Distributed Graph Algorithms

Alessio Guerrieri

University of Trento, Italy

2016/04/26

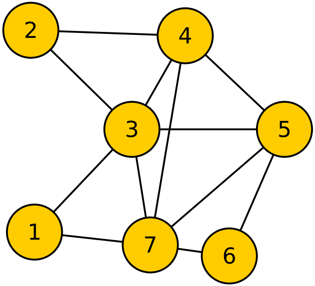
Contents

Who am I?

- My name is Alessio Guerrieri
- Postdoctoral researcher with prof. Montresor
- Specialization on **distributed large-scale graph processing**

Graph $G = (V, E)$

- V set of nodes
- E set of edges (links, connections) between pair of nodes

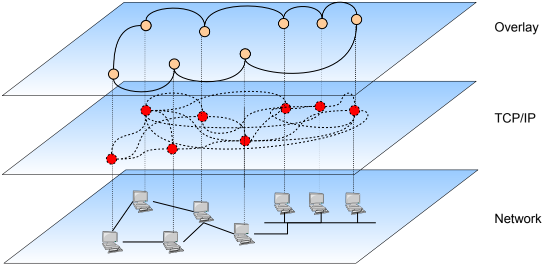


Today's topic

- Vertex-centric model for graph computation
- Possible applications of vertex-centric model
- Vertex-centric graph algorithms

Peer-to-Peer network

- Collection of peer (nodes)
- Have physical connections
- Compute an overlay graph
- Nodes know existence of neighbors
- Can discover other nodes via peer-sampling techniques



Computing in Peer-to-Peer systems

Peers might want to run protocols on the network:

- Finding centrality of peer
- Test properties of network
- Compute distances
- ...

Can we simply reuse distributed algorithms for Computer Networks?

Example: decentralized distance computation

Problem

For each peer x in overlay G, compute its distance from target peer $Target$.

Vertex-centric model

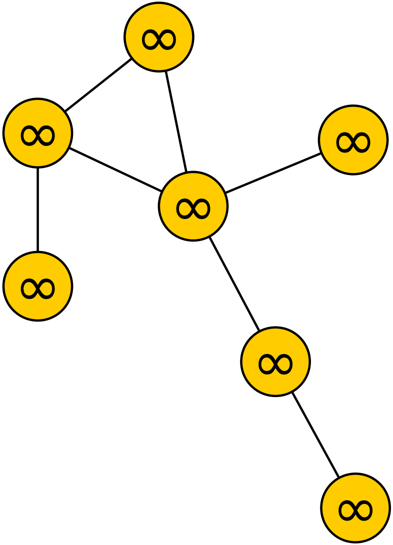
- Initially each peer only knows its direct neighbors in the overlay
- Each peer can send messages to each peer of which it knows the existence
- Amount of memory used by each peer should be sub-linear
- No control on rate of activity of peers and speed of messages (asynchronous execution)

Distributed Breadth-First-Search

D-BFS protocol executed by p:

upon initialization **do**
- **if** $p = Target$ **then**
  - $D_p \leftarrow 0$
  - **send** $\langle D_p + 1 \rangle$ **to** neighbors
- **else**
  - $D_p \leftarrow \infty$

upon receive $\langle d \rangle$ **do**
- **if** $d < D_p$ **then**
  - $D_p \leftarrow d$
  - **send** $\langle D_p + 1 \rangle$ **to** neighbors

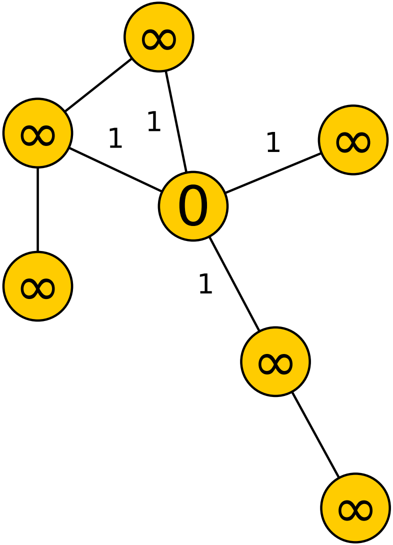


Distributed Breadth-First-Search

D-BFS protocol executed by p:

upon initialization **do**
- **if** $p = Target$ **then**
  - $D_p \leftarrow 0$
  - **send** $\langle D_p + 1 \rangle$ **to** neighbors
- **else**
  - $D_p \leftarrow \infty$

upon receive $\langle d \rangle$ **do**
- **if** $d < D_p$ **then**
  - $D_p \leftarrow d$
  - **send** $\langle D_p + 1 \rangle$ **to** neighbors

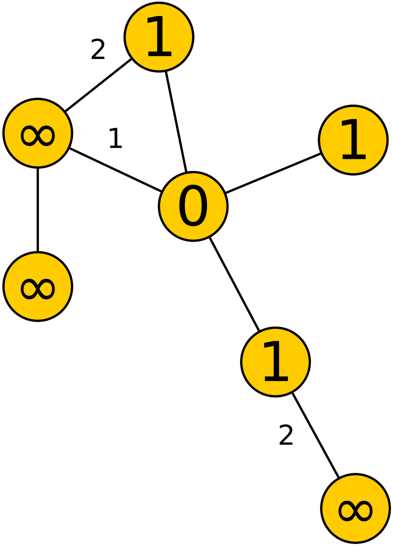


Distributed Breadth-First-Search

D-BFS protocol executed by p:

upon initialization **do**
- **if** $p = Target$ **then**
  - $D_p \leftarrow 0$
  - **send** $\langle D_p + 1 \rangle$ **to** neighbors
- **else**
  - $D_p \leftarrow \infty$

upon receive $\langle d \rangle$ **do**
- **if** $d < D_p$ **then**
  - $D_p \leftarrow d$
  - **send** $\langle D_p + 1 \rangle$ **to** neighbors

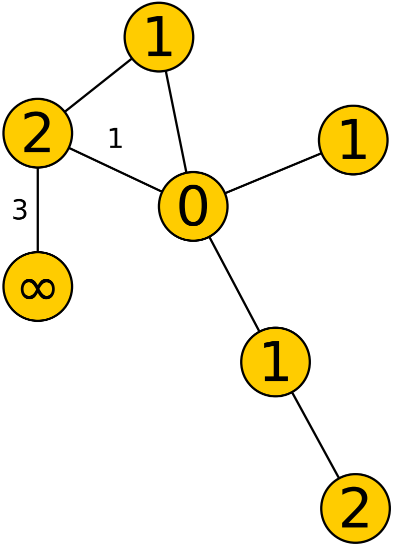


Distributed Breadth-First-Search

D-BFS protocol executed by p:

upon initialization **do**
  if $p = Target$ **then**
    $D_p \leftarrow 0$
    send $\langle D_p + 1 \rangle$ **to** neighbors
  else
    $D_p \leftarrow \infty$

upon receive $\langle d \rangle$ **do**
  if $d < D_p$ **then**
    $D_p \leftarrow d$
    send $\langle D_p + 1 \rangle$ **to** neighbors

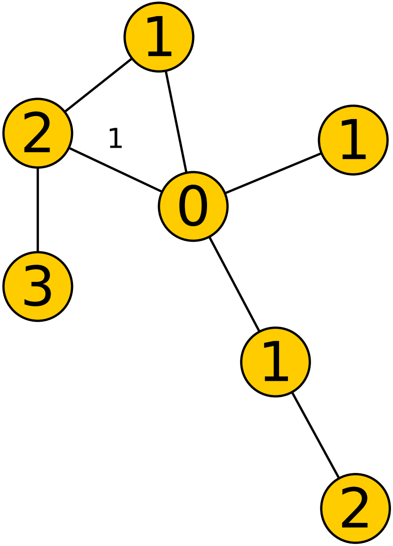


Distributed Breadth-First-Search

D-BFS protocol executed by p:

upon initialization **do**
- **if** $p = Target$ **then**
  - $D_p \leftarrow 0$
  - **send** $\langle D_p + 1 \rangle$ **to** neighbors
- **else**
  - $D_p \leftarrow \infty$

upon receive $\langle d \rangle$ **do**
- **if** $d < D_p$ **then**
  - $D_p \leftarrow d$
  - **send** $\langle D_p + 1 \rangle$ **to** neighbors

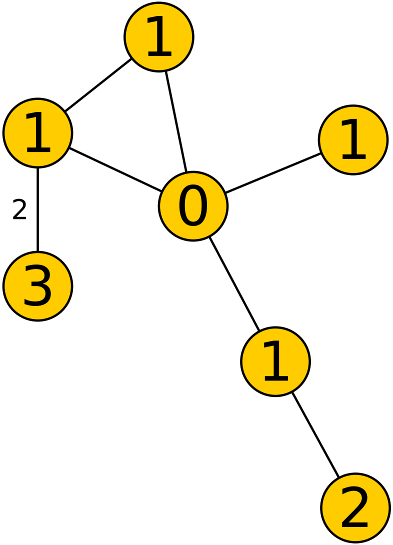


Distributed Breadth-First-Search

D-BFS protocol executed by p:

upon initialization **do**
  if $p = Target$ **then**
    $D_p \leftarrow 0$
    send $\langle D_p + 1 \rangle$ **to** neighbors
  else
    $D_p \leftarrow \infty$

upon receive $\langle d \rangle$ **do**
  if $d < D_p$ **then**
    $D_p \leftarrow d$
    send $\langle D_p + 1 \rangle$ **to** neighbors

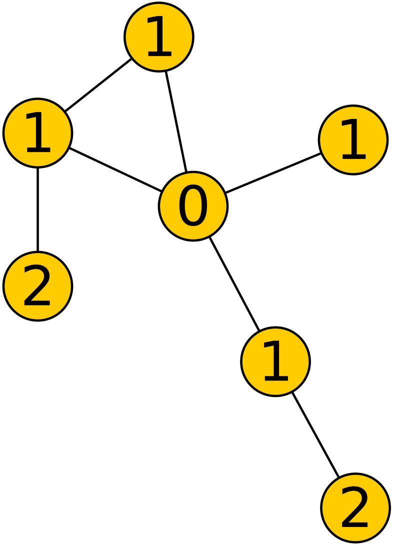


Distributed Breadth-First-Search

D-BFS protocol executed by p:

upon initialization **do**
  if $p = Target$ **then**
    $D_p \leftarrow 0$
    send $\langle D_p + 1 \rangle$ **to** neighbors
  else
    $D_p \leftarrow \infty$

upon receive $\langle d \rangle$ **do**
  if $d < D_p$ **then**
    $D_p \leftarrow d$
    send $\langle D_p + 1 \rangle$ **to** neighbors



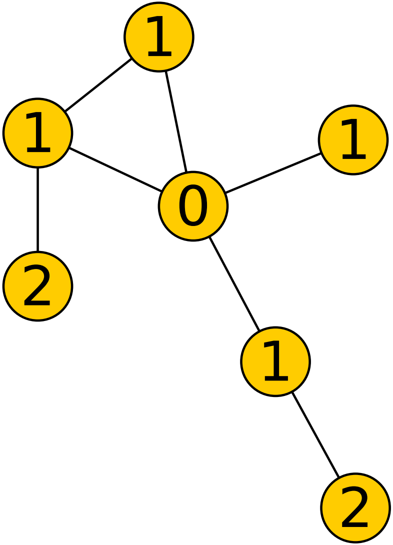
Distributed Breadth-First-Search

D-BFS protocol executed by p:

upon initialization **do**
  if $p = Target$ **then**
    $D_p \leftarrow 0$
    send $\langle D_p + 1 \rangle$ **to** neighbors
  else
    $D_p \leftarrow \infty$

upon receive $\langle d \rangle$ **do**
  if $d < D_p$ **then**
    $D_p \leftarrow d$
    send $\langle D_p + 1 \rangle$ **to** neighbors

repeat every Δ time units
  send $\langle D_p + 1 \rangle$ **to** neighbors



Notes

Works if:

- Nodes execute in any order
- Messages arrive in any order
- Nodes or edges are added
- Messages are lost

Fails if:

- Nodes fail
- Edges are removed

New Graph Applications

Many settings in which there is need to analyze graph:

- Web data
- Computer networks
- Biological networks
- Social networks
- ...

Size can be extremely large

Approaches

Centralized/Sequential

- Collect data in one place
- Load it in memory
- Analyze it with traditional techniques

Decentralized

- One machine for each node
- Construct overlay network
- Run vertex-centric protocol

Both unfeasible! These are better:

Centralized/Parallel

Parallel processes and threads in shared memory

Distributed system

Multiple (not N) machines, each hosting part of the graph

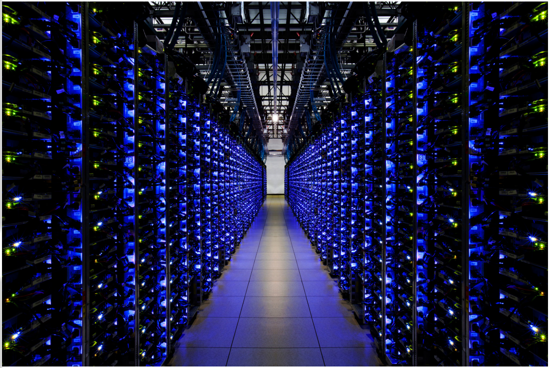
Why distributed?

Disadvantages

- Slower than parallel systems
- Younger field of research
- Needs distribution of data

Advantages

- Fault tolerant
- Cheaper
- Extendible



What do we want?

Nice interface for distributed graph algorithms and fast framework to run those algorithms

Graph analyst are not distributed systems experts!

Issues with:

- Data replication
- Fault tolerance
- Message deliverance
- ...

should be dealt by the system!

Vertex centric

Why not reuse vertex-centric interface?

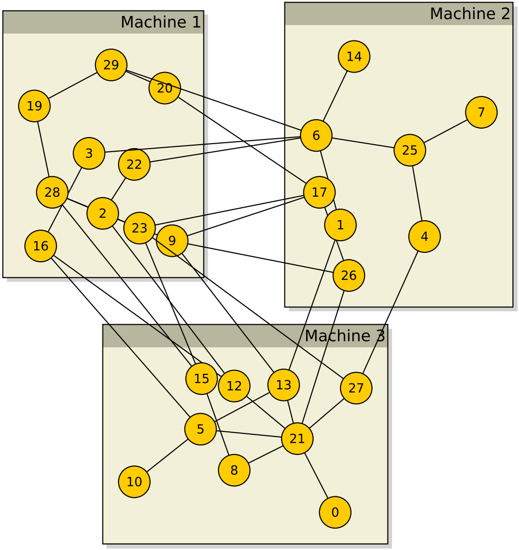
User

- Defines data associated to vertex
- Defines type of messages
- Defines code executed by single vertex when it receives messages

System

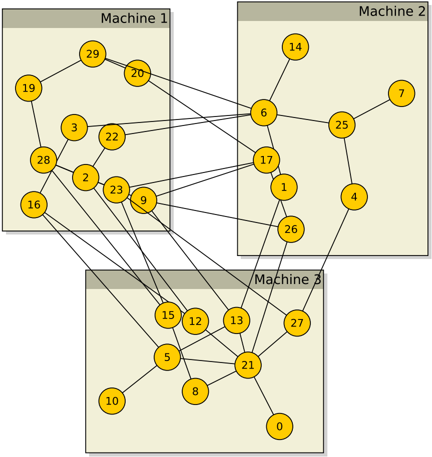
- Divides input graph between the available machines
- Each machine *simulates* each vertex independently
- Takes care of fault tolerance

Vertex centric distributed system



Pregel

- Developed by Google
- First large-scale graph processing system with vertex-centric interface
- Created for PageRank
- Code automatically deployed on thousands of machines



Programming API

Pregel-like frameworks

Many:

- Giraph: on top of Hadoop
- GraphX: on top of Spark
- Gelly: on top of Flink
- Graphlab: standalone

All frameworks allow user to run vertex-centric programs on distributed systems

Example (Giraph)

```java
public void compute(Iterable<IntWritable> messages) {
    int minDist = Integer.MAX_VALUE;
    for (IntWritable message : messages) {
        minDist = Math.min(minDist, message.get());
    }
    if (minDist < getValue().get()) {
        setValue(new IntWritable(minDist));
        IntWritable distance = new IntWritable(minDist + 1)
        for (Edge<...> edge : getEdges()) {
            sendMessage(edge.getTargetVertexId(),distance);
        }
    }
    voteToHalt();
}
```

Common characteristics of Vertex-centric models

Independence from problem size

Code stays the same regardless of

- size of graph
- number of machines used

Embedded fault-tolerance

- Periodic benchmarking
- Machines can ”steal nodes” from slower machines

Distributed File System

- Usually HDFS
- Allow all machines to read and write in common space

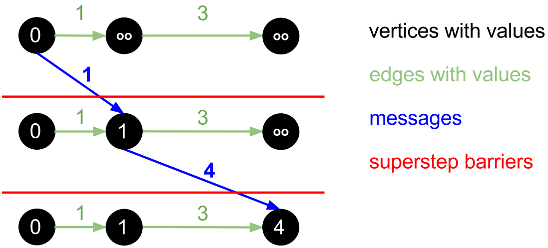
Synchronicity of Vertex-centric models

Synchronous

- Execution in rounds
- Each vertex receives in round i all messages sent to it in round $i-1$
- Better guarantees $\rightarrow$ easiest model

Asynchronous

- No rounds
- Messages processed without guarantees
- More difficult, more efficient



Graph partitioning

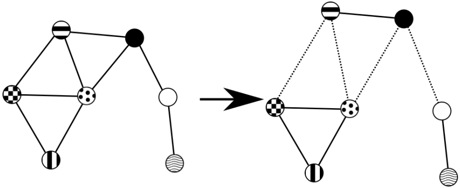
First step of analysis
Which machine hosts which subset of nodes/edges?

Balance

Size of partitions should be balanced

Quality

- Messages between nodes hosted in same machine are cheap
- Messages between machines are costly



Heuristic solutions

Graph partitioning is NP-Hard

Hash-based heuristics

Node n ends in machine $H(n)\%K$

Iterative algorithms

Start from random and incrementally improve

Heuristics can be devised for specific data-sets

Graph Algorithms

What algorithms are available

This area is not completely new: there are distributed algorithms for computer networks
But:

- Computer networks are small
- Many interesting graph problems are not interesting on computer networks

Some ideas can be taken from research in PRAM (Parallel Random Access Machine)

Problems

We'll look at:

- Triangle counting
- Connected components
- Strongly connected components
- Clustering

We won't look at:

- Pagerank
- Single-Value decomposition

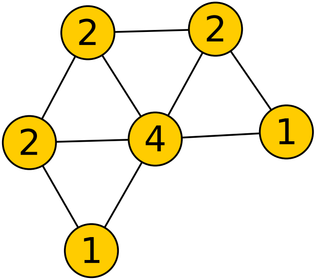
Triangle Counting

Definition

In an undirected graph Compute, for each node n, how many pairs of nodes (v, w) exists such that $(n, v), (v, w), (w, n)$ are edges.

Application on social networks

Seems easier than it is...



Triangle counting protocol

Send neighbor-list to neighbors and see which neighbors we have in common

---

Executed by node p

---

upon initialization **do**

  $T_p \leftarrow 0$

  send neighbor-list **to** neighbors

upon receive $\langle M \rangle$ **do**

  $Common \leftarrow$ neighbor-list $\bigcap M$

  $T_p \leftarrow T_p + |Common|$

Issues

Multi-counting

- Each triangle is counted 6 times!
- Should make sure that every triangle is counted only once!

Hubs

- Hubs are nodes with disproportionally large degree
- Exists in almost all real graphs
- Will send large neighbor-lists to large number of neighbors
- Will receive large quantity of messages

Triangle counting protocol (2)

Send list of neighbors with smaller id

Executed by node p

---

upon initialization **do**

  $T_p \leftarrow 0$

  $M \leftarrow \{n : n \in \text{neighbor-list}, id_n < id_p\}$

  send M **to** neighbors

upon receive $\langle M \rangle$ **do**

  $Common \leftarrow \text{neighbor-list} \bigcap M$

  $T_p \leftarrow T_p + |Common|$

Cut messages by half

Triangle counting protocol (3)

Executed by node p

upon initialization **do**
- $T_p \leftarrow 0$
- **foreach** $n \in$ neighbor-list **do**
  - **if** $id_n < id_p$ **then**
    - $M \leftarrow \{q : q \in \text{neighbor-list}, id_q < id_n\}$
    - **send** $LIST(M)$ **to** n

upon receive $\langle LIST(M) \rangle$ from n **do**
- $Common \leftarrow \text{neighbor-list} \bigcap M$
- $T_p \leftarrow T_p + |Common|$ **send** $TR(|Common|)$ **to** n
- **send** $TR(1)$ **to** $v \in Common$

upon receive $\langle TR(t) \rangle$ **do**
- $T_p \leftarrow T_p + t$

Connected components (problem)

Problem definition

Two nodes v, w of an undirected graph are in the same weakly connected component (WCC) if exists a path from v to w.
For each node n compute value C_n such that:

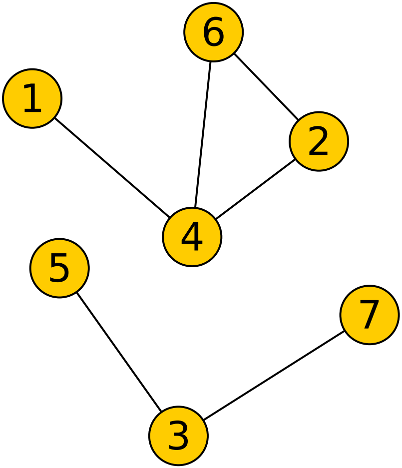
$$\forall v, w \in V, C_v = C_w \iff WCC_v = WCC_w$$

Connected components (solution)

Executed by node p:

upon initialization **do**
- $C_p \leftarrow p.id$
- **send** $\langle C_p \rangle$ **to** neighbors

upon receive $\langle c \rangle$ **do**
- **if** $c < C_p$ **then**
  - $C_p \leftarrow c$
  - **send** $\langle c \rangle$ **to** neighbors

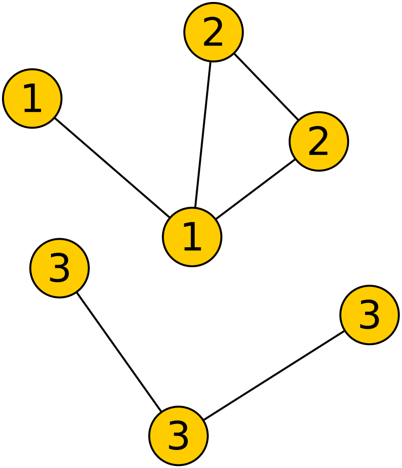


Connected components (solution)

Executed by node p:

upon initialization **do**
  $C_p \leftarrow p.id$
  send $\langle C_p \rangle$ **to** neighbors

upon receive $\langle c \rangle$ **do**
  if $c < C_p$ **then**
    $C_p \leftarrow c$
    send $\langle c \rangle$ **to** neighbors

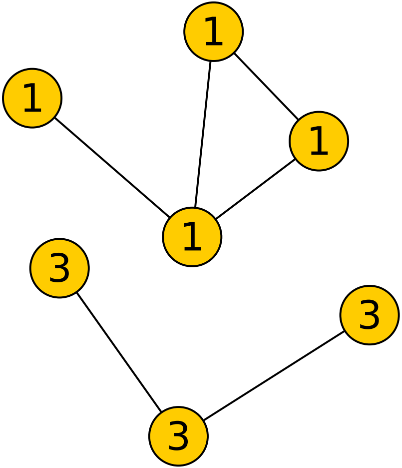


Connected components (solution)

Executed by node p:

upon initialization **do**
  $C_p \leftarrow p.id$
  send $\langle C_p \rangle$ **to** neighbors

upon receive $\langle c \rangle$ **do**
  if $c < C_p$ **then**
    $C_p \leftarrow c$
    send $\langle c \rangle$ **to** neighbors



Strongly connected components

Problem definition

Two nodes v, w of an *directed graph* are in the same strongly connected component (SCC) if exist paths from v to w and from w to v (both directions).

Centralized solution

- Single depth-first search visit using Tarjan's algorithm
- Two depth-first search visits using Kosaraju's algorithm
- Distributed solution much more difficult!

Strongly Connected components (solution)

SCC: Executed by node p:

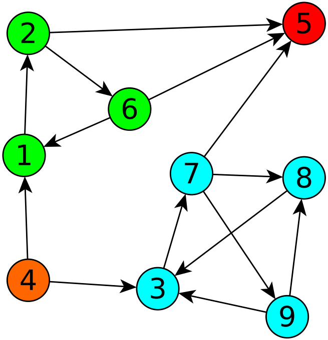
upon initialization **do**
- $F_p \leftarrow p.id$ // Lowest id of nodes that reaches p
- $B_p \leftarrow p.id$ // Lowest id of nodes reachable from p
- **send** $\langle FOR(F_p) \rangle$ **to** out-neighbors
- **send** $\langle BACK(B_p) \rangle$ **to** in-neighbors

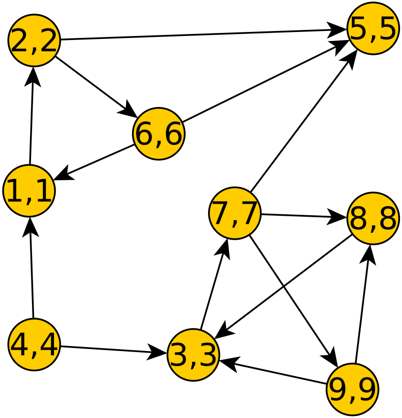
upon receive $\langle FOR(c) \rangle$ **do**
- **if** $c < F_p$ **then**
  - $F_p \leftarrow c$ **send** $\langle FOR(c) \rangle$ **to** out-neighbors

upon receive $\langle BACK(c) \rangle$ **do**
- **if** $c < B_p$ **then**
  - $B_p \leftarrow c$ **send** $\langle BACK(c) \rangle$ **to** in-neighbors

Properties

B_p = minimum id of nodes reachable from p
F_p = minimum id of nodes that can reach p

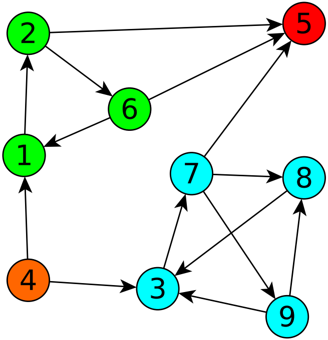


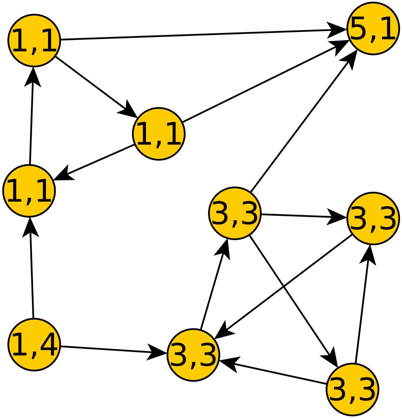


Properties

B_p = minimum id of nodes reachable from p

F_p = minimum id of nodes that can reach p





Properties

Lemma1

$(F_p = B_p = i) \rightarrow$ p and i are in the same SCC

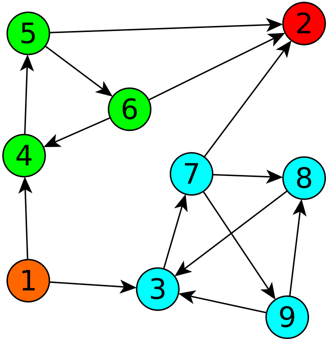
Lemma2

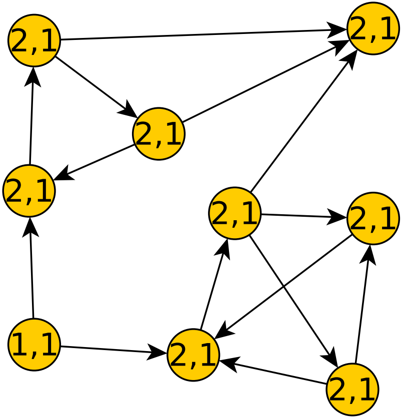
$(F_p = F_q$ and $B_P = B_1 \rightarrow$ p and q are in the same SCC

Are the two lemmas correct?

Properties

B_p = minimum id of nodes reachable from p
F_p = minimum id of nodes that can reach p





Complete algorithm

Complete algorithm

While graph not empty:

- Run SCC algorithm
- remove each node p such that $F_p = B_p$

Can improve number of SCC found through heuristics

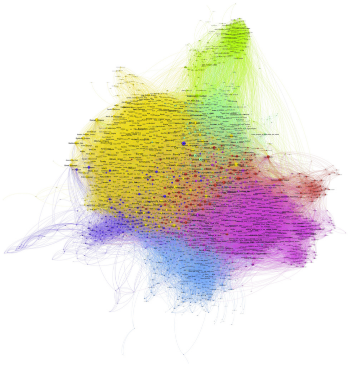
Quicker convergence for largest SCC

Graph clustering

Given a graph, divide its vertices in *clusters* such that:

- Most edges connect nodes in same cluster
- Few edges go across cluster
- Clusters are significant

Precise definition depends from application scenario
All versions are NP-Complete



Label-propagation algorithm

Executed by node p:

upon initialization **do**

  $C_p \leftarrow p.id$ // starting label equal to id

  $N_p \leftarrow \{\}$ // dictionary containing labels of neighbors

  send C_p **to** neighbors

upon receive c **from** q **do**

  $N_p[q] = c$ // update dictionary with new label of neighbor

repeat every Δ time units

  $c' \leftarrow$ most common label in neighbors

  if $C_p \neq c'$ **then**

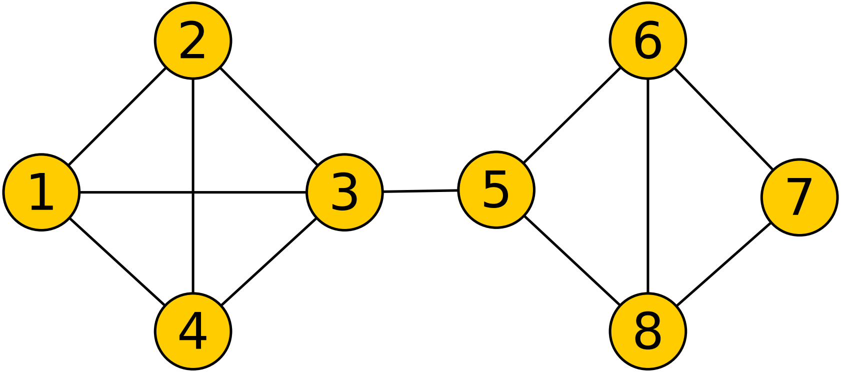
    send c' **to** neighbors

    $C_p \leftarrow c'$

Note:ties resolved randomly

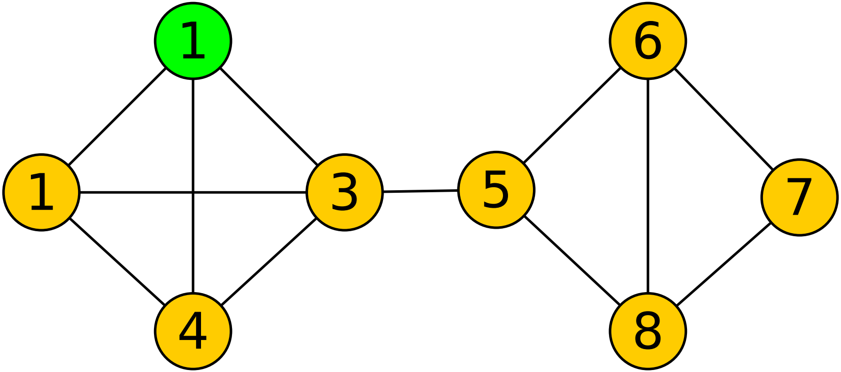
Sample run

Initialization:



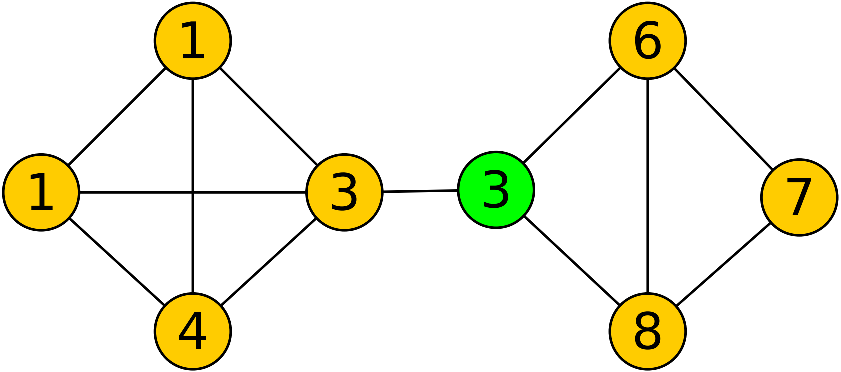
Sample run

Active node chooses label 1 (randomly)



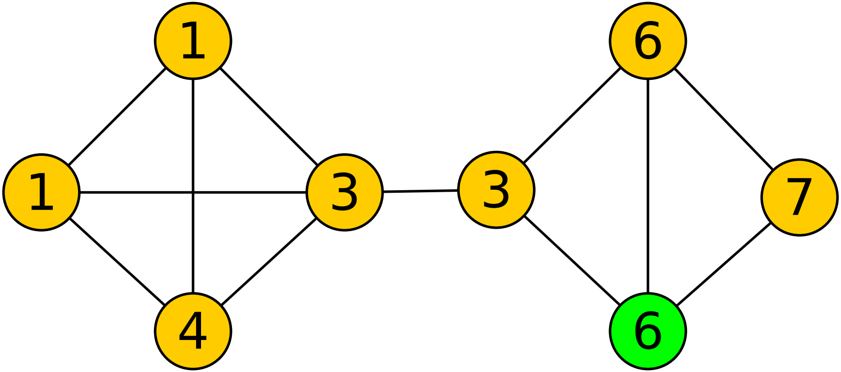
Sample run

Active node chooses label 3(randomly)



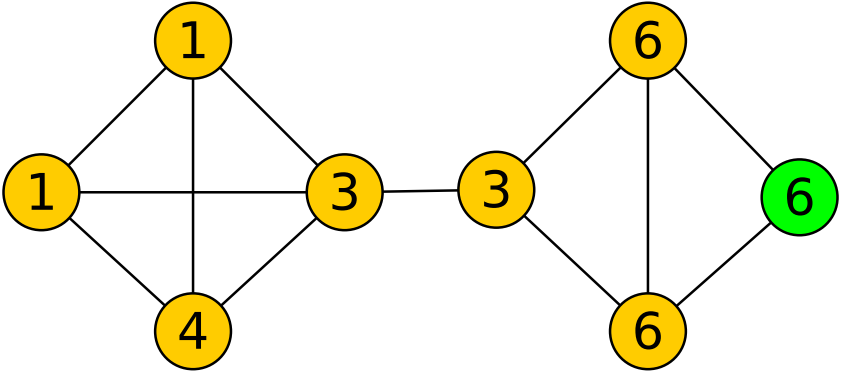
Sample run

Active node chooses label 6(randomly)



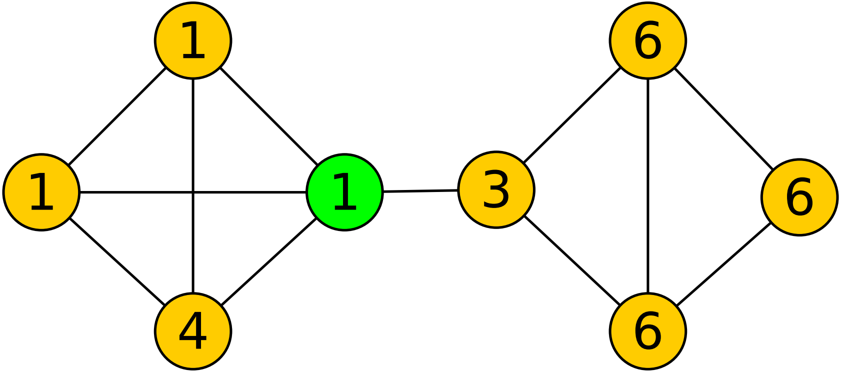
Sample run

Active node chooses label 6



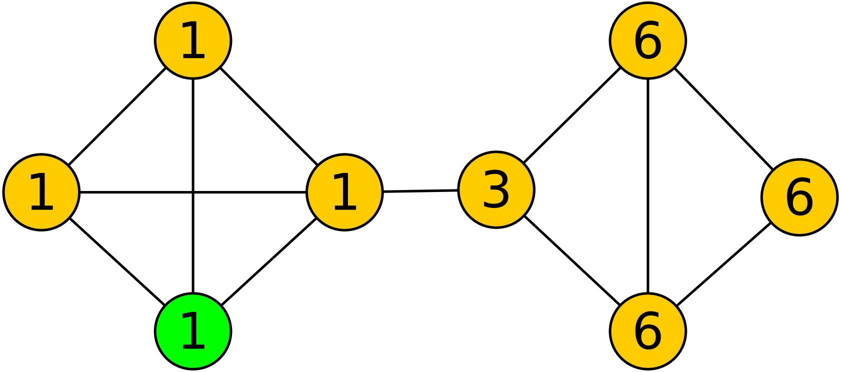
Sample run

Active node chooses label 1



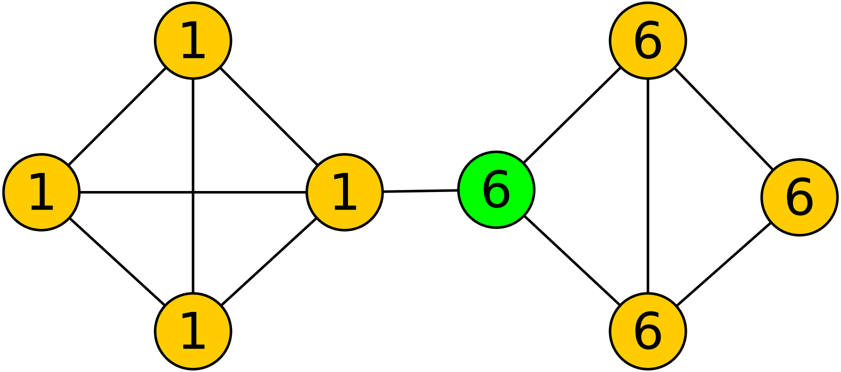
Sample run

Active node chooses label 1



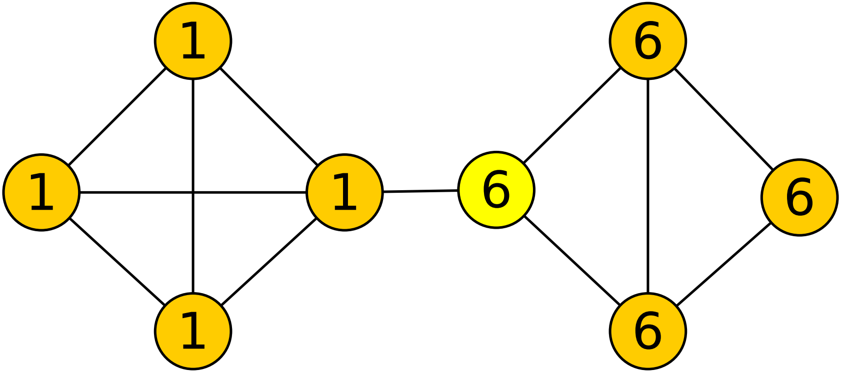
Sample run

Active node chooses label 6



Sample run

Converged state: no node change its mind



Disadvantages

Non-deterministic

Non-null probability of collapsing to single cluster

Low quality

In practice, it's not that good

Better options

- Slow, iterative algorithms
- Fast, heuristics
- My algorithm!

Conclusions

- Vertex-centric algorithms can be used in:
  - P2P systems
  - Computer networks
  - Large-scale graph analysis
- Field still in flux
  - New frameworks
  - New programming models
  - New algorithms
- Many applications
  - Everything is a graph
  - Could be interesting for a project

Reading Material

- *Pregel: A System for Large-Scale Graph Processing* by Malewicz et al.
- *Pregel Algorithms for Graph Connectivity Problems with Performance Guarantees* by Yan et al.
- *Giraph* `http://giraph.apache.org/`
- *Graphx* `http://spark.apache.org/graphx/`