAGraph_TRY
  - Ensure proper freeing of memory
  - Perform any other necessary tasks
- Similar mechanism provided for GraphBLAS calls.

LAGraph: Pushing the state of the art in algorithms using GraphBLAS

- Libraries are important, but we also want to drive research on algorithms that use the GraphBLAS.
- We found that it was hard to share algorithms given everyone used different notations.
- Hence … we are working to define a consensus notation for expressing graph algorithms using linear algebra

GraphBLAS Notation is Evolving

Operation	Notation	Operation	Notation
mxm	$\mathbf{C}\langle\mathbf{M}\rangle \odot= (\mathbf{A}^T \oplus.\otimes \mathbf{B}^T)$	transpose	$\mathbf{C}\langle\mathbf{M}\rangle \odot= \mathbf{A}^T$
mxv vxm	$\mathbf{w}\langle\mathbf{m}\rangle \odot= (\mathbf{A}^T \oplus.\otimes \mathbf{u})$ $\mathbf{w}^T\langle\mathbf{m}\rangle \odot= (\mathbf{u}^T \oplus.\otimes \mathbf{A}^T)$	reduce (row) reduce (scalar)	$\mathbf{w}\langle\mathbf{m}\rangle \odot= [\oplus_j \mathbf{A}^T(:,j)]$ $s \odot= [\oplus_{i,j} \mathbf{A}^T(i,j)]$ $s \odot= [\oplus_i \mathbf{u}(i)]$
eWiseMult	$\mathbf{C}\langle\mathbf{M}\rangle \odot= (\mathbf{A}^T \otimes \mathbf{B}^T)$ $\mathbf{w}\langle\mathbf{m}\rangle \odot= (\mathbf{u} \otimes \mathbf{v})$	eWiseAdd	$\mathbf{C}\langle\mathbf{M}\rangle \odot= (\mathbf{A}^T \oplus \mathbf{B}^T)$ $\mathbf{w}\langle\mathbf{m}\rangle \odot= (\mathbf{u} \oplus \mathbf{v})$
extract	$\mathbf{C}\langle\mathbf{M}\rangle \odot= \mathbf{A}^T(\boldsymbol{i},\boldsymbol{j})$ $\mathbf{w}\langle\mathbf{m}\rangle \odot= \mathbf{A}^T(:,j)$ $\mathbf{w}\langle\mathbf{m}\rangle \odot= \mathbf{w}(\boldsymbol{i})$	assign	$\mathbf{C}\langle\mathbf{M}\rangle\ (\boldsymbol{i},\boldsymbol{j}) \odot= \mathbf{A}^T$, $\mathbf{w}\langle\mathbf{m}\rangle\ (\boldsymbol{i}) \odot= \mathbf{u}$ $\mathbf{C}\langle\mathbf{M}\rangle\ (:,j) \odot= \mathbf{u}$, $\mathbf{C}\langle\mathbf{M}\rangle\ (i,:) \odot= \mathbf{u}^T$ $\mathbf{C}\langle\mathbf{M}\rangle\ (\boldsymbol{i},\boldsymbol{j}) \odot= s$, $\mathbf{w}\langle\mathbf{m}\rangle\ (\boldsymbol{i}) \odot= s$
apply	$\mathbf{C}\langle\mathbf{M}\rangle \odot= f(\mathbf{A}^T, s)$ $\mathbf{w}\langle\mathbf{m}\rangle \odot= f(\mathbf{u}, s)$	select (NEW)	$\mathbf{C}\langle\mathbf{M}\rangle \odot= \mathbf{A}^T\langle f(\mathbf{A}^T, s)\rangle$ $\mathbf{w}\langle\mathbf{m}\rangle \odot= \mathbf{u}\langle f(\mathbf{u}, s)\rangle$
build	$\mathbf{C} \leftarrow \{\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{x}\}$ $\mathbf{w} \leftarrow \{\boldsymbol{i}, \boldsymbol{x}\}$	extractTuples	$\{\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{x}\} \leftarrow \mathbf{A}$ $\{\boldsymbol{i}, \boldsymbol{x}\} \leftarrow \mathbf{u}$
extractElement	$s = \mathbf{A}(i, j)$ $s = \mathbf{u}(i)$	setElement	$\mathbf{C}(i, j) = s$ $\mathbf{w}(i) = s$

Notation: i, j – indices, $\boldsymbol{i},\boldsymbol{j}$ – (bold) index arrays, $\boldsymbol{x}$ – (bold) scalar array, **m** – 1D mask, **u,v,w** – vectors (column), **M** – 2D mask, **A,B,C** – matrices, ^{T} – optional transpose, ¬ - structural complement, r – clear output, $\odot$, $\oplus$, or $\otimes$ monoid/binary function, $\oplus.\otimes$ semiring, **blue** – optional parameters, **red** – optional modifiers

GraphBLAS Notation is Evolving

Operation	Notation	Operation	Notation
mxm	$\mathbf{C}\langle\mathbf{M}\rangle \odot\!= (\mathbf{A}^{\mathrm{T}} \oplus.\otimes \mathbf{B}^{\mathrm{T}})$	transpose	$\mathbf{C}\langle\mathbf{M}\rangle \odot\!= \mathbf{A}^{\mathrm{T}}$
mxv vxm	$\mathbf{w}\langle\mathbf{m}\rangle \odot\!= (\mathbf{A}^{\mathrm{T}} \oplus.\otimes \mathbf{u})$	reduce (row)	$\mathbf{w}\langle\mathbf{m}\rangle \odot\!= [\oplus_j \mathbf{A}^{\mathrm{T}}(:,j)]$
eWiseMult			
extract			$) \odot\!= \mathbf{u}$ $:) \odot\!= \mathbf{u}^{\mathrm{T}}$ $) \odot\!= s$
apply			
build			
extractElement	$s = \mathbf{A}(i,j)$ $s = \mathbf{u}(i)$	setElement	$\mathbf{C}(i,j) = s$ $\mathbf{w}(i) = s$

Main points of contention among GraphBLAS researchers:

1. Mask options: how to incorporate all the mask options (complement, structure-only, replace vs. merge, etc.) with an intuitive notation:

   $\mathbf{C}\langle \neg\, s(\mathbf{M}), r \rangle$???

2. eWiseAdd/eWiseMult: two options

   $\mathbf{C}\langle\mathbf{M}\rangle \odot\!= (\mathbf{A}^{\mathrm{T}} \otimes \mathbf{B}^{\mathrm{T}})$ $\quad$ $\mathbf{C}\langle\mathbf{M}\rangle \odot\!= (\mathbf{A}^{\mathrm{T}} \oplus \mathbf{B}^{\mathrm{T}})$

   $\mathbf{C}\langle\mathbf{M}\rangle \odot\!= (\mathbf{A}^{\mathrm{T}}\ \mathrm{op}_{\cap}\ \mathbf{B}^{\mathrm{T}})$ $\quad$ $\mathbf{C}\langle\mathbf{M}\rangle \odot\!= (\mathbf{A}^{\mathrm{T}}\ \mathrm{op}_{\cup}\ \mathbf{B}^{\mathrm{T}})$

Notation: i, j – indices, $\boldsymbol{i},\boldsymbol{j}$ – (bold) index arrays, $\boldsymbol{x}$ – (bold) scalar array, **m** – 1D mask, **u,v,w** – vectors (column), **M** – 2D mask, **A,B,C** – matrices, $^{\mathrm{T}}$ – optional transpose, ¬ - structural complement, r – clear output, ⊙, ⊕, or ⊗ monoid/binary function, ⊕.⊗ semiring, **blue** – optional parameters, **red** – optional modifiers

Example of Notation: Triangle Counting

Inputs: $\mathbf{A} \in \mathbb{B}^{nxn}$ (symmetric bool adjacency matrix)

Result: $t \in$ UINT64

Function *TriangleCount*

p = permutation to sort by degree descending

$\mathbf{A} = \mathbf{A}(p, p)$

$\mathbf{L} = \mathbf{A}\langle tril(\mathbf{A})\rangle$

$\mathbf{C}\langle s(\mathbf{L})\rangle = \mathbf{L}\ \ +.\&\ \ \mathbf{L}$

$t = [+_{ij}\ \mathbf{C}(i, j)]$

```
int TriangleCount(uint64_t *t, GrB_Matrix A, char *msg) {
  ...
  LAGraph_TRY( LAGraph_SortByDegree(&P, A, descending, msg) );

  GrB_TRY( GrB_extract(A,...,A, P, n, P, n, NULL) );

  GrB_TRY( GrB_select(L,...,GrB_TRIL, A, NULL) );

  GrB_TRY( GrB_mxm(C, L,..., GrB_PLUS_AND, L, L, NULL) );
  GrB_TRY( GrB_reduce(*t,..., GrB_PLUS_MONOID_UINT64, C, NULL) );

  LAGRAPH_FREE_ALL;
  return 0;
}
```

Algorithms

- **BFS: level and parent***
- SSSP: Bellman Ford, **Delta-stepping***
- Connected Components: LACC, **FastSV***, SCC-min-label, Boruvka's
- Minimum Spanning Forest: Boruvka's
- **Vertex Betweenness Centrality***
- PageRank: **GAP-PR***, Graphalytics-PR
- **Triangle Counting*** (6 methods)
- K-truss enumeration, All K-truss
- Community detection/clustering: label propagation, Louvain, markov, peer pressure
- Local clustering coefficient
- DNN inference

***GAP algorithms targeted for first release**

Utilities

- Loaders: matrix market, binary
- Memory management: override malloc/free
- Cached property operations: verify, compute, query, delete
- Sorting: vectors, graphs
- Sampling: graph degrees
- Output: "pretty print" graphs

Next Steps

- Continue to refactor existing algorithms
- Developing the evaluation and testing processes to ensure high-quality implementations:
  - Correctness
  - Robustness
  - Performance

***** REQUEST FOR PARTICIPATION *****

- Contribute new/improved algorithms (in order of preference)
  - Fork the repo, develop code in Experimental, **write tests**, issue Pull Requests
  - Email code to the working group
  - Publish (we *might* read and incorporate)

Repository Information

- Location
  - https://github.com/GraphBLAS/LAGraph
- License for all contributed code: BSD 2-clause
  - https://github.com/GraphBLAS/LAGraph/blob/master/LICENSE

- Contents [FUTURE]
  - Doc: documentation
  - Include: contains the LAGraph.h header file
  - Source: *will* contain stable (curated) source code for the library (contains Algorithms and Utilities)
  - Experimental: draft, experimental, submissions, etc. (contains Algorithms and Utilities)
  - Test: programs to verify correctness of LAGraph code

Questions?

Presenters / POC

Scott McMillan, PhD
Principal Research Engineer
Carnegie Mellon University / SEI
Email: smcmillan@sei.cmu.edu

Timothy G. Mattson, PhD
Senior Principal Engineer
Intel Corp.
Email: timothy.g.mattson@intel.com

Repository: https://github.com/GraphBLAS/LAGraph

Minutes at: https://github.com/GraphBLAS/LAGraph-Working-Group/tree/master/minutes