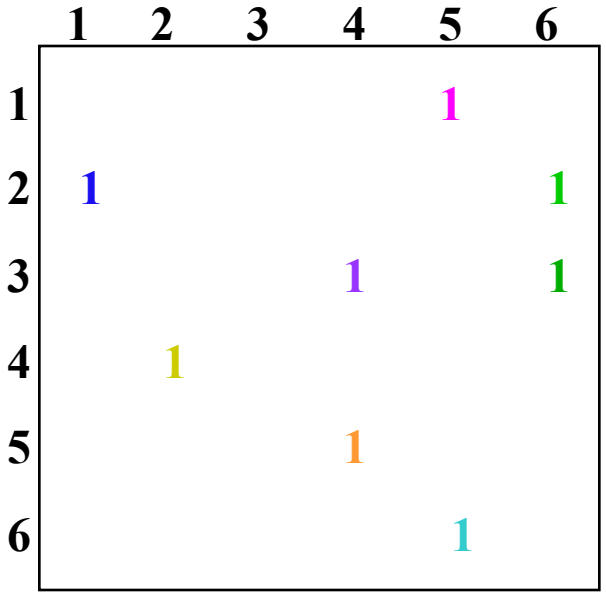
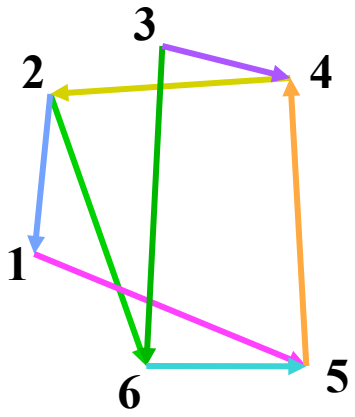
CS240A: Computation on Graphs

Graphs and Sparse Matrices

- Sparse matrix is a representation of a (sparse) graph

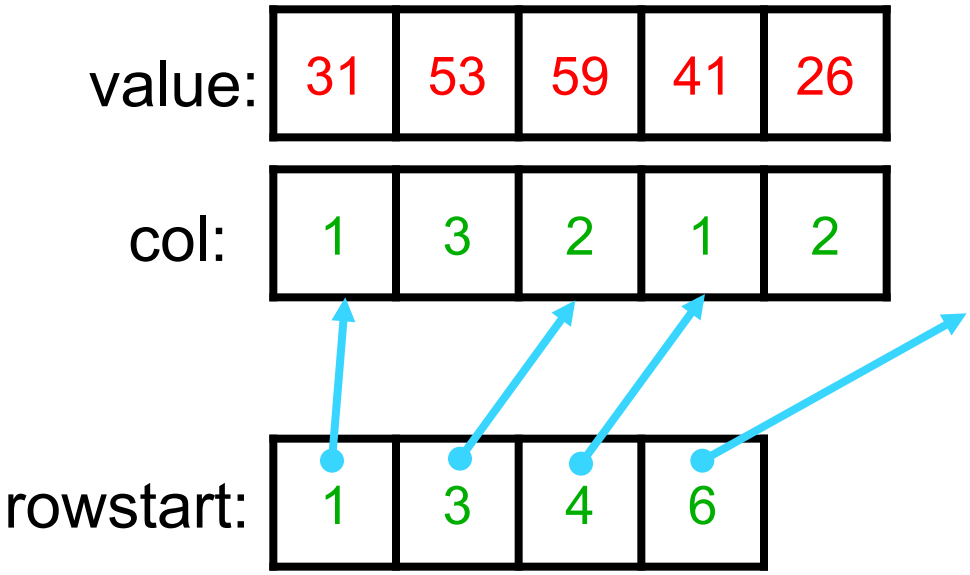




- Matrix entries can be just 1's, or edge weights
- Diagonal can represent self-loops or vertex weights
- Nnz per row (off diagonal) is vertex out-degree

Sparse matrix data structure (stored by rows, CSR)

31	0	53
0	59	0
41	26	0

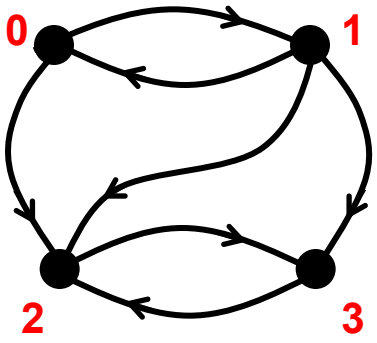


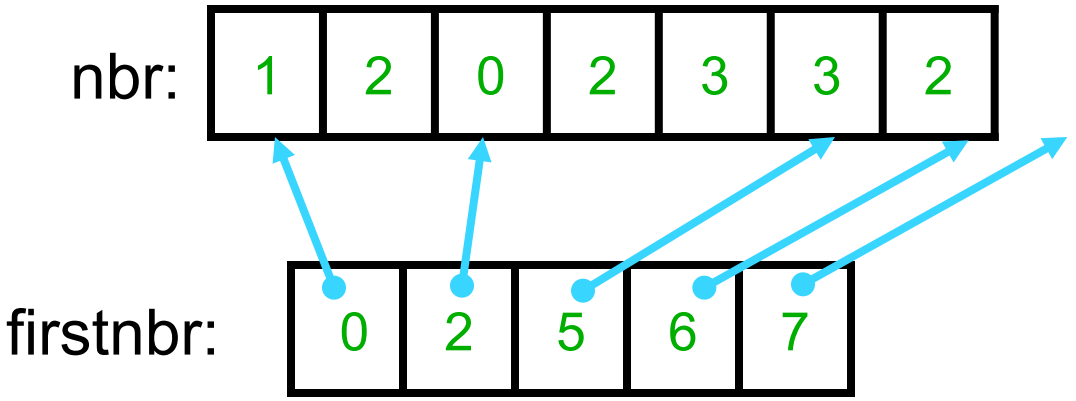
- Full storage:
  - 2-dimensional array of real or complex numbers
  - (nrows*ncols) memory

- Sparse storage:
  - compressed storage by rows (CSR)
  - three 1-dimensional arrays
  - (2*nzs + ncols + 1) memory
  - similarly, CSC

Compressed graph data structure (CSR)

Like matrix CSR, but indices & vertex numbers start at 0

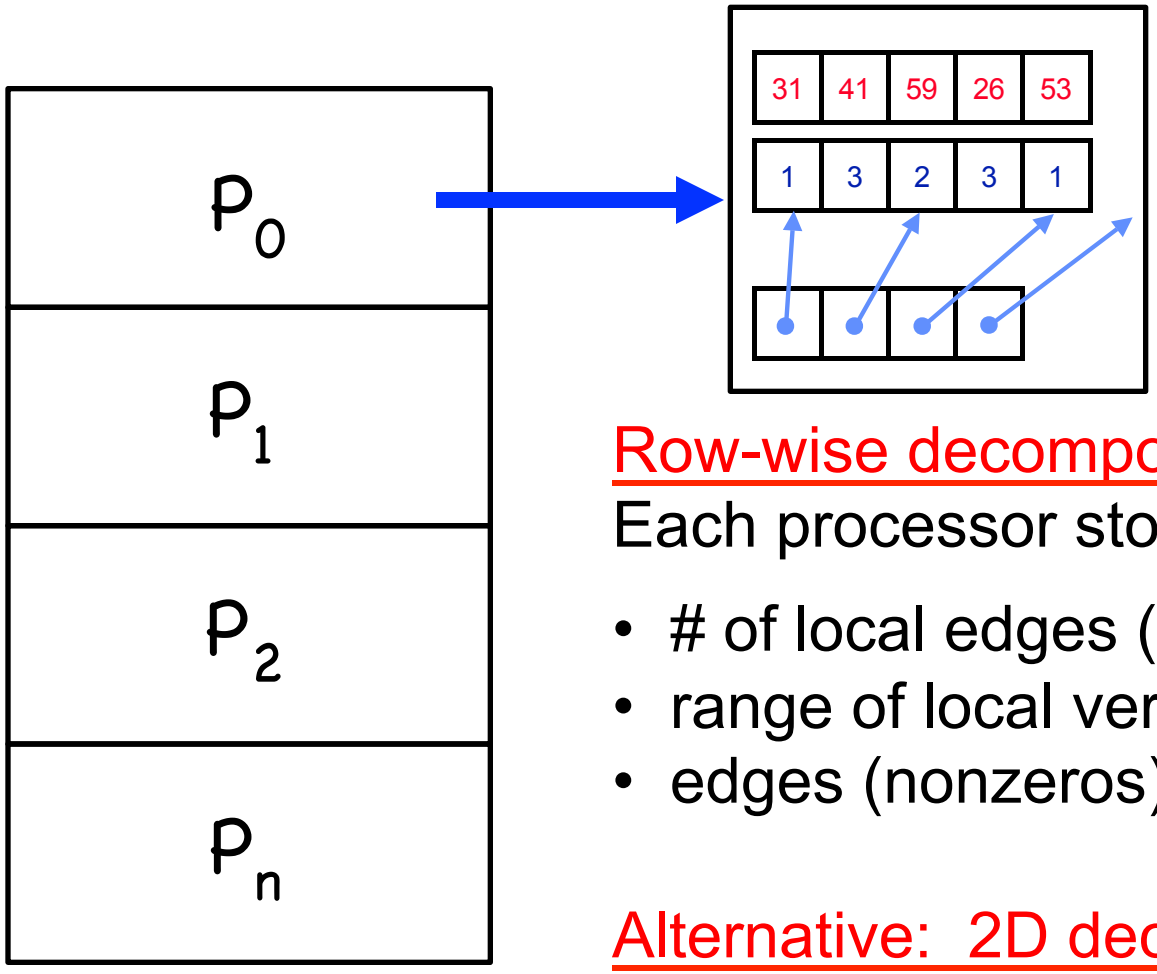




CSR graph storage:

- three 1-dimensional arrays
- digraph: ne + nv + 1 memory
- undirected graph: 2*ne + nv + 1 memory; edge {v,w} appears once for v, once for w
- firstnbr[0] = 0; for a digraph, firstnbr[nv] = ne

Graph (or sparse matrix) in distributed memory, CSR



Row-wise decomposition

Each processor stores:

- # of local edges (nonzeros)
- range of local vertices (rows)
- edges (nonzeros) in CSR form

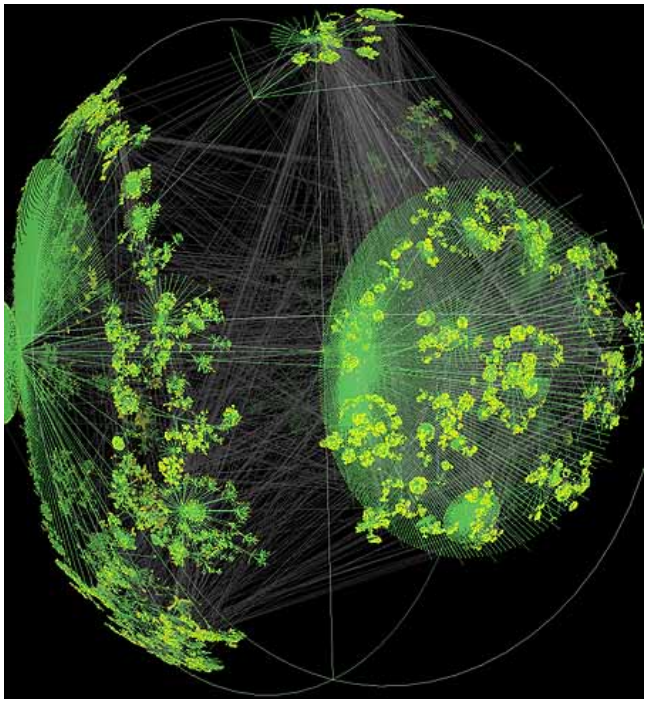
Alternative: 2D decomposition

Large graphs are everywhere…

Internet structure

Social interactions

Scientific datasets: biological, chemical, cosmological, ecological, …



WWW snapshot, courtesy Y. Hyun

Yeast protein interaction network, courtesy H. Jeong

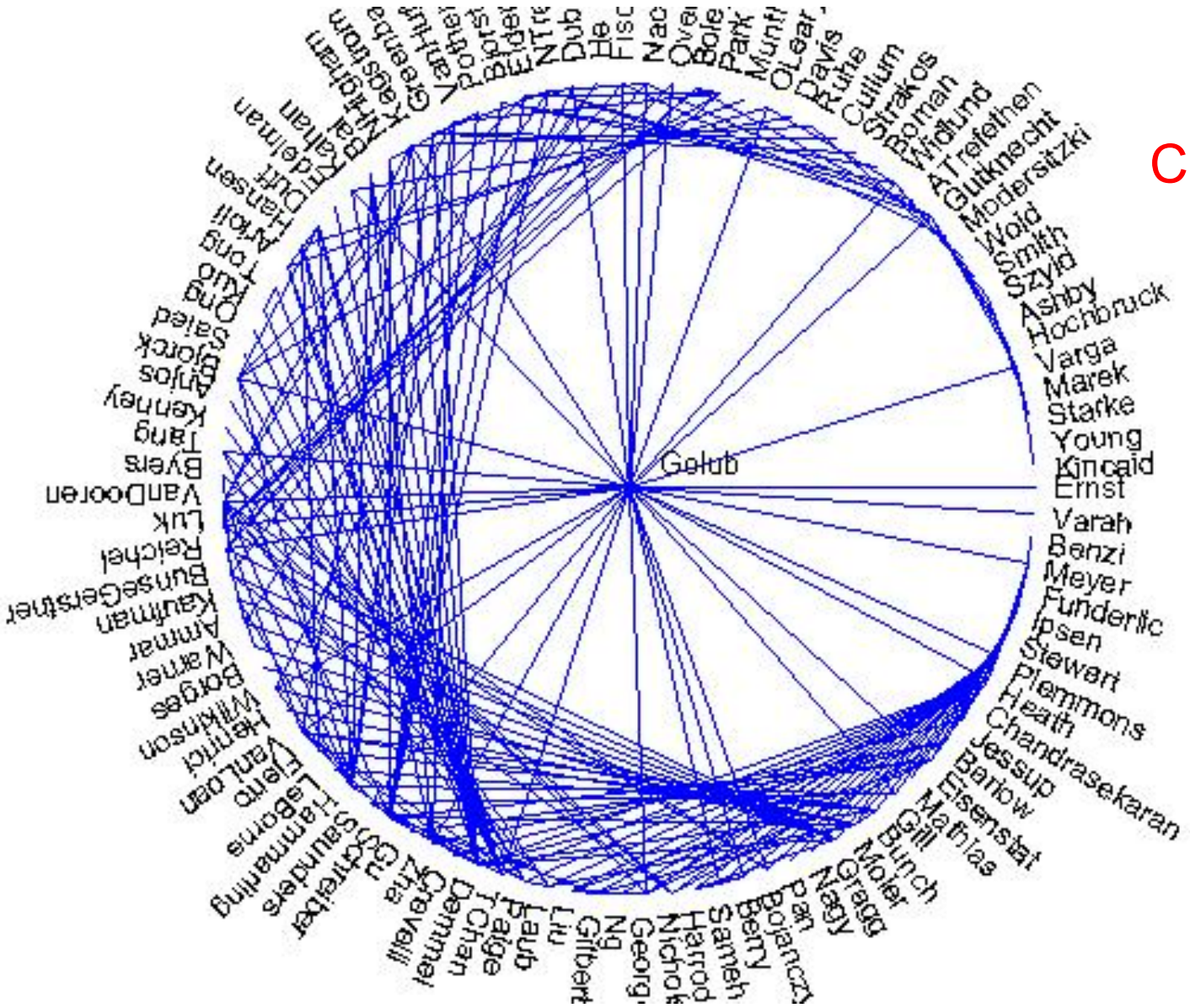
UCSB

Node-to-node searches in graphs …

- Who are my friends' friends?
- How many hops from A to B? (six degrees of Kevin Bacon)
- What's the shortest route to Las Vegas?
- Am I related to Abraham Lincoln?
- Who likes the same movies I do, and what other movies do they like?
- . . .

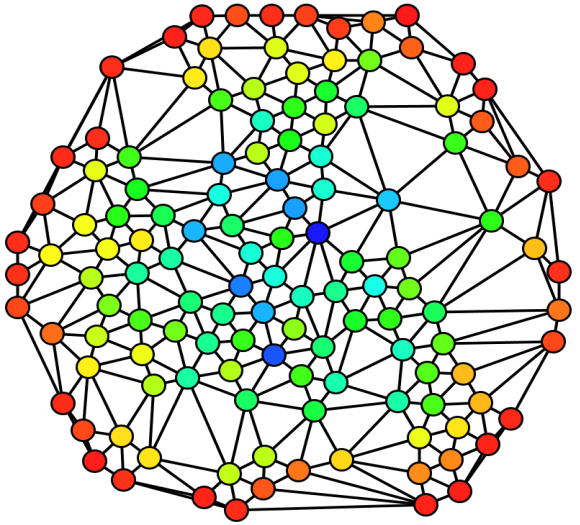
- See breadth-first search example slides

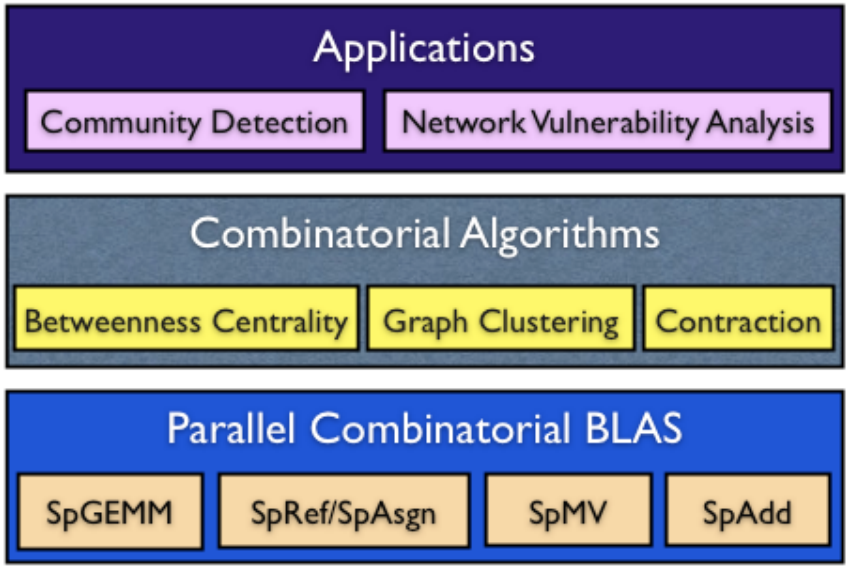
Social Network Analysis in Matlab: 1993



Co-author graph from 1993 Householder symposium

Social network analysis





A typical software stack for an application enabled with the Combinatorial BLAS

Betweenness Centrality (BC)

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

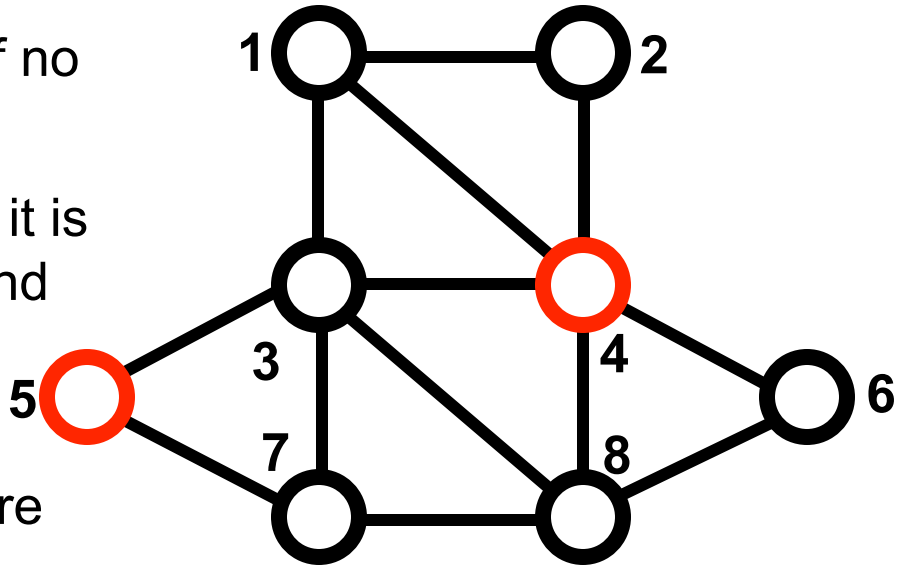
$$C_B(v) = \sum_{\substack{s \neq v \neq t \in V \\ s \neq t}} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

Brandes' algorithm

UCSB

A graph problem: Maximal Independent Set

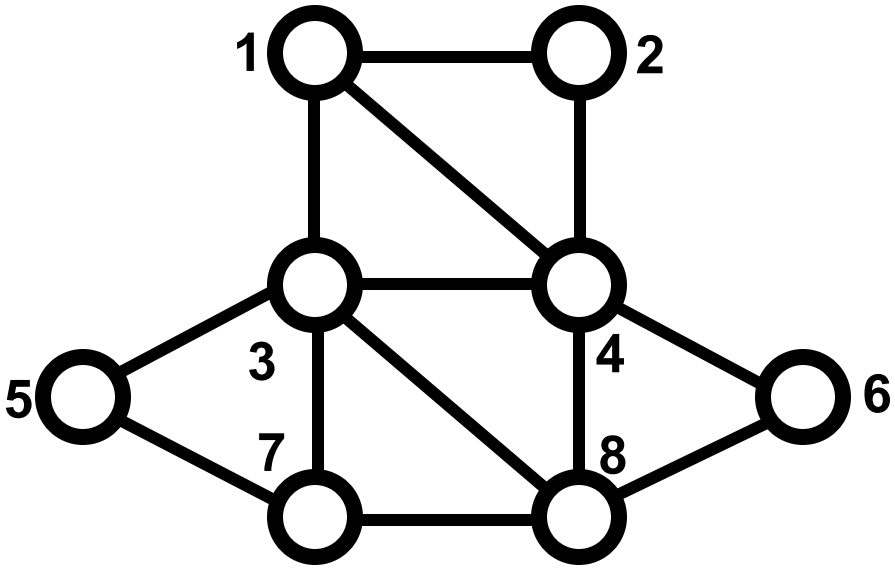
- Graph with vertices V = {1,2,…,n}
- A set S of vertices is independent if no two vertices in S are neighbors.
- An independent set S is maximal if it is impossible to add another vertex and stay independent
- An independent set S is maximum if no other independent set has more vertices
- Finding a *maximum* independent set is intractably difficult (NP-hard)
- Finding a *maximal* independent set is easy, at least on one processor.



The set of red vertices
S = {4, 5} is *independent*
and is *maximal*
but not *maximum*

Sequential Maximal Independent Set Algorithm

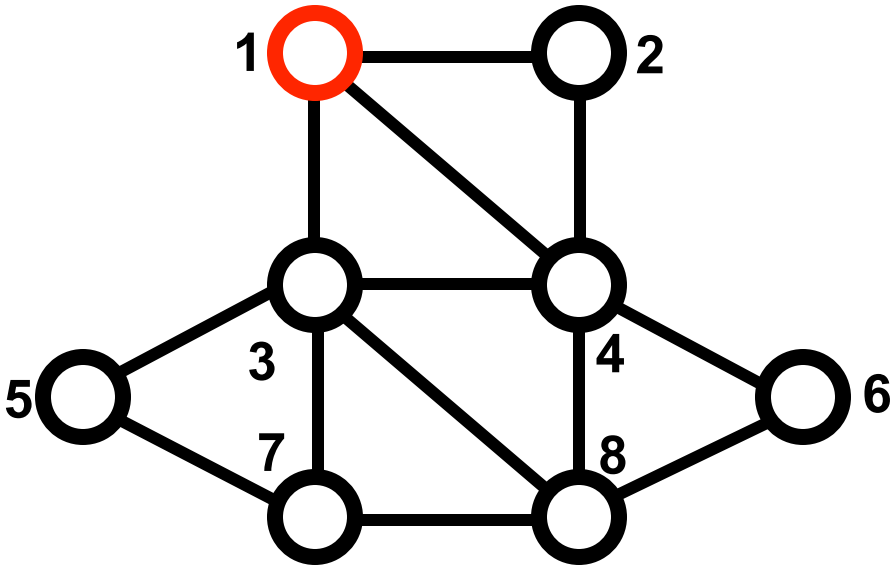
1. S = empty set;
2. for vertex v = 1 to n {
3. if (v has no neighbor in S) {
4. add v to S
5. }
6. }



S = { }

Sequential Maximal Independent Set Algorithm

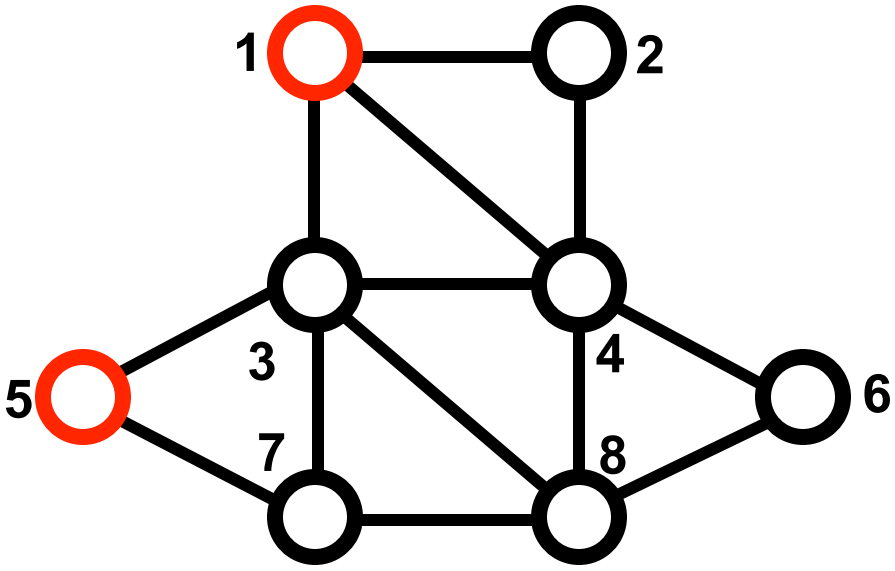
```
1.  S = empty set;
2.  for  vertex v = 1 to n {
3.      if (v has no neighbor in S) {
4.          add v to S
5.      }
6.  }
```



S = { 1 }

Sequential Maximal Independent Set Algorithm

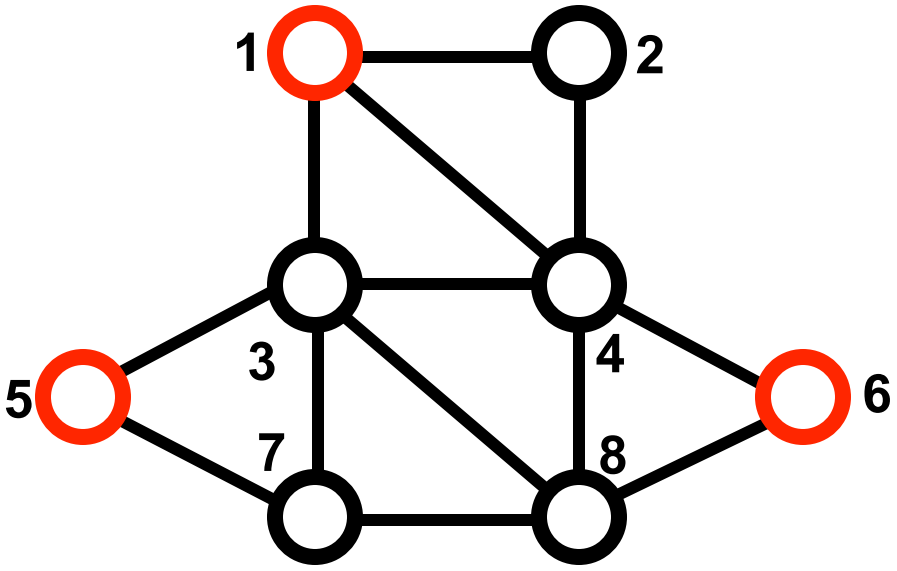
```
1.  S = empty set;
2.  for  vertex v = 1 to n {
3.      if (v has no neighbor in S) {
4.          add v to S
5.      }
6.  }
```



S = { 1, 5 }

Sequential Maximal Independent Set Algorithm

```
1.  S = empty set;
2.  for  vertex v = 1 to n {
3.      if (v has no neighbor in S) {
4.          add v to S
5.      }
6.  }
```

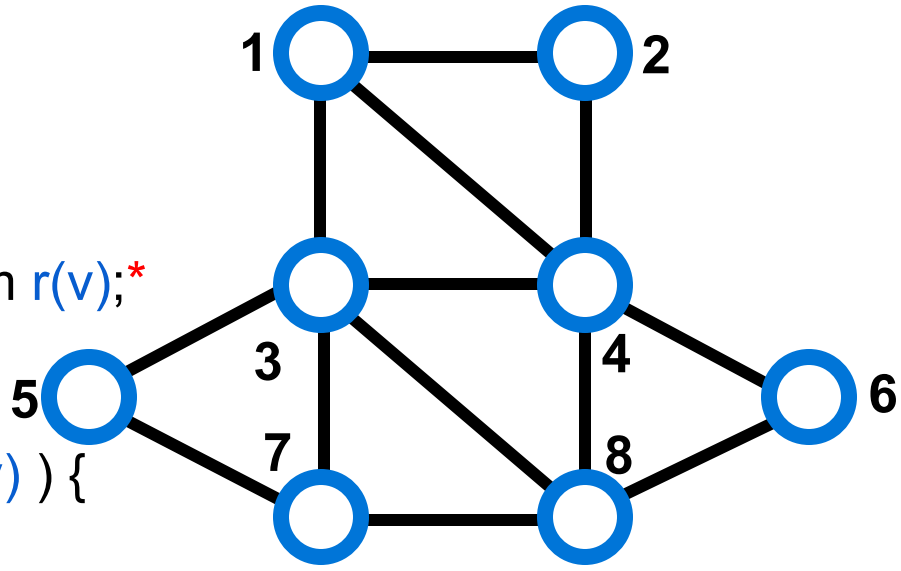


S = { 1, 5, 6 }

work **~ O(n), but** ***span*** **~O(n) and** ***parallelism*** **~O(1)**

Parallel, Randomized MIS Algorithm *[Luby]*

```
1.  S = empty set;  C = V;
2.  while  C  is not empty {
3.      label each v in C with a random r(v);*
4.      for all v in C in parallel {
5.          if r(v) < min( r(neighbors of v) ) {
6.              move v from C to S;
7.              remove neighbors of v from C;
8.          }
9.      }
10. }
```



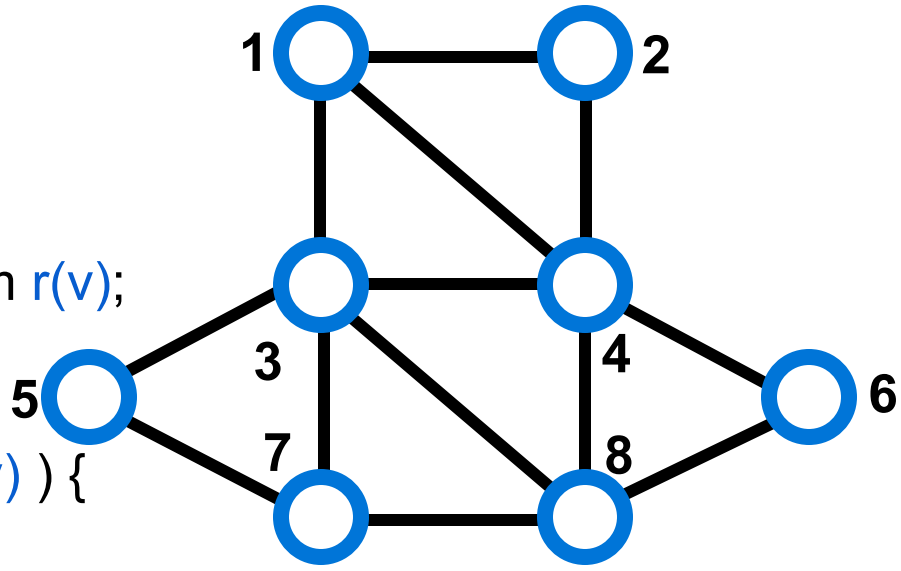
S = { }

C = { 1, 2, 3, 4, 5, 6, 7, 8 }

* (simplified version with some details omitted)

Parallel, Randomized MIS Algorithm *[Luby]*

```
1.  S = empty set;  C = V;
2.  while  C  is not empty {
3.      label each v in C with a random r(v);
4.      for all v in C in parallel {
5.          if r(v) < min( r(neighbors of v) ) {
6.              move v from C to S;
7.              remove neighbors of v from C;
8.          }
9.      }
10. }
```

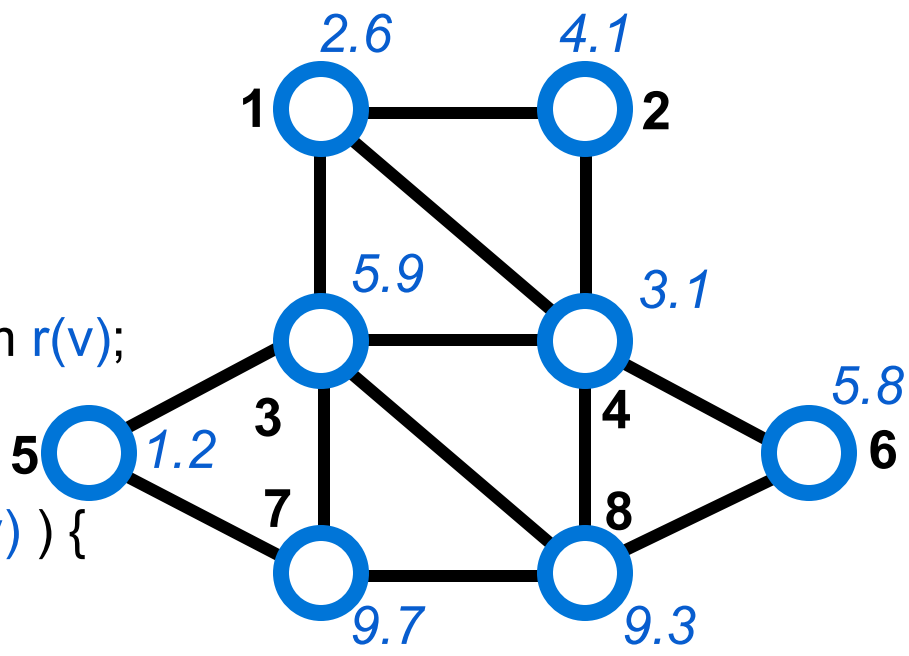


S = { }

C = { 1, 2, 3, 4, 5, 6, 7, 8 }

Parallel, Randomized MIS Algorithm *[Luby]*

```
1.  S = empty set;  C = V;
2.  while  C  is not empty {
3.      label each v in C with a random r(v);
4.      for all v in C in parallel {
5.          if r(v) < min( r(neighbors of v) ) {
6.              move v from C to S;
7.              remove neighbors of v from C;
8.          }
9.      }
10. }
```

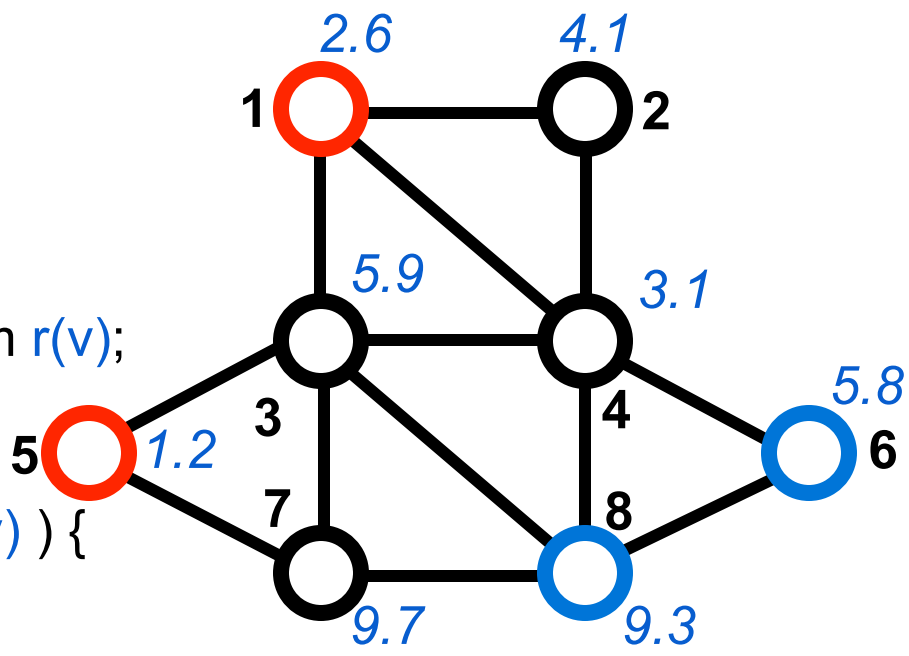


S = { }

C = { 1, 2, 3, 4, 5, 6, 7, 8 }

Parallel, Randomized MIS Algorithm *[Luby]*

```
1.  S = empty set;  C = V;
2.  while  C  is not empty {
3.      label each v in C with a random r(v);
4.      for all v in C in parallel {
5.          if r(v) < min( r(neighbors of v) ) {
6.              move v from C to S;
7.              remove neighbors of v from C;
8.          }
9.      }
10. }
```

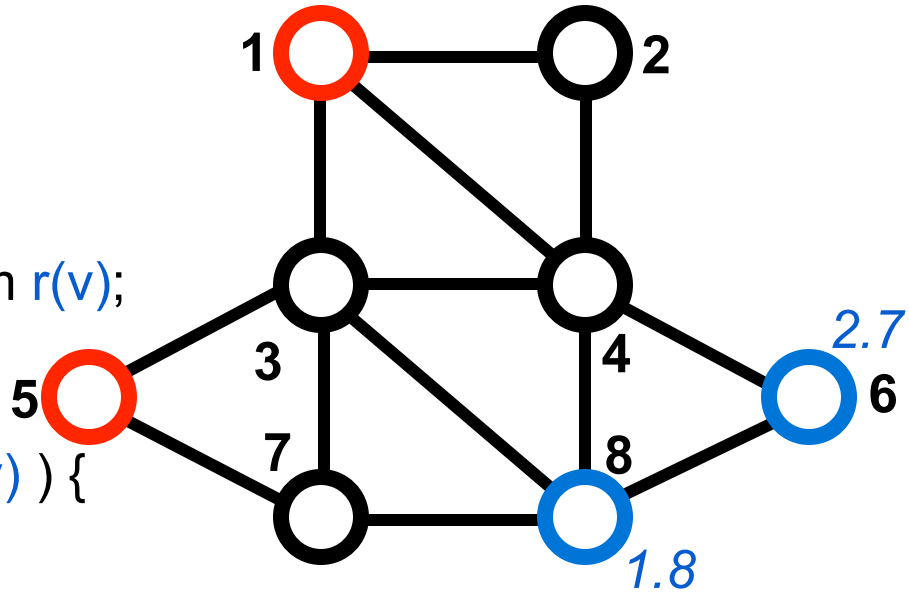


S = { 1, 5 }

C = { 6, 8 }

Parallel, Randomized MIS Algorithm *[Luby]*

```
1.  S = empty set;  C = V;
2.  while  C  is not empty {
3.      label each v in C with a random r(v);
4.      for all v in C in parallel {
5.          if r(v) < min( r(neighbors of v) ) {
6.              move v from C to S;
7.              remove neighbors of v from C;
8.          }
9.      }
10. }
```

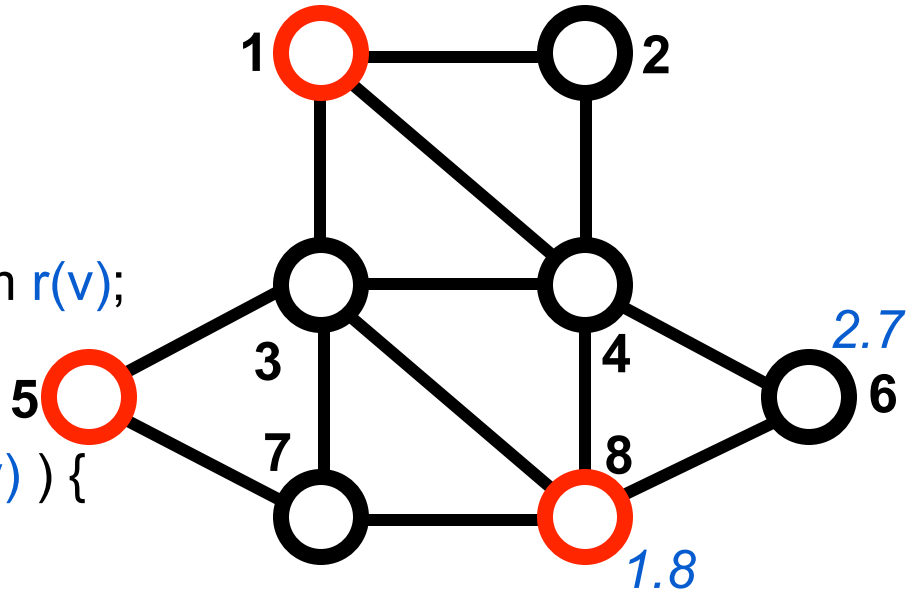


S = { 1, 5 }

C = { 6, 8 }

Parallel, Randomized MIS Algorithm *[Luby]*

```
1.  S = empty set;  C = V;
2.  while  C  is not empty {
3.      label each v in C with a random r(v);
4.      for all v in C in parallel {
5.          if r(v) < min( r(neighbors of v) ) {
6.              move v from C to S;
7.              remove neighbors of v from C;
8.          }
9.      }
10. }
```

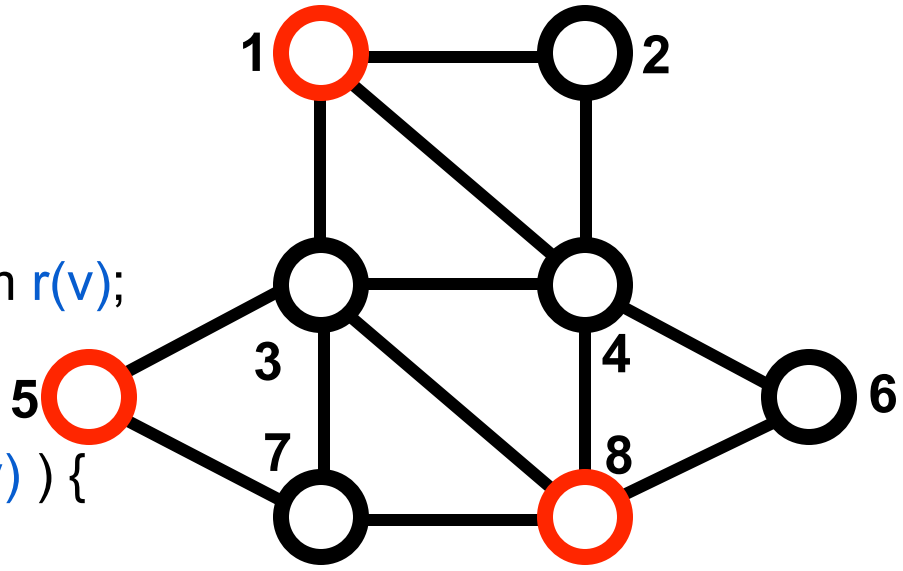


S = { 1, 5, 8 }

C = { }

Parallel, Randomized MIS Algorithm *[Luby]*

```
1.  S = empty set;  C = V;
2.  while  C  is not empty {
3.      label each v in C with a random r(v);
4.      for all v in C in parallel {
5.          if r(v) < min( r(neighbors of v) ) {
6.              move v from C to S;
7.              remove neighbors of v from C;
8.          }
9.      }
10. }
```



Theorem: This algorithm "very probably" finishes within O(log n) rounds.

work **~ O(n log n), but** ***span*** **~O($\log^2$ n), so** ***parallelism*** **~O(n/log n)**

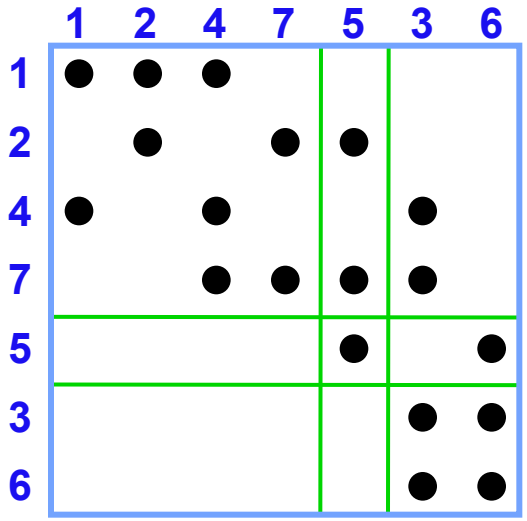
Connected components of undirected graph

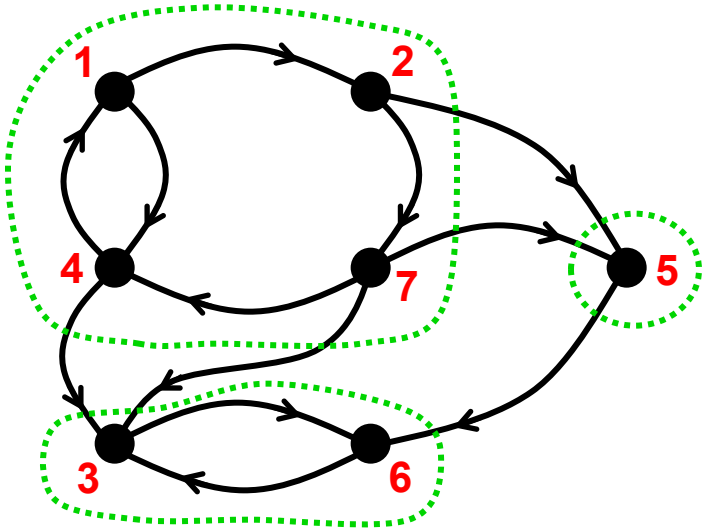
- Sequential: use any search (BFS, DFS, etc.); work O(nv+ne):

```
1.  for  vertex v = 1 to n
2.       if (v is not labeled)
3.            search from v to label a component
```

- Parallel:
  - Various heuristics using BFS, e.g. “bully algorithm” (Berry et al. paper); most with worst-case span O(n) but okay in practice.
  - Linking / pointer-jumping algorithms with theoretical span O(log n) or $O(\log^2 n)$ (Greiner paper).

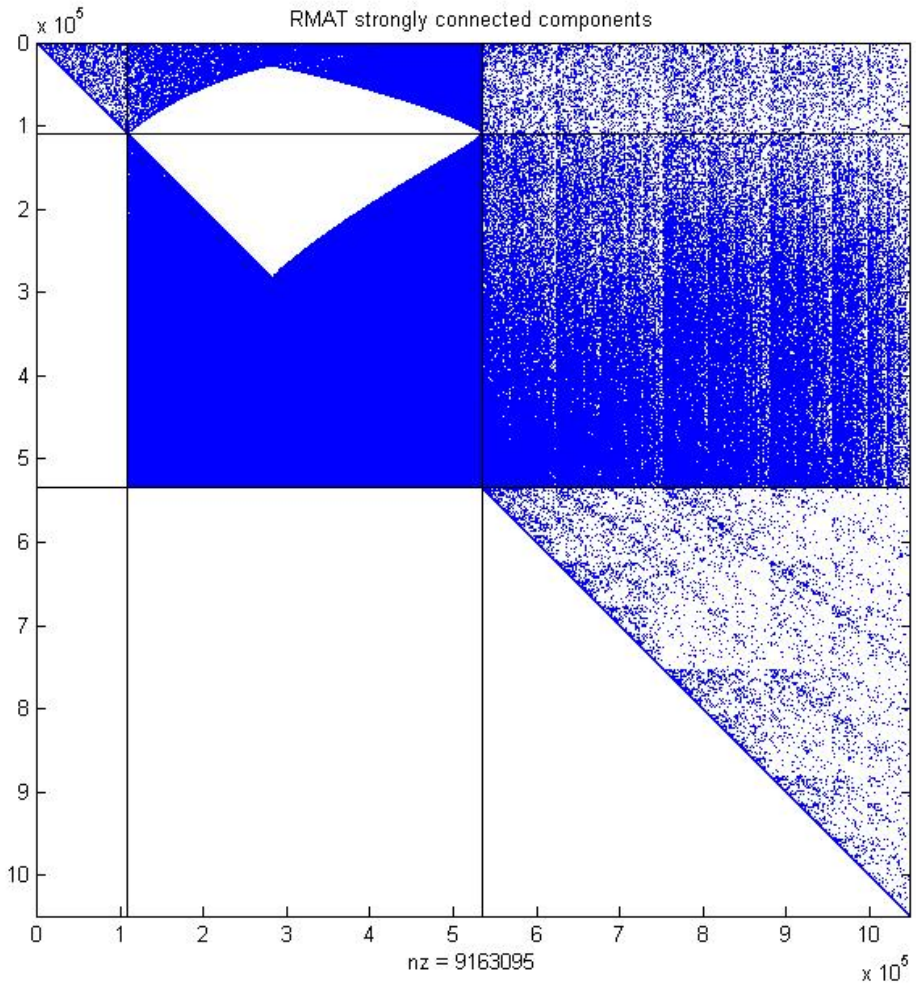
Strongly connected components





- Symmetric permutation to block triangular form
- Find P in linear time by depth-first search [Tarjan]

Strongly Connected Components



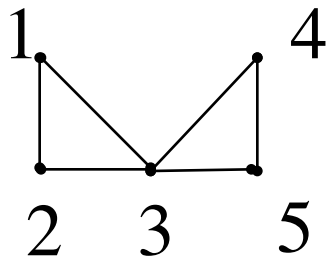
Strongly connected components of directed graph

- Sequential: depth-first search (Tarjan paper); work O(nv+ne).
- DFS seems to be inherently sequential.
- Parallel: divide-and-conquer and BFS (Fleischer et al. paper); worst-case span O(n) but good in practice on many graphs.

Laplacian Matrix

- *Definition*: The Laplacian matrix L(G) of a graph G(N,E) is an |N| by |N| symmetric matrix, with one row and column for each node. It is defined by
  - L(G) (i,i) = degree of node I (number of incident edges)
  - L(G) (i,j) = -1 if i != j and there is an edge (i,j)
  - L(G) (i,j) = 0 otherwise

G =



$$\mathbf{L(G) =} \begin{pmatrix} 2 & -1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ -1 & -1 & 4 & -1 & -1 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & -1 & 2 \end{pmatrix}$$

Properties of Laplacian Matrix

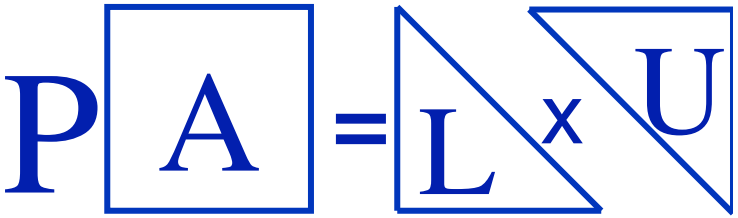
- *Theorem:* L(G) has the following properties
  - L(G) is symmetric.
    - This implies the eigenvalues of L(G) are real, and its eigenvectors are real and orthogonal.
  - Rows of L sum to zero:
    - Let e = $[1,\ldots,1]^T$, i.e. the column vector of all ones. Then L(G)*e=0.
  - The eigenvalues of L(G) are nonnegative:
    - $0 = \lambda_1 <= \lambda_2 <= \ldots <= \lambda_n$
  - The number of connected components of G is equal to the number of λ_i that are 0.

EXTRA SLIDES

Top 500 List (November 2010)

Top500 Benchmark:

Solve a large system of linear equations by Gaussian elimination

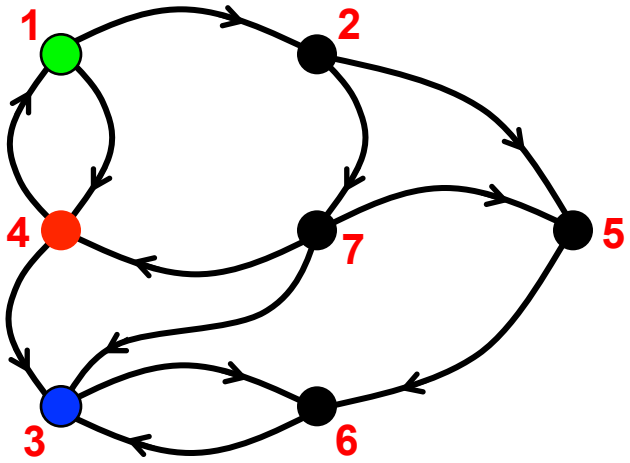


Rank	Site	Computer/Year Vendor	Cores	R_{max}
1	National Supercomputing Center in Tianjin China	Tianhe-1A - NUDT TH MPP, X5670 2.93Ghz 6C, NVIDIA GPU, FT-1000 8C / 2010 NUDT	186368	2566.00
2	DOE/SC/Oak Ridge National Laboratory United States	Jaguar - Cray XT5-HE Opteron 6-core 2.6 GHz / 2009 Cray Inc.	224162	1759.00
3	National Supercomputing Centre in Shenzhen (NSCS) China	Nebulae - Dawning TC3600 Blade, Intel X5650, NVidia Tesla C2050 GPU / 2010 Dawning	120640	1271.00
4	GSIC Center, Tokyo Institute of Technology Japan	TSUBAME 2.0 - HP ProLiant SL390s G7 Xeon 6C X5670, Nvidia GPU, Linux/Windows / 2010 NEC/HP	73278	1192.00
5	DOE/SC /LBNL/NERSC United States	Hopper - Cray XE6 12-core 2.1 GHz / 2010 Cray Inc.	153408	1054.00

Graph 500 List (November 2010)

Graph500 Benchmark:

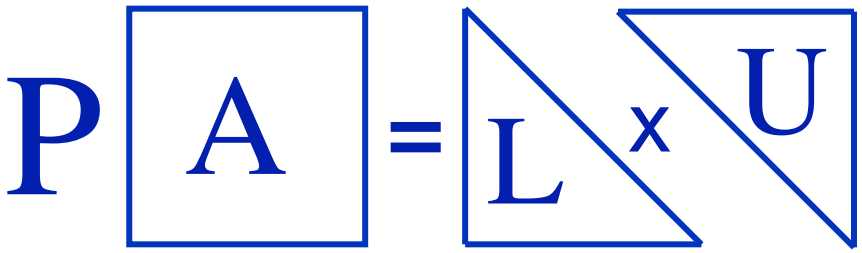
Breadth-first search in a large power-law graph



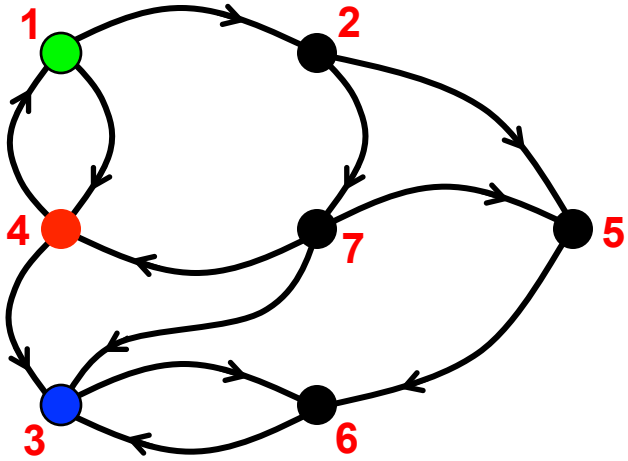
Rank	Machine	Owner	Problem Size	TEPS
1	DOE/SC/ANL Intrepid (IBM BlueGene/P, 8192 of 40960 nodes / 32k of 163840 cores)	Argonne National Laboratory	Scale 36 (Medium)	6.6 GE/s
2	Franklin (Cray XT4, 500 of 9544 nodes)	NERSC	Scale 32 (Small)	5.22 GE/s
3	cougarxmt (128 node Cray XMT)	Pacific Northwest National Laboratory	Scale 29 (Mini)	1.22 GE/s
4	graphstorm (128 node Cray XMT)	Sandia National Laboratories	Scale 29 (Mini)	1.17 GE/s
5	Endeavor (256 node, 512 core Westmere X5670 2.93, IB network)	Intel Corporation	Scale 29 (Mini)	533 ME/s
6	Erdos (64 node Cray XMT)	Oak Ridge National Laboratory	Scale 29 (Mini)	50.5 ME/s
7	Red Sky (Nehalem X5570 @2.93 GHz, IB Torus, 512 processors)	Sandia National Laboratories	Scale 28 (Toy++)	477.5 ME/s
8	Jaguar (Cray XT5-HE, 512 node subset)	Oak Ridge National Laboratory	Scale 27 (Toy+)	800 ME/s
9	Endeavor (128 node, 256 core Westmere X5670 2.93, IB network)	Intel Corporation	Scale 26 (Toy)	615.8 ME/s

Floating-Point vs. Graphs

2.5 Petaflops



6.6 Gigateps



2.5 Peta / 6.6 Giga is about 380,000!

Betweenness centrality

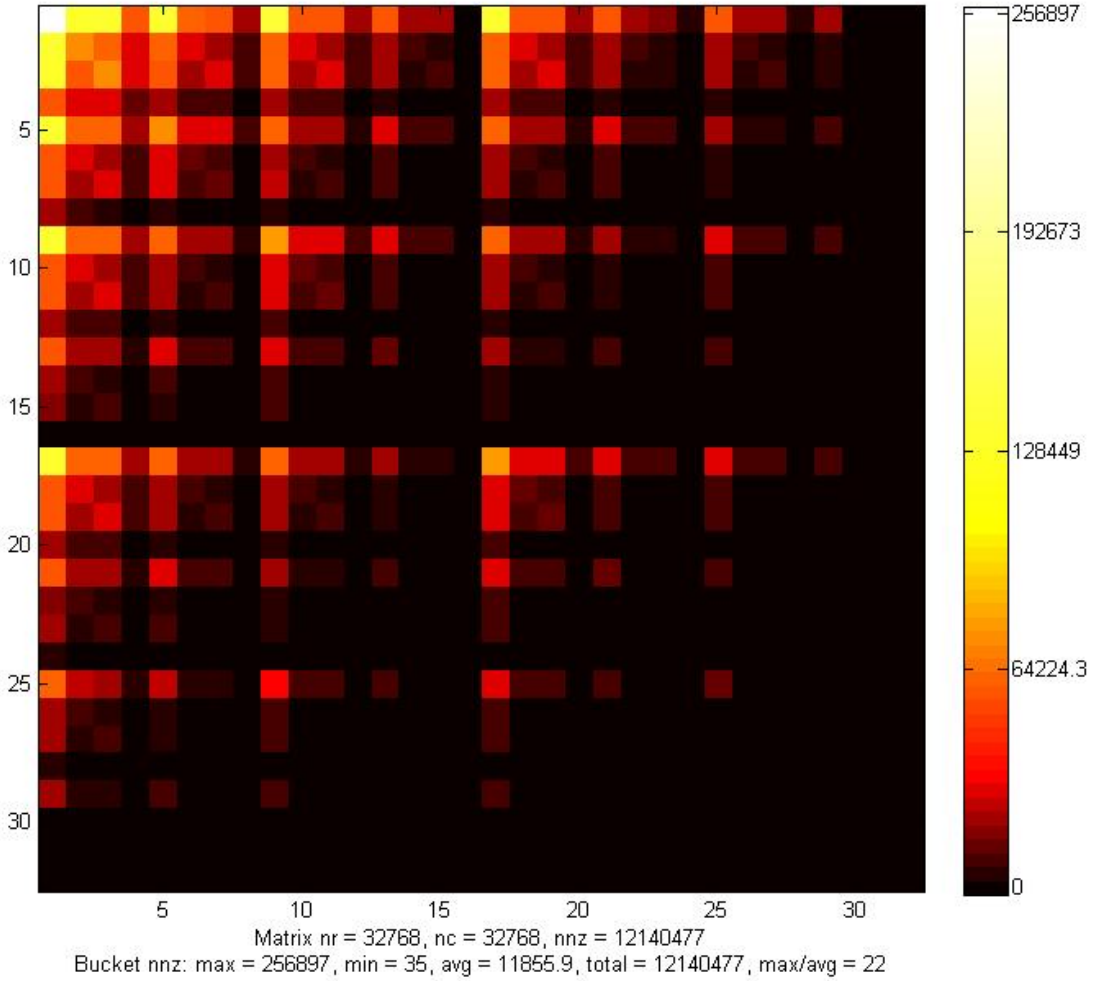
- BC example from Robinson slides
- BC sequential algorithm from Brandes paper
- BC demo
- Several potential sources of parallelism in BC

Characteristics of graphs

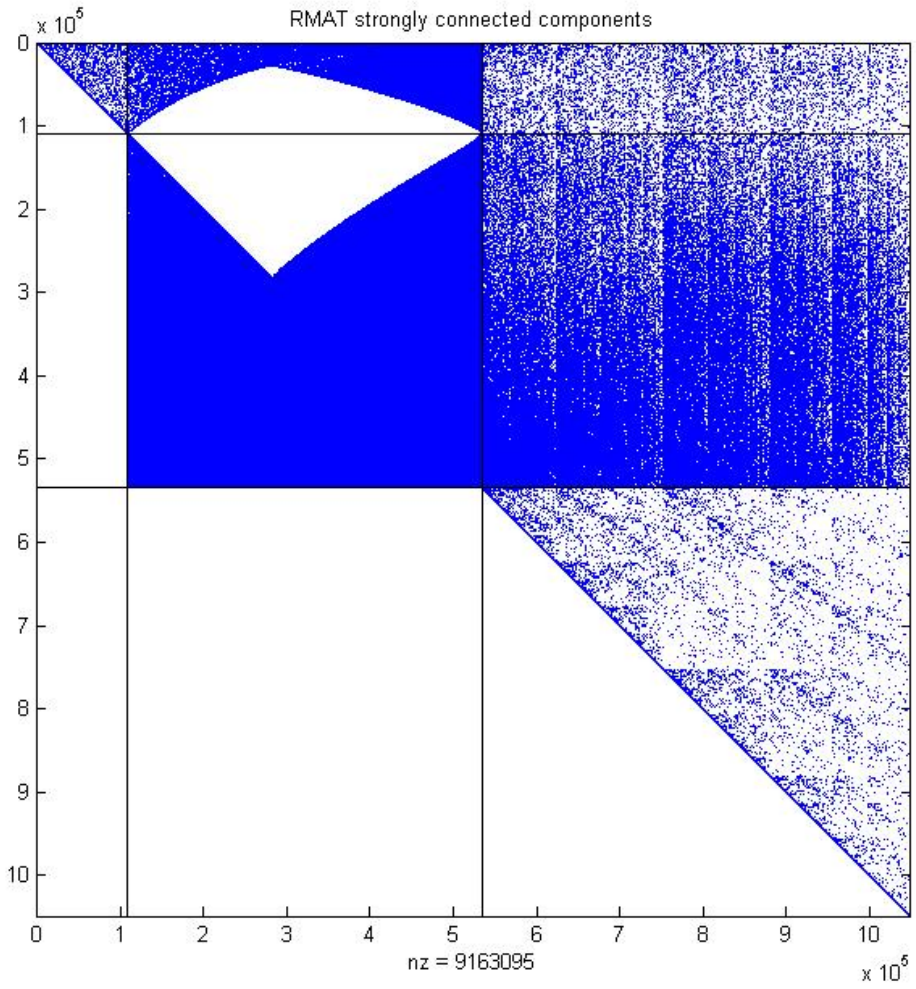
- Vertex degree histogram
- Average shortest path length
- Clustering coefficient
  - c = 3*(# triangles) / (# connected triples)
- Separator size
- Gaussian elimination fill (chordal completion size)

- Finite element meshes
- Circuit simulation graphs
- Relationship network graphs
- Erdos-Renyi random graphs
- Small world graphs
- Power law graphs
- RMAT graph generator

RMAT Approximate Power-Law Graph

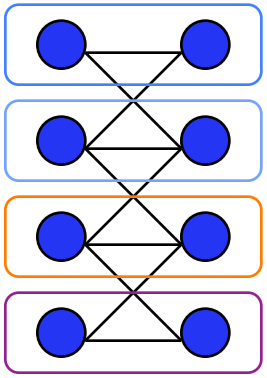


Strongly Connected Components

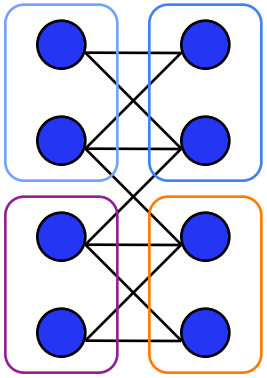


Graph partitioning

- Assigns subgraphs to processors
- Determines parallelism and locality.
- Tries to make subgraphs all same size (load balance)
- Tries to minimize edge crossings (communication).
- Exact minimization is NP-complete.

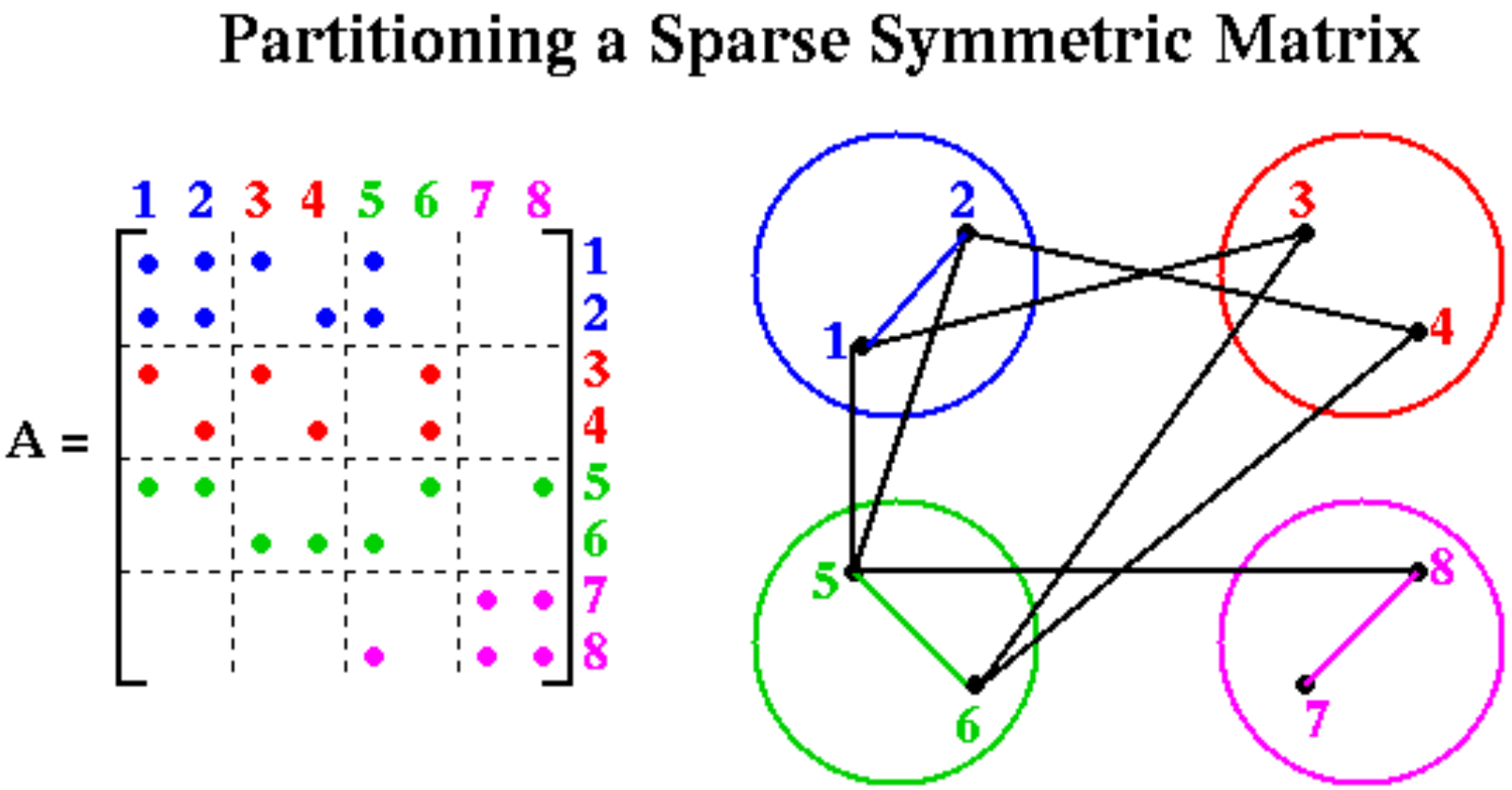


edge crossings = 6

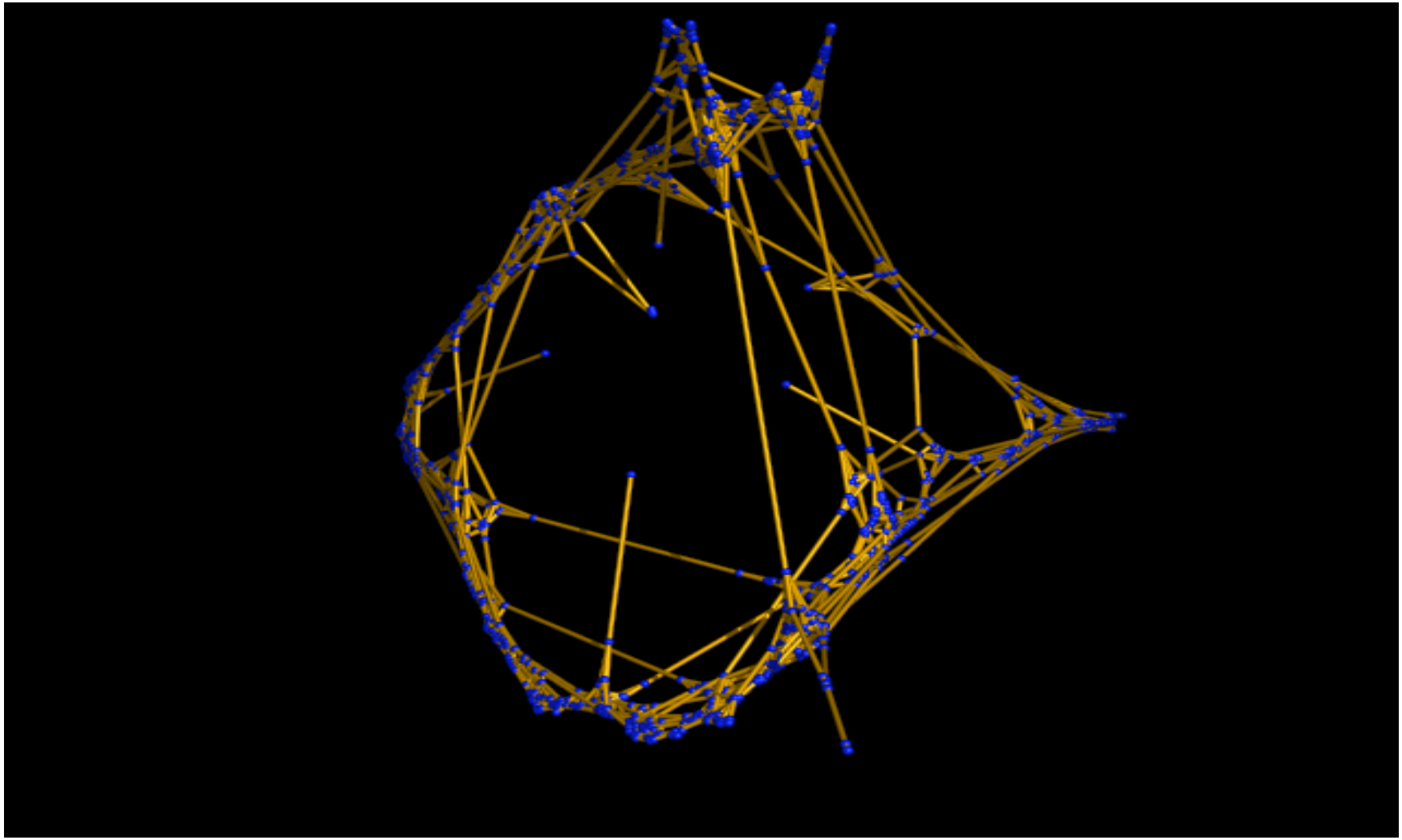


edge crossings = 10

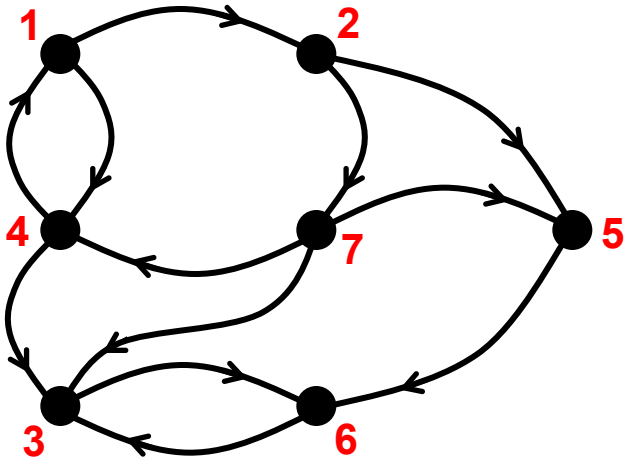
Sparse Matrix-Vector Multiplication

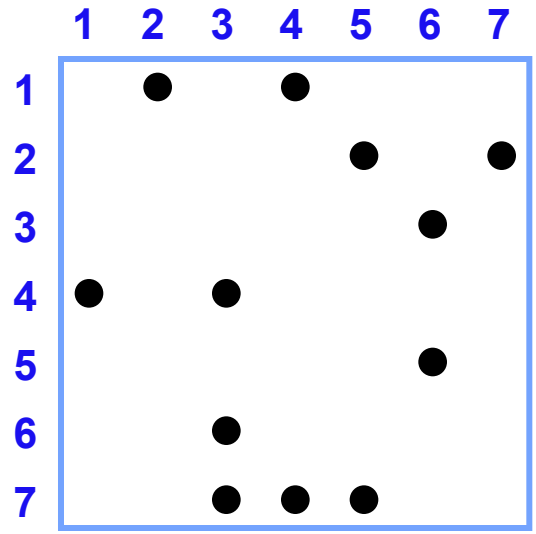


Clustering benchmark graph



Example: Web graph and matrix

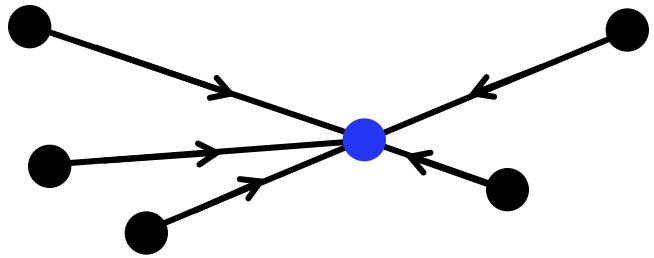




- Web page = vertex
- Link = directed edge
- Link matrix: $A_{ij} = 1$ if page i links to page j

Web graph: PageRank (Google)

[Brin, Page]

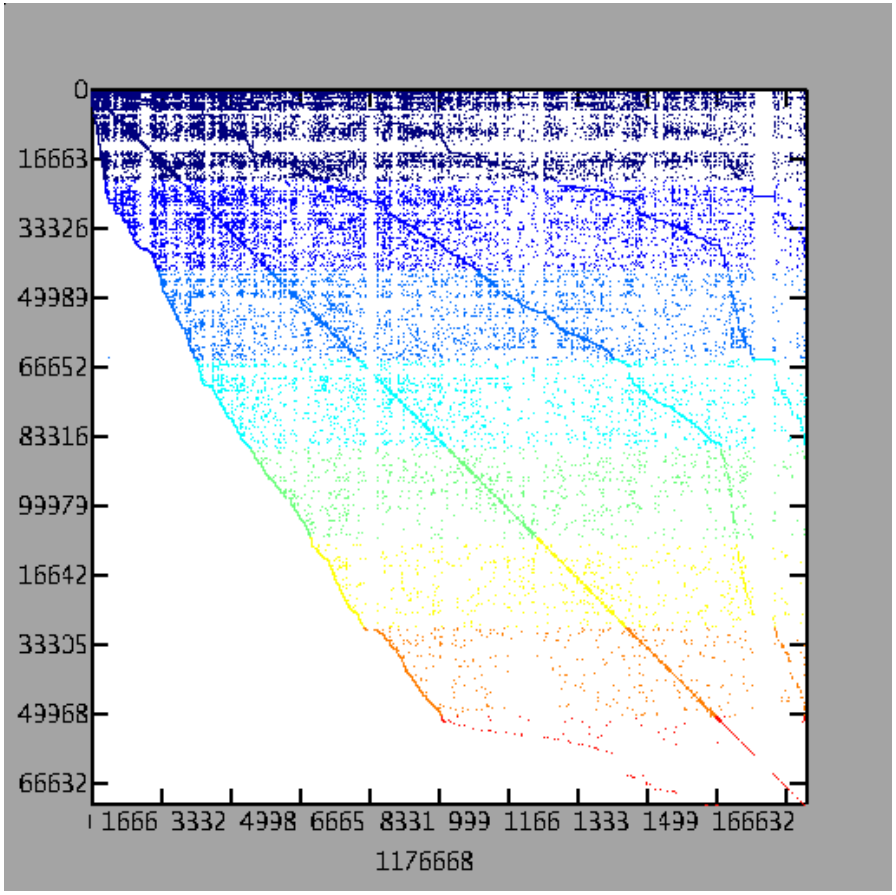


An important page is one that many important pages point to.

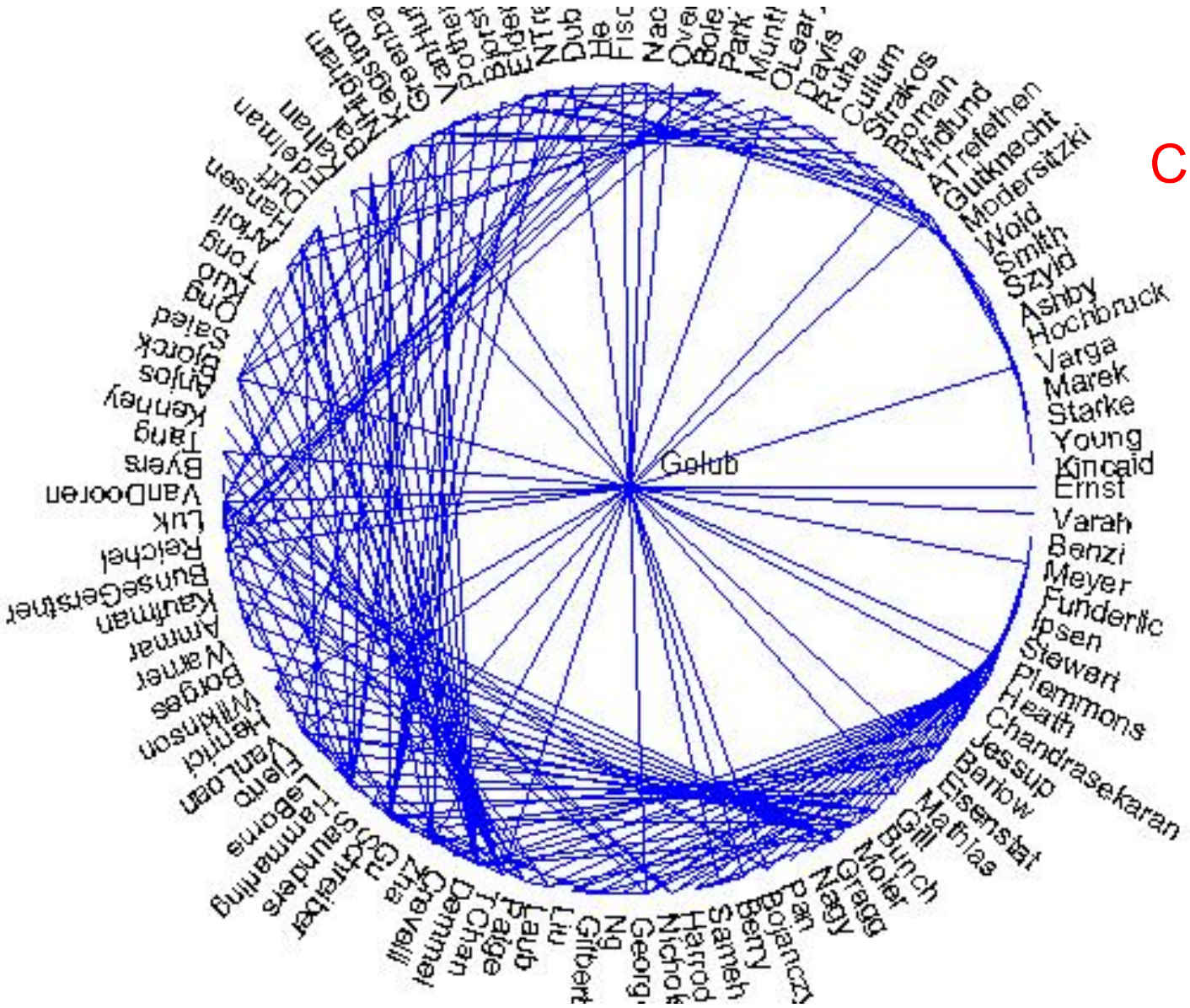
- Markov process: follow a random link most of the time; otherwise, go to any page at random.
- Importance = stationary distribution of Markov process.
- Transition matrix is p*A + (1-p)*ones(size(A)), scaled so each column sums to 1.
- Importance of page i is the i-th entry in the principal eigenvector of the transition matrix.
- But the matrix is 1,000,000,000,000 by 1,000,000,000,000.

A Page Rank Matrix

- Importance ranking of web pages
- Stationary distribution of a Markov chain
- Power method: matvec and vector arithmetic
- Matlab*P page ranking demo (from SC'03) on a web crawl of mit.edu (170,000 pages)



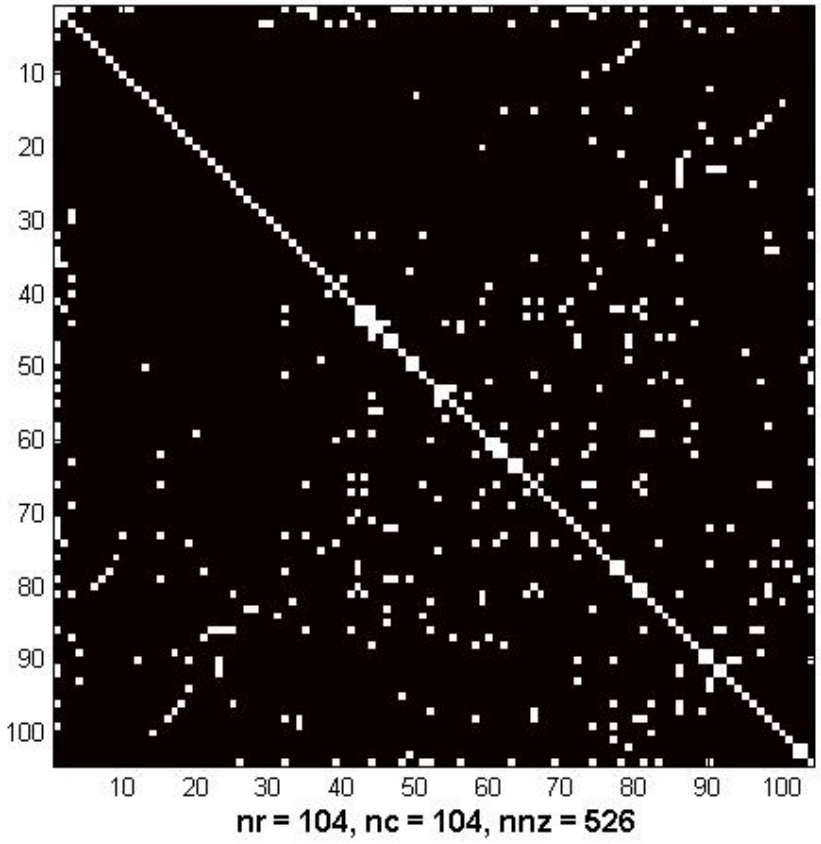
Social Network Analysis in Matlab: 1993



Co-author graph from 1993 Householder symposium

Social Network Analysis in Matlab: 1993

Sparse Adjacency Matrix



Which author has the most collaborators?

```
>>[count,author] = max(sum(A))
   count = 32
   author = 1

>>name(author,:)
   ans = Golub
```

Social Network Analysis in Matlab: 1993

Have Gene Golub and Cleve Moler ever been coauthors?

```
>> A(Golub,Moler)
ans = 0
```

No.

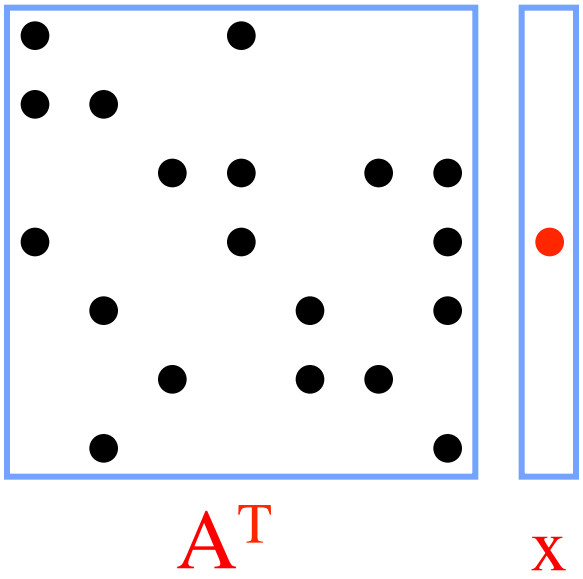
But how many coauthors do they have in common?

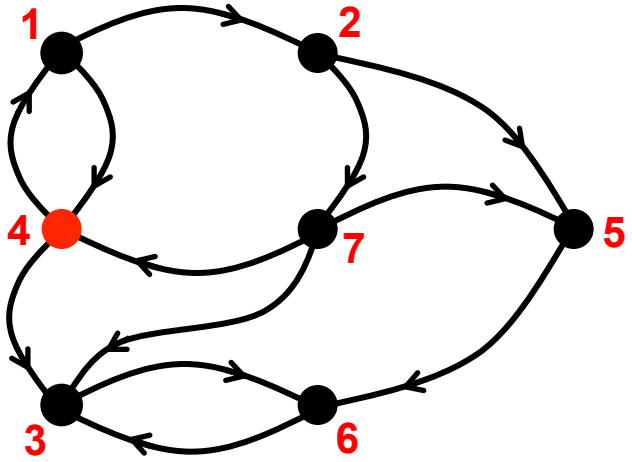
```
>> AA = A^2;
>> AA(Golub,Moler)
ans = 2
```

And who are those common coauthors?

```
>> name( find ( A(:,Golub) .* A(:,Moler) ), :)
ans =
Wilkinson
VanLoan
```

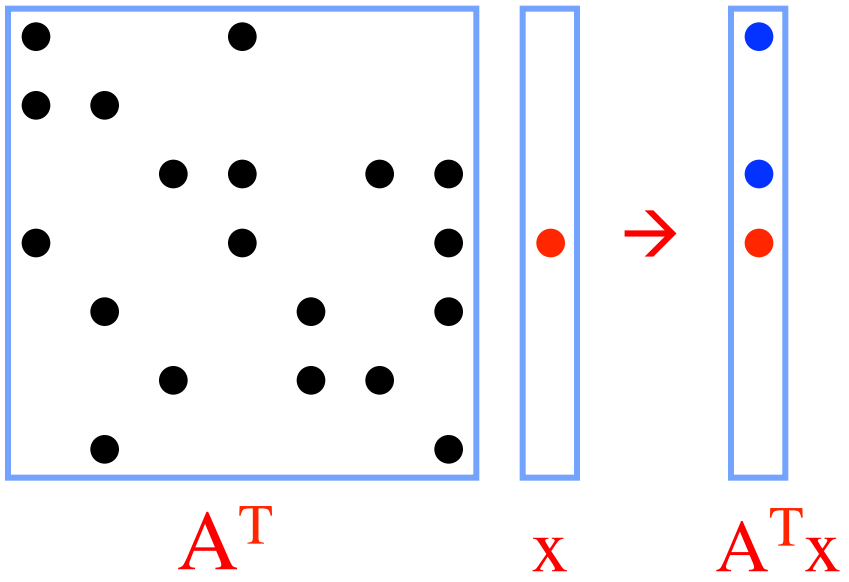
Breadth-First Search: Sparse mat * vec

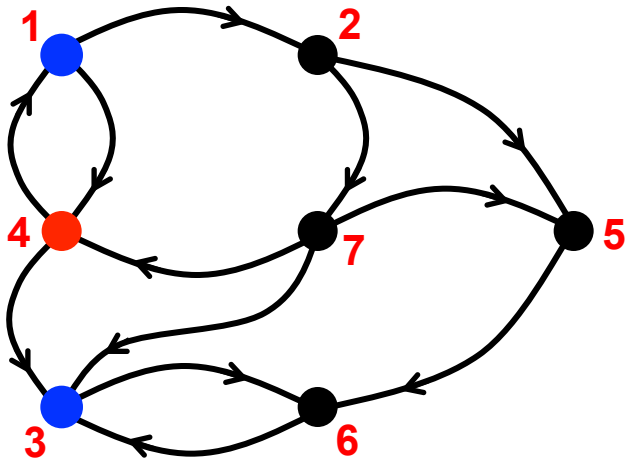




- Multiply by adjacency matrix → step to neighbor vertices
- Work-efficient implementation from sparse data structures

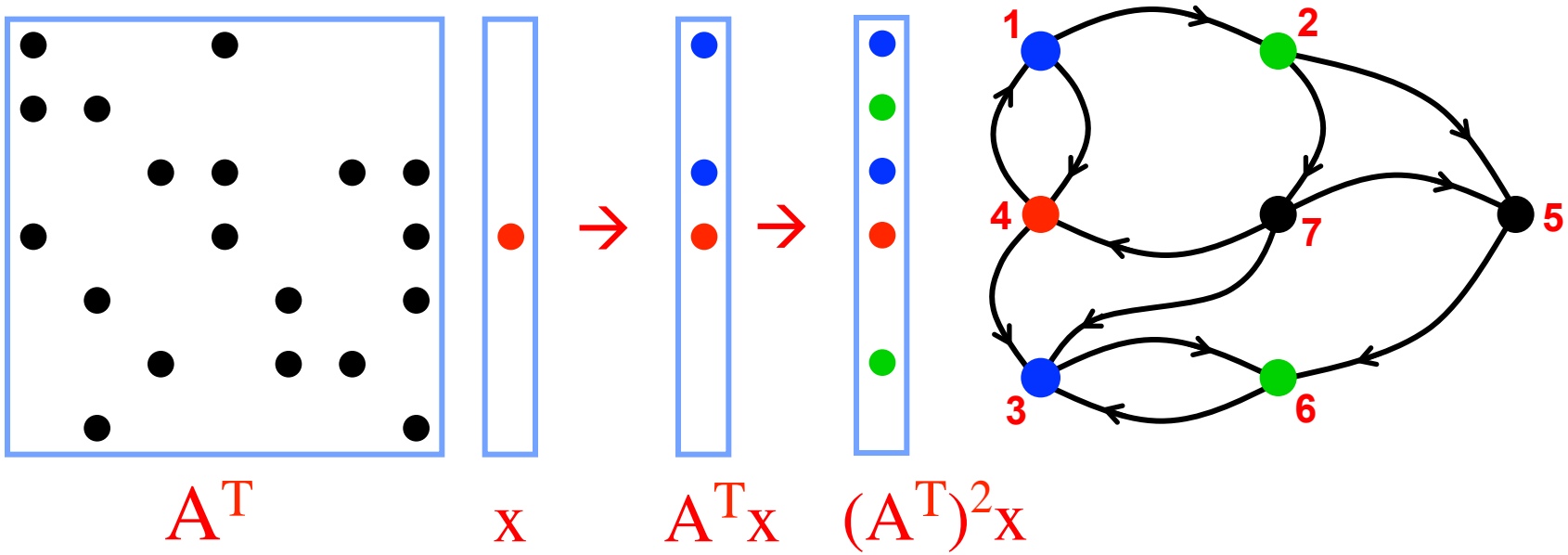
Breadth-First Search: Sparse mat * vec





- Multiply by adjacency matrix → step to neighbor vertices
- Work-efficient implementation from sparse data structures

Breadth-First Search: Sparse mat * vec



- Multiply by adjacency matrix → step to neighbor vertices
- Work-efficient implementation from sparse data structures