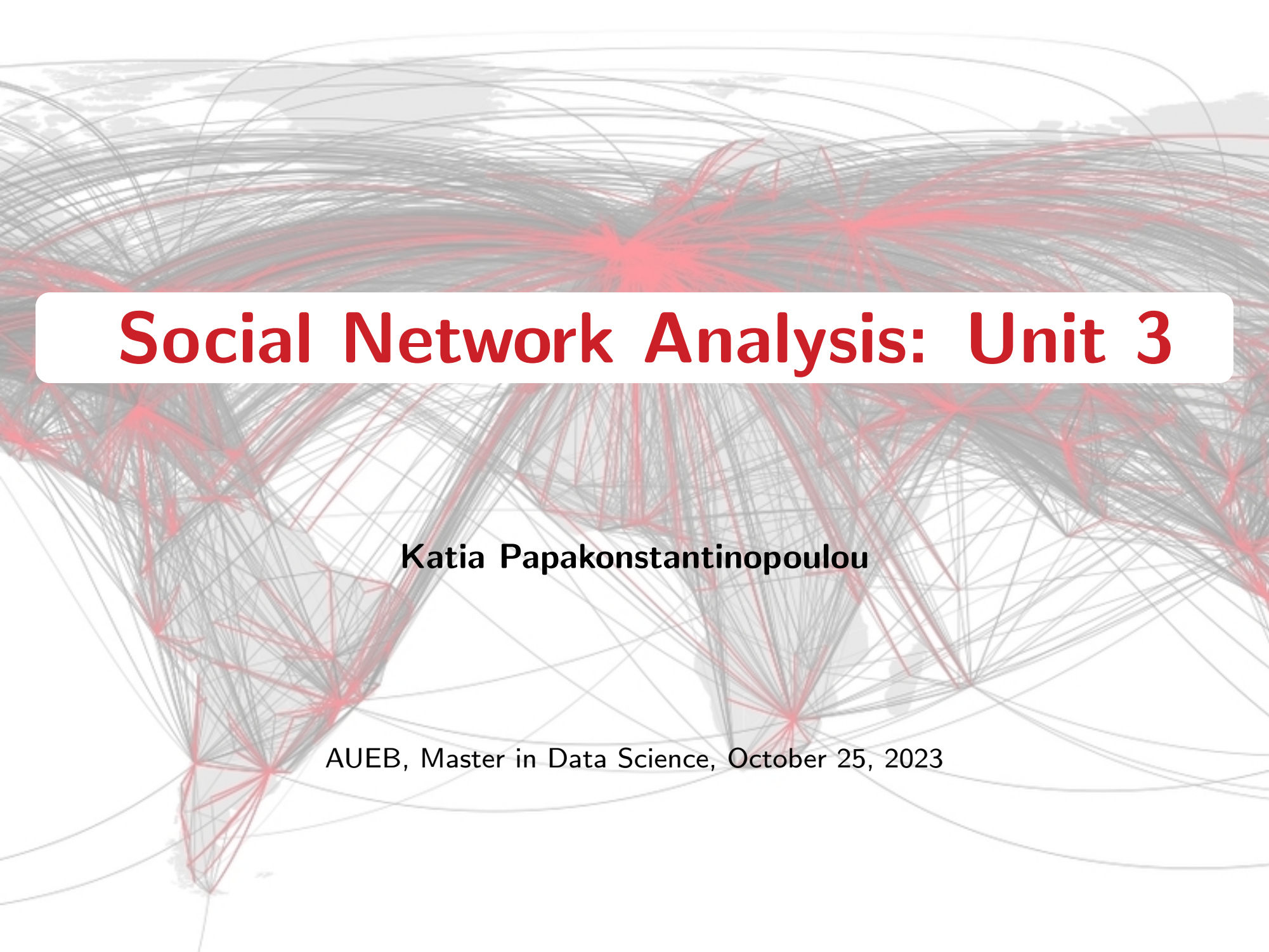


Social Network Analysis: Unit 3

Katia Papakonstantinopoulou

AUEB, Master in Data Science, October 25, 2023

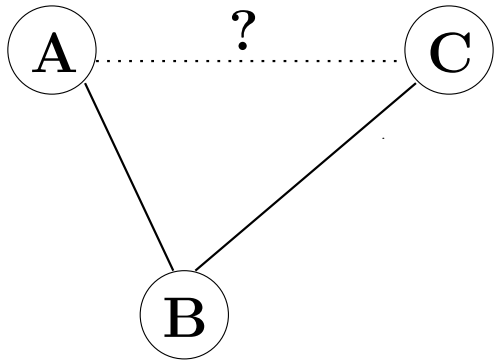
Outline:

- Transitivity, triadic closure
- Clustering
- Strong / weak ties
- Network motifs
- Game Theory basics

Transitivity, triadic closure

Transitivity

If A is connected to B and B is connected to C,
what is the probability that A is connected to C?



Main Idea:

My friends' friends are likely to be my friends!

Clustering

Transitivity leads to the creation of clusters.
The tendency of nodes to cluster together is measured by the coefficient below.

Global clustering coefficient

It is a measure of the degree to which nodes in a graph tend to cluster together.
It is defined as: $C = \frac{3\times\text{number of triangles in the graph}}{\text{number of connected triples of vertices}}$.

Obviously, $0 \leq C \leq 1$.

Local clustering coefficient (Watts & Strogatz 1998)

It expresses the fraction of pairs of neighbors of the node that are themselves connected.
For a vertex i it is defined as:

$$C_i = \frac{\#\text{ of connections among } i\text{'s neighbors}}{\max \#\text{ of possible connections among } i\text{'s neighbors}}.$$

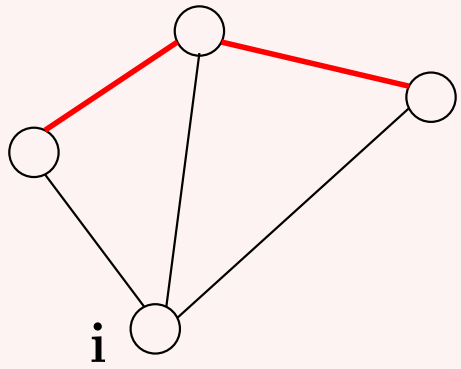
If d_i is the degree of i, $C_i = \frac{\#\text{ of connections among } i\text{'s neighbors}}{d_i(d_i-1)/2}$.
We also define the directed version of this coefficient:

$$C_{i,\text{ directed}} = \frac{\#\text{ of directed connections among } i\text{'s neighbors}}{d_i(d_i-1)}.$$

The average over all n vertices is $C = \frac{1}{n}\sum_i C_i$.

Local clustering coefficient

Example: Computation of clustering coefficient C_i in the network below

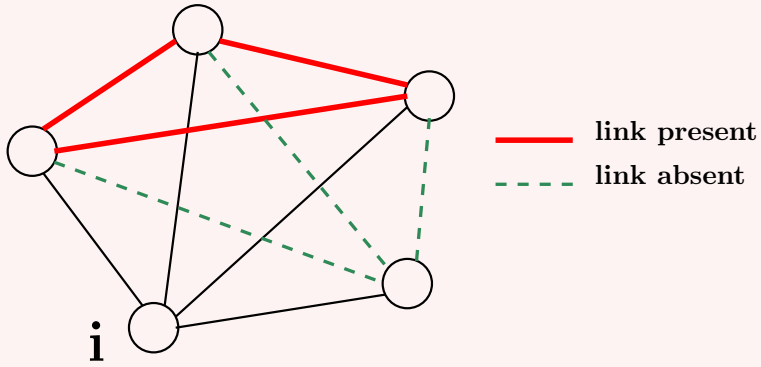


$d_i = 3 \Rightarrow \max$ number of connections $= 3$,
2 connections present,
Hence $C_i = \frac{2}{3}$.

Local clustering coefficient

Quiz

What is the local clustering coefficient of i in the network below?



Answer

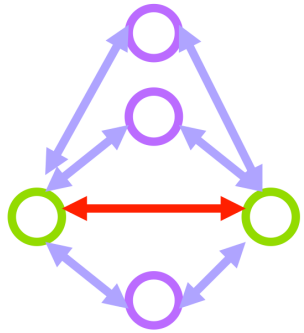
$d_i = 4 \Rightarrow \max$ number of connections $= 4 \cdot \frac{3}{2} = 6$,
3 connections present,
Therefore $C_i = \frac{3}{6} = 0.5$.

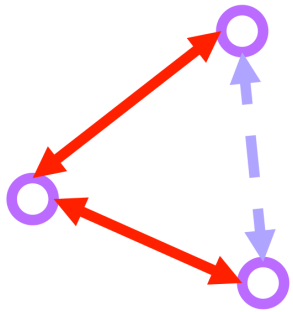
Strong ties

We now move to the level of ties. Ties may be *strong* or *weak*.

Strong ties in social networks usually express:

- frequent contact
- affinity
- many mutual contacts





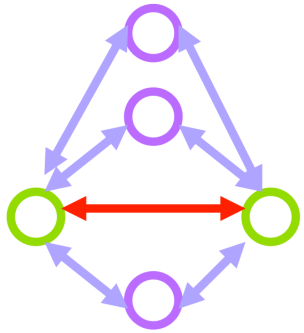
Strong ties are likely to 'close' !

Edge Embeddedness & Neighborhood Overlap

Measures that can be used to assess the importance of an edge:

Embeddedness

is defined as number of common neighbors A and B have



Neighborhood Overlap

is defined as

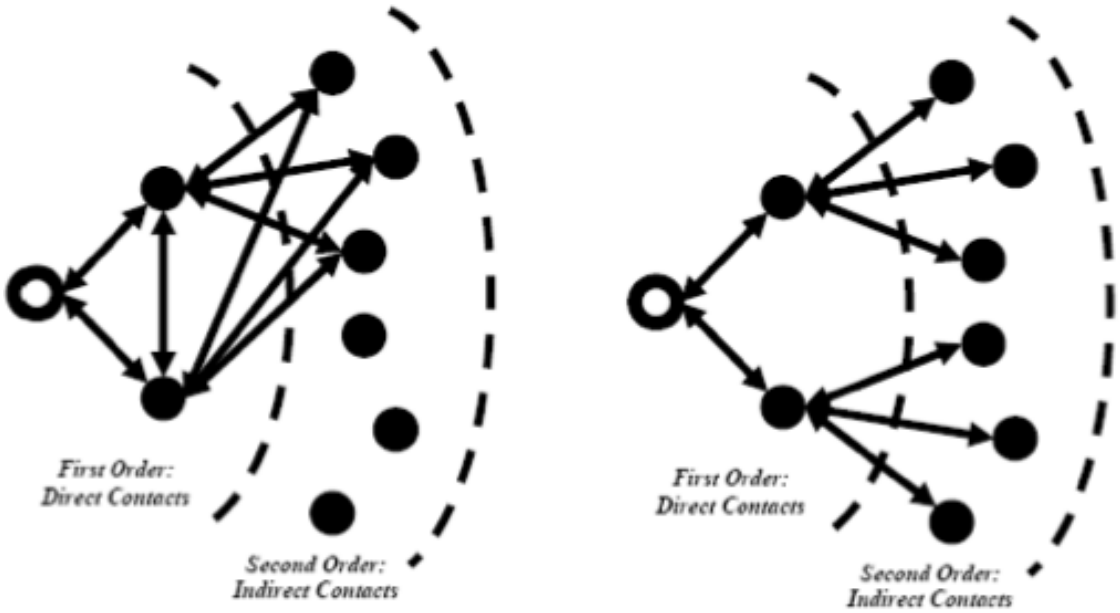
$$\frac{\#\text{ of nodes who are neighbors of both A and B}}{\#\text{ of nodes who are neighbors of at least one of A or B}}$$

Note: High embeddedness $\Rightarrow$ high neighborhood overlap.

Edge Embeddedness – Example

Snowball sampling

Will you reach more different kids by asking each kid to name their 2 best friends, or their 7th and 8th closest friend?



Source: M. van Alstyne, S. Aral. Networks, Information & Social Capital

Is it good to be embedded?

- What are the advantages of occupying an embedded position in the network?
- What are the advantages of being a broker?

The strength of intermediate ties

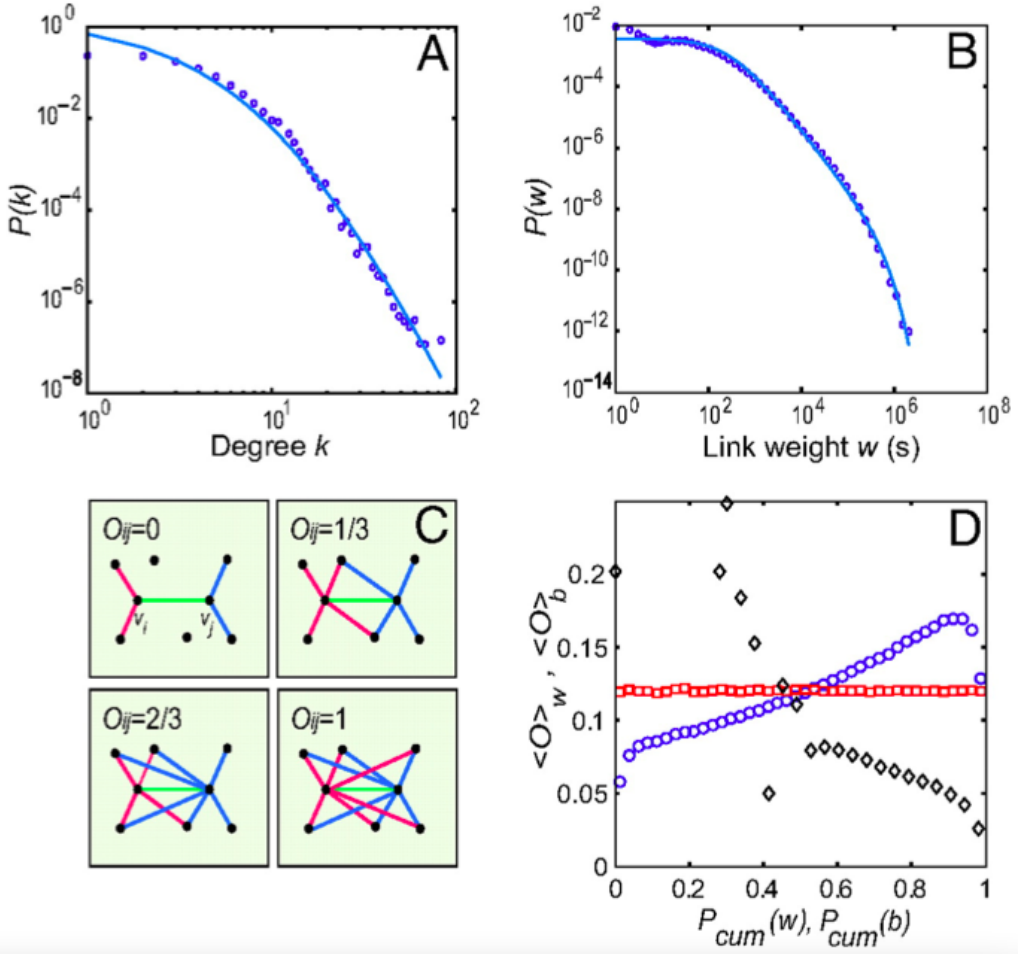
Study of a large call graph: use nation-wide cellphone call records and simulate diffusion using actual call timing (Onnela J. et.al. PNAS 2007;104:7332-7336)

- strong ties: frequent communication, but ties are redundant due to high clustering
- weak ties: reach far across network, but communication is infrequent...

In simulation, individuals are most likely to obtain novel information through ties of *intermediate* strength

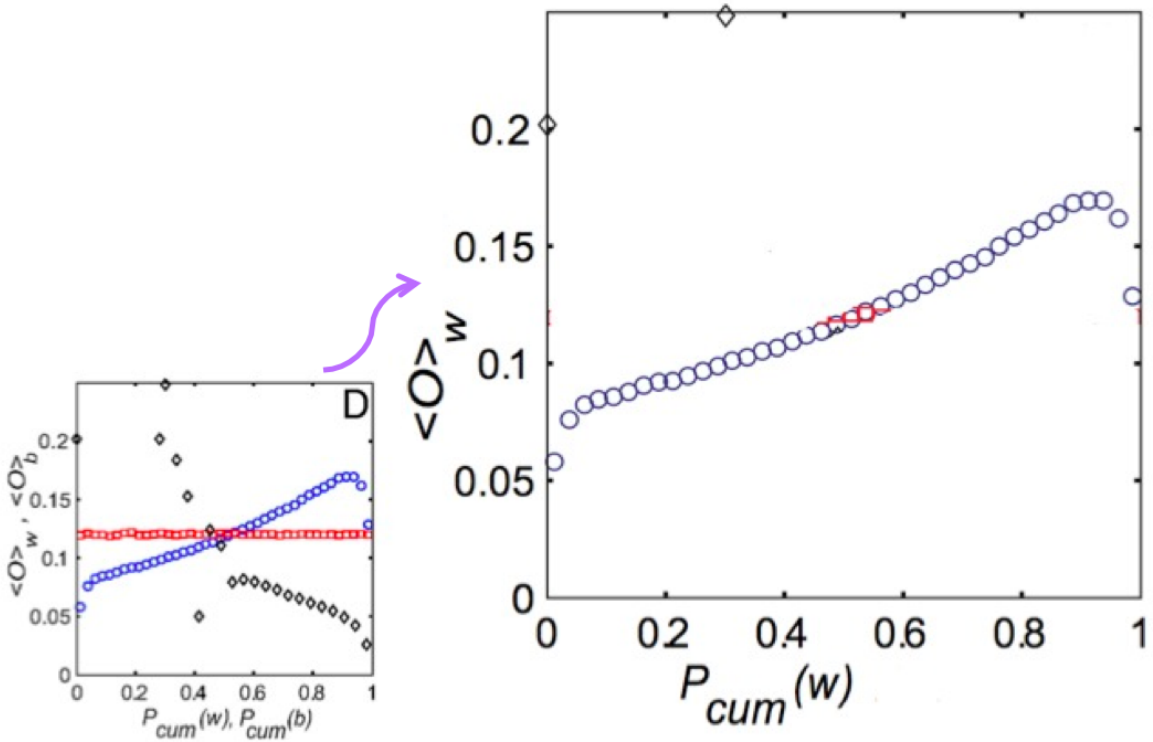
Call graph example

Characterizing the large-scale structure and the tie strengths



Call graph example (cont'd)

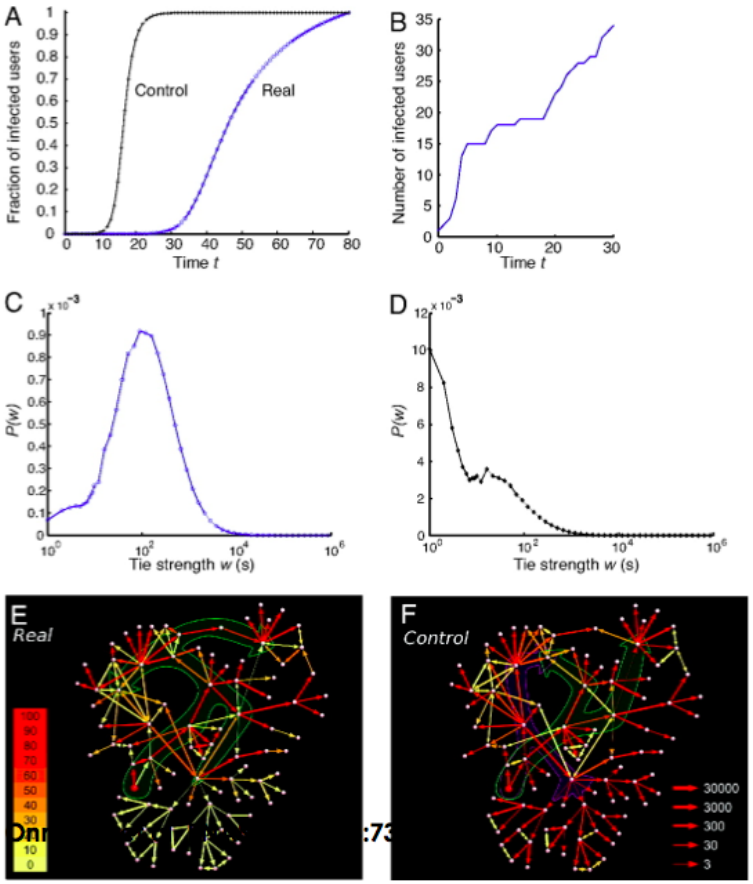
Edge neighborhood overlap as a function of tie strength



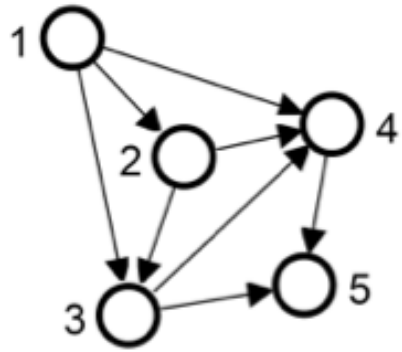
Positive relationship between embeddedness and frequency of communication (i.e., embedded ties use to be closer ties)

Call graph example (cont'd)

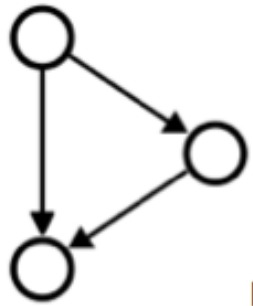
The dynamics of spreading on the weighted graph, assuming node v_i passes on the information to its neighbor v_j in one time step with probability $P_{ij} = xw_{ij}$, with $x = 2..59 \times 10^{-4}$.



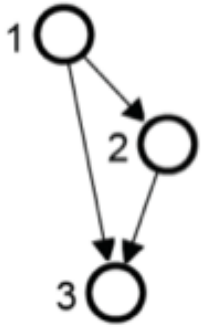
Resolving local structure: network motifs



graph

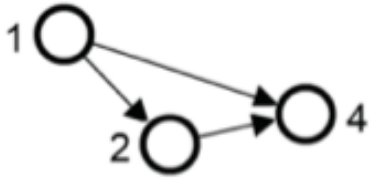


motif to be found

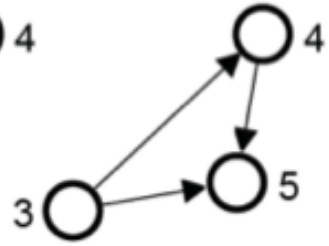


M1

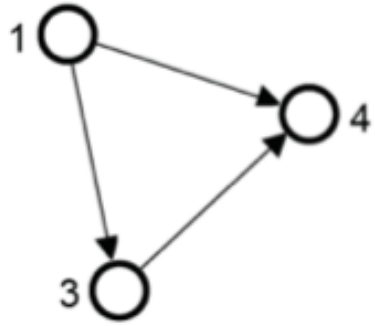
M2



M3



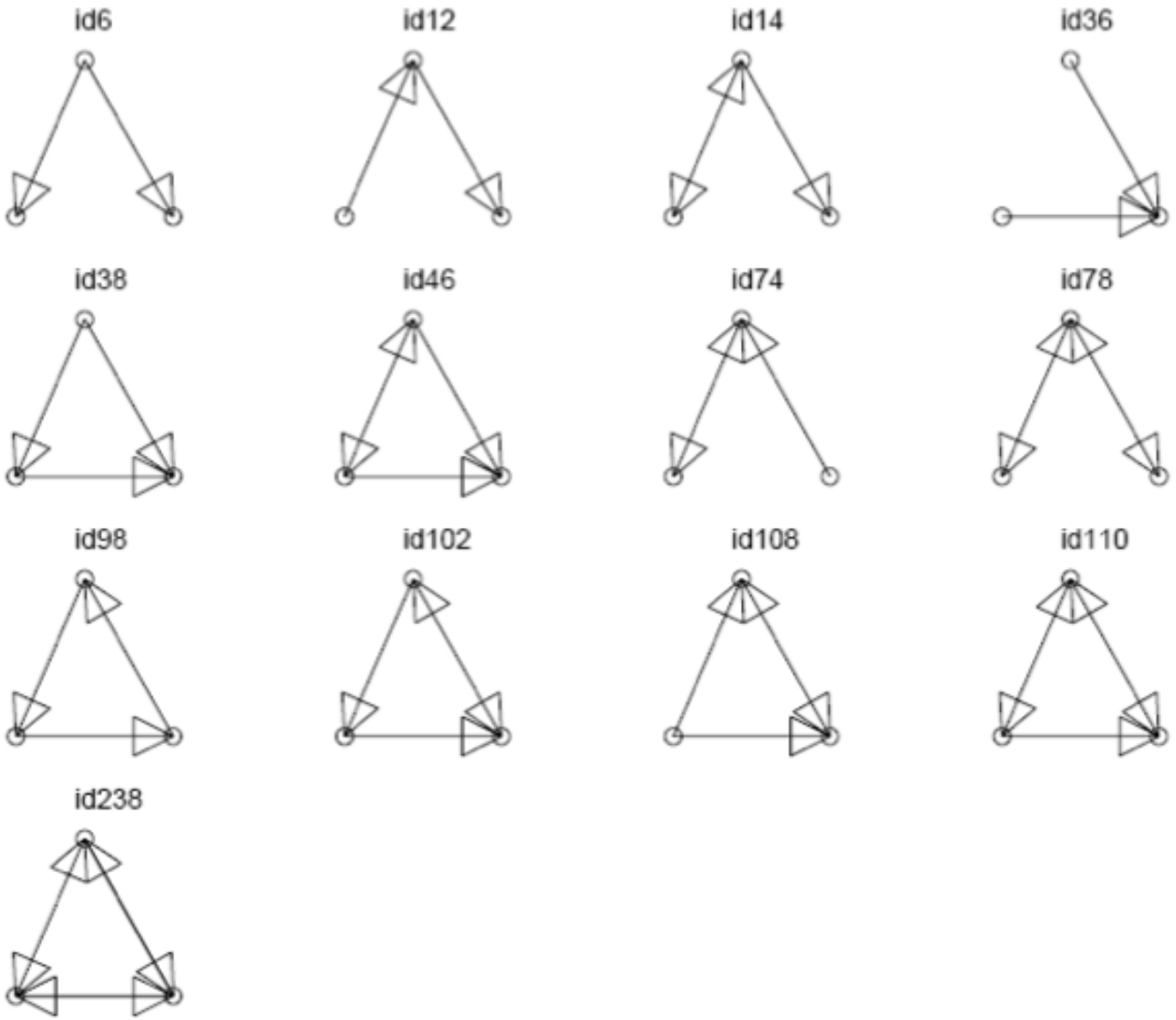
M4



M5

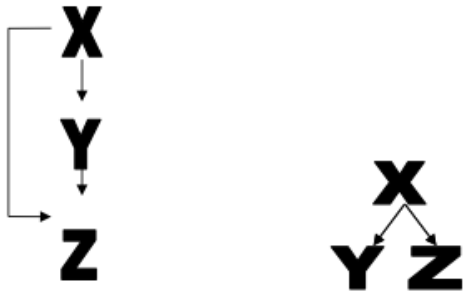
motif matches in the target graph

All 3-node motifs



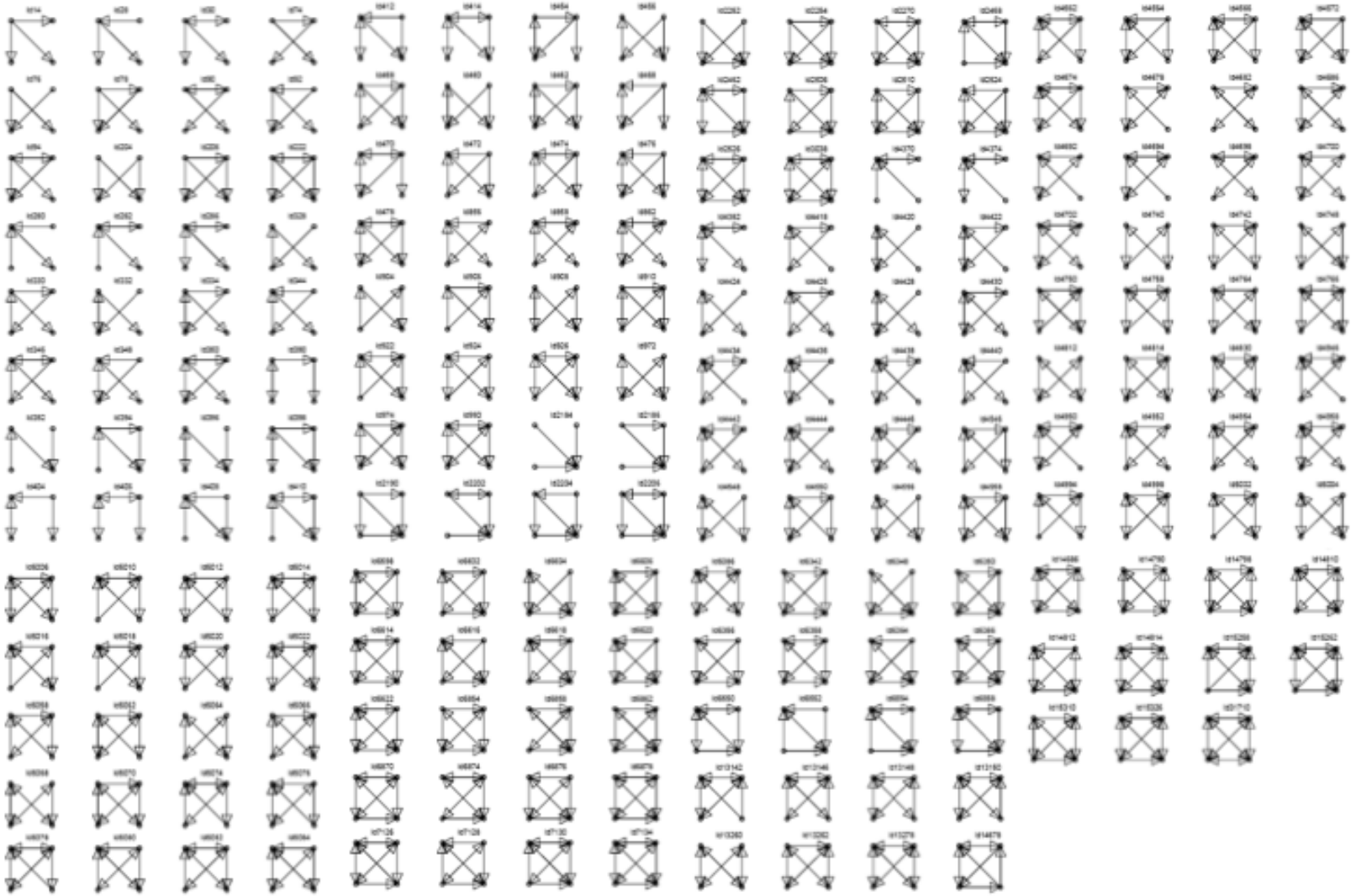
Examples of 3-node network motifs

Feed forward loop (NNs), Single-Input Module (gene control networks):



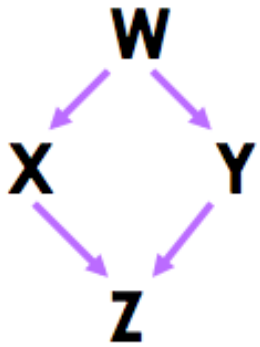
All 4-node motifs

Computational expense increases with the size of the graph!



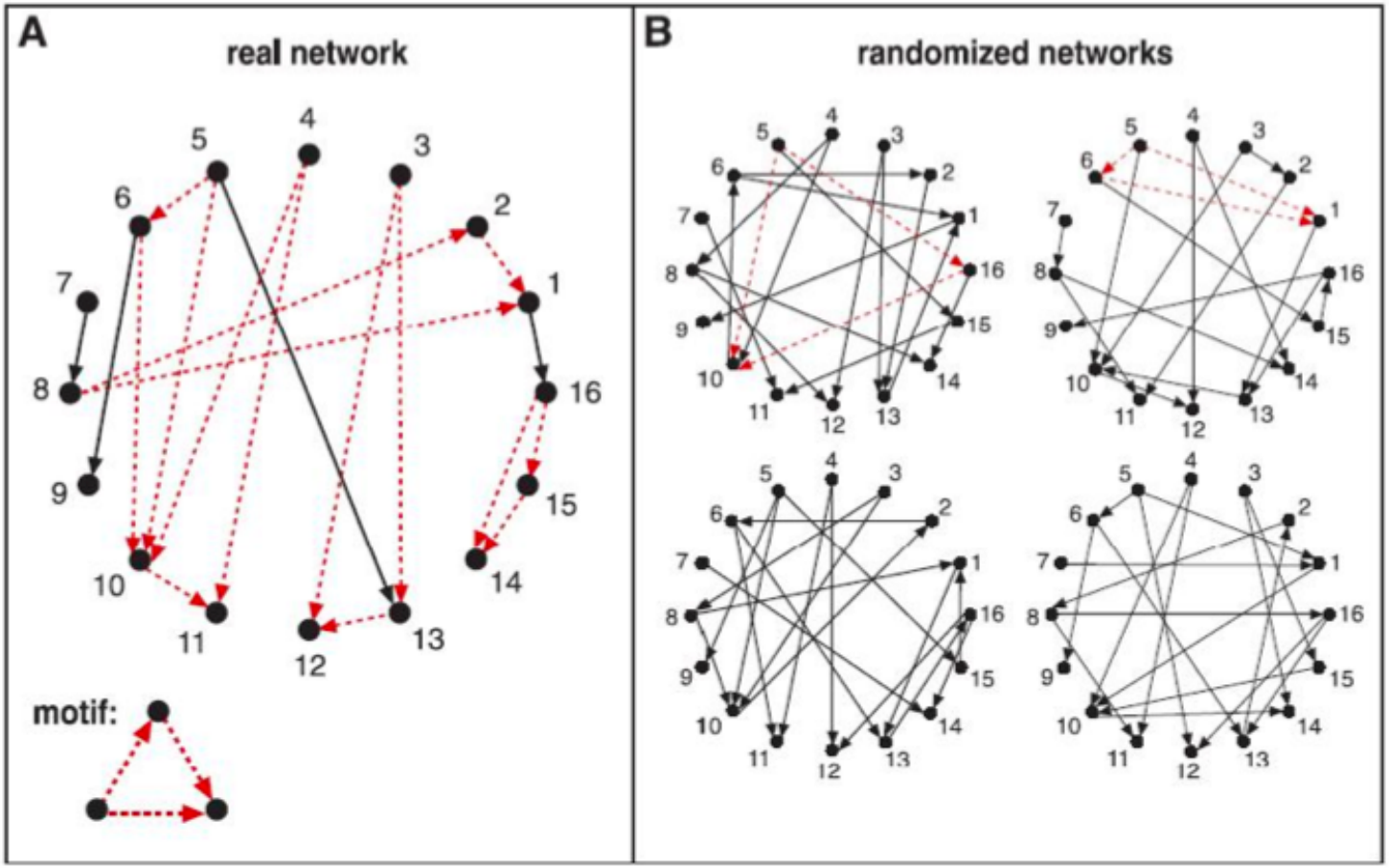
Example of 4-node network motif

Parallel paths (NNs, food webs)



Comparing to 'equivalent' random networks

Observation: The motif below (Fig. A) is met much more often in real networks than in 'equivalent' random networks (Fig. B).



Milo et al., Network motifs: Simple building blocks of complex networks, Science 298:824-827, 2002

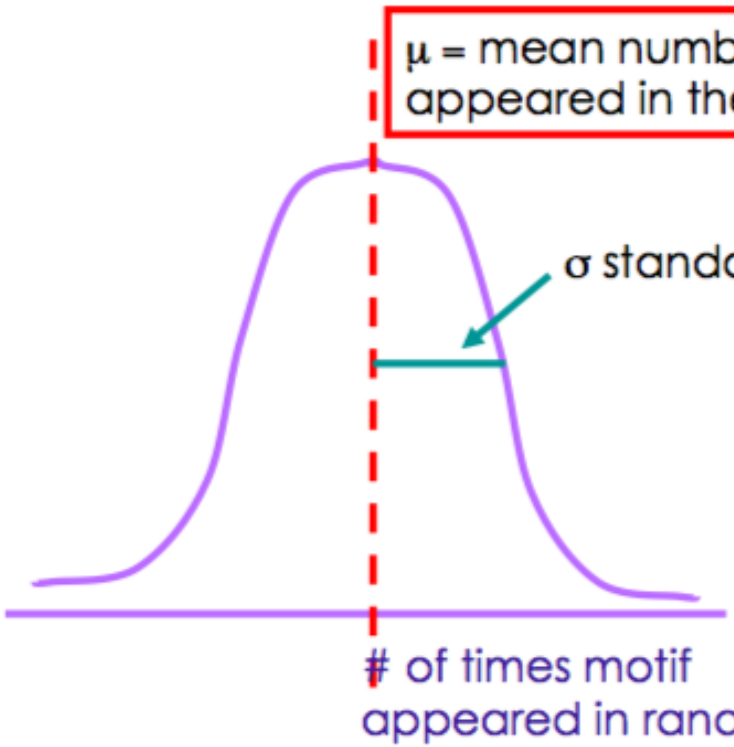
Network motif detection

Main idea: Some motifs will occur more often in real world networks than random networks

Method:

- construct many random graphs with the same number of nodes and edges (same node degree distribution?)
- count the number of motifs in those graphs
- calculate the Z score: the probability that the given number of motifs in the real world network could have occurred by chance

What the Z score means



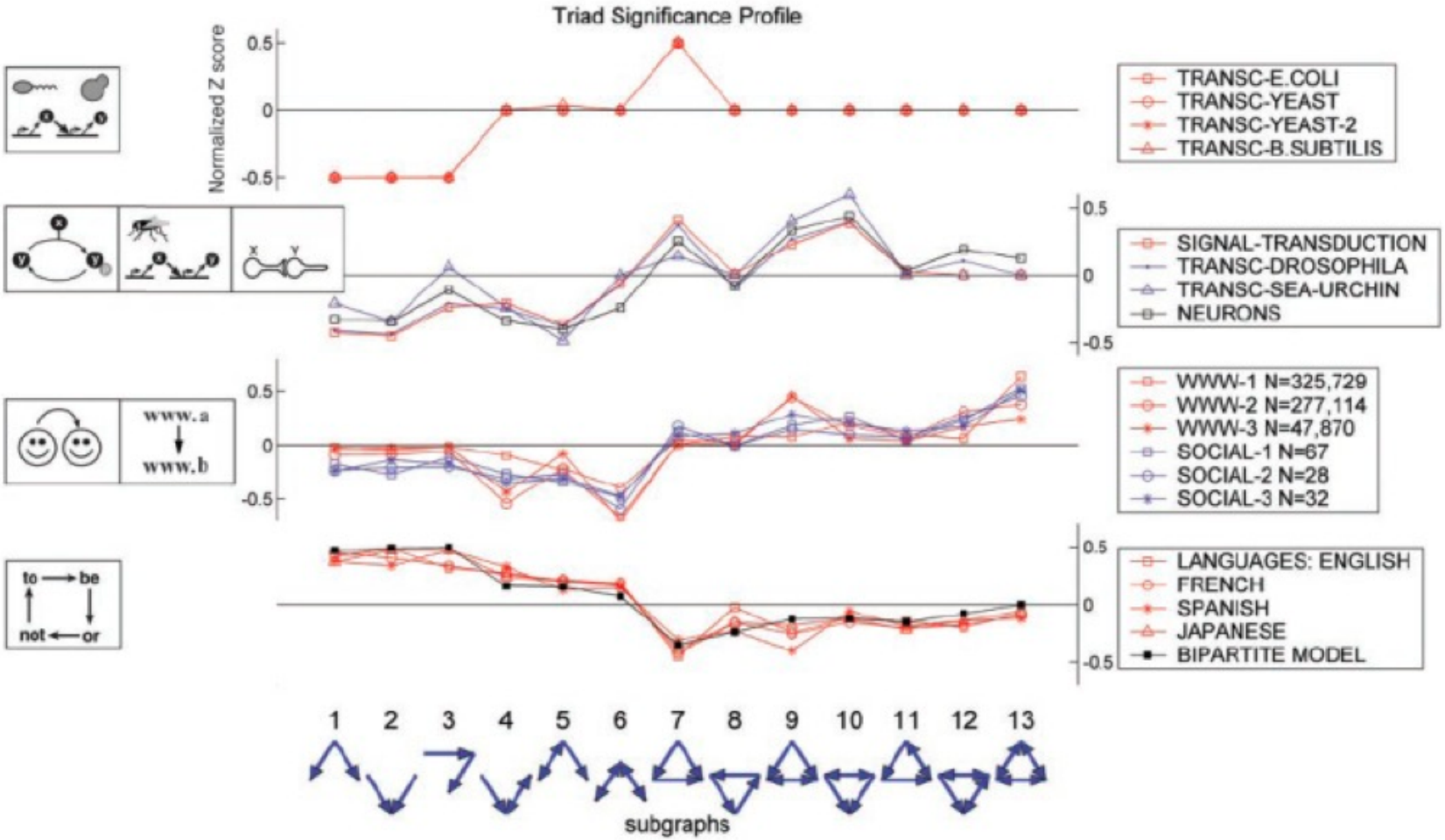
the probability observing a Z score of 2 is 0.02275

In the context of motifs:
Z > 0, motif occurs more often than for random graphs
Z < 0, motif occurs less often than in random graphs

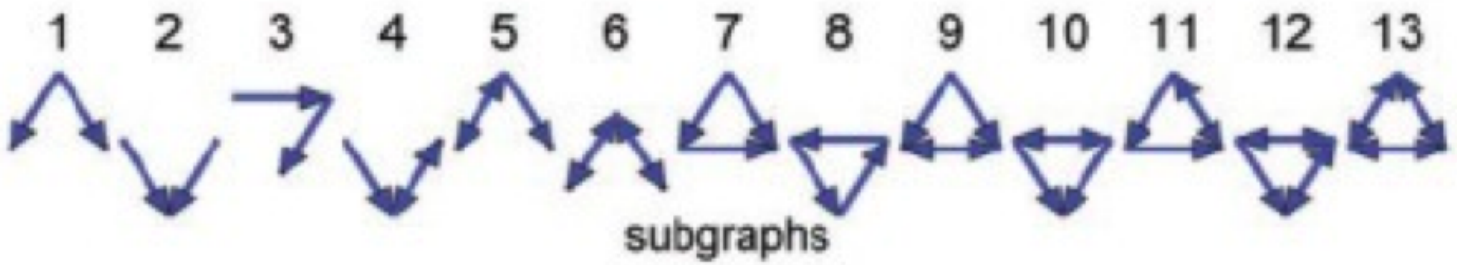
$$Z_x = \frac{x - \mu_x}{\sigma_x}$$

|Z| > 1.65, only a 5% chance of random occurence

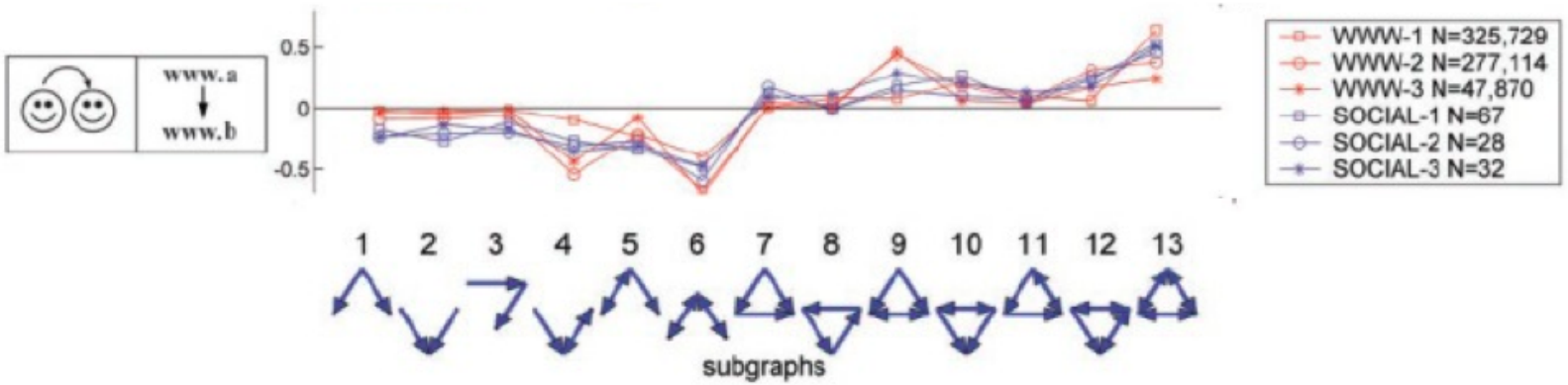
superfamilies



superfamilies



superfamilies



Motifs

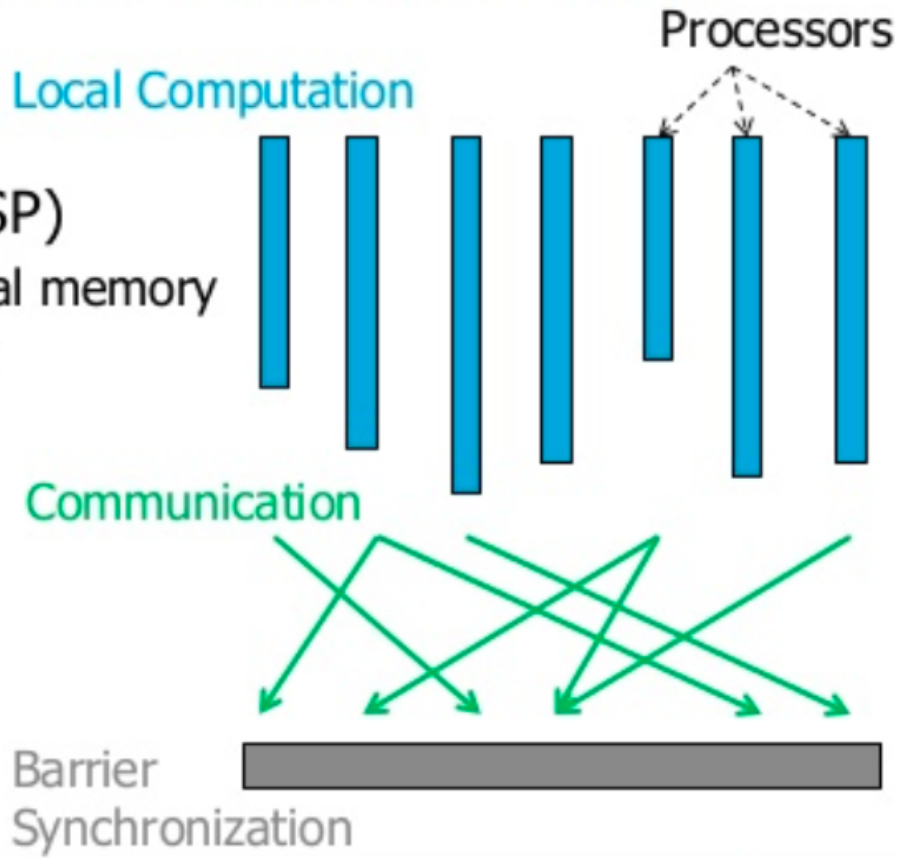
- Given a particular structure, search for it in the network, e.g. complete triads
  - Advantage: motifs can correspond to particular functions, e.g. in biological networks
  - Disadvantage: don't know if motif is part of a larger cohesive community
- We can construct superfamilies of networks based on the pattern the Z score follows.

Pregel is a programming model for graph processing

Based on Valiant's
Bulk Synchronous Parallel (BSP)

- N processing units with fast local memory
- Shared communication medium
- Series of **Supersteps**
- Global Synchronization Barrier
- Ends when all voteToHalt

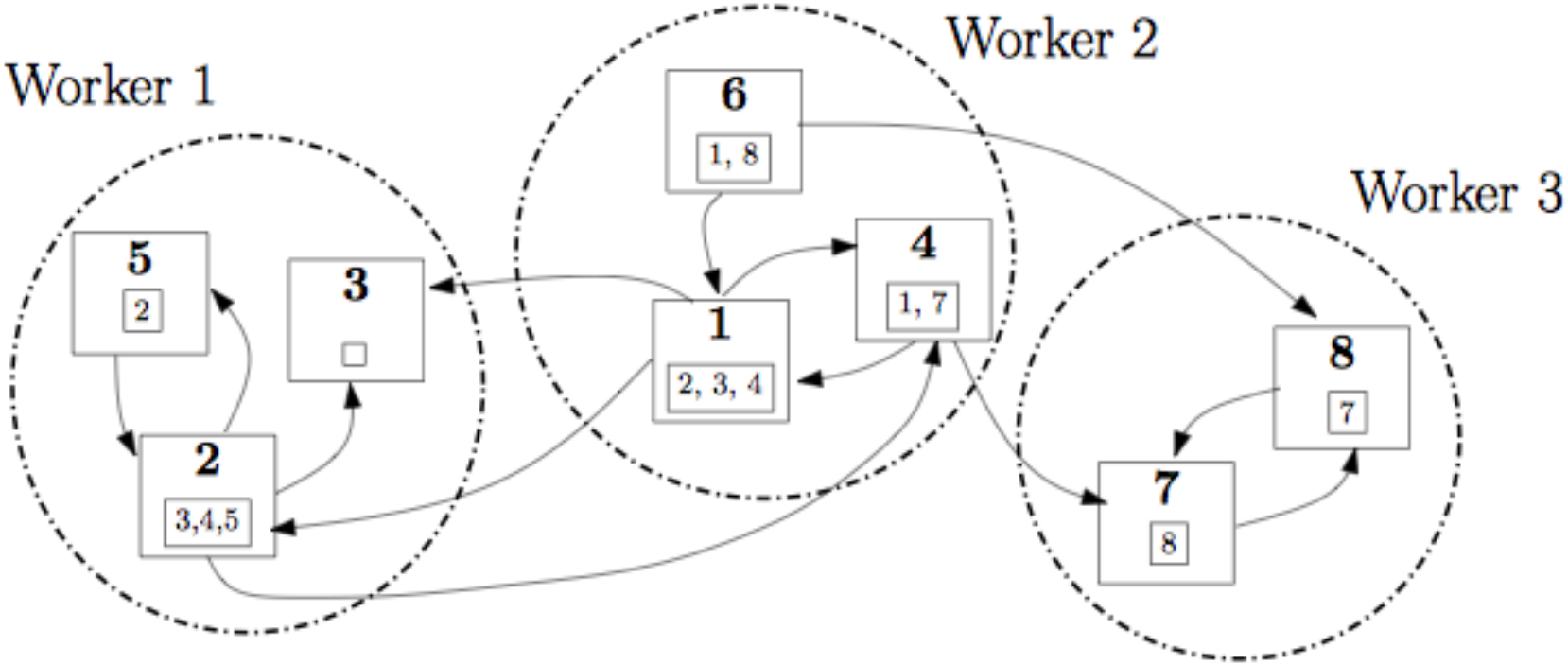
- Pregel executes initialization, one or several supersteps, shutdown



Pregel - Features

- Batch-oriented processing
- Runs in-memory
- Vertex-centric API
- Fault-tolerant
- Follows Master-Slave architecture

Pregel - The Vertex-Based API



Pregel data flow model

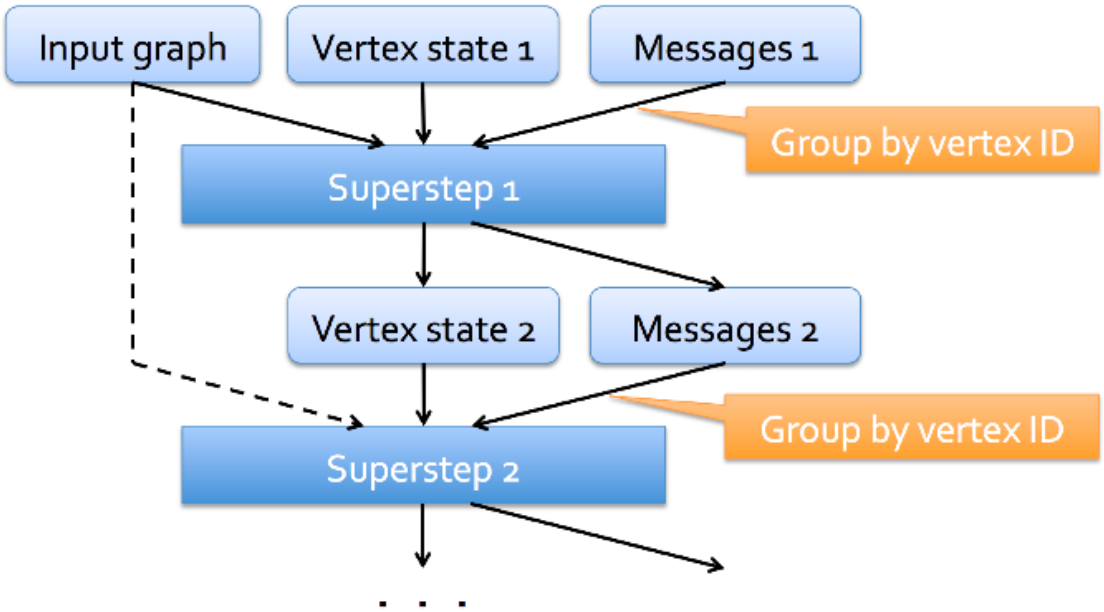


Figure 1: The Pregel paradigm's data flow model.

Pregel - The Superstep

- Each Vertex (execution in parallel)
  - Receive messages from other vertices
  - Perform own computation (user-defined function)
  - Modify own state or state of outgoing messages
  - Mutate topology of the graph
  - Send messages to other vertices
- Termination Condition
  - All vertices inactive
  - All messages have been transmitted

Pregel - The Master-Worker Architecture

- Master assigns vertices to Workers ($\rightsquigarrow$ graph partitioning)
- Master coordinates Supersteps
- Master coordinates Checkpoints
- Workers execute vertices compute()
- Workers exchange messages directly

LAB SESSION: Intro to Apache Giraph

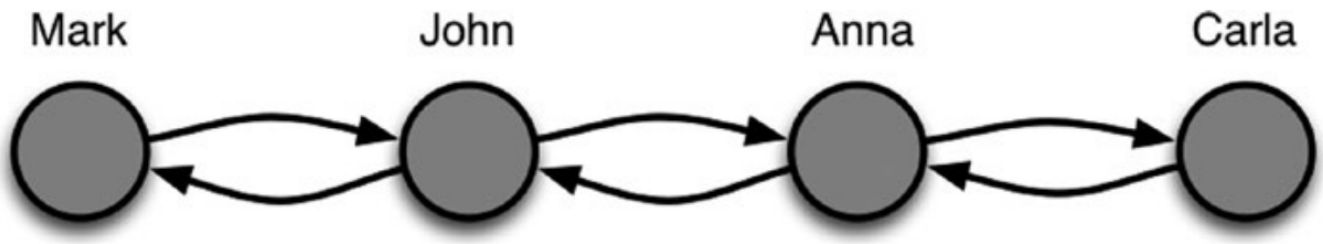
Apache Giraph

A graph processing system

It is an open-source implementation of Pregel

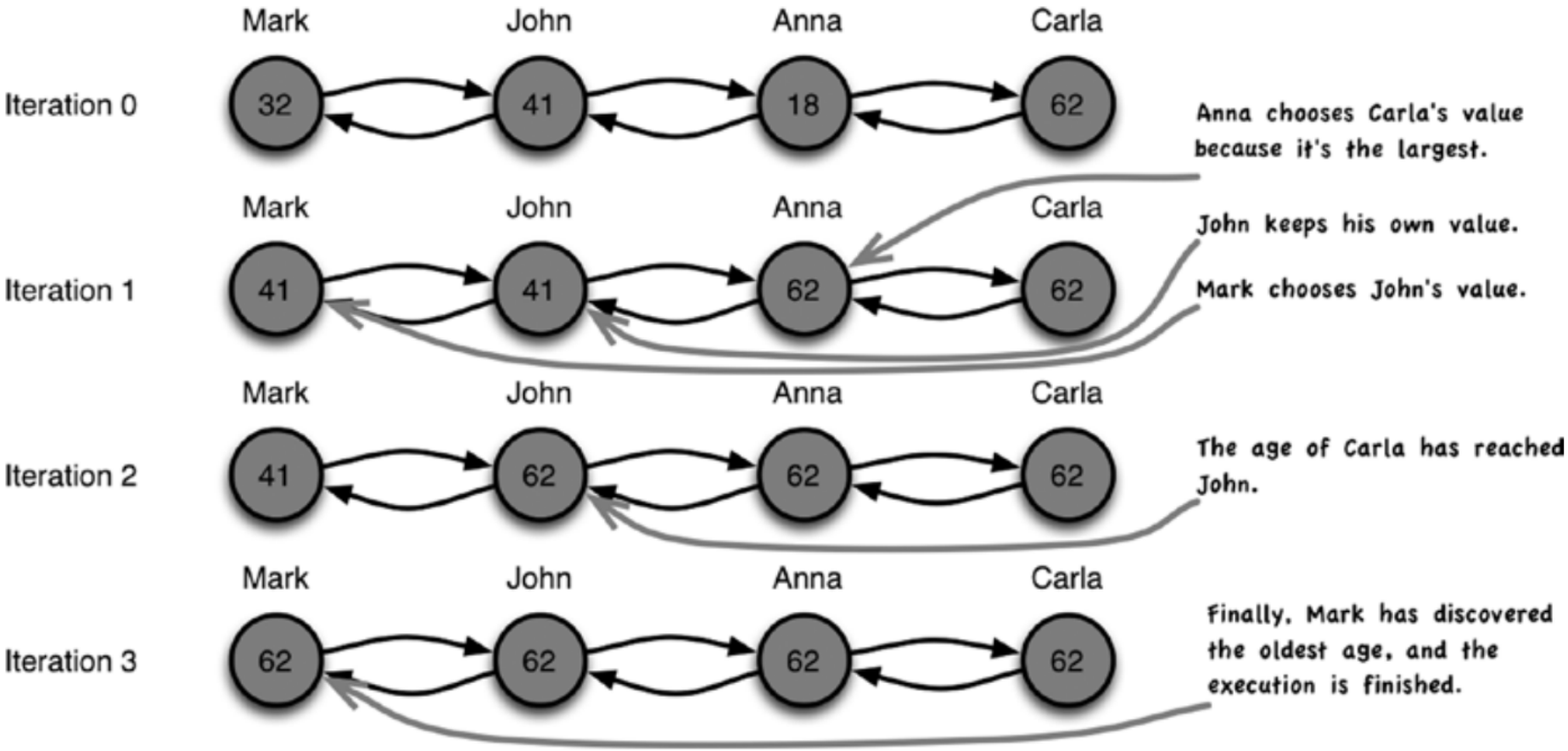
Problem

Find out who of the following individuals is the oldest:



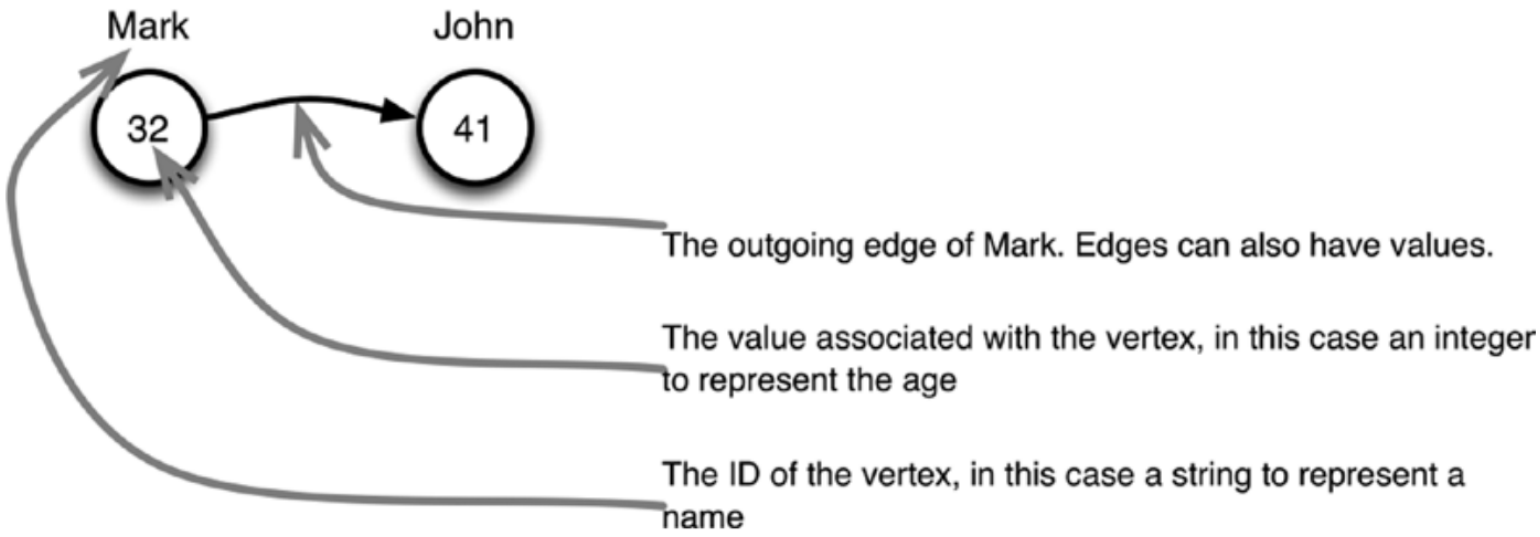
We can assign to each vertex a value that we initialize with the age of the person,
and solve the problem *recursively* by defining the value of each vertex as *the largest value among its own value and the values of its neighbors*.

Iterative Graph Algorithms (cont'd)



Pregel (or Apache Giraph) asks developers to "think like a vertex"!

Apache Giraph Data Model



A graph is a set of vertices and edges, and each vertex:

- Has a unique ID, defined by a type (an integer, a string, and so on)
- Has a value, also defined by a type (a double, an integer, and so on)
- Has a number of outgoing edges that point to other vertices.
  - Each edge can have a value, also defined by a type (an integer, a double, and so on).

Apache Giraph Vertex Class

class Vertex:

- function getId() #1
- function getValue() #2
- function setValue(value) #3
- function getEdges() #4
- function getNumEdges() #5
- function getEdgeValue(targetId) #6
- function setEdgeValue(targetId, value) #7
- function getAllEdgeValues(targetId) #8
- function voteToHalt() #9
- function addEdge(edge) #10
- function removeEdges(targetId) #11

Apache Giraph Vertex Class (cont'd)

- #1 Returns the ID of the vertex.
- #2 Returns the value of the vertex.
- #3 Sets the value of the vertex.
- #4 Returns all the outgoing edges for the vertex in the form of an iterable of Edge objects.
- #5 Returns the number of outgoing edges for the vertex.
- #6 Returns the value of the first edge connecting to the target vertex, if any.
- #7 Sets the value of the first edge connecting to the target vertex, if any.
- #8 Returns the values associated with all the edges connecting to a specific vertex.
- #9 Makes the vertex vote to halt.
- #10 Adds an edge to the vertex.
- #11 Removes all edges pointing to the target vertex.

Apache Giraph Edge Class

class Edge:

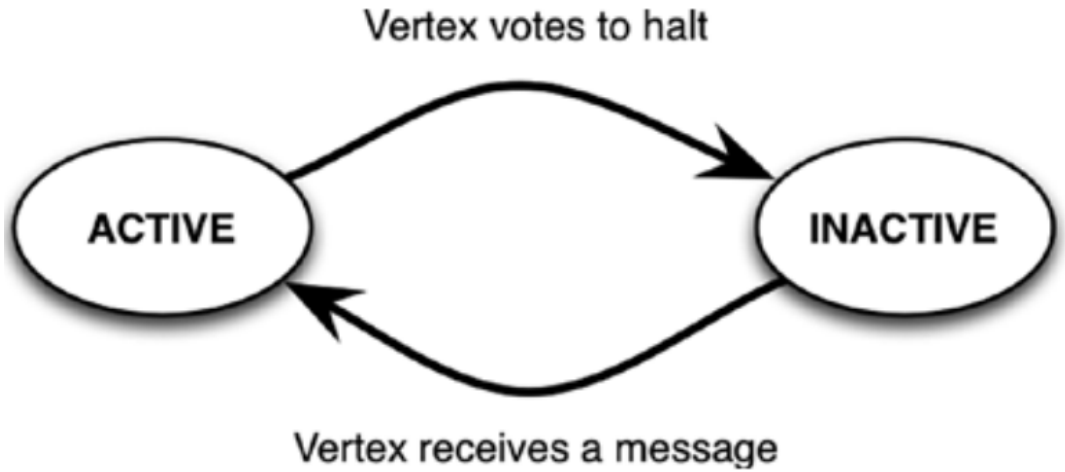
- function getTargetVertexId() #1
- function getValue() #2
- function setValue(value) #3

- #1 Returns the ID of the target vertex.
- #2 Returns the value attached to the edge.
- #3 Sets the value of the edge.

A Giraph computation is organized in a series of supersteps.

At each superstep, a vertex can send messages to other vertices, access its vertex value and its edges, and vote to halt.

Sent messages are delivered to the destination vertex at the beginning of the next superstep.



Apache Giraph BasicComputation Class

class BasicComputation:

- function compute(vertex, messages) #1
- function getSuperstep() #2
- function getTotalNumVertices() #3
- function getTotalNumEdges() #4
- function sendMessage(targetId, message) #5
- function sendMessagetoAllEdges(vertex, message) #6
- function addVertexRequest(vertexId) #7
- function removeVertexRequest(vertexId) #8

Apache Giraph BasicComputation Class (cont'd)

- #1 The method to implement, which is called by the Giraph runtime.
- #2 Returns the current superstep.
- #3 Returns the total number of vertices in the graph.
- #4 Returns the total number of edges in the graph.
- #5 Sends a message to the target vertex.
- #6 Sends a message to the endpoints of all the outgoing edges of a vertex.
- #7 Requests the addition of a vertex to the graph.
- #8 Requests the removal of a vertex from the graph.

MaxValue Algorithm

```
function compute(vertex, messages):
    maxValue = max(messages) #1
    if maxValue > vertex.getValue(): #2
        vertex.setValue(maxValue) #2
        sendMessageToAllEdges(vertex, maxValue) #2
    vertex.voteToHalt() #3
```

- #1 Identify the largest value across those sent as a message.
- #2 The value is larger than the value discovered so far by this vertex, so update and propagate.
- #3 Vote to halt.

MaxValue Execution

Superstep 0:

1. All vertices start active with their value initialized to the age.
2. All vertices send their value to their neighbors as a message.
3. All vertices vote to halt.

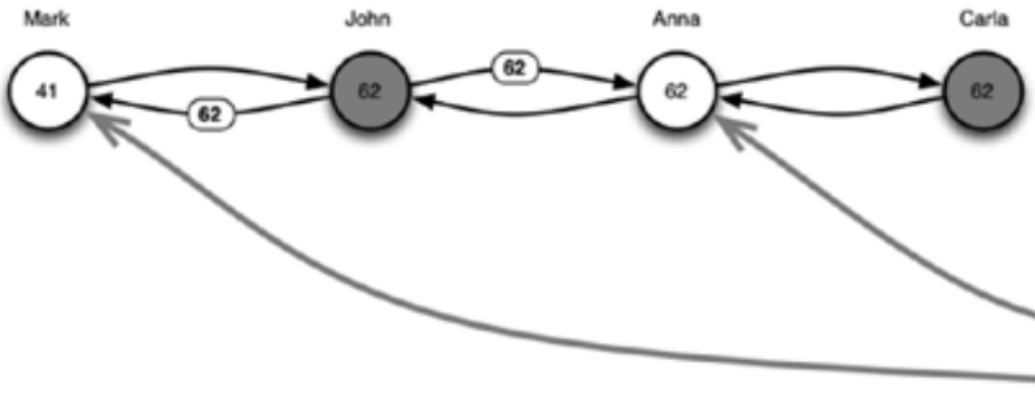
Carla starts propagating her largest value to the neighbors.

Superstep 1:

1. Vertices receive the values from their neighbors.
2. They update their values.
3. They propagate the new value to the neighbors.
4. All vertices vote to halt.

Anna and Mark update their values according to their neighbors.

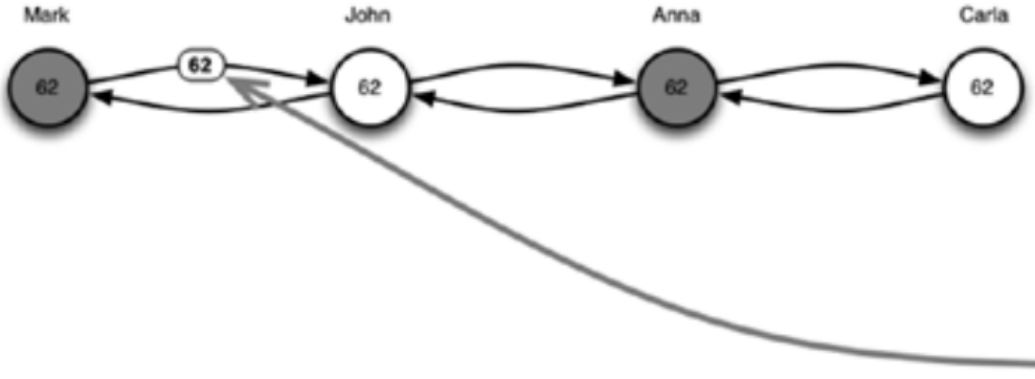
MaxValue Execution (cont'd)



Superstep 2:

1. Vertices receive the values from their neighbors.
2. They update their values.
3. They propagate the new value to the neighbors.
4. All vertices vote to halt.

Anna and Mark are inactive because no messages were sent to them in the previous superstep.



Superstep 3:

1. Vertices receive the values from their neighbors.
2. They update their values.
3. They propagate the new value to the neighbors.
4. All vertices vote to halt.

The largest value has reached Mark. He sends it back to John (vertices do not know who has what value).

MaxValue Execution (cont'd)

Superstep 4:

1. No more updates are necessary.
2. All vertices vote to halt.
3. The computation is over.

John does not need to update his value, so the computation is over.

PageRank algorithm

Algorithm 1 PageRank

input: $G : Graph[V, E])$

while $err \geq \epsilon$ **do**

for vertex i **do**

$$R[i] = 0.15 + 0.85 \sum_{j \in N_{\text{in}}(i)} M[j]$$

$$M[i] = R[i]/|N_{\text{out}}|$$

Send $M[i]$ to all $N_{\text{out}}(i)$

end for

$err = |R - previousR|$

end while

PageRank implementation in Giraph

```
public void compute(
      Vertex<LongWritable, DoubleWritable, FloatWritable> vertex,
      Iterable<DoubleWritable> messages) throws IOException {
    if (getSuperstep() >= 1) {    #1
      double sum = 0;
      for (DoubleWritable message : messages) {    #3
        sum += message.get();    #3
      }
      DoubleWritable vertexValue =
          new DoubleWritable((0.15f / getTotalNumVertices()) + 0.85f * sum);
      vertex.setValue(vertexValue);    #1
      aggregate(MAX_AGG, vertexValue);
      aggregate(MIN_AGG, vertexValue);
      aggregate(SUM_AGG, new LongWritable(1));
      LOG.info(vertex.getId() + ": PageRank=" + vertexValue +
          " max=" + getAggregatedValue(MAX_AGG) +
          " min=" + getAggregatedValue(MIN_AGG));
    }

    if (getSuperstep() < MAX_SUPERSTEPS) {
      long edges = vertex.getNumEdges();
      sendMessageToAllEdges(vertex,
          new DoubleWritable(vertex.getValue().get() / edges));    #5
    } else {
      vertex.voteToHalt();    } }
```

Giraph PageRank implementation

Source file in github:
`https://github.com/apache/giraph/blob/trunk/giraph-examples/src/main/java/org/apache/giraph/examples/SimplePageRankComputation.java`

Assignment on Giraph

2 Problems:

1. Execute an algorithm, that is already implemented in Apache Giraph, to compute some measure in a graph.
2. Implement an iterative Pregel-like algorithm.

Unit 3: Recap

We discussed the following aspects:

- Transitivity, triadic closure
- Clustering
- Strong / weak ties
- Network motifs
- Pregel Programming Model
- LAB SESSION: Apache Giraph