GraphBLAS: A linear algebraic approach for high-performance graph algorithms

Gábor Szárnyas
szarnyas@mit.bme.hu

MŰEGYETEM 1782

ftsrg

Hungarian Academy of Sciences

WHAT MAKES GRAPH PROCESSING DIFFICULT?

connectedness — the "curse of connectedness"

computer architectures — contemporary computer architectures are good at processing linear and hierarchical data structures, such as *Lists*, *Stacks*, or *Trees*

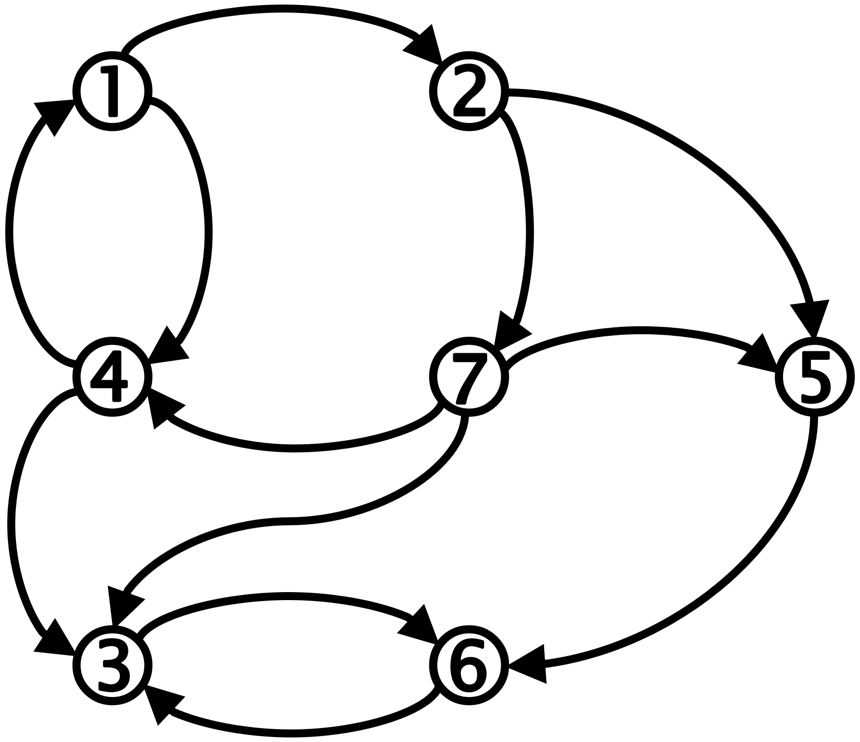
caching and parallelization — a massive amount of random data access is required, CPU has frequent cache misses, and implementing parallelism is difficult

B. Shao, Y. Li, H. Wang, H. Xia (Microsoft Research),
Trinity Graph Engine and its Applications,
IEEE Data Engineering Bulleting 2017

Graph processing in linear algebra

ADJACENCY MATRIX

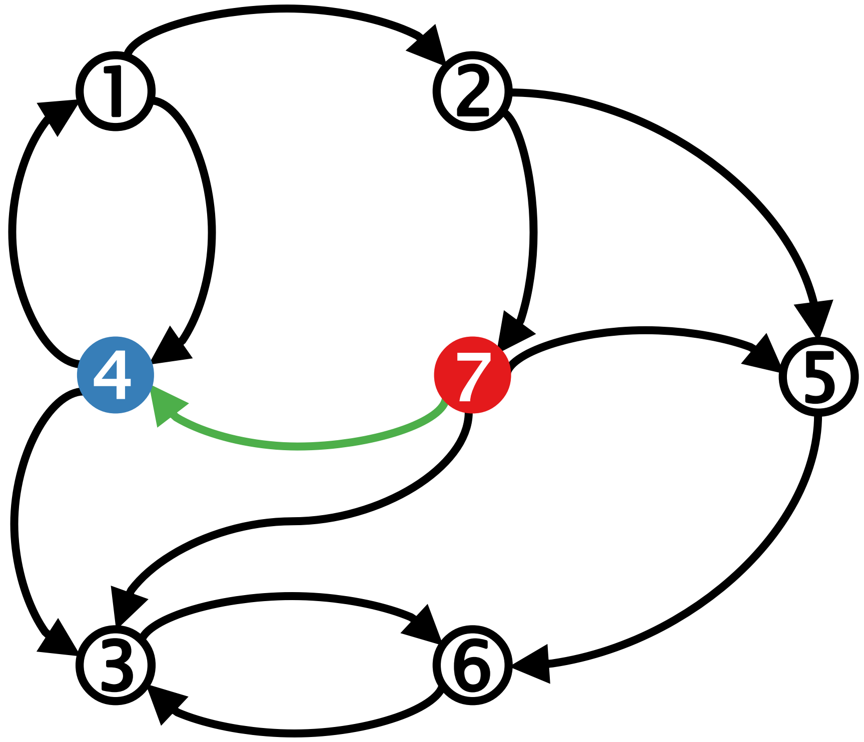
$$\mathbf{A}_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E \\ 0 & \text{if } (v_i, v_j) \notin E \end{cases}$$



A	①	②	③	④	⑤	⑥	⑦
①	0	1	0	1	0	0	0
②	0	0	0	0	1	0	1
③	0	0	0	0	0	1	0
④	1	0	1	0	0	0	0
⑤	0	0	0	0	0	1	0
⑥	0	0	1	0	0	0	0
⑦	0	0	1	1	1	0	0

ADJACENCY MATRIX

$$\mathbf{A}_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E \\ 0 & \text{if } (v_i, v_j) \notin E \end{cases}$$



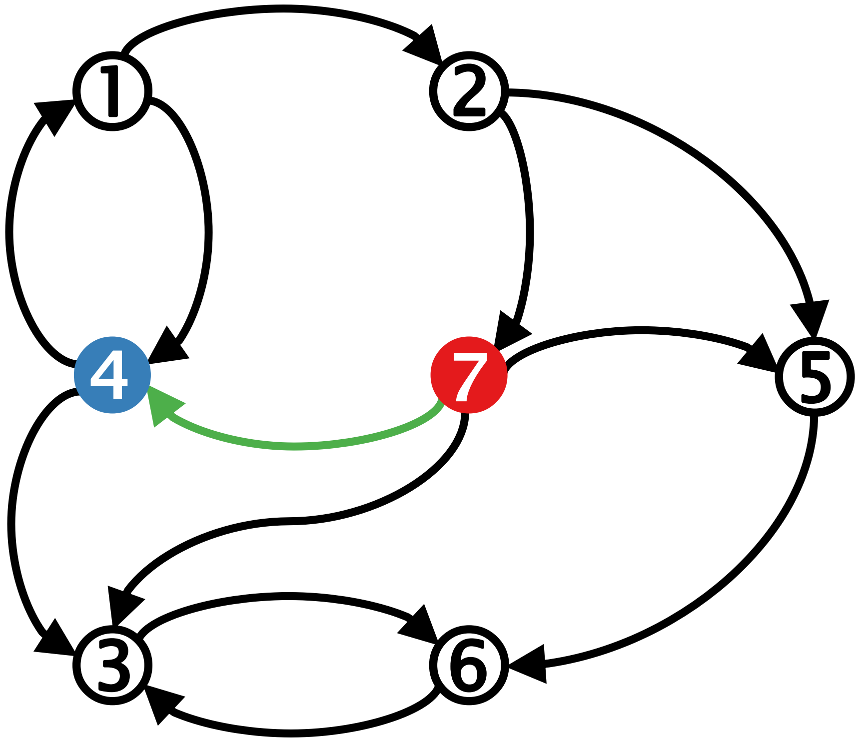
Most cells are zero:
sparse matrix

target

A	①	②	③	④	⑤	⑥	⑦
①	0	1	0	1	0	0	0
②	0	0	0	0	1	0	1
③	0	0	0	0	0	1	0
④	1	0	1	0	0	0	0
⑤	0	0	0	0	0	1	0
⑥	0	0	1	0	0	0	0
source ⑦	0	0	1	1	1	0	0

ADJACENCY MATRIX

$$\mathbf{A}_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E \\ 0 & \text{if } (v_i, v_j) \notin E \end{cases}$$



Most cells are zero:
sparse matrix

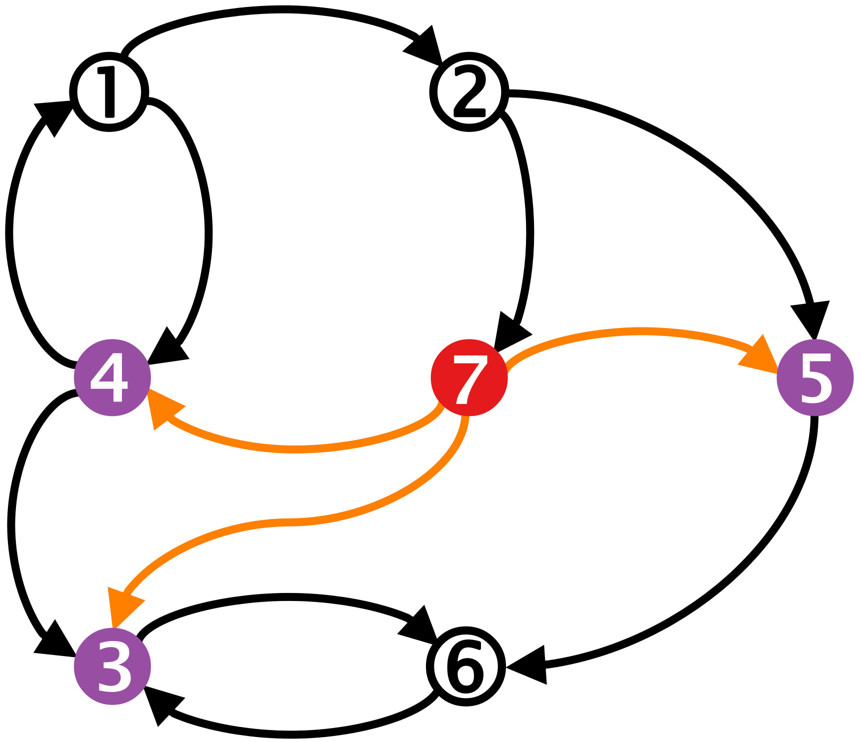
target

	A	1	2	3	4	5	6	7
	1		1		1			
	2					1		1
	3						1	
	4	1		1				
	5						1	
	6			1				
source	7			1	1	1		

GRAPH TRAVERSAL WITH MATRIX MULTIPLICATION

Use **vector/matrix operations** to express graph algorithms:

$\mathbf{vA}^k$ means k hops in the graph



$\mathbf{A}$

①		1		1			
②					1		1
③						1	
④	1		1				
⑤						1	
⑥			1				
⑦			1	1	1		

	①	②	③	④	⑤	⑥	⑦
$\mathbf{v}$							1

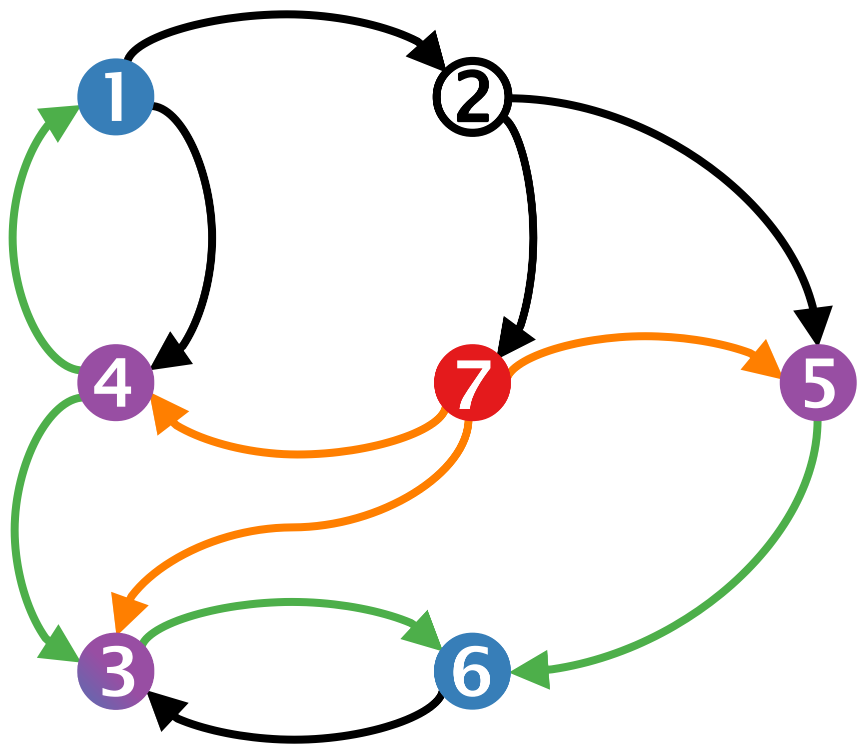
		1	1	1		
①	②	③	④	⑤	⑥	⑦

one-hop: $\mathbf{vA}$

GRAPH TRAVERSAL WITH MATRIX MULTIPLICATION

Use **vector/matrix operations** to express graph algorithms:

$\mathbf{vA}^k$ means k hops in the graph



$\mathbf{A}$

	①	②	③	④	⑤	⑥	⑦
①		1		1			
②					1		1
③						1	
④	1		1				
⑤						1	
⑥			1				
⑦			1	1	1		

	①	②	③	④	⑤	⑥	⑦
$\mathbf{v}$							1

①	②	③	④	⑤	⑥	⑦
		1	1	1		

one-hop: $\mathbf{vA}$

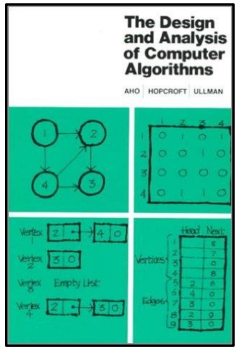
$\mathbf{A}$

	①	②	③	④	⑤	⑥	⑦
①		1		1			
②					1		1
③						1	
④	1		1				
⑤						1	
⑥			1				
⑦			1	1	1		

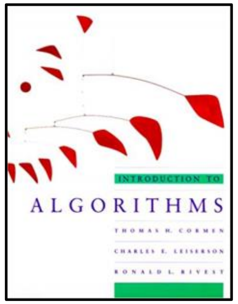
①	②	③	④	⑤	⑥	⑦
1		1			2	

two-hop: $\mathbf{vA}^2$

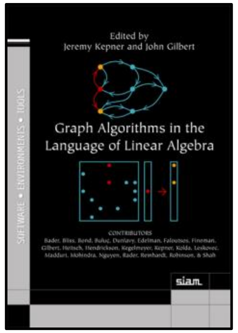
BOOKS ON LINEAR ALGEBRA FOR GRAPH PROCESSING



- 1974: Aho-Hopcroft-Ullman book
  - *The Design and Analysis of Computer Algorithms*



- 1990: Cormen-Leiserson-Rivest book
  - *Introduction to Algorithms*



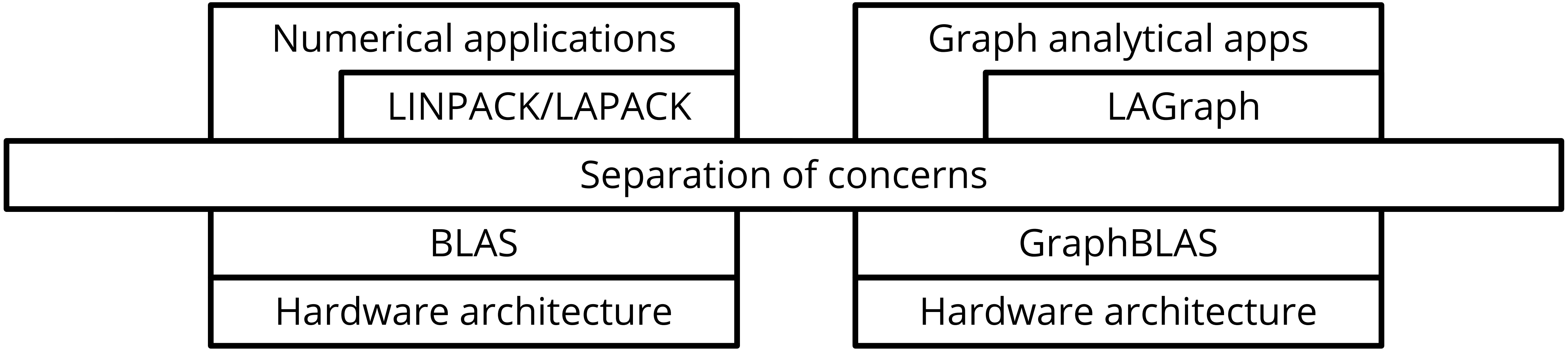
- 2011: GALLA book (ed. Kepner and Gilbert)
  - *Graph Algorithms in the Language of Linear Algebra*

A lot of literature but few practical implementations and particularly few easy-to-use libraries.

THE GRAPHBLAS STANDARD

Goal: separate the concerns of the hardware/library/application designers.

- 1979: BLAS — Basic Linear Algebra Subprograms (dense)
- 2001: Sparse BLAS — an extension to BLAS (insufficient for graphs, little uptake)
- 2013: GraphBLAS — standard building blocks for graph algorithms in LA



Semiring-based graph computations

MATRIX MULTIPLICATION

Definition:

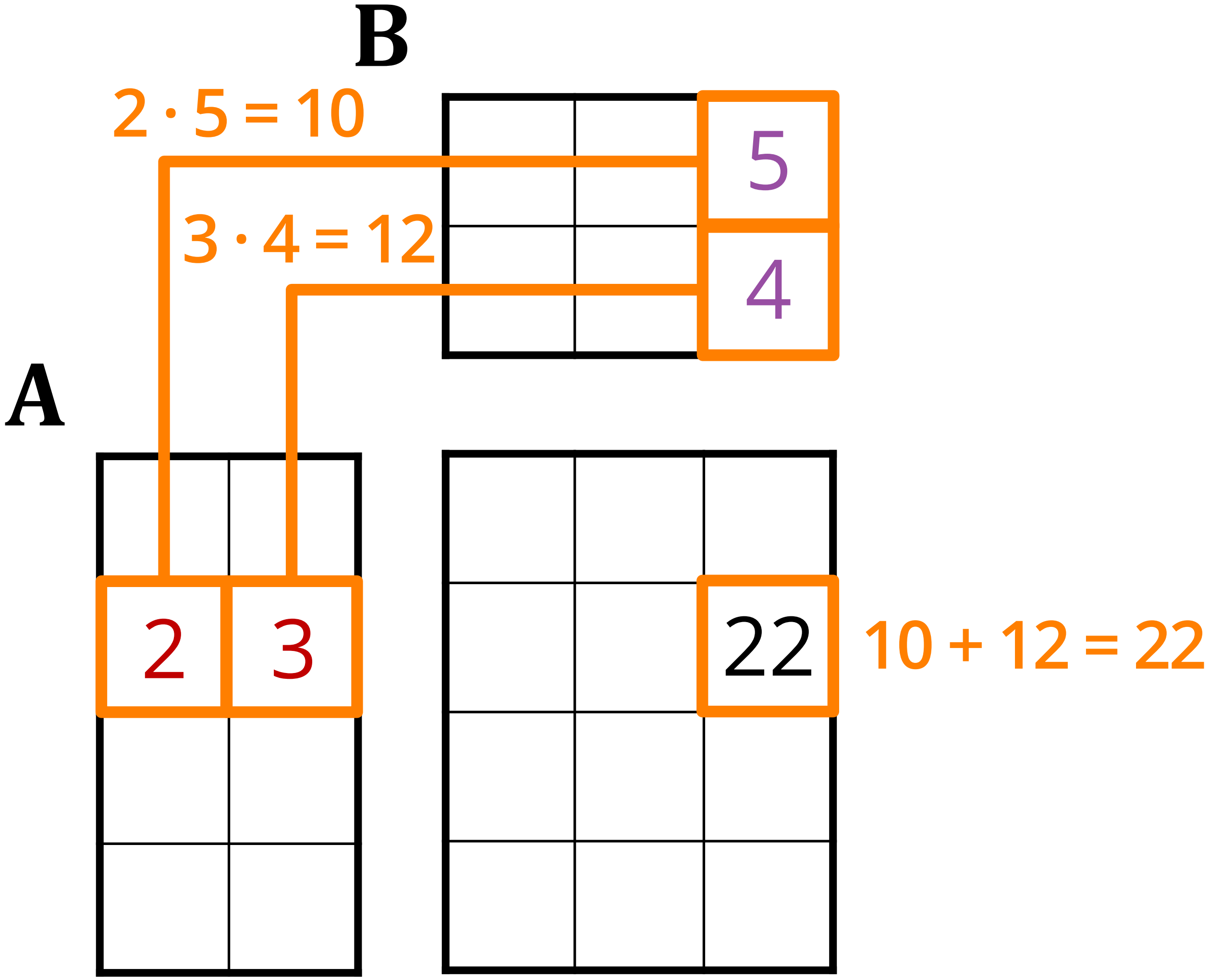
$$\mathbf{C} = \mathbf{A}\mathbf{B}$$

$$\mathbf{C}(i,j) = \sum_k \mathbf{A}(i,k) \cdot \mathbf{B}(k,j)$$

Example:

$$\mathbf{C}(2{,}3) = \mathbf{A}(2{,}1) \cdot \mathbf{B}(1{,}3) + \mathbf{A}(2{,}2) \cdot \mathbf{B}(2{,}3)$$

$$= 2 \cdot 5 + 3 \cdot 4 = 22$$



$$\mathbf{C} = \mathbf{A} \cdot \mathbf{B}$$

MATRIX MULTIPLICATION ON SEMIRINGS

- Using the conventional semiring

$$\mathbf{C} = \mathbf{A}\mathbf{B}$$
$$\mathbf{C}(i,j) = \sum_k \mathbf{A}(i,k) \cdot \mathbf{B}(k,j)$$

- Use arbitrary semirings that override the $\oplus$ addition and $\otimes$ multiplication operators. Generalized formula (simplified)

$$\mathbf{C} = \mathbf{A} \oplus.\otimes\ \mathbf{B}$$
$$\mathbf{C}(i,j) = \bigoplus_k \mathbf{A}(i,k) \otimes \mathbf{B}(k,j)$$

GRAPHBLAS SEMIRINGS

The $\langle D, \oplus, \otimes, 0 \rangle$ algebraic structure is a GraphBLAS semiring if

- $\langle D, \oplus, 0 \rangle$ is a commutative monoid over domain D with an addition operator $\oplus$ and identity 0, where $\forall a, b, c \in D$:
  - Commutative $a \oplus b = b \oplus a$
  - Associative $(a \oplus b) \oplus c = a \oplus (b \oplus c)$
  - Identity $a \oplus 0 = a$
- The multiplication operator is a closed binary operator $\otimes: D \times D \rightarrow D$.

This is less strict than the standard mathematical definition which requires that $\otimes$ is a monoid and distributes over $\oplus$.

COMMON SEMIRINGS

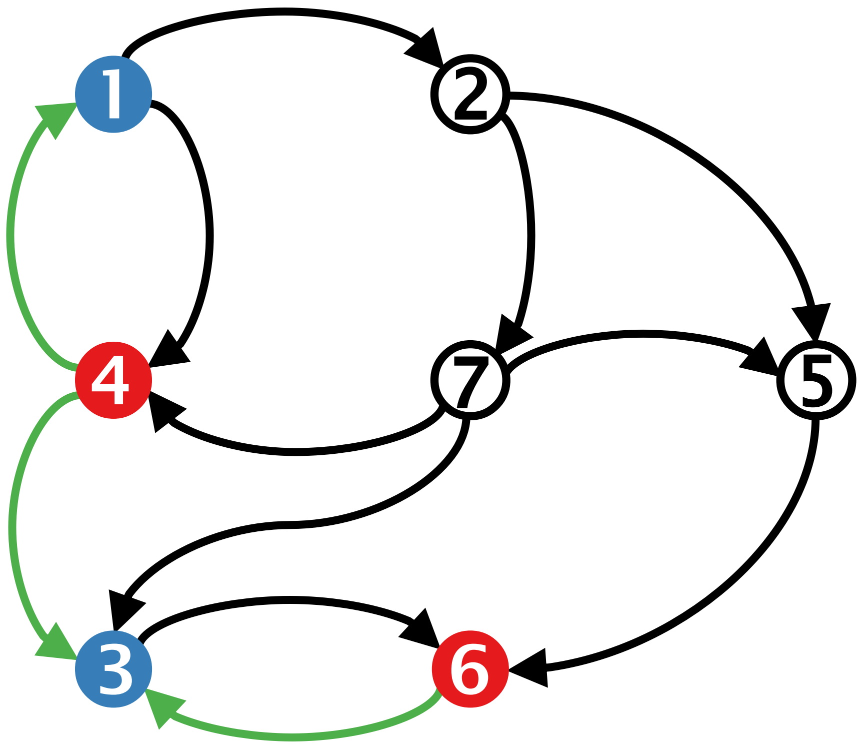
semiring	domain	$\oplus$	$\otimes$	0
integer arithmetic	$a \in \mathbb{N}$	$+$	$\cdot$	0
real arithmetic	$a \in \mathbb{R}$	$+$	$\cdot$	0
lor-land	$a \in \{\mathrm{F}, \mathrm{T}\}$	$\vee$	$\wedge$	F
Galois field	$a \in \{0,1\}$	xor	$\wedge$	0
power set	$a \subset \mathbb{Z}$	$\cup$	$\cap$	$\emptyset$

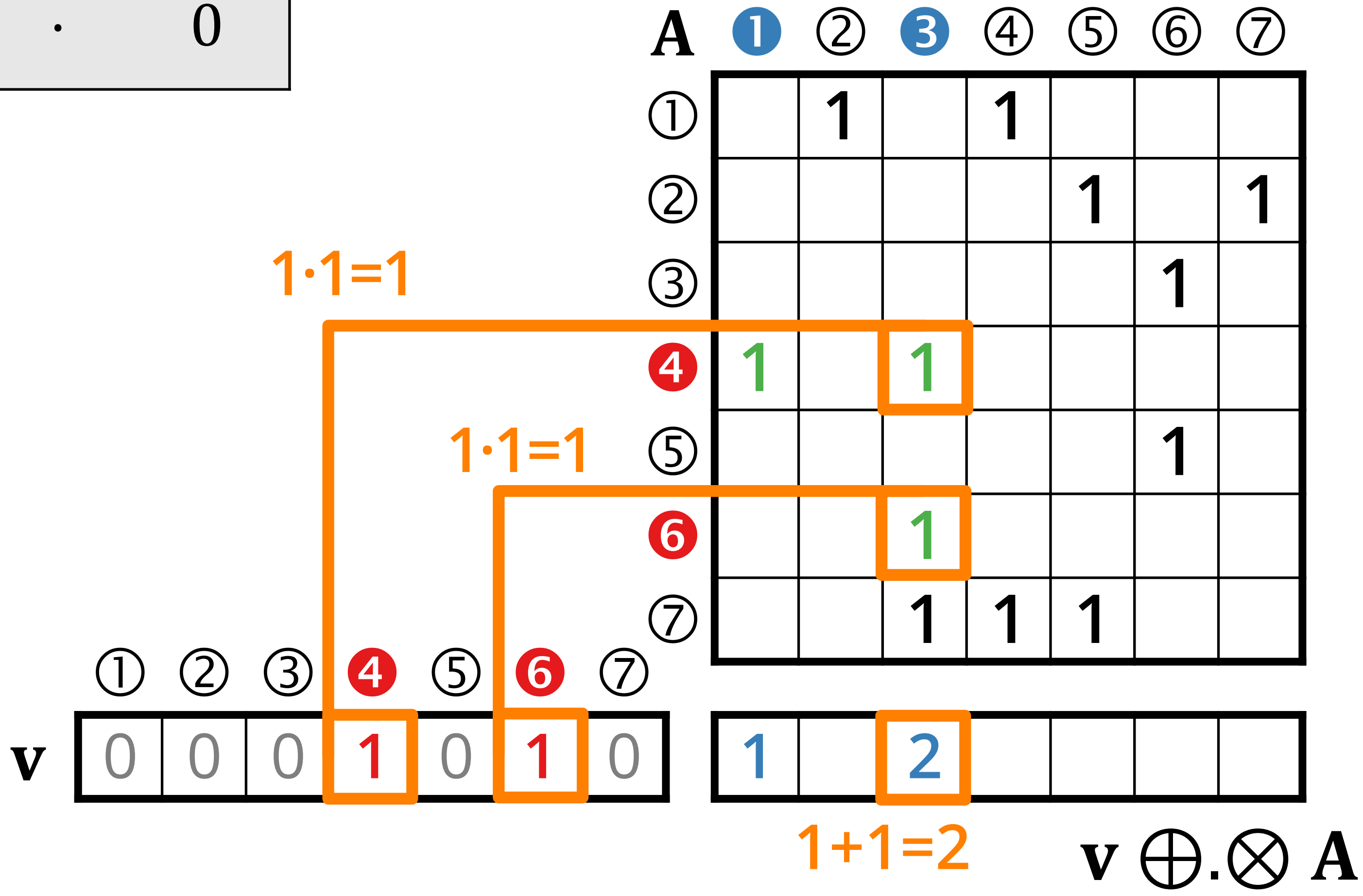
Notation: $\mathbf{A} \oplus.\otimes \mathbf{B}$ is a matrix multiplication using addition $\oplus$ and multiplication $\otimes$, e.g. $\mathbf{A} \vee.\wedge \mathbf{B}$. The default is $\mathbf{A} + . \cdot \mathbf{B}$

MATRIX MULTIPLICATION SEMANTICS

semiring	domain	$\oplus$	$\otimes$	0
integer arithmetic	$a \in \mathbb{N}$	$+$	$\cdot$	0

Semantics: **number of paths**

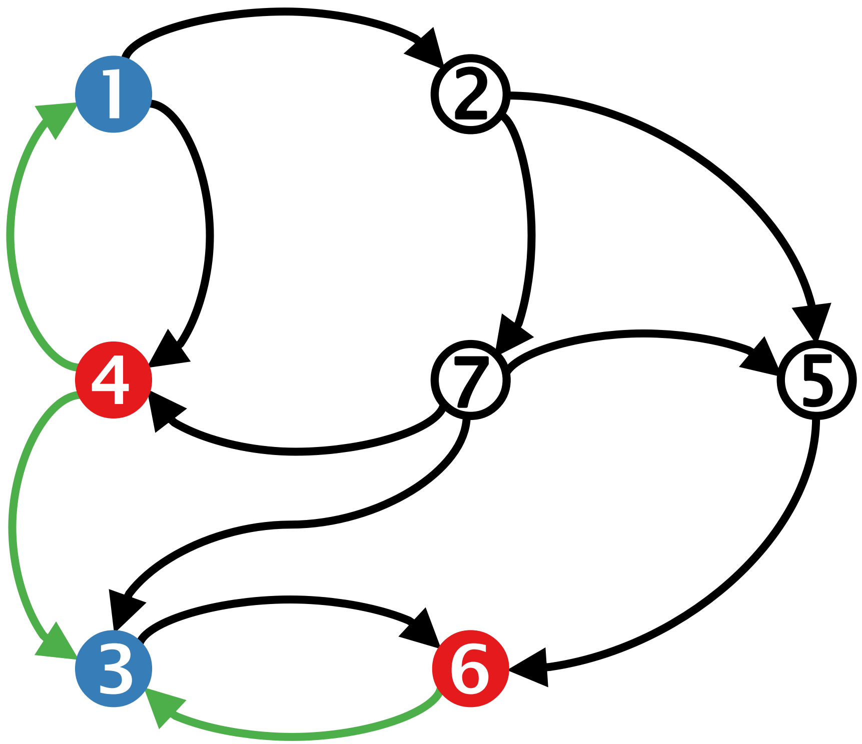


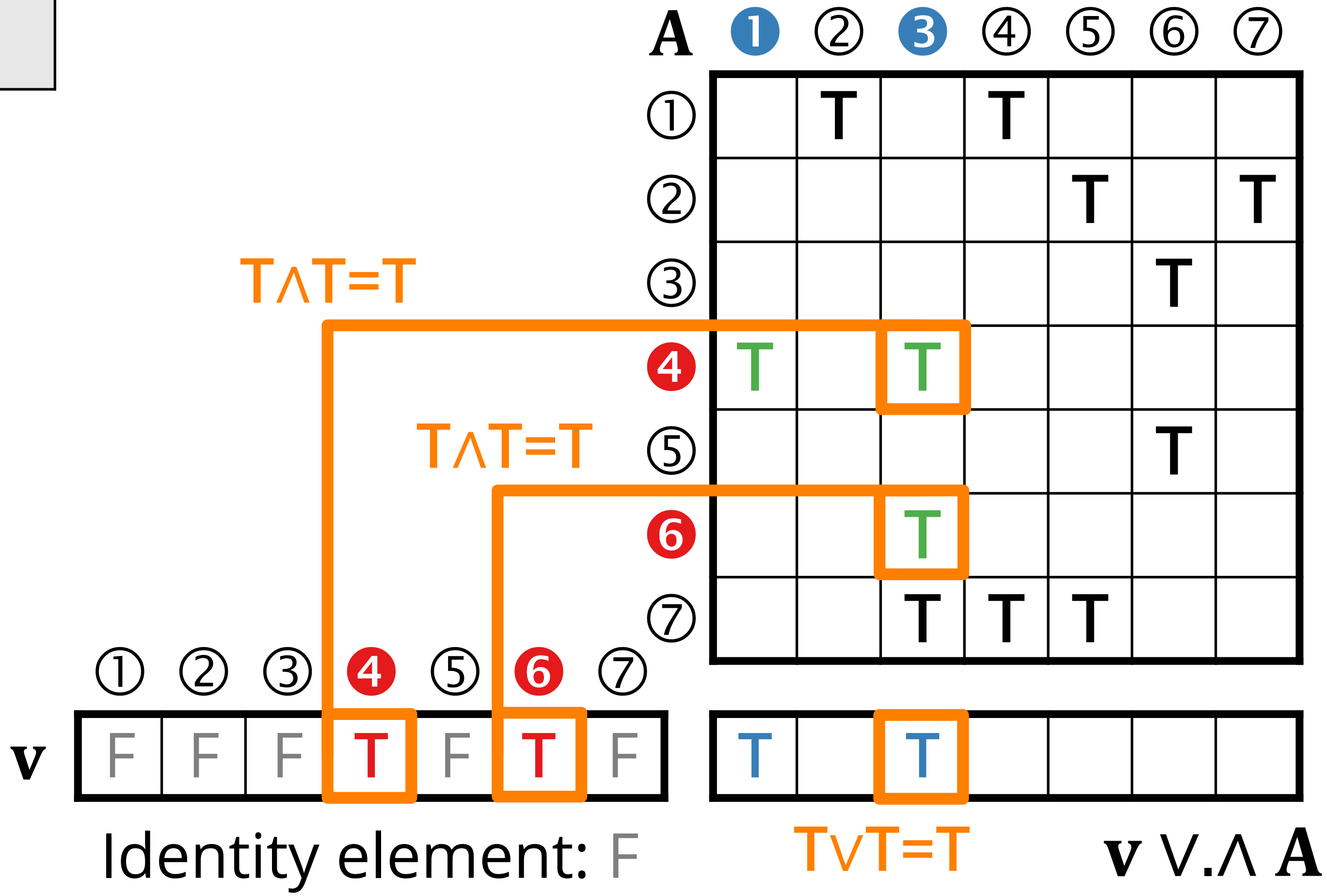


MATRIX MULTIPLICATION SEMANTICS

semiring	domain	$\oplus$	$\otimes$	0
lor-land	$a \in \{F, T\}$	$\vee$	$\wedge$	F

Semantics: **reachability**

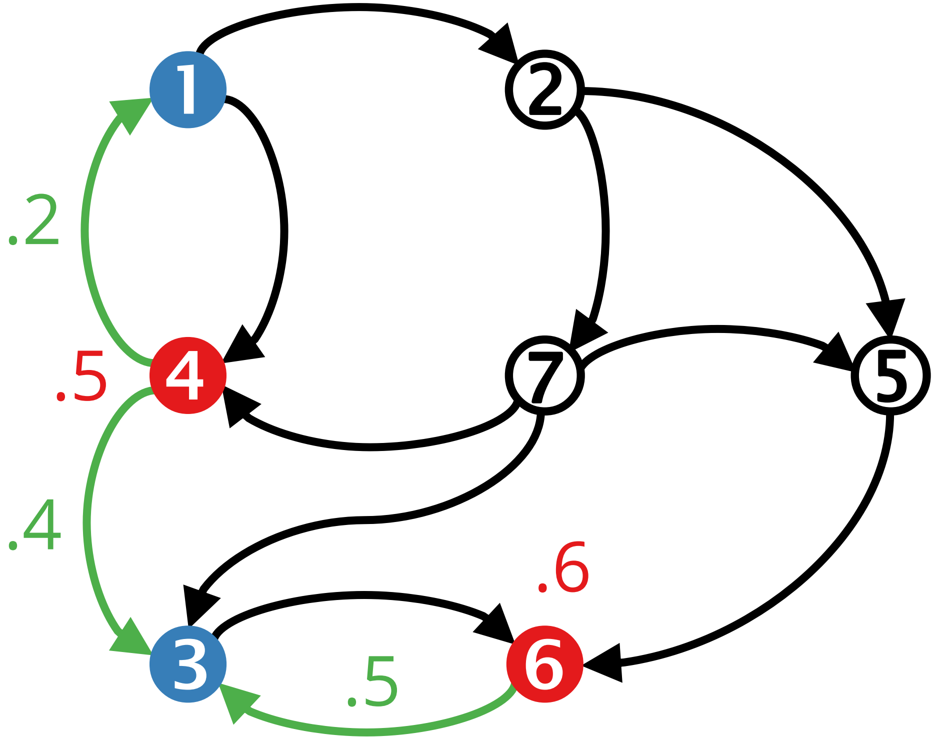


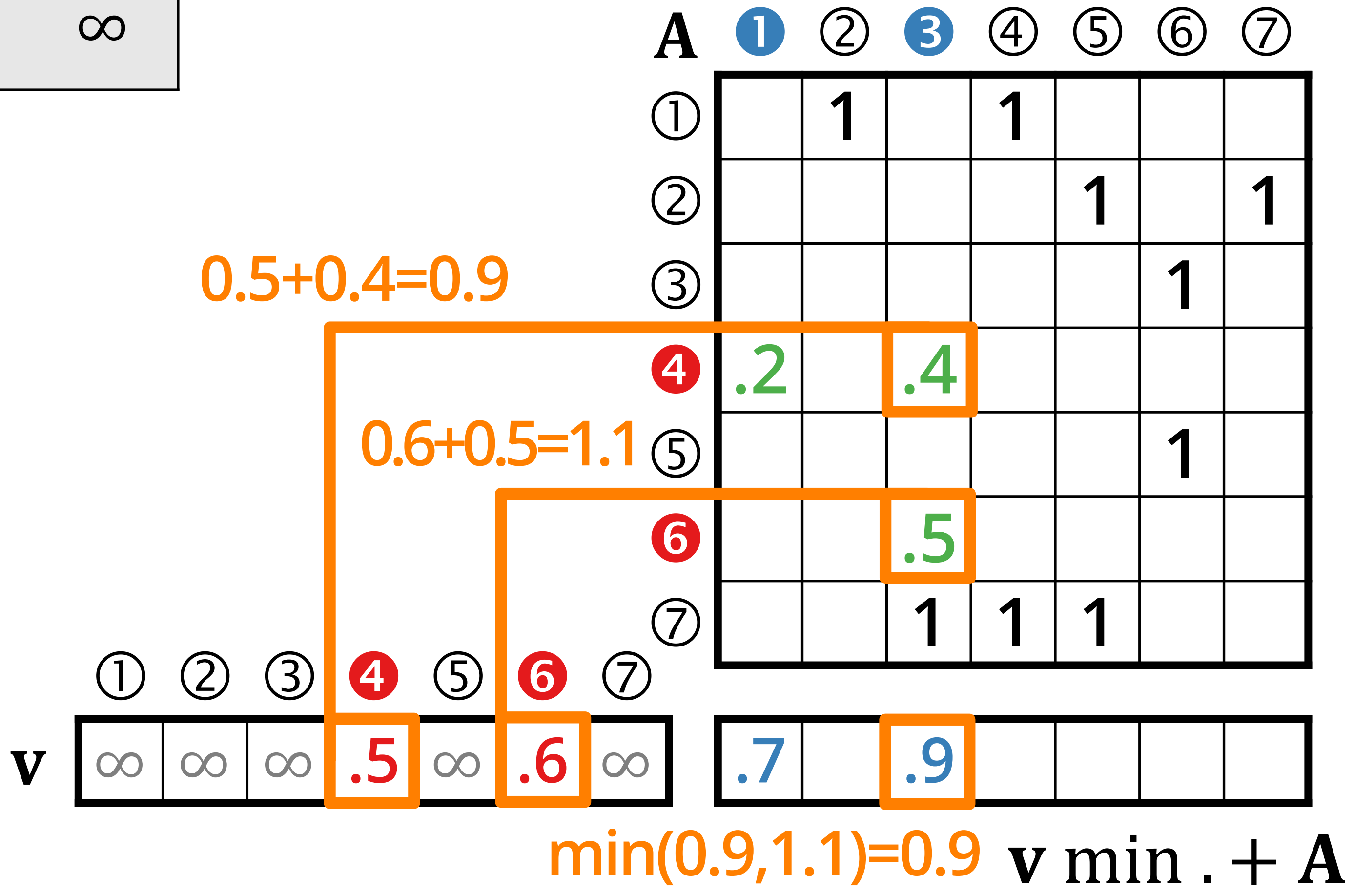


MATRIX MULTIPLICATION SEMANTICS

semiring	domain	⊕	⊗	0
min-plus	$a \in \mathbb{R} \cup \{\infty\}$	min	$+$	∞

Semantics: **shortest path**





Graph algorithms in GraphBLAS

Single-source shortest path

SSSP – SINGLE-SOURCE SHORTEST PATHS

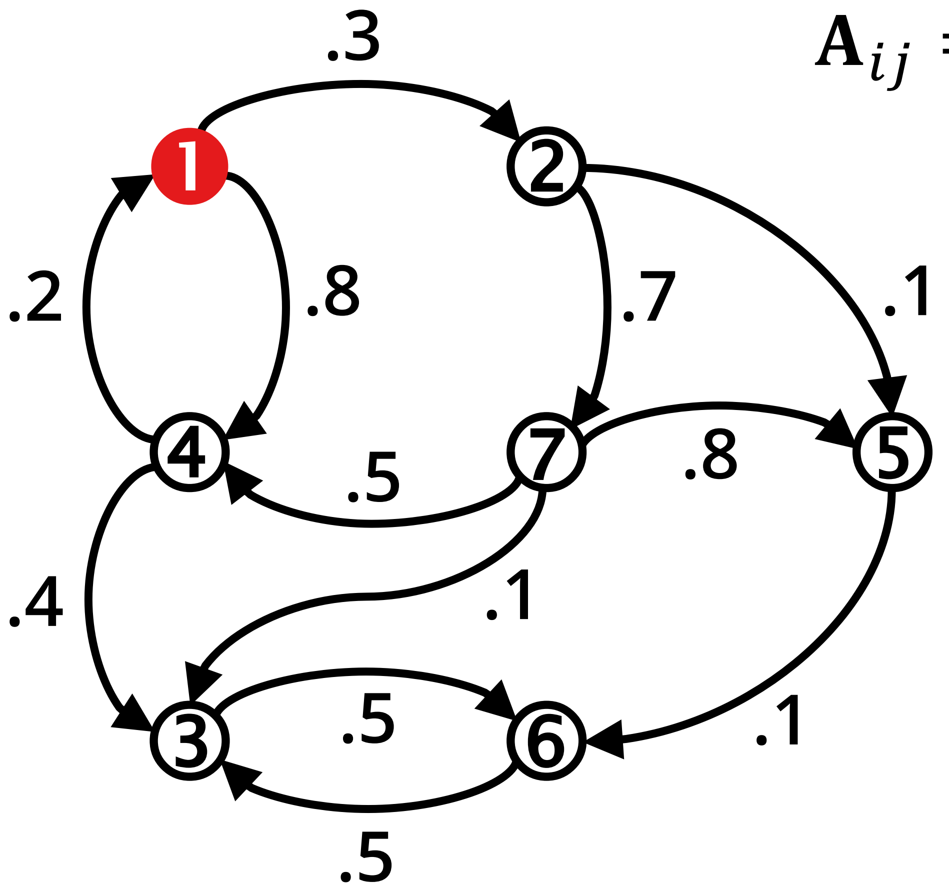
- **Problem:**
  - From a given start node s, find the shortest paths to every other (reachable) node in the graph

- **Bellman-Ford algorithm:**
  - Relaxes all edges in each step
  - Guaranteed to find the shortest paths using at most $n - 1$ steps

- **Observation:**
  - The relaxation step can be captured using a VM multiplication

SSSP – ALGEBRAIC BELLMAN-FORD

We use the **min-plus** semiring with identity ∞.



$$\mathbf{A}_{ij} = \begin{cases} 0 & \text{if } i = j \\ w(e_{ij}) & \text{if } e_{ij} \in E \\ \infty & \text{if } e_{ij} \notin E \end{cases}$$

$$\mathbf{d} = [\infty\ \infty \ldots \infty]$$

$$\mathbf{d}(s) = 0$$

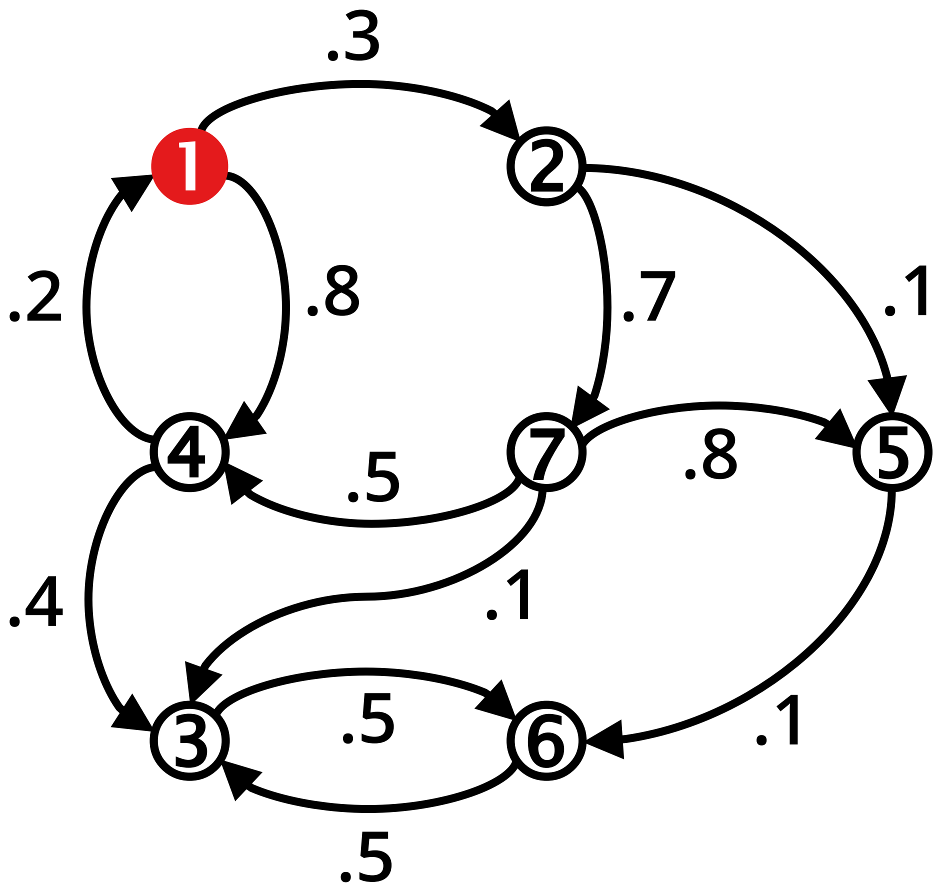
A	①	②	③	④	⑤	⑥	⑦
①	0	.3	∞	.8	∞	∞	∞
②	∞	0	∞	∞	.1	∞	.7
③	∞	∞	0	∞	∞	.5	∞
④	.2	∞	.4	0	∞	∞	∞
⑤	∞	∞	∞	∞	0	.1	∞
⑥	∞	∞	.5	∞	∞	0	∞
⑦	∞	∞	.1	.5	.9	∞	0

	1	②	③	④	⑤	⑥	⑦
d	0	∞	∞	∞	∞	∞	∞

d min.+ **A**

SSSP – ALGEBRAIC BELLMAN-FORD

semiring	set	$\oplus$	$\otimes$	0
min-plus	$a \in \mathbb{R} \cup \{\infty\}$	min	+	∞



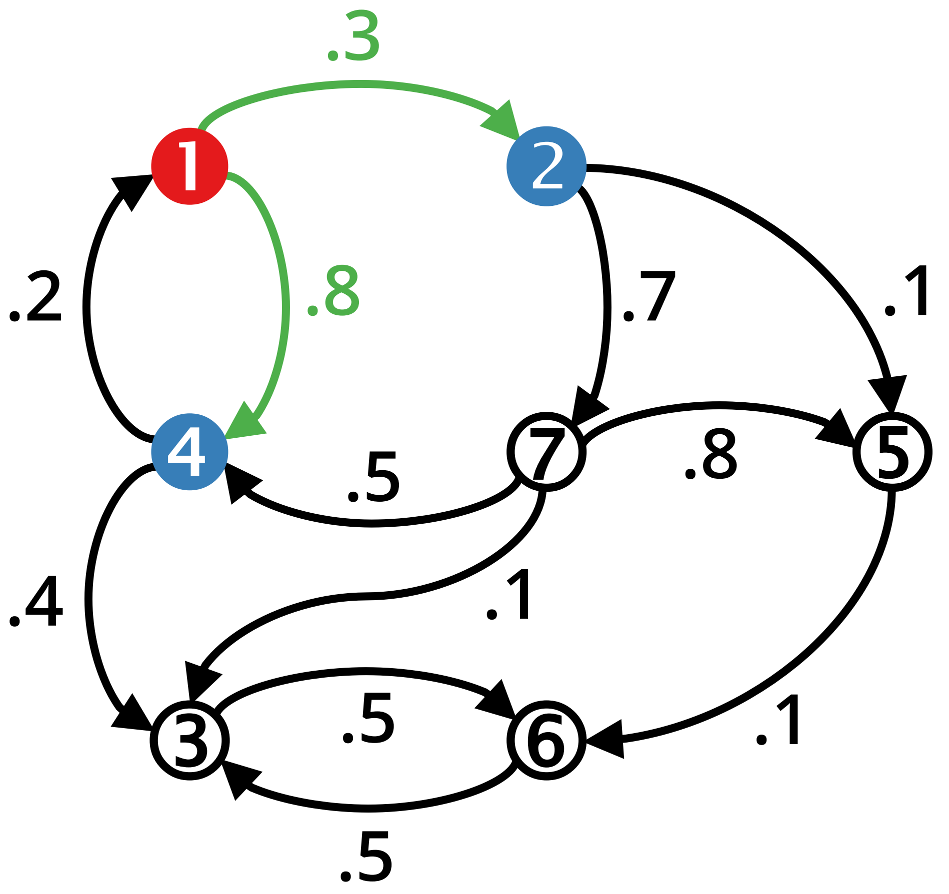
A	①	②	③	④	⑤	⑥	⑦
①	0	.3		.8			
②		0			.1		.7
③			0			.5	
④	.2		.4	0			
⑤					0	.1	
⑥			.5			0	
⑦			.1	.5	.9		0

	❶	②	③	④	⑤	⑥	⑦
d	0	∞	∞	∞	∞	∞	∞

d min.+ **A**

SSSP – ALGEBRAIC BELLMAN-FORD

semiring	set	$\oplus$	$\otimes$	0
min-plus	$a \in \mathbb{R} \cup \{\infty\}$	min	+	∞



A	①	②	③	④	⑤	⑥	⑦
①	0	.3		.8			
②		0			.1		.7
③			0			.5	
④	.2		.4	0			
⑤					0	.1	
⑥			.5			0	
⑦			.1	.5	.9		0

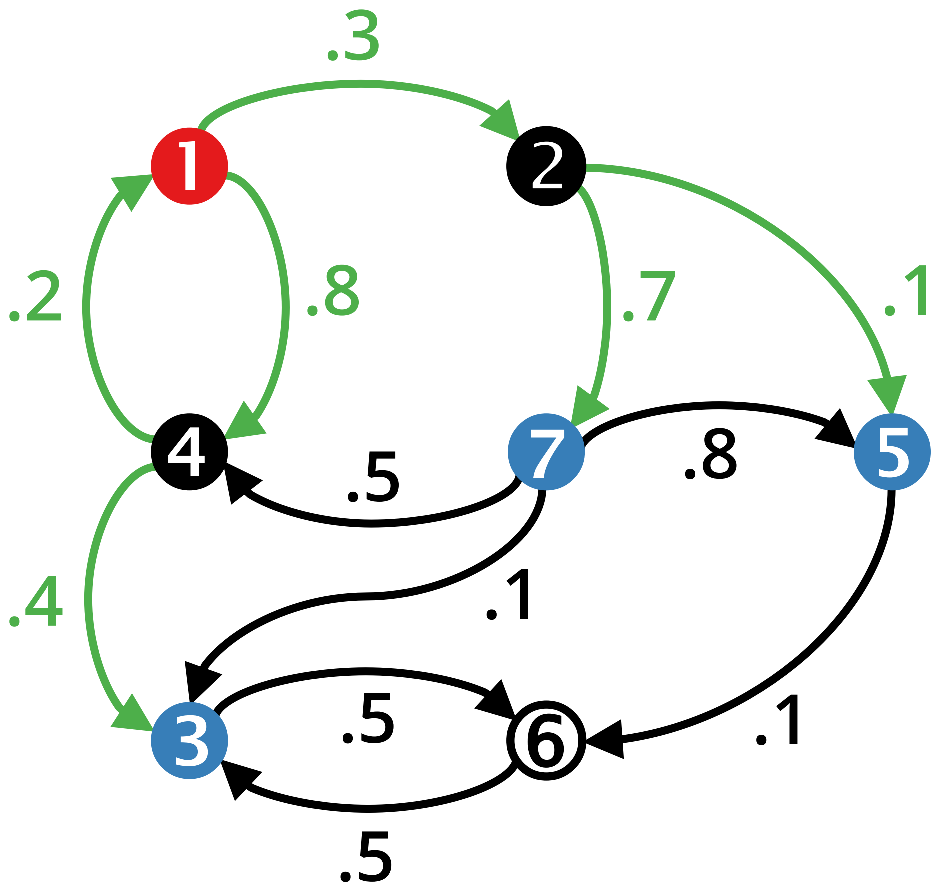
	❶	②	③	④	⑤	⑥	⑦
d	0	∞	∞	∞	∞	∞	∞

0	.3		.8			

d min.+ **A**

SSSP – ALGEBRAIC BELLMAN-FORD

semiring	set	⊕	⊗	0
min-plus	$a \in \mathbb{R} \cup \{\infty\}$	min	+	∞



A	①	②	③	④	⑤	⑥	⑦
①	0	.3		.8			
②		0			.1		.7
③			0			.5	
④	.2		.4	0			
⑤					0	.1	
⑥			.5			0	
⑦			.1	.5	.9		0

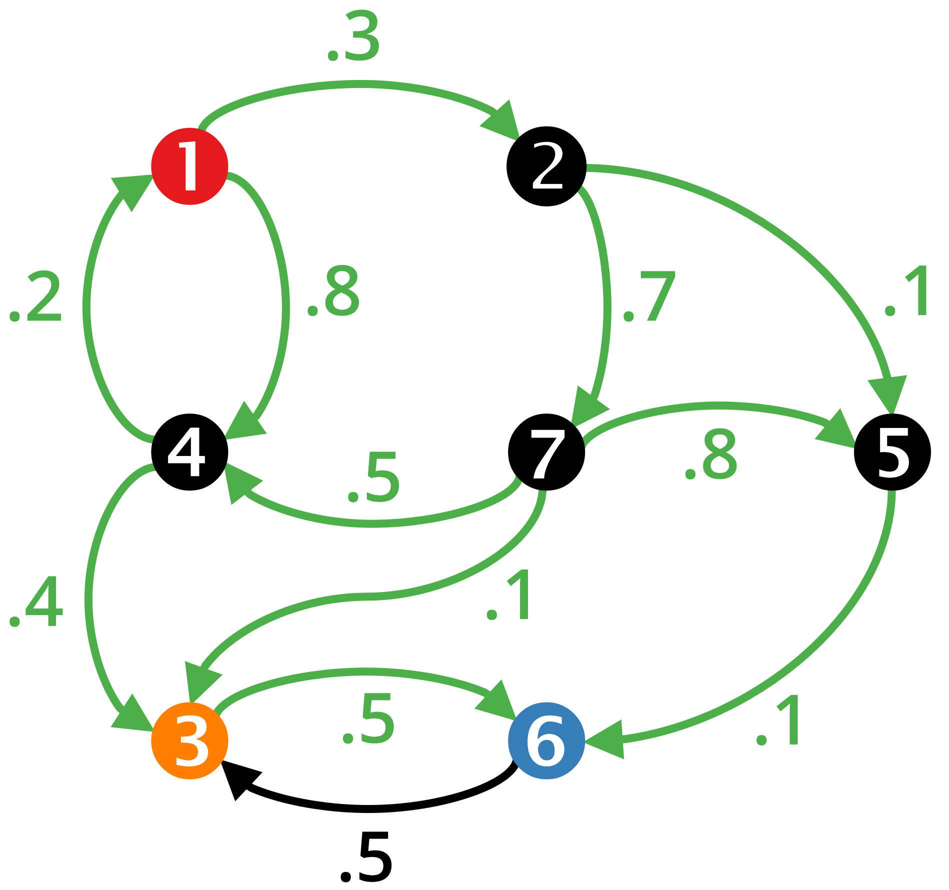
	❶	❷	③	❹	⑤	⑥	⑦
d	0	.3	∞	.8	∞	∞	∞

0	.3	1.2	.8	.4		1

d min.+ **A**

SSSP – ALGEBRAIC BELLMAN-FORD

semiring	set	$\oplus$	$\otimes$	0
min-plus	$a \in \mathbb{R} \cup \{\infty\}$	min	$+$	∞



A	①	②	③	④	⑤	⑥	⑦
①	0	.3		.8			
②		0			.1		.7
③			0			.5	
④	.2		.4	0			
⑤					0	.1	
⑥			.5			0	
⑦			.1	.5	.9		0

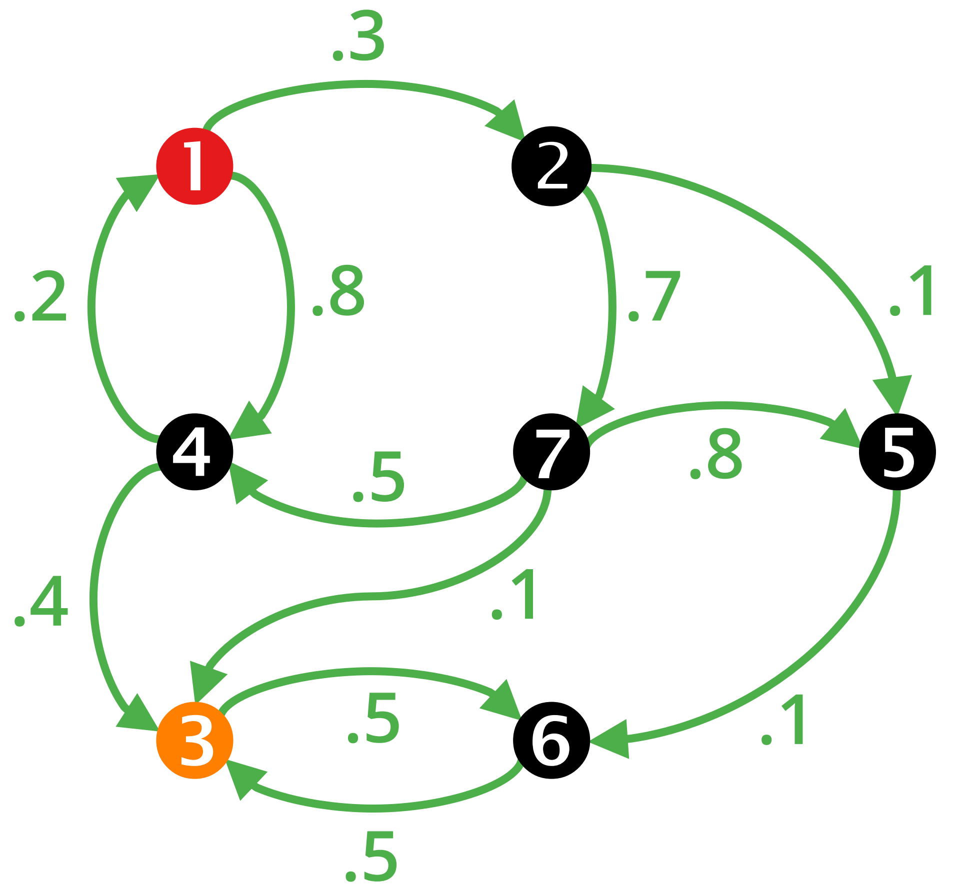
	1	2	3	4	5	6	7
d	0	.3	1.2	.8	.4	∞	1

0	.3	1.1	.8	.4	.5	1

d min.+ **A**

SSSP – ALGEBRAIC BELLMAN-FORD

semiring	set	⊕	⊗	0
min-plus	$a \in \mathbb{R} \cup \{\infty\}$	min	+	∞



A	①	②	③	④	⑤	⑥	⑦
①	0	.3		.8			
②		0			.1		.7
③			0			.5	
④	.2		.4	0			
⑤					0	.1	
⑥			.5			0	
⑦			.1	.5	.9		0

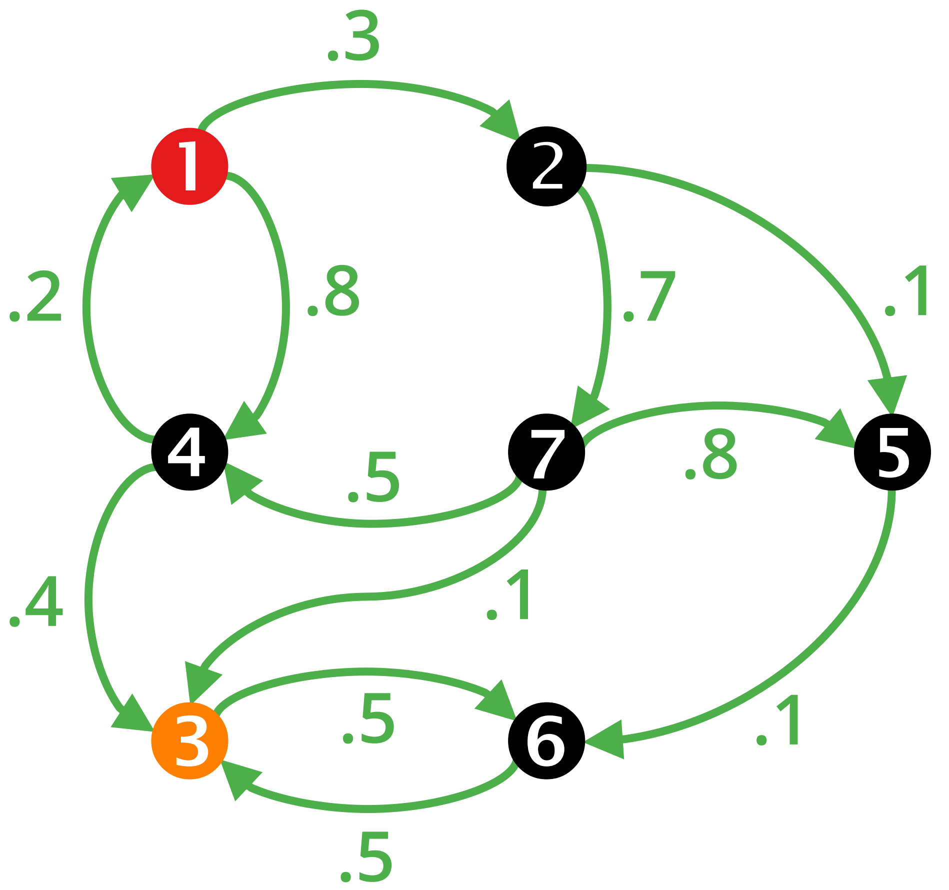
	1	2	3	4	5	6	7
d	0	.3	1.1	.8	.4	.5	1

0	.3	1	.8	.4	.5	1

d min.+ **A**

SSSP – ALGEBRAIC BELLMAN-FORD

semiring	set	$\oplus$	$\otimes$	0
min-plus	$a \in \mathbb{R} \cup \{\infty\}$	min	$+$	∞



A

	①	②	③	④	⑤	⑥	⑦
①	0	.3		.8			
②		0			.1		.7
③			0			.5	
④	.2		.4	0			
⑤					0	.1	
⑥			.5			0	
⑦			.1	.5	.9		0

	1	2	3	4	5	6	7
d	0	.3	1	.8	.4	.5	1

0	.3	1	.8	.4	.5	1

d min.+ **A**

SSSP – ALGEBRAIC BELLMAN-FORD ALGO.

Input: adjacency matrix $\mathbf{A}$, source node s, #nodes n

$$\mathbf{A}_{ij} = \begin{cases} 0 & \text{if } i = j \\ w(e_{ij}) & \text{if } e_{ij} \in E \\ \infty & \text{if } e_{ij} \notin E \end{cases}$$

Output: distance vector $\mathbf{d} \in (\mathbb{R} \cup \{\infty\})^n$

1. $\mathbf{d} = [\infty\ \infty \dots \infty]$
2. $\mathbf{d}(s) = 0$
3. $\text{for } k = 1 \text{ to } n - 1$ *terminate earlier if we reach a fixed point
4. $\quad \mathbf{d} = \mathbf{d} \text{ min.+ } \mathbf{A}$

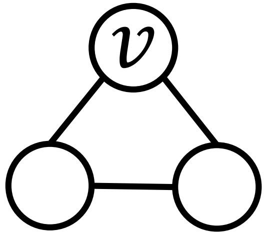
Optimization: switch between $\mathbf{d} \text{ min.+ } \mathbf{A}$ and $\mathbf{A}^\top \text{ min.+ } \mathbf{d}$ (push/pull).

Graph algorithms in GraphBLAS

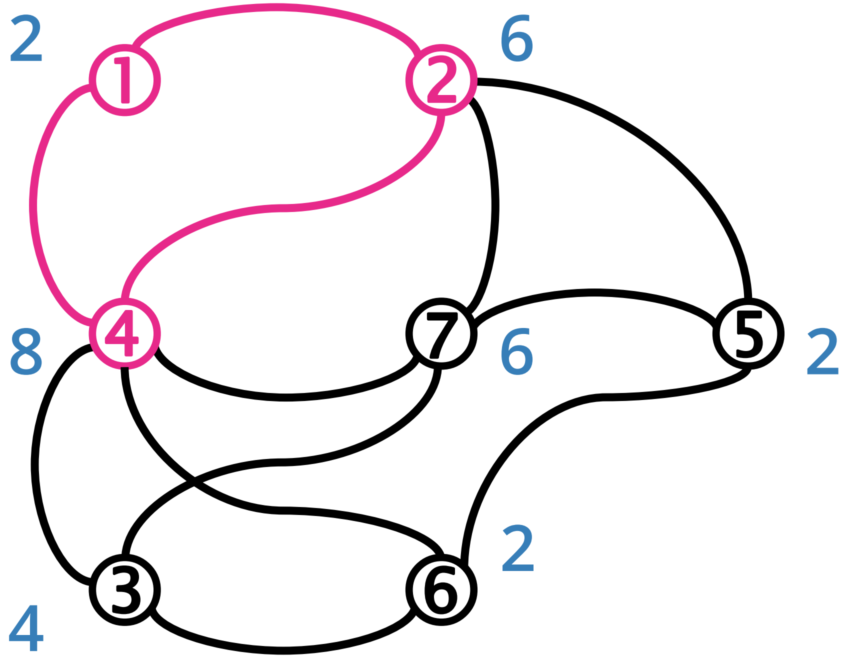
Node-wise triangle count

NODE-WISE TRIANGLE COUNT

Triangle – Def 1: a set of three mutually adjacent nodes.



Def 2: a three-length closed path.



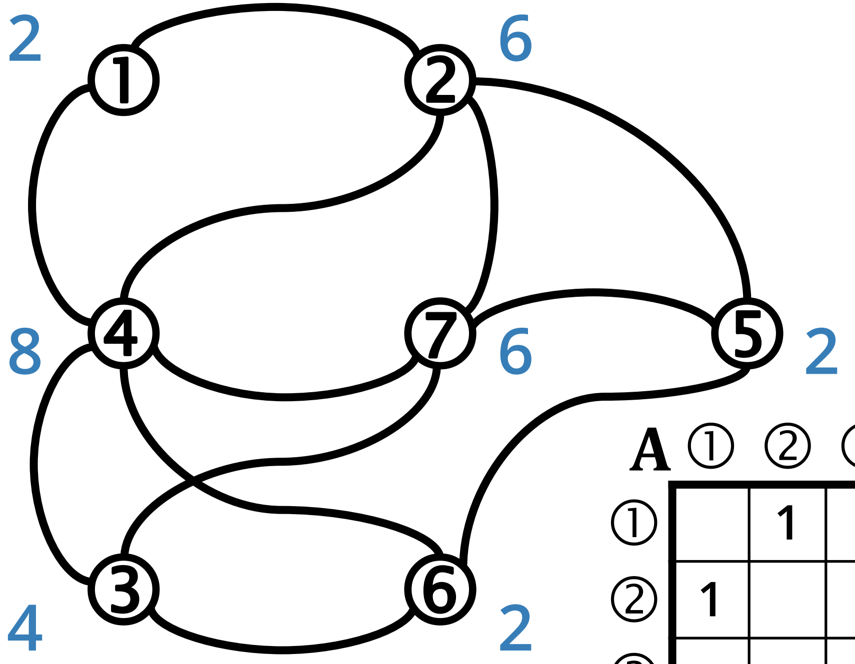
Usages:

- Global clustering coefficient
- Local clustering coefficient
- Finding communities

GraphChallenge.org: Raising the Bar on Graph Analytic Performance, HPEC 2018

TC: NAÏVE APPROACH

$$\mathbf{tri} = \mathrm{diag}^{-1}(\mathbf{A} \oplus.\otimes \mathbf{A} \oplus.\otimes \mathbf{A})$$



A	①	②	③	④	⑤	⑥	⑦
①		1		1			
②	1			1	1		1
③				1		1	1
④	1	1	1			1	1
⑤		1				1	1
⑥			1	1	1		
⑦		1	1	1	1		

A	①	②	③	④	⑤	⑥	⑦
①		1		1			
②	1			1	1		1
③				1		1	1
④	1	1	1			1	1
⑤		1				1	1
⑥			1	1	1		
⑦		1	1	1	1		

A	①	②	③	④	⑤	⑥	⑦
①		1		1			
②	1			1	1		1
③				1		1	1
④	1	1	1			1	1
⑤		1				1	1
⑥			1	1	1		
⑦		1	1	1	1		

2	1	1	1	1	1	2
1	4	2	2	1	2	2
1	2	3	2	2	1	1
1	2	2	5	3	1	2
1	1	2	3	3		1
1	2	1	1		3	3
2	2	1	2	1	3	4

2	6	4	7	4	3	4
6	6	6	11	8	5	9
4	6	4	8	4	7	9
7	11	8	8	5	10	12
4	8	4	5	2	8	9
3	5	7	10	8	2	4
4	9	9	12	9	4	6

tri
2
6
4
8
2
2
6

TC: OPTIMIZATION

Observation: Matrix $\mathbf{A} \oplus.\otimes \mathbf{A} \oplus.\otimes \mathbf{A}$ is no longer sparse.

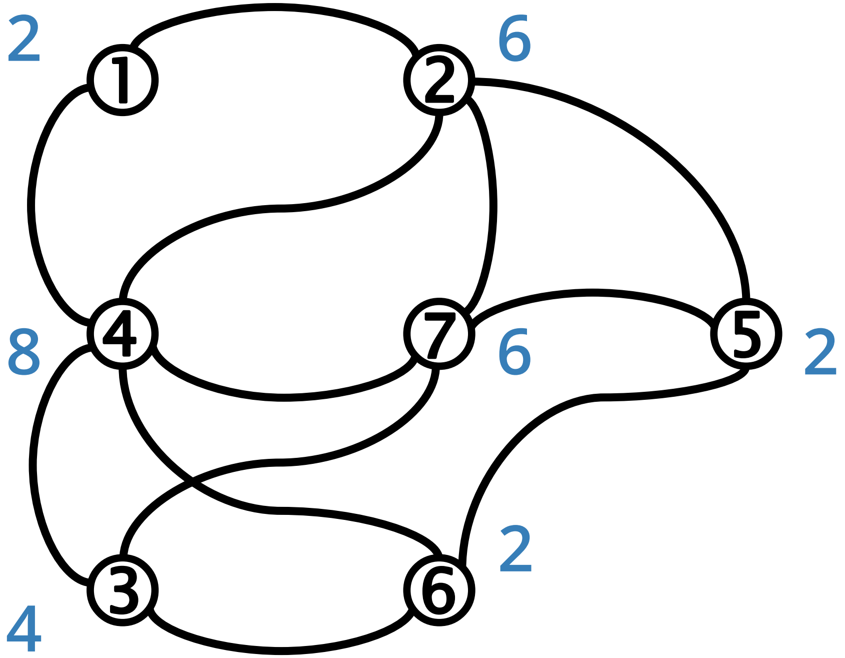
Optimization: Use element-wise multiplication $\otimes$ to close wedges into triangles:

$$\mathbf{TRI} = \mathbf{A} \oplus.\otimes \mathbf{A} \otimes \mathbf{A}$$

Then, perform a row-wise summation to get the number of triangles in each row:

$$\mathbf{tri} = \left[\bigoplus_j \mathbf{TRI}(:,j)\right]$$

TC: ELEMENT-WISE MULTIPLICATION



$$\mathbf{TRI} = \mathbf{A} \oplus.\otimes \mathbf{A} \otimes \mathbf{A}$$

$$\mathbf{tri} = \left[\oplus_j \mathbf{TRI}(:,j)\right]$$

$\mathbf{A} \oplus.\otimes \mathbf{A}$ is still very dense.

A	①	②	③	④	⑤	⑥	⑦
①		1		1			
②	1			1	1		1
③				1		1	1
④	1	1	1			1	1
⑤		1				1	1
⑥			1	1	1		
⑦		1	1	1	1		

A	①	②	③	④	⑤	⑥	⑦
①		1		1			
②	1			1	1		1
③				1		1	1
④	1	1	1			1	1
⑤		1				1	1
⑥			1	1	1		
⑦		1	1	1	1		

2	1	1	1	1	1	2
1	4	2	2	1	2	2
1	2	3	2	2	1	1
1	2	2	5	3	1	2
1	1	2	3	3		1
1	2	1	1		3	3
2	2	1	2	1	3	4

$\xrightarrow{\otimes\ \mathbf{A}}$

TRI

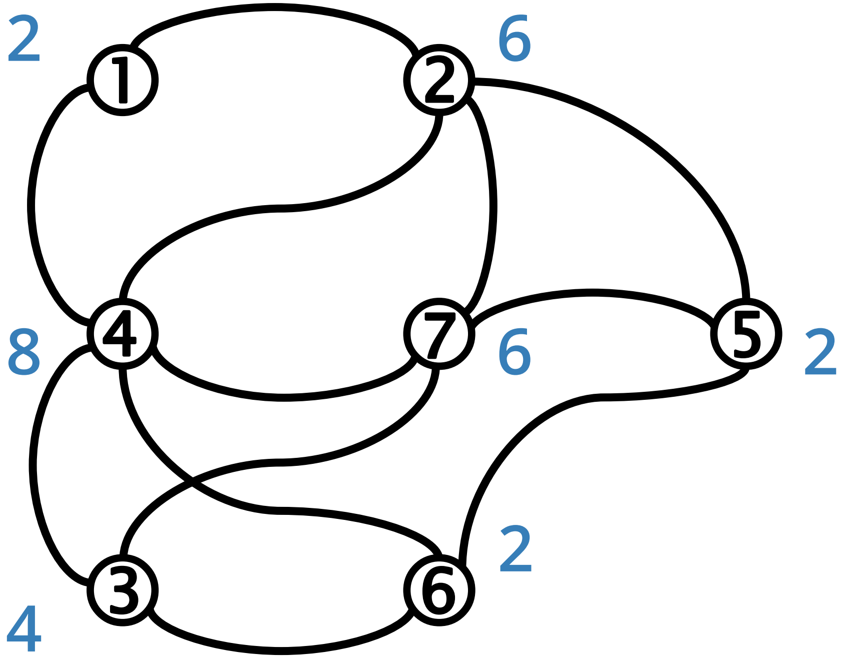
	1		1			
1			2	1		2
			2		1	1
1	2	2			1	2
	1					1
		1	1			
	2	1	2	1		

$\xrightarrow{[\oplus_j \cdots]}$

tri

2
6
4
8
2
2
6

TC: ELEMENT-WISE MULTIPLICATION



Masking limits where the operation is computed. Here, we use $\mathbf{A}$ as a mask for $\mathbf{A} \oplus.\otimes\ \mathbf{A}$.

$$\mathbf{TRI}\langle\mathbf{A}\rangle = \mathbf{A} \oplus.\otimes\ \mathbf{A}$$

$$\mathbf{tri} = \left[\bigoplus_j \mathbf{TRI}(:, j)\right]$$

A	①	②	③	④	⑤	⑥	⑦
①		1		1			
②	1			1	1		1
③				1		1	1
④	1	1	1			1	1
⑤		1				1	1
⑥			1	1	1		
⑦		1	1	1	1		

A	①	②	③	④	⑤	⑥	⑦
①		1		1			
②	1			1	1		1
③				1		1	1
④	1	1	1			1	1
⑤		1				1	1
⑥			1	1	1		
⑦		1	1	1	1		

	1		1			
1			2	1		2
			2		1	1
1	2	2			1	2
	1					1
		1	1			
	2	1	2	1		

$$\xrightarrow{\left[\bigoplus_j \cdots\right]}$$

tri
2
6
4
8
2
2
6

TC: ALGORITHM

Input: adjacency matrix $\mathbf{A}$

Output: vector $\mathbf{tri}$

Workspace: matrix $\mathbf{TRI}$

1. $\mathbf{TRI}\langle\mathbf{A}\rangle = \mathbf{A} \oplus.\otimes \mathbf{A}$ compute the triangle count matrix
2. $\mathbf{tri} = \left[\bigoplus_j \mathbf{TRI}(:, j)\right]$ compute the triangle count vector

Optimization: use $\mathbf{L}$, the lower triangular part of $\mathbf{A}$ to avoid duplicates.

$$\mathbf{TRI}\langle\mathbf{A}\rangle = \mathbf{A} \oplus.\otimes \mathbf{L}$$

Worst-case optimal joins: There are deep theoretical connections between masked matrix multiplication and relational joins. It has been proven in 2013 that for the triangle query, binary joins always provide suboptimal runtime, which gave rise to new research on the family of worst-case optimal multi-way joins algorithms.

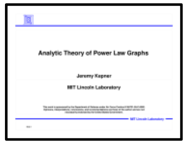
Graph algorithms in GraphBLAS

Other algorithms

GRAPH ALGORITHMS IN GRAPHBLAS

Notation: $n = |V|, m = |E|$. The complexity cells contain asymptotic bounds.
Takeaway: The majority of common graph algorithms can be expressed efficiently in LA.

problem category	algorithm	canonical complexity Θ	LA-based complexity Θ
breadth-first search		m	m
single-source shortest paths	Dijkstra	$m + n \log n$	n^2
	Bellman-Ford	mn	mn
all-pairs shortest paths	Floyd-Warshall	n^3	n^3
minimum spanning tree	Prim	$m + n \log n$	n^2
	Borůvka	$m \log n$	$m \log n$
maximum flow	Edmonds-Karp	$m^2 n$	$m^2 n$
maximal independent set	greedy	$m + n \log n$	$mn + n^2$
	Luby	$m + n \log n$	$m \log n$


Analytic Theory of Power Law Graphs
Jeremy Kepner
MIT Lincoln Laboratory

Based on the table in J. Kepner:
Analytic Theory of Power Law Graphs,
SIAM Workshop for HPC on Large Graphs, 2008

See also L. Dhulipala, G.E. Blelloch, J. Shun:
Theoretically Efficient Parallel Graph Algorithms Can Be Fast and Scalable, SPAA 2018

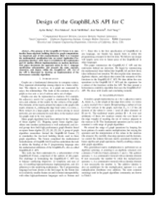
API and implementations

GRAPHBLAS C API

- “A crucial piece of the GraphBLAS effort is to translate the mathematical specification to an API that
  - is faithful to the mathematics as much as possible, and
  - enables efficient implementations on modern hardware.”

$$\mathbf{C}\langle\neg\mathbf{M}\rangle \odot= \oplus.\otimes\,(\mathbf{A}^{\top},\mathbf{B}^{\top})$$

```
mxm(Matrix *C, Matrix M, BinaryOp accum, Semiring op, Matrix A, Matrix B, Descriptor desc)
```

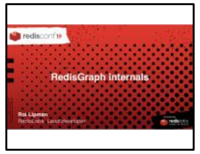


A. Buluç et al.: *Design of the GraphBLAS C API,* GABB@IPDPS 2017

SUITESPARSE:GRAPHBLAS

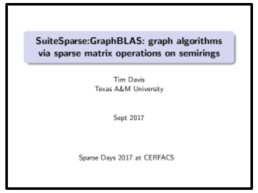
- Authored by Prof. Tim Davis at Texas A&M University, based on his SuiteSparse library (used in MATLAB).
- Additional extension operations for efficiency.
- Sophisticated load balancer for multi-threaded execution.
- CPU-based, single machine implementation.
- Powers the RedisGraph graph database.

R. Lipman, T.A. Davis: *Graph Algebra – Graph operations in the language of linear algebra*, RedisConf 2018



R. Lipman: *RedisGraph internals*, RedisConf 2019

T.A. Davis: *Algorithm 1000: SuiteSparse:GraphBLAS: graph algorithms in the language of sparse linear algebra*, ACM TOMS, 2019



T.A. Davis: *SuiteSparse:GraphBLAS: graph algorithms via sparse matrix operations on semirings*, Sparse Days 2017

PYTHON WRAPPERS

Two libraries, both offer:

- Concise GraphBLAS operations
- Wrapping SuiteSparse:GrB
- Jupyter support

Difference: pygraphblas is more Pythonic, grblas strives to stay close to the C API.

[michelp/pygraphblas](https://github.com/michelp/pygraphblas)

[jim22k/grblas](https://github.com/jim22k/grblas)

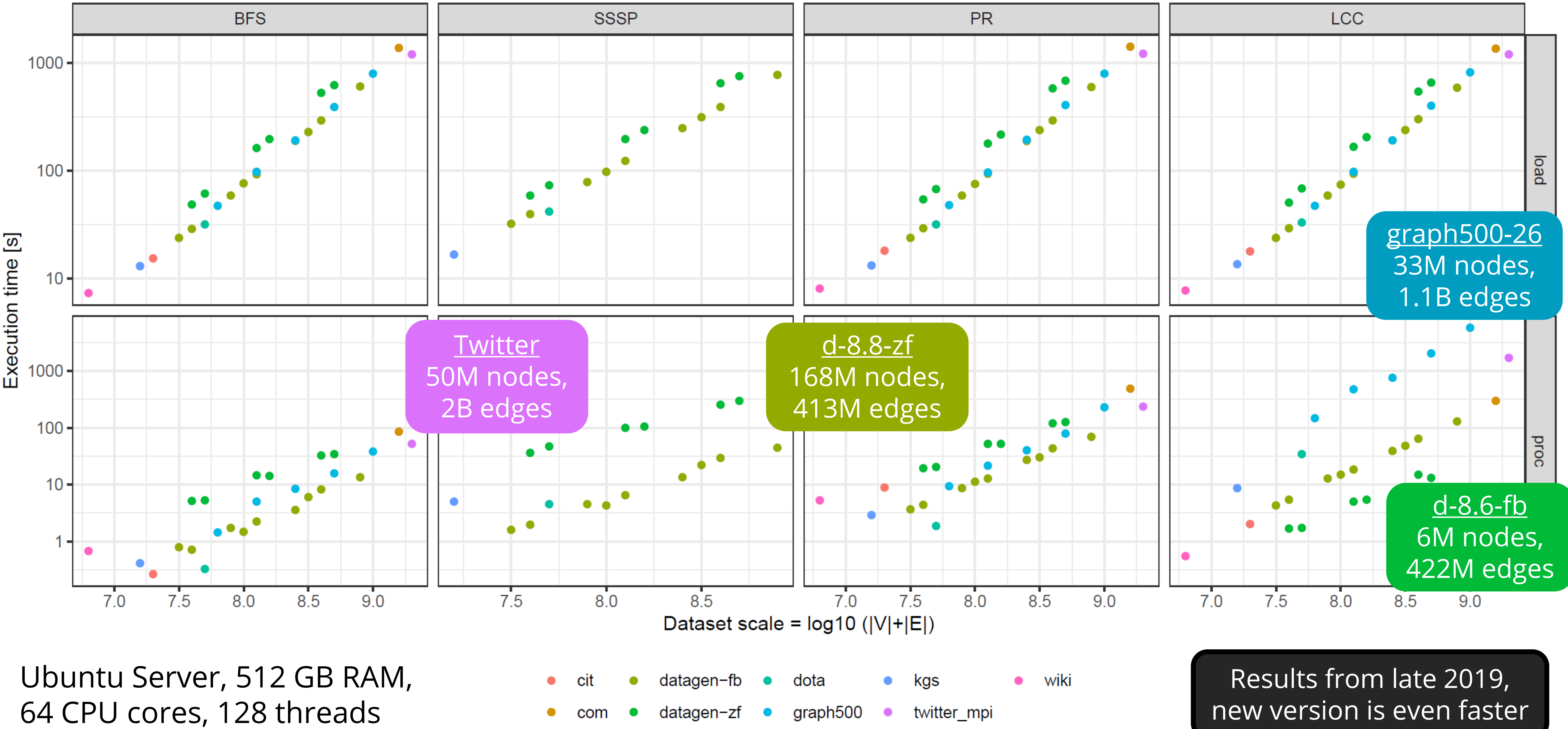
```
def sssp(A, s):
    d = Vector.from_type(A.type, A.nrows)
    d[s] = 0

    with min_plus_int64, Accum(min_int64):
        for _ in range(A.nrows):
            dn = Vector.dup(d)
            d @= A
            if dn == d:
                break
        return dn

def tricount(A):
    return A.mxm(A, mask=A).reduce_vector()
```

Benchmark results

SUITESPARSE:GRAPHBLAS / LDBC GRAPHALYTICS



Ubuntu Server, 512 GB RAM,
64 CPU cores, 128 threads

Results from late 2019,
new version is even faster

THE GAP BENCHMARK SUITE

- Part of the *Berkeley Graph Algorithm Platform* project
- **Algorithms:**
  - BFS, SSSP, PageRank, connected components
  - betweenness centrality, triangle count
- Very efficient baseline implementation in C++
- Comparing executions of implementations that were carefully optimized and fine-tuned by research groups
- Ongoing benchmark effort, paper to be submitted in Q2

S. Beamer, K. Asanovic, D. Patterson:
The GAP Benchmark Suite, arXiv, 2017

gap.cs.berkeley.edu/benchmark.html

Further reading and summary

RESOURCES

List of GraphBLAS-related books, papers, presentations, posters, and software
szarnyasg/graphblas-pointers

Library of GraphBLAS algorithms
GraphBLAS/LAGraph

Extended version of this talk: 200+ slides

- Theoretical foundations
- BFS variants, PageRank
- clustering coefficient, k-truss and triangle count variants
- Community detection using label propagation
- Luby's maximal independent set algorithm
- computing connected components on an overlay graph
- connections to relational algebra

SUMMARY

- Linear algebra is a powerful abstraction
  - Good expressive power
  - Concise formulation of most graph algorithms
  - Very good performance
  - Still lots of ongoing research
- Trade-offs:
  - Learning curve (theory and GraphBLAS API)
  - Some algorithms are difficult to formulate in linear algebra
  - Only a few GraphBLAS implementations (yet)
- **Overall:** a very promising programming model for graph algorithms suited to the age of heterogeneous hardware

ACKNOWLEDGEMENTS

- Tim Davis and Tim Mattson for helpful discussions, members of GraphBLAS mailing list for their detailed feedback.
- The LDBC Graphalytics task force for creating the benchmark and assisting in the measurements.
- Master's students at BME for developing GraphBLAS-based algorithms: Bálint Hegyi, Márton Elekes, Petra Várhegyi, Lehel Boér