

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

Gábor Szárnyas
szarnyas@mit.bme.hu



**Hungarian Academy of
Sciences**

OUTLINE

- What makes graph computations difficult?
- The GraphBLAS standard
- Theoretical foundations of GraphBLAS
- Graph algorithms in GraphBLAS
- Graph processing in relational algebra
- GraphBLAS internals and API
- Further reading and libraries
- Summary

What makes graph computations difficult?

GRAPH PROCESSING CHALLENGES

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 Bulletin 2017

GRAPH PROCESSING CHALLENGES

What does it mean that graph algorithms have “a high communication to computation ratio”? [ICCS'15]

Most of the time is spent on chasing pointers.

Speedup with a CPU that has better arithmetic performance:

- machine learning → **a lot**
- relational query → **some**
- graph processing → **very little**

Standard latency hiding techniques break down:

- pre-fetching and branch prediction provide little benefit

DISTRIBUTED GRAPH PROCESSING

- Many recent works focused on computation models for **distributed execution** allowing systems to scale out.
 - BSP (Bulk Synchronous Parallel) model by Leslie Valiant
 - MapReduce and Pregel models by Google
 - Vertex-centric, Scatter-Gather, Gather-Apply-Scatter models
 - Apache projects: Giraph, Spark GraphX, Flink Gelly, Hama
- Lots of research, summarized in survey/experiments papers



V. Kalavri et al.: *High-Level Programming Abstractions for Distributed Graph Processing*, TKDE 2018



K. Ammar, M.T. Özsu: *Experimental Analysis of Distributed Graph Systems*, VLDB 2018



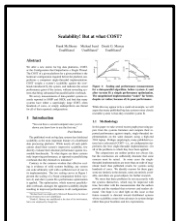
O. Batarfi et al.: *Large scale graph processing systems: survey and an experimental evaluation*, Cluster 2015



M.T. Özsu: *Graph Processing: A Panoramic View and Some Open Problems*, VLDB 2019

SCALING OUT VS. SCALING UP

- Distributed approaches are **scalable but comparatively slow**
 - large communication overhead
 - load balancing issues due to irregular distributions
- Many systems **struggle to outperform a single-threaded setup**
 - COST = Configuration that Outperforms a Single Thread
- Alternatives:
 - Partition-centric programming model (Blogel, etc.)
 - Linear algebra-based programming model



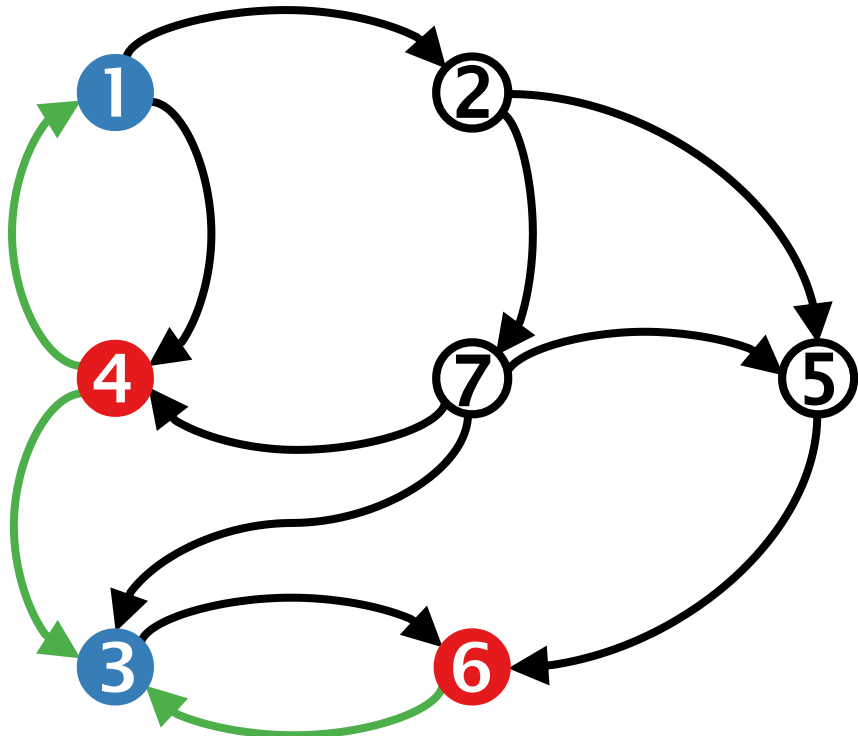
F. McSherry et al.:
Scalability! But at what COST?
HotOS 2015



N. Satish et al.:
*Navigating the Maze of Graph Analytics Frameworks
using Massive Graph Datasets*, SIGMOD 2014

LINEAR ALGEBRA-BASED GRAPH PROCESSING

- Graphs are encoded as sparse adjacency matrices.
- Use vector/matrix operations to express graph algorithms.

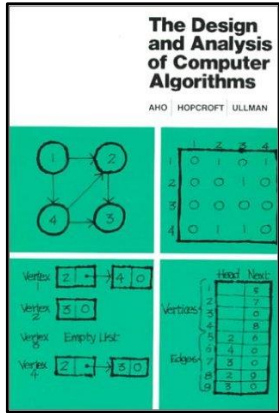


A	①	②	③	④	⑤	⑥	⑦
①		1		1			
②					1		1
③						1	
④	1		1				
⑤						1	
⑥			1				
⑦			1	1	1		

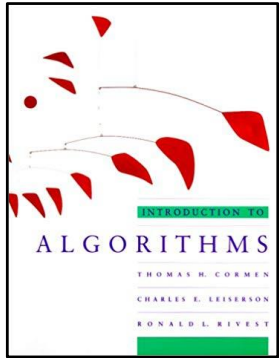
v	①	②	③	④	⑤	⑥	⑦
				1		1	

vA	①	②	③	④	⑤	⑥	⑦
	1		2				

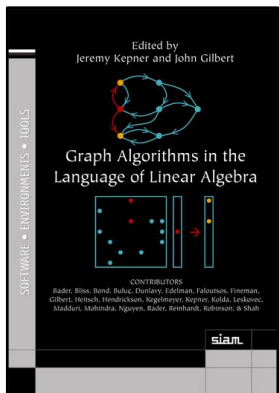
TEXTBOOKS ON SEMIRING-BASED GRAPH PROCESSING



- 1974: Aho-Hopcroft-Ullman book
 - *The Design and Analysis of Computer Algorithms*
 - Sec. 5.6: *Path-finding problems*
 - Sec. 5.9: *Path problems and matrix multiplication*



- 1990: Cormen-Leiserson-Rivest book
 - *Introduction to Algorithms*
 - Sec. 26.4: *A general framework for solving path problems in directed graphs*



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

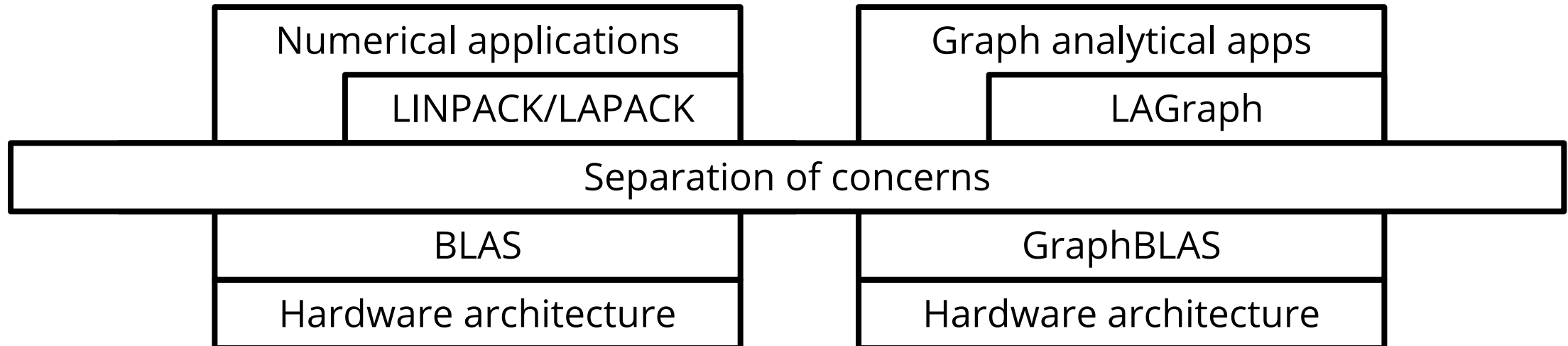
A lot of literature but few practical implementations.

The GraphBLAS standard

THE GRAPHBLAS STANDARD

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

- 1979: BLAS Basic Linear Algebra Subprograms
- 2001: Sparse BLAS an extension to BLAS (little uptake)
- 2013: GraphBLAS an effort to define standard building blocks for graph algorithms in the language of linear algebra

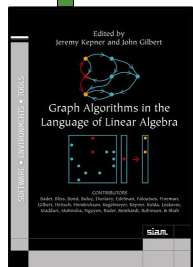
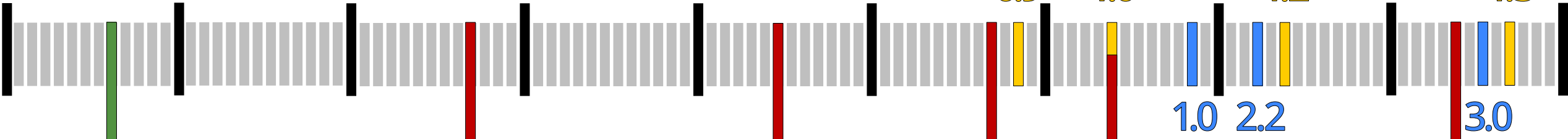


S. McMillan @ SEI Research Review (Carnegie Mellon University, 2015):
Graph algorithms on future architectures

GRAPHBLAS TIMELINE

Book — Papers — GraphBLAS standards — SuiteSparse:GraphBLAS releases

2011 2012 2013 2014 2015 2016 2017 2018 2019



Graph Algorithms in the Language of Linear Algebra



Standards for graph algorithm primitives, HPEC



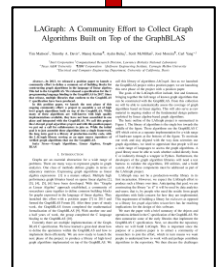
Seven good reasons, ICCS



Mathematical foundations, HPEC



C API, GABB@ IPDPS



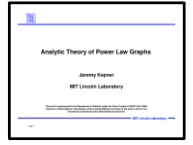
LAGraph, GrAPL@ IPDPS

GRAPH ALGORITHMS IN LINEAR ALGEBRA

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	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	$m + n \log n$	$m \log n$



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

Theoretical foundations of GraphBLAS

MATRIX MULTIPLICATION

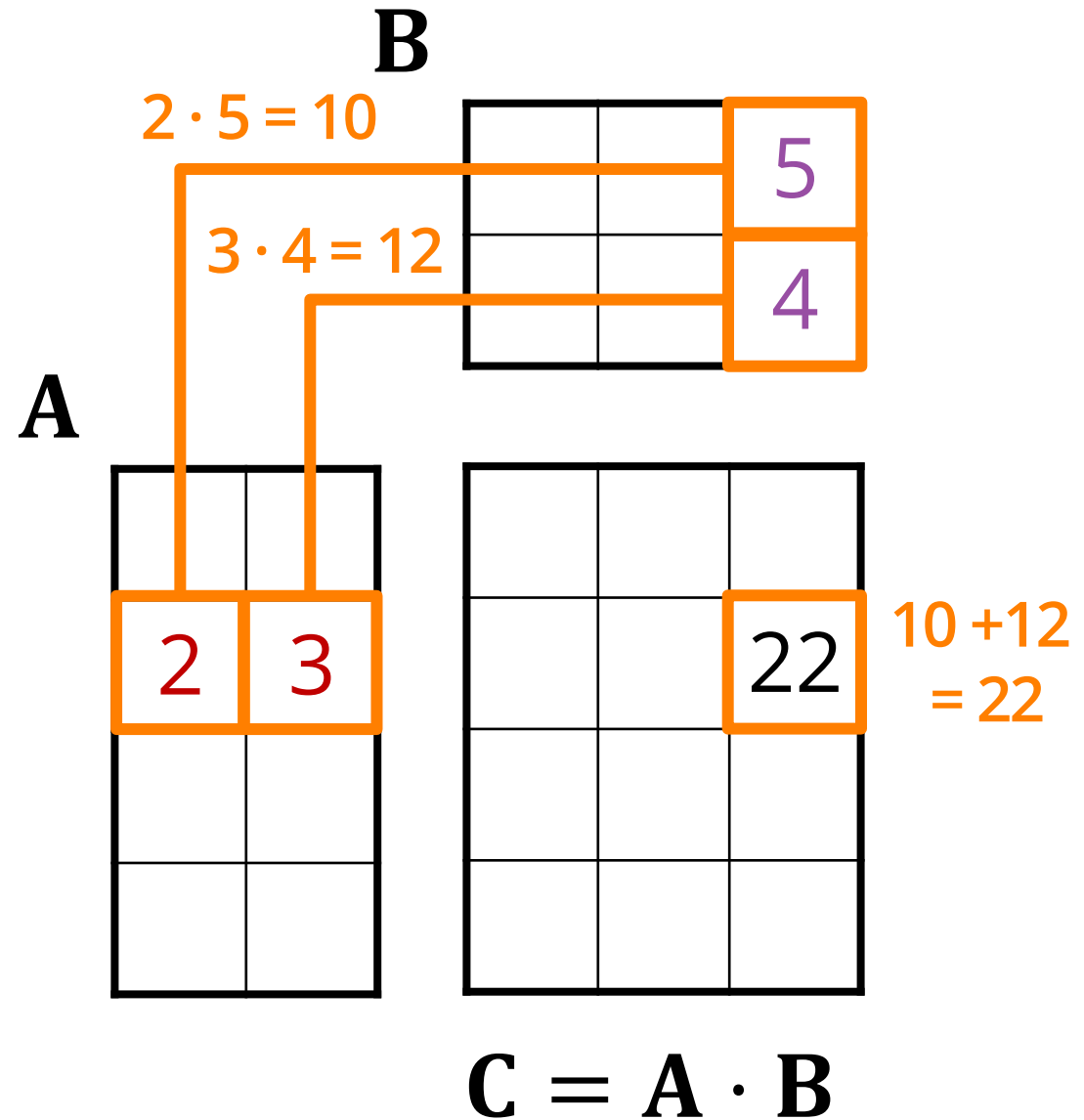
Definition:

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

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

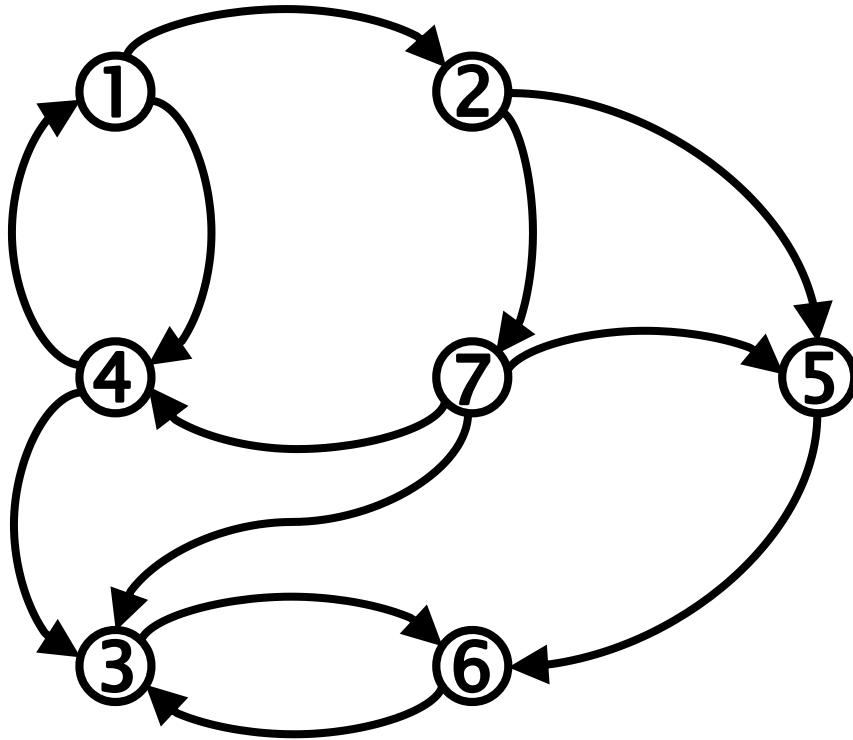
Example:

$$\begin{aligned}\mathbf{C}(2,3) &= \mathbf{A}(2,1) \cdot \mathbf{B}(1,3) + \\ &\quad \mathbf{A}(2,2) \cdot \mathbf{B}(2,3) \\ &= 2 \cdot 5 + 3 \cdot 4 = 22\end{aligned}$$



ADJACENCY MATRIX

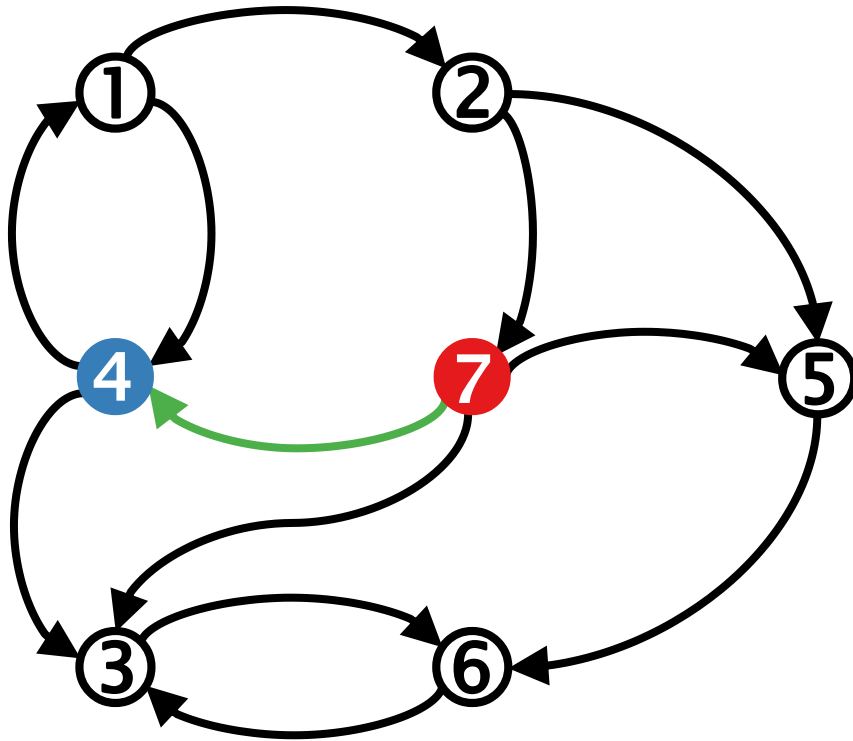
$$A_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E \\ 0 & \text{if } (v_i, v_j) \notin E \end{cases}$$



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

ADJACENCY MATRIX

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

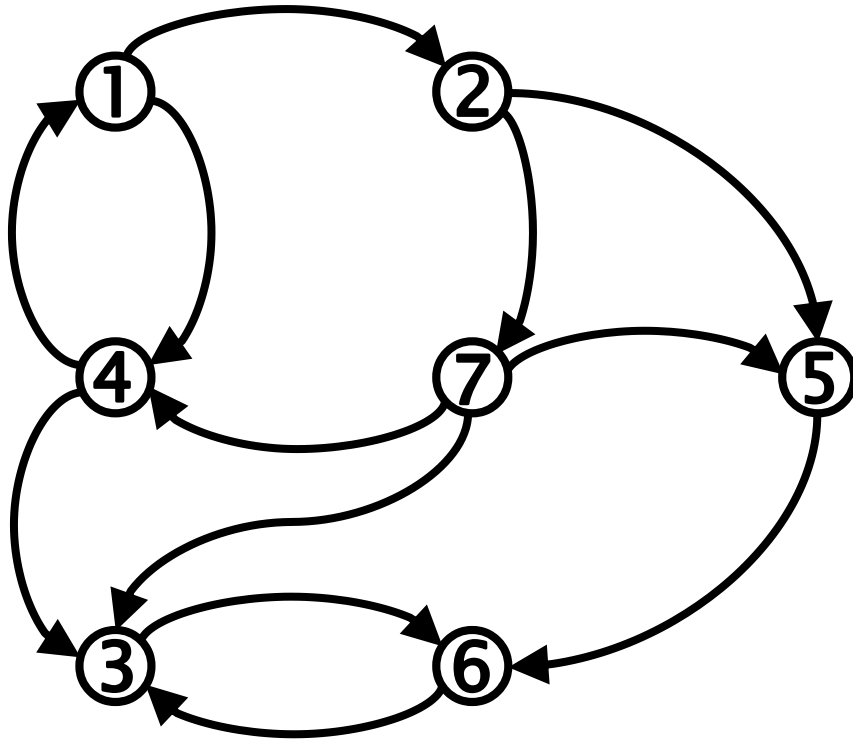


target

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		

ADJACENCY MATRIX

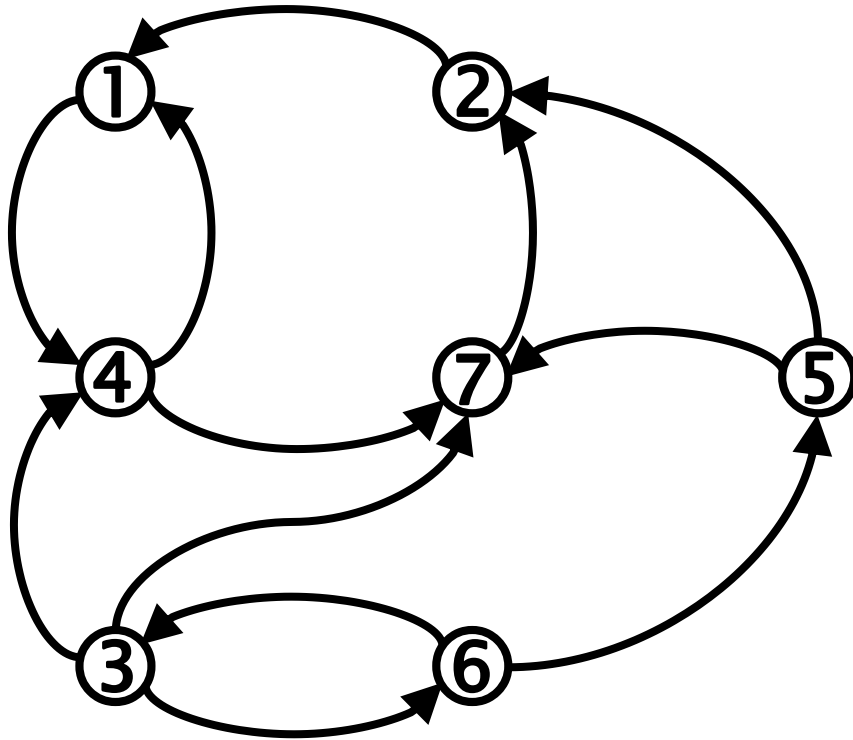
$$A_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E \\ 0 & \text{if } (v_i, v_j) \notin E \end{cases}$$



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

ADJACENCY MATRIX TRANSPOSED

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

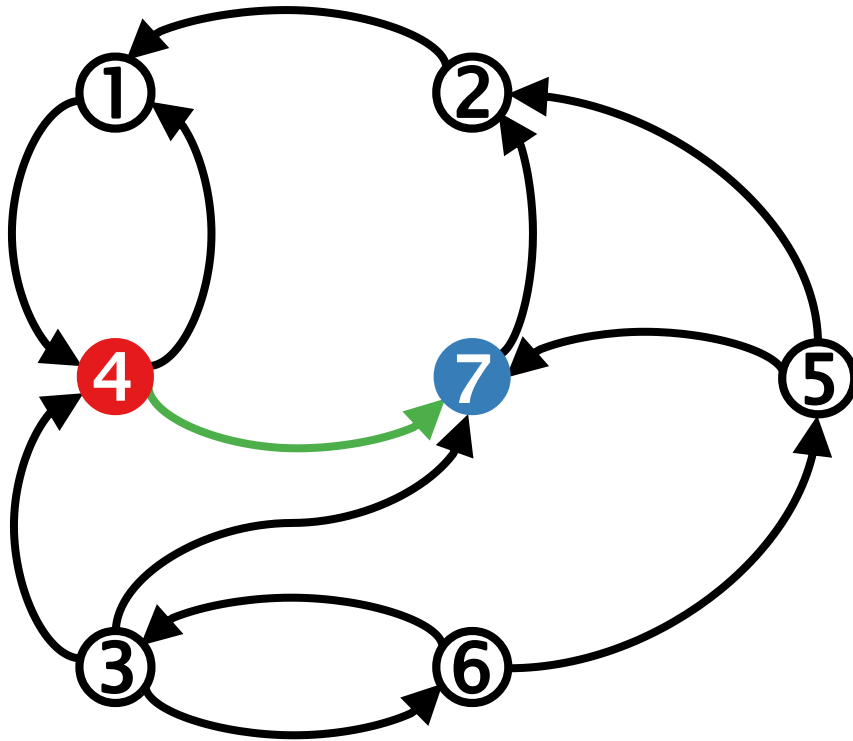


\mathbf{A}^T

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

ADJACENCY MATRIX TRANSPOSED

$$A_{ij}^T = \begin{cases} 1 & \text{if } (v_j, v_i) \in E \\ 0 & \text{if } (v_j, v_i) \notin E \end{cases}$$



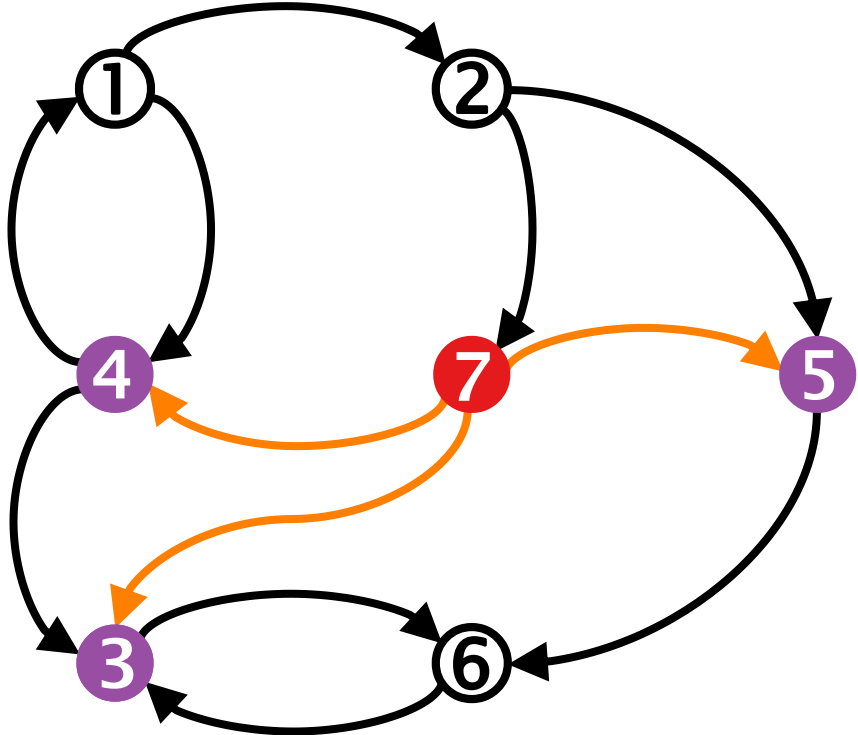
source

target

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

GRAPH TRAVERSAL WITH MATRIX MULTIPLICATION

vA^k means k hops in the graph



A

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

v

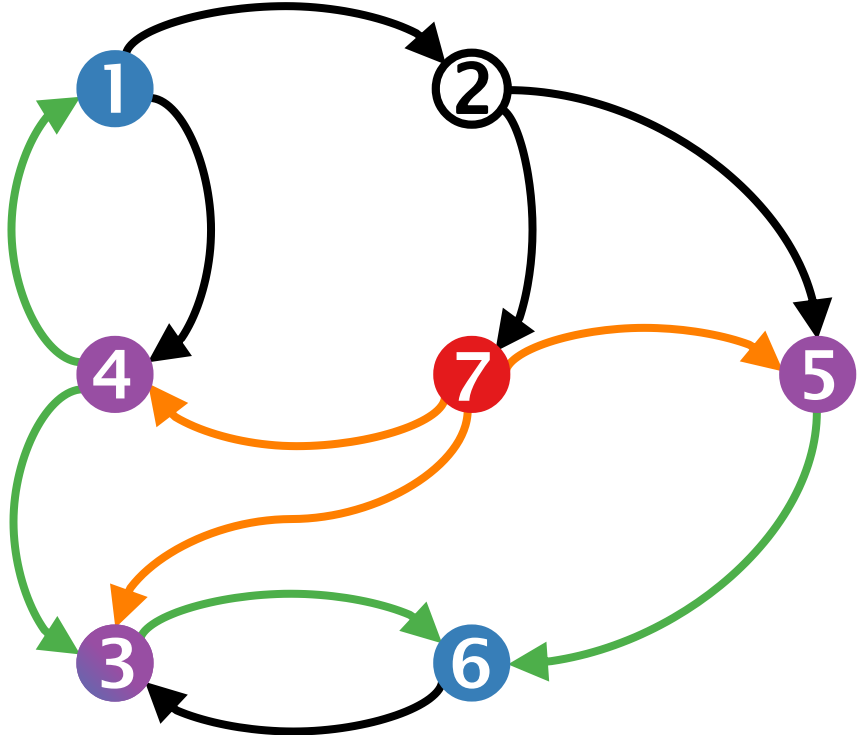
①	②	③	④	⑤	⑥	⑦
						1

①	②	③	④	⑤	⑥	⑦
		1	1	1		

one-hop: vA

GRAPH TRAVERSAL WITH MATRIX MULTIPLICATION

vA^k means k hops in the graph



v

①	②	③	④	⑤	⑥	⑦
						1

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
		1	1	1		

one-hop: vA

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
1		1			2	

two-hop: vA^2

MATRIX MULTIPLICATION ON SEMIRINGS

- Using the conventional semiring

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

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

- Generalized formula*

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

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

- **A cornerstone of GraphBLAS:** Use arbitrary semirings that override the \oplus addition and \otimes multiplication operators.

* **Remark:** Some definitions in the presentation are simplified. The full definitions are given at the end of the slideshow.

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 using the addition operation $\oplus: D \times D \rightarrow D$, 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 $\otimes: D \times D \rightarrow D$.

The mathematical definition of a semiring requires that \otimes is a monoid and distributes over \oplus . GraphBLAS omits these requirements.

SEMIRINGS

semiring	set	\oplus	\otimes	0	graph semantics
lor-land	$a \in \{F, T\}$	\vee	\wedge	F	connectivity
integer arithmetic	$a \in \mathbb{N}$	+	\cdot	0	number of paths
real arithmetic	$a \in \mathbb{R}$	+	\cdot	0	strength of all paths
min-plus	$a \in \mathbb{R} \cup \{+\infty\}$	min	+	$+\infty$	shortest path
max-plus	$a \in \mathbb{R} \cup \{-\infty\}$	max	+	$-\infty$	graph matching

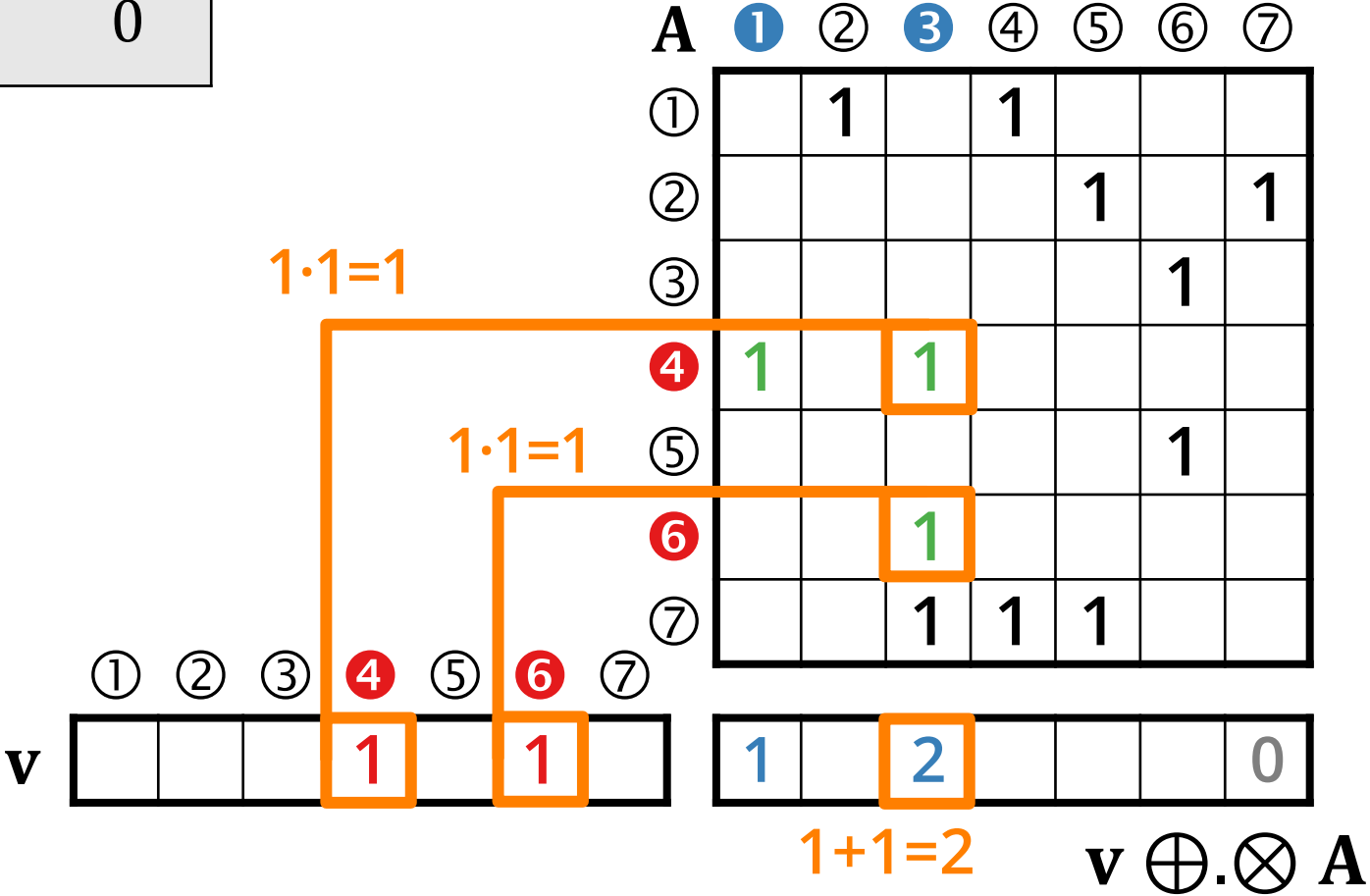
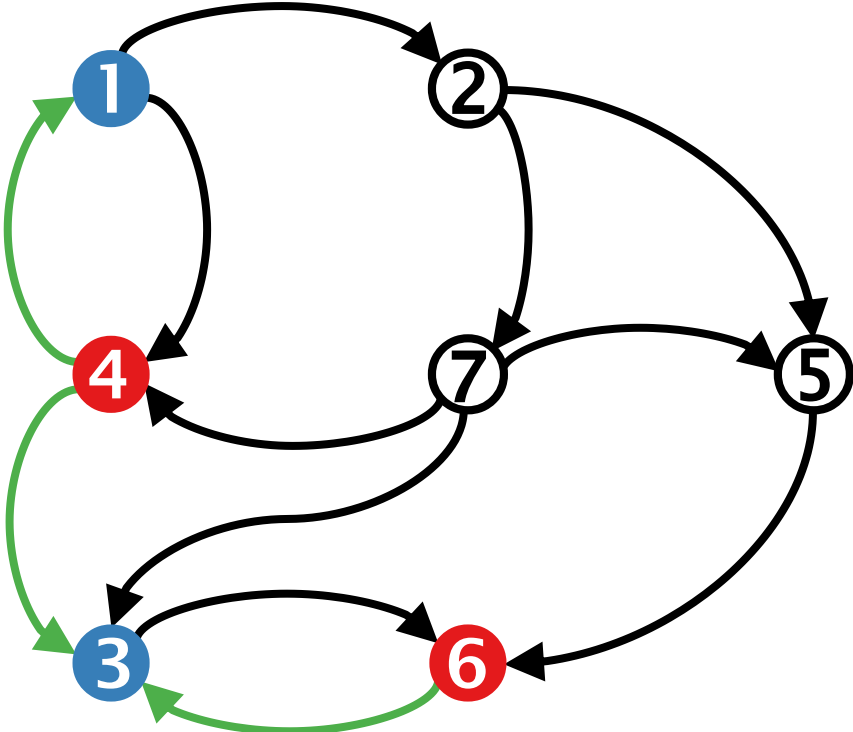
The default semiring is the conventional one:

- Operator \otimes defaults to floating point multiplication.
- Operator \oplus defaults to floating point addition.

MATRIX-VECTOR MULTIPLICATION SEMANTICS

semiring	set	\oplus	\otimes	0
integer arithmetic	$a \in \mathbb{N}$	+	·	0

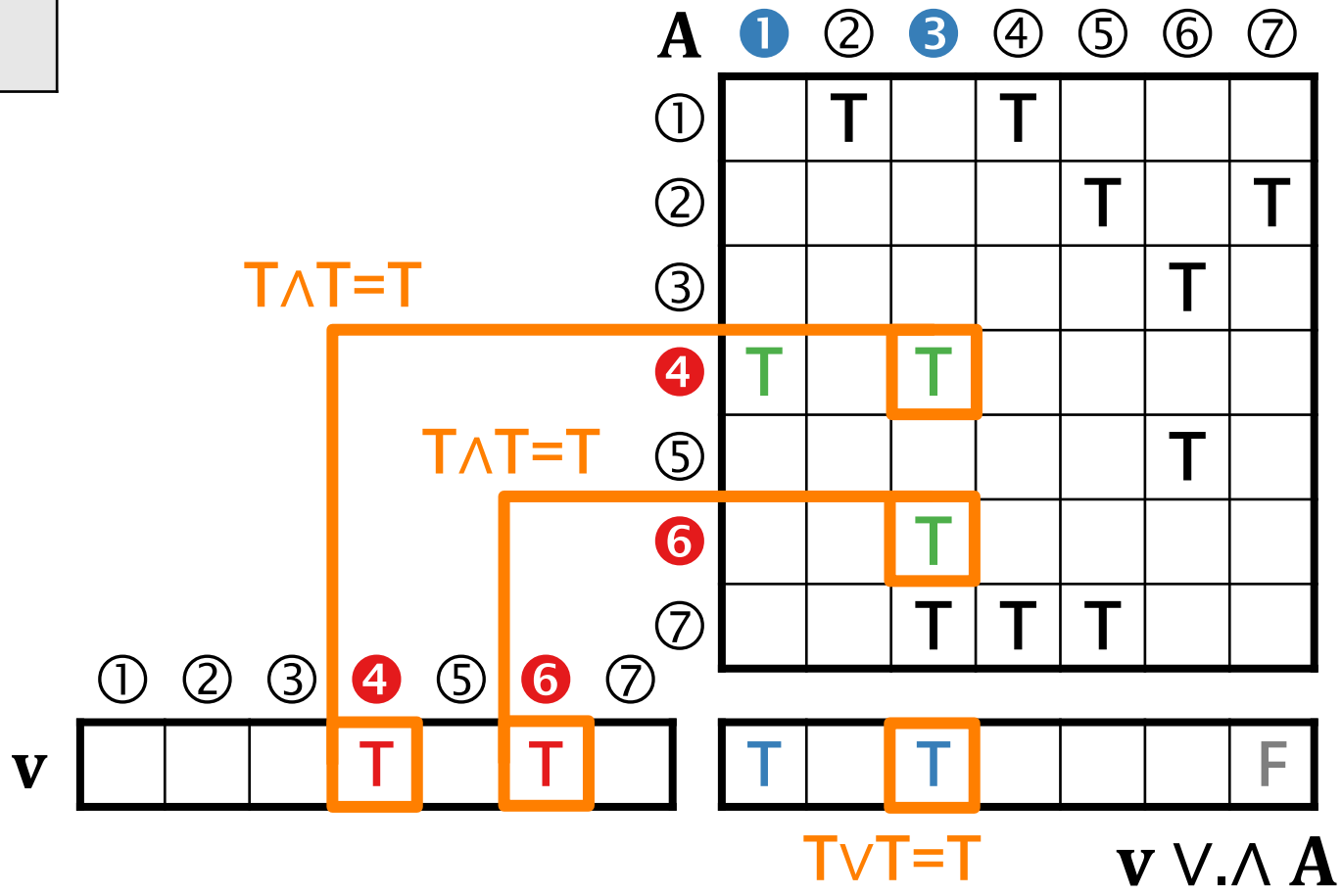
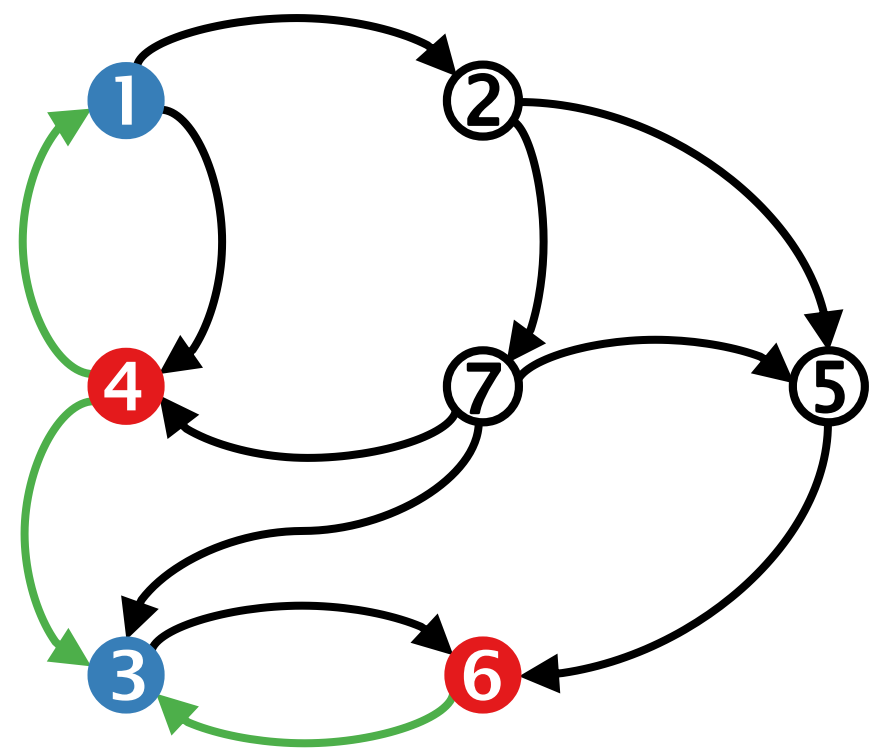
Semantics: number of paths



MATRIX-VECTOR MULTIPLICATION SEMANTICS

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

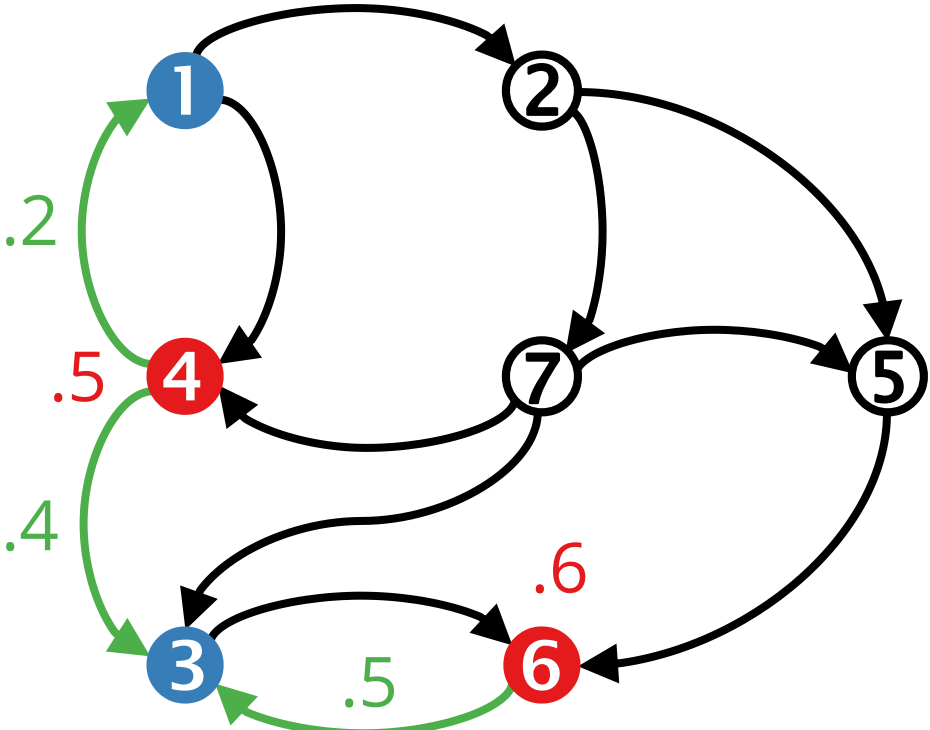
Semantics: reachability



MATRIX-VECTOR MULTIPLICATION SEMANTICS

semiring	set	\oplus	\otimes	0
real arithmetic	$a \in \mathbb{R}$	+	\cdot	0

Semantics: strength of all paths



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

v	①	②	③	④	⑤	⑥	⑦
				.5		.6	

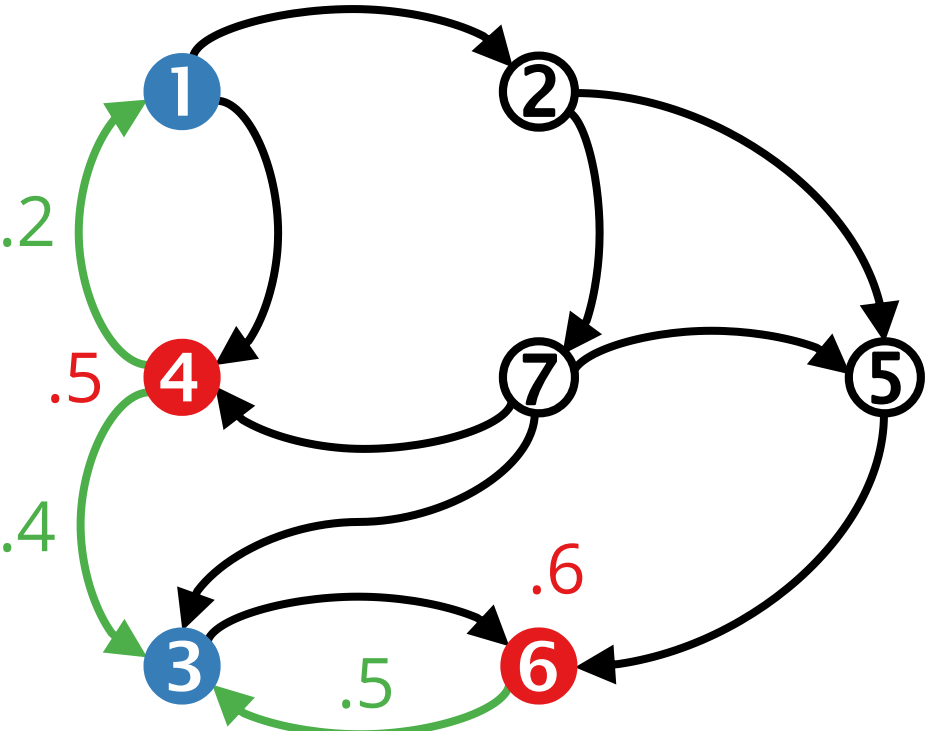
$v \oplus \cdot \otimes A$	①	②	③	④	⑤	⑥	⑦
	.1		.5				0

$0.5 \cdot 0.4 = 0.2$
 $0.6 \cdot 0.5 = 0.3$
 $0.2 + 0.3 = 0.5$

MATRIX-VECTOR MULTIPLICATION SEMANTICS

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

Semantics: shortest path



Matrix A and vector v illustrating matrix-vector multiplication in the min-plus semiring.

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

Vector v:

v	①	②	③	④	⑤	⑥	⑦
				.5		.6	

Calculation steps:

- $0.5 + 0.4 = 0.9$ (path 1 → 4 → 3)
- $0.6 + 0.5 = 1.1$ (path 3 → 6 → 7)
- $\min(0.9, 1.1) = 0.9$

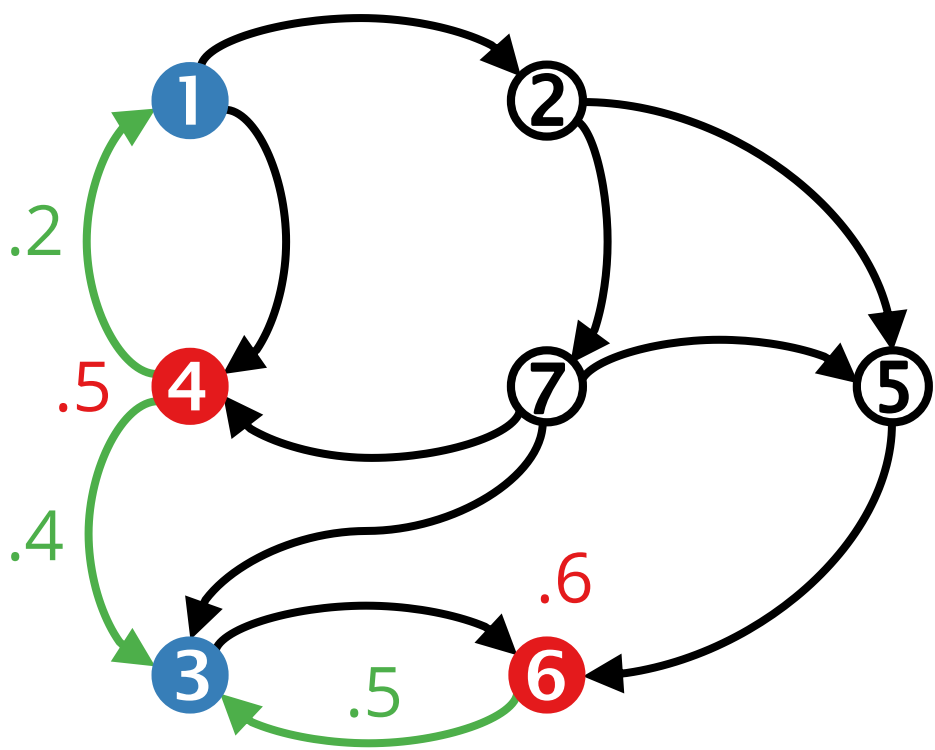
Resulting vector v min . + A:

	①	②	③	④	⑤	⑥	⑦
	.7		.9				∞

MATRIX-VECTOR MULTIPLICATION SEMANTICS

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

Semantics: matching (independent edge set)



Matrix A and vector v illustrating matrix-vector multiplication in the max-plus semiring.

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

v	①	②	③	④	⑤	⑥	⑦
				.5		.6	
	.7			1.1			$-\infty$

Calculations shown:

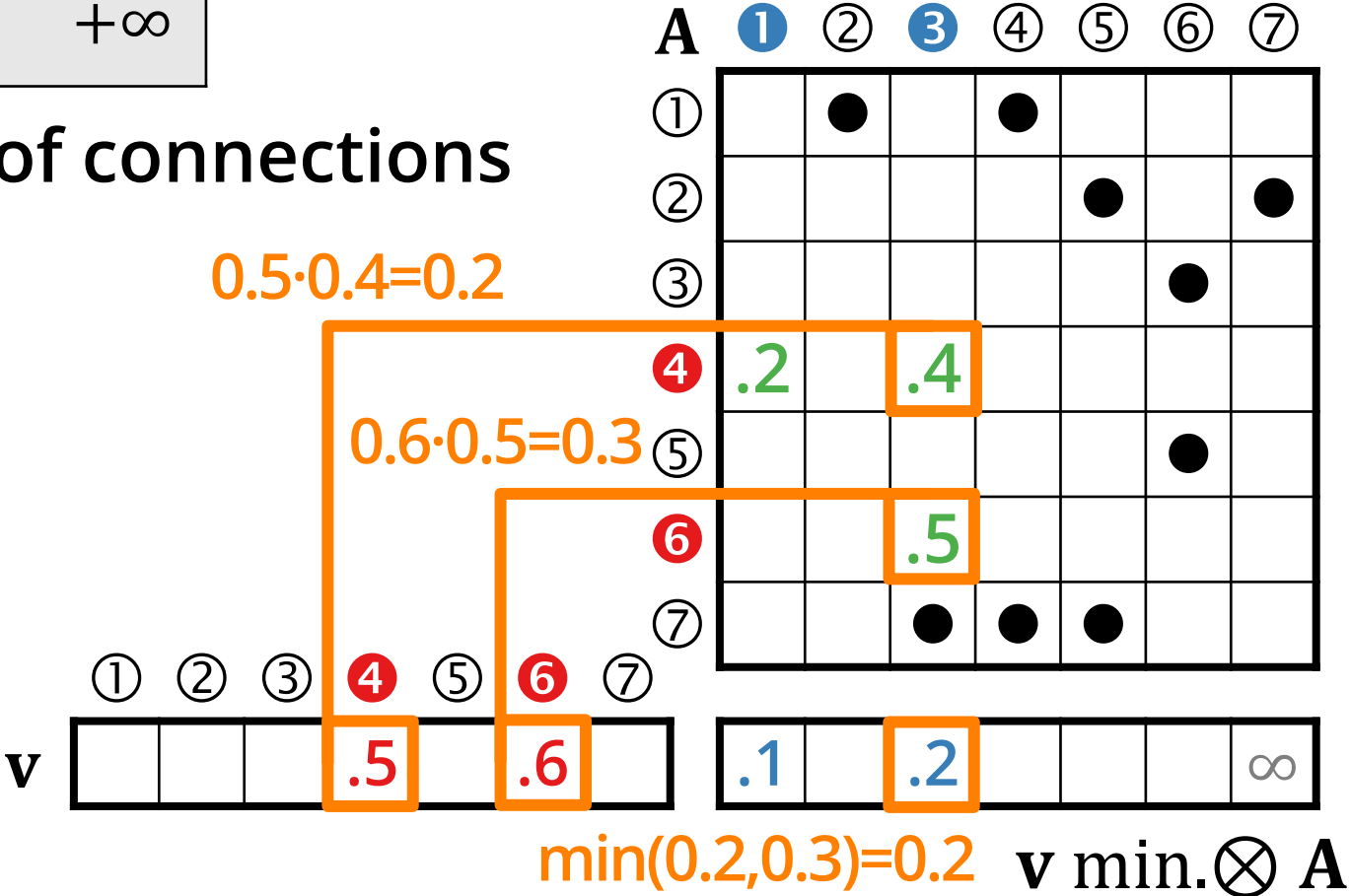
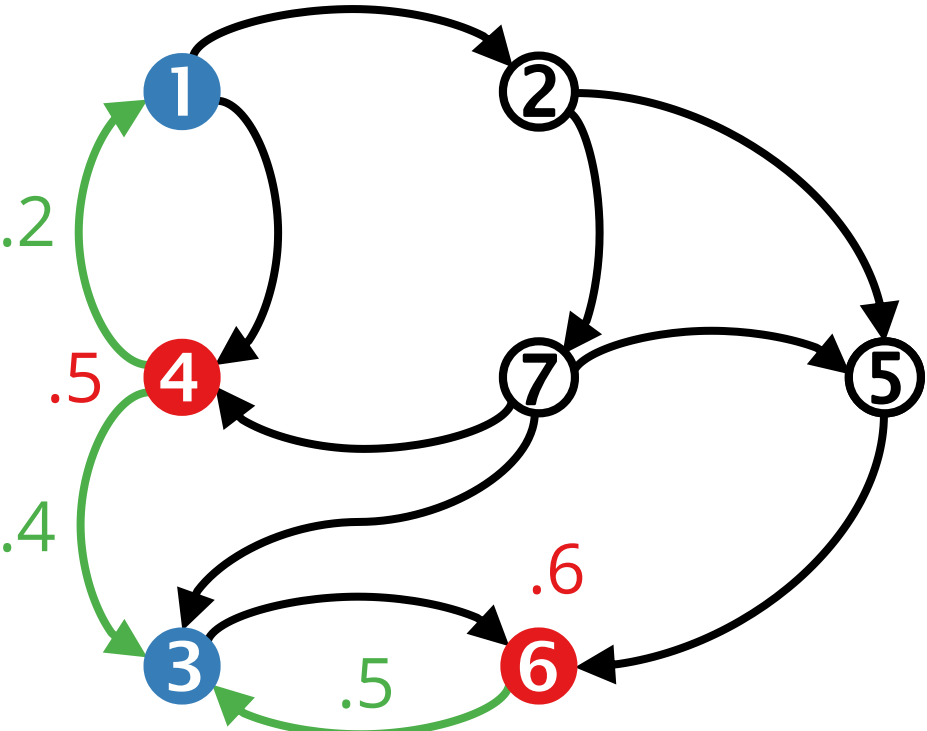
- $0.5 + 0.4 = 0.9$ (for path 1→4→3)
- $0.6 + 0.5 = 1.1$ (for path 1→4→6)
- $\max(0.9, 1.1) = 1.1$ (for node 3)

Final result: $v \max . + A$

MATRIX-VECTOR MULTIPLICATION SEMANTICS

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

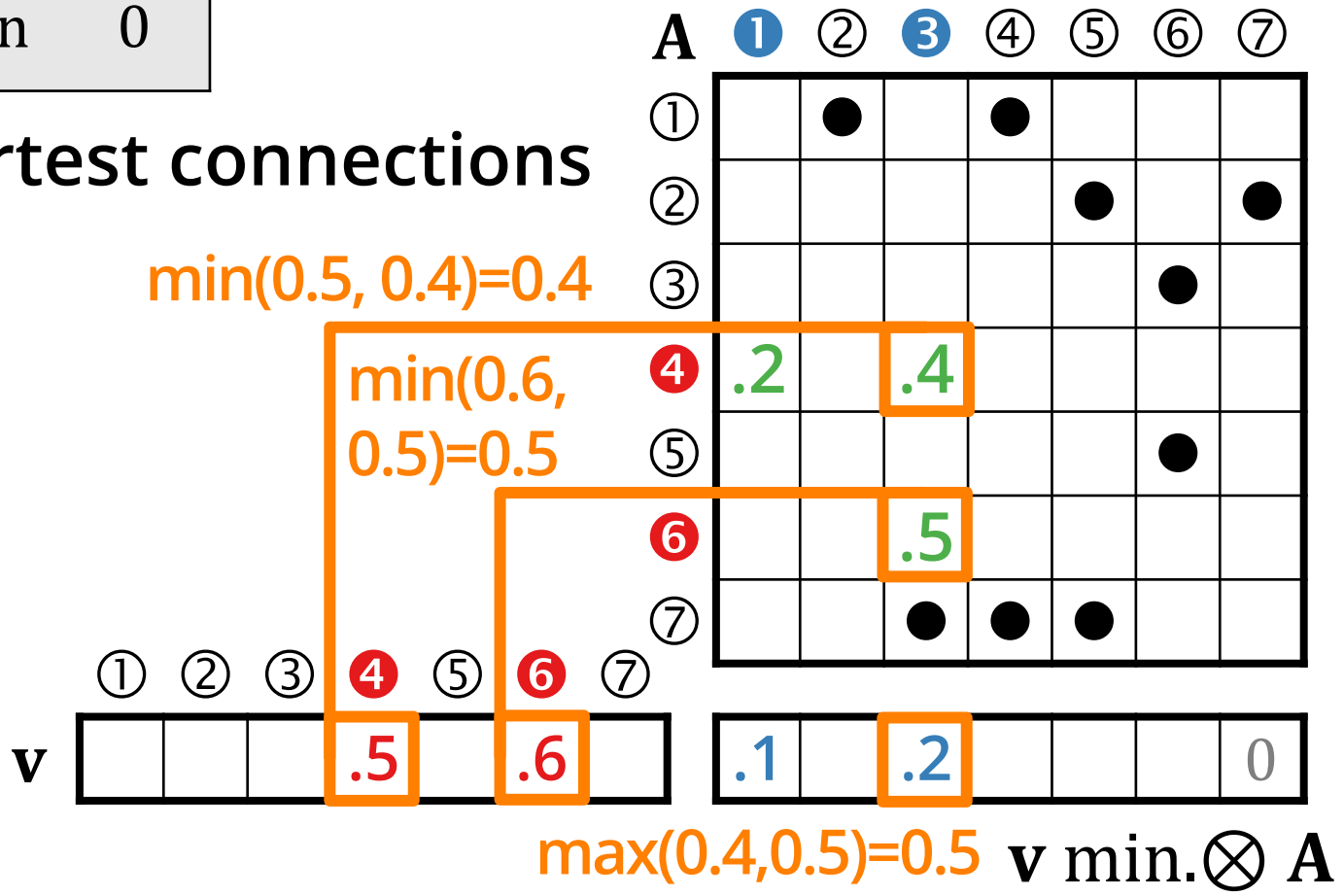
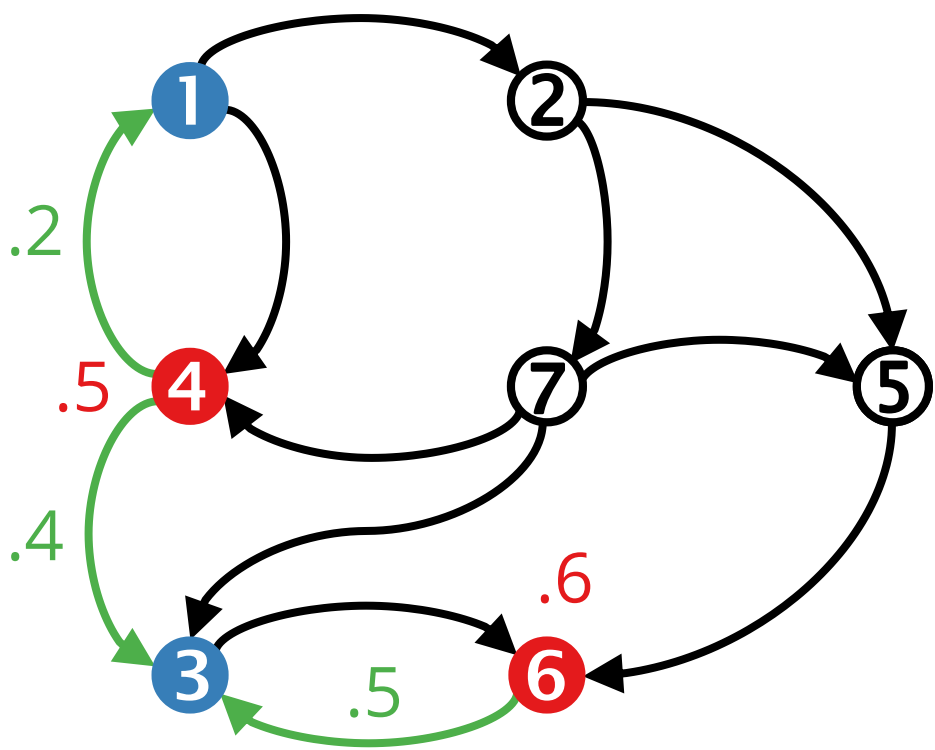
Semantics: shortest product of connections



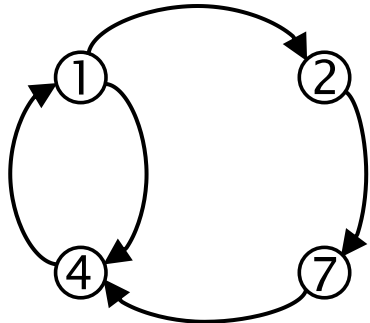
MATRIX-VECTOR MULTIPLICATION SEMANTICS

semiring	set	\oplus	\otimes	0
max-min	$a \in \{0, +\infty\}$	max	min	0

Semantics: longest of all shortest connections



ELEMENT-WISE MULTIPLICATION: $A \wedge B$



⑤

\wedge

①

②

⑦

⑤

=

①

②

⑦

⑤

③

⑥

③

⑥

③

⑥

A

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

\wedge

B

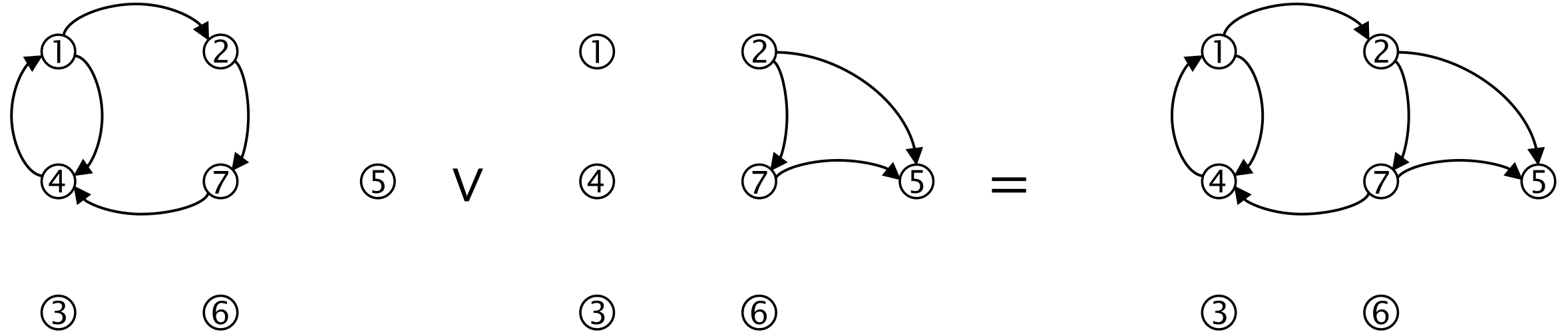
	①	②	③	④	⑤	⑥	⑦
①							
②					●		●
③							
④							
⑤							
⑥							
⑦					●		

=

A \wedge B

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

ELEMENT-WISE ADDITION: $A \vee B$



A

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

\vee

B

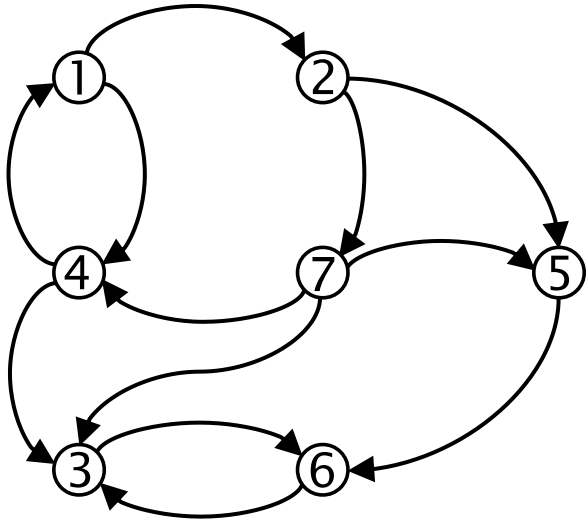
	①	②	③	④	⑤	⑥	⑦
①							
②					●		●
③							
④							
⑤							
⑥							
⑦					●		

$=$

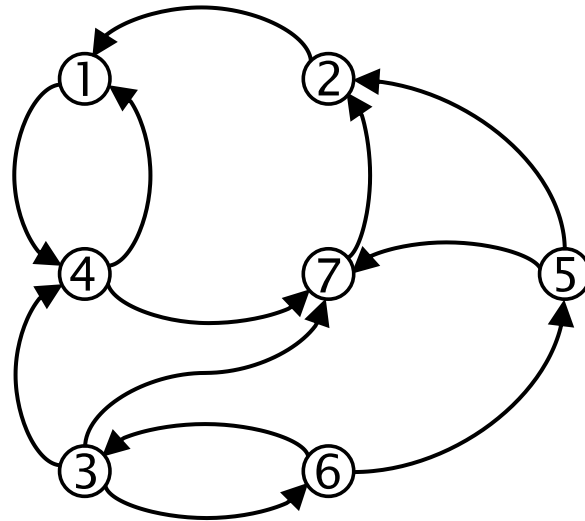
A v B

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

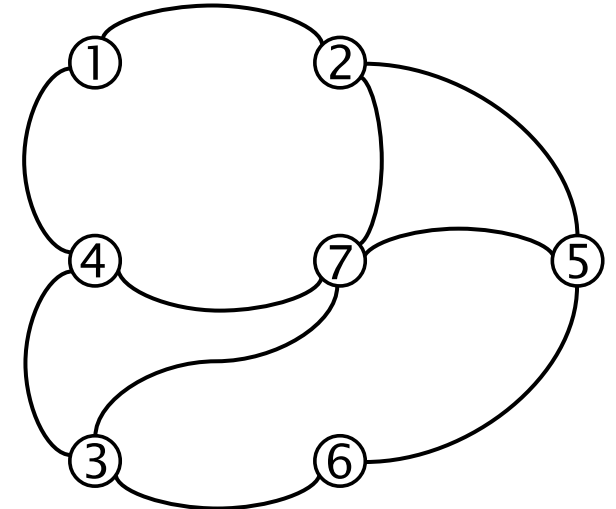
TURNING A GRAPH INTO UNDIRECTED: $A \vee A^T$



\vee



$=$



A

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

\vee

A^T

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

$=$

$A \vee A^T$

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

NOTATION*

- Symbols:
 - **A, B, C, M** – matrices
 - **u, v, w, m** – vectors
 - *s* – scalar
 - *i, j* – indices
 - **⟨M⟩, ⟨m⟩** – masks
- Operators:
 - \oplus – addition
 - \otimes – multiplication
 - \top – transpose
 - \oslash – element-wise division

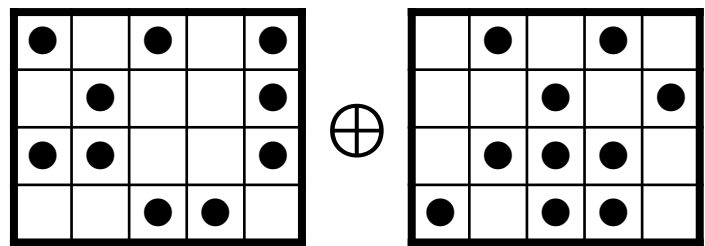
symbol	operation	notation
$\oplus.\otimes$	matrix-matrix multiplication	$\mathbf{C}\langle\mathbf{M}\rangle = \mathbf{A} \oplus.\otimes \mathbf{B}$
	vector-matrix multiplication	$\mathbf{w}\langle\mathbf{m}\rangle = \mathbf{v} \oplus.\otimes \mathbf{A}$
	matrix-vector multiplication	$\mathbf{w}\langle\mathbf{m}\rangle = \mathbf{A} \oplus.\otimes \mathbf{v}$
\otimes	element-wise multiplication	$\mathbf{C}\langle\mathbf{M}\rangle = \mathbf{A} \otimes \mathbf{B}$
	(set intersection of patterns)	$\mathbf{w}\langle\mathbf{m}\rangle = \mathbf{u} \otimes \mathbf{v}$
\oplus	element-wise addition	$\mathbf{C}\langle\mathbf{M}\rangle = \mathbf{A} \oplus \mathbf{B}$
	(set union of patterns)	$\mathbf{w}\langle\mathbf{m}\rangle = \mathbf{u} \oplus \mathbf{v}$
<i>f</i>	apply unary operator	$\mathbf{C}\langle\mathbf{M}\rangle = f(\mathbf{A})$
		$\mathbf{w}\langle\mathbf{m}\rangle = f(\mathbf{v})$
$[\oplus \dots]$	reduce to vector	$\mathbf{w}\langle\mathbf{m}\rangle = [\oplus_j \mathbf{A}(:, j)]$
	reduce to scalar	$s = [\oplus_{ij} \mathbf{A}(i, j)]$
\mathbf{A}^\top	transpose matrix	$\mathbf{C}\langle\mathbf{M}\rangle = \mathbf{A}^\top$

Vectors can act as both column and row vectors.

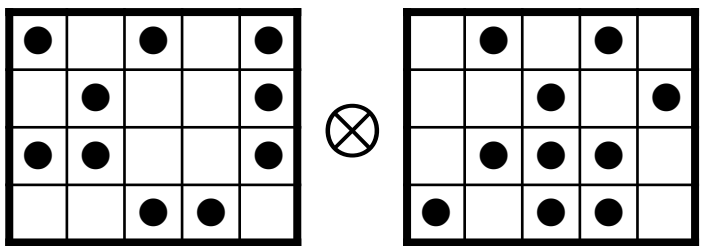
(Notation omitted for accumulator, selection, extraction, assignment...)

LINEAR ALGEBRAIC PRIMITIVES FOR GRAPHS #1

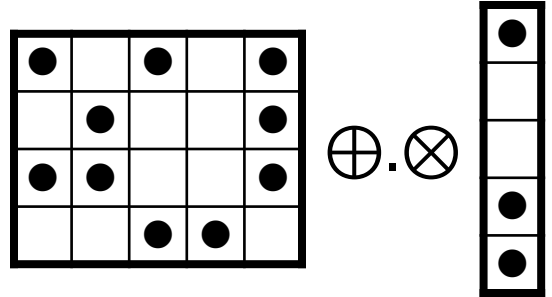
Element-wise addition:
union of non-zero elements



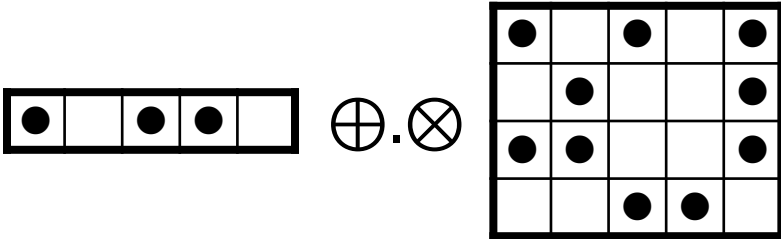
Element-wise multiplication:
intersection of non-zero elements



Sparse matrix times sparse vector:
process incoming edges

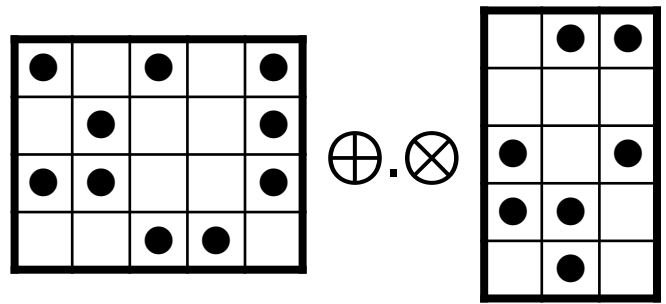


Sparse vector times sparse matrix:
process outgoing edges

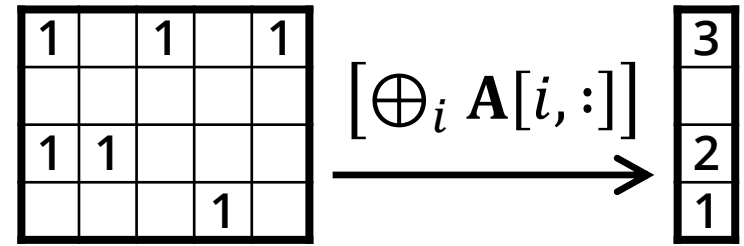


LINEAR ALGEBRAIC PRIMITIVES FOR GRAPHS #2

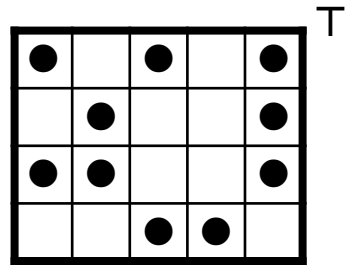
Sparse matrix times sparse matrix:
process connecting outgoing edges



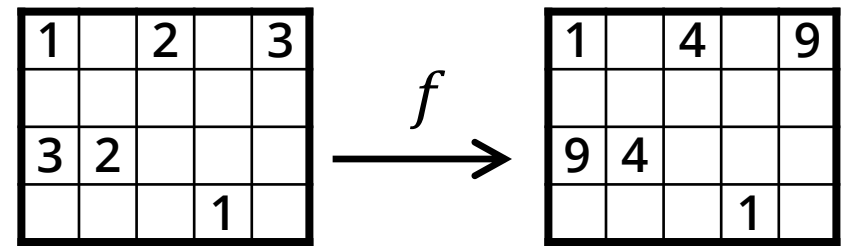
Reduction:
aggregate values in each row



Matrix transpose:
reverse edges



Apply:
apply unary operator on all values



Graph algorithms in GraphBLAS

ALGORITHMS

- Breadth-first search
 - Levels
 - Parents
 - Multi-source
- Bellman-Ford
- PageRank
- Triangle count
 - Naïve
 - Masked
 - Cohen's algorithm
 - Sandia
 - CMU
- Local clustering coefficient
- k -truss
- CDLP (Community Detection using Label Propagation)

Graph algorithms in GraphBLAS

Breadth-first search

BFS: BREADTH-FIRST SEARCH

- **Algorithm:**

- Start from a given vertex
- “Explore all neighbour vertices at the present level prior to moving on to the vertices at the next level” [Wikipedia]

- **Variants:**

- **Levels** compute traversal level for each vertex
- **Parents** compute parent for each vertex
- **MSBFS** start traversal from multiple source vertices

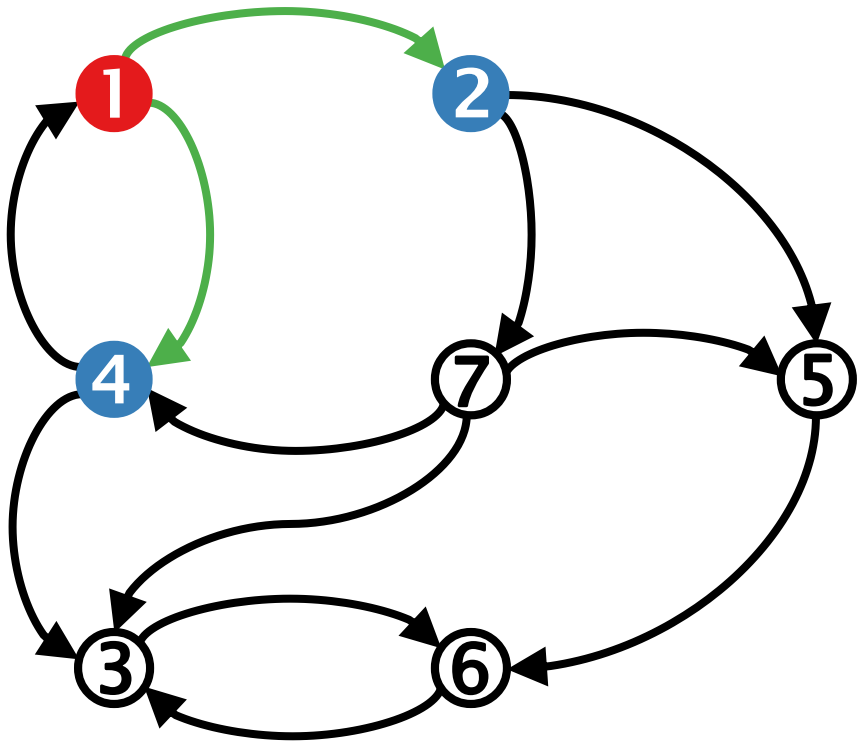
Graph algorithms in GraphBLAS

Breadth-first search / Levels

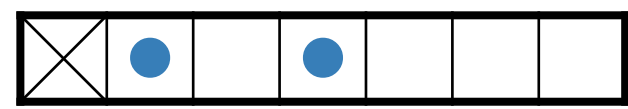
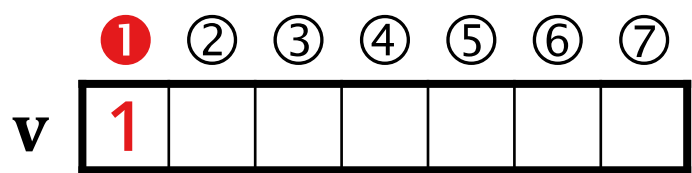
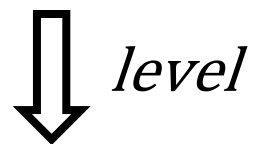
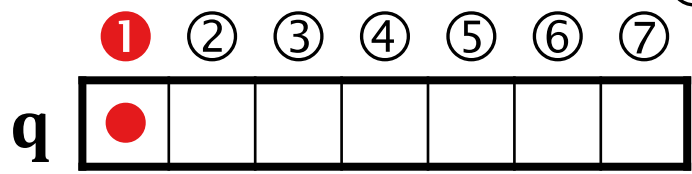
BFS - LEVELS

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

level = 1



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

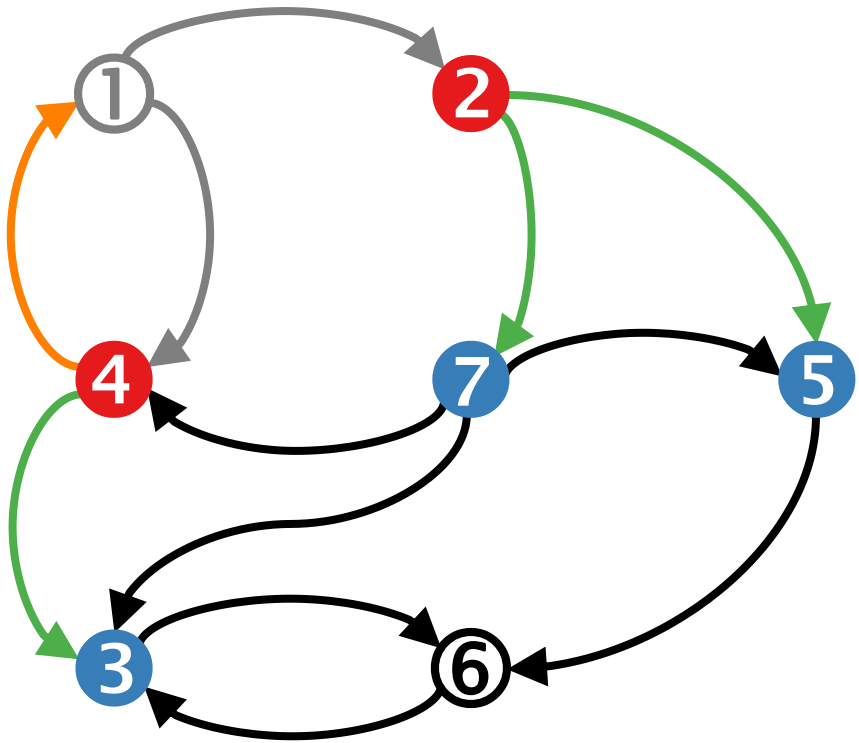


$q' \langle \bar{v} \rangle = q \vee \wedge A$

BFS - LEVELS

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

level = 2



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

q	①	②	③	④	⑤	⑥	⑦
		●		●			

level

v	①	②	③	④	⑤	⑥	⑦
	1	2		2			

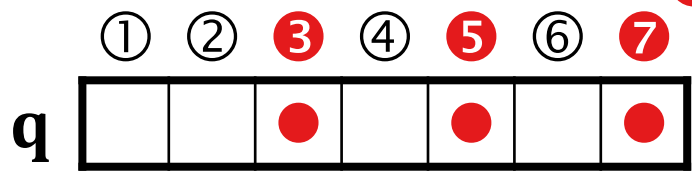
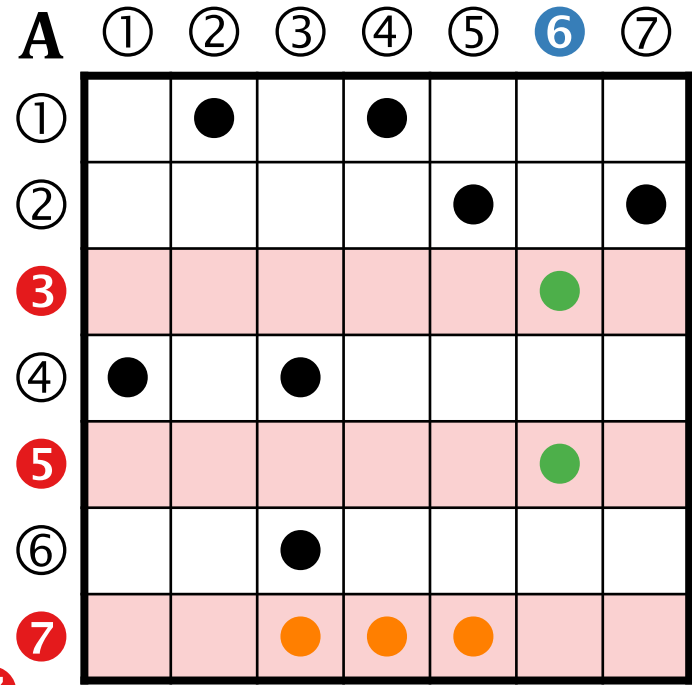
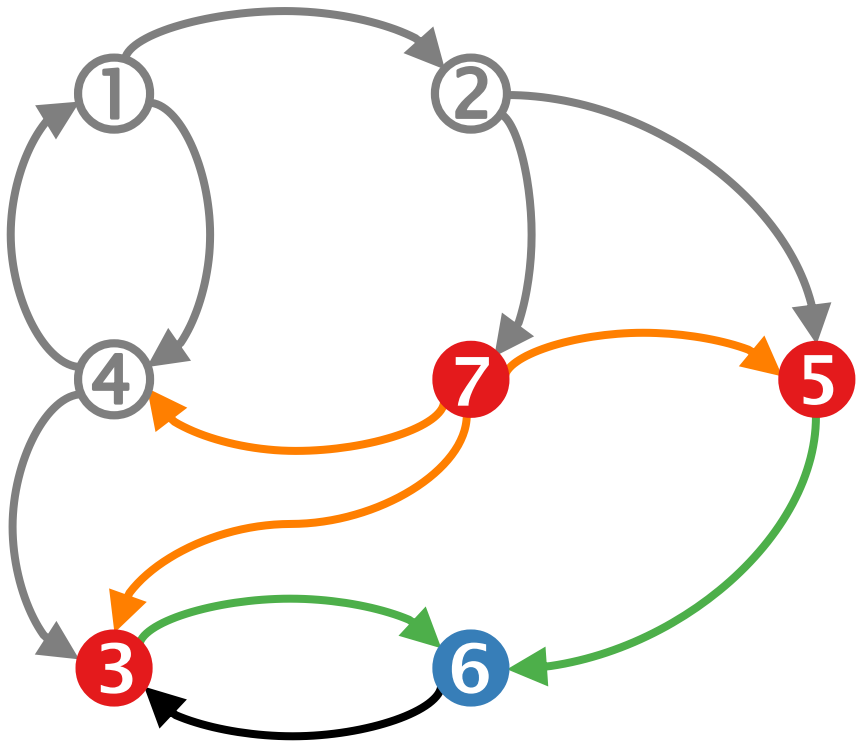
⊗	⊗	●	⊗	●		●
---	---	---	---	---	--	---

$$q' \langle \bar{v} \rangle = q \vee \wedge A$$

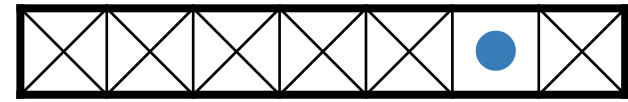
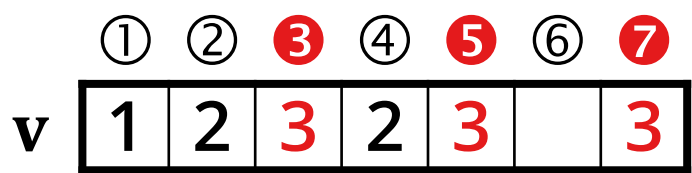
BFS - LEVELS

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

level = 3



level

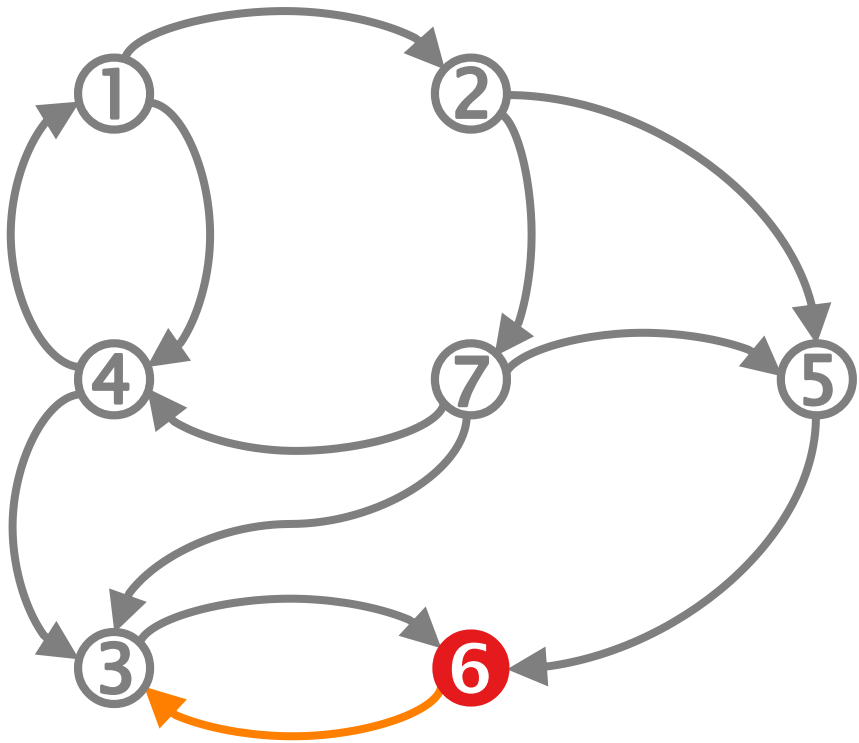


$$q' \langle \bar{v} \rangle = q \vee \wedge A$$

BFS - LEVELS

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

level = 4



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

q	①	②	③	④	⑤	⑥	⑦
						●	

level

v	①	②	③	④	⑤	⑥	⑦
	1	2	3	2	3	4	3

X	X	X	X	X	X	X	X
---	---	---	---	---	---	---	---

$$q' \langle \bar{v} \rangle = q \vee \wedge A$$

q' is empty
→ terminate



BFS – LEVELS: ALGORITHM

- **Input:** adjacency matrix \mathbf{A} , source vertex s , #vertices n
- **Output:** vector of visited vertices \mathbf{v} (integer)
- **Workspace:** queue vector \mathbf{q} (Boolean)

1. $\mathbf{q}(s) = \text{T}$
2. for $level = 1$ to $n - 1$ *terminate earlier if \mathbf{q} is empty
3. $\mathbf{v}(\mathbf{q}) = level$ only change the values under the mask
4. $\text{clear}(\mathbf{q})$ set all elements of \mathbf{q} to F
5. $\mathbf{q}(\bar{\mathbf{v}}) = \mathbf{q} \vee .\wedge \mathbf{A}$ use the lor-land semiring

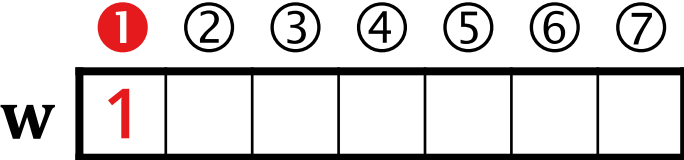
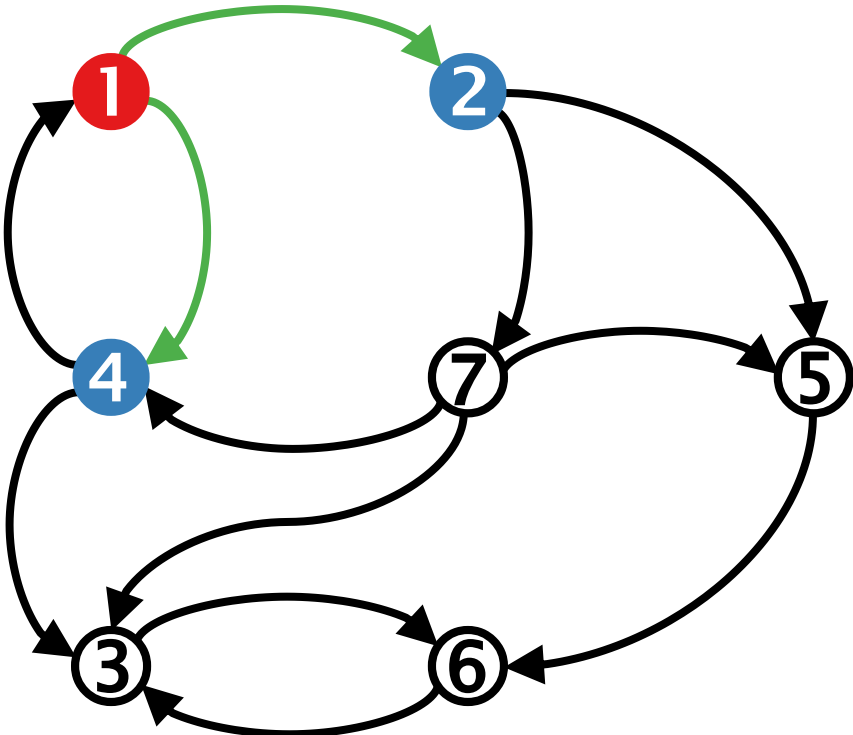
Graph algorithms in GraphBLAS

Breadth-first search / Parents

BFS - PARENTS

semiring	set	\oplus	\otimes	0
min-select 1st	$a \in \mathbb{N} \cup \{+\infty\}$	min	sel1st	$+\infty$

$\text{sel1st}(x, y) = x$

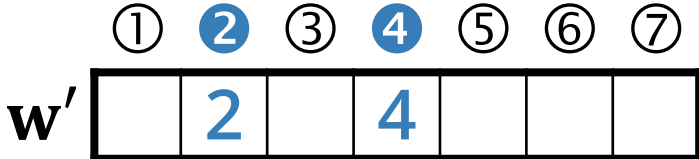
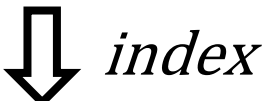


A

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



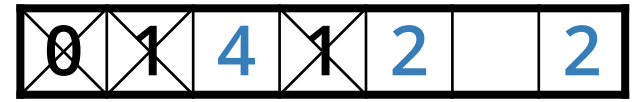
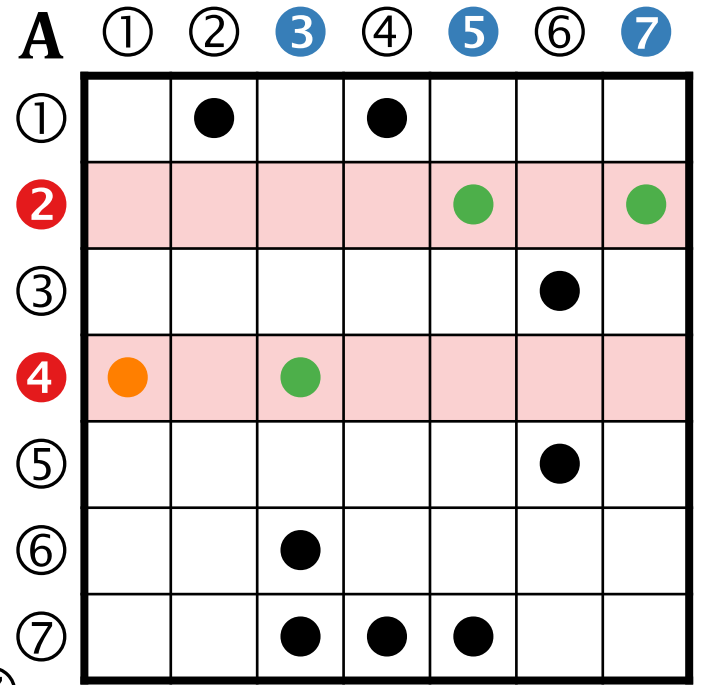
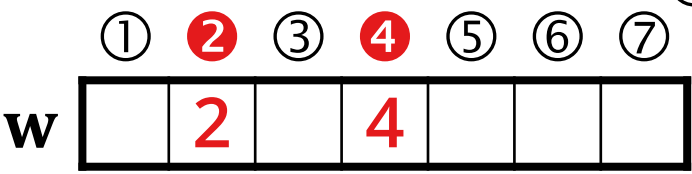
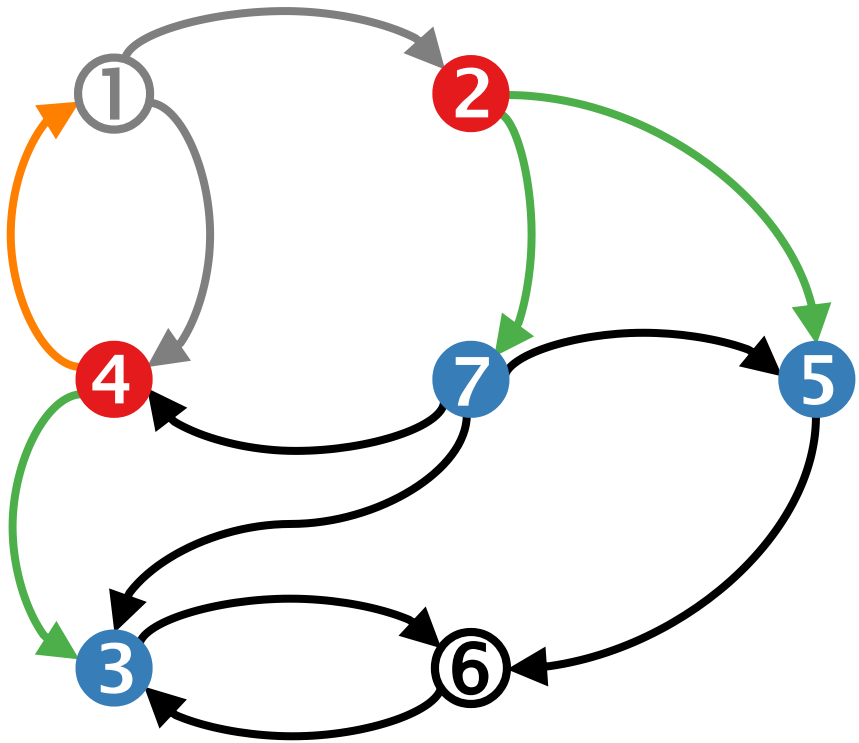
$p\langle \bar{p} \rangle = w \text{ min. sel1st } A$



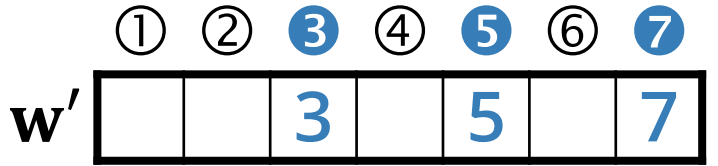
BFS - PARENTS

semiring	set	\oplus	\otimes	0
min-select 1st	$a \in \mathbb{N} \cup \{+\infty\}$	min	sel1st	$+\infty$

$\text{sel1st}(x, y) = x$



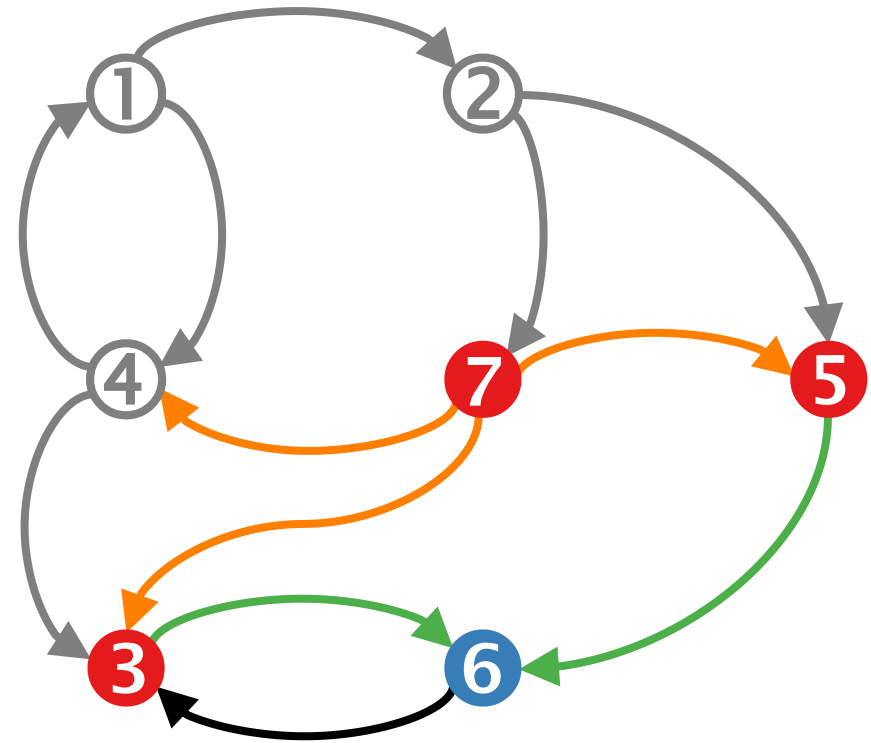
$p(\bar{p}) = w \text{ min. sel1st } A$



BFS - PARENTS

semiring	set	\oplus	\otimes	0
min-select 1st	$a \in \mathbb{N} \cup \{+\infty\}$	min	sel1st	$+\infty$

$\text{sel1st}(x, y) = x$

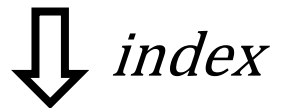


	①	②	③	④	⑤	⑥	⑦
w			3		5		7

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

0	1	4	1	2	3	2
--------------	--------------	--------------	--------------	--------------	---	--------------

$p(\bar{p}) = w \text{ min. sel1st } A$

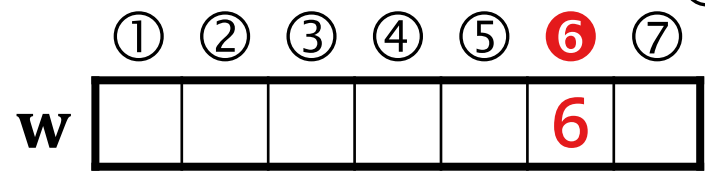
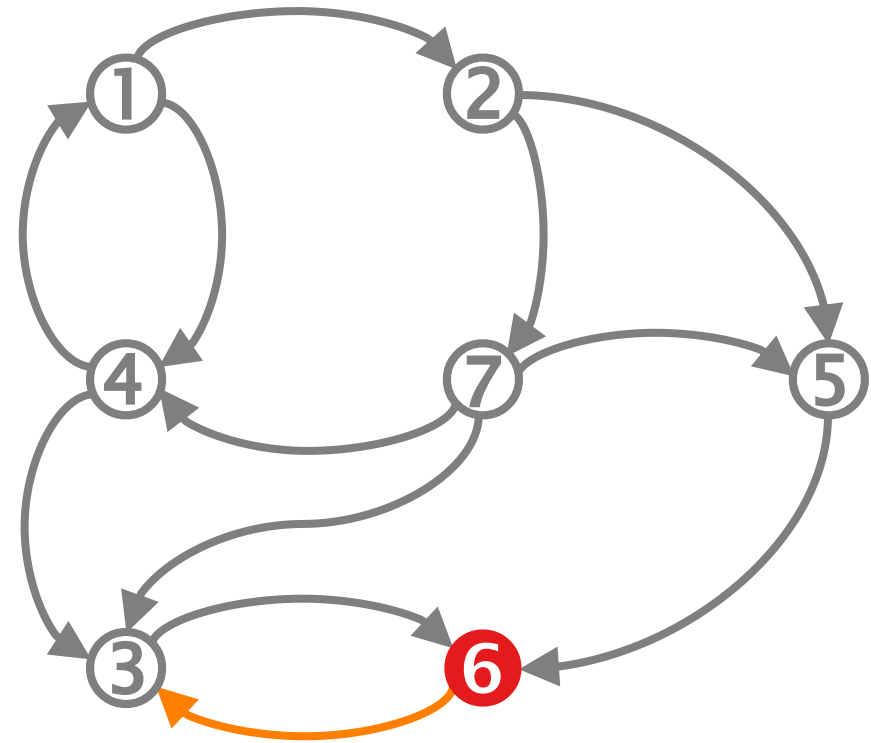


	①	②	③	④	⑤	⑥	⑦
w'						6	

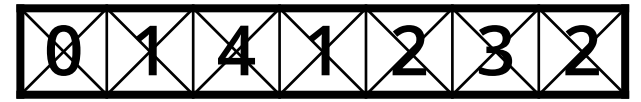
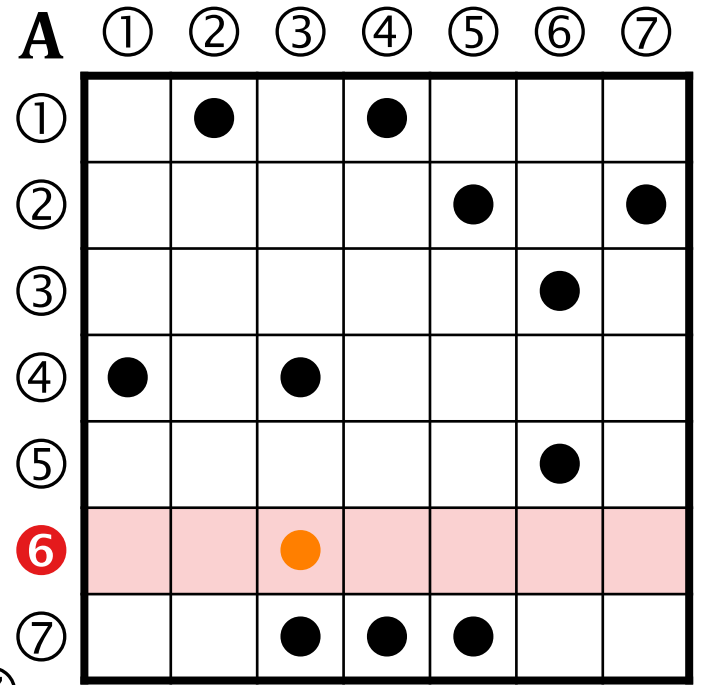
BFS - PARENTS

semiring	set	\oplus	\otimes	0
min-select 1st	$a \in \mathbb{N} \cup \{+\infty\}$	min	sel1st	$+\infty$

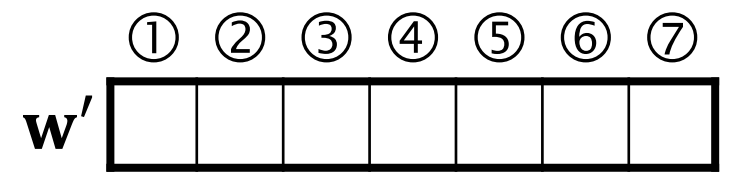
$\text{sel1st}(x, y) = x$



w' is empty
 \rightarrow terminate



$p(\bar{p}) = w \text{ min. sel1st } A$





BFS – PARENTS: ALGORITHM

- **Input:** adjacency matrix \mathbf{A} , source vertex s , #vertices n
- **Output:** parent vertices vector \mathbf{p} (integer)
- **Workspace:** vertex index vector \mathbf{x} (integer), wavefront vector \mathbf{w} (integer), unvisited vertices vector \mathbf{u} (Boolean)

1. $\mathbf{x} = [1 \ 2 \ \dots \ n]$

2. $\mathbf{w}(s) = s$

3. for $l = 1$ to $n - 1$ *terminate earlier if we reach a fixed point

4. $\mathbf{u} = \text{pattern}(\bar{\mathbf{p}})$

5. $\mathbf{p}\langle\mathbf{u}\rangle = \mathbf{w} \text{ min. sel1st } \mathbf{A}$

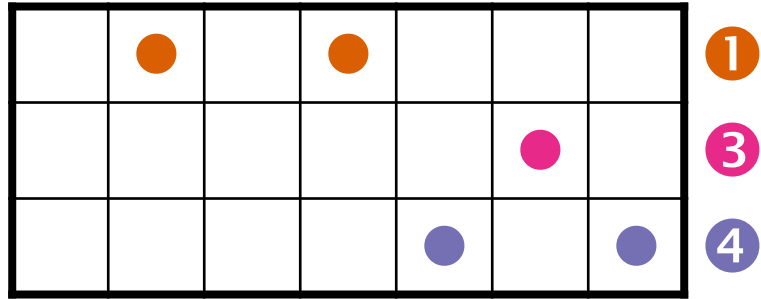
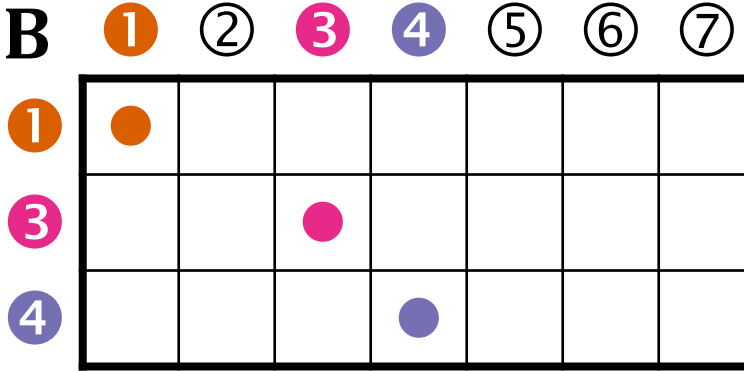
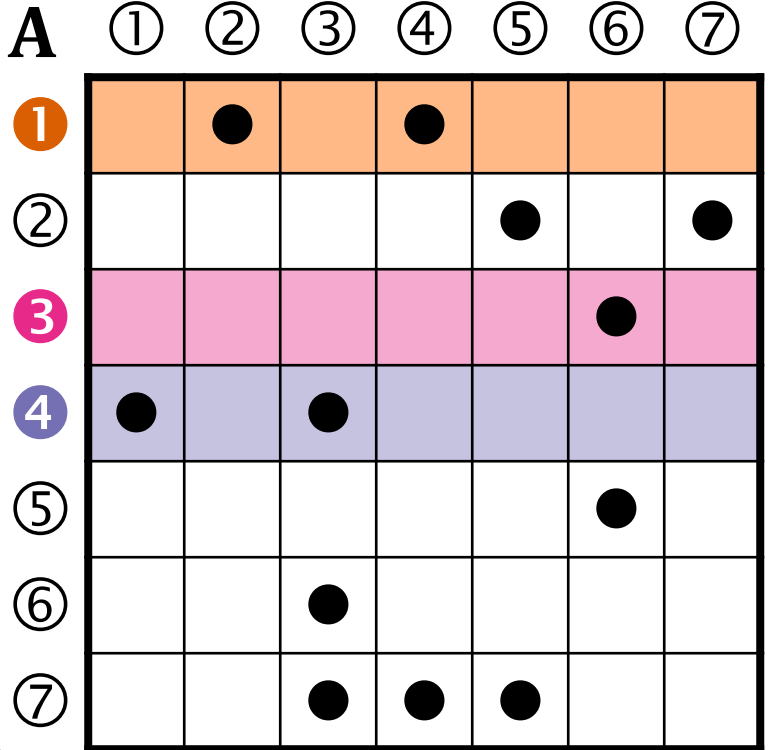
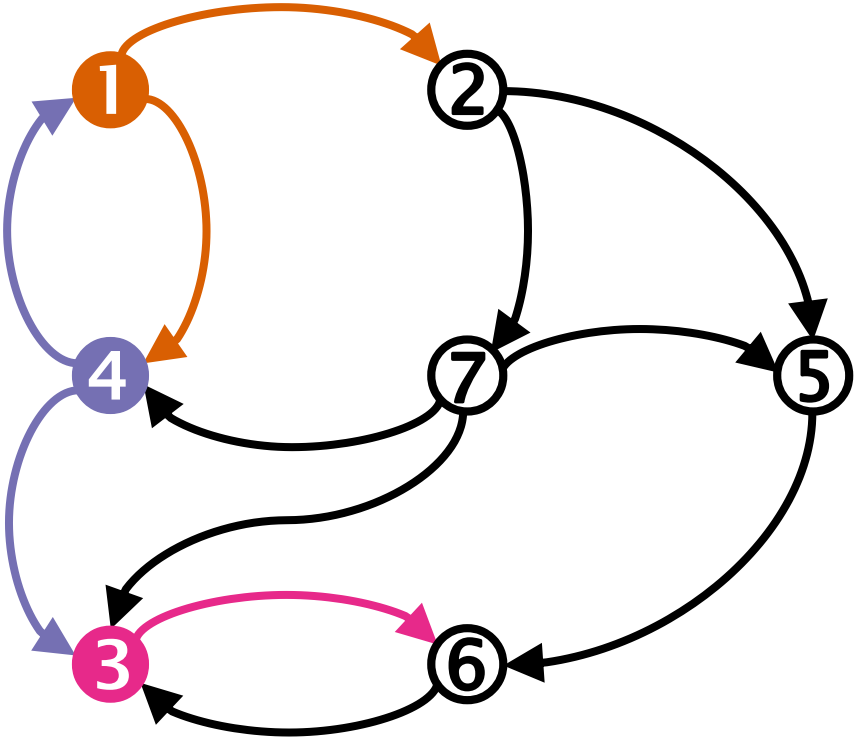
6. $\mathbf{w}\langle\mathbf{u}\rangle = \text{pattern}(\mathbf{p}) \otimes \mathbf{x}$

Graph algorithms in GraphBLAS

Multi-source BFS

MULTI-SOURCE BFS – LEVELS

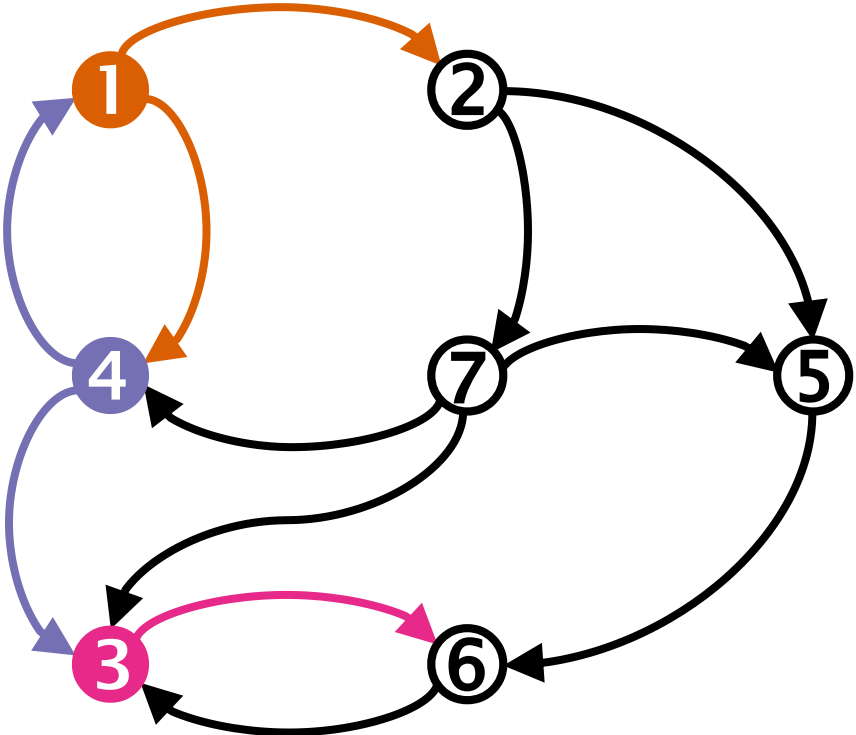
semiring	set	\oplus	\otimes	0
lor-land	$a \in \{F, T\}$	\vee	\wedge	F



B \vee \wedge **A** \Rightarrow *level ...*

MULTI-SOURCE BFS – PARENTS

semiring	set	\oplus	\otimes	0
min-select 1st	$a \in \mathbb{N} \cup \{+\infty\}$	min	sel1st	$+\infty$



A

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

B

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

	1		1					①
						3		③
					4		4	④

B min. sel1st A

BFS – PERFORMANCE

- Naïve BFS impls are sometimes slow on real graphs with skewed distributions – further optimizations are needed.
- Direction-optimizing BFS was published in 2012.
 - Switches between push (\mathbf{vA}) and pull ($\mathbf{A}^T\mathbf{v}$) during execution:
 - Use the push direction when the frontier is small
 - Use the pull direction when the frontier becomes large
 - Adopted to GraphBLAS in 2018



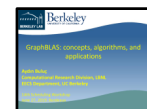
S. Beamer, K. Asanovic, D. Patterson:
Direction-Optimizing Breadth-First Search, SC 2012



C. Yang: *High-performance linear algebra-based graph framework on the GPU*, PhD thesis, UC Davis, 2019



C. Yang, A. Buluç, J.D. Owens: *Implementing Push-Pull Efficiently in GraphBLAS*, ICPP 2018



A. Buluç: *GraphBLAS: Concepts, algorithms, and applications*, Scheduling Workshop, 2019

Graph algorithms in GraphBLAS

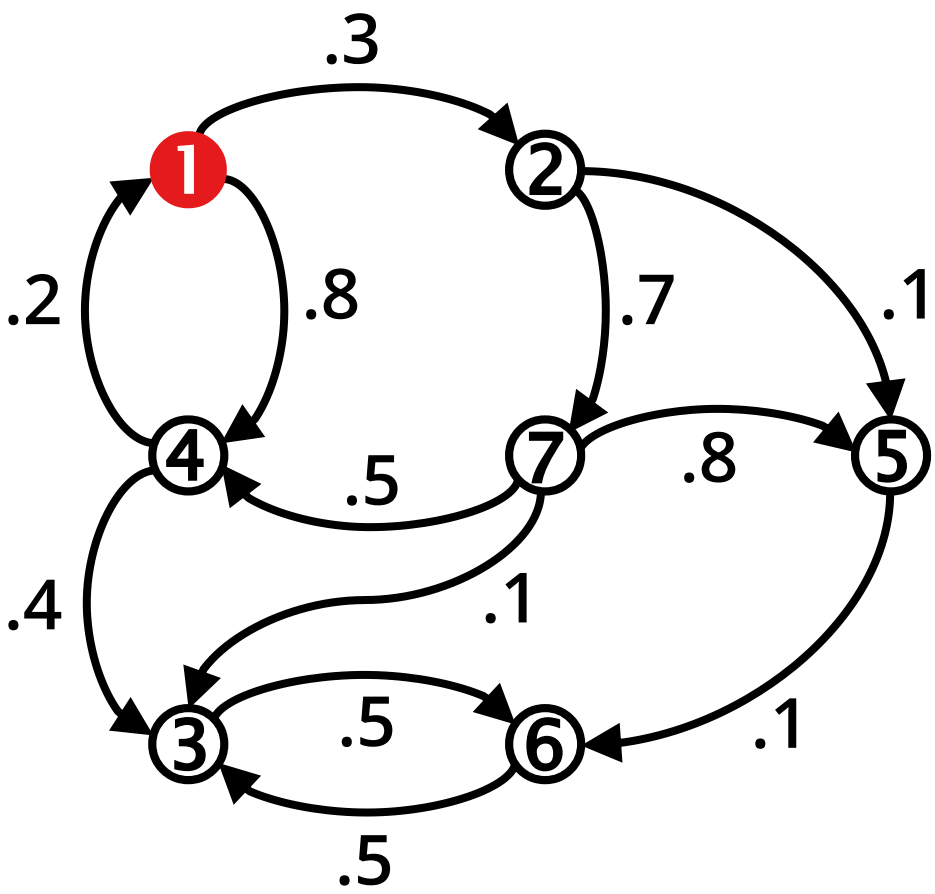
Single-source shortest path

SSSP – SINGLE-SOURCE SHORTEST PATHS

- **Problem:**
 - From a given start vertex s , find the shortest paths to every other (reachable) vertex in the graph
- **Bellman-Ford algorithm:**
 - Relaxes all edges in each step
 - Guaranteed to find the shortest path using at most $n - 1$ steps
- **Observation:**
 - The relaxation step can be captured using a VM multiplication

SSSP – ALGEBRAIC BELLMAN-FORD

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



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

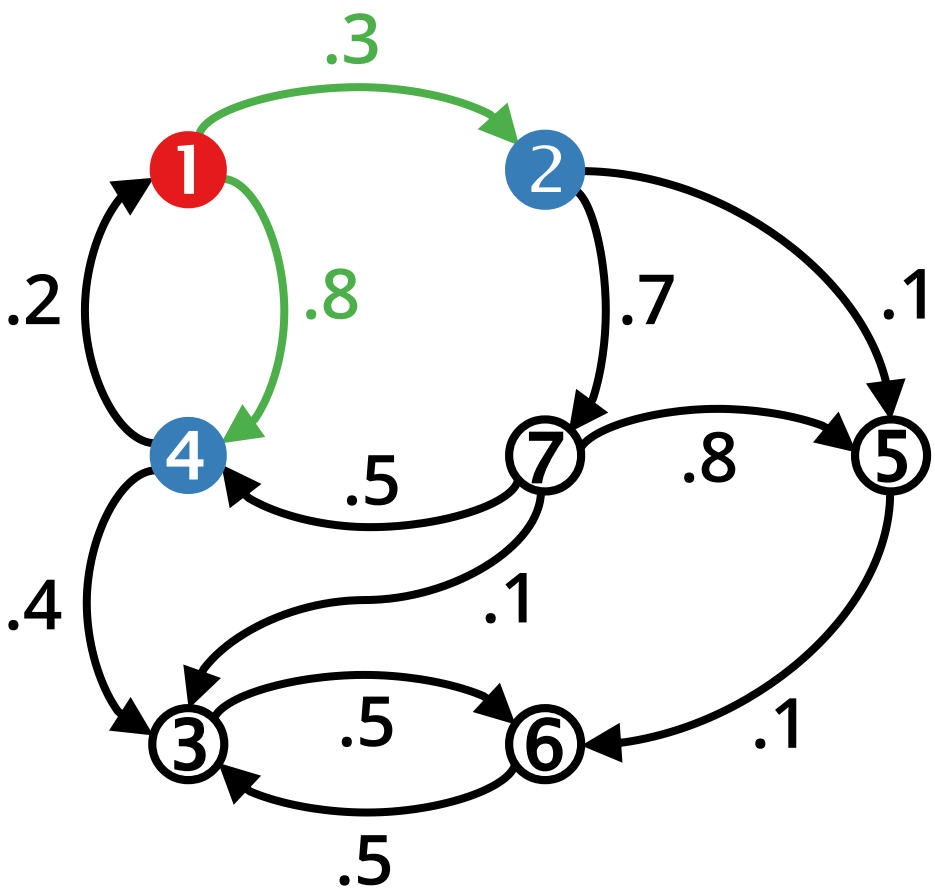
d	①	②	③	④	⑤	⑥	⑦
0							

--	--	--	--	--	--	--	--

d min.+ A

SSSP – ALGEBRAIC BELLMAN-FORD

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



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

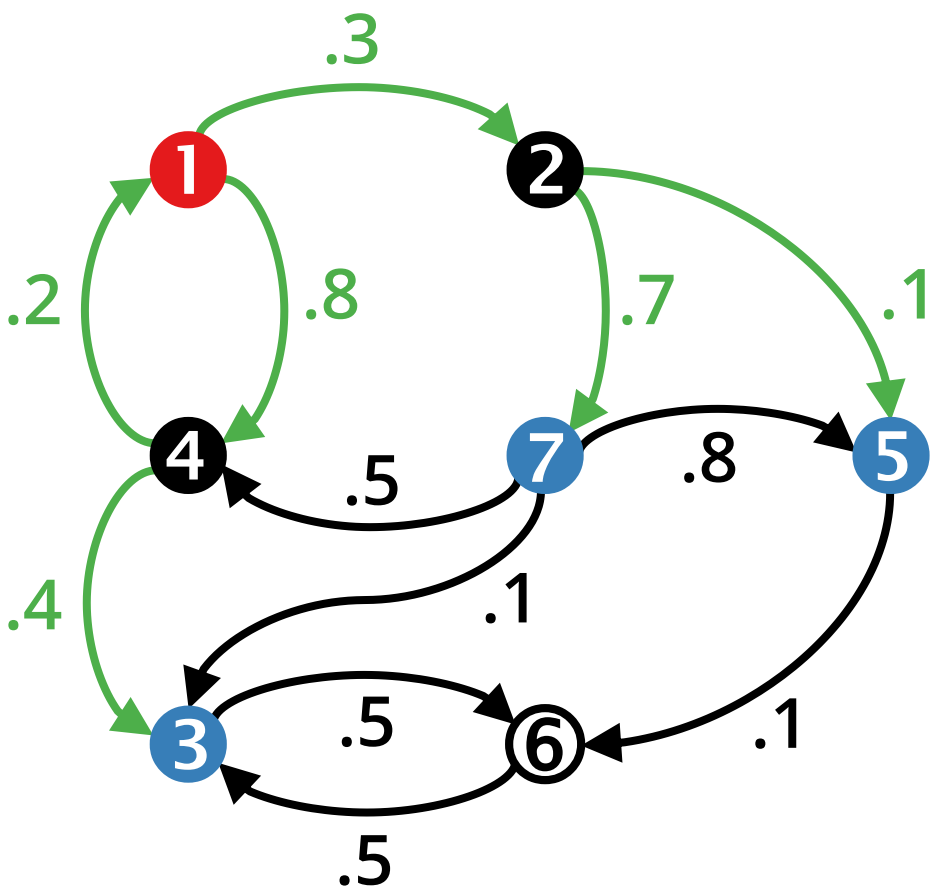
d	①	②	③	④	⑤	⑥	⑦
	0						

0	.3		.8				
---	----	--	----	--	--	--	--

d min.+ A

SSSP – ALGEBRAIC BELLMAN-FORD

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



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

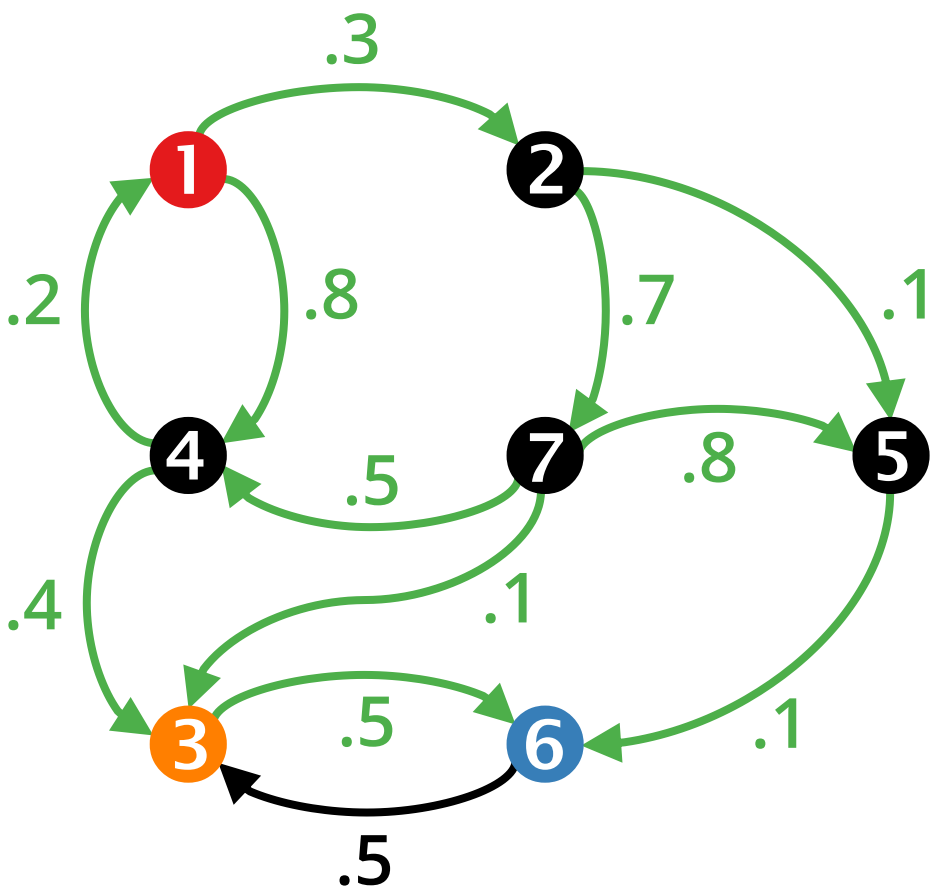
d	①	②	③	④	⑤	⑥	⑦
	0	.3		.8			

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	+	$+\infty$



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

d

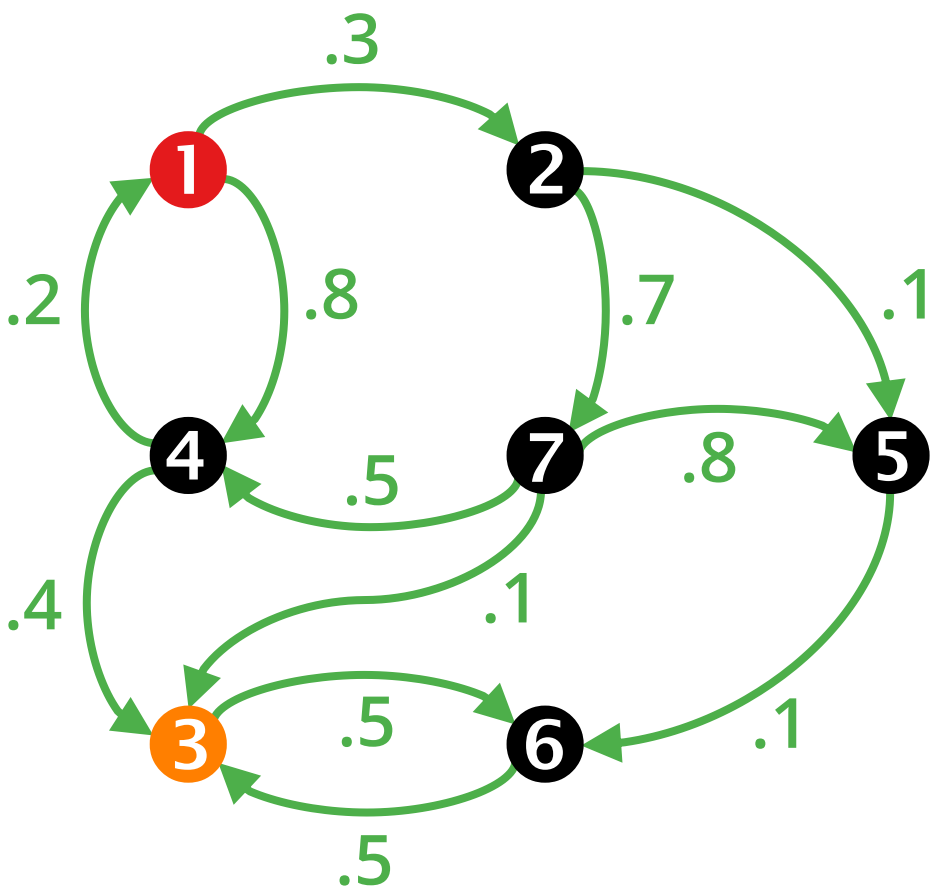
	①	②	③	④	⑤	⑥	⑦
d	0	.3	1.2	.8	.4		1

0	.3	1.1	.8	.4	.5	1
---	----	-----	----	----	----	---

d min.+ A

SSSP – ALGEBRAIC BELLMAN-FORD

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



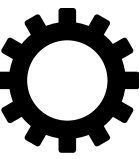
A	①	②	③	④	⑤	⑥	⑦
①	0	.3		.8			
②		0			.1		.7
③			0			.5	
④	.2		.4	0			
⑤					0	.1	
⑥			.5			0	
⑦			.1	.5	.9		0

d	①	②	③	④	⑤	⑥	⑦
	0	.3	1.1	.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 vertex s , #vertices 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} (real)

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

Optimizations for BFS (push/pull) also work here.

Graph algorithms in GraphBLAS

PageRank

PAGERANK – DEFINITION (AS IN GRAPHALYTICS)

We use one of the simpler definitions. For $k = 1$ to t iterations:

$$\text{PR}_0(v) = \frac{1}{n}, \quad \mathbf{d}(v) = \begin{cases} 1 & \text{if } \text{deg}(v) \geq 1 \\ 0 & \text{otherwise} \end{cases}$$

$$\text{PR}_k(v) = \underbrace{\frac{1 - \alpha}{n}}_{\text{teleport}} + \underbrace{\alpha \cdot \sum_{u \in N_{\text{in}}(v)} \frac{\text{PR}_{k-1}(u)}{|N_{\text{out}}(u)|}}_{\text{influence}} + \underbrace{\frac{\alpha}{n} \cdot \sum_{w \in d} \text{PR}_{k-1}(w)}_{\text{dangling}}$$

α : damping factor; d : dangling vertices, $d = \{w \in V \mid |N_{\text{out}}| = 0\}$

There are dozens of PR definitions, some treat dangling vertices differently.

PAGERANK – IN LINEAR ALGEBRA

Initially:

$$\mathbf{pr}_0 = [1 \ 1 \ \dots \ 1] \oslash n, \quad \mathbf{d} = [v_j \ \mathbf{A}(:, j)]$$

In each iteration:

$$\text{PR}_k(v) = \frac{1 - \alpha}{n} + \alpha \cdot \sum_{u \in N_{\text{in}}(v)} \frac{\text{PR}_{k-1}(u)}{|N_{\text{out}}(u)|} + \frac{\alpha}{n} \cdot \sum_{w \in d} \text{PR}_{k-1}(w)$$

$$\mathbf{pr}_k = \underbrace{\frac{1 - \alpha}{n}}_{\text{constant}} \oplus \underbrace{\alpha \otimes \left(\frac{\mathbf{pr}_{k-1}}{\mathbf{d}} \right) \oplus \cdot \otimes \mathbf{A}}_{\text{SpMV}} \oplus \underbrace{\frac{\alpha}{n} \otimes [\oplus_i (\mathbf{pr}_k \otimes \mathbf{d})_i]}_{\text{element-wise sparse vector-dense vector multiplication}}$$



PAGERANK – ALGORITHM

Input: adjacency matrix \mathbf{A} , damping factor α , #iterations t , #vertices n

Output: PageRank \mathbf{pr} (real)

Workspace: dangling vertices \mathbf{d} (Boolean)

1. $\mathbf{pr} = [1 \ 1 \ \dots \ 1] \oslash n$

2. $\mathbf{d} = [\vee_j \mathbf{A}(:, j)]$

3. for $k = 1$ to t

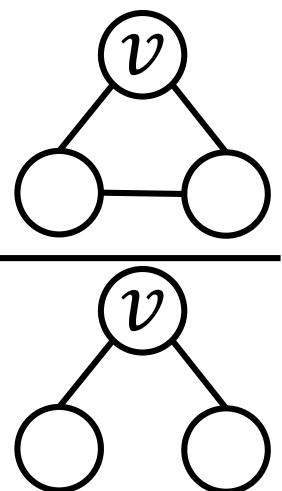
4.
$$\mathbf{pr} = \frac{1 - \alpha}{n} \oplus \alpha \otimes \left(\frac{\mathbf{pr}}{\mathbf{d}} \right) \oplus \cdot \otimes \mathbf{A} \oplus \frac{\alpha}{n} \otimes [\oplus_i (\mathbf{pr} \otimes \mathbf{d})_i]$$

PageRank variant using Markov models are more difficult to express.

Graph algorithms in GraphBLAS

Local clustering coefficient

LCC: LOCAL CLUSTERING COEFFICIENT

$$\text{LCC}(v) = \frac{\text{\#edges between neighbours of } v}{\text{\#possible edges between neighbours of } v} = \frac{\text{tri}(v)}{\text{wed}(v)}$$


If $|N(v)| \leq 1$, $\text{LCC}(v) = 0$

Important metric in social network analysis.

The numerator is the number of *triangles* in v , $\text{tri}(v)$.

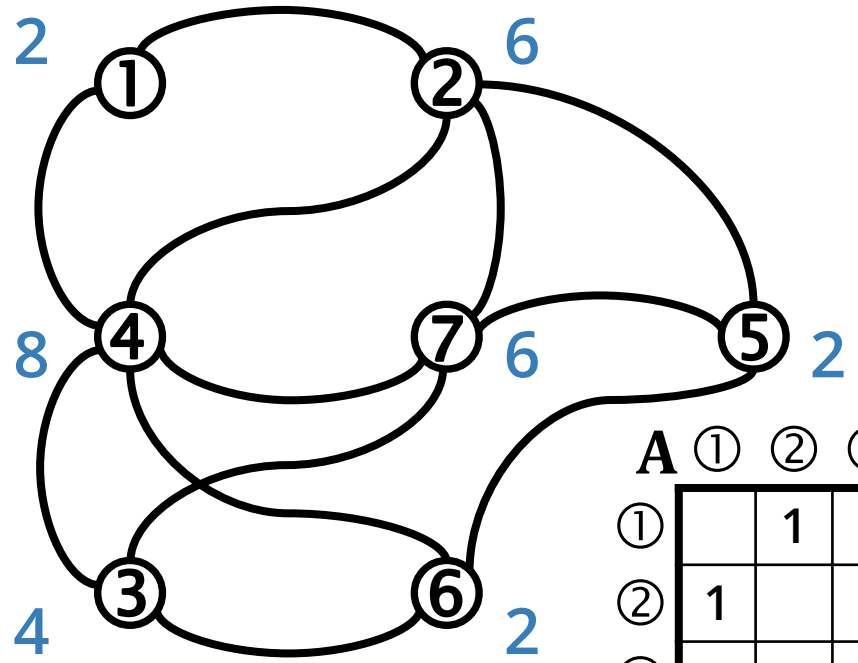
The denominator is the number of *wedges* in v , $\text{wed}(v)$.

$$\text{LCC}(v) = \frac{\text{tri}(v)}{\text{wed}(v)}$$

The difficult part is $\text{tri}(v)$.

LCC EXAMPLE: NAÏVE APPROACH

$$\text{tri} = \text{diag}^{-1}(A \oplus \cdot \otimes A \oplus \cdot \otimes A)$$



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

LCC: NUMBER OF TRIANGLES IN EACH VERTEX

Observation: Matrix $A \oplus \cdot \otimes A \oplus \cdot \otimes A$ is not sparse.

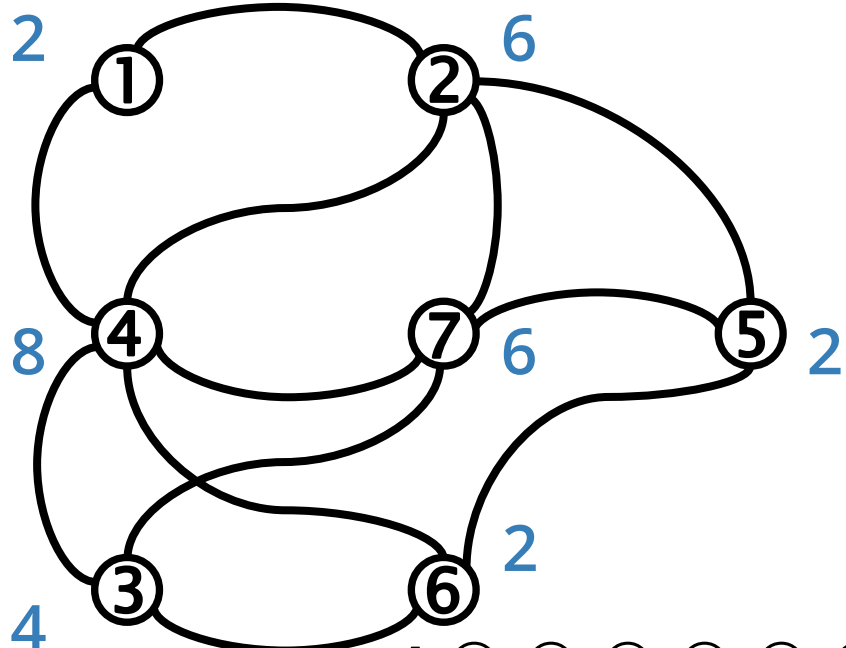
Optimization: Use element-wise multiplication \otimes to close wedges into triangles:

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

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

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

LCC EXAMPLE: ELEMENT-WISE MULTIPLICATION



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		

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

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

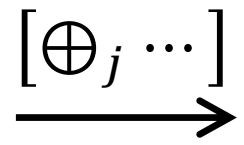
A

	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		

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

TRI

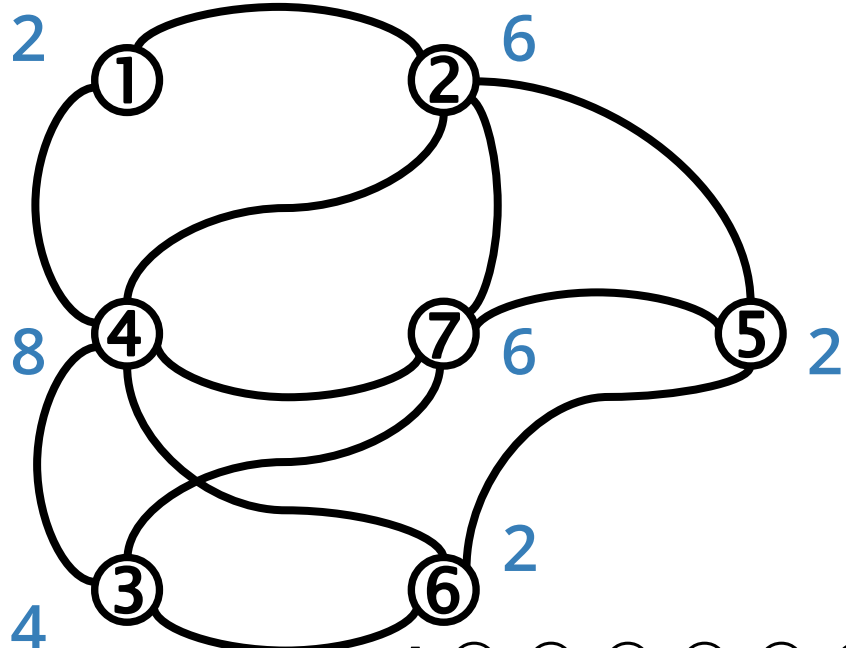
	1		1				
1			2	1			2
			2		1	1	1
1	2	2			1	2	2
	1					1	1
		1	1				
	2	1	2	1			



tri

2
6
4
8
2
2
6

LCC EXAMPLE: ELEMENT-WISE MULTIPLICATION



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			

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

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

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

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		

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

$\left[\oplus_j \dots \right]$
 \longrightarrow

tri

2
6
4
8
2
2
6

LCC: NUMBER OF WEDGES IN EACH VERTEX

$$\text{LCC}(v) = \frac{\text{tri}(v)}{\text{wed}(v)}$$

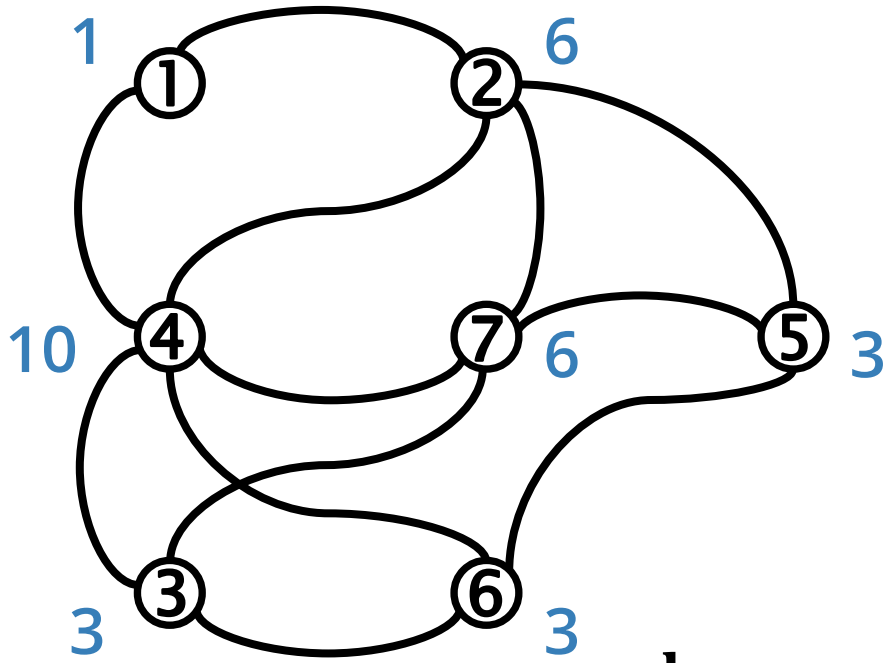
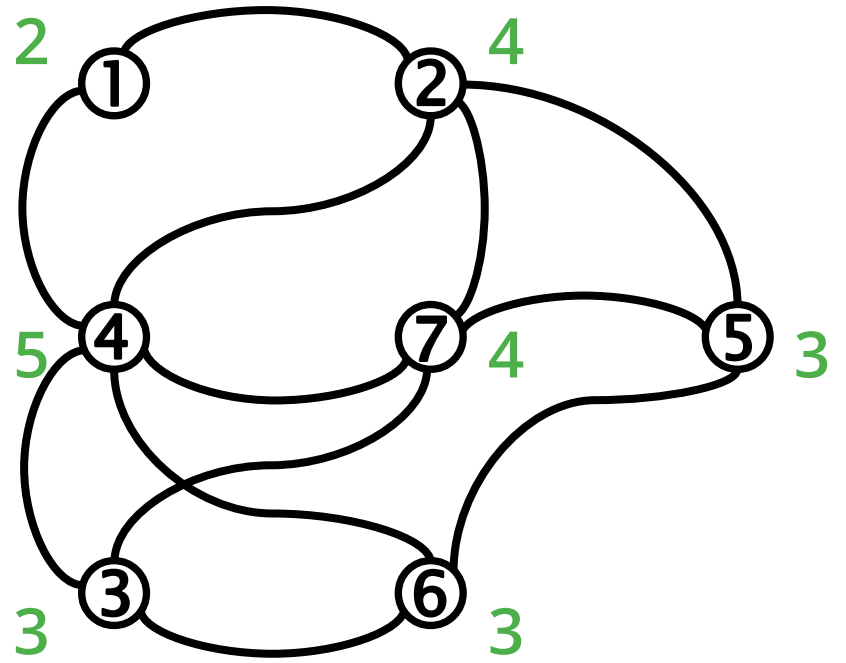
- For $\text{wed}(v)$, we determine the #wedges for each vertex as the 2-permutation of its degree:

$$\text{perm2}(x) = x \cdot (x - 1)$$

- Given the degrees $\mathbf{deg} = [\oplus_j \mathbf{A}(:, j)]$, we compute \mathbf{wed} by applying a unary function on the elements of the vector:

$$\mathbf{wed} = \text{perm2}(\mathbf{deg})$$

LCC EXAMPLE: NUMBER OF WEDGES



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		

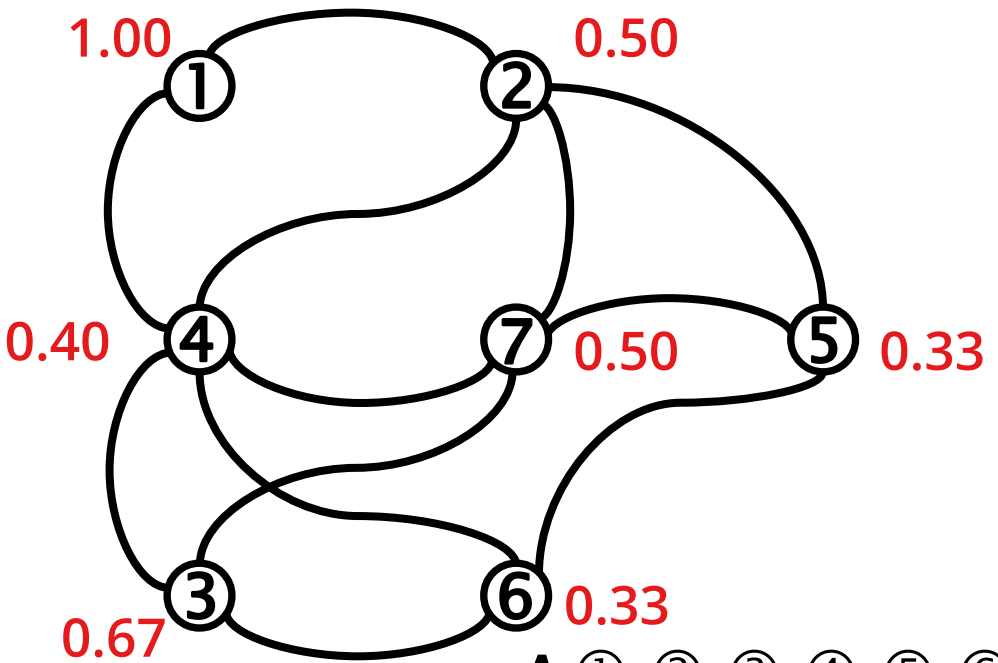
$$\xrightarrow{[\oplus_j A(:,j)]}$$

deg
2
4
3
5
3
3
4

$$\xrightarrow{\text{perm2}(\dots)}$$

wed
2
12
6
20
6
6
12

LCC EXAMPLE: COMPLETE ALGORITHM



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		

$TRI\langle A \rangle = A \oplus . \otimes A$

deg

2
4
3
5
3
3
4

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

perm2(...)

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		

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

tri	2	wed	2	lcc	1.00
	6		12		0.50
	4		6		0.67
	8	\oslash	20	=	0.40
	2		6		0.33
	2		6		0.33
	6		12		0.50



LCC: ALGORITHM

Input: adjacency matrix **A**

Output: vector **lcc**

Workspace: matrix **TRI**, vectors **deg**, **lcc**, and **wed**

1. **TRI** \langle **A** $\rangle = \mathbf{A} \oplus . \otimes \mathbf{A}$ triangle count matrix
2. **tri** = $[\oplus_j \mathbf{TRI}(:, j)]$ triangle count vector
3. **deg** = $[\oplus_j \mathbf{A}(:, j)]$ vertex degree vector
4. **wed** = perm2(**deg**) wedge count vector
5. **lcc** = **tri** \oslash **wed** LCC vector



M. Aznaveh, J. Chen, T.A. Davis, B. Hegyi, S.P. Kolodziej, T.G. Mattson, G. Szárnyas:
Parallel GraphBLAS with OpenMP, Workshop on Combinatorial Scientific Computing 2020

LCC: FURTHER OPTIMIZATIONS

Further optimization: use \mathbf{L} , the lower triangular part of \mathbf{A} .

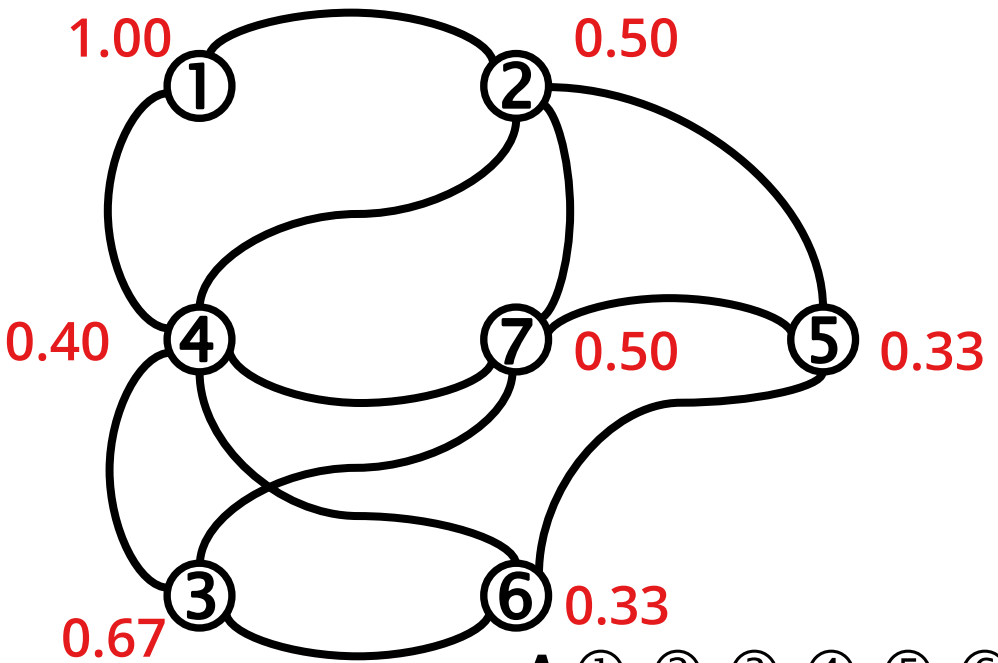
$$\mathbf{TRI}\langle\mathbf{A}\rangle = \mathbf{A} \oplus \cdot \otimes \mathbf{L}$$

The number of wedges is now the 2-combination of **deg**.

$$\text{comb2}(x) = \frac{x \cdot (x - 1)}{2}$$

Permuting the adjacency matrix allows further optimizations.

LCC EXAMPLE: LOWER TRIANGULAR PART OF MX.



L

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

$TRI\langle A \rangle = A \oplus . \otimes L$

deg

2
4
3
5
3
3
4

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

comb2(...)

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	1			
				2			
1	1	2					
	1						
		1					
			1				
	2	1					

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

tri

1
3
2
4
1
1
3

wed

1
6
3
10
3
3
6

\ominus

lcc

1.00
0.50
0.67
0.40
0.33
0.33
0.50

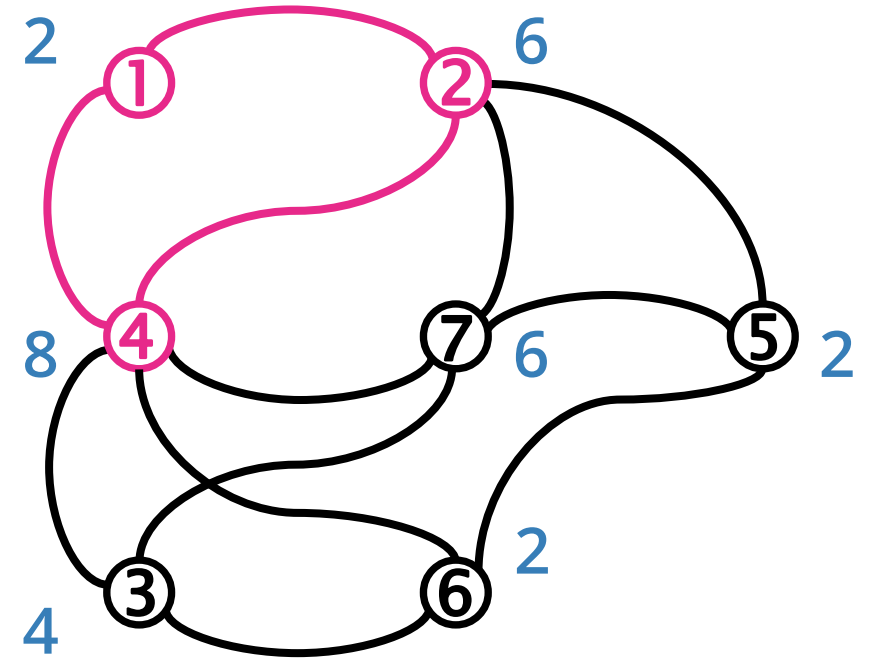
=

Graph algorithms in GraphBLAS

Triangle count / Definition

TRIANGLE COUNT

- IEEE GraphChallenge: an annual competition at the HPEC conference
- The task of the 2017 GraphChallenge was **triangle count**: given a graph G , count the number of triangles.
- **Triangle** = “*set of three mutually adjacent vertices in a graph*”
- Many solutions employed a linear algebraic computation model



Number of unique triangles:

$$\frac{30}{6}$$



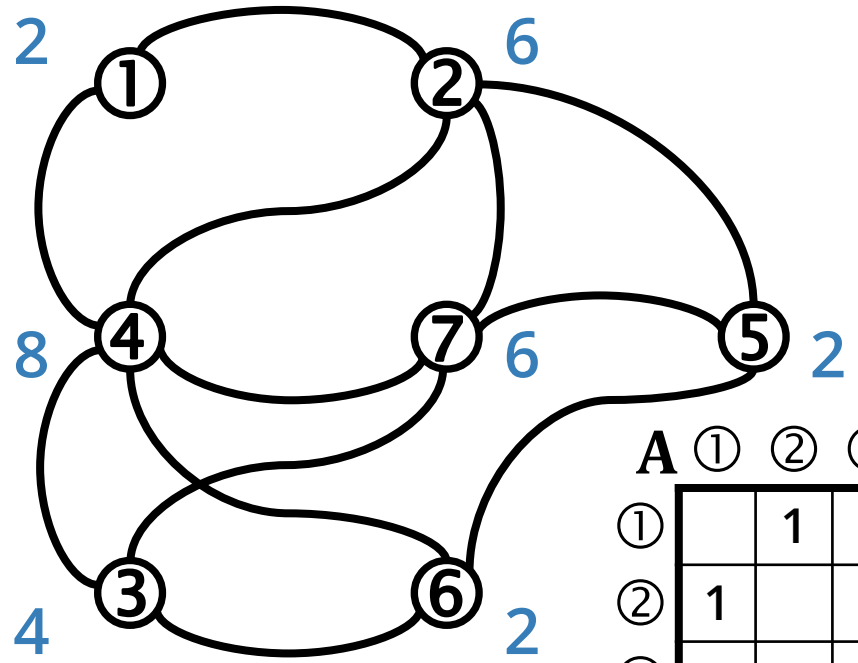
GraphChallenge.org: Raising the Bar on Graph Analytic Performance, HPEC 2018

Graph algorithms in GraphBLAS

Triangle count / Naïve algorithm

LCC EXAMPLE: NAÏVE APPROACH

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



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

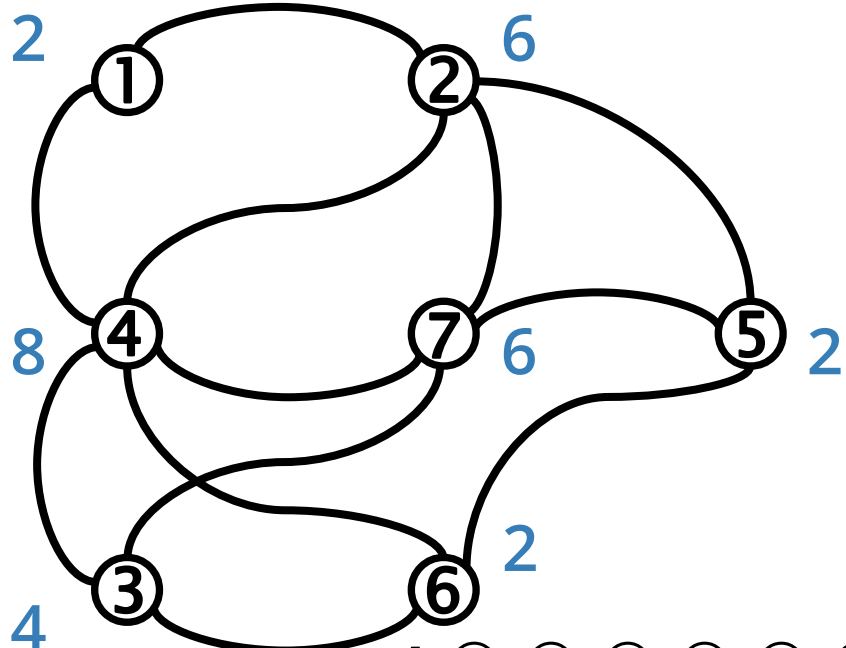
2	6	4	7	4	3	4
6	6	6	11	8	5	9
4	6	4	8	4	7	9
7	11	8	8	5	10	12
4	8	4	5	2	8	9
3	5	7	10	8	2	4
4	9	9	12	9	4	6

tri
2
6
4
8
2
2
6

Graph algorithms in GraphBLAS

Triangle count / Masked algorithm

LCC EXAMPLE: ELEMENT-WISE MULTIPLICATION



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		

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

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

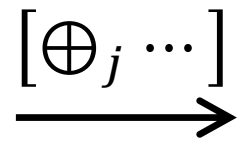
A

	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		

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

TRI

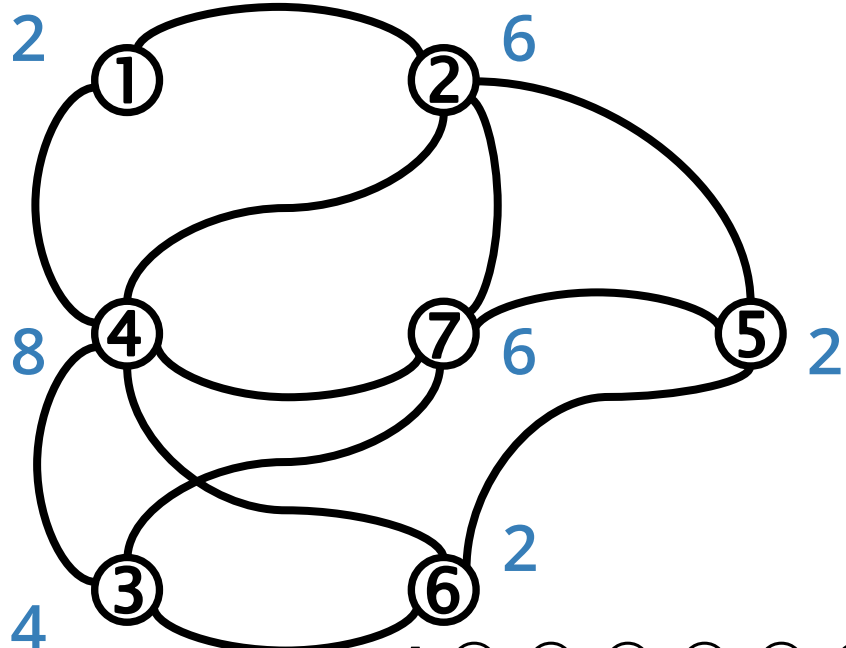
	1		1				
1			2	1			2
			2		1	1	
1	2	2			1	2	
	1					1	
		1	1				
	2	1	2	1			



tri

2
6
4
8
2
2
6

LCC EXAMPLE: ELEMENT-WISE MULTIPLICATION



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			

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

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

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

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		

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

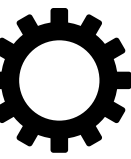
$\left[\oplus_j \dots \right]$

tri

2
6
4
8
2
2
6

Graph algorithms in GraphBLAS

Triangle count / Cohen's algorithm



COHEN'S ALGORITHM: PSEUDOCODE

Input: adjacency matrix **A**

Output: triangle count **t**

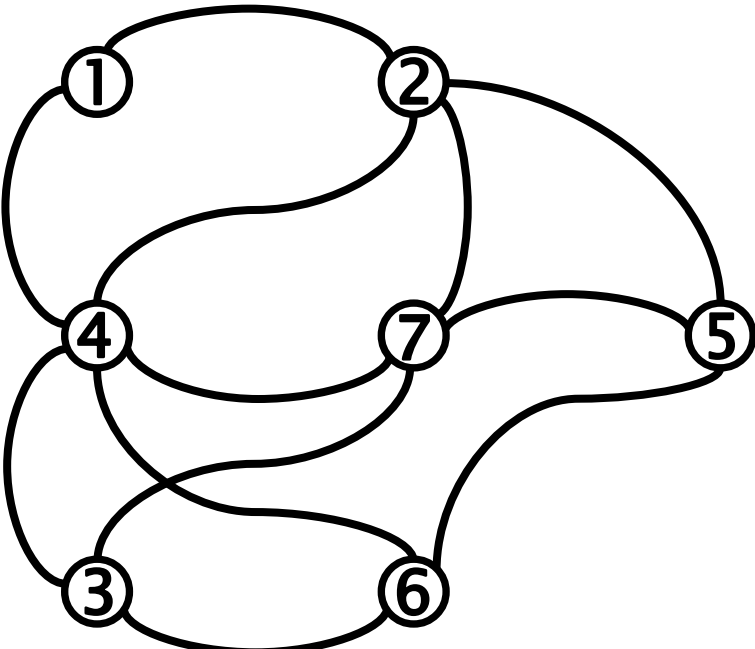
Workspace: matrices **L, U, B, C**

1. $\mathbf{L} = \text{tril}(\mathbf{A})$ extract the lower triangle from A
2. $\mathbf{U} = \text{triu}(\mathbf{A})$ extract the upper triangle from A
3. $\mathbf{B} = \mathbf{L} \oplus \otimes \mathbf{U}$ multiply matrices L and U
4. $\mathbf{C} = \mathbf{B} \otimes \mathbf{A}$ element-wise multiplication
5. $t = \sum \mathbf{C} / 2$ sum the values in C and divide by 2



J. Cohen: *Graph Twiddling in a MapReduce World*, Comput. Sci. Eng. 2009

COHEN'S ALGORITHM



U

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

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		

⊗

L

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

B

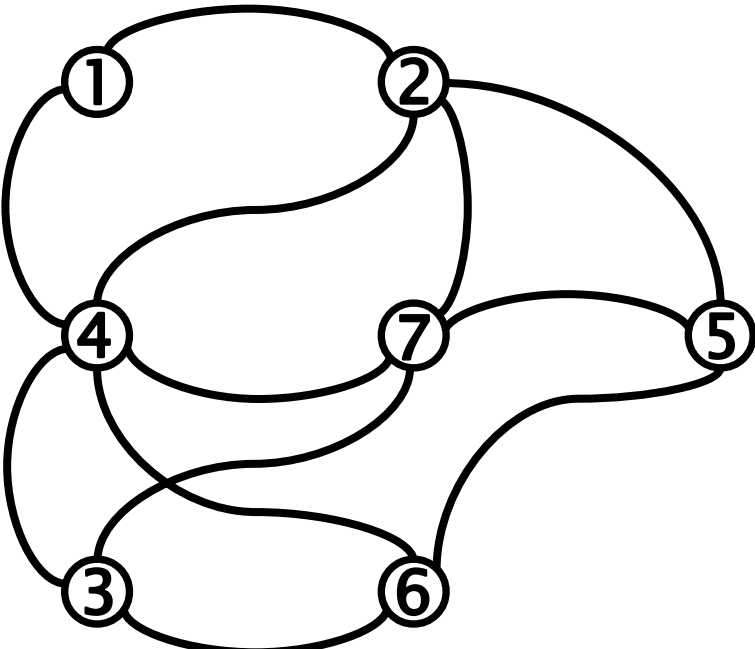
	①	②	③	④	⑤	⑥	⑦
①							
②		1		1			
③							
④		1		3	1	1	2
⑤				1	1		1
⑥				1		3	3
⑦				2	1	3	4

C

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

$$t = \sum C / 2$$

COHEN'S ALGORITHM: MASKING



U

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

L

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

C

			1				
		1				1	2
							1
				1			
				2	1		

$$C\langle A \rangle = L \oplus . \otimes U$$

$$t = \sum C / 2$$

Graph algorithms in GraphBLAS

Triangle count / Sandia algorithm



SANDIA ALGORITHM

Input: adjacency matrix **A**

Output: triangle count ***t***

Workspace: matrices **L, U, B, C**

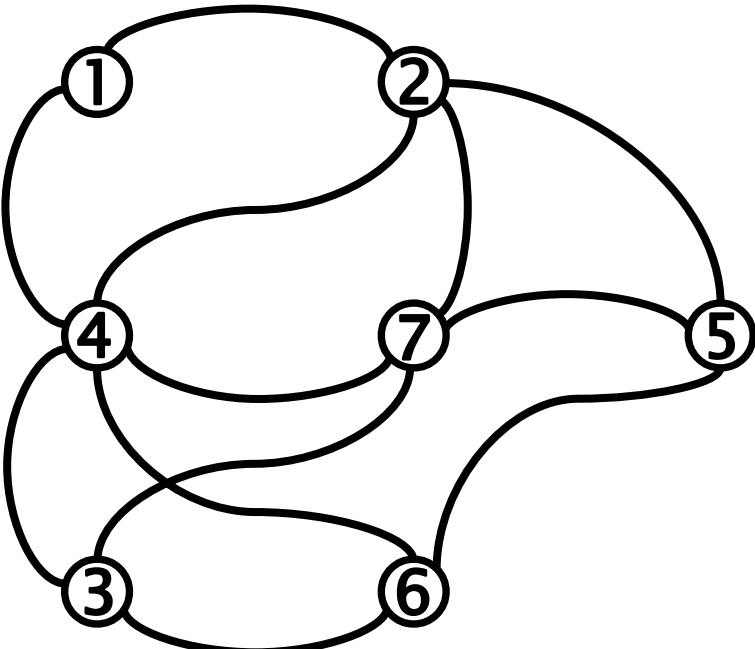
1. $\mathbf{L} = \text{tril}(\mathbf{A})$ extract the lower triangle from A
2. $\mathbf{C}(\mathbf{L}) = \mathbf{L} \oplus \cdot \otimes \mathbf{L}$ multiply matrices L and L using mask L
3. $t = \sum \mathbf{C}$ sum the values in C



M.M. Wolf et al. (Sandia National Laboratories):

Fast linear algebra-based triangle counting with KokkosKernels, HPEC 2017

SANDIA ALGORITHM



L

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

L

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

C

	1						
			1				
		2	1				

$$C\langle L \rangle = L \oplus . \otimes L$$

$$t = \sum C$$

Graph algorithms in GraphBLAS

Triangle count / CMU algorithm

CMU ALGORITHM

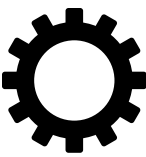
- Iterates on the vertices of the graph, extracts corresponding submatrices and computes $t = t + a_{10}^T \oplus \cdot \otimes A_{20} \oplus \cdot \otimes a_{12}$
- Tradeoffs:
 - does not require mxm, only vxm and mxv
 - slower than mxm-based algorithms

A	<i>i</i>	
	A_{00}	a_{01}
<i>i</i>	a_{01}^T	0
	A_{20}	a_{21}
		A_{22}

The formula is derived using the matrix trace $\text{tr}(\mathbf{A}) = \sum_{i=0}^{n-1} \mathbf{A}_{ii}$ and its invariant property under cyclic permutation, e.g. $\text{tr}(\mathbf{ABC}) = \text{tr}(\mathbf{BCA}) = \text{tr}(\mathbf{CAB})$. See the paper for details.



T.M. Low et al. (Carnegie Mellon University):
First look: linear algebra-based triangle counting without matrix multiplication, HPEC 2017



CMU ALGORITHM: PSEUDOCODE

Input: adjacency matrix \mathbf{A}

Output: triangle count t

Workspace: matrix \mathbf{A}_{20} , \mathbf{a}_{10} , \mathbf{a}_{12}^\top , \mathbf{C}

1. for $i = 2$ to $n - 1$
2. $\mathbf{A}_{20} = \mathbf{A}[i + 1:n, 0:i - 1]$
3. $\mathbf{a}_{10} = \mathbf{A}[0:i - 1, i]$
4. $\mathbf{a}_{12} = \mathbf{A}[i, i + 1:n]$
5. $t = t + \mathbf{a}_{10}^\top \oplus \cdot \otimes \mathbf{A}_{20} \oplus \cdot \otimes \mathbf{a}_{12}$

		i	
	\mathbf{A}_{00}	\mathbf{a}_{01}	\mathbf{A}_{20}^\top
i	\mathbf{a}_{01}^\top	0	\mathbf{a}_{21}^\top
	\mathbf{A}_{20}	\mathbf{a}_{21}	\mathbf{A}_{22}

T.M. Low et al. (Carnegie Mellon University):

First look: linear algebra-based triangle counting without matrix multiplication, HPEC 2017



PROVABLY CORRECT ALGORITHMS

The “CMU algorithm” belongs to a family of algorithms which can be derived using the “FLAME approach”.

There are 8 similar algorithms in total, the one presented here is Algorithm 2.

Algorithm: $\Delta := \frac{1}{6}\Gamma(\tilde{A}^3)$

$$A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{TR}^T & A_{BR} \end{array} \right)$$

where A_{TL} is a 0×0 matrix
 while $m(A_{TL}) < m(A)$ do

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{TR}^T & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{01}^T & \alpha_{11} & a_{12}^T \\ \hline A_{02}^T & a_{12} & A_{22} \end{array} \right)$$

where α_{11} is a 1×1 matrix

Algorithm 1	Algorithm 2
$\Delta := \Delta + \frac{1}{2}a_{01}^T A_{00}a_{01}$	$\Delta := \Delta + a_{01}^T A_{02}a_{21}$
Algorithm 3	Algorithm 4
$\Delta := \Delta + \frac{1}{2}a_{01}^T A_{00}a_{01}$ $\Delta := \Delta + \frac{1}{2}a_{12}^T A_{22}a_{12}$ $\Delta := \Delta - a_{01}^T A_{02}a_{21}$	$\Delta := \Delta + \frac{1}{2}a_{12}^T A_{22}a_{12}$

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{TR}^T & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{01}^T & \alpha_{11} & a_{12}^T \\ \hline A_{02}^T & a_{12} & A_{22} \end{array} \right)$$

endwhile

Algorithm: $t := \frac{1}{6}\Gamma(A^3)$

$$A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{TR}^T & A_{BR} \end{array} \right)$$

where A_{BR} is a 0×0 matrix
 while $m(A_{TL}) < m(A)$ do

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{TR}^T & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{01}^T & \alpha_{11} & a_{12}^T \\ \hline A_{02}^T & a_{12} & A_{22} \end{array} \right)$$

where α_{11} is a 1×1 matrix

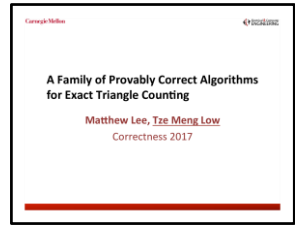
Algorithm 5	Algorithm 6
$\Delta := \Delta + \frac{1}{2}a_{01}^T A_{00}a_{01}$	$\Delta := \Delta + a_{01}^T A_{02}a_{21}$
Algorithm 7	Algorithm 8
$\Delta := \Delta + \frac{1}{2}a_{01}^T A_{00}a_{01}$ $\Delta := \Delta + \frac{1}{2}a_{12}^T A_{22}a_{12}$ $\Delta := \Delta - a_{01}^T A_{02}a_{21}$	$\Delta := \Delta + \frac{1}{2}a_{12}^T A_{22}a_{12}$

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{TR}^T & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{01}^T & \alpha_{11} & a_{12}^T \\ \hline A_{02}^T & a_{12} & A_{22} \end{array} \right)$$

endwhile



M. Lee, T.M. Low (Carnegie Mellon University):
A family of provably correct algorithms for exact triangle counting,
 CORRECTNESS @ SC 2017



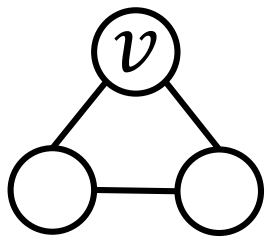
Source of the figure

Graph algorithms in GraphBLAS

Node-wise triangle count

NODE-WISE TRIANGLE COUNT

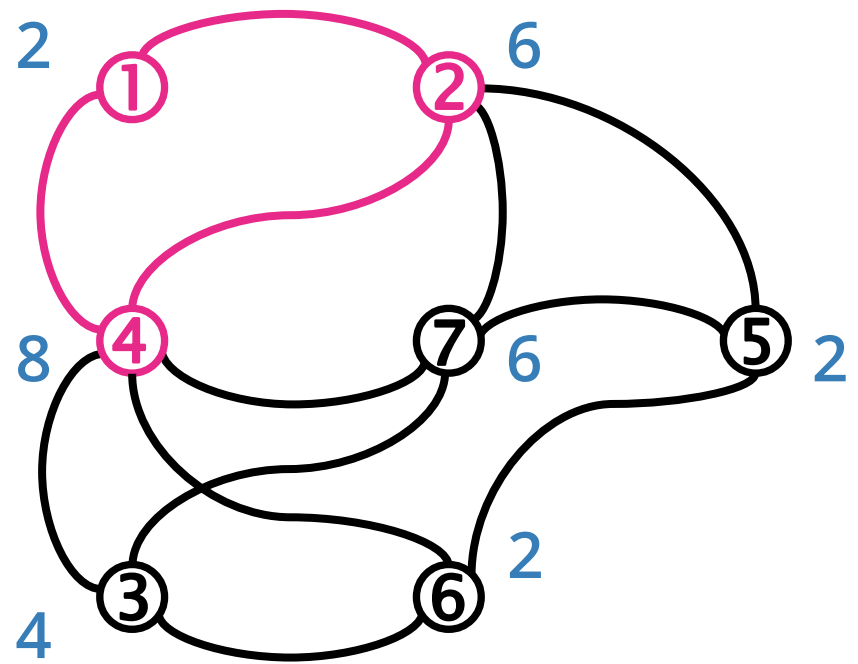
Triangle - Def 1: a set of three mutually adjacent nodes.



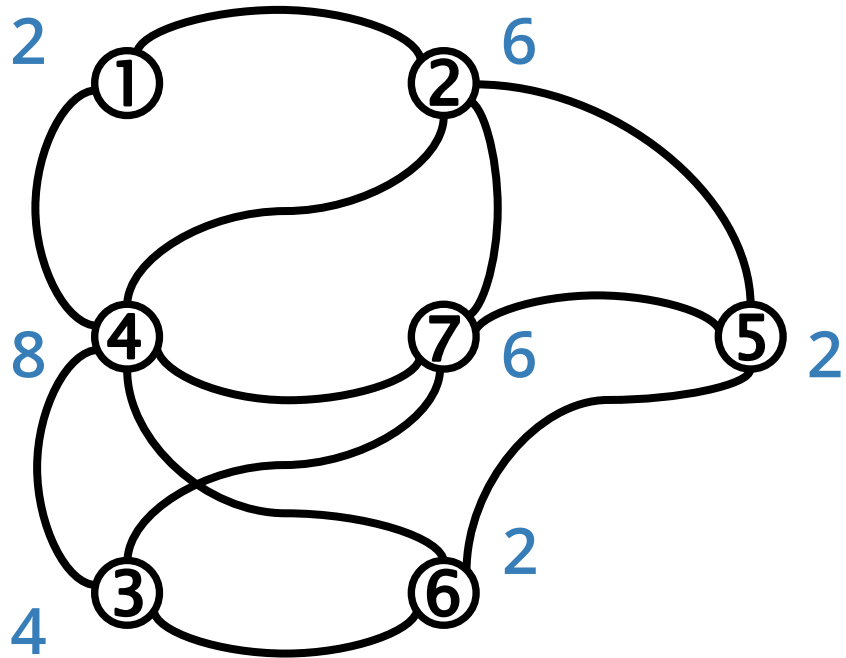
Def 2: a three-length closed path.

Usages:

- Global clustering coefficient
- Local clustering coefficient
- Finding communities



TC: ELEMENT-WISE MULTIPLICATION



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		

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

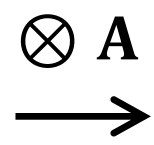
$$\mathbf{tri} = \left[\oplus_j \mathbf{TRI}(:, j) \right]$$

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

A

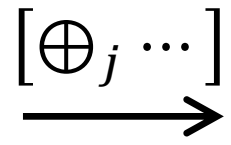
	①	②	③	④	⑤	⑥	⑦
①		1		1			
②	1			1	1		1
③				1		1	1
④	1	1	1			1	1
⑤		1				1	1
⑥			1	1	1		
⑦		1	1	1	1		

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



TRI

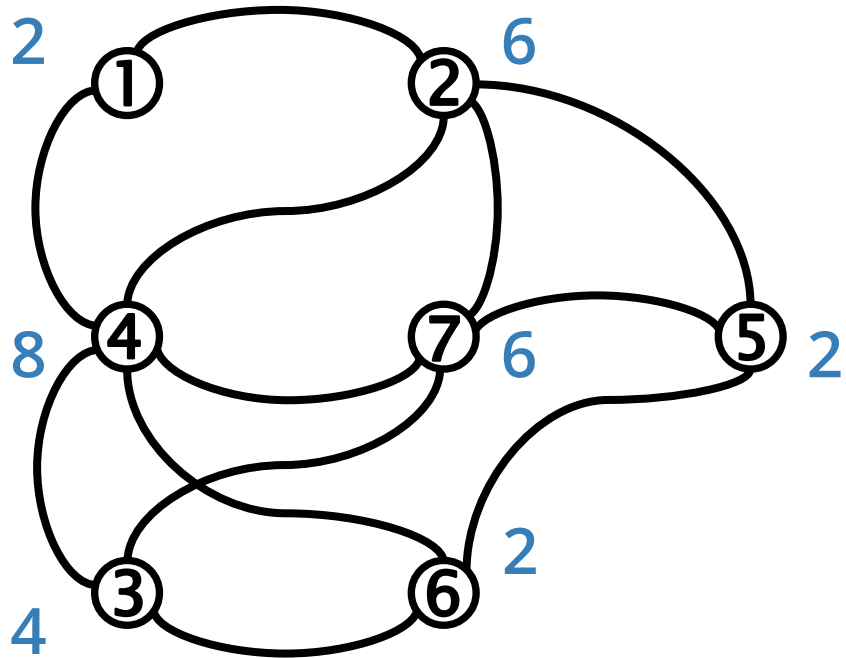
	1		1				
1			2	1		2	
			2		1	1	
1	2	2			1	2	
	1					1	
		1	1				
	2	1	2	1			



tri

2
6
4
8
2
2
6

TC: ELEMENT-WISE MULTIPLICATION



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		

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

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

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

A

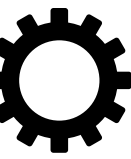
	①	②	③	④	⑤	⑥	⑦
①		1		1			
②	1			1	1		1
③				1		1	1
④	1	1	1			1	1
⑤		1				1	1
⑥			1	1	1		
⑦		1	1	1	1		

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

$[\oplus_j \dots]$

tri

2
6
4
8
2
2
6



TC: ALGORITHM

Input: adjacency matrix **A**

Output: vector **tri**

Workspace: matrix **TRI**

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

Optimization: use **L**, the lower triangular part of **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

k-truss

K-TRUSS

- **Definition:** the k -truss is a subset of the graph with the same number of vertices, where each edge appears in at least $k - 2$ triangles in the original graph.



K-TRUSS ALGORITHM

- **Input:** adjacency matrix \mathbf{A} , scalar k
- **Output:** k -truss adjacency matrix \mathbf{C}
- **Helper:** $f(x, support) = x \geq support$

1. $\mathbf{C} = \mathbf{A}$
2. for $i = 1$ to $n - 1$
3. $\mathbf{C} \langle \mathbf{C} \rangle = \mathbf{C} \oplus \wedge \mathbf{C}$ use the “plus-and” semiring
4. $\mathbf{C} = f(\mathbf{C}, k - 2)$ drop entries in \mathbf{C} less than $k - 2$
5. terminate if the number of nonzero values in \mathbf{C} did not change



Graph algorithms in GraphBLAS

Community detection using label propagation

CDLP: COMMUNITY DETECTION USING LABEL PROPAGATION

Goal: assign a label to each vertex representing the community it belongs to. The algorithm (originally published in network science) is slightly altered to ensure deterministic execution. Initially:

$$L_0(v) = v$$

In the k^{th} iteration:

$$L_k(v) = \min(\arg \max_l |u \in N(v) \mid L_{k-1}(u) = l|),$$

where $N(v)$ is the set of neighbours of v .

Run for t iterations or until reaching a fixed point.



U.N. Raghavan, R. Albert, S. Kumara: *Near linear time algorithm to detect community structures in large-scale networks*, Phys. Rev. E, 2007

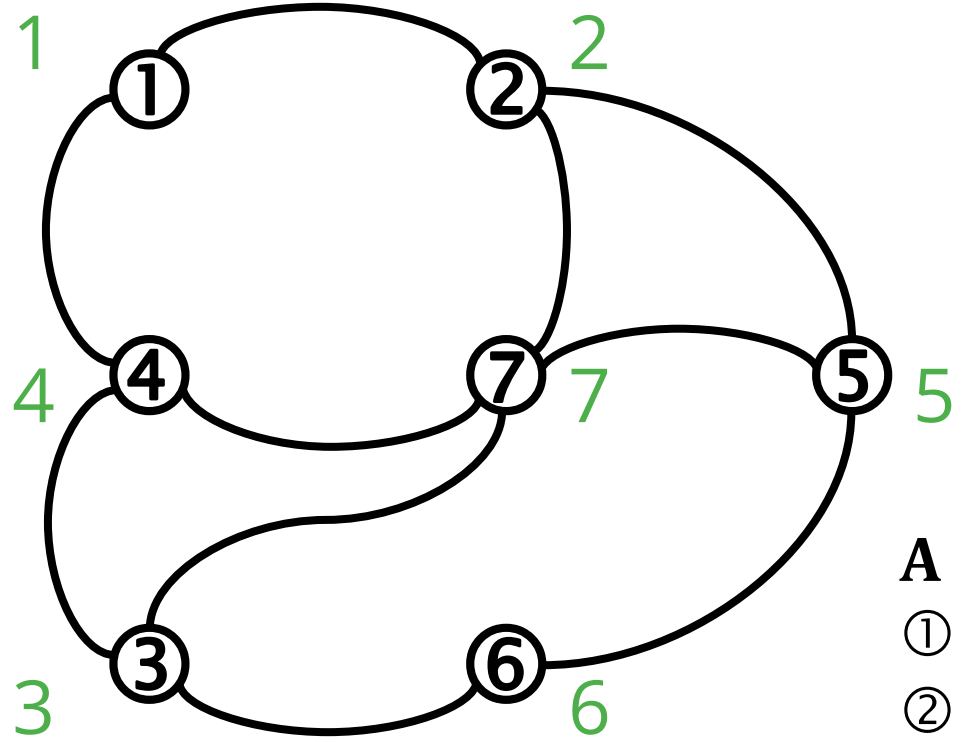
IDEA: CAPTURE CDLP IN PURE GRAPHBLAS

- Define a semiring that operates over **occurrence vectors**
- \oplus operator: combines two occurrence vectors
 - $\{6 \rightarrow 1, 9 \rightarrow 1\} \oplus \{6 \rightarrow 1, 7 \rightarrow 2\} = \{6 \rightarrow 2, 7 \rightarrow 2, 9 \rightarrow 1\}$
- Convert each element in a row to an occurrence vector
 - $\{6 \rightarrow 1\}, \{6 \rightarrow 1\}, \{7 \rightarrow 1\}, \{7 \rightarrow 1\}, \{9 \rightarrow 1\}$
- Reduce each row into a single occurrence vector:
 - $\{6 \rightarrow 2, 7 \rightarrow 2, 9 \rightarrow 1\}$
- Select the min. mode element from the occurrence vector
 - 6
- Work on paper, but occurrence vectors need dynamic memory allocation, which leads to very poor performance

CDLP IN LINEAR ALGEBRA

- Extract each row from \mathbf{F}
 - Easy if the matrix is stored in CSR format
- Select the minimum mode value in each row
 - Sort elements using parallel merge sort
 - Pick the min value that has the longest run (done in a single pass)
- Sort each row \mathbf{r}
- Use the sorted list to compute $\text{mode}(\mathbf{r})$

CDLP EXAMPLE



diag(lab)

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

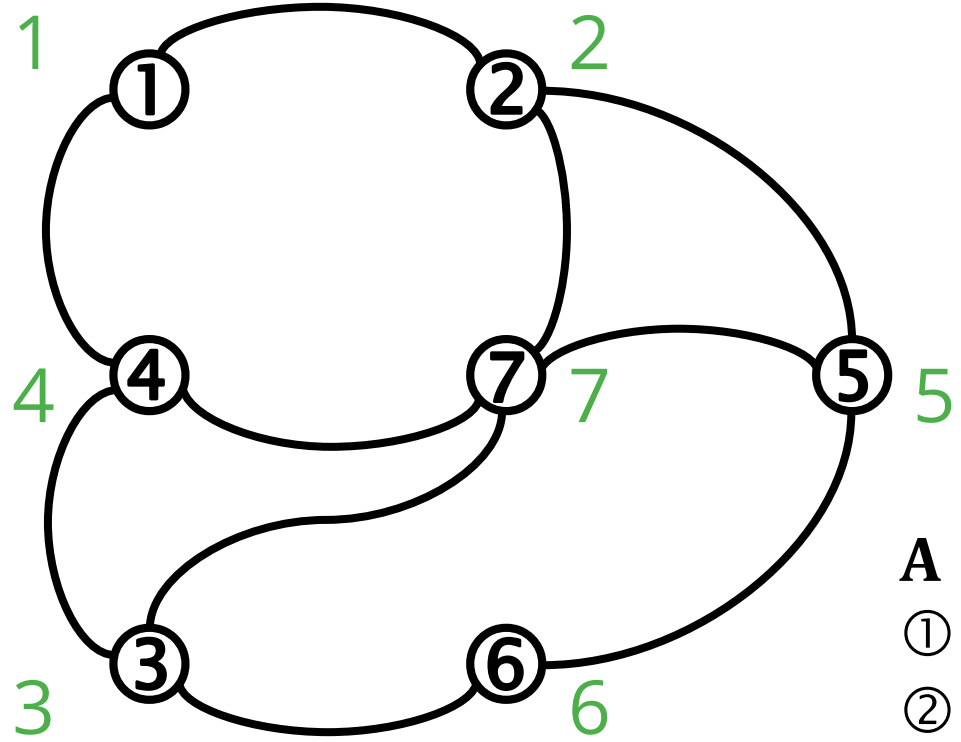
A

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

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

- Initially, $\mathbf{lab} = [1\ 2\ \dots\ n]$
- Propagate labels to create a "frequency matrix":
 $\mathbf{F} = \mathbf{A} \text{ any.sel2nd } \mathbf{diag}(\mathbf{lab})$

CDLP EXAMPLE



step: 1

diag(lab) ① ② ③ ④ ⑤ ⑥ ⑦

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

A ① ② ③ ④ ⑤ ⑥ ⑦

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

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

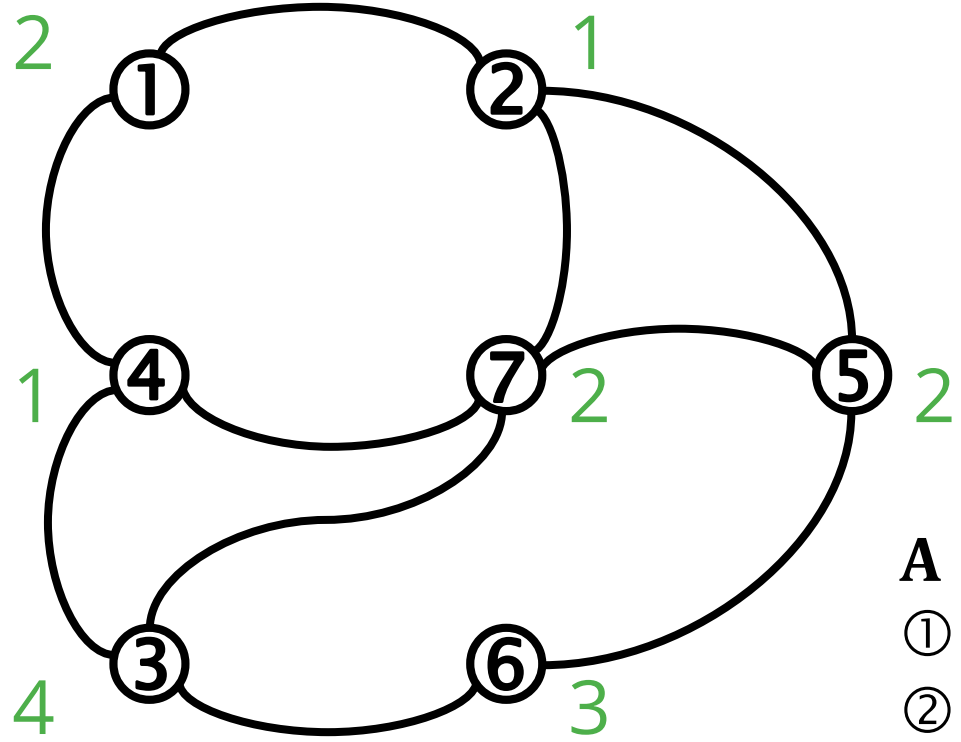
min.
mode



lab'

2
1
4
1
2
3
2

CDLP EXAMPLE



step: 2

diag(lab) ① ② ③ ④ ⑤ ⑥ ⑦

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

A ① ② ③ ④ ⑤ ⑥ ⑦

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

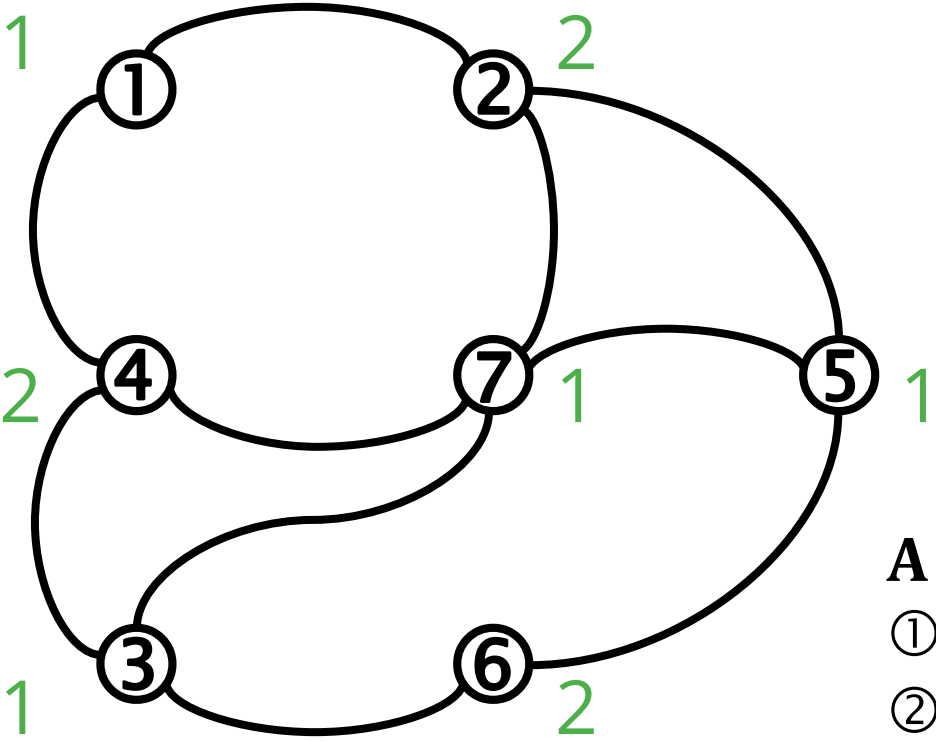
		1		1			
2					2		2
				1		3	2
2		4					2
	1					3	2
			4		2		
	1	4	1	2			

min.
mode
➔

lab'

1
2
1
2
1
2
1

CDLP EXAMPLE



step: 3

diag(lab) ① ② ③ ④ ⑤ ⑥ ⑦

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

A ① ② ③ ④ ⑤ ⑥ ⑦

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

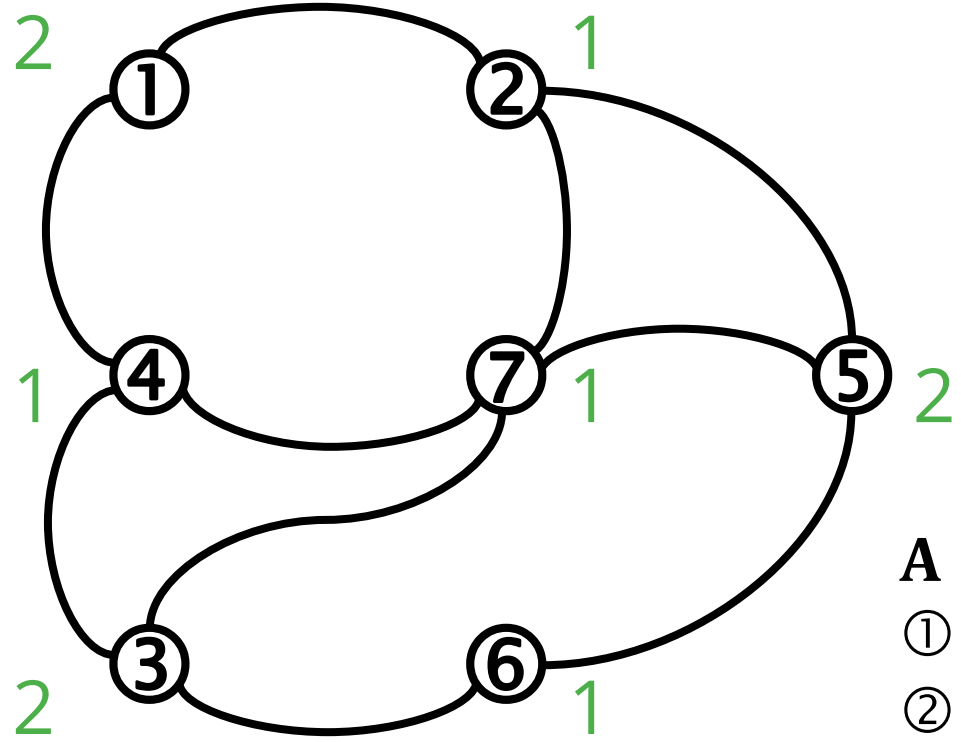
		2		2			
1					1		1
				2		2	1
1		1					1
	2					2	1
			1	1			
	2	1	2	1			

min.
mode
➔

lab'

2
1
2
1
2
1
1

CDLP EXAMPLE



step: 4
same result as in step 2

diag(lab)

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

A

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

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

min.
mode
➔

lab'

1
2
1
2
1
2
1



CDLP: ALGORITHM

Input: adjacency matrix \mathbf{A} , #vertices n , #iterations t

Output: vector \mathbf{lab}

Workspace: matrix \mathbf{F} , vector \mathbf{r}

1. $\mathbf{lab} = [1 \ 2 \ \dots \ n]$
 2. for $k = 1$ to t
 3. $\mathbf{F} = \mathbf{A}$ any.sel2nd diag(\mathbf{lab})
 4. for $i = 1$ to n
 5. $\mathbf{r} = \mathbf{F}(i, :)$
 6. sort(\mathbf{r})
 7. $\mathbf{L}(i) = \text{select_min_mode}(\mathbf{r})$
- } Can be batched and parallelized



CDLP: ALGORITHM

Input: adjacency matrix \mathbf{A} , #vertices n , #iterations t

Output: vector \mathbf{lab}

Workspace: matrix \mathbf{F} , vector \mathbf{r} , array of row indices \mathbf{I} , array of values \mathbf{X}

1. $\mathbf{lab} = [1 \ 2 \ \dots \ n]$
2. for $k = 1$ to t
3. $\mathbf{F} = \mathbf{A}$ any.sel2nd diag(\mathbf{lab})
4. $\langle \mathbf{I}, _ , \mathbf{X} \rangle = \text{extract_tuples}(\mathbf{F})$
5. $\text{merge_sort_pairs}(\langle \mathbf{I}, \mathbf{X} \rangle)$
6. $\mathbf{lab} =$ for each row in \mathbf{I} , select min mode value from \mathbf{X}

CDLP ON DIRECTED GRAPHS

For directed graphs, we compute the labels $L_k(v)$ as:

$$\min(\arg \max_l [|u \in N_{\text{in}}(v) \mid L_{k-1}(u) = l| + |u \in N_{\text{out}}(v) \mid L_{k-1}(u) = l|])$$

- In linear algebra, this can be expressed with two matrices:
 - $\mathbf{F}_{\text{in}} = \mathbf{A}$ any.sel2nd diag(**lab**)
 - $\mathbf{F}_{\text{out}} = \mathbf{A}^\top$ any.sel2nd diag(**lab**)
- Simultaneously iterate over rows \mathbf{r}_{in} of \mathbf{F}_{in} and \mathbf{r}_{out} of \mathbf{F}_{out}
- For each row pair, sort $\mathbf{r}_{\text{in}} \cup \mathbf{r}_{\text{out}}$ and select the minimum mode value
- Batching also works:
 - $\langle \mathbf{I}_{\text{in}}, _ , \mathbf{X}_{\text{in}} \rangle = \text{extract_tuples}(\mathbf{F}_{\text{in}})$
 - $\langle \mathbf{I}_{\text{out}}, _ , \mathbf{X}_{\text{out}} \rangle = \text{extract_tuples}(\mathbf{F}_{\text{out}})$ } merge_sort_pairs($\langle \mathbf{I}_{\text{in}} \cup \mathbf{I}_{\text{out}}, \mathbf{X}_{\text{in}} \cup \mathbf{X}_{\text{out}} \rangle$)

Graph algorithms in GraphBLAS

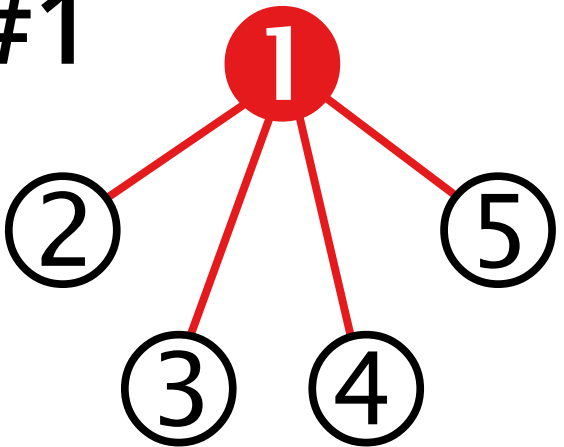
The importance of masking

THE IMPORTANCE OF MASKING

Q: Is masking absolutely necessary?

A: Yes, it can reduce the complexity of some algorithms.
We demonstrate this with two examples.

#1



A ① ② ③ ④ ⑤ A ① ② ③ ④ ⑤

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

	1	1	1	1
1				
1				
1				
1				

A ① ② ③ ④ ⑤

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

4				
	1	1	1	1
	1	1	1	1
	1	1	1	1
	1	1	1	1

A

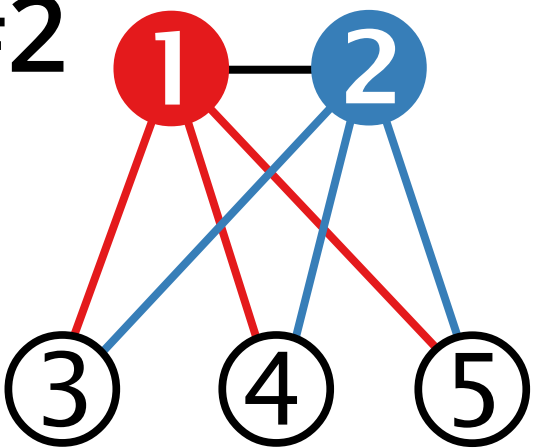
A ⊕ . ⊗ A

A ⊕ . ⊗ A ⊗ A

A simple corner case is the star graph: there are $(n - 1)^2$ wedges but none of them close into triangles.

We do quadratic work while it's clear that there are no triangles in the graph (it's a tree).

#2



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

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

A full bipartite graph $K_{2,3}$ with the vertices in the top partition connected.

A bipartite graph only has cycles of even length, so it's easy to see that all triangles will contain the two vertices in the top partition. Still, $A \oplus \cdot \otimes A$ enumerates all wedges starting and ending in the bottom partition, thus performing a lot of unnecessary work.

A

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

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

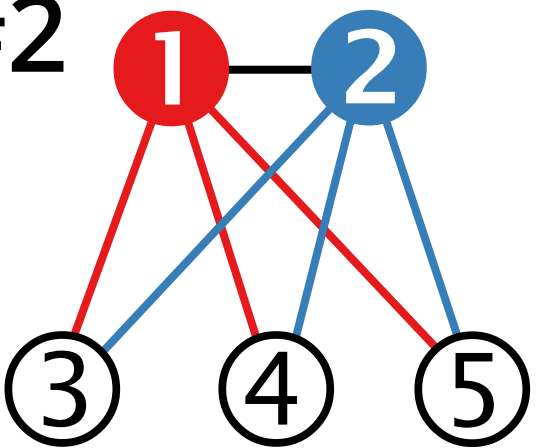
	①	②	③	④	⑤
		3	1	1	1
3			1	1	1
1	1				
1	1				
1	1				

A

$A \oplus \cdot \otimes A$

$A \oplus \cdot \otimes A \otimes A$

#2



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

A

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

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

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

6
6
2
2
2

A **TRI(A) = A \oplus . \otimes A**

tri

Masking avoids the materialization of large interim data sets by ensuring that we only enumerate wedges whose endpoints are already connected.

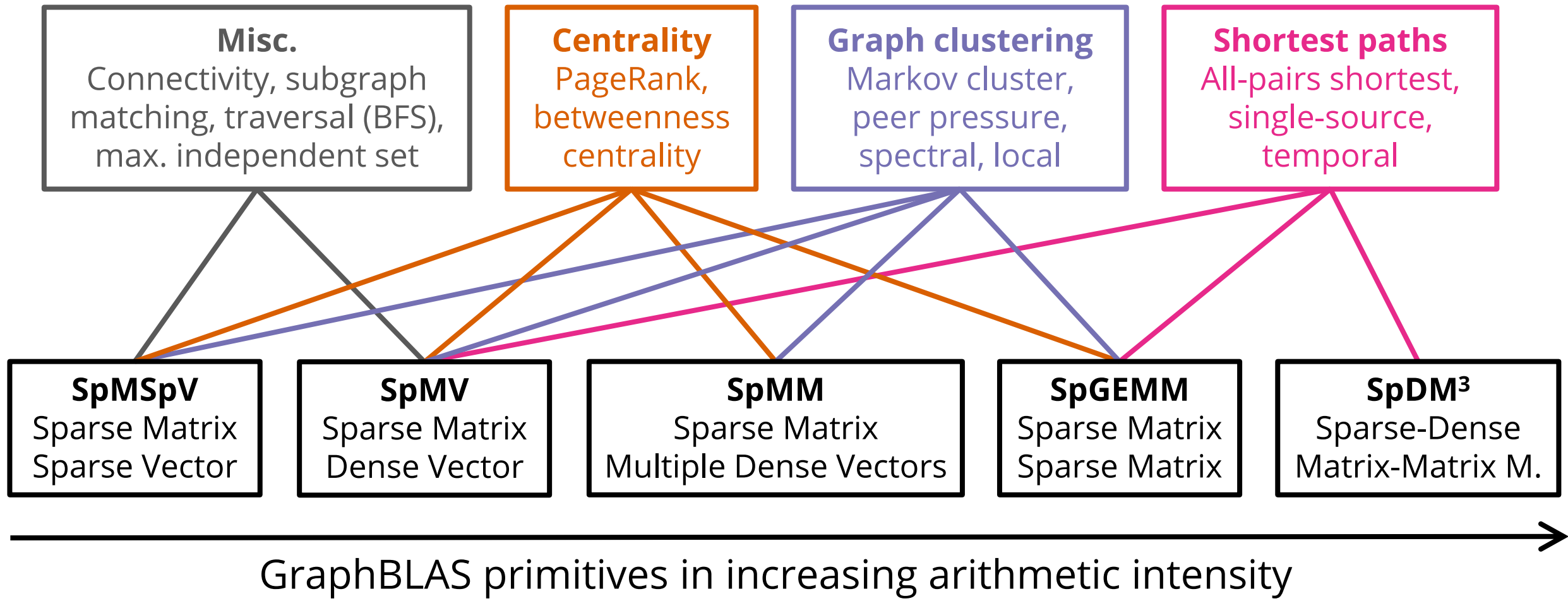
Graph algorithms in GraphBLAS

Summary and other algorithms

SUMMARY OF THE ALGORITHMS DISCUSSED

Algo.	Features demonstrated
BFS	lor-land and min-select semirings, masked SpMV
SSSP	min-plus semiring
PR	real semiring, row-wise reduce
LCC	apply operation, masked matrix-market multiplication

GRAPH ALGORITHMS & GRAPHBLAS PRIMITIVES



Based on the figure in A. Buluç:
Graph algorithms, computational motifs, and GraphBLAS, ECP Meeting 2018

OTHER ALGORITHMS IN GRAPHBLAS

Betweenness centrality: Brandes' algorithm



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

k -truss: a subset of the graph with the same number of vertices, where each edge appears in at least $k - 2$ triangles in the original graph.



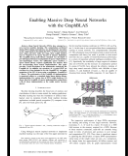
T.A. Davis: *Graph algorithms via SuiteSparse:GraphBLAS: triangle counting and k -truss*, HPEC 2018

Maximal independent set: Luby's randomized algorithm



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

Sparse DNNs: represent sparse deep neural networks as graphs



J. Kepner et al.: *Enabling Massive Deep Neural Networks with the GraphBLAS*, HPEC 2017



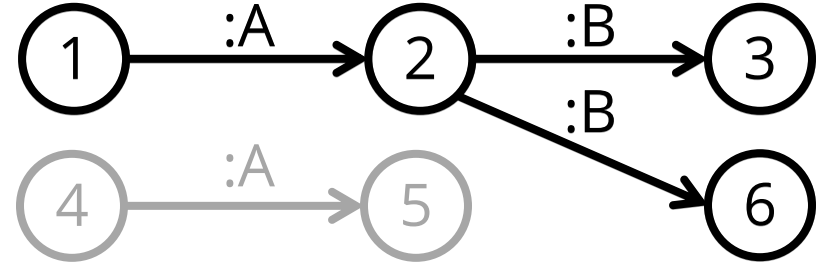
T.A. Davis et al.: *Write Quick, Run Fast: Sparse Deep Neural Network in 20 Minutes of Development Time via SuiteSparse:GraphBLAS*, HPEC 2019

Graph processing in relational algebra

RELATIONAL ALGEBRA ON GRAPHS

Natural join: $A \bowtie B$

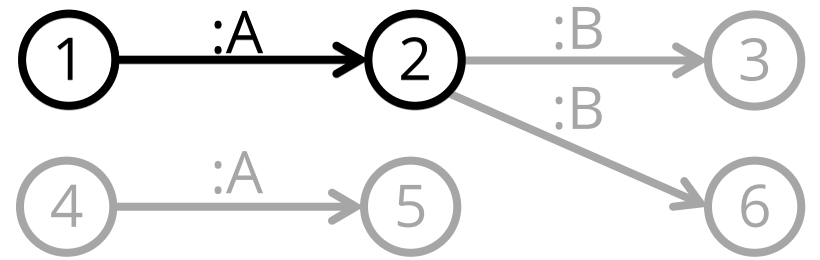
A natural join connects edges from A with an edge in B .



i	j	k
1	2	3
1	2	6

Semijoin: $A \ltimes B$

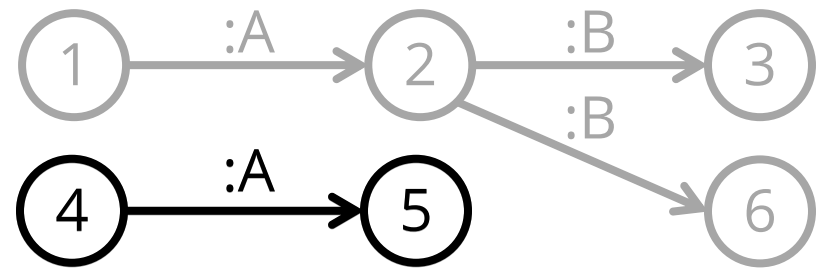
A semijoin keeps edges from A with a connecting edge in B .



i	j
1	2

Antijoin: $A \bar{\bowtie} B$

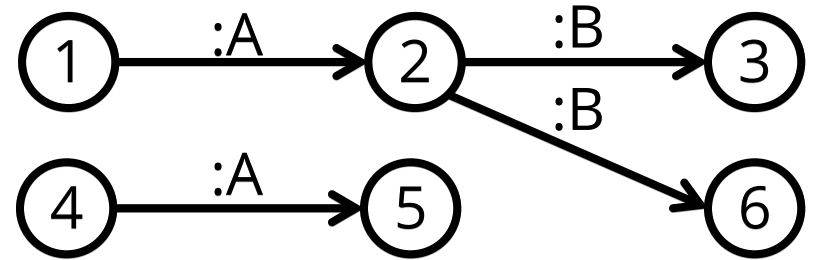
An antijoin keeps edges from A without a connecting edge in B .



i	j
4	5

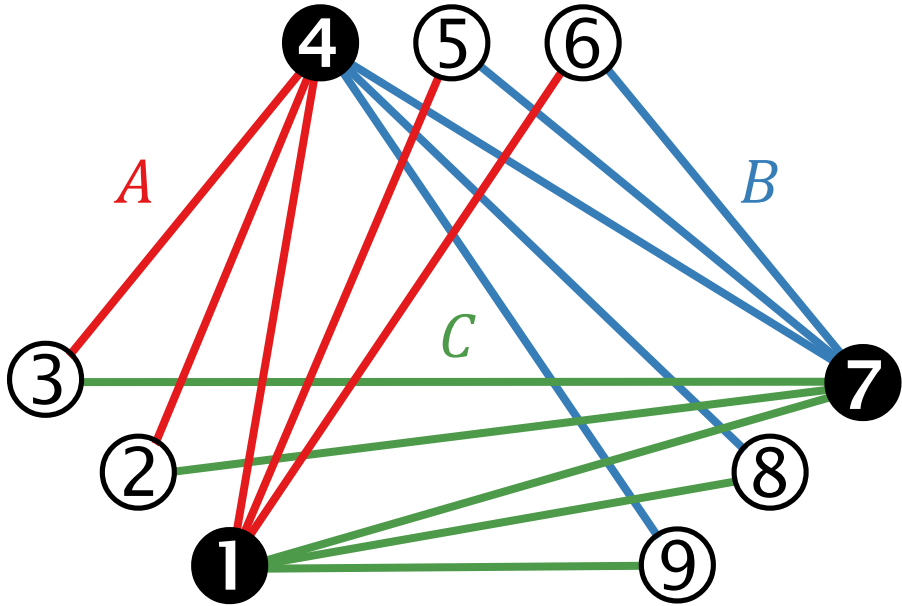
Left outer join: $A \ltimes B$

A left outer join keeps all edges from A with or without a connecting edge in B .



i	j	k
1	2	3
1	2	6
4	5	null

TRIANGLE QUERY



Given relations $A(i, j)$, $B(j, k)$, $C(i, k)$ triangles can be enumerated with:

$$A \bowtie B \bowtie C$$

$$A \bowtie B$$

i	j	k
1	4	7
1	4	8
1	4	9
2	4	7
2	4	8
2	4	9
3	4	7
3	4	8
3	4	9
1	5	7
1	6	7

wedges that do not close into a triangle

$$A \bowtie B \bowtie C$$

i	j	k
1	4	7
1	5	7
1	6	7
1	4	8
1	4	9
2	4	7
3	4	7

A

i	j
1	4
2	4
3	4
1	5
1	6

B

j	k
4	7
4	8
4	9
5	7
6	7

C

i	k
1	7
1	8
1	9
2	7
3	7

* Edges are stored in one direction: $R[1] < R[2]$

COMPLEXITY OF THE TRIANGLE QUERY

For m edges, the AGM-bound of the triangle query is $\mathcal{O}(m^{3/2})$

Theorem #1:

Any binary join plan for the triangle query takes $\Omega(m^2)$ time.

To mitigate this, worst-case optimal join (WCOJ) algorithm use multi-way joins such as $\bowtie (A, B, C)$.

Theorem #2:

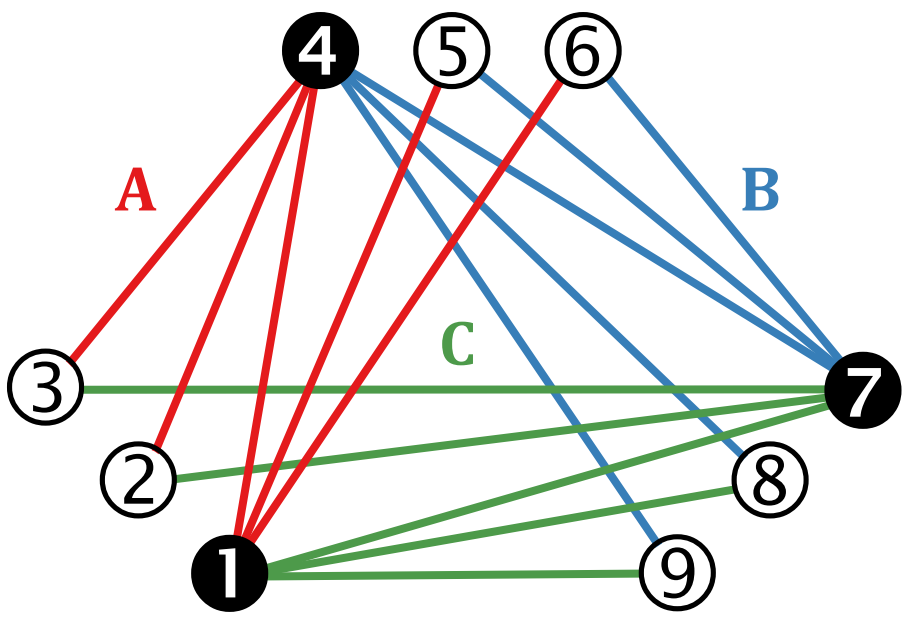
The worst-case complexity of the generic join algorithm (a WCOJ algorithm) \leq AGM bound.



A. Atserias, M. Grohe, D. Marx,
Size Bounds and Query Plans for Relational Joins,
FOCS 2008



H.Q. Ngo et al.: *Skew strikes back: New developments in the theory of join algorithms*, SIGMOD Record 2013



	B									C								
	1	2	3	4	5	6	7	8	9	1	2	3	4	5	6	7	8	9
1																		
2																		
3																		
4							1	1	1									
5							1											
6							1											
7																		
8																		
9																		

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

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

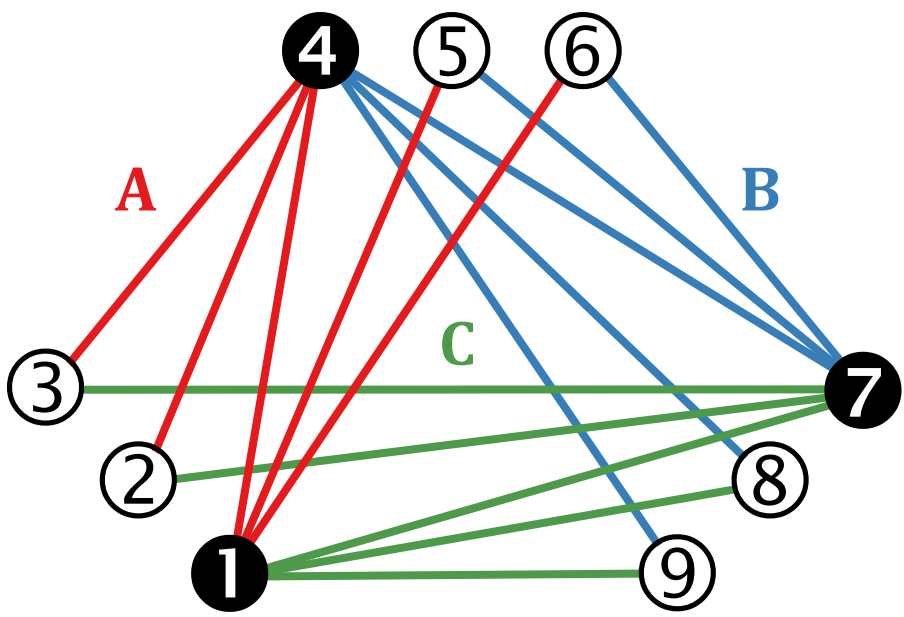
1							3	1	1	5
2							1			1
3							1			1
4										
5										
6										
7										
8										
9										

*We only use the lower triangular part.

A

A ⊕ ⊗ **B**

A ⊕ ⊗ **B** ⊗ **C** **tri**



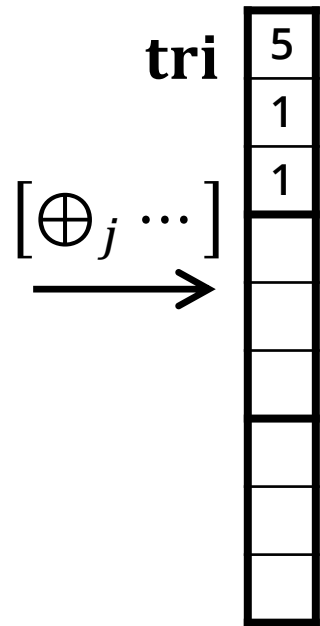
B ① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨

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

A ① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨

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

						3	1	1
						1		
						1		



$\text{TRI}\langle C \rangle = A \oplus . \otimes B$

*We only use the lower triangular part.

MASKING IN RELATIONAL ALGEBRA

Vertex-wise triangle count in linear algebra:

$$\begin{aligned}\mathbf{TRI}\langle \mathbf{C} \rangle &= \mathbf{A} \oplus . \otimes \mathbf{B} \\ \mathbf{tri} &= \left[\oplus_j \mathbf{TRI}(:, j) \right]\end{aligned}$$

Notice that in relational algebra, given $A(i, j)$, $B(j, k)$, $C(i, k)$:

$$\begin{aligned}A \bowtie B \bowtie C &\equiv (A \bowtie B) \bowtie C \\ A \bowtie B \bowtie C &\equiv (B \bowtie C) \bowtie A \equiv (B \bowtie C) \bowtie A \\ A \bowtie B \bowtie C &\equiv (A \bowtie C) \bowtie B \equiv (A \bowtie C) \bowtie B\end{aligned}$$

Where the semijoin operator \bowtie is defined as:

$$R \bowtie S = \pi_{\text{schema}(R)}(R \bowtie S) = R \bowtie \pi_{\text{schema}(R) \cap \text{schema}(S)}(S)$$

MASKING IN RELATIONAL ALGEBRA

Given relational schemas $A(i, j)$, $B(j, k)$, $C(i, k)$ and expression:

$$A \bowtie B \bowtie C \equiv (A \bowtie B) \bowtie C$$

The “masking” technique cannot be translated to relational algebra as a single operation – we have to use an outside loop to express it. Using relation $T(i, k)$:

1. for all $c \in C$ do

2. $TRI = TRI \cup \pi_{i,k}((A \bowtie \{c\}) \bowtie (B \bowtie \{c\}))$

3. $tri = \gamma_{i, \text{count}(k) \rightarrow \text{triCount}}(TRI)$

<i>i</i>	<i>triCount</i>
1	5
2	1
3	1

Cf. $\mathbf{TRI}\langle \mathbf{C} \rangle = \mathbf{A} \oplus \cdot \otimes \mathbf{B}$, $\mathbf{tri} = [\oplus_j \mathbf{TRI}(:, j)]$

TRIANGLE ENUMERATION IN RELATIONAL ALG.

Relational algebra also allows triangle enumeration.

Given $A(i, j)$, $B(j, k)$, $C(i, k)$, instead of the previous expression:

1. for all $c \in C$ do
2. $TRI = TRI \cup \pi_{i,k}((A \bowtie \{c\}) \bowtie (B \bowtie \{c\}))$
3. $tri = \gamma_{i,\text{count}(k) \rightarrow \text{triCount}}(TRI)$

Compute:

1. for all $c \in C$ do
2. $TRI = TRI \cup ((A \bowtie \{c\}) \bowtie (B \bowtie \{c\}))$

TRIANGLE ENUMERATION IN RELATIONAL ALG.

1. for all $c \in C$ do

2. $TRI = TRI \cup ((A \bowtie \{c\}) \bowtie (B \bowtie \{c\}))$

C	i	k	A	i	j	B	j	k	$A \bowtie B \bowtie C$	i	j	k
	1	7		1	4		4	7		1	4	7
	1	8		2	4		4	8		1	5	7
	1	9		3	4		4	9		1	6	7
	2	7		1	5		5	7		1	4	8
	3	7		1	6		6	7		1	4	9
										2	4	7
										3	4	7

TRIANGLE ENUMERATION IN RELATIONAL ALG.

1. for all $c \in C$ do

2. $TRI = TRI \cup ((A \bowtie \{c\}) \bowtie (B \bowtie \{c\}))$

C	<table border="1"><thead><tr><th>i</th><th>k</th></tr></thead><tbody><tr><td>1</td><td>7</td></tr><tr><td>1</td><td>8</td></tr><tr><td>1</td><td>9</td></tr><tr><td>2</td><td>7</td></tr><tr><td>3</td><td>7</td></tr></tbody></table>	i	k	1	7	1	8	1	9	2	7	3	7	A	<table border="1"><thead><tr><th>i</th><th>j</th></tr></thead><tbody><tr><td>1</td><td>4</td></tr><tr><td>2</td><td>4</td></tr><tr><td>3</td><td>4</td></tr><tr><td>1</td><td>5</td></tr><tr><td>1</td><td>6</td></tr></tbody></table>	i	j	1	4	2	4	3	4	1	5	1	6	B	<table border="1"><thead><tr><th>j</th><th>k</th></tr></thead><tbody><tr><td>4</td><td>7</td></tr><tr><td>4</td><td>8</td></tr><tr><td>4</td><td>9</td></tr><tr><td>5</td><td>7</td></tr><tr><td>6</td><td>7</td></tr></tbody></table>	j	k	4	7	4	8	4	9	5	7	6	7	$A \bowtie B \bowtie C$	<table border="1"><thead><tr><th>i</th><th>j</th><th>k</th></tr></thead><tbody><tr><td>1</td><td>4</td><td>7</td></tr><tr><td>1</td><td>5</td><td>7</td></tr><tr><td>1</td><td>6</td><td>7</td></tr><tr><td>1</td><td>4</td><td>8</td></tr><tr><td>1</td><td>4</td><td>9</td></tr><tr><td>2</td><td>4</td><td>7</td></tr><tr><td>3</td><td>4</td><td>7</td></tr></tbody></table>	i	j	k	1	4	7	1	5	7	1	6	7	1	4	8	1	4	9	2	4	7	3	4	7
i	k																																																																		
1	7																																																																		
1	8																																																																		
1	9																																																																		
2	7																																																																		
3	7																																																																		
i	j																																																																		
1	4																																																																		
2	4																																																																		
3	4																																																																		
1	5																																																																		
1	6																																																																		
j	k																																																																		
4	7																																																																		
4	8																																																																		
4	9																																																																		
5	7																																																																		
6	7																																																																		
i	j	k																																																																	
1	4	7																																																																	
1	5	7																																																																	
1	6	7																																																																	
1	4	8																																																																	
1	4	9																																																																	
2	4	7																																																																	
3	4	7																																																																	
		$A \bowtie \{c\}$	$B \bowtie \{c\}$																																																																

TRIANGLE ENUMERATION IN RELATIONAL ALG.

1. for all $c \in C$ do

2. $TRI = TRI \cup ((A \bowtie \{c\}) \bowtie (B \bowtie \{c\}))$

C	<table border="1"><thead><tr><th>i</th><th>k</th></tr></thead><tbody><tr><td>1</td><td>7</td></tr><tr><td>1</td><td>8</td></tr><tr><td>1</td><td>9</td></tr><tr><td>2</td><td>7</td></tr><tr><td>3</td><td>7</td></tr></tbody></table>	i	k	1	7	1	8	1	9	2	7	3	7	A	<table border="1"><thead><tr><th>i</th><th>j</th></tr></thead><tbody><tr><td>1</td><td>4</td></tr><tr><td>2</td><td>4</td></tr><tr><td>3</td><td>4</td></tr><tr><td>1</td><td>5</td></tr><tr><td>1</td><td>6</td></tr></tbody></table>	i	j	1	4	2	4	3	4	1	5	1	6	B	<table border="1"><thead><tr><th>j</th><th>k</th></tr></thead><tbody><tr><td>4</td><td>7</td></tr><tr><td>4</td><td>8</td></tr><tr><td>4</td><td>9</td></tr><tr><td>5</td><td>7</td></tr><tr><td>6</td><td>7</td></tr></tbody></table>	j	k	4	7	4	8	4	9	5	7	6	7	$A \bowtie B \bowtie C$	<table border="1"><thead><tr><th>i</th><th>j</th><th>k</th></tr></thead><tbody><tr><td>1</td><td>4</td><td>7</td></tr><tr><td>1</td><td>5</td><td>7</td></tr><tr><td>1</td><td>6</td><td>7</td></tr><tr><td>1</td><td>4</td><td>8</td></tr><tr><td>1</td><td>4</td><td>9</td></tr><tr><td>2</td><td>4</td><td>7</td></tr><tr><td>3</td><td>4</td><td>7</td></tr></tbody></table>	i	j	k	1	4	7	1	5	7	1	6	7	1	4	8	1	4	9	2	4	7	3	4	7
i	k																																																																		
1	7																																																																		
1	8																																																																		
1	9																																																																		
2	7																																																																		
3	7																																																																		
i	j																																																																		
1	4																																																																		
2	4																																																																		
3	4																																																																		
1	5																																																																		
1	6																																																																		
j	k																																																																		
4	7																																																																		
4	8																																																																		
4	9																																																																		
5	7																																																																		
6	7																																																																		
i	j	k																																																																	
1	4	7																																																																	
1	5	7																																																																	
1	6	7																																																																	
1	4	8																																																																	
1	4	9																																																																	
2	4	7																																																																	
3	4	7																																																																	
		$A \bowtie \{c\}$	$B \bowtie \{c\}$																																																																

TRIANGLE ENUMERATION IN RELATIONAL ALG.

1. for all $c \in C$ do

2. $TRI = TRI \cup ((A \bowtie \{c\}) \bowtie (B \bowtie \{c\}))$

C	<table border="1"><thead><tr><th>i</th><th>k</th></tr></thead><tbody><tr><td>1</td><td>7</td></tr><tr><td>1</td><td>8</td></tr><tr><td>1</td><td>9</td></tr><tr><td>2</td><td>7</td></tr><tr><td>3</td><td>7</td></tr></tbody></table>	i	k	1	7	1	8	1	9	2	7	3	7	A	<table border="1"><thead><tr><th>i</th><th>j</th></tr></thead><tbody><tr><td>1</td><td>4</td></tr><tr><td>2</td><td>4</td></tr><tr><td>3</td><td>4</td></tr><tr><td>1</td><td>5</td></tr><tr><td>1</td><td>6</td></tr></tbody></table>	i	j	1	4	2	4	3	4	1	5	1	6	B	<table border="1"><thead><tr><th>j</th><th>k</th></tr></thead><tbody><tr><td>4</td><td>7</td></tr><tr><td>4</td><td>8</td></tr><tr><td>4</td><td>9</td></tr><tr><td>5</td><td>7</td></tr><tr><td>6</td><td>7</td></tr></tbody></table>	j	k	4	7	4	8	4	9	5	7	6	7	$A \bowtie B \bowtie C$	<table border="1"><thead><tr><th>i</th><th>j</th><th>k</th></tr></thead><tbody><tr><td>1</td><td>4</td><td>7</td></tr><tr><td>1</td><td>5</td><td>7</td></tr><tr><td>1</td><td>6</td><td>7</td></tr><tr><td>1</td><td>4</td><td>8</td></tr><tr><td>1</td><td>4</td><td>9</td></tr><tr><td>2</td><td>4</td><td>7</td></tr><tr><td>3</td><td>4</td><td>7</td></tr></tbody></table>	i	j	k	1	4	7	1	5	7	1	6	7	1	4	8	1	4	9	2	4	7	3	4	7
i	k																																																																		
1	7																																																																		
1	8																																																																		
1	9																																																																		
2	7																																																																		
3	7																																																																		
i	j																																																																		
1	4																																																																		
2	4																																																																		
3	4																																																																		
1	5																																																																		
1	6																																																																		
j	k																																																																		
4	7																																																																		
4	8																																																																		
4	9																																																																		
5	7																																																																		
6	7																																																																		
i	j	k																																																																	
1	4	7																																																																	
1	5	7																																																																	
1	6	7																																																																	
1	4	8																																																																	
1	4	9																																																																	
2	4	7																																																																	
3	4	7																																																																	
		$A \bowtie \{c\}$	$B \bowtie \{c\}$																																																																

TRIANGLE ENUMERATION IN RELATIONAL ALG.

1. for all $c \in C$ do

2. $TRI = TRI \cup ((A \bowtie \{c\}) \bowtie (B \bowtie \{c\}))$

C	i	k	A	i	j	B	j	k	$A \bowtie B \bowtie C$	i	j	k
	1	7		1	4		4	7		1	4	7
	1	8		2	4		4	8		1	5	7
	1	9		3	4		4	9		1	6	7
	2	7		1	5		5	7		1	4	8
	3	7		1	6		6	7		1	4	9
										2	4	7
										3	4	7

TRIANGLE ENUMERATION IN RELATIONAL ALG.

1. for all $c \in C$ do
2. $TRI = TRI \cup ((A \bowtie \{c\}) \bowtie (B \bowtie \{c\}))$

C	i	k	A	i	j	B	j	k	$A \bowtie B \bowtie C$	i	j	k
	1	7		1	4		4	7		1	4	7
	1	8		2	4		4	8		1	5	7
	1	9		3	4		4	9		1	6	7
	2	7		1	5		5	7		1	4	8
	3	7		1	6		6	7		1	4	9
			$A \bowtie \{c\}$			$B \bowtie \{c\}$				2	4	7
										3	4	7

Using semijoin pushdown on each tuple, we computed $A \bowtie B \bowtie C$ **without enumerating a quadratic-size interim relation**. However, for efficient implementations, relations should also be sorted (see ref. for details).



H.Q. Ngo et al.: *Skew strikes back: New developments in the theory of join algorithms*, SIGMOD Record 2013

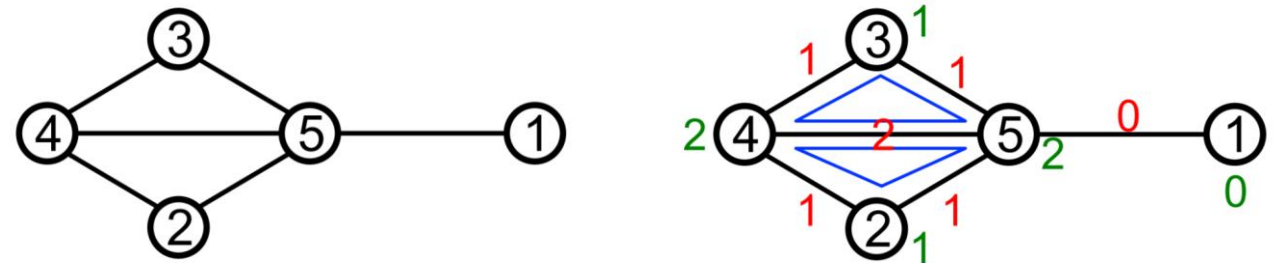
TRIANGLE ENUMERATION IN LINEAR ALGEBRA

(Note that this work is yet to be ported to GraphBLAS.)

Triangle enumeration is possible by using triples as matrix elements and overloading " $\oplus \cdot \otimes$ " so that multiplying the adjacency matrix **A** with the incidence matrix **B** results in **C**:

$$\mathbf{C}(i, j) = (i, x, y) \text{ iff } \mathbf{A}(i, x) = \mathbf{A}(i, y) = 1 \text{ and } \mathbf{B}(x, j) = \mathbf{B}(y, j) = 1$$

$$A = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}, B = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}, C = \begin{pmatrix} \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & (2, 4, 5) \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & (3, 4, 5) \\ \emptyset & (4, 2, 5) & (4, 3, 5) & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & (5, 3, 4) & (5, 2, 4) & \emptyset \end{pmatrix}$$



M.M. Wolf et al. *A task-based linear algebra building blocks approach for scalable graph analytics*, HPEC 2015

MATRIX MULTIPLICATION IN RELATIONAL ALG.

	set	\oplus	\otimes	0
set relations	$R \subseteq D_1 \times \dots \times D_k$	\cup	\bowtie	empty rel.
matrix relations	$R \subseteq D_1 \times D_2$	\cup	\boxtimes	empty rel.

Observations:

- a binary relation is analogous to a Boolean sparse matrix stored in coordinate list (COO) format
- a matrix multiplication is analogous to an equijoin on a binary relation that uses a single join attribute, and discards it after performing the join.

Definition: Given two binary relations R, S with a common attribute x , we define “multjoin” as :

$$R \boxtimes S = \pi_{R \cup S \setminus \{x\}} \left(R \bowtie_{R.x=S.x} S \right)$$

Example:

A	<table border="1"> <thead> <tr><th>i</th><th>j</th></tr> </thead> <tbody> <tr><td>1</td><td>3</td></tr> <tr><td>1</td><td>4</td></tr> <tr><td>2</td><td>4</td></tr> </tbody> </table>	i	j	1	3	1	4	2	4	B	<table border="1"> <thead> <tr><th>j</th><th>k</th></tr> </thead> <tbody> <tr><td>4</td><td>6</td></tr> <tr><td>4</td><td>7</td></tr> <tr><td>5</td><td>7</td></tr> </tbody> </table>	j	k	4	6	4	7	5	7	$A \boxtimes B$	<table border="1"> <thead> <tr><th>i</th><th>k</th></tr> </thead> <tbody> <tr><td>1</td><td>6</td></tr> <tr><td>1</td><td>7</td></tr> <tr><td>2</td><td>6</td></tr> <tr><td>2</td><td>7</td></tr> </tbody> </table>	i	k	1	6	1	7	2	6	2	7
i	j																														
1	3																														
1	4																														
2	4																														
j	k																														
4	6																														
4	7																														
5	7																														
i	k																														
1	6																														
1	7																														
2	6																														
2	7																														

MM VS. MULTJOIN: REACHABILITY

Matrices:

$$\mathbf{A} \in \mathbb{B}^{m \times n}, \mathbf{B} \in \mathbb{B}^{n \times o}, \mathbf{C} \in \mathbb{B}^{o \times p}$$

For $\mathbf{A} \vee \wedge \mathbf{B}$: $\langle \mathbf{AB} \rangle_{ik} = \bigvee_k (\mathbf{A}_{ij} \wedge \mathbf{B}_{jk})$

For $\mathbf{A} \vee \wedge \mathbf{B} \vee \wedge \mathbf{C}$:

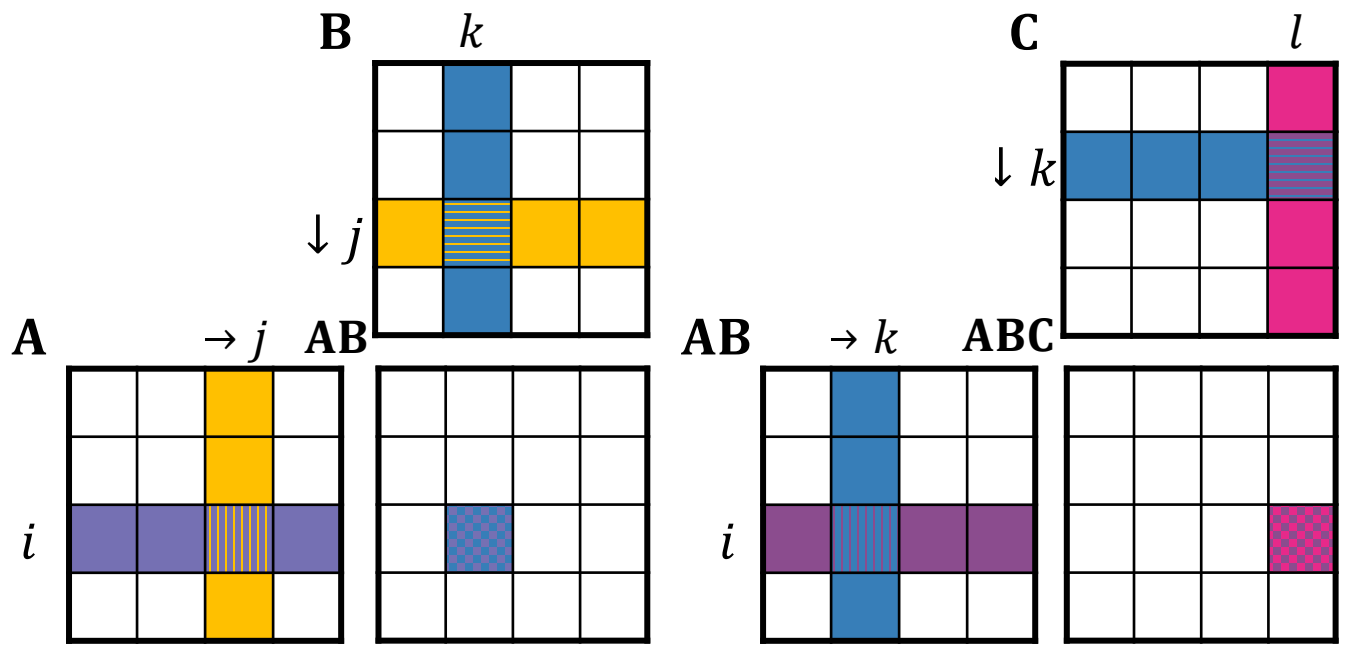
$$\begin{aligned} \langle \mathbf{ABC} \rangle_{il} &= \\ &= \bigvee_k (\langle \mathbf{AB} \rangle_{ik} \wedge \mathbf{C}_{kl}) \\ &= \bigvee_k \left(\bigvee_j (\mathbf{A}_{ij} \wedge \mathbf{B}_{jk}) \right) \wedge \mathbf{C}_{kl} \\ &= \bigvee_k \left(\bigvee_j (\mathbf{A}_{ij} \wedge \mathbf{B}_{jk} \wedge \mathbf{C}_{kl}) \right) \end{aligned}$$

Relations:

$$A(i, j), B(j, k), C(k, l)$$

$$A \boxtimes B \quad \text{sch: } (i, k)$$

$$A \boxtimes B \boxtimes C \quad \text{sch: } (i, l)$$



MM VS. JOIN: COUNTING PATHS

Matrices:

$$\mathbf{A} \in \mathbb{N}^{m \times n}, \mathbf{B} \in \mathbb{N}^{n \times o}, \mathbf{C} \in \mathbb{N}^{o \times p}$$

$$\langle \mathbf{AB} \rangle_{ik} = \sum_j \mathbf{A}_{ij} \otimes \mathbf{B}_{jk}$$

$$\begin{aligned} \langle \mathbf{ABC} \rangle_{il} &= \sum_k \left(\sum_j \mathbf{A}_{ij} \otimes \mathbf{B}_{jk} \right) \otimes \mathbf{C}_{kl} \\ &= \sum_k \left(\sum_j \mathbf{A}_{ij} \otimes \mathbf{B}_{jk} \otimes \mathbf{C}_{kl} \right) \end{aligned}$$

Relations:

$$A(i, j, va), B(j, k, vb), C(k, l, vc)$$

$$\gamma_{\text{sum}(A_{va} \cdot B_{vb})}^{A.i, B.k} (A \bowtie B)$$

$$\gamma_{\text{sum}(A_{va} \cdot B_{vb} \cdot C_{vc})}^{A.i, C.l} (A \bowtie B \bowtie C)$$

Enumerating triangles requires applying $\sigma_{A.i=C.l}(\dots)$

MATRIX MULTIPLICATION IN SQL: PATHS

$$\langle \mathbf{AB} \rangle_{ik} = \sum_j \mathbf{A}_{ij} \otimes \mathbf{B}_{jk} \Rightarrow \gamma_{\Sigma(A_{va} \cdot B_{vb})}^{A.i, B.k} (A \bowtie B)$$

```
select A.i, B.k, sum(A.va * B.vb)
from A, B
where A.j = B.j
group by A.i, B.k;
```

$$\langle \mathbf{ABC} \rangle_{il} = \sum_k \left(\sum_j \mathbf{A}_{ij} \otimes \mathbf{B}_{jk} \otimes \mathbf{C}_{kl} \right) \Rightarrow \gamma_{\Sigma(A_{va} \cdot B_{vb} \cdot C_{vc})}^{A.i, C.l} (A \bowtie B \bowtie C)$$

```
select A.i, C.l, sum(A.va * B.vb * C.vc)
from A, B, C
where A.j = B.j
    and B.k = C.k
group by A.i, C.l;
```

MATRIX MULTIPLICATION IN SQL: TRIANGLES

Relations $A(i, j, va)$, $B(j, k, vb)$, $C(k, l, vc)$

$$\langle \mathbf{ABC} \rangle_{ii} = \sum_k \left(\sum_j \mathbf{A}_{ij} \otimes \mathbf{B}_{jk} \otimes \mathbf{C}_{kl} \right) \Rightarrow \gamma_{\Sigma(A_{va} \cdot B_{vb} \cdot C_{vc})}^{A.i, C.l} \left(\sigma_{A.i=C.l} (A \bowtie B \bowtie C) \right)$$

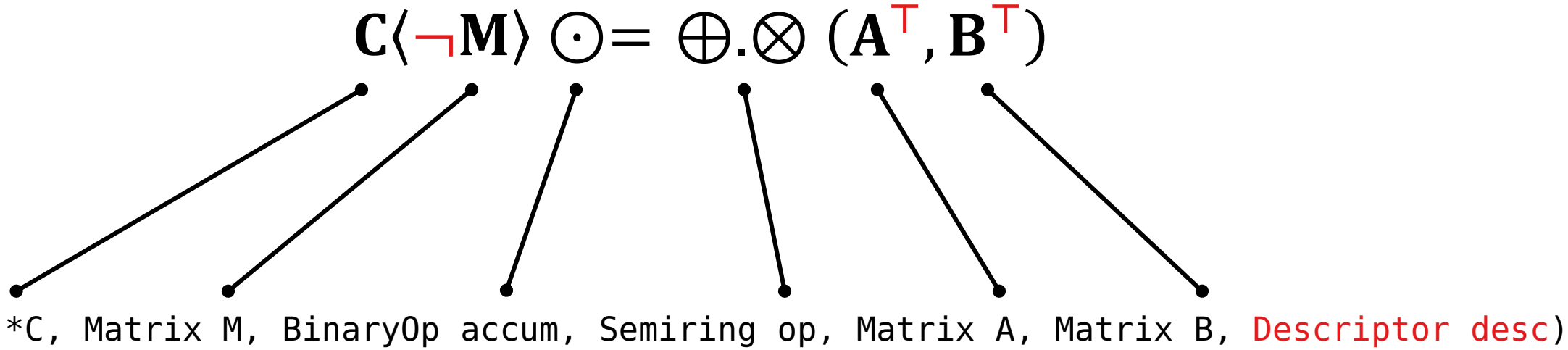
```
select A.i, C.l, sum(A.va * B.vb * C.vc)
from A, B, C
where A.j = B.j
      and B.k = C.k
      and A.i = C.l
group by A.i, C.l;
```

Given relations $A(i, j, va)$, $B(j, k, vb)$, $C(i, k, vc)$, we can simply use two natural joins: $A \bowtie B \bowtie C$.

GraphBLAS and SuiteSparse internals

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.”



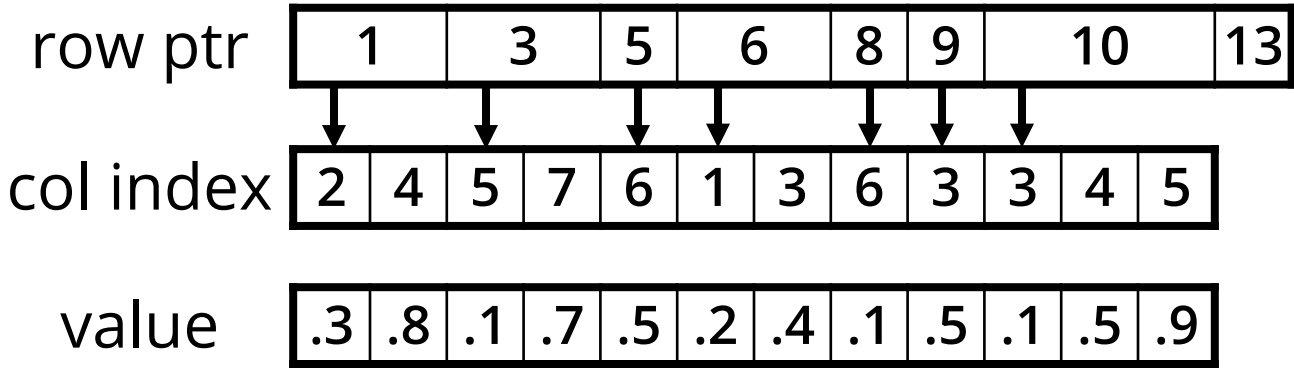
A. Buluç et al.: *Design of the GraphBLAS C API*, GABB@IPDPS 2017

GRAPHBLAS OBJECTS

- GraphBLAS objects are opaque: the matrix representation can be adjusted to suit the data distribution, hardware, etc.
- The typical representations are compressed formats are:
 - CSR: Compressed Sparse Row (also known as CRS)
 - CSC: Compressed Sparse Column (also known as CCS)

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

CSR representation of **A**:

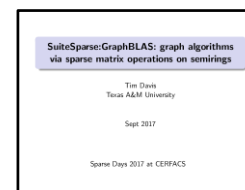


SUITESPARSE:GRAPHBLAS INTERNALS

- Authored by Prof. Tim Davis at Texas A&M University, based on his SuiteSparse library (used in MATLAB).
- Design decisions, algorithms and data structures are discussed in the TOMS paper and in the User Guide.
- Extensions: methods and types prefixed with GxB.
- Sophisticated load balancer for multi-threaded execution.
- GPU implementation is a work-in-progress.



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

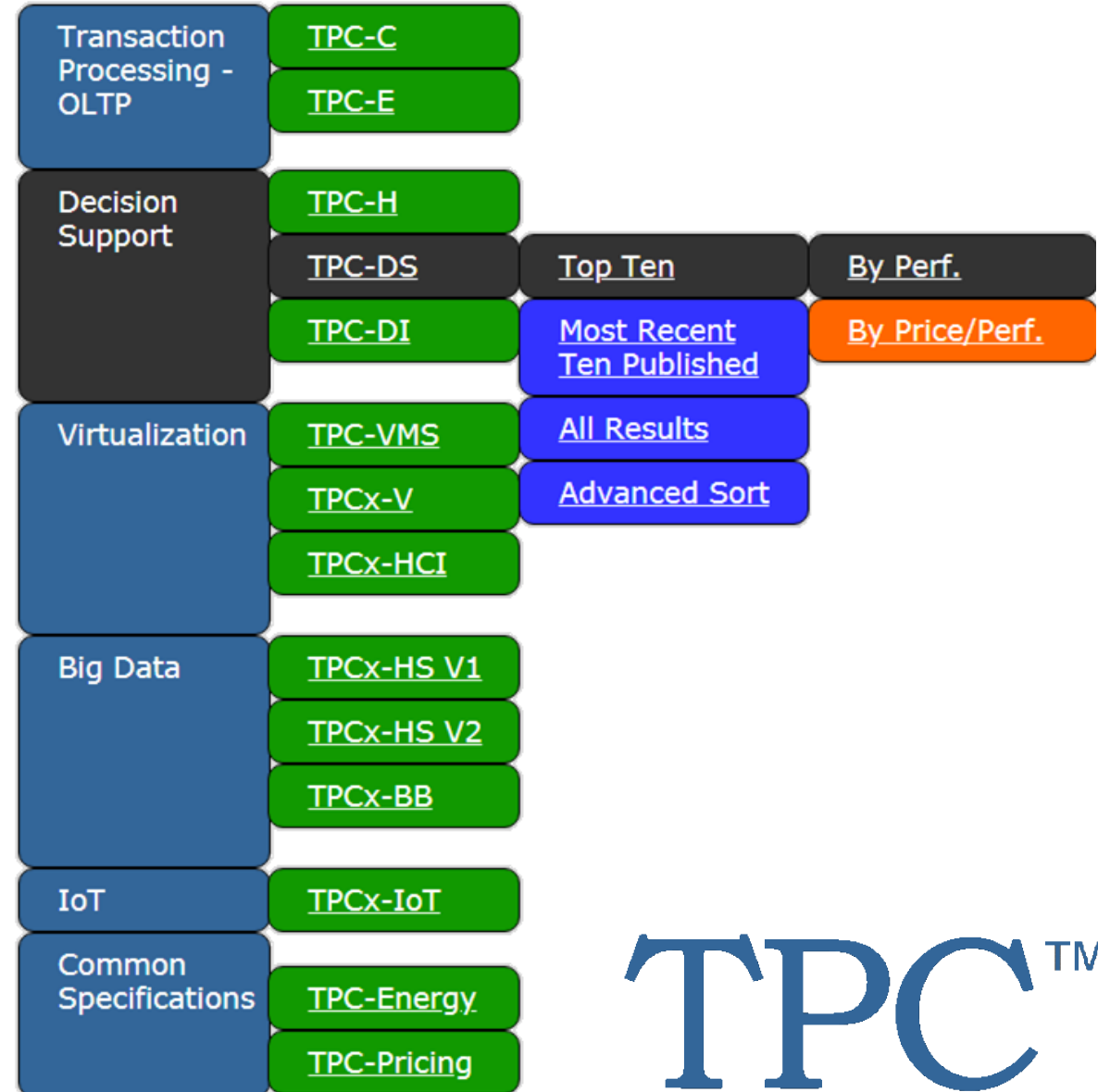
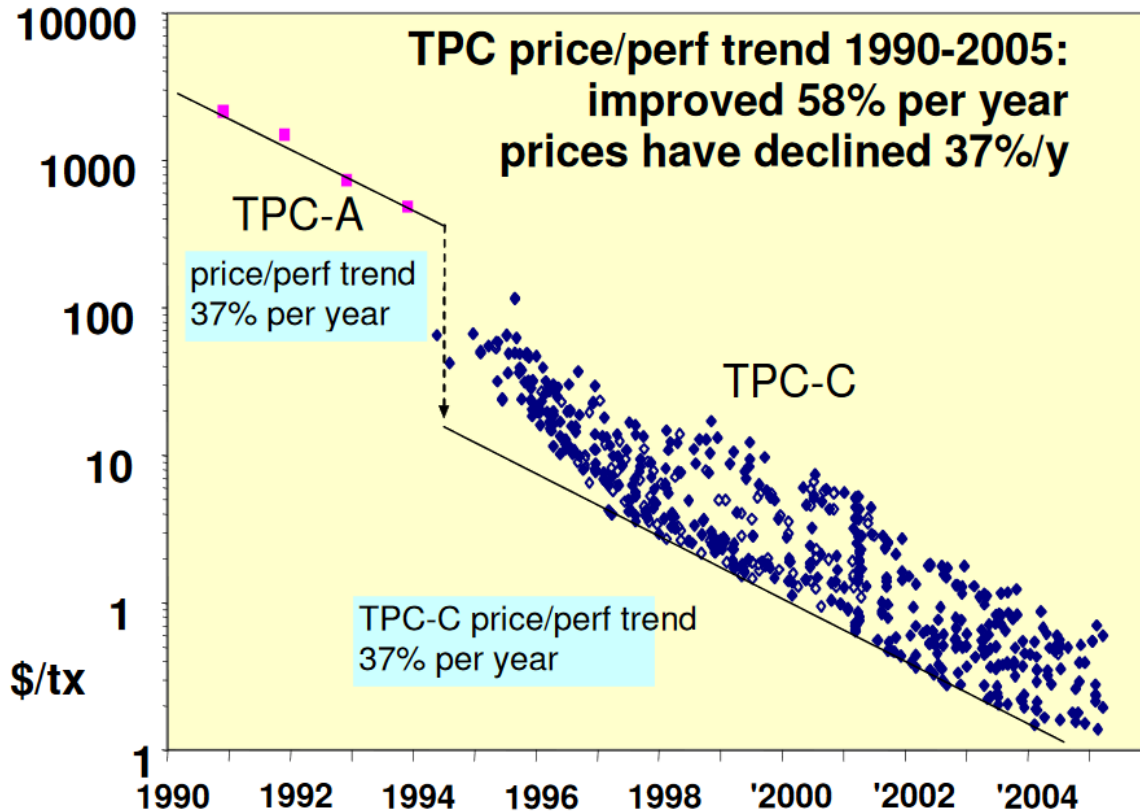


T.A. Davis: *SuiteSparse:GraphBLAS: graph algorithms via sparse matrix operations on semirings*, Sparse Days 2017

Graph benchmarks

TPC: TRANSACTION PROCESSING PERFORMANCE COUNCIL

Standard specifications for benchmarking certain aspects of relational DBs



TPC™

LDBC: LINKED DATA BENCHMARK COUNCIL

- A non-profit academic & industrial organization
- Dedicated to define standard benchmarks for *graph databases*

EU FP7 project — Community meetings — Benchmark papers



Social Network Benchmark
Interactive workload
SIGMOD 2015
Erling et al.

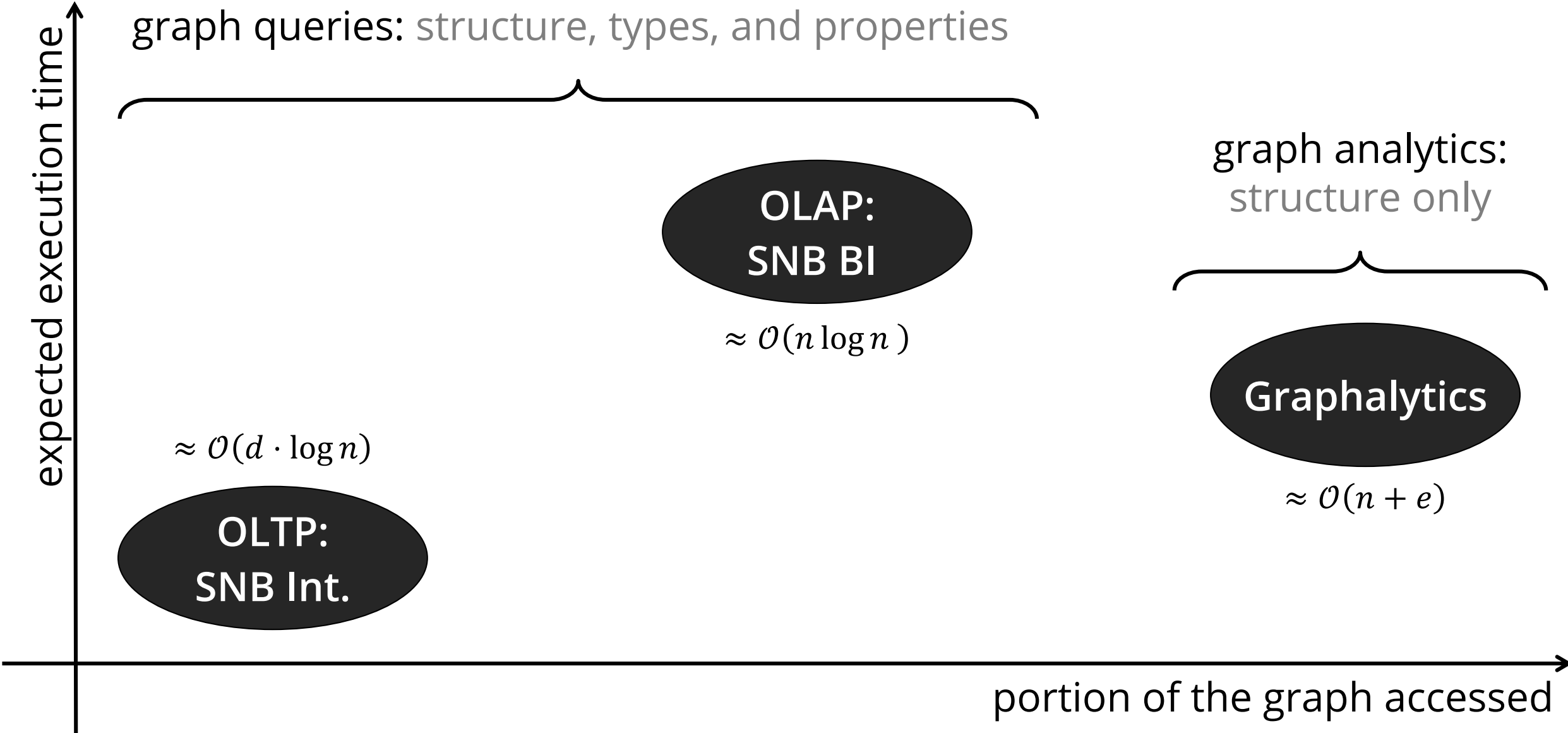


Graphalytics
VLDB 2016, Iosup et al.

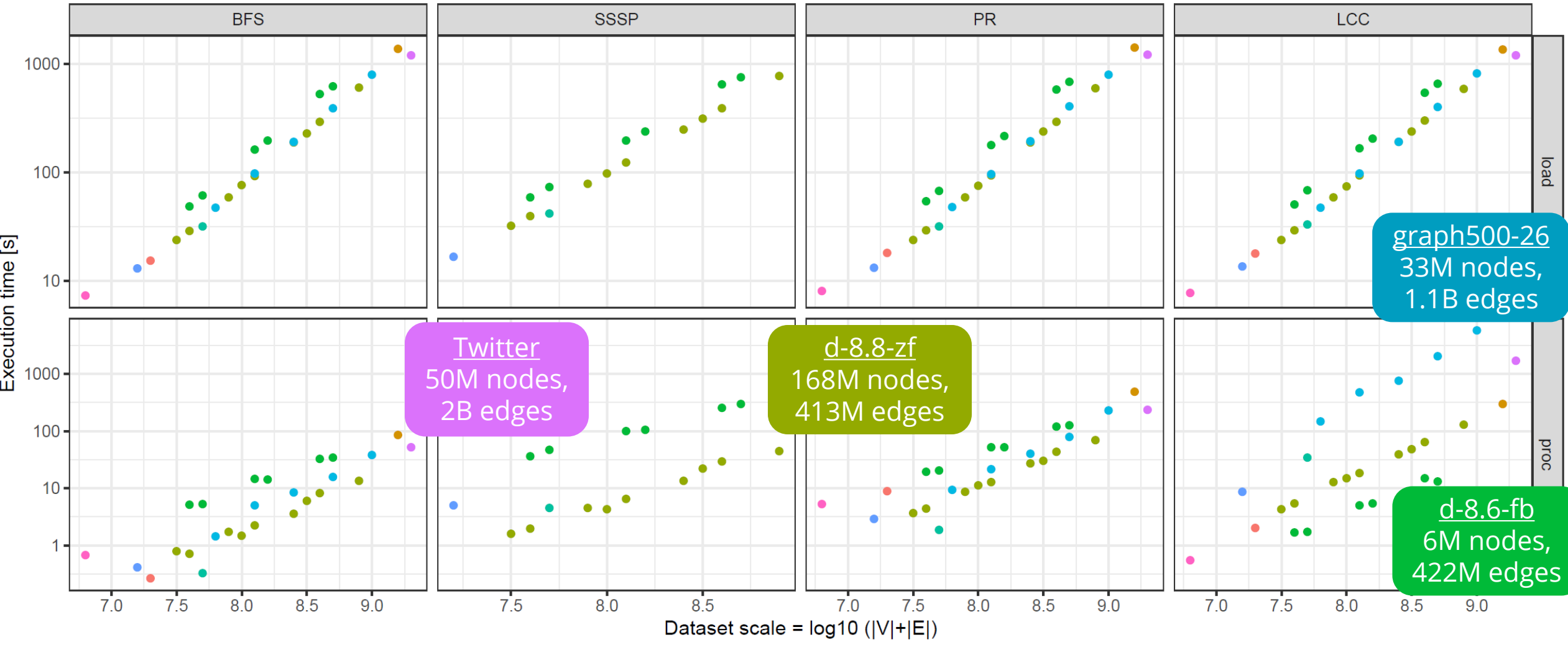


SNB
Business Intelligence
GRADES@SIGMOD 2018
Szárnyas et al.

GRAPH PROCESSING CATEGORIES IN LDDB



PRELIMINARY RESULTS WITH SUITESPARSE:GB



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

- cit
- datagen-fb
- dota
- kgs
- wiki
- com
- datagen-zf
- graph500
- twitter_mpi

loading often takes
longer than processing

THE GAP BENCHMARK SUITE

- Part of the *Berkeley Graph Algorithm Platform* project
- Same authors as the direction-optimizing BFS algorithm
- **Six algorithms:** BFS, SSSP, PageRank, connected components, betweenness centrality, triangle count
- **Five data sets:** Twitter, Web, Road, Kron, Urand
 - 24M to 134M vertices, 68M to 2.1B edges
 - Weighted/unweighted, directed/undirected
- Very efficient baseline implementation in C++
- Maintained on arXiv – v1: 2015, ..., v4: 2017



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

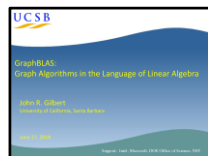


gap.cs.berkeley.edu/benchmark.html

Further reading and libraries

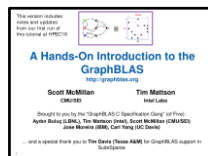
RESOURCES

Presentations and tutorials for learning GraphBLAS along with a link to a collection of pointers.



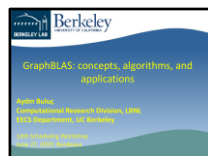
J.R. Gilbert:

GraphBLAS: Graph Algorithms in the Language of Linear Algebra, Seminar talk since 2014



S. McMillan and T.G. Mattson:

A Hands-On Introduction to the GraphBLAS, Tutorial at HPEC since 2018



A. Buluç:

GraphBLAS: Concepts, algorithms, and applications, Scheduling Workshop, 2019



M. Kumar, J.E. Moreira, P. Pattnaik:

GraphBLAS: Handling performance concerns in large graph analytics, Computing Frontiers 2018



List of GraphBLAS-related books, papers, presentations, posters, software, etc.
[szarnyasz/graphblas-pointers](https://github.com/szarnyasz/graphblas-pointers)

THE LAGRAPH LIBRARY

- Similar to the LAPACK library for BLAS
- Uses SuiteSparse:GraphBLAS
- Implementations of common algorithms
 - BFS, SSSP, LCC, PageRank, Boruvka
 - Triangle count, k -truss
 - CDLP (community detection using label propagation)
 - Weakly connected components, Strongly Conn. Comps (draft)
 - Betweenness centrality
 - Deep neural network



T.G. Mattson et al.: *LAGraph: A Community Effort to Collect Graph Algorithms Built on Top of the GraphBLAS*, GrAPL@IPDPS 2019



[GraphBLAS/LAGraph](https://github.com/GraphBLAS/LAGraph)

REQUIREMENTS BY GRAPH COMPUTATIONS

Libraries for linear-algebra based graph processing support the following features (prioritized):

1. Sparse matrices For reasonable performance
2. Arbitrary semirings For expressive power
3. Masking A big reduction in complexity for some algos
4. Parallel execution Constant speedup, ideally by `#threads`

Most matrix libraries **only satisfy requirement #1:**

- Intel MKL, Eigen, Boost uBLAS, MTL4, Armadillo, NIST Sparse BLAS, GMM++, CUSP (Thrust), Numpy, Efficient Java Matrix Library (EJML)
- The only exception is Julia's `SparseArrays` library, where arbitrary semirings can be used (#2) due to Julia's sophisticated type system

GRAPHBLAS PAPERS AND BOOKS



- *Standards for Graph Algorithm Primitives*

- Position paper by 19 authors @ IEEE HPEC 2013



- *Novel Algebras for Advanced Analytics in Julia*

- Technical paper on semirings in Julia @ IEEE HPEC 2013



- *Mathematical Foundations of the GraphBLAS*

- Theory paper by 16 authors @ IEEE HPEC 2016



- *Design of the GraphBLAS C API*

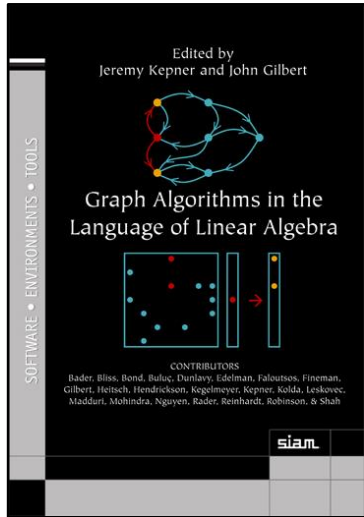
- Design decisions and overview of the C API @ GABB@IPDPS 2017



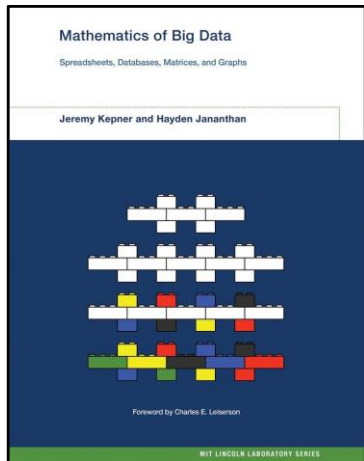
- *Algorithm 1000: SuiteSparse:GraphBLAS: graph algorithms in the language of sparse linear algebra*

- Algorithms in the SuiteSparse implementation @ ACM TOMS 2019

BOOKS



- *Graph Algorithms in the Language of Linear Algebra*
 - Edited by J. Kepner and J.R. Gilbert, published by SIAM in 2011
 - Algorithms for connected components, shortest paths, max-flow, betweenness centrality, spanning tree, graph generation, etc.
 - Algorithms and data structure for fast matrix multiplication
 - Predates GraphBLAS: preliminary notation, no API usage



- *Mathematics of Big Data*
 - Authored by Jananthan & Kepner, published by MIT Press in 2018
 - Generalizes the semiring-based approach for associative arrays
 - Contains reprints of papers, including the HPEC'16 paper above
 - Discusses D4M (Dynamic Distributed Dimensional Data Model)

GRAPHBLAS COMMUNITY

Wiki: graphblas.org | Communication: mailing list, monthly telcos

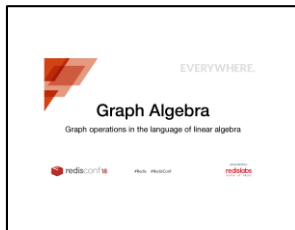
Annual events:

- May: IEEE IPDPS conference
 - GrAPL workshop (Graphs, Architectures, Programming and Learning), a merger of
 - GABB (Graph Algorithms Building Blocks)
 - GraML (Graph Algorithms and Machine Learning)
 - See graphanalysis.org for previous editions
- Sep: IEEE HPEC conference
 - GraphBLAS BoF meeting
- Nov: IEEE/ACM Supercomputing conference
 - GraphBLAS Working Group
 - IA³ workshop (Workshop on Irregular Applications: Architectures and Algorithms)

Blog: AldenMath by Timothy Alden Davis

REDISGRAPH

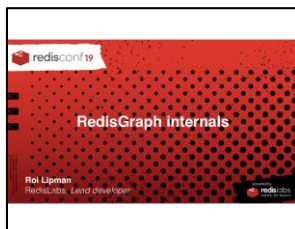
- Graph database built on top of Redis with partial (but extending) support for the Cypher language
- Uses SuiteSparse:GraphBLAS for graph operations
- Preliminary benchmark results show good performance on traversal-heavy workloads



R. Lipman, T.A. Davis:

Graph Algebra – Graph operations in the language of linear algebra

RedisConf 2018



R. Lipman:

RedisGraph internals

RedisConf 2019

GRAPHBLAS IMPLEMENTATIONS

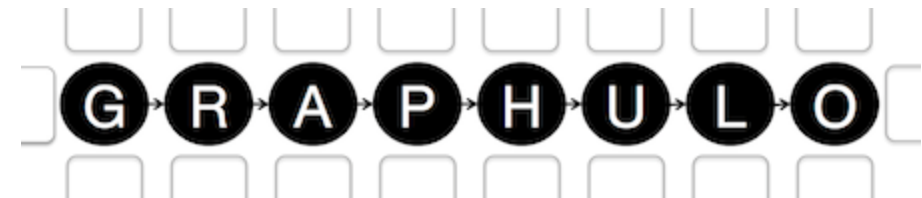
- SuiteSparse:GraphBLAS
 - v1.0.0: Nov 2017 – sequential
 - v3.0.1: July 2019 – parallel
 - v3.1.0: Oct 2019 – supports MATLAB bindings
- IBM GraphBLAS
 - Complete implementation in C++, released in May 2018
 - Concise but sequential
- GBTL (GraphBLAS Template Library): C++
 - v1.0: parallel but no longer maintained
 - v2.0: sequential
- GraphBLAST: GPU implementation, based on GBTL

GRAPHULO

- Build on top of the Accumulo distributed key-value store
- Written in Java
- Focus on scalability



V. Gadepally et al.:
*Graphulo: Linear Algebra Graph Kernels
for NoSQL Databases*, GABB@IPDPS 2015



COMBBLAS: COMBINATORIAL BLAS

- “an extensible distributed memory parallel graph library offering a small but powerful set of linear algebra primitives”
- Not a GraphBLAS implementation but serves as an incubator for new ideas that may later find their way into GraphBLAS
- Scales to 250k+ CPU cores
 - Used on supercomputers such as Cray



A. Buluç, J.R. Gilbert: *The Combinatorial BLAS: design, implementation, and application*, International Journal of High Performance Computing Applications, 2011

Learning GraphBLAS

LEARNING GRAPHBLAS

- Challenging to get started – need to have an understanding of:
 - Linear algebra (semirings, matrix multiplication, etc.)
 - Connection between linear algebra and graph operations (intuition)
 - Sizeable C API
- 200+ slides presenting algorithms in GraphBLAS
 - BFS variants, PageRank, Triangle count, k -truss
 - Community detection using label propagation
 - Luby's maximal independent set algorithm
 - computing connected components on an overlay graph
 - etc.

PYGRAPHBLAS: PYTHON WRAPPER #1

- A Python wrapper
- See example code for SSSP and triangle count
- Close to pseudo-code
- Jupyter support

Added benefit: both Python and GraphBLAS use 0-based indexing



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

```
def sssp(matrix, start):
    v = Vector.from_type(matrix.gb_type, matrix.nrows)
    v[start] = 0

    with min_plus, Accum(min_int64):
        for _ in range(matrix.nrows):
            w = Vector.dup(v)
            v @= matrix
            if w == v:
                break

    return v
```

```
def sandia(A, L):
    return L.mxm(L, mask=L).reduce_int()
```

```
sandia(M, M.tril())
```

GRBLAS: PYTHON WRAPPER #2

- Goal: one-to-one mapping of the GraphBLAS API
 - less Pythonic
 - comes with a Conda package
 - compiles user-defined functions to C

TRADEOFFS OF GRAPHBLAS

- Updating CSR/CSC matrices is slow
 - SuiteSparse mitigates this to some extent by allowing incremental build before using the matrix but it still needs to rebuild the matrix before major operations such as mxv , mxm
 - But: GraphBLAS objects are opaque and alternative formats can be used
 - Hornet [HPEC'18]
 - faimGraph [SC'18]
 - STINGER [HPEC'12]

Summary

VENUES ON GRAPH PROCESSING TECHNIQUES

- Database systems SIGMOD, VLDB, ICDE, EDBT, VLDB J., TKDE
- Database theory PODS, ICDT, TODS
- Information retrieval KDD, CIKM, WSDM, TKDD, DMKD J., & Big Data
- Operating systems OSDI, SOSP, NSDI, ICPE, EuroSys
- High-perf. computing SC, HPEC, HiPC, HPCC, HPDC, IPDPS
- Distributed computing PODC, DISC, ICDCS
- Parallel programming SPAA, ICPP, PLDI, PPOPP, ACM TOPC, Euro-Par
- Algorithm design TOMS, FOCS, STOC, SODA, SIAM J. Comput.
- Software engineering ICSE, MODELS, FSE, ASE, FASE, OOPSLA, SoSyM
- Semantic web WWW (TheWebConf), ISWC, WWW J., SWJ
- Graph transformation ICMT/ICGT, CAV, FMCAD, AGTIVE, ENTCS
- ...and many other mathematics and computer science venues.

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 (maths, C programming, 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

SOME OPEN QUESTIONS

- What primitives should be added to GraphBLAS?

- Sorting (CDLP)
- Submatrix selection (k -truss)
- Permutations (DFS)



D.G. Spampinato et al.:
Linear Algebraic Depth-First Search,
ARRAYS@PLDI 2019

- How to efficiently express SCC?

- How to implement GraphBLAS in a distributed system?



Y. Ahmad et al.: *LA3: A Scalable Link- and Locality-Aware Linear Algebra-Based Graph Analytics System*, VLDB 2018

- Incremental evaluation of SpMM/SpMV operations

- Use an updatable matrix representation
- Semirings \rightarrow rings \rightarrow additive inverse \rightarrow addition and removal
- Incremental view maintenance techniques need rings (?)

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

“Nuances” – Some important adjustments to the definitions

GRAPHBLAS SEMIRINGS*

The GraphBLAS specification defines semirings as follows:

$\langle D_{\text{out}}, D_{\text{in}_1}, D_{\text{in}_2}, \oplus, \otimes, 0 \rangle$ structure is a *GraphBLAS semiring* defined by

- $D_{\text{out}}, D_{\text{in}_1}$, and D_{in_2} three domains
- $\oplus: D_{\text{out}} \times D_{\text{out}} \rightarrow D_{\text{out}}$ an associative and commutative addition operation
- $\otimes: D_{\text{in}_1} \times D_{\text{in}_2} \rightarrow D_{\text{out}}$ a multiplicative operation
- $0 \in D_{\text{out}}$ an identity element for \oplus

$A = \langle D_{\text{out}}, \oplus, 0 \rangle$ is a commutative monoid.

$F = \langle D_{\text{out}}, D_{\text{in}_1}, D_{\text{in}_2}, \otimes \rangle$ is a closed binary operator.

“It is expected that implementations will utilize IEEE-754 floating point arithmetic, which is not strictly associative.” (C API specification)

MATRIX MULTIPLICATION*

The simple formula of matrix multiplication

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

can lead to more complicated definitions. For example, what should the binary operator $\text{sel1st}(x, y) = x$ return if $y = 0$?

$$\text{sel1st}(x, y) = \begin{cases} 0 & \text{if } y = 0 \\ x & \text{otherwise} \end{cases}$$

To simplify definitions, we use a generalized form of multiplication:

$$\mathbf{C}(i, j) = \bigoplus_{k \in \text{ind}(\mathbf{A}(i,:)) \cap \text{ind}(\mathbf{B}(:,j))} \mathbf{A}(i, k) \otimes \mathbf{B}(k, j)$$

where we only perform the multiplication in the intersection of the non-zero indices of \mathbf{A} 's i th row and \mathbf{B} 's j th column.

NOTATION*

- Symbols:
 - **A, B, C, M** – matrices
 - **u, v, w, m** – vectors
 - *s, k* – scalar
 - *i, j* – indices
 - $\langle \mathbf{m} \rangle, \langle \mathbf{M} \rangle$ – masks

- Operators:

- \oplus – addition
- \otimes – multiplication
- T – transpose
- \odot – accumulator

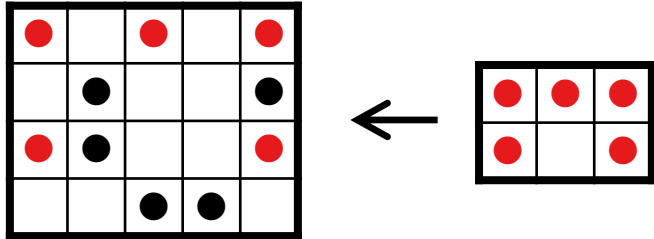
} Not included in the simplified table

This table contains all GrB and GxB (SuiteSparse-specific) operations.

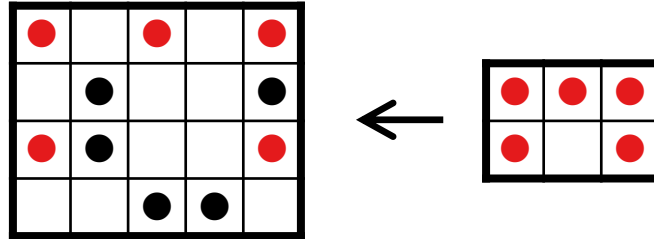
symbol	operation	notation
$\oplus \cdot \otimes$	matrix-matrix multiplication	$\mathbf{C}\langle \mathbf{M} \rangle \odot = \mathbf{A} \oplus \cdot \otimes \mathbf{B}$
	vector-matrix multiplication	$\mathbf{w}\langle \mathbf{m} \rangle \odot = \mathbf{v} \oplus \cdot \otimes \mathbf{A}$
	matrix-vector multiplication	$\mathbf{w}\langle \mathbf{m} \rangle \odot = \mathbf{A} \oplus \cdot \otimes \mathbf{v}$
\otimes	element-wise multiplication (set intersection of patterns)	$\mathbf{C}\langle \mathbf{M} \rangle \odot = \mathbf{A} \otimes \mathbf{B}$
		$\mathbf{w}\langle \mathbf{m} \rangle \odot = \mathbf{u} \otimes \mathbf{v}$
\oplus	element-wise addition (set union of patterns)	$\mathbf{C}\langle \mathbf{M} \rangle \odot = \mathbf{A} \oplus \mathbf{B}$
		$\mathbf{w}\langle \mathbf{m} \rangle \odot = \mathbf{u} \oplus \mathbf{v}$
<i>f</i>	apply unary operator	$\mathbf{C}\langle \mathbf{M} \rangle \odot = f(\mathbf{A})$
		$\mathbf{w}\langle \mathbf{m} \rangle \odot = f(\mathbf{v})$
$[\oplus \dots]$	reduce to vector	$\mathbf{w}\langle \mathbf{m} \rangle \odot = [\oplus_j \mathbf{A}(:, j)]$
	reduce to scalar	$s \odot = [\oplus_{ij} \mathbf{A}(i, j)]$
\mathbf{A}^T	transpose matrix	$\mathbf{C}\langle \mathbf{M} \rangle \odot = \mathbf{A}^T$
-	extract submatrix	$\mathbf{C}\langle \mathbf{M} \rangle \odot = \mathbf{A}(i, j)$
		$\mathbf{w}\langle \mathbf{m} \rangle \odot = \mathbf{v}(i)$
-	assign submatrix with submask for $\mathbf{C}\langle \mathbf{I}, \mathbf{J} \rangle$	$\mathbf{C}\langle \mathbf{M} \rangle(i, j) \odot = \mathbf{A}$
		$\mathbf{w}\langle \mathbf{m} \rangle(i) \odot = \mathbf{v}$
-	assign submatrix with mask for \mathbf{C}	$\mathbf{C}(i, j)\langle \mathbf{M} \rangle \odot = \mathbf{A}$
		$\mathbf{w}(i)\langle \mathbf{m} \rangle \odot = \mathbf{v}$
-	apply select operator (GxB)	$\mathbf{C}\langle \mathbf{M} \rangle \odot = f(\mathbf{A}, k)$
		$\mathbf{w}\langle \mathbf{m} \rangle \odot = f(\mathbf{v}, k)$
-	Kronecker product	$\mathbf{C}\langle \mathbf{M} \rangle \odot = \text{kron}(\mathbf{A}, \mathbf{B})$

LINEAR ALGEBRAIC PRIMITIVES FOR GRAPHS #3*

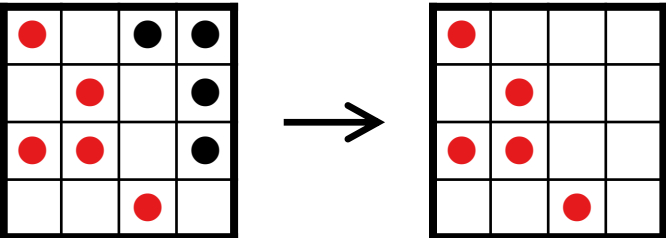
Sparse matrix extraction:
induced subgraph



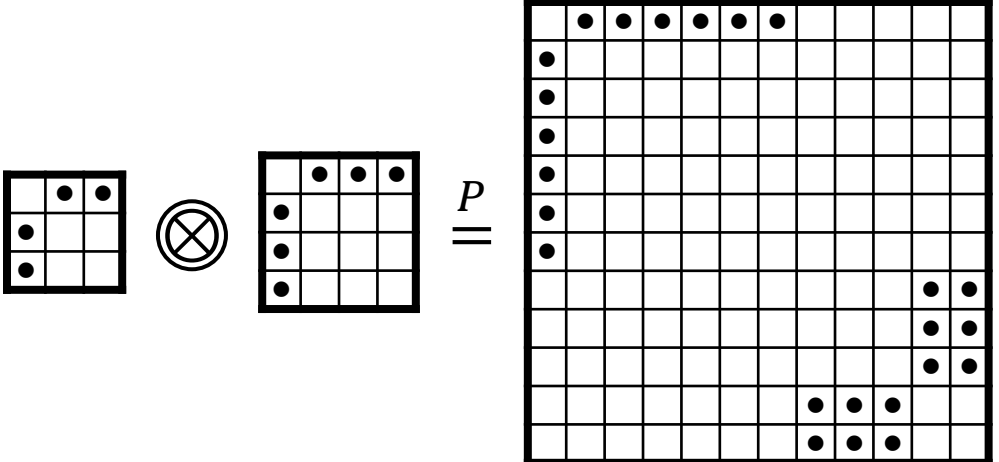
Sparse submatrix assignment:
replace subgraph



Sparse matrix selection (SuiteSparse):
filtering edges



Kronecker product:
graph generation



MATRIX-VECTOR MULTIPLICATION*

The operation $\mathbf{v} \oplus.\otimes \mathbf{A}$ gives the vertices reachable from the ones in \mathbf{v} . However, GraphBLAS publications and implementations often use $\mathbf{A}^\top \oplus.\otimes \mathbf{v}$ instead. The difference between these is that the former produces a row vector, while the latter produces a column vector:

$$\mathbf{v} \oplus.\otimes \mathbf{A} \equiv (\mathbf{A}^\top \oplus.\otimes \mathbf{v}^\top)^\top$$

GraphBLAS does not distinguish row/column vectors, therefore the notations are (formally) equivalent:

$$\mathbf{v} \oplus.\otimes \mathbf{A} \equiv \mathbf{A}^\top \oplus.\otimes \mathbf{v}$$

In practice, the order of the operands can have a significant impact on performance. For example, if matrices are stored in CSR format, computing $\mathbf{v} \oplus.\otimes \mathbf{A}$ is expensive compared to $\mathbf{A}^\top \oplus.\otimes \mathbf{v}$.

Ω

Extra slides

THE CASE FOR LINEAR ALGEBRA-BASED GRAPH ALGORITHMS

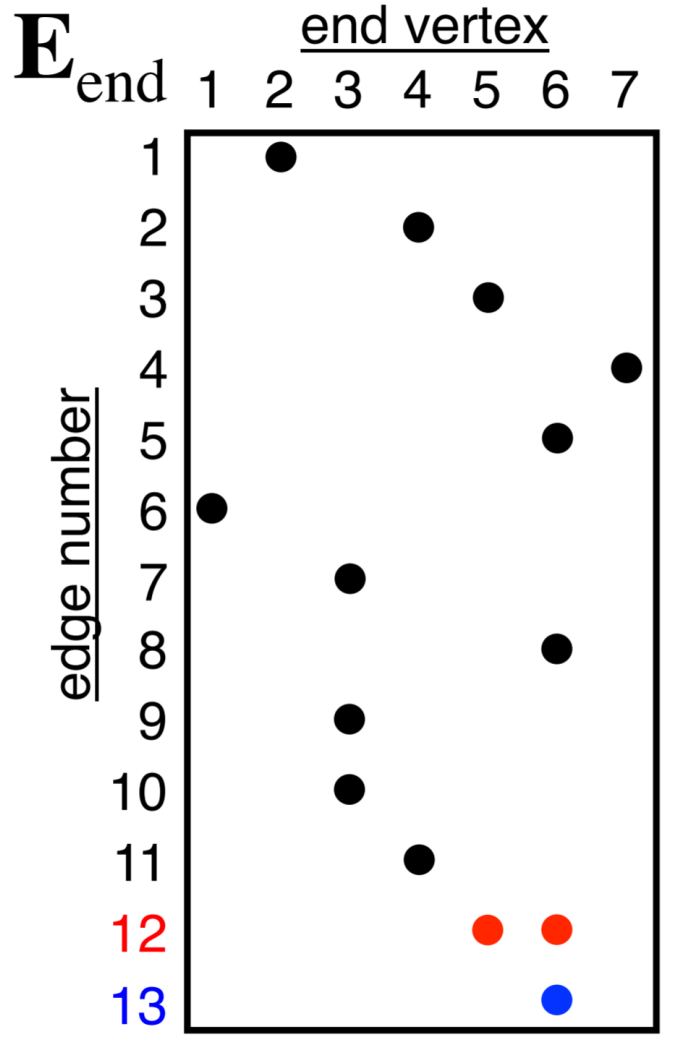
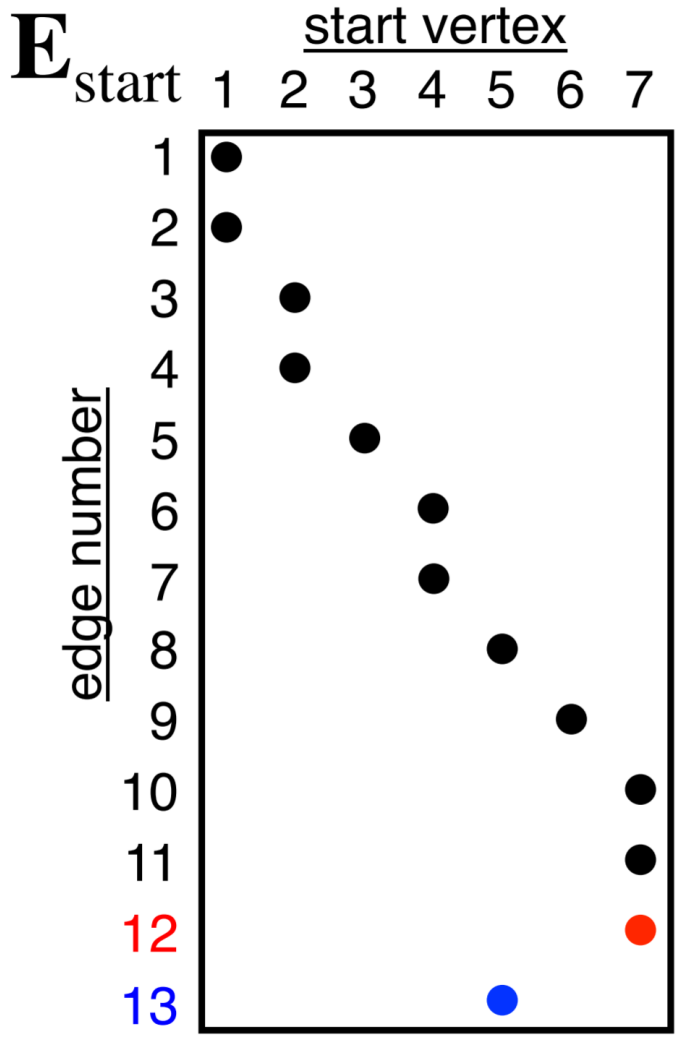
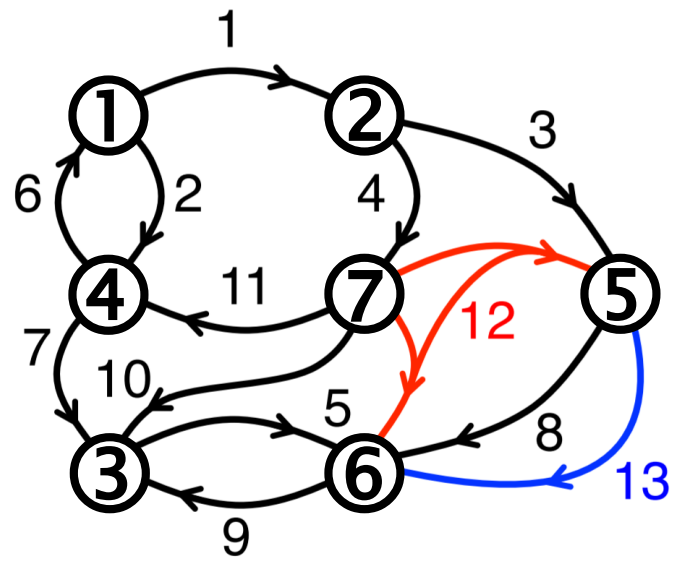
Many irregular applications contain coarse-grained parallelism that can be exploited by abstractions at the proper level.

Traditional graph computation	Graphs in the language of linear algebra
Data-driven, unpredictable communication	Fixed communication patterns
Irregular and unstructured, poor locality of reference	Operations on matrix blocks exploit memory hierarchy
Fine-grained data accesses, dominated by latency	Coarse-grained parallelism, bandwidth-limited



D. Bader et al., *The GraphBLAS effort and its implications for Exascale*, SIAM Workshop on Exascale Applied Mathematics Challenges and Opportunities, 2014

DIRECTED HYPERGRAPH EXAMPLE



Graph algorithms in GraphBLAS

Weakly connected components

WCC: WEAKLY CONNECTED COMPONENTS (SKETCH)

$$C = I \vee A \vee A^2 \vee A^3 \vee A^4 \vee \dots$$

$$D = I + A + A^2 + A^3 + A^4 \vee \dots \vee A^K$$

$$E \equiv (I - A)D$$

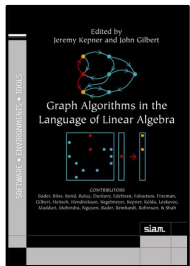
$$E = D - AD$$

$$= I + A + A^2 + A^3 + A^4 + \dots - A - A^2 - A^3 - A^4 - \dots$$

$$= I$$

$$= (I - A)D$$

$$D = (I - A)^{-1}$$



J. Kepner, J.R. Gilbert:

Graph Algorithms in the Language of Linear Algebra, Chapter 3,
SIAM, 2011

WCC: WEAKLY CONNECTED COMPONENTS

- The series for D does usually not converge. However, by replacing A with αA , where α is a sufficiently small positive number, the series converges. The derivation presented in the previous slide is still valid, resulting in $D = (I - \alpha A)^{-1}$.
- While this trick ensures that the presented algorithm works correctly, this approach is still impractical as *it computes the inverse of a sparse matrix* which is dense in most cases.

WCC: WEAKLY CONNECTED COMPONENTS

- Linear Algebraic Connected Components algorithm (LACC)
 - Recent result, based on the Awerbuch-Shiloach algorithm.
 - Designed for supercomputers, scales to 250,000+ CPU cores.
 - “Distributed-memory LACC code that is used in our experiments is publicly available as part of the CombBLAS library. A simplified unoptimized serial GraphBLAS implementation is also committed to the LAGraph Library for educational purposes.”
- See also FastSV.



A. Azad, A. Buluç: *LACC: A Linear-Algebraic Algorithm for Finding Connected Components in Distributed Memory*, IPDPS 2019

Graph algorithms in GraphBLAS

Maximal independent set: Luby's algorithm

LUBY'S ALGORITHM

To be done later.

Notes

ABOUT THIS PRESENTATION

- This presentation is intended to serve as an introduction to semiring-based graph processing and the GraphBLAS.
- Common graph algorithms (BFS, shortest path, PageRank, etc.) are used to demonstrate the features of GraphBLAS. Many of the algorithms presented are part of the LAGraph library.
- The presentation complements existing technical talks on GraphBLAS and can form a basis of anything from a short 20min overview to 2×90min lectures on the GraphBLAS.
- The slides contain numerous references to papers and talks.
- There are detailed textual explanations on some slides to make them usable as a learning material.

TECHNICAL DETAILS

- The slides were created with PowerPoint 2016 using the Open Sans and DejaVu Sans Mono font families (embedded in the presentation).
- The mathematical expressions are typeset with PowerPoint's built-in Equation Editor.
- The circled numbers (denoting graph vertices) are rendered using the standard Wingdings font.
- The text is written in Oxford English.
- The icons for referenced papers and talks are clickable and will lead to an open-access / preprint / author's copy version of the referred work (if such copy exists). The icons depict the first page of the cited document.

FUTURE WORK

- Matrix representations
 - The 'typical' sparse matrix data structures: CSR, CRS
 - Basic updatable data structures: COO, LIL, DOK
 - Advanced updatable data structures: Hornet, faim
- Important algorithms
 - Matrix multiplication (dot, hash, heap, Gustavson)
- Advanced GraphBLAS features
 - the accumulator
 - merge/replace
 - submatrix extraction, assignment, Kronecker product