Genomics, Graphs and the GraphBLAS

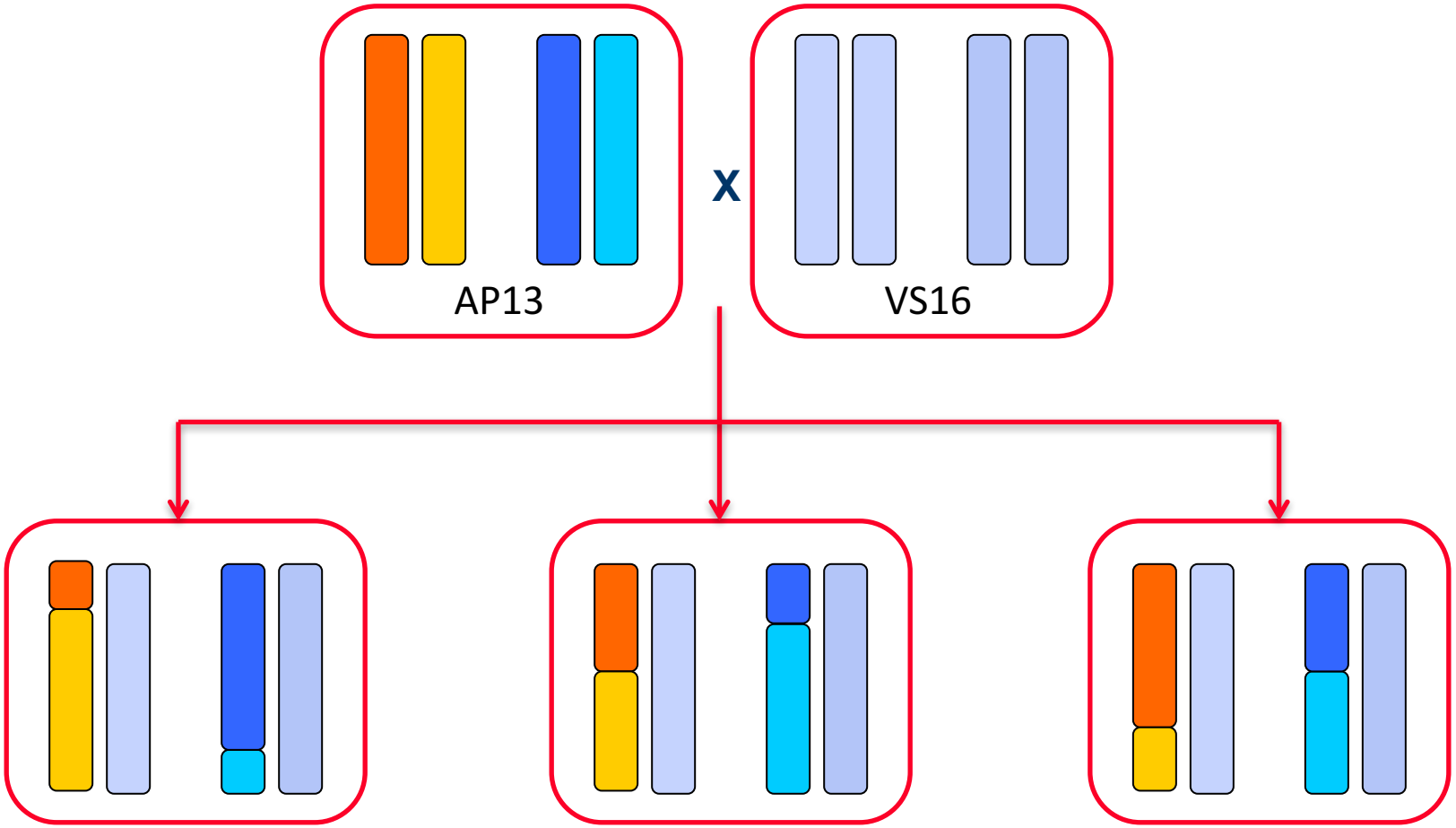
Aydın Buluç
Computational Research Division, LBNL
EECS Department, UC Berkeley

Graphs Across Domains Workshop
Berkeley Institute of Data Science
March 27, 2018

Outline

- **Constructing genetic linkage maps and its graph theoretical formulations**
- (Protein) sequence similarity graphs and their clustering
- GraphBLAS: Linear-algebraic building blocks for graph algorithms

Genetic mapping with millions of markers



F1 recombinants track “orange” vs “yellow” in offspring

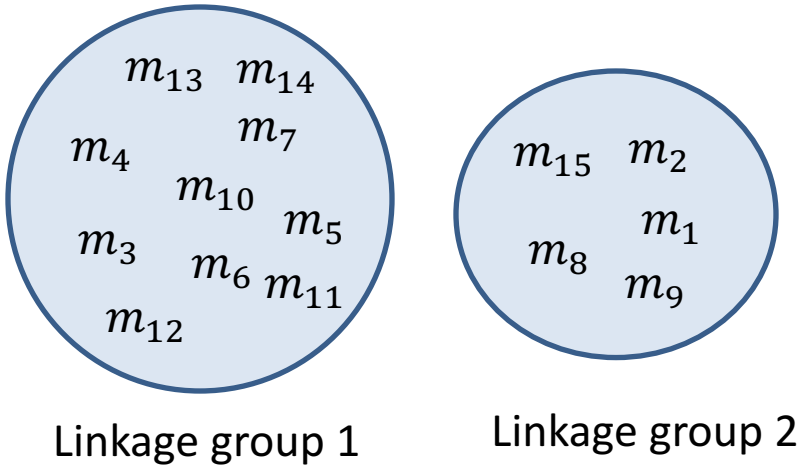
Chapman, J.A., Mascher, M., Buluç, A., Barry, K., Georganas, E., … Rokhsar, D., 2015. A whole-genome shotgun approach for assembling and anchoring the hexaploid bread wheat genome. *Genome biology*

Data

	i_1	i_2	i_3	i_4	i_5	i_6
m_1	A	B	-	-	A	-
m_2	A	B	A	A	B	A
m_3	A	A	-	-	-	B
m_4	A	-	B	-	B	B
m_5	B	-	B	A	-	A
m_6	A	A	B	A	-	-
m_7	-	-	-	A	B	B
m_8	A	B	A	B	-	A
m_9	A	B	-	B	-	-
m_{10}	B	B	B	-	A	A
m_{11}	A	A	A	A	B	B
m_{12}	B	-	A	B	A	-
m_{13}	B	B	-	A	A	-
m_{14}	-	-	-	B	A	A
m_{15}	B	-	-	A	A	B

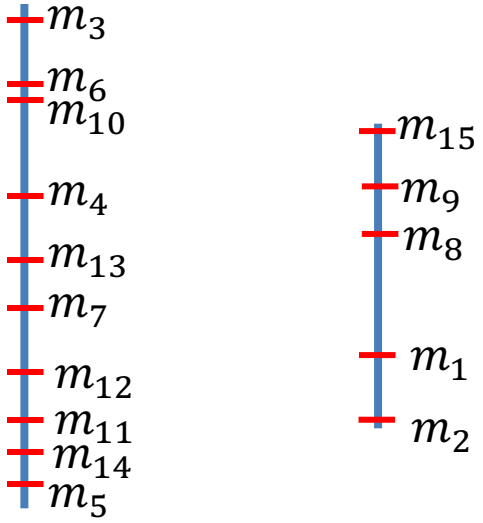
(missing data)

cluster



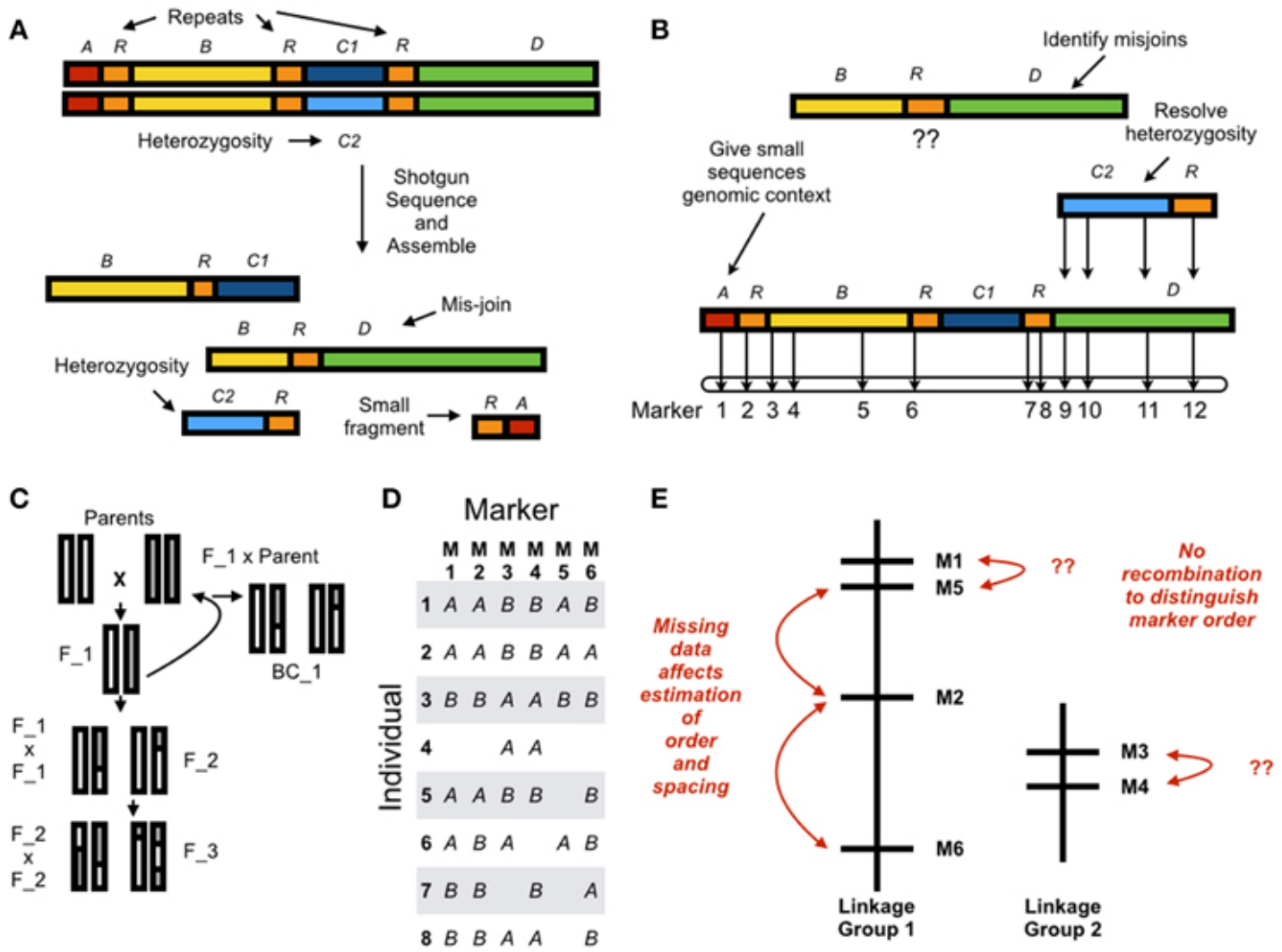
Linkage group 1 Linkage group 2

order



Linkage group 1 Linkage group 2

Genetic mapping: 2010s motivation



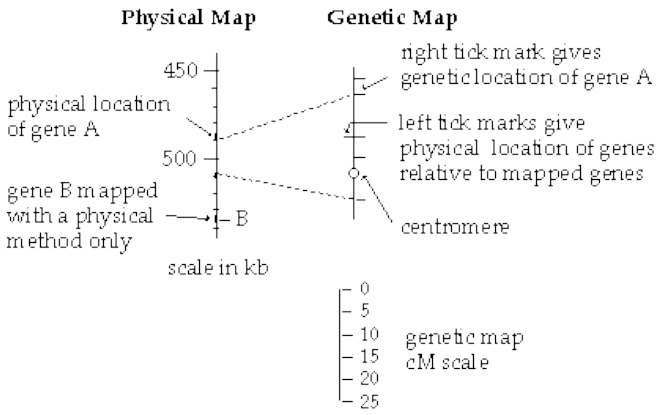
Fierst, Janna L. "Using linkage maps to correct and scaffold de novo genome assemblies: methods, challenges, and computational tools." *Frontiers in genetics* 6 (2015): 220.

Linkage disequilibrium makes map construction feasible

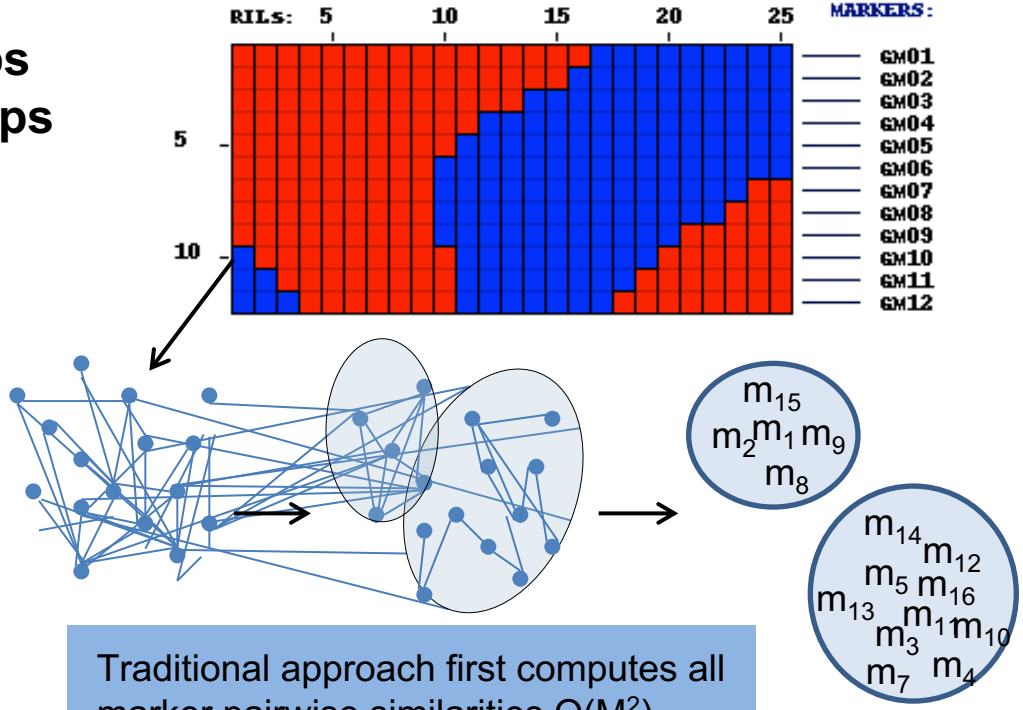
- Genetic maps are constructed by recombination frequencies.
- Markers (think of single nucleotide polymorphisms – or SNPs – for simplicity) that are physically close to each other are less likely to segregate during meiosis.

Procedure:

1. **Clustering for linkage groups**
2. **Marker ordering within groups**
3. **Genetic distance estimation**



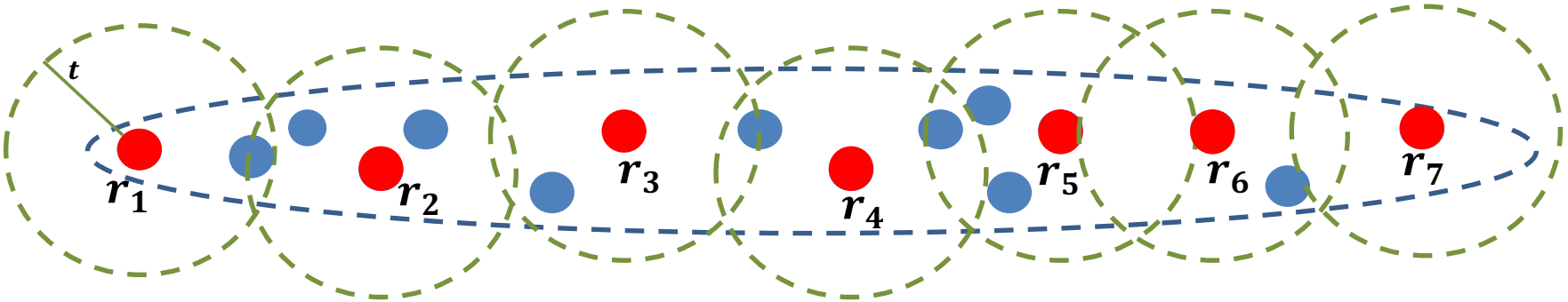
M = markers, P = size of offspring (MxP Matrix)



Traditional approach first computes all marker pairwise similarities $O(M^2)$

Graph Problems in Genetic Mapping #1

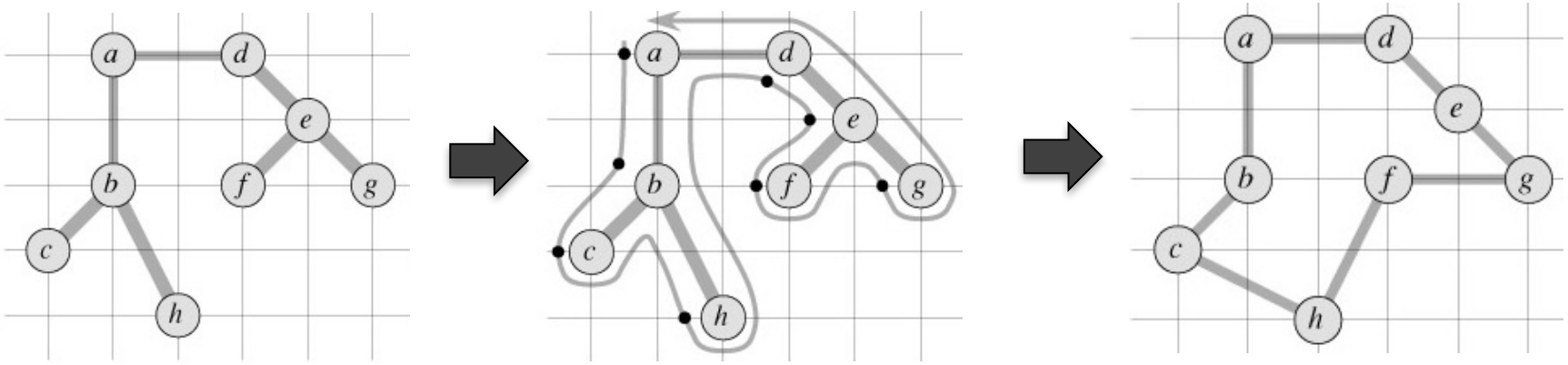
- Linkage group construction is traditionally done via **single-linkage clustering**
  - Naïve O(M^2) computation, metric tricks don't seem to apply due to the use of LOD score for distance.
  - **Bubblecluster** helps reduce this to O(M log(M))
- Main idea: Clusters have a "quasi-linear structure"
  - Linear as they represent chromosomes
  - Quasi because of sequencing errors and missing data



Strnadova, V., Buluç, A., Chapman, J., Gilbert, J.R., Gonzalez, J., Jegelka, S., Rokhsar, D. and Oliker, L., 2014. Efficient and accurate clustering for large-scale genetic mapping. In *BIBM*

Graph Problems in Genetic Mapping #2

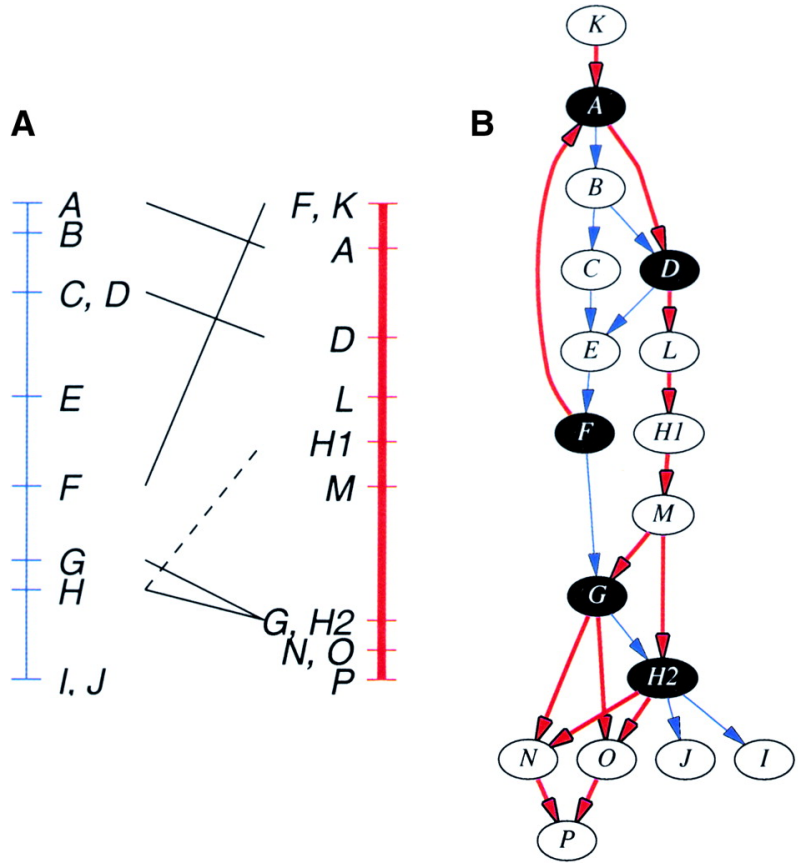
- Ordering step is naively a Travelling Salesman Problem (TSP)
- Not feasible for many markers; but the marker count does not dictate the complexity, **distinguishable markers (a.k.a. bins)** do. The latter is limited by population size.
- Even then, TSP is overkill.
- MSTMap exploits the 2-approximation of minimum spanning tree to TSP



Wu, Yonghui, et al. "Efficient and accurate construction of genetic linkage maps from the minimum spanning tree of a graph." *PLoS genetics* 4.10 (2008):

Graph Problems in Genetic Mapping #3

- **Integrating two genetic/physical/optical maps**
- What to do when two genetic maps differ?



- Identify strongly connected components (SCCs).
- Contract them into supervertices
- Rest of the graph has consistent ordering

Yap, I.V., Schneider, D., Kleinberg, J., Matthews, D., Cartinhour, S. and McCouch, S.R., 2003. A graph-theoretic approach to comparing and integrating genetic, physical and sequence-based maps. Genetics

Outline

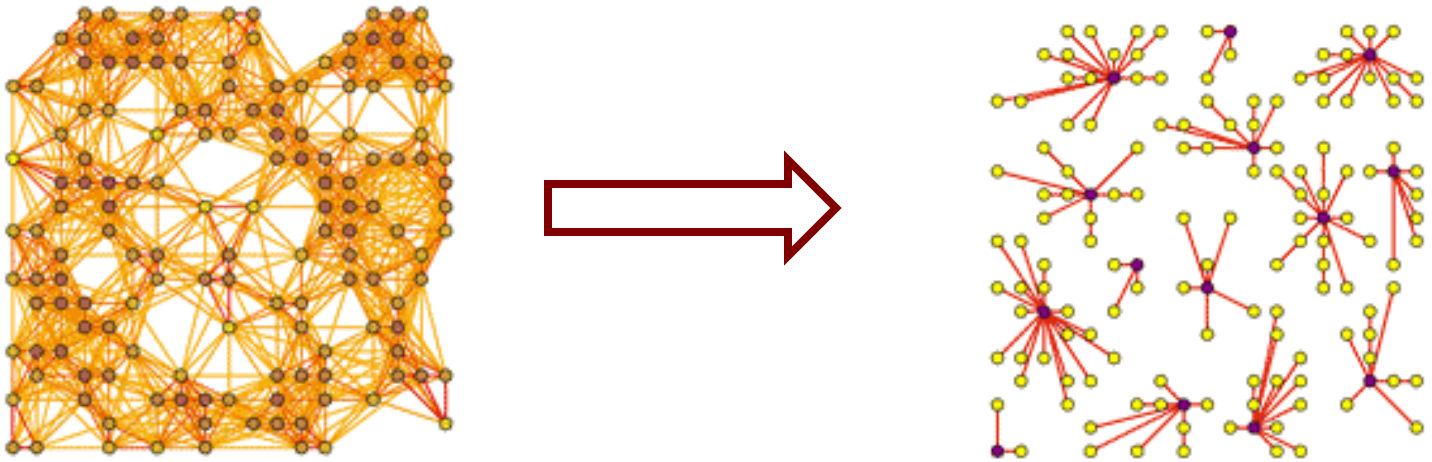
- Constructing genetic linkage maps and its graph theoretical formulations
- **(Protein) sequence similarity graphs and their clustering**
- GraphBLAS: Linear-algebraic building blocks for graph algorithms

Identifying protein families

- A **protein family**: group of proteins that share a common evolutionary origin, reflected by their related functions and similarities in sequence or structure

Input: pairwise similarities between proteins (Sparse)

Output: clusters of similar proteins



- **Desired scale:** 10s of billions of genes/proteins, trillions of nonzero pairwise similarities

Markov Cluster (MCL) Algorithm

- MCL simulates random walks in a graph

Stijn van Dongen, Graph Clustering by Flow Simulation. PhD thesis, University of Utrecht, May 2000

- One of the most popular algorithms in community for finding protein families from sequence data

"…but MCL continued to outperform all other algorithms after a threshold was applied. As a result, we believe researchers may now more confidently use the time-efficient MCL clustering technique for most of their protein sequence analysis needs."

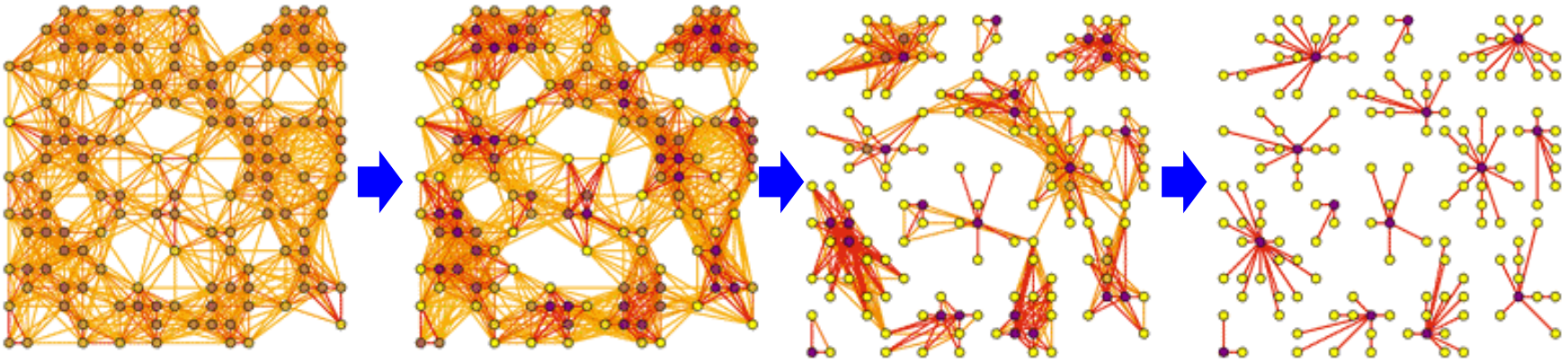
Apeltsin, Leonard, et al. "Improving the quality of protein similarity network clustering algorithms using the network edge weight distribution." *Bioinformatics* 27.3 (2010): 326-333.

"This analysis shows that MCL is remarkably robust to graph alterations…"

Brohee, S. and Van Helden, J., 2006. Evaluation of clustering algorithms for protein-protein interaction networks. *BMC bioinformatics*

Markov Cluster Algorithm (MCL)

Widely popular and successful algorithm for discovering clusters in protein interaction and protein similarity networks



At each iteration:

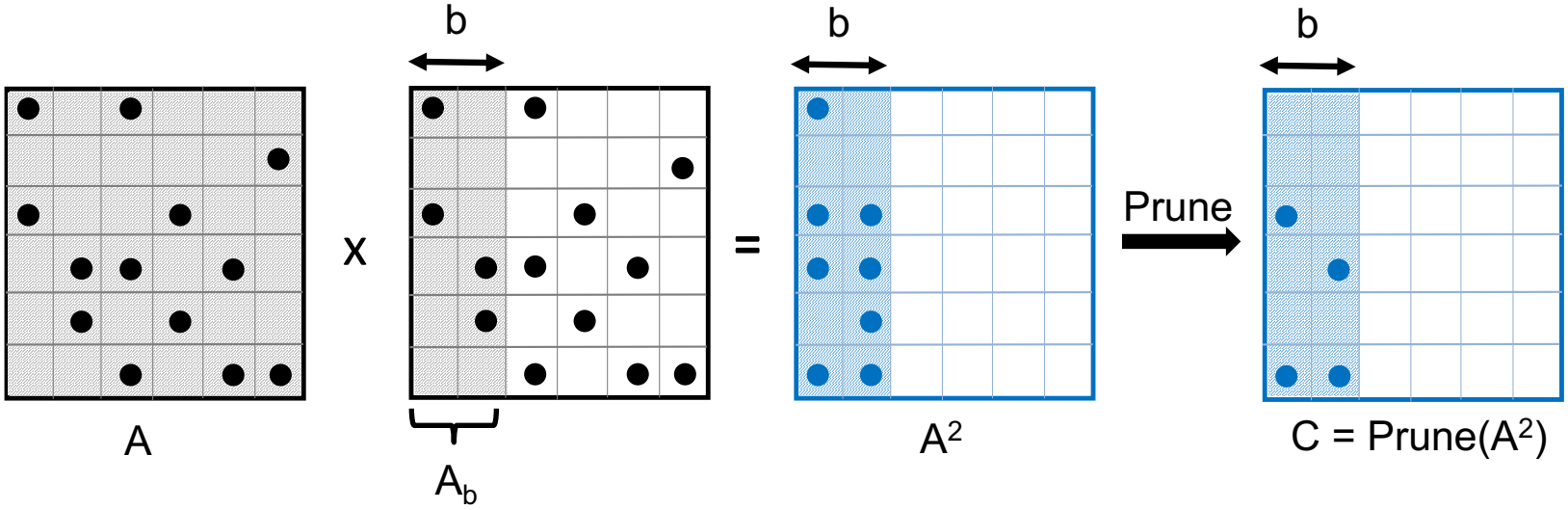
Step 1 (Expansion): Squaring the matrix while pruning (a) small entries, (b) denser columns

Naïve implementation: sparse matrix-matrix product (SpGEMM), followed by column-wise top-K selection and column-wise pruning

Step 2 (Inflation) : taking powers entry-wise

HipMCL: High-performance MCL

MCL process is both **computationally expensive** and **memory hungry**, limiting the sizes of networks that can be clustered



- HipMCL overcomes such limitation via **sparse parallel algorithms**.
- **Up to 1000X times faster** than original MCL with same accuracy.
- Easily clusters a network of ~75M nodes with ~68B edges in ~2.4 hours using ~2000 nodes of Cori/NERSC.

A. Azad, G. Pavlopoulos, C. Ouzounis, N. Kyrpides, A. Buluç; HipMCL: a high-performance parallel implementation of the Markov clustering algorithm for large-scale networks, *Nucleic Acids Research, 2018*

Outline

- Constructing genetic linkage maps and its graph theoretical formulations
- (Protein) sequence similarity graphs and their clustering
- **GraphBLAS: Linear-algebraic building blocks for graph algorithms**

The GraphBLAS effort

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)

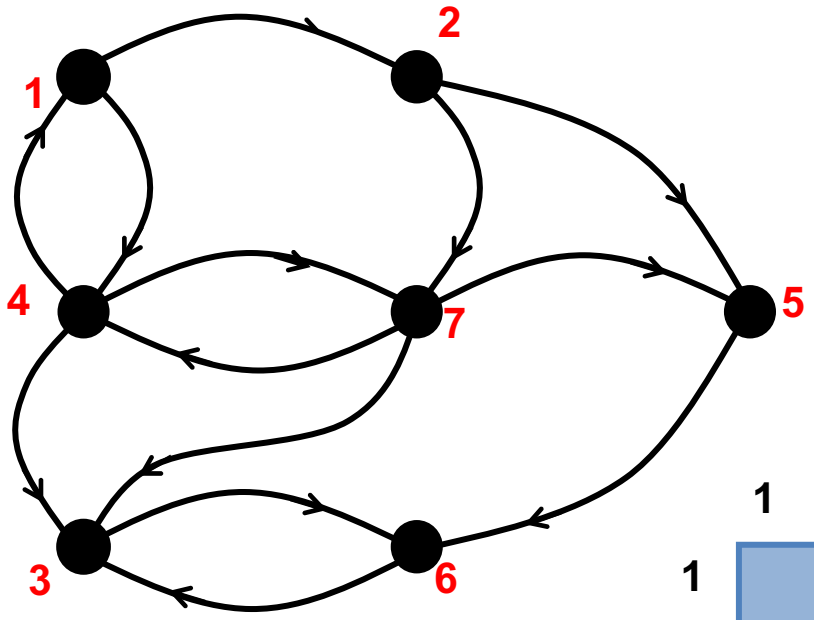
Abstract-- It is our view that the state of the art in constructing a large collection of graph algorithms in terms of linear algebraic operations is mature enough to support the emergence of a standard set of primitive building blocks. This paper is a position paper defining the problem and announcing our intention to launch an open effort to define this standard.

- The GraphBLAS Forum: http://graphblas.org
- IEEE Workshop on Graph Algorithms Building Blocks (at IPDPS): http://www.graphanalysis.org/workshop2018.html

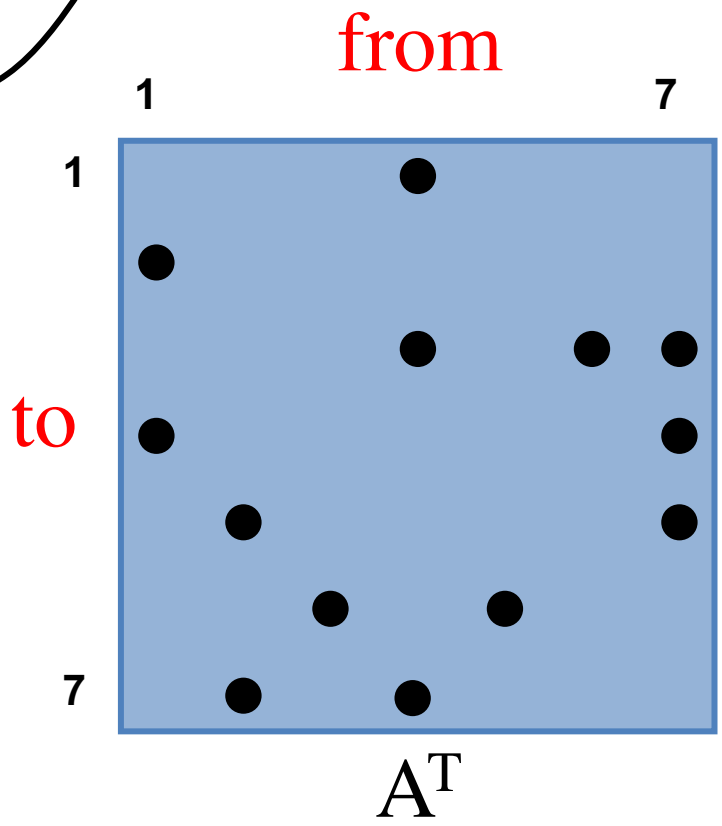
Purpose of GraphBLAS

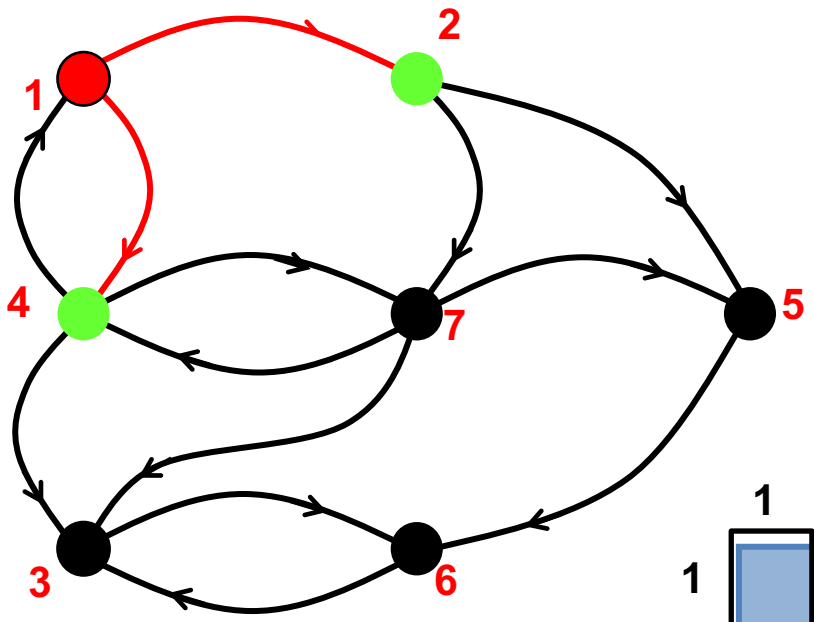
- Combinatorial graph algorithms, such as those involving graph traversals, did not map well to existing hardware and did not parallelize well.
- GraphBLAS is about making such **traversal-based and other combinatorial graph algorithms** faster
- Its primary motivation and drive is **not spectral methods**
- Instead, GraphBLAS examples include **betweenness centrality, Markov clustering, breadth-first search, maximal independent sets, PageRank, triangle counting, bipartite graph matching, graph ordering,** and **connected components**.
- The vision for linear-algebraic graph algorithms (there is a SIAM book for that) and several high performance systems based on the idea existed (Combinatorial BLAS, GraphMat, GPI).
- Standardization is to avoid divergence of APIs.

A.Buluç, T. Mattson, S. McMillan, J. Moreira, **C. Yang**. "The GraphBLAS C API Specification", version 1.1



Breadth-first search using matrix algebra

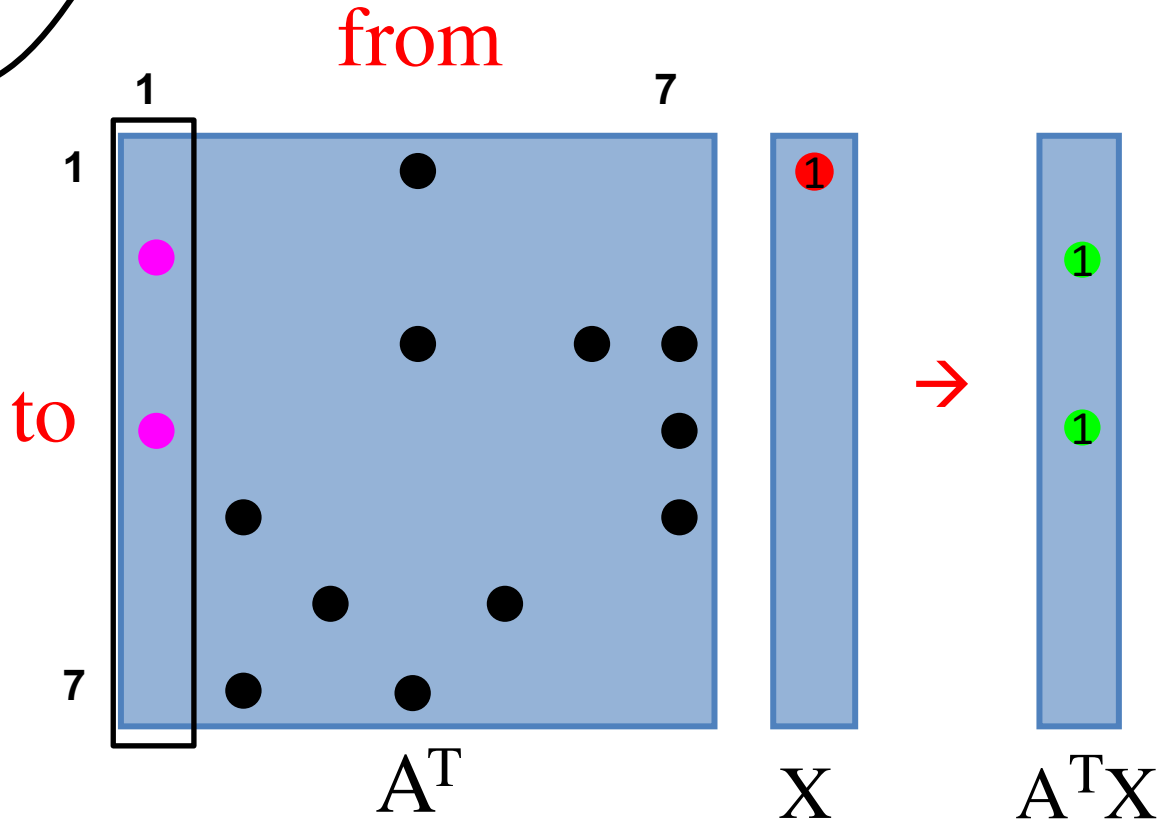




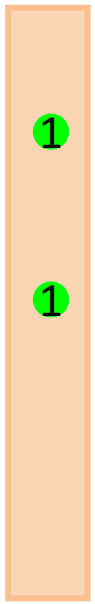
Replace scalar operations

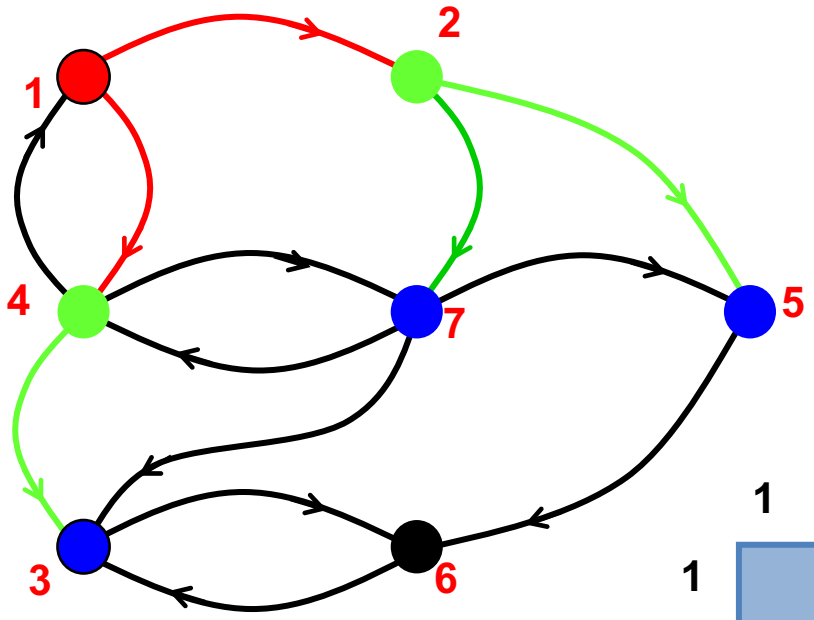
Multiply -> select

Add -> minimum



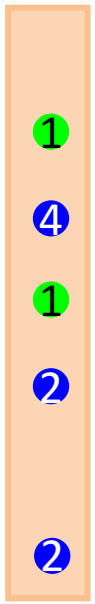
parents:





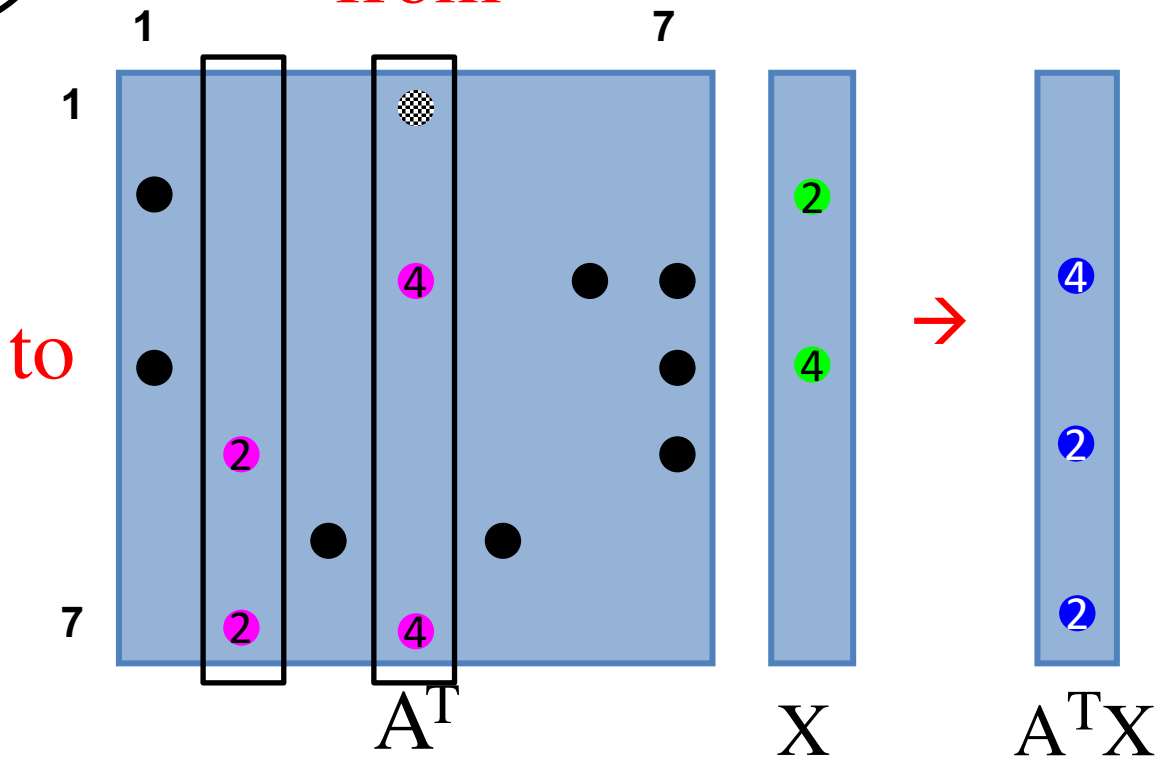
Select vertex with
minimum label as parent

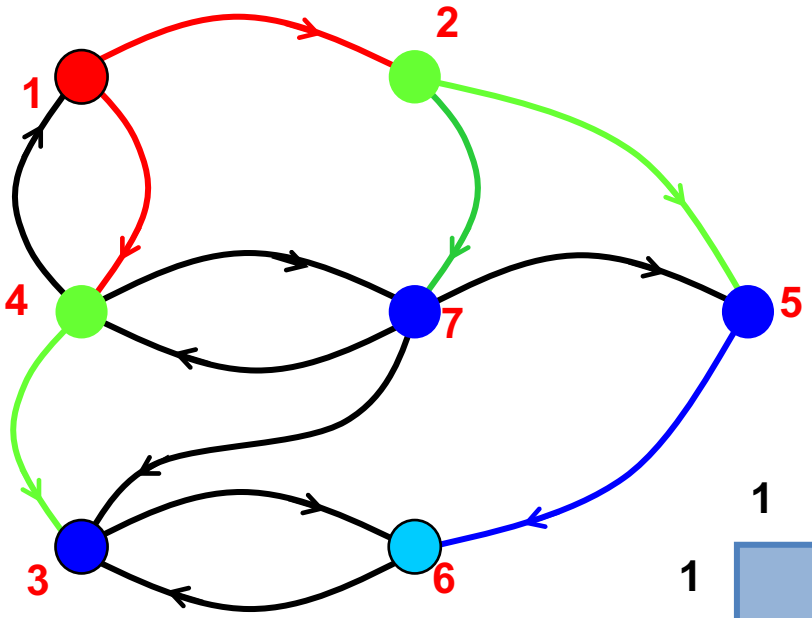
parents:



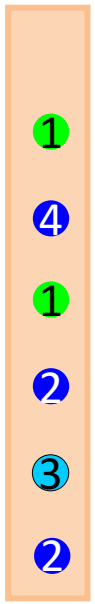
from

to



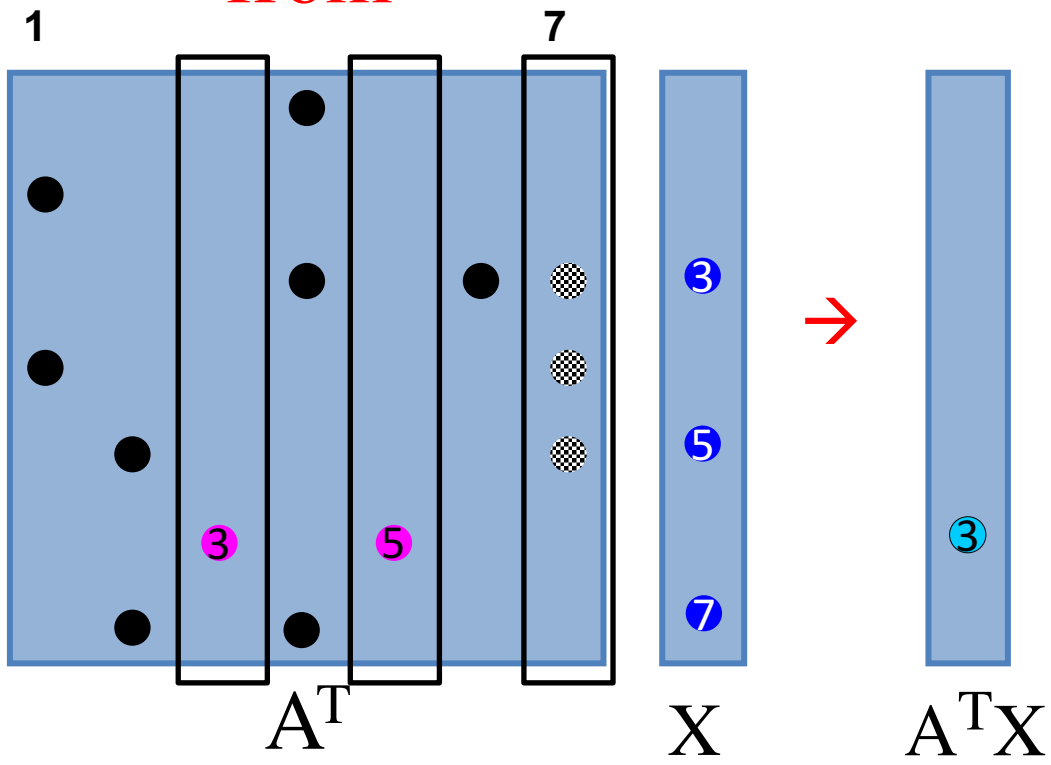


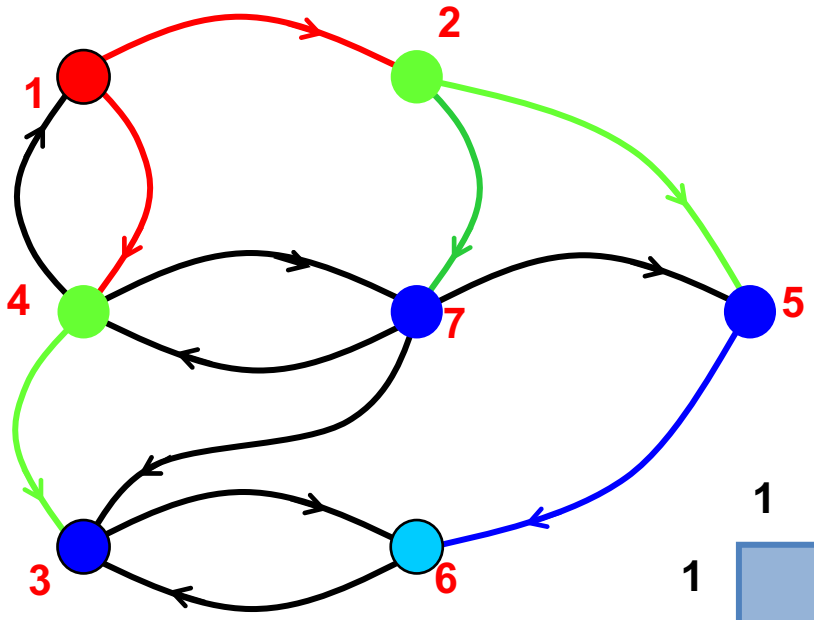
parents:



from

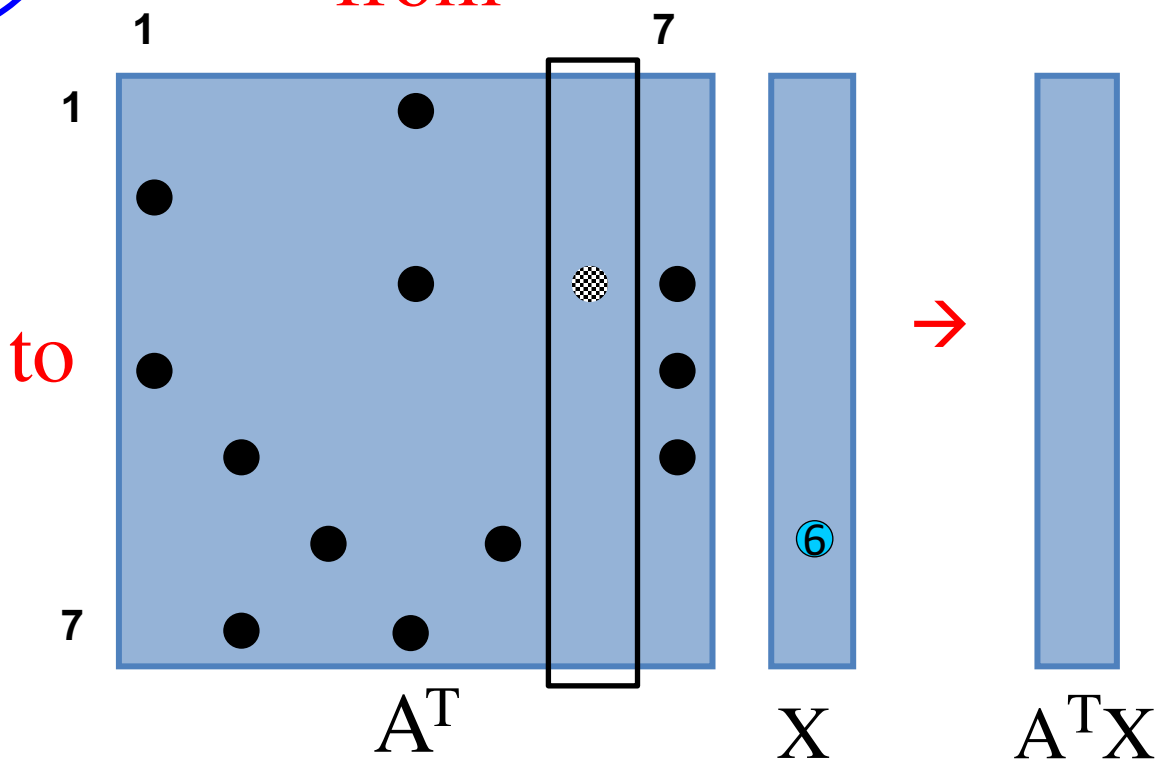
to





from

to



Breadth-First Search in GraphBLAS

```
GrB_Vector q;                                   // vertices visited in each level
GrB_Vector_new(&q,GrB_BOOL,n);                  // Vector<bool> q(n) = false
GrB_Vector_setElement(q,(bool)true,s);          // q[s] = true, false everywhere else

GrB_Monoid Lor;                                 // Logical-or monoid
GrB_Monoid_new(&Lor,GrB_LOR,false);

GrB_Semiring Boolean;                           // Boolean semiring
GrB_Semiring_new(&Boolean,Lor,GrB_LAND);

GrB_Descriptor desc;                            // Descriptor for vxm
GrB_Descriptor_new(&desc);
GrB_Descriptor_set(desc,GrB_MASK,GrB_SCMP);     // invert the mask
GrB_Descriptor_set(desc,GrB_OUTP,GrB_REPLACE);// clear the output before assignment

GrB_UnaryOp apply_level;
GrB_UnaryOp_new(&apply_level,return_level,GrB_INT32,GrB_BOOL);

/*
 * BFS traversal and label the vertices.
 */
level = 0;
GrB_Index nvals;
do {
  ++level;                                      // next level (start with 1)
  GrB_apply(*v,GrB_NULL,GrB_PLUS_INT32,apply_level,q,GrB_NULL);  // v[q] = level
  GrB_vxm(q,*v,GrB_NULL,Boolean,q,A,desc);      // q[!v] = q ||.&& A ; finds all the
                                                // unvisited successors from current q
  GrB_Vector_nvals(&nvals, q);
} while (nvals);                                // if there is no successor in q, we are done.
```

GraphBLAS C API Spec (http://graphblas.org)

- **Goal:** A crucial piece of the GraphBLAS effort is to translate the mathematical specification to an actual Application Programming Interface (API) that
  i. is faithful to the mathematics as much as possible, and
  ii. enables efficient implementations on modern hardware.
- **Impact:** All graph and machine learning algorithms that can be expressed in the language of linear algebra
- **Innovation:** Function signatures (e.g. mxm, vxm, assign, extract), parallelism constructs (blocking v. non-blocking), fundamental objects (masks, matrices, vectors, descriptors), a hierarchy of algebras (functions, monoids, and semiring)

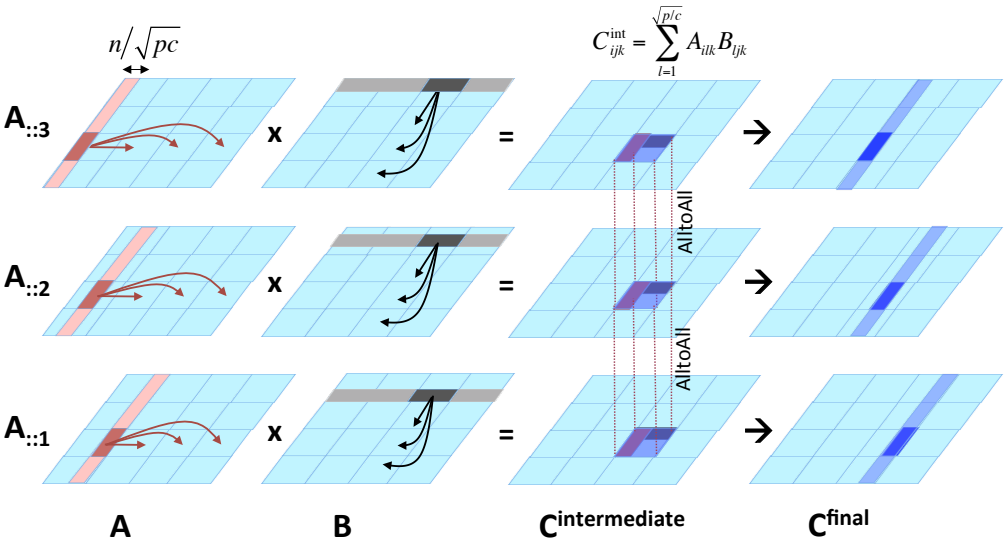
```
GrB_info GrB_mxm(GrB_Matrix          *C,      // destination
                 const GrB_Matrix     Mask,
                 const GrB_BinaryOp   accum,
                 const GrB_Semiring   op,
                 const GrB_Matrix     A,
                 const GrB_Matrix     B
                [, const Descriptor   desc]);
```

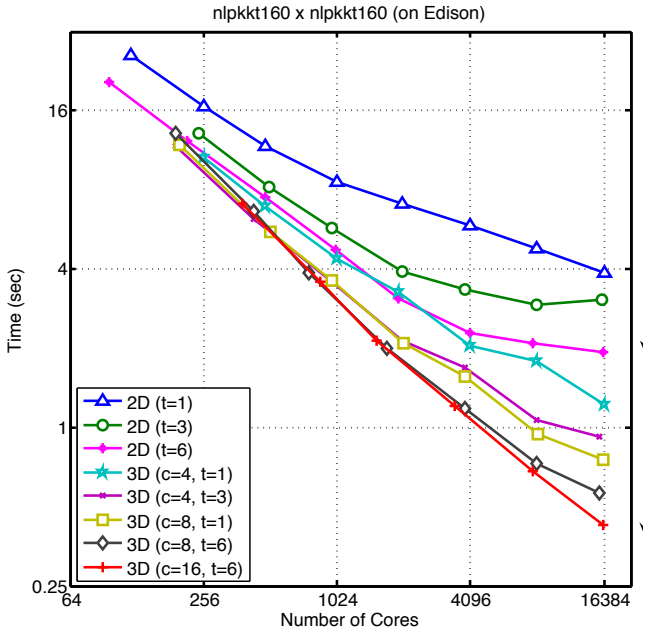
$$C(\neg M) \oplus= A^T \oplus.\otimes B^T$$

Aydin Buluc, Timothy Mattson, Scott McMillan, Jose Moreira, and Carl Yang. Design of the GraphBLAS API for C. In Intl. Parallel & Distributed Processing Symposium Workshops (IPDPSW), 2017.

Parallel algorithms for sparse-matrix- sparse matrix multiplication (SpGEMM)

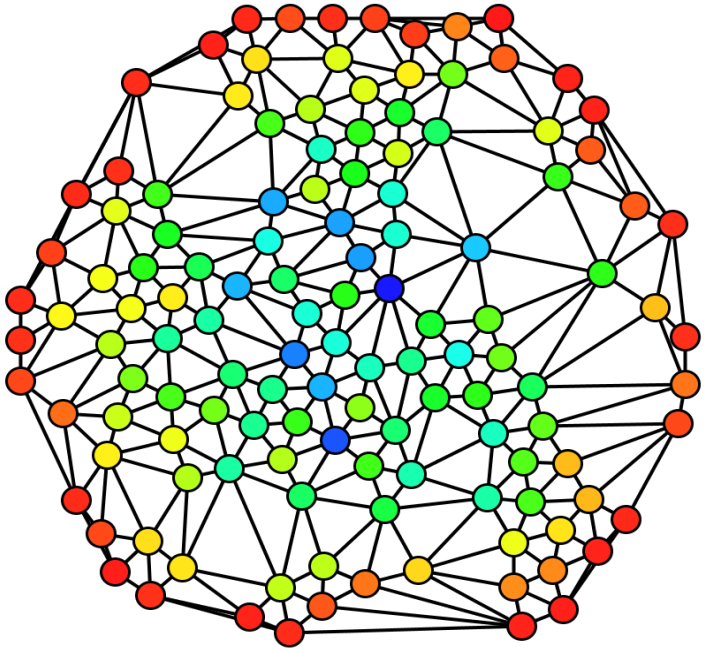
- **Goal:** More scalable SpGEMM algorithms in shared and distributed-memory
- **Applications:** Algebraic multigrid (AMG) restriction, graph computations, quantum chemistry, data mining, interior-point optimization
- **Algorithmic innovations:** (1) Novel shared-memory kernel for in-node parallelism, (2) Split-3D-SpGEMM: an efficient implementation of communication-avoiding SpGEMM
- **Performance:** Split-3D-SpGEMM with new shared-memory kernel (red) beats old state-of-the-art (blue) by 8X at large concurrencies





A. Azad, G. Ballard, **A. Buluç**, J. Demmel, L. Grigori, O. Schwartz, S. Toledo, S. Williams. Exploiting multiple levels of parallelism in sparse matrix-matrix multiplication. SIAM Journal of Scientific Computing (SISC), 2016.

Betweenness Centrality



Definition:

$C_B(v)$: Among all the shortest paths, what fraction of them pass through the node of interest?

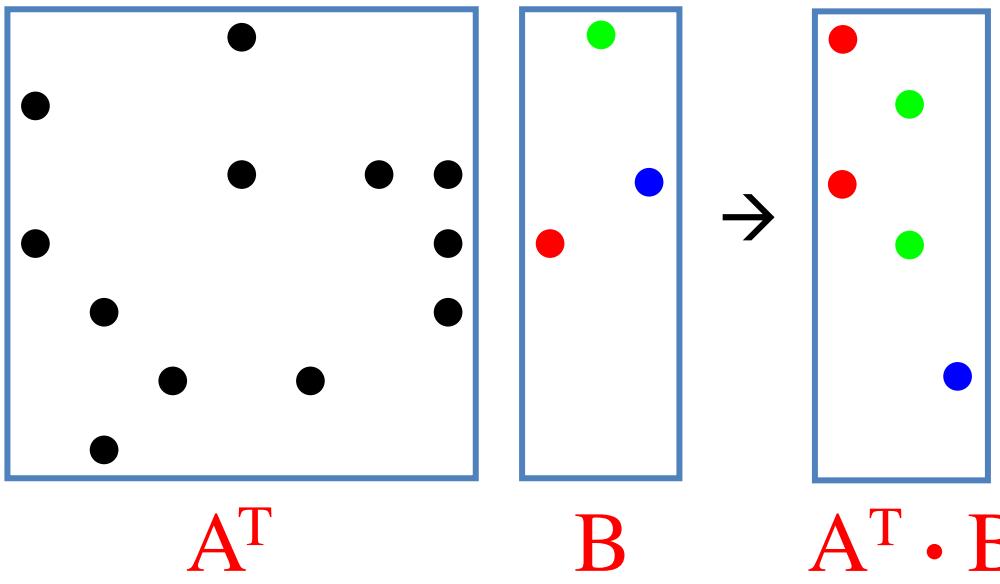
$$BC(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

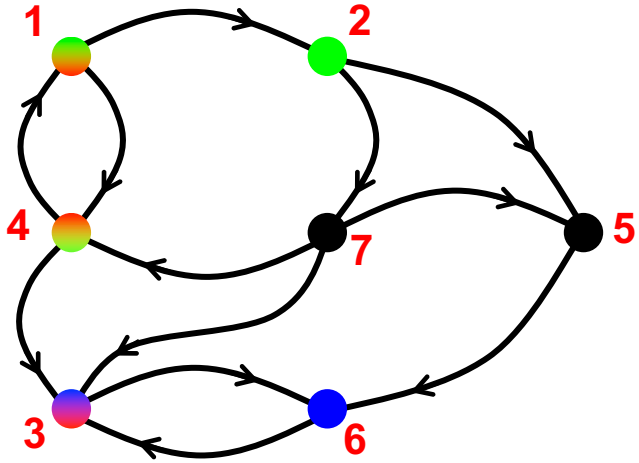
σ_{st} is the number of shortest paths between vertices s and t

σ_{st} (v) is the number of such paths that pass through vertex v

- APSP is wasteful for sparse graphs
- Brandes' algorithm is O(mn) for unweighted graphs

Driver: Multiple-source breadth-first search





- Sparse array representation => space efficient
- Sparse matrix-matrix multiplication => work efficient
- **Three possible levels of parallelism: searches, vertices, edges**
- Highly-parallel implementation for Betweenness Centrality*

*: A measure of influence in graphs, based on shortest paths

Forward sweep of BC in GraphBLAS C API

```
#include "GraphBLAS.h"

GrB_Info BC_update(GrB_Vector *delta, GrB_Matrix A, GrB_Index *s, GrB_Index nsver)
{
  GrB_Index n;
  GrB_Matrix_nrows(&n, A);                              // n = # of vertices in graph
  GrB_Vector_new(delta,GrB_FP32,n);                     // Vector<float> delta(n)
  GrB_Monoid Int32Add;                                  // Monoid <int32_t,+,0>
  GrB_Monoid_new(&Int32Add,GrB_INT32,GrB_PLUS_INT32,0);
  GrB_Semiring Int32AddMul;               // Semiring <int32_t,int32_t,int32_t,+,*,0>
  GrB_Semiring_new(&Int32AddMul,Int32Add,GrB_TIMES_INT32);

  GrB_Descriptor desc_tsr;                              // Descriptor for BFS phase mxm

  GrB_Descriptor_new(&desc_tsr);
  GrB_Descriptor_set(desc_tsr,GrB_INP0,GrB_TRAN);     // transpose of the adjacency matrix
  GrB_Descriptor_set(desc_tsr,GrB_MASK,GrB_SCMP);     // structural complement of the mask
  GrB_Descriptor_set(desc_tsr,GrB_OUTP,GrB_REPLACE);  // clear output before result is stored

  // index and value arrays needed to build numsp
  GrB_Index *i_nsver = malloc(sizeof(GrB_Index)*nsver);
  int32_t   *ones    = malloc(sizeof(int32_t)*nsver);
  for(int i=0; i<nsver; ++i) {
    i_nsver[i] = i;
    ones[i] = 1;
  }
…
```

Forward sweep of BC in GraphBLAS C API

```
…
  GrB_Matrix numsp;  // Its nonzero structure holds all vertices that have been discovered
  GrB_Matrix_new( , GrB_INT32, n, nsver); // also stores # of shortest paths so far

  GrB_Matrix_build( ,GrB_NULL,GrB_NULL,s,i_nsver,ones,nsver,GrB_PLUS_INT32,GrB_NULL);
  free(i_nsver); free(ones);

  GrB_Matrix frontier;     // Holds the current frontier where values are path counts.
  GrB_Matrix_new(&frontier, GrB_INT32, n, nsver); // Initialized: neighbors of each source
  GrB_extract(&frontier,numsp,GrB_NULL,A,GrB_ALL,n,s,nsver,desc_tsr);

  // The memory for an entry in sigmas is only allocated within the do-while loop if needed
  GrB_Matrix *sigmas = malloc(sizeof(GrB_Matrix)*n);     // n is an upper bound on diameter
  int32_t d = 0;                                         // BFS level number
  int32_t nvals = 0;                                     // nvals == 0 when BFS phase is complete
  do {  // ---------------------- The BFS phase (forward sweep) --------------------------
    GrB_Matrix_new(&(sigmas[d]), GrB_BOOL, n, nsver);
    // sigmas[d](:,s) = d^th level frontier from source vertex s

    GrB_apply(&(sigmas[d]),GrB_NULL,GrB_NULL,GrB_IDENTITY_BOOL,frontier,GrB_NULL);
    GrB_eWiseAdd( ,GrB_NULL,GrB_NULL,Int32Add,numsp,frontier,GrB_NULL);
    // numsp += frontier (accum path counts)

    GrB_mxm(&frontier,numsp,GrB_NULL,Int32AddMul,A,frontier,desc_tsr);
    // f<!numsp> = A' +.* f (update frontier)
    GrB_Matrix_nvals(&nvals,frontier)
    d++;
  } while (nvals);
…
```

Forward sweep of BC in GraphBLAS C API

```
...
  GrB_Matrix nu
  GrB_Matrix_ne

  GrB_Matrix_bu
  free(i_nsver)

  GrB_Matrix fr
  GrB_Matrix_ne
  GrB_extract(&

  // The memory
  GrB_Matrix *s
  int32_t d = 0
  int32_t nvals
  do {  // ----
    GrB_Matrix_
    // sigmas[d

    GrB_apply(&
    GrB_eWiseAd
    // numsp += frontier (accum path counts)

    GrB_mxm(&frontier,numsp,GrB_NULL,Int32AddMul,A,frontier,desc_tsr);
    // f<!numsp> = A' +.* f (update frontier)
    GrB_Matrix_nvals(&nvals,frontier)
    d++;
  } while (nvals);
...
```

- The GrB_mxm call forms the next frontier in one step by both expanding the current frontier (i.e., discovering the 1-hop neighbors of the set of vertices in the current frontier) and pruning the vertices that have already been discovered.
- The former is achieved by setting the descriptor, desc_tsr, to use the transpose of the adjacency matrix. The latter is achieved by setting the descriptor to use the structural complement of the mask and by passing the numsp matrix as the mask parameter.
- The implicit cast of numsp to Boolean allows GrB_mxm to interpret numsp as the set of previously discovered vertices.
- Note that the descriptor is also set to GrB_REPLACE to ensure that the frontier is overwritten with new values.

Conclusions

- GraphBLAS enables one to efficiently cast graph algorithms and machine learning methods into the languages of sparse matrices
- While elegant and efficient for problems that fit into the linear algebra framework, it is admittedly not fully universal.
- The standard definition by the C API group and a compliant implementation by Tim Davis available at http://graphblas.org
- More parallel implementations in the works. Currently one can use approximate GraphBLAS implementations from Combinatorial BLAS, Kokkos, and Cyclops Tensor Framework.