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
Copyright 2021 Carnegie Mellon University and Timothy G. Mattson

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.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

DM21-0172
A Brief History…

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

- Sep. 2013: GraphBLAS “position paper” at IEEE HPEC
- Jun. 2015: GraphBLASForum kickoff
- 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
- 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
- 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)
A Brief History…

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

- Construction “moves” matrix into LAGraph object
- Utilities provided for explicit computation of cached properties

```c
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...
};
```

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

// 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

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

```c
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

- **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)

```c
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
  - \( = 0 \rightarrow \text{success} \)
  - \( > 0 \rightarrow \text{algorithm specific warning code} \)
  - \( < 0 \rightarrow \text{algorithm specific error code} \)

- **Error messages**, \texttt{msg}
  - User-allocated char buffer of size \texttt{LAGRAPH_MSG_LEN}
  - Holds algorithm-specific error or warning messages

```c
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:

  ```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

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

Notation: $i, j$ – indices, $i,j$ – (bold) index arrays, $x$ – (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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Notation</th>
<th>Operation</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>mxm</td>
<td>$C\langle M \rangle \odot = (A^T \oplus \otimes B^T)$</td>
<td>transpose</td>
<td>$C\langle M \rangle \odot = A^T$</td>
</tr>
<tr>
<td>mxv</td>
<td>$w\langle m \rangle \odot = (A^T \oplus \otimes u)$</td>
<td>reduce (row)</td>
<td>$w\langle m \rangle \odot = [\oplus_i A^T(i,:)]$</td>
</tr>
<tr>
<td>vxm</td>
<td>$w\langle m \rangle \odot = (A^T \otimes \oplus u)$</td>
<td>reduce (scalar)</td>
<td>$w\langle m \rangle \odot = [\oplus_i A^T(i,:)]$</td>
</tr>
<tr>
<td>eWiseMult</td>
<td>$C\langle M \rangle \odot = (A^T \otimes B^T)$</td>
<td>eWiseAdd</td>
<td>$C\langle M \rangle \odot = (A^T \oplus B^T)$</td>
</tr>
<tr>
<td>extract</td>
<td>$C\langle - s(M), r \rangle$</td>
<td>eWiseMult</td>
<td>$C\langle M \rangle \odot = (A^T \otimes B^T)$</td>
</tr>
<tr>
<td>apply</td>
<td>$C\langle M \rangle \odot = (A^T \cap \otimes B^T)$</td>
<td>apply</td>
<td>$C\langle M \rangle \odot = (A^T \cup B^T)$</td>
</tr>
<tr>
<td>build</td>
<td>$w\langle m \rangle \odot = (A \cap u)$</td>
<td>build</td>
<td>$w\langle m \rangle \odot = (A \cup u)$</td>
</tr>
<tr>
<td>extractElement</td>
<td>$s = A(i,j)$</td>
<td>setElement</td>
<td>$C(i,j) = s$</td>
</tr>
<tr>
<td></td>
<td>$s = u(i)$</td>
<td></td>
<td>$w(i) = s$</td>
</tr>
</tbody>
</table>

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:

   $C\langle - s(M), r \rangle$ ???

2. eWiseAdd/eWiseMult: two options

   $C\langle M \rangle \odot = (A^T \otimes B^T)$
   $C\langle M \rangle \odot = (A^T \oplus B^T)$

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

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

Result: $t \in \text{UINT64}$

Function $\text{TriangleCount}$

$p =$ permutation to sort by degree descending

$A = A(p, p)$

$L = A(\text{tril}(A))$

$C(\text{s}(L)) = L + L$

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

```c
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