Data-Intensive Distributed Computing

CS 451/651 (Fall 2018)

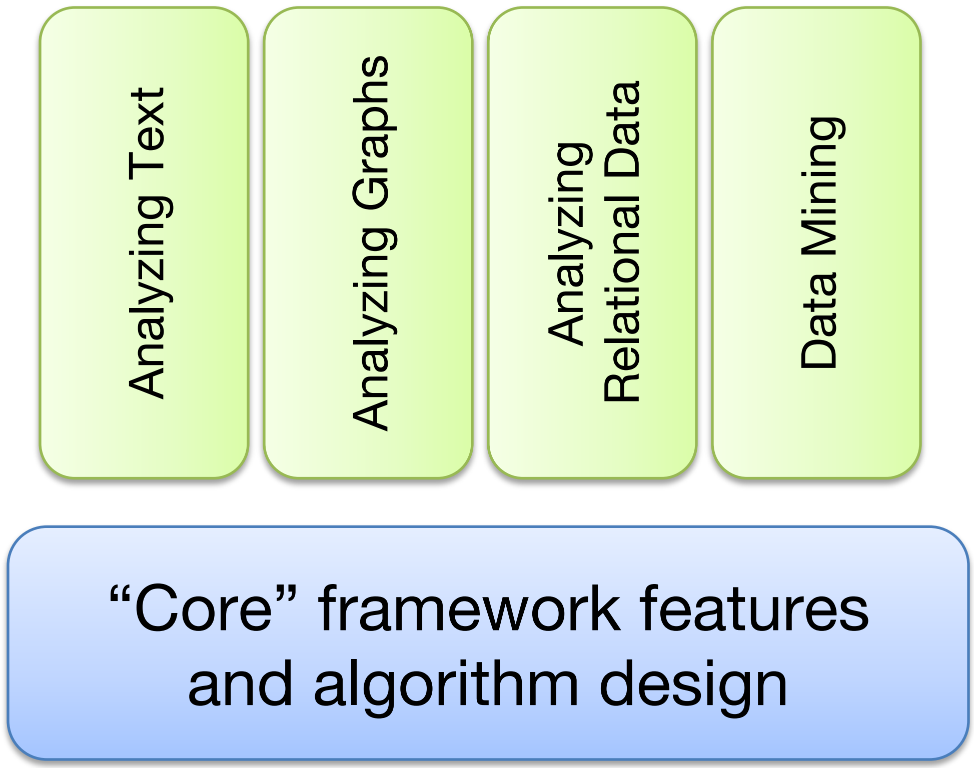
Part 4: Analyzing Graphs (1/2)

October 4, 2018

Jimmy Lin
David R. Cheriton School of Computer Science
University of Waterloo

These slides are available at http://lintool.github.io/bigdata-2018f/

Structure of the Course



What's a graph?

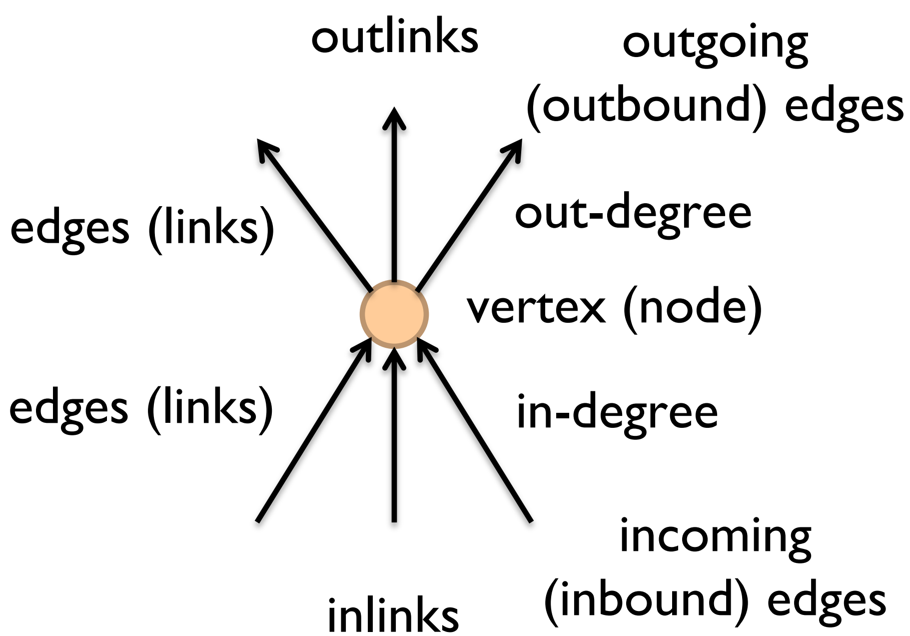
G = (V,E), where

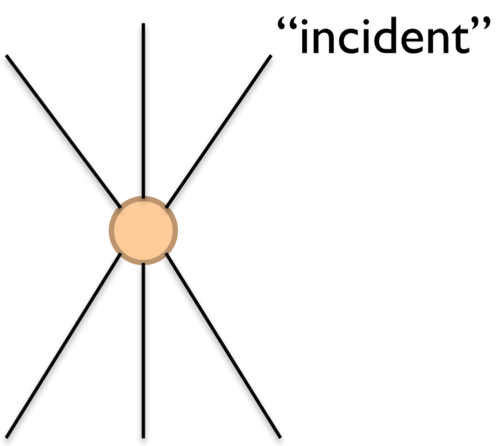
V represents the set of vertices (nodes)

E represents the set of edges (links)

Edges may be directed or undirected

Both vertices and edges may contain additional information

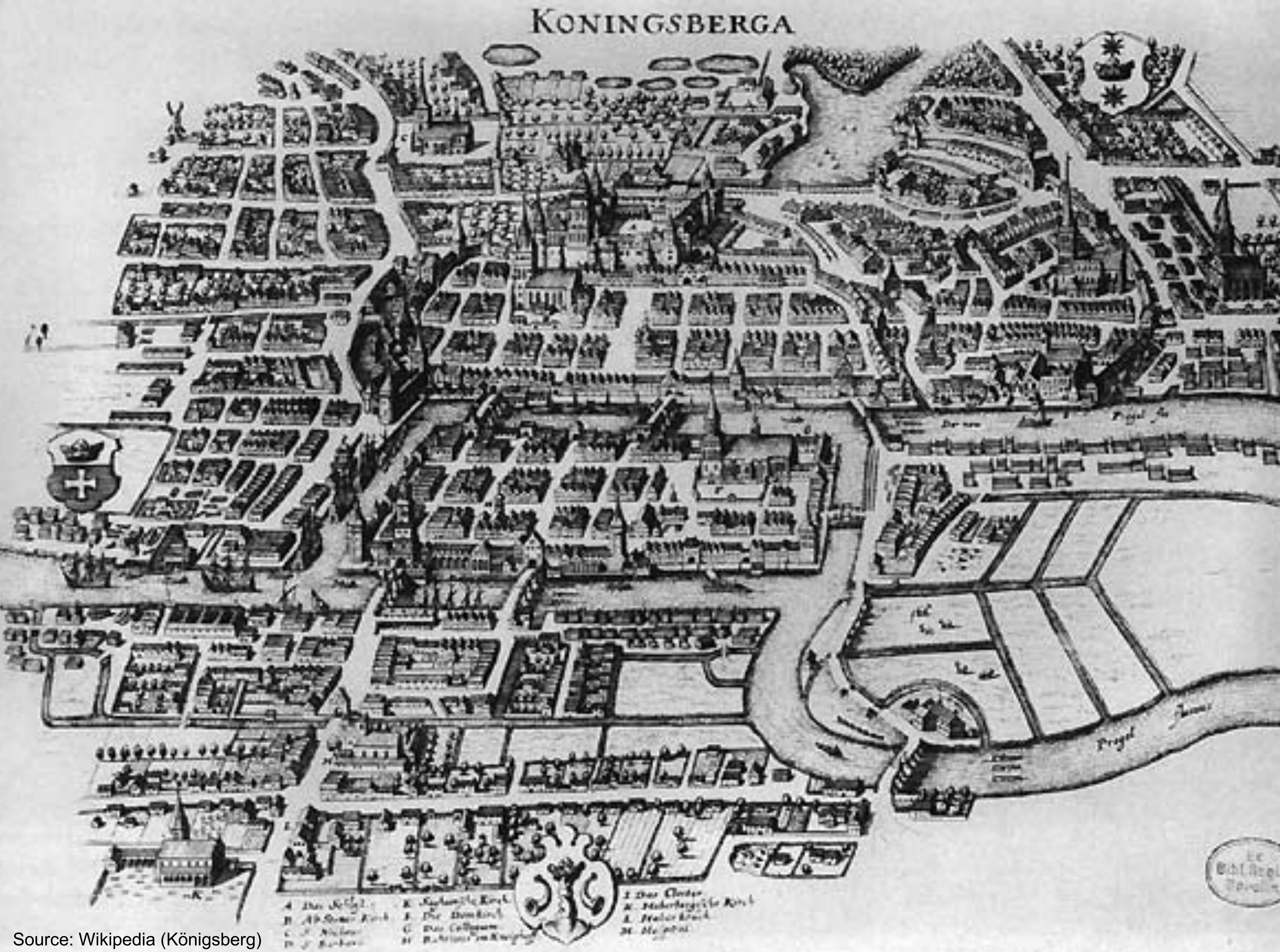




Examples of Graphs

Hyperlink structure of the web
Physical structure of computers on the Internet
Interstate highway system
Social networks

We're mostly interested in sparse graphs!



Source: Wikipedia (Königsberg)


KONINGSBERGA

Source: Wikipedia (Kaliningrad)

Some Graph Problems

Finding shortest paths

Routing Internet traffic and UPS trucks

Finding minimum spanning trees

Telco laying down fiber

Finding max flow

Airline scheduling

Identify "special" nodes and communities

Halting the spread of avian flu

Bipartite matching

match.com

Web ranking

PageRank

What makes graphs hard?

Irregular structure

Fun with data structures!

Irregular data access patterns

Fun with architectures!

Iterations

Fun with optimizations!

Graphs and MapReduce (and Spark)

A large class of graph algorithms involve:

Local computations at each node

Propagating results: “traversing” the graph

Key questions:

How do you represent graph data in MapReduce (and Spark)?

How do you traverse a graph in MapReduce (and Spark)?

Representing Graphs

Adjacency matrices

Adjacency lists

Edge lists

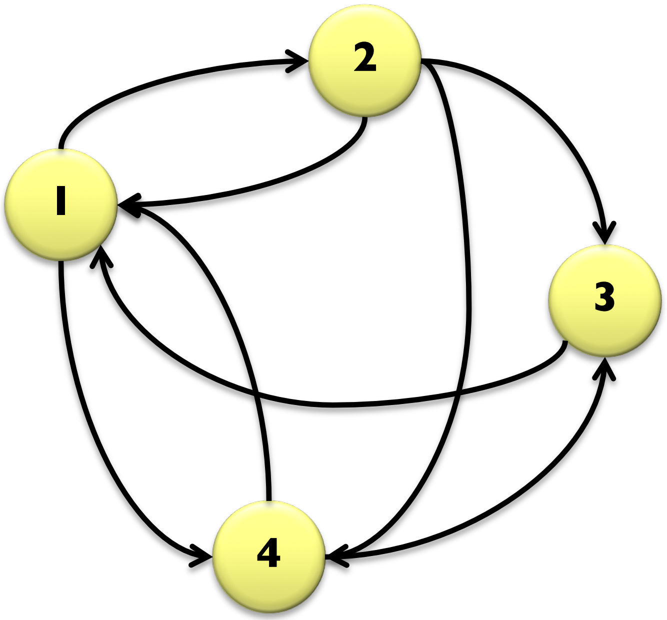
Adjacency Matrices

Represent a graph as an n x n square matrix M

$n = |V|$

$M_{ij} = 1$ iff an edge from vertex i to j

	1	**2**	**3**	**4**
1	0	1	0	1
2	1	0	1	1
3	1	0	0	0
4	1	0	1	0



Adjacency Matrices: Critique

Advantages

Amenable to mathematical manipulation

Intuitive iteration over rows and columns

Disadvantages

Lots of wasted space (for sparse matrices)

Easy to write, hard to compute

Adjacency Lists

Take adjacency matrix… and throw away all the zeros

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

1: 2, 4
2: 1, 3, 4
3: 1
4: 1, 3

Wait, where have we seen this before?

Adjacency Lists: Critique

Advantages

Much more compact representation (compress!)

Easy to compute over outlinks

Disadvantages

Difficult to compute over inlinks

Edge Lists

Explicitly enumerate all edges

	1	**2**	**3**	**4**
1	0	1	0	1
2	1	0	1	1
3	1	0	0	0
4	1	0	1	0

(1, 2)
(1, 4)
(2, 1)
(2, 3)
(2, 4)
(3, 1)
(4, 1)
(4, 3)

Edge Lists: Critique

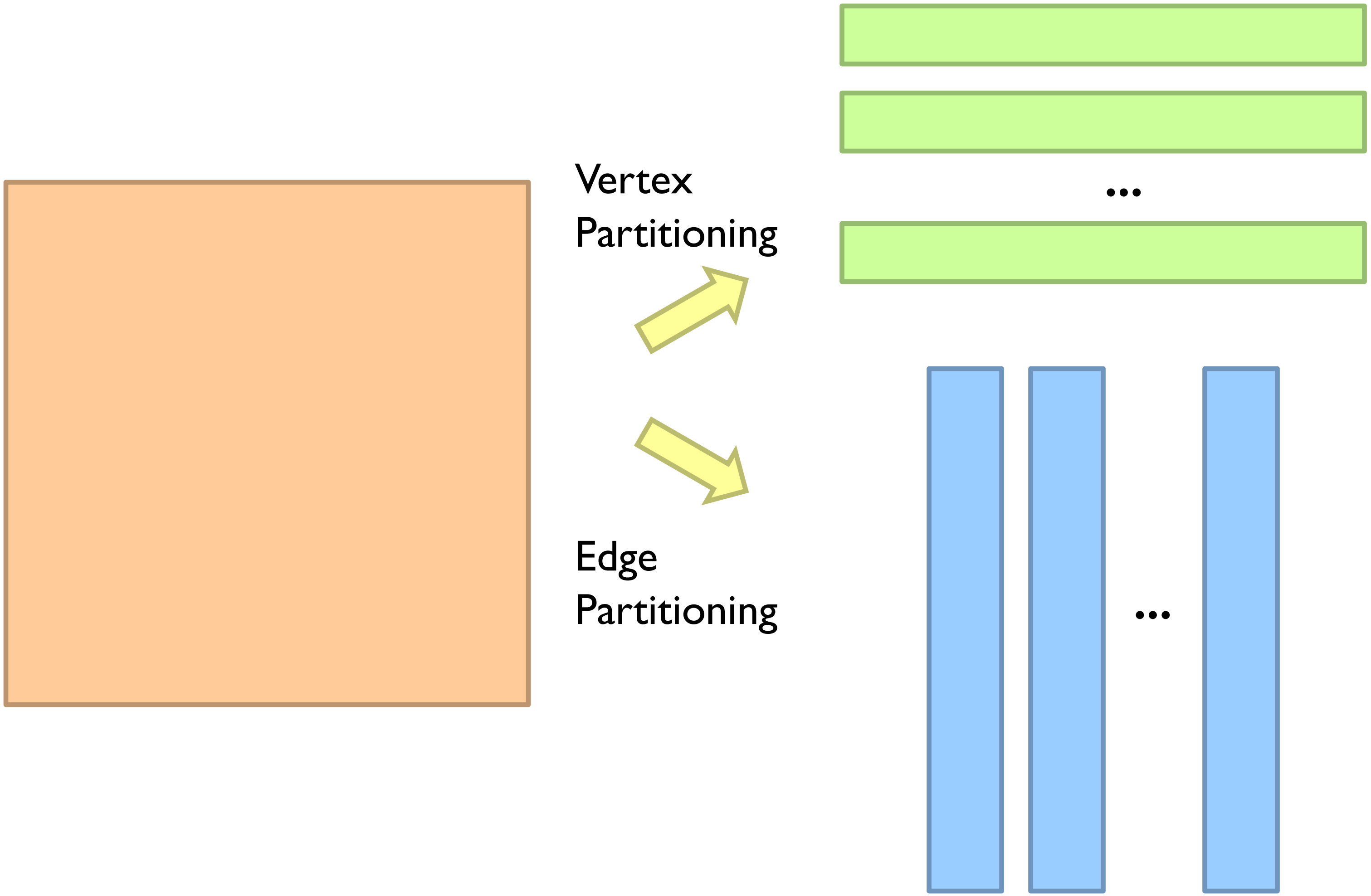
Advantages

Easily support edge insertions

Disadvantages

Wastes spaces

Graph Partitioning



(A lot more detail later…)

Storing Undirected Graphs

Standard Tricks

1. Store both edges

Make sure your algorithm de-dups

2. Store one edge, e.g., (x, y) st. $x < y$

Make sure your algorithm handles the asymmetry

Basic Graph Manipulations

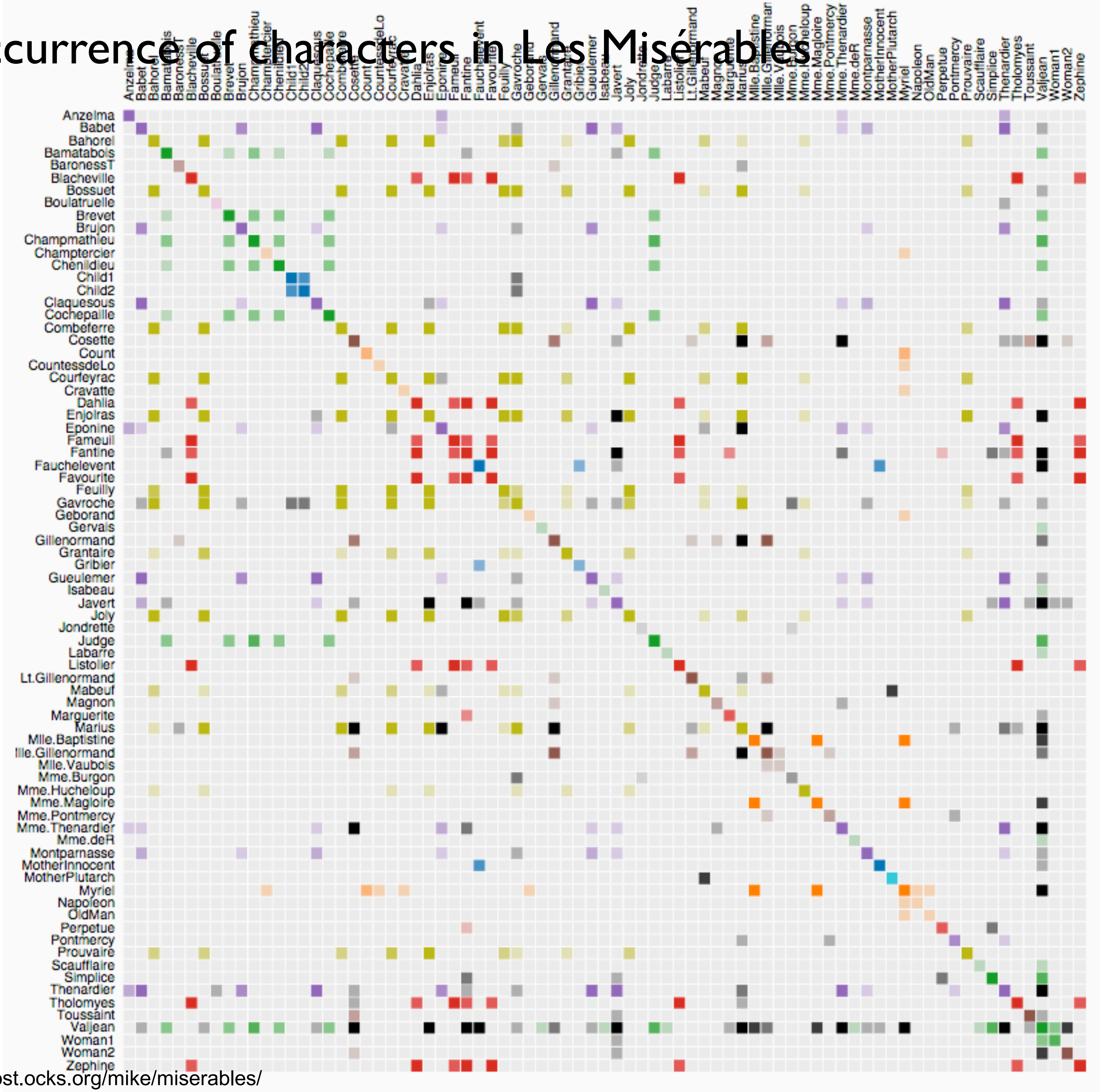
Invert the graph
flatMap and regroup

Adjacency lists to edge lists
flatMap adjacency lists to emit tuples

Edge lists to adjacency lists
groupBy

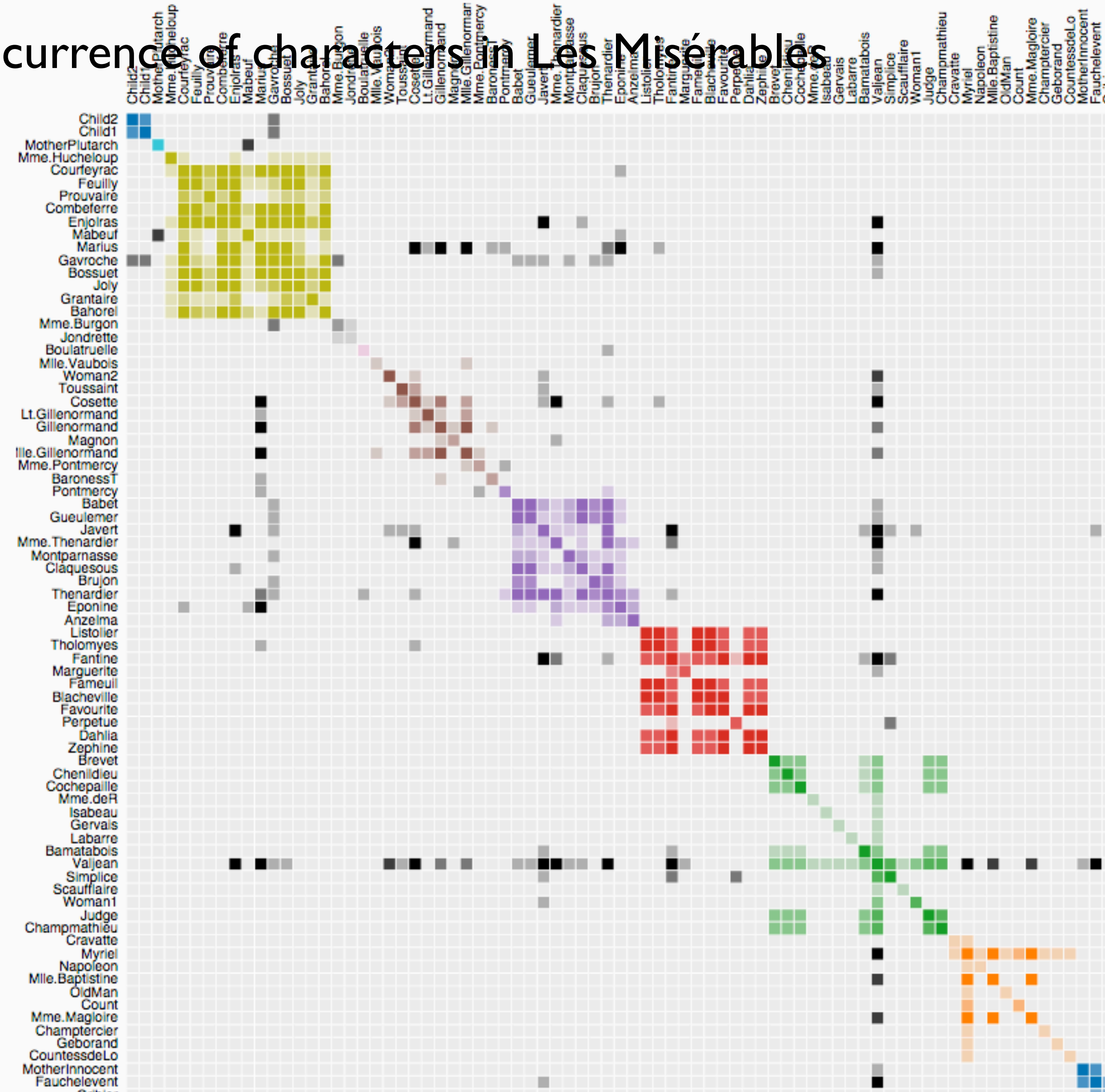
Framework does all the heavy lifting!

Co-occurrence of characters in Les Misérables



Source: http://bost.ocks.org/mike/miserables/

Co-occurrence of characters in Les Misérables



Source: http://bost.ocks.org/mike/miserables/

Co-occurrence of characters in Les Misérables

How are visualizations like this generated?
Limitations?

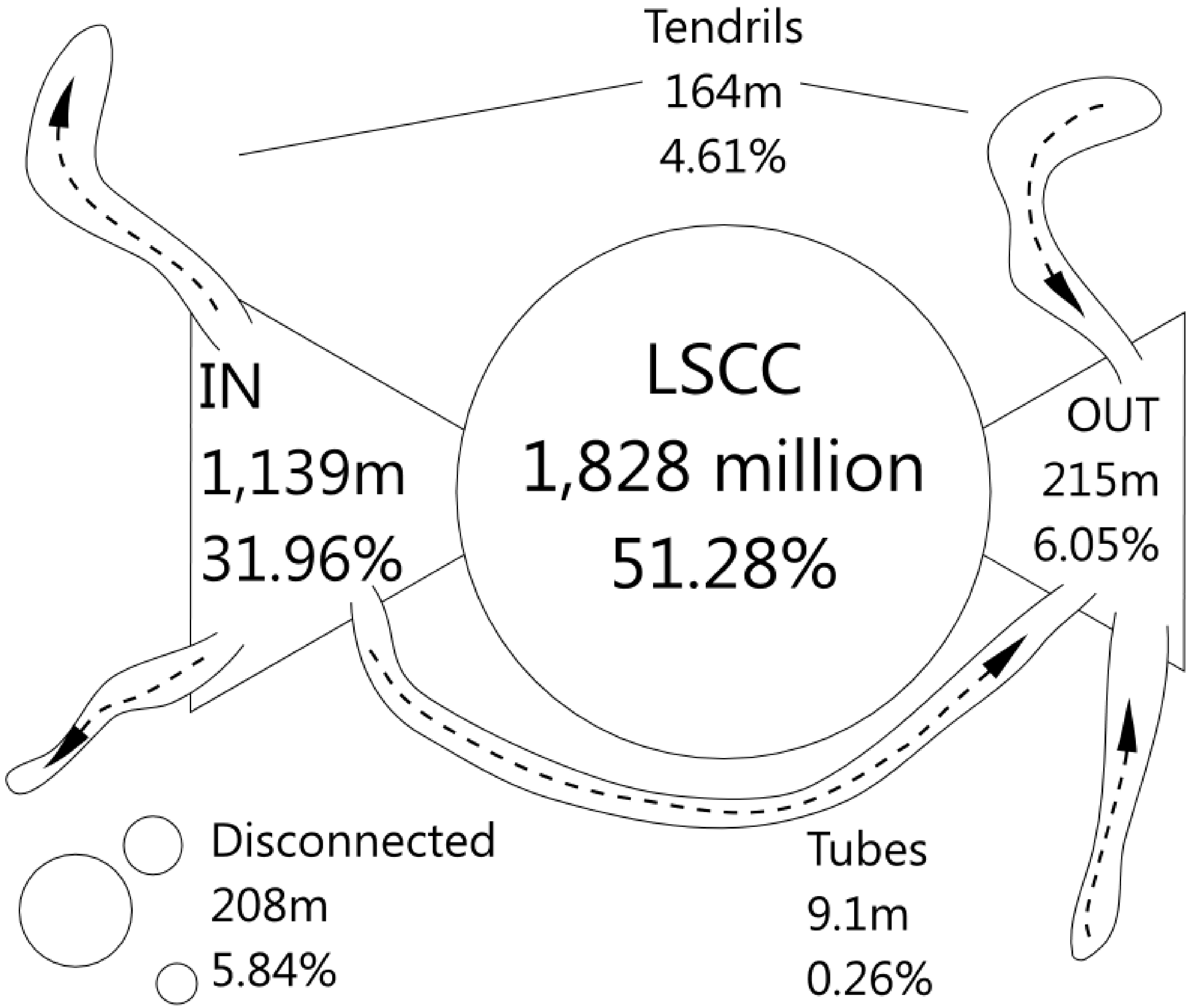
Source: http://bost.ocks.org/mike/miserables/

What does the web look like?

Analysis of a large webgraph from the common crawl: 3.5 billion pages, 129 billion links

Meusel et al. Graph Structure in the Web — Revisited. WWW 2014.

Broder's Bowtie (2000) – revisited



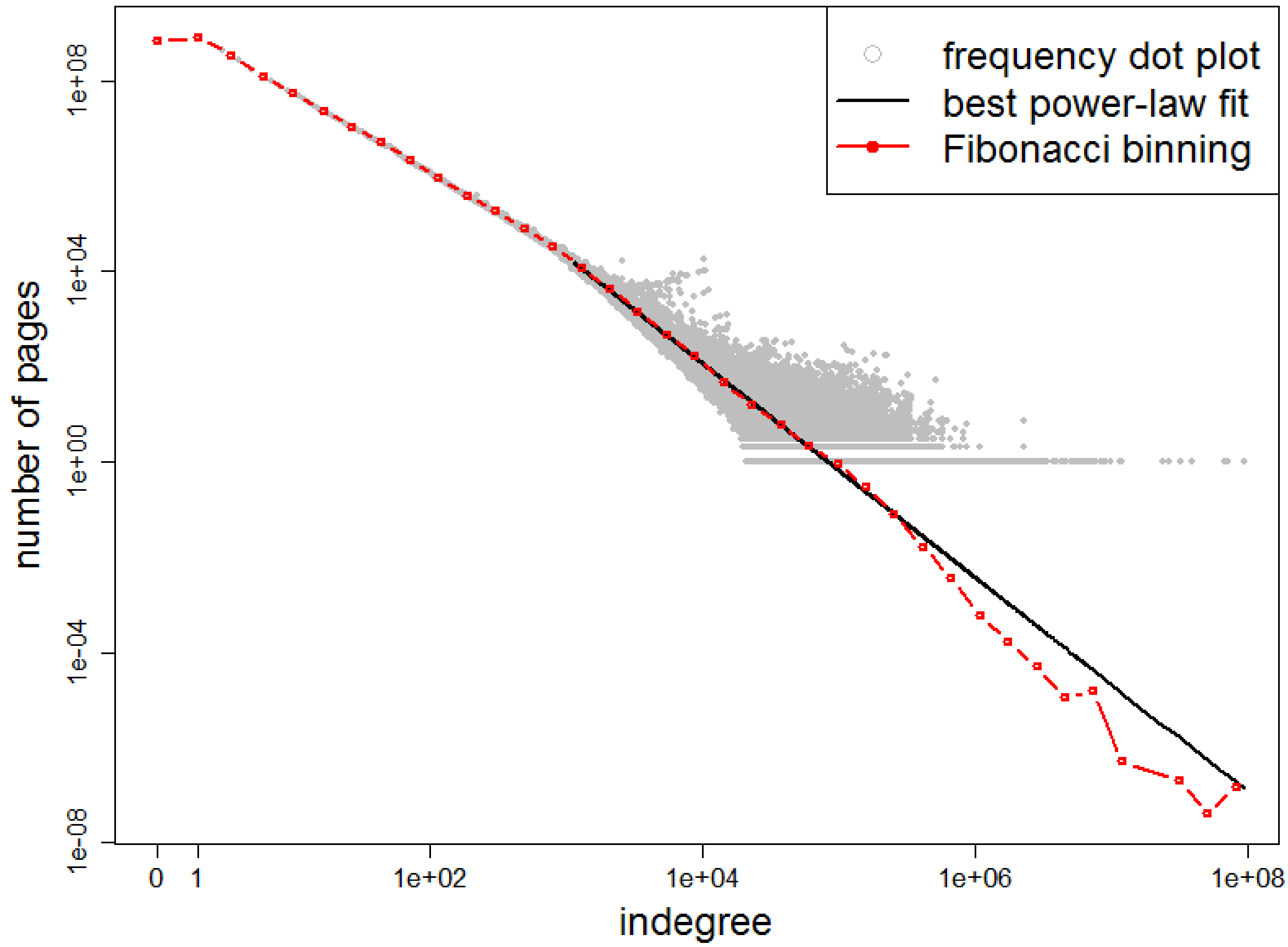
What does the web look like?

Very roughly, a scale-free network

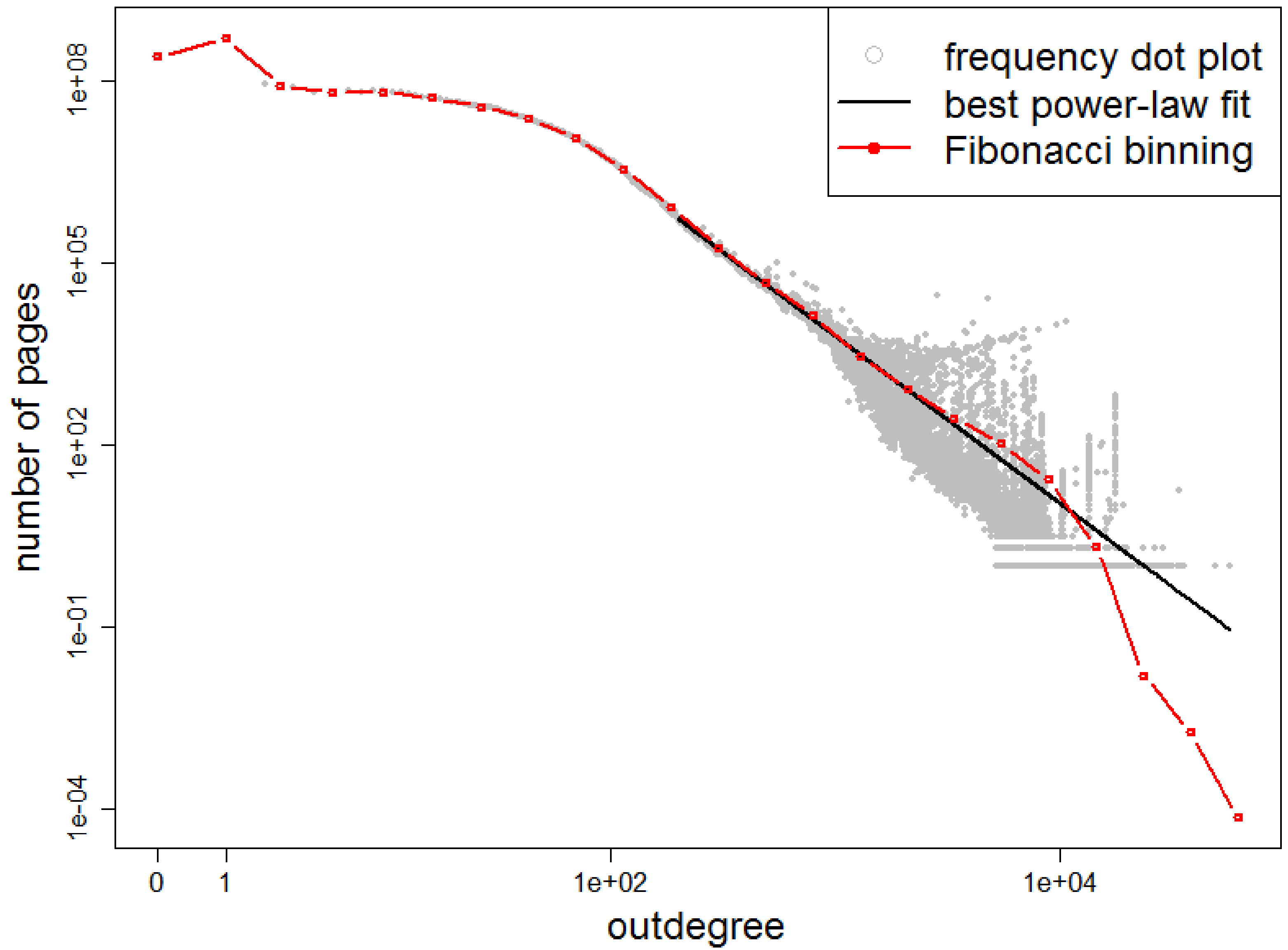
Fraction of k nodes having k connections:

$$P(k) \sim k^{-\gamma}$$

(i.e., distribution follows a power law)



frequency dot plot
best power-law fit
Fibonacci binning
number of pages
1e+08
1e+04
1e+00
1e-04
1e-08
0
1
1e+02
1e+04
1e+06
1e+08
indegree



frequency dot plot
best power-law fit
Fibonacci binning
number of pages
1e+08
1e+05
1e+02
1e-01
1e-04
0
1
1e+02
1e+04
outdegree

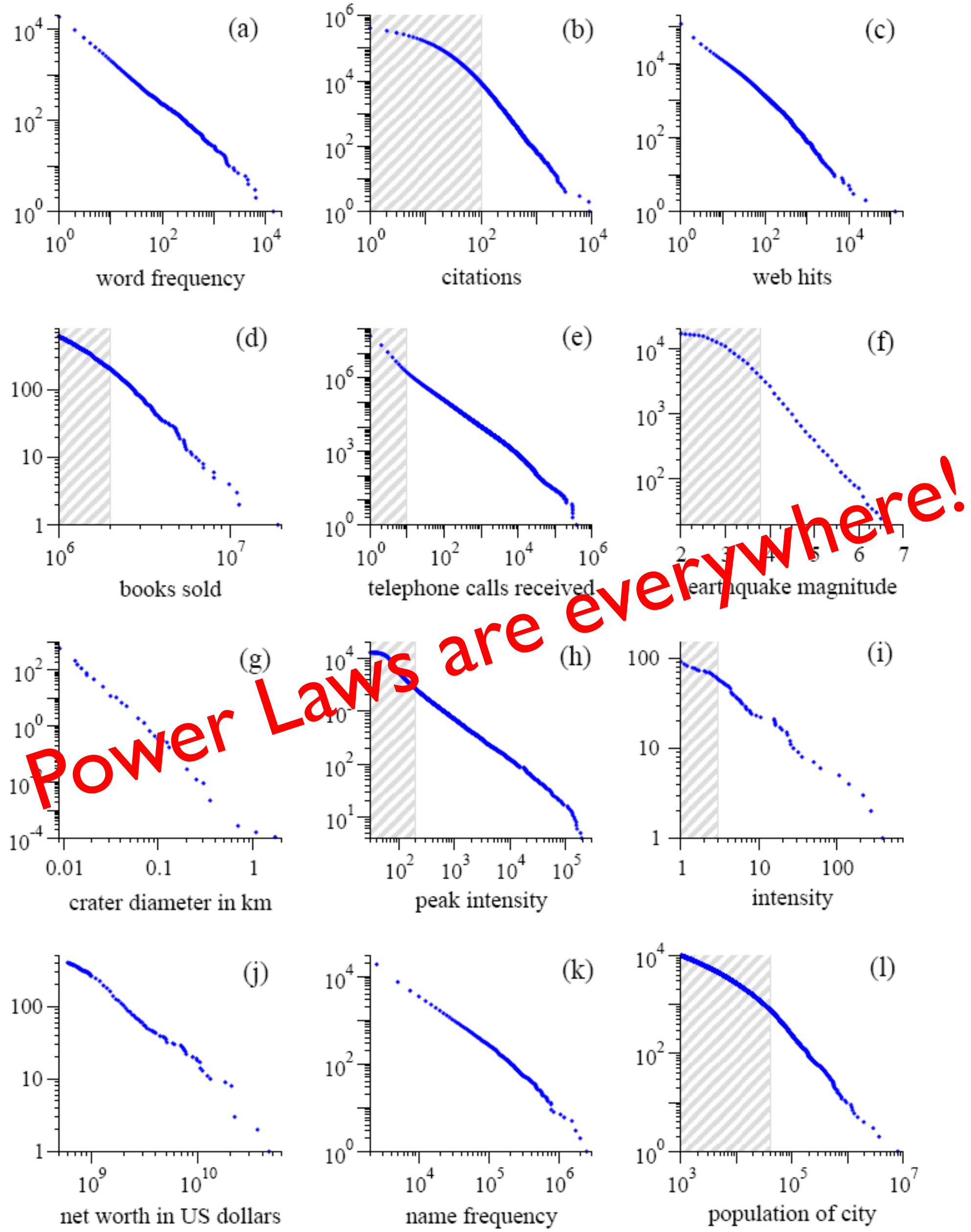
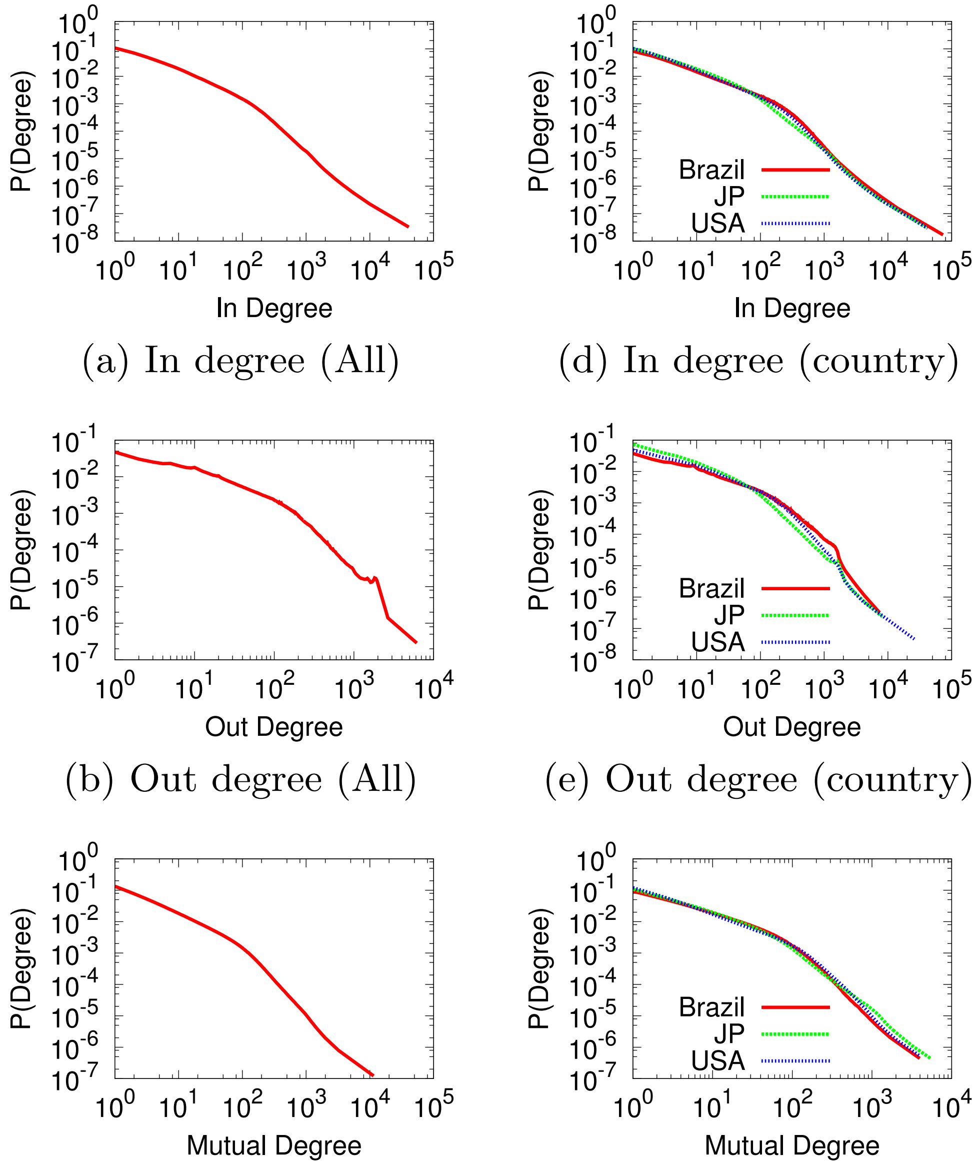


Figure from: Newman, M. E. J. (2005) "Power laws, Pareto distributions and Zipf's law." Contemporary Physics 46:323–351.



(a) In degree (All) (d) In degree (country)

(b) Out degree (All) (e) Out degree (country)

(c) Mutual degree (All) (f) Mutual degree (country)

What about Facebook?

Figure from: Seth A. Myers, Aneesh Sharma, Pankaj Gupta, and Jimmy Lin. Information Network or Social Network? The Structure of the Twitter Follow Graph. WWW 2014.

What does the web look like?

Very roughly, a scale-free network

Other Examples:

Internet domain routers

Co-author network

Citation network

Movie-Actor network

Why?

(In this installment of “learn fancy terms for simple ideas”)

Preferential Attachment

Also:

Matthew Effect

For unto every one that hath shall be given, and he shall have abundance: but from him that hath not shall be taken even that which he hath.

— Matthew 25:29, King James Version.

BTW, how do we compute these graphs?

Source: http://www.flickr.com/photos/guvnah/7861418602/

BTW, how do we extract the webgraph?
The webgraph… is big?!

A few tricks:

Integerize vertices (montone minimal perfect hashing)
Sort URLs
Integer compression

webgraph from the common crawl: 3.5 billion pages, 129 billion links
Meusel et al. Graph Structure in the Web — Revisited. WWW 2014.
58 GB!

Graphs and MapReduce (and Spark)

A large class of graph algorithms involve:

Local computations at each node

Propagating results: "traversing" the graph

Key questions:

How do you represent graph data in MapReduce (and Spark)?

How do you traverse a graph in MapReduce (and Spark)?

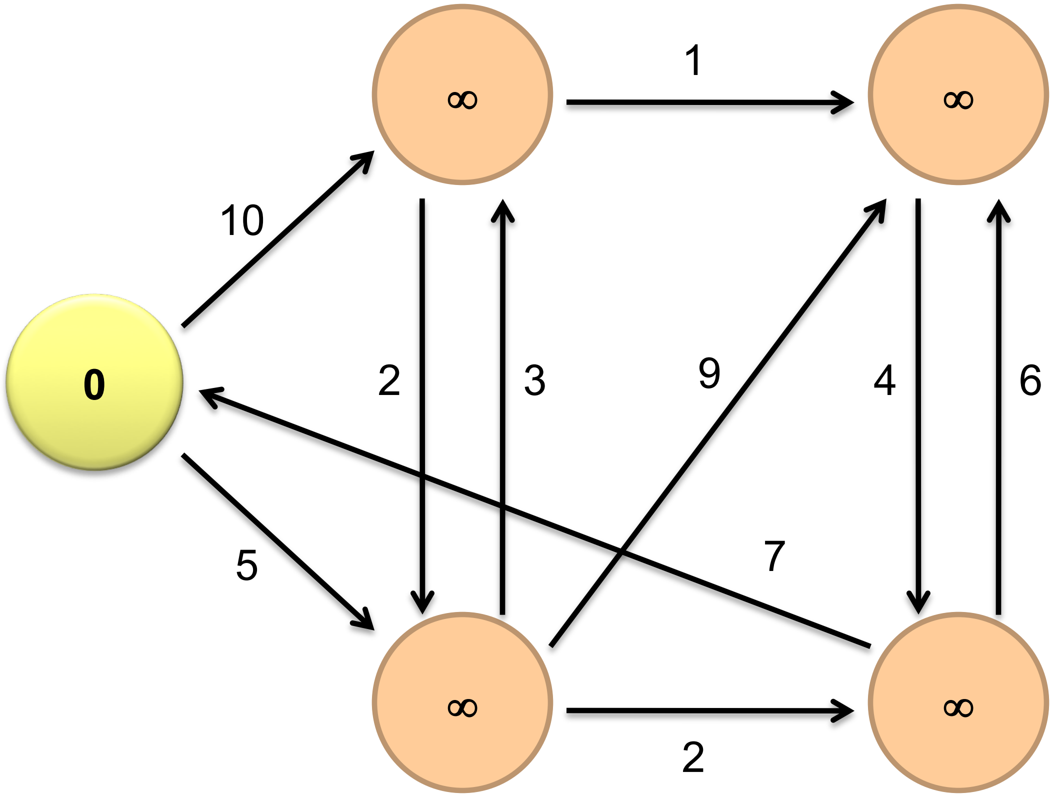
Single-Source Shortest Path

Problem: find shortest path from a
source node to one or more target nodes

Shortest might also mean lowest weight or cost

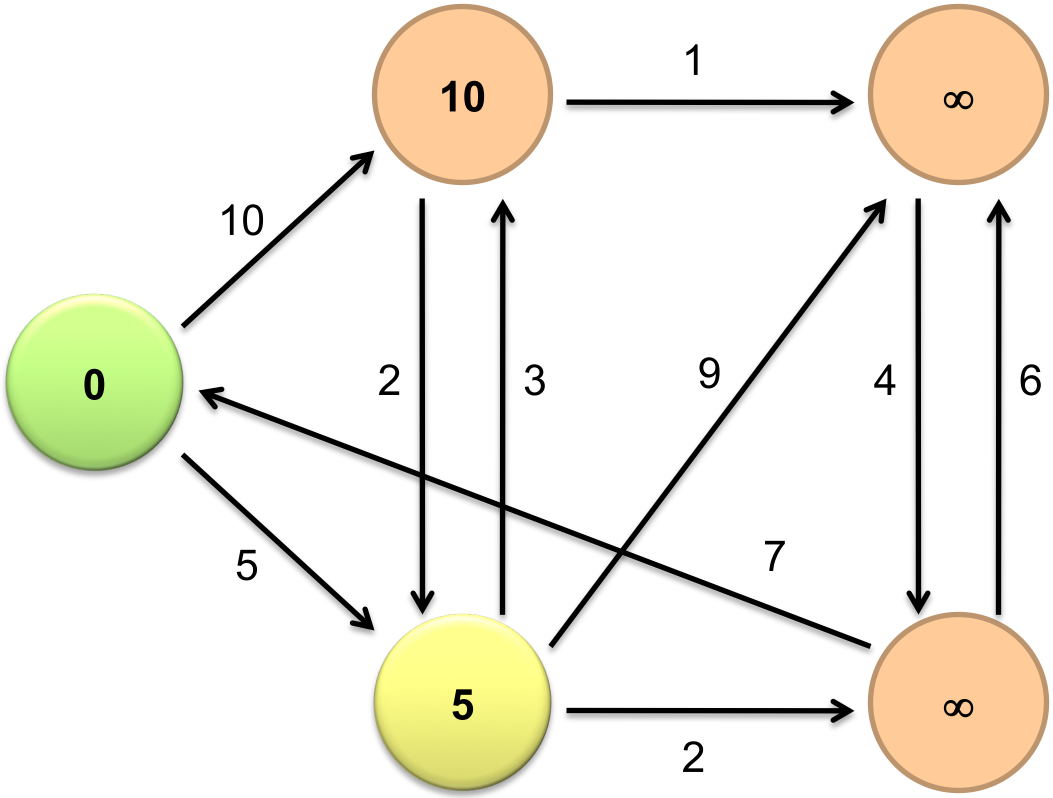
First, a refresher: Dijkstra's Algorithm…

Dijkstra's Algorithm Example



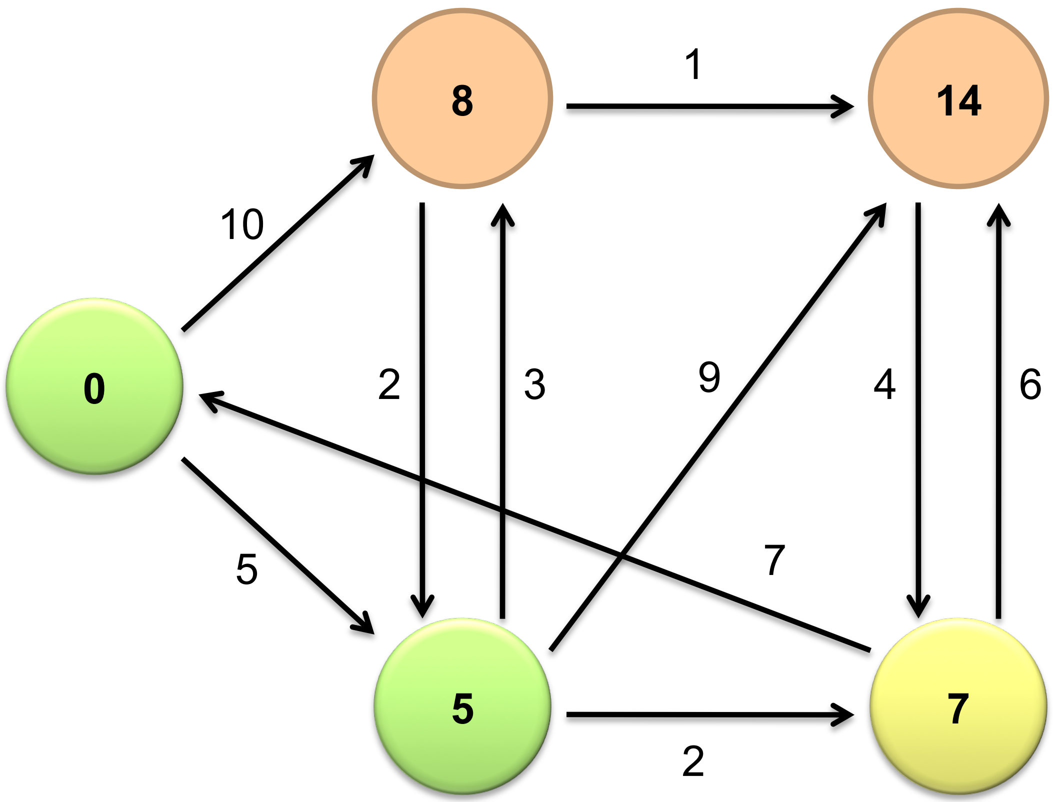
Example from CLR

Dijkstra's Algorithm Example



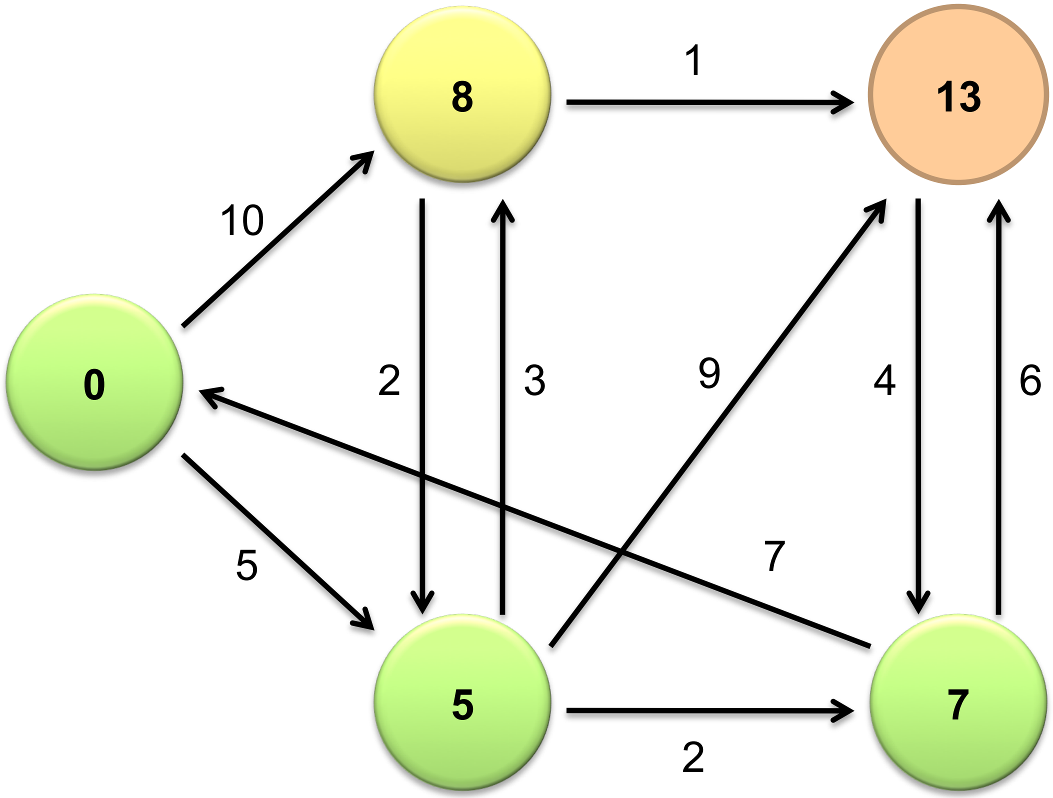
Example from CLR

Dijkstra’s Algorithm Example



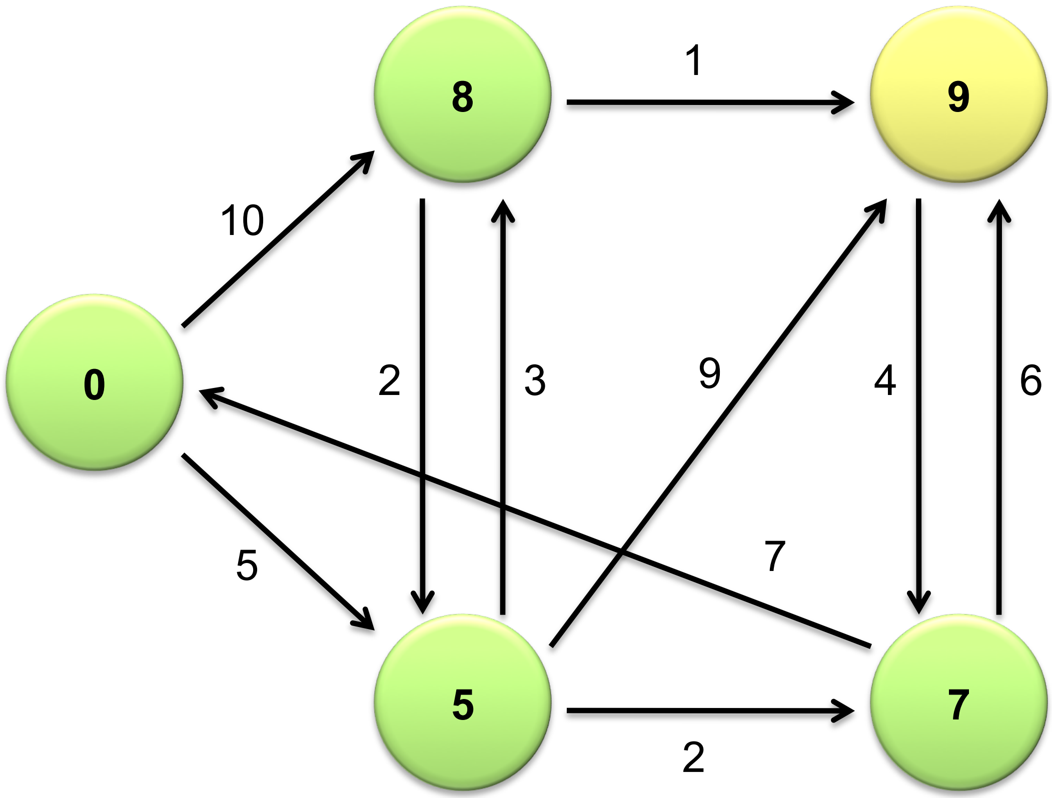
Example from CLR

Dijkstra's Algorithm Example



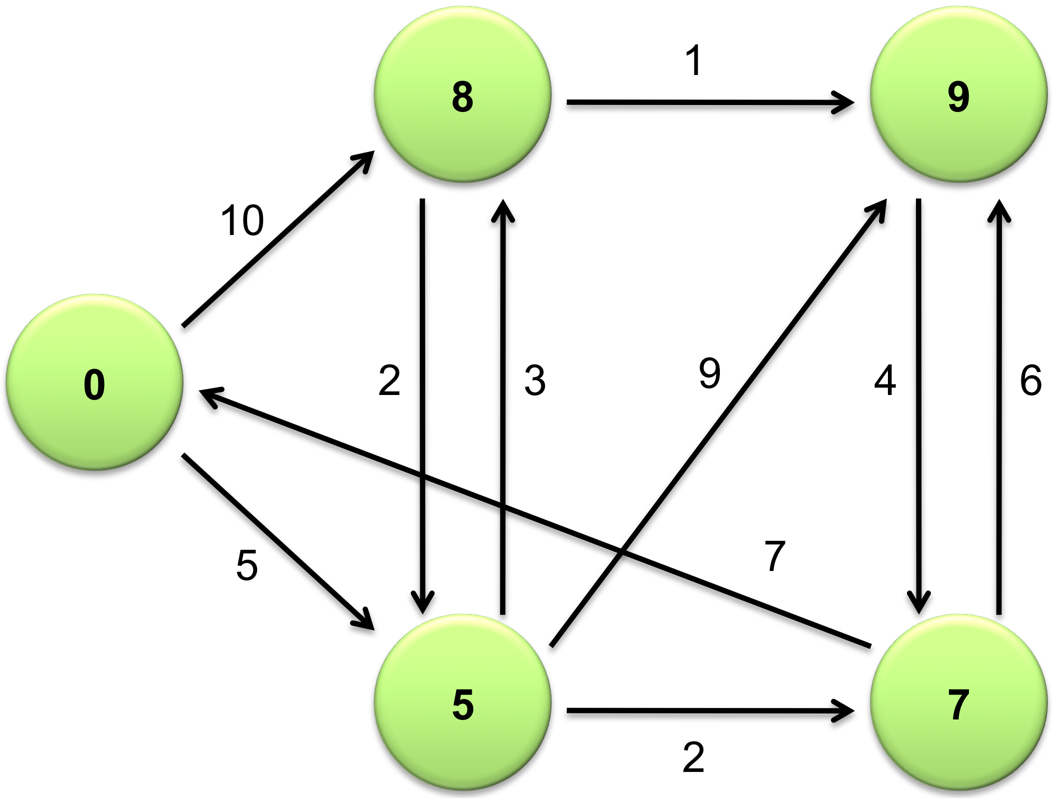
Example from CLR

Dijkstra’s Algorithm Example



Example from CLR

Dijkstra’s Algorithm Example



Example from CLR

Single-Source Shortest Path

Problem: find shortest path from a
source node to one or more target nodes
Shortest might also mean lowest weight or cost

Single processor machine: Dijkstra's Algorithm
MapReduce: parallel breadth-first search (BFS)

Finding the Shortest Path

Consider simple case of equal edge weights

Solution to the problem can be defined inductively:

Define: *b* is reachable from *a* if *b* is on adjacency list of *a*

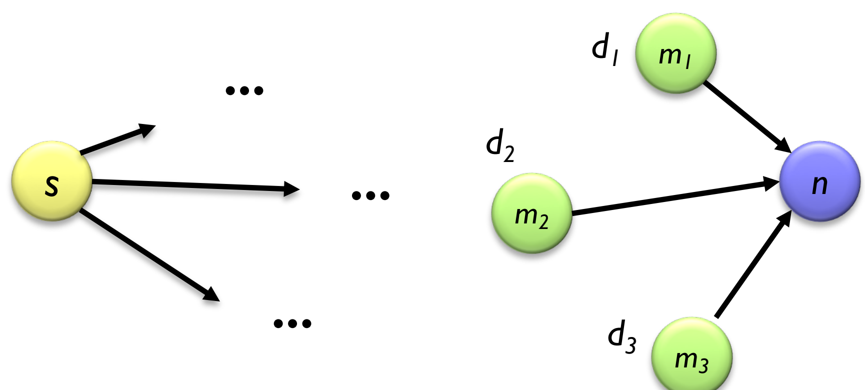
DISTANCETO(s) = 0

For all nodes p reachable from s,

DISTANCETO(p) = 1

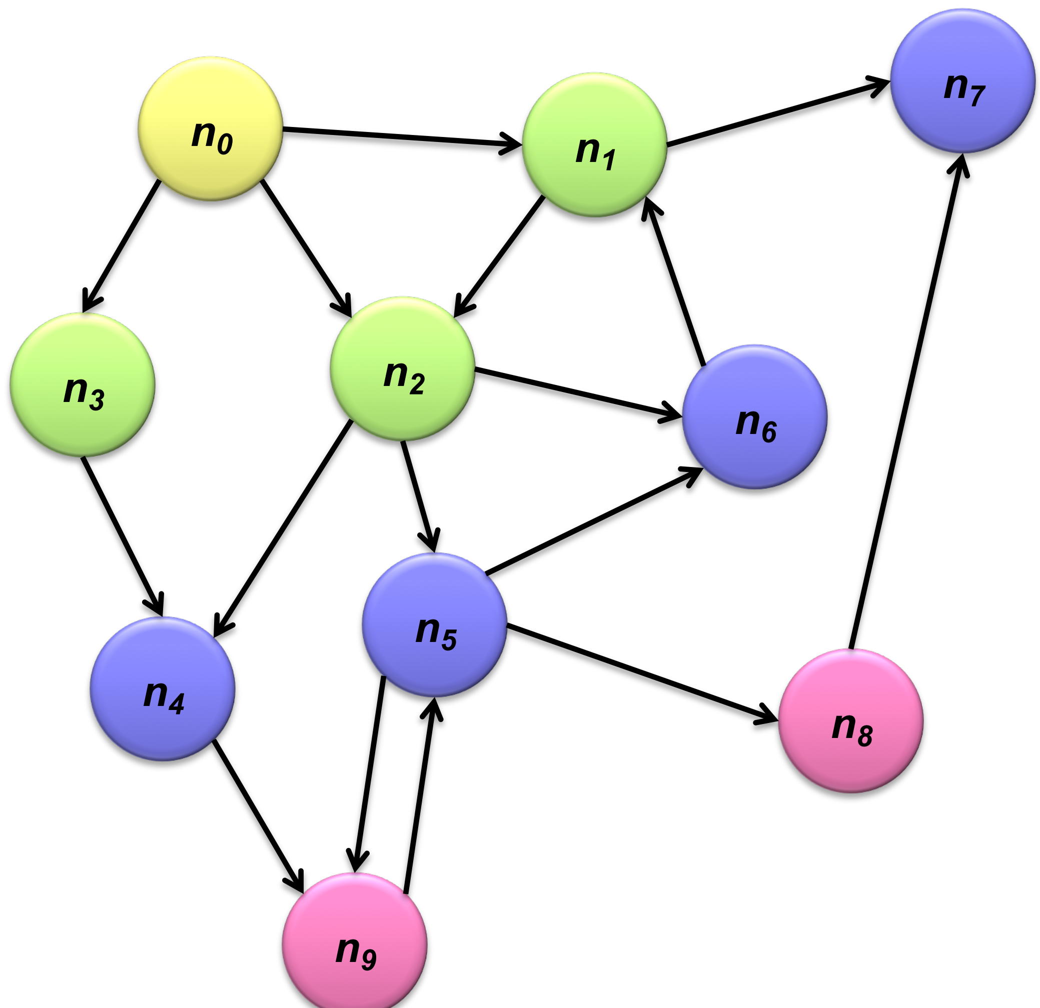
For all nodes n reachable from some other set of nodes M,

DISTANCETO(n) = 1 + min(DISTANCETO(m), $m \in M$)




Source: Wikipedia (Wave)

Visualizing Parallel BFS



From Intuition to Algorithm

Data representation:

Key: node n

Value: d (distance from start), adjacency list

Initialization: for all nodes except for start node, $d = \infty$

Mapper:

$\forall m \in$ adjacency list: emit (m, d + 1)

Remember to also emit distance to yourself

Sort/Shuffle:

Groups distances by reachable nodes

Reducer:

Selects minimum distance path for each reachable node

Additional bookkeeping needed to keep track of actual path

Multiple Iterations Needed

Each MapReduce iteration advances the "frontier" by one hop

Subsequent iterations include more reachable nodes as frontier expands

Multiple iterations are needed to explore entire graph

Preserving graph structure:

Problem: Where did the adjacency list go?

Solution: mapper emits (n, adjacency list) as well

Ugh! This is ugly!

BFS Pseudo-Code

```
class Mapper {
  def map(id: Long, n: Node) = {
    emit(id, n)
    val d = n.distance
    emit(id, d)
    for (m <- n.adjacenyList) {
      emit(m, d+1)
    }
}

class Reducer {
  def reduce(id: Long, objects: Iterable[Object]) = {
    var min = infinity
    var n = null
    for (d <- objects) {
      if (isNode(d))     n = d
      else if d < min    min = d
    }
    n.distance = min
    emit(id, n)
  }
}
```

Stopping Criterion
(equal edge weight)

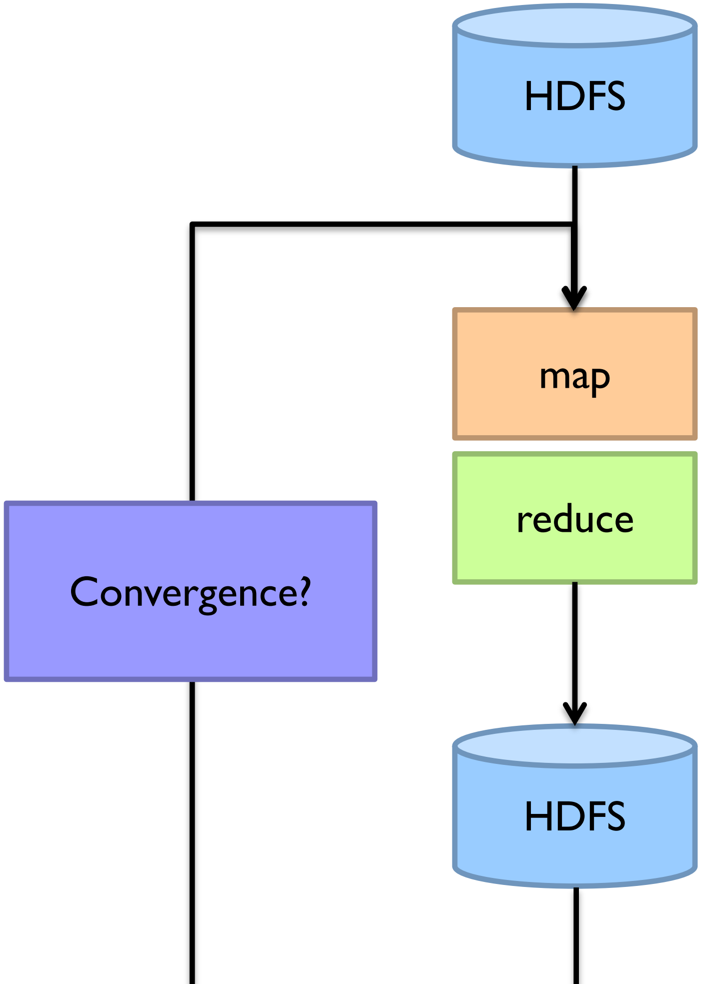
How many iterations are needed in parallel BFS?

Convince yourself: when a node is first "discovered", we've found the shortest path

What does it have to do with six degrees of separation?

Practicalities of MapReduce implementation…

Implementation Practicalities



Comparison to Dijkstra

Dijkstra's algorithm is more efficient

At each step, only pursues edges from minimum-cost path inside frontier

MapReduce explores all paths in parallel

Lots of "waste"

Useful work is only done at the "frontier"

Why can't we do better using MapReduce?

Single Source: Weighted Edges

Now add positive weights to the edges

Simple change: add weight w for each edge in adjacency list

Simple change: add weight *w* for each edge in adjacency list

In mapper, emit (m, $d + w_p$) instead of (m, $d + 1$) for each node m

That's it?

Stopping Criterion

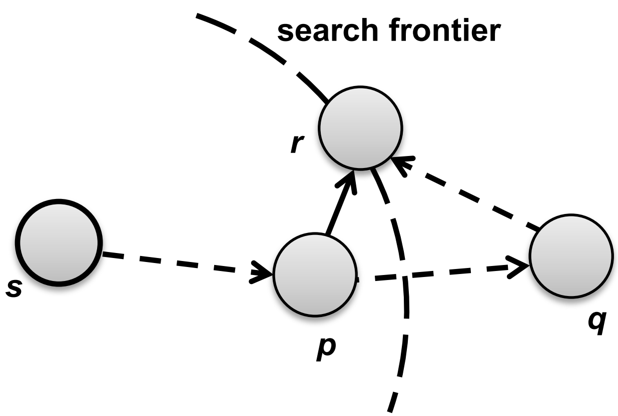
(positive edge weight)

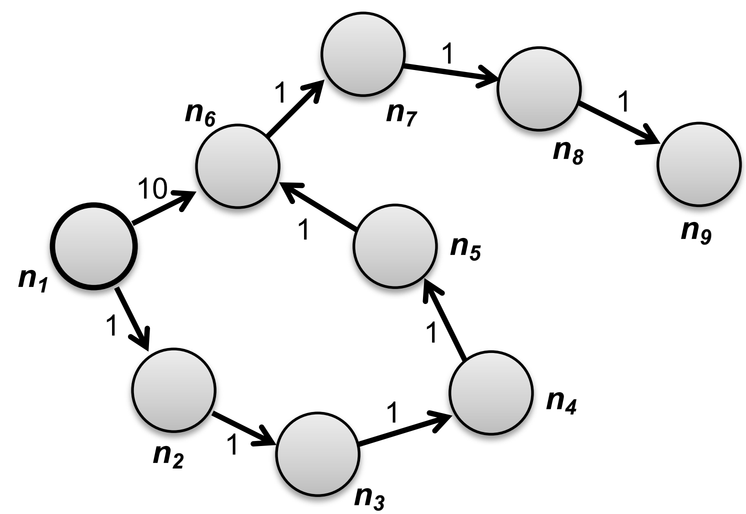
How many iterations are needed in parallel BFS?

Convince yourself: when a node is first "discovered", we've found the shortest path

Not true!

Additional Complexities





Stopping Criterion

(positive edge weight)

How many iterations are needed in parallel BFS?

Practicalities of MapReduce implementation…

Source: Wikipedia (Japanese rock garden)