Managing Big Data Graphs

Evaggelia Pitoura
Computer Science and Engineering Department
University of Ioannina, Greece
http://www.cs.uoi.gr/~pitoura

Short version of a presentation given at SummerSOC, Crete, July 3, 2014

Talk Outline

Why?

What?

How?

Very large body of current research, just an *overview*

Why graphs?

Social networks

>$10B revenue

>0.5B users

Social network graphs

- A social network is usually modeled as a graph
  - A node → a user/ an actor
  - An edge → a relationship
    or interaction

Information/Media networks

- Nodes: Twitter users
- Edges: Follows/conversations

Communication networks

- Nodes: People
- Edges: email exchange, phone calls

The Internet

- Nodes: Internet nodes
- Edges: communication between nodes

Financial Networks

- Nodes: Companies
- Edges: relationships (financial, collaboration)

Biological networks

- Nodes: Proteins
- Edges: interactions

- Nodes: metabolites, enzymes
- Edges: chemical reactions

Information networks

- Nodes: Web Pages
- Edges: Links

More graphs

Linked open data

Food Web (what-eats-what

IR: bipartite graphs of documents and terms

Diversity of models

- Scale free graphs
  - Power law degree distribution
- Community structure
- Small world

Why graphs?

- Ranking nodes and information
- Locate information

- Identify influential people
- Find communities
- Model complex dependencies
- Meaningful recommendations
- Link prediction

Why graphs?

Understanding information cascades and virus contagion

- “viral’ marketing
- web-log (‘blog’) news propagation
- Decease propagation

Event detection

Computer network security: email/IP traffic and anomaly detection

Talk Outline

Why?

What?

How?

What kind of analysis?

Core operations?

Offline graph analytics

- PageRank, betweenness
- Triangle counting
- Clustering, community detection
- Bipartite matching

Online query processing

- Reachability, distance query
- Subgraph matching query
- SPARQL query
- ...

Core operations?

Plus

Modeling and generation, visualization, interactive exploration …

- ✓ Macroscopic or global: traversal of the whole graph
- ✓ Microscopic or node centric (egonet): neighbors of specific nodes

- ✓ Structure
- ✓ Attribute or Label constraints

Core operations?

Let us see two examples

1. offline + global and
2. online + global

Both pure structural

An (offline) example: Betweeness and clustering

- Centrality Analysis:
  - Find out the most important (central) nodes in a graph
- Commonly-used centrality Measures
  - Degree Centrality
  - Closeness Centrality
  - Betweenness Centrality
  - PageRank
  - Eigenvector Centrality

Betweenness Centrality

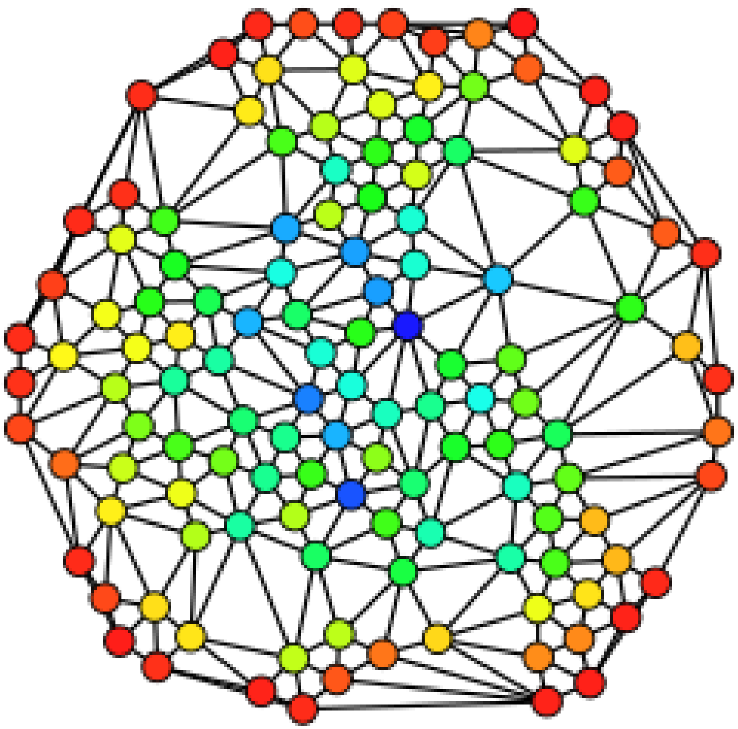
- For a node: Counts *the number of shortest paths* that pass through one node

$$C_B(v_i) = \sum_{v_s \neq v_i \neq v_t \in V, s<t} \frac{\sigma_{st}(v_i)}{\sigma_{st}}$$

σ_{st} : The number of shortest paths between *s* and *t*

$\sigma_{st}(v_i)$: The number of shortest paths between *s* and *t* that pass v_i

Betweenness Centrality Example

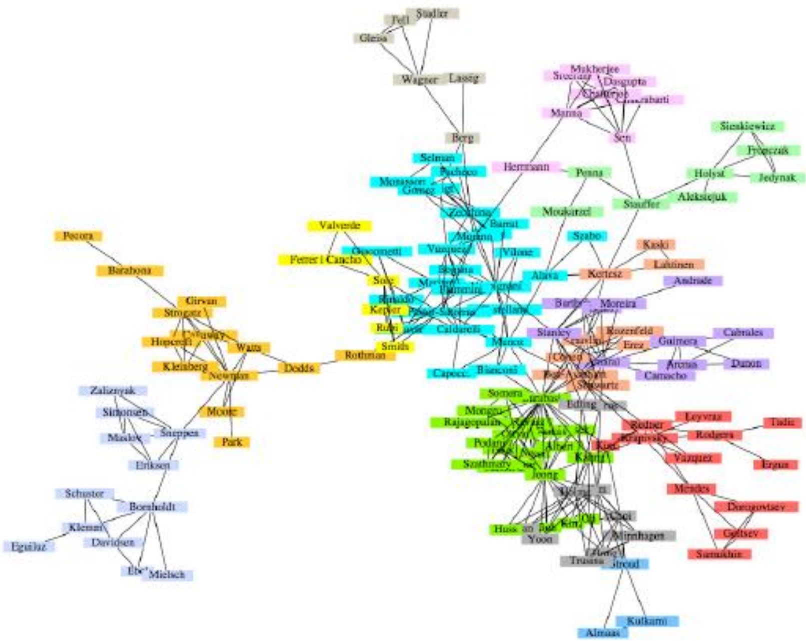


Blue (max)
Red (0)

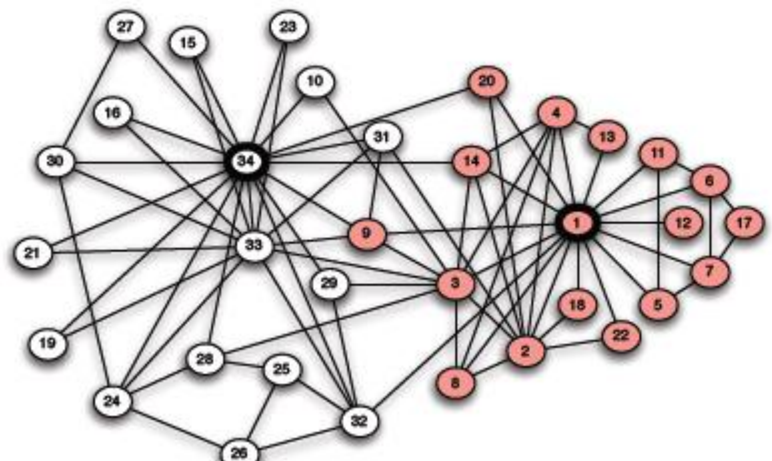
- Nodes with high betweenness are important in communication and information diffusion

Betweenness and Graph Partitioning

Identity densely connected subgraphs



Co-authorship network of physicists and applied mathematicians



Karate club

34 president -- 1 instructor

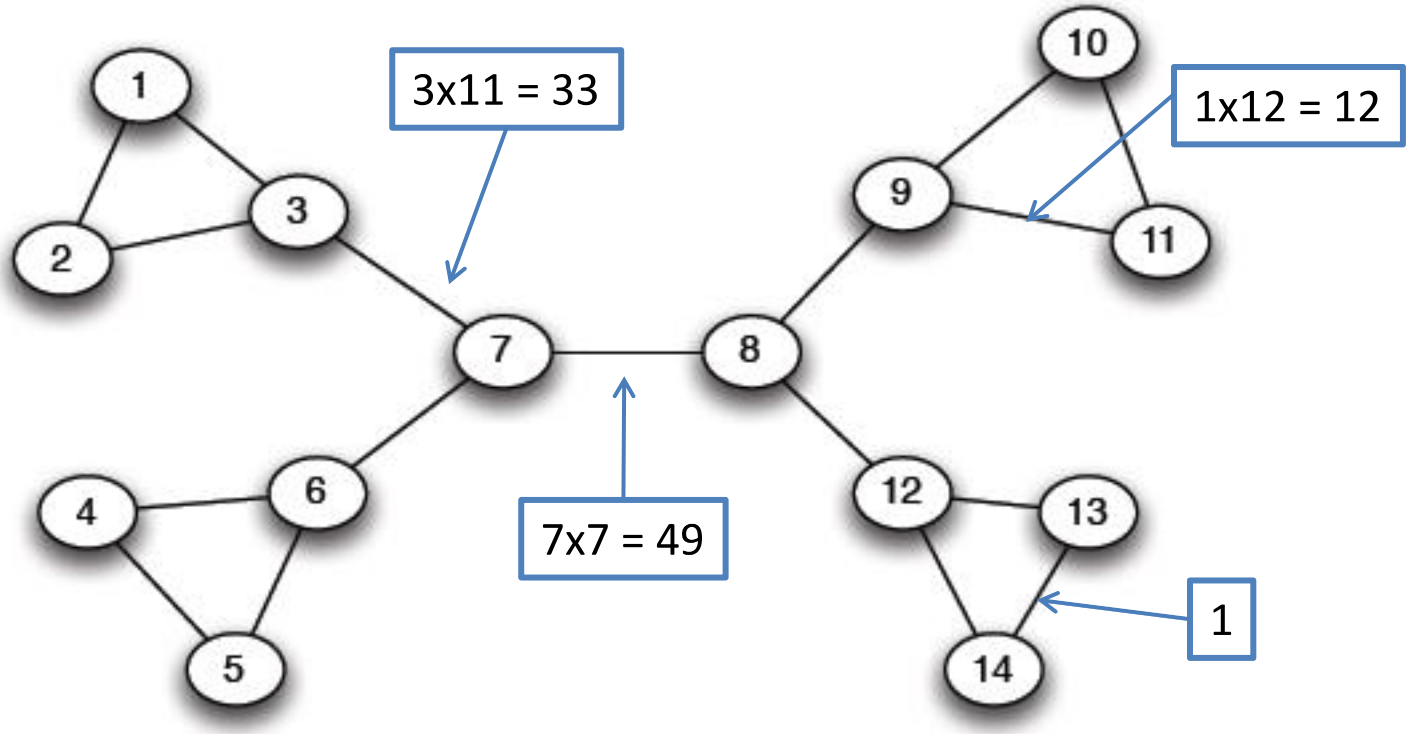
Edge Betweenness

Betweenness of an edge (a, b)

$$\mathrm{bt}(a,b) = \sum_{x,y} \frac{\#shortest\_paths(x,y)through(a,b)}{\#shortest\_paths(x,y)}$$

Edges that have a high probability to occur on a randomly chosen shortest path

An example



Girvan and Newman clustering

1. The betweenness of all edges in the network is calculated

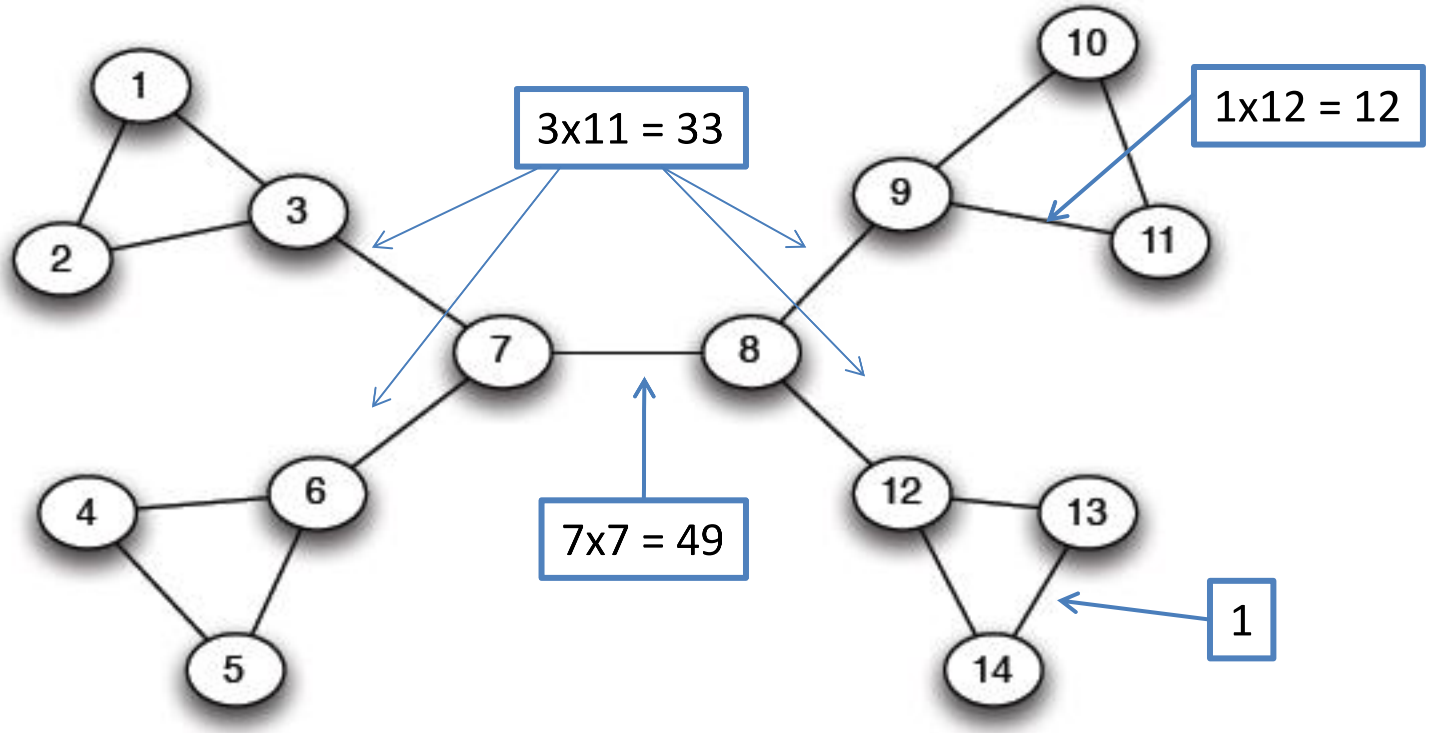
2. The edge with the highest betweenness is removed.

   If this separates the graph -> partition.

3. The betweenness of all edges affected by the removal is recalculated.

Steps 2 and 3 are repeated until no edges remain.

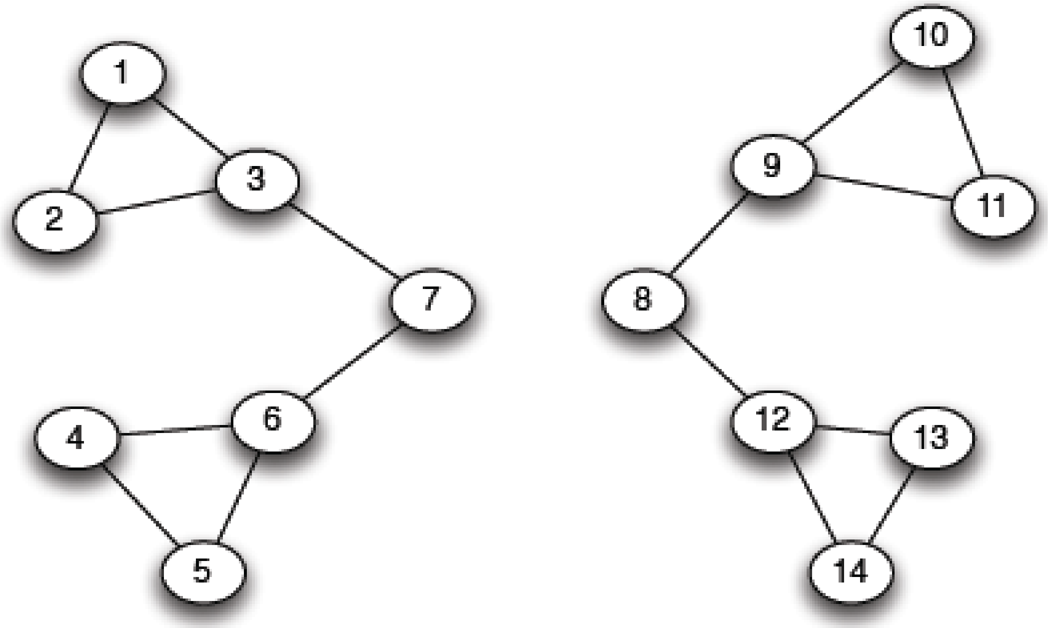
Girvan Newman method: An example



Betweenness(7, 8)= 7x7 = 49 Betweenness(1, 3) = 1X12=12

Betweenness(3, 7)=Betweenness(6-7)=Betweenness(8, 9) = Betweenness(8, 12)= 3X11=33

Girvan Newman method: An example

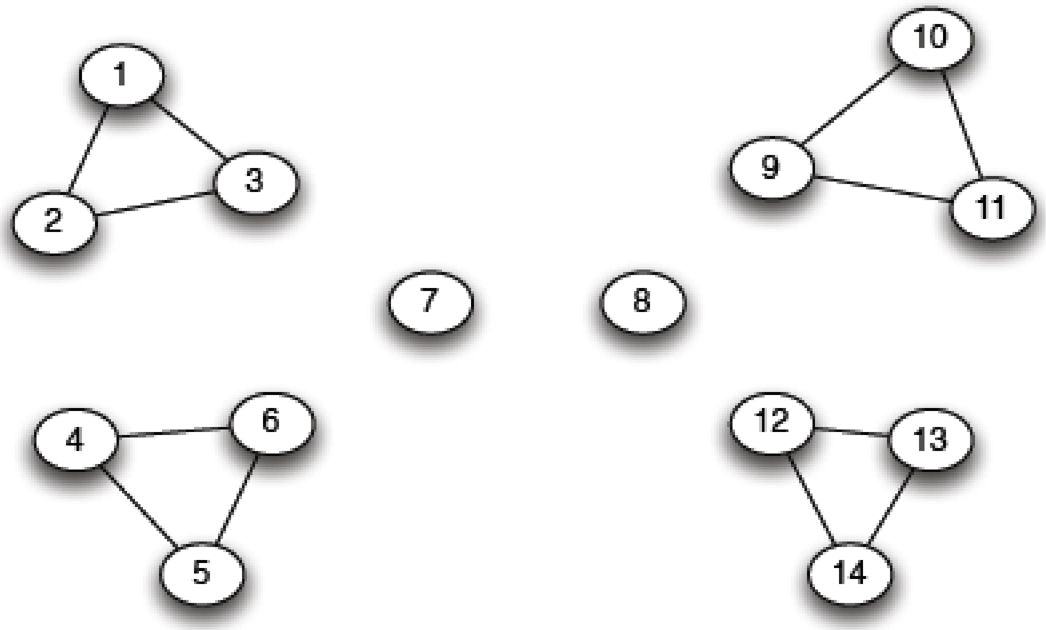


(a) *Step 1*

Betweenness(1, 3) = 1X5=5

Betweenness(3,7)=Betweenness(6,7)=Betweenness(8-9) = Betweenness(8,12)= 3X4=12

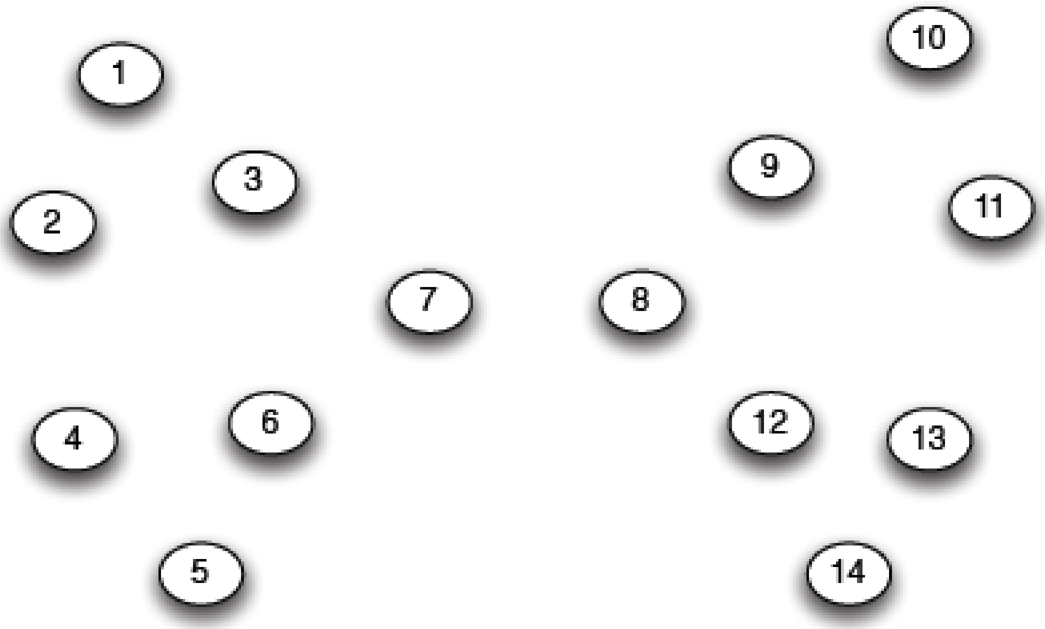
Girvan Newman method: An example



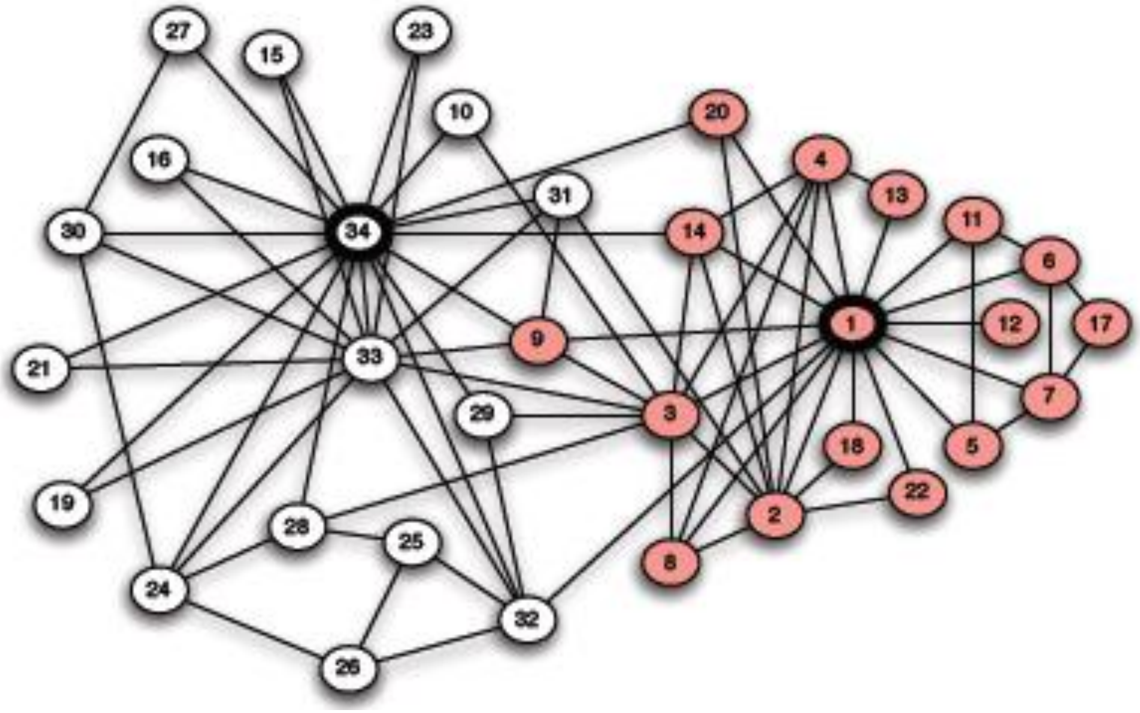
(b) *Step 2*

Betweenness of every edge = 1

Girvan Newman method: An example



Girvan and Newman



34 president -- 1 instructor

- Correct *but node 9* (attached it to 34)
  - why? 3 weeks away from getting a black belt

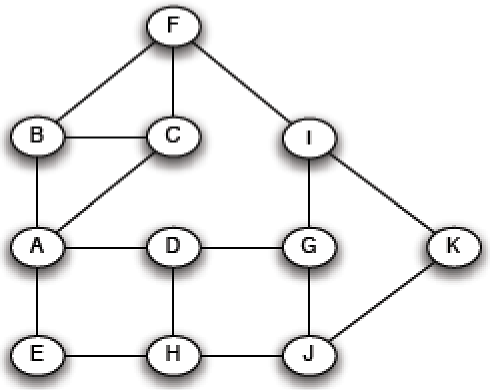
Computing Betweenness

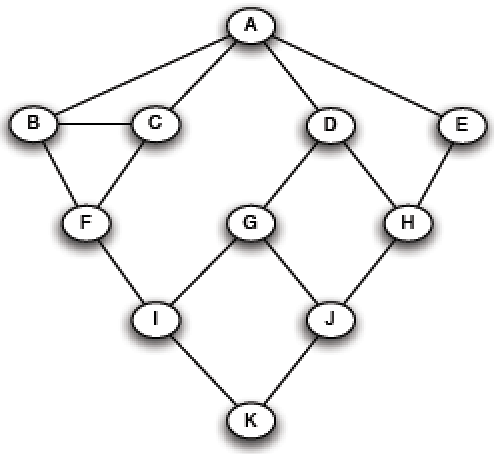
1. Perform a *BFS* starting *from each node A*
2. Determine the shortest path from A to each other node
3. Based on these numbers, determine the amount of flow from A to all other nodes that uses each edge

Repeat the process *for all nodes*

Sum over all BFSs

Step 1





Initial network

BFS on A

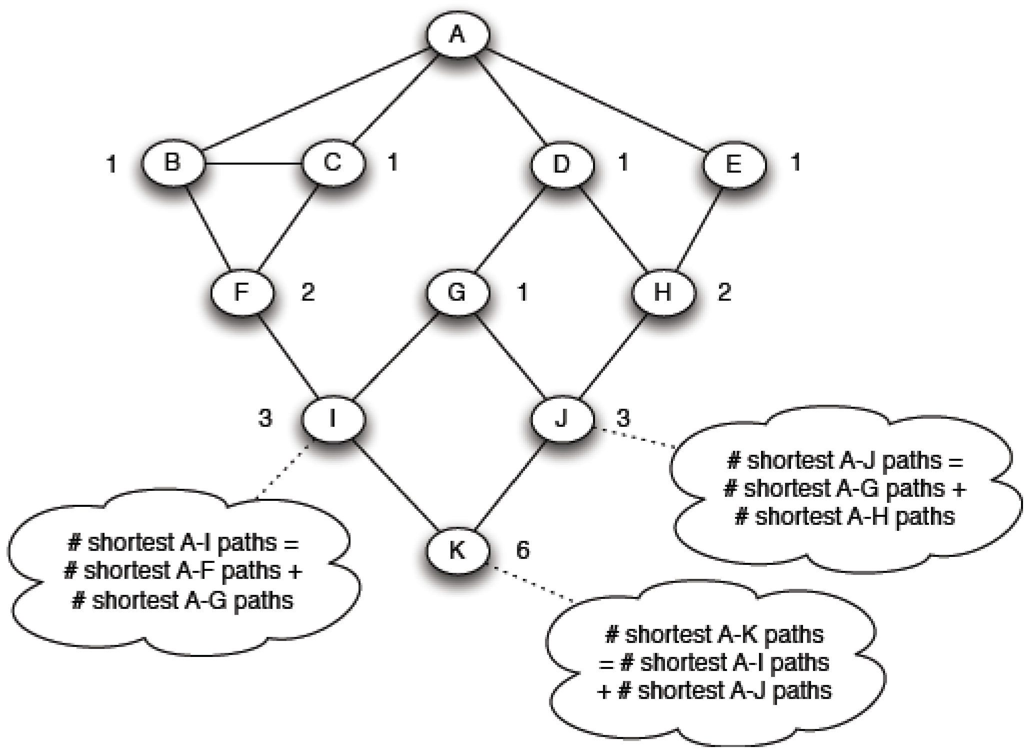
Step 2

Level 1

Level 2

Level 3

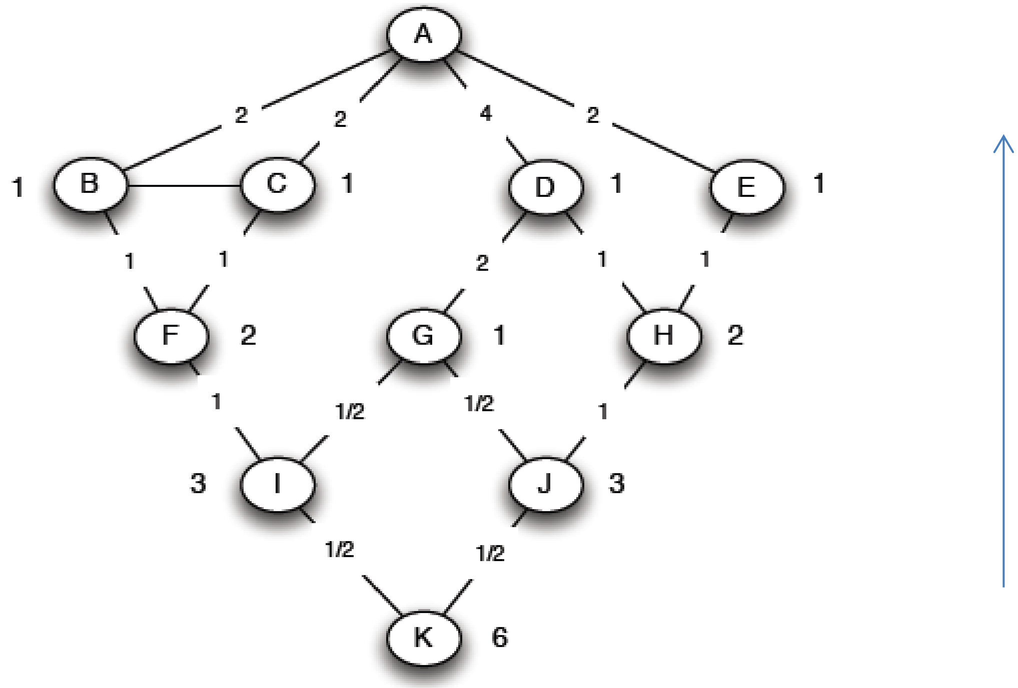
Level 4



Top-down

Count how many shortest paths from A to a specific node

Step 3



The portion of the shortest paths to a node that go through the edge

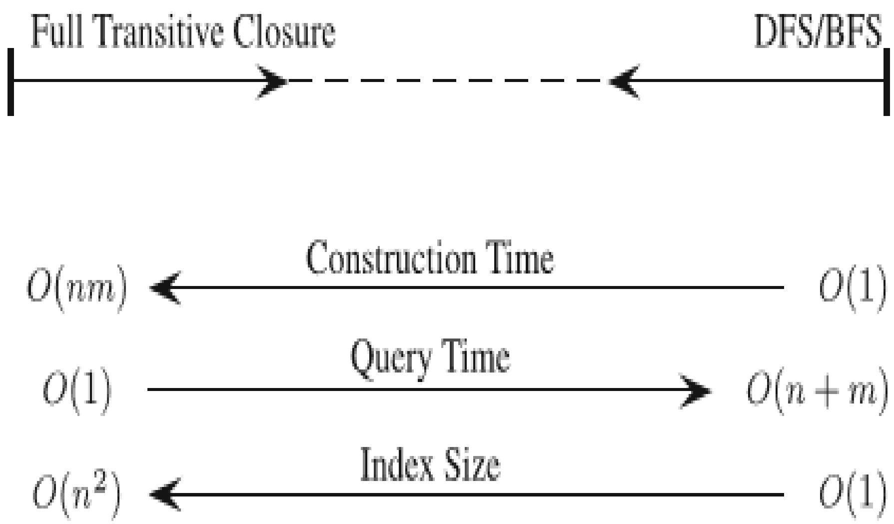
Example H-J: ½ for K and ½ for H -> 1

An (offline) example: Betweeness and clustering

- What is the core operation?
- Is it parallelizable?
- Sampling

An (online) example: 2HOP index

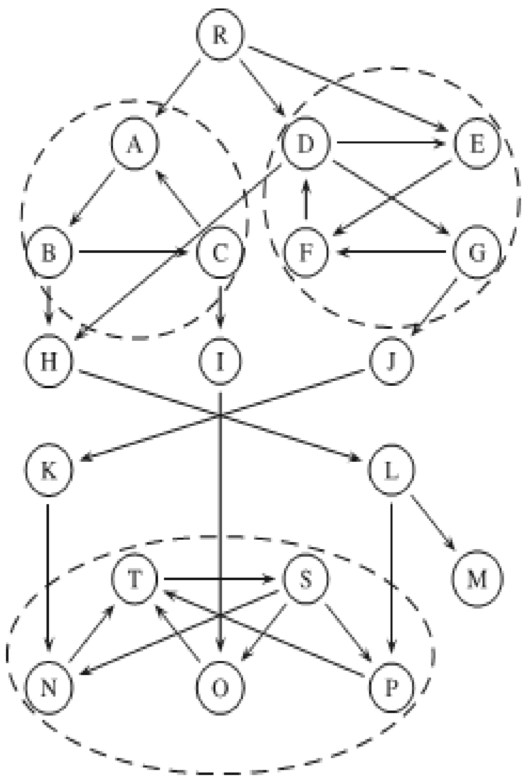
Given a directed graph G(V, E) and two nodes *u* and *v* a *reachability query* asks if there exists a path from *u* to *v* in G

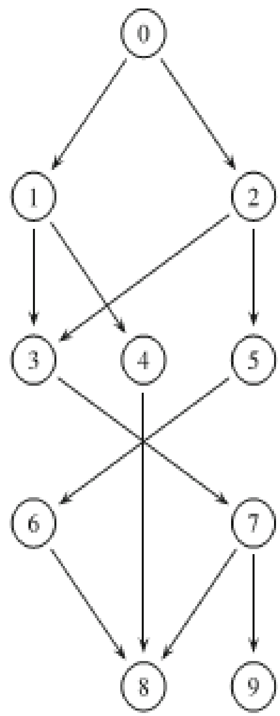


✓ In between, reachability indexes

An example: 2HOP index

Reachability on directed graphs can be reduced to reachability on *directed acyclic graphs (DAGs)*





Each node represents a strongly connected component of the original graph,
an edge if one component can reach another.

An example: 2HOP index

In the transitive closure, for each node u we have the full list of nodes that are reachable from it

Instead of keeping the whole transitive closure, compression

Instead:

For each node in G, a *2 hop-code or label* (Lin(u), Lout(u)) such that

For each pair of nodes u, v in G, v is reachable from u, if and only if,

$$\mathrm{Lin}(u) \cap \mathrm{Lout}(v) \neq \emptyset$$

Variety of graph analysis task

Can we build just *one* system?

Graph data are different

- *Poor locality* which means that random access is often required)

  Accessing the neighbors of a node requires "jumping" around independently of how we represent the graph .

- Graph *structure driven* computation

Talk Outline

Why?

What?

How?

Systems

Overview

Systems for non-graph data

- RDBMS
- MapReduce

Systems for graph data

- Graph database, e.g. Neo4j, HypergraphDB
- Graph analytics system, e.g. Pregel, Trinity, GraphLab
- Matrix-based graph processing system, e.g. Pegasus

RDBMS: traverse graphs using joins

id	name	value	…
1	N1		
2	N2		
3	N3		
4	N4		
5	N5		
6	N5		
…			

V: Vertex Table

src	dst
1	2
2	6
1	4
1	6
3	5
…	

E: Edge Table

Get neighbors of N1

```
SELECT  *
FROM V
LEFT JOIN E ON V.ID = E.dst
WHERE E.src = 1;
```

Get neighbors of N1 at distance 2?

RDBMS

- Widespread use
- Strong consistency guarantees via ACID transactions
- Support complex queries
  - Standard query language: SQL
- May be useful when not pure structural to ***filter*** out parts of the graph based on attributes

MapReduce

General purpose data processing platform optimized for *offline analytics* on large data *partitioned* on hundreds of machines (no online query support)

MapReduce

- The MapReduce data model is *not a native graph model*
  - Graph algorithms *cannot be expressed intuitively*
- Graph processing is *inefficient* on MapReduce
  - Intermediate results of each iteration need to be materialized
  - *Entire graph structure* need to be sent over the network at each iteration, unnecessary data movement

Pregel: Think like a vertex

Pregel: A large-scale distributed framework for graph data developed by Google

Giraph: open-source implementation of Pregel

Graph computation is modeled as many supersteps

- Each vertex reads messages sent in previous the previous superstep
- Each vertex performs computation in parallel
- Each vertex can send messages to other vertices in the end of an iteration

APACHE GIRAPH

The Graph-Parallel Abstraction

A user-defined **Vertex-Program** runs on each vertex

Graph constrains interaction along edges

- Using messages (e.g. Pregel)
- Through shared state (e.g., GraphLab)

Parallelism is achieved by running multiple vertex programs simultaneously

Termination

- Algorithm termination is based on every vertex *voting to halt*
- The algorithm as a whole terminates when all vertices are simultaneously inactive

Implementation of Pregel

- Basic architecture (similar to MapReduce)
  - Master ← *coordinates* computation
  - Workers ← *perform* computation

- Basic stages
  1. The master partitions the graph
  2. The master assigns the input to each Worker
  3. Supersteps begin at Workers
  4. The master can tell Workers to save graphs

Pregel-like Parallelism

- Bulk synchronous parallel model
- Exploits fine-grained parallelism at *node level*

(+) *does not move graph partitions* over the network, only messages at the end of each iteration
(-) *not many graph algorithms* can be implemented using vertex-based computation model elegantly

Pegasus: matrix

www.cs.cmu.edu/~pegasus

An open source large graph mining system
Implemented on Hadoop

Key Idea

Convert graph mining operations into iterative matrix-vector multiplication

A graph with n vertices is represented by an $n \times n$ matrix

each cell (i, j) in the matrix represents an edge (src=i, dst=j)

Pegasus

- A matrix represents a graph
  - Each column or row represents a node
  - $m_{i,j}$ represents the weight of the edge from *i* to *j*
- A vector represents some value of the nodes, e.g., PageRank

Main Idea – Generalized Iterative Matrix-Vector Multiplication (GIM-V)

- The matrix-vector multiplication $M \times v = v'$ where $v_i' = \sum_{j=1}^{n} m_{i,j} v_j$

- Three operations in the above formula
  - **combine2**: multiply $m_{i,j}$ and v_j
  - **combineAll**: sum n multiplication results for a node *i*
  - **assign**: overwrite the previous value of v_i with a new result to make $v_i{}'$

Generalized Iterative Matrix-Vector Multiplication (GIM-V)

- The operator $\times_G$ is defined as:
  - $v' = M \times_G v$, where v_i' = assign(v_i, combineAll$_i$($\{x_j \mid j$=1, …, n and x_j = combine2($m_{i,j}$, v_j)$\}$))
    - combine2($m_{i,j}$, v_j): combine $m_{i,j}$ and v_j
    - combineAll$_i$($x_1, \ldots, x_n$): combine all the results from combine2() for a node *i*
    - assign(v_i, v_{new}): decide how to update v_i with v_{new}

- $\times_G$ is applied until a convergence criterion is met
- Customizing these three functions implements several graph mining operations

GIM-V and PageRank (skipped)

GIM-V and Connected Components (skipped)

Pegasus

Matrix based graph mining platform

(+) Support large scale graphs
(-) Many graph operations cannot be modeled by matrix-vector multiplications
(-) Not a very natural programming model

Graph Database

Data model

- A property graph: nodes and directed edges
- Node and edges can have properties
  - Properties are key-value pairs
    - Keys are strings; values are arbitrary data types
- The basic operation is a *traversal*
  - It starts from a *given node* and *explores* portions of the graph based on the query

Neo4j

- Neo4j is an open source graph database
- Cypher is a graph query language implemented in Neo4j

Cypher

- The most basic Cypher query (traversal) includes the following structure:
  - Starting node(s)
    - used to limit the search to certain areas of the graph
    - Found via node ID, list of node IDs, or by an index lookup
  - Pattern matching expression
    - for examining patterns in relationships
    - w.r.t. the starting node(s)
  - Return expression
    - based on variables bound in the pattern matching
    - defines retrieval set of nodes, relationships, or properties of nodes or relationships

Graph Databases

(+) Powerful data model
(+) Fast for connected data

(-) How to partition/distribute
sharding on key-value and key-document on NoSQL databases, difficult for graph databases
Memory sharding

Systems

No consensus on a single model

Partition is a big issue when it comes to distribution

Our own work

Graph evolve over time

- How to query graph history (indexes)
- Diffusion (propagation of information in graphs)

Summary

Why?

Graphs are ubiquitous -- many useful applications

What?

Variety of problems for online and offline analysis

How?

Many graph-oriented parallel processing systems and databases

Take-away message

Very active area of research

Many research problems and opportunities for most computer science fields

Graphs as a Service?

Thank you!