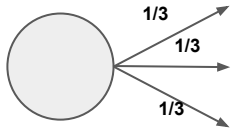
Computing PageRank & HITS

Fabrizio Silvestri

SAPIENZA
UNIVERSITÀ DI ROMA

Recap: The Google Problem

- Imagine a user doing a random walk on web pages:
  - Start at a random page
  - At each step, go out of the current page along one of the links on that page, equiprobably



- In the long run" each page has a long-term visit rate – use this as the page's score

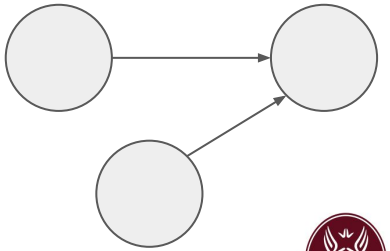
- Variant: rather than equiprobable, use text and link information to have probability of following a link: **intelligent surfer**

Recap: Ergodic Markov Chains

- For any ergodic Markov chain, there is a unique long-term visit rate for each state.
  - Steady-state probability distribution.
  - Over a long time-period, we visit each state in proportion to this rate.
  - It doesn't matter where we start.

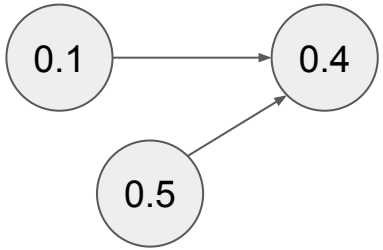
- Ergodic: no periodic patterns
  - Teleportation ensures ergodicity

Not Ergodic



Recap: Probability Vectors

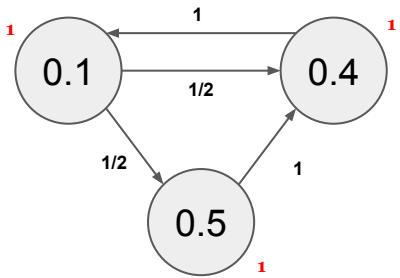
- A probability (row) vector x = (x1, … xn) tells us where the walk is at any point.
- E.g., (000…1…000) means we're in state i.



- More generally, the vector x = (x_1, … x_n) means the walk is in state i with probability x_i.
  - $x_1+x_2+ \ldots + x_n = 1$

Recap: Dynamics of Markov Chains

- u_0=(0.1, 0.4, 0.5)
- $A = \begin{pmatrix} 0, & 0.5, & 0.5 \\ 1, & 0, & 0 \\ 0, & 1, & 0 \end{pmatrix}$



- $u_1 = u_0A = (0.4, 0.55, 0.05)$
- $u_2 = u_1A = (u_0A)A = u_0A^2 = (0.55, 0.25, 0.2)$
- $u_3 = u_0A^3 = (0.25, 0.475, 0.275)$

Perron-Frobenius Theorem

Theorem (5)

Let $A \geq 0$ be an irreducible $n \times n$ matrix. Then,

1. *A has a positive real eigenvalue equal to its spectral radius $\rho(A)$.*
2. *To $\rho(A)$ there corresponds an eigenvector $x > 0$.*
3. *$\rho(A)$ increases when any entry of A increases.*
4. *$\rho(A)$ is a simple eigenvalue of A.*
5. *There is not other nonnegative eigenvector of A different from x.*

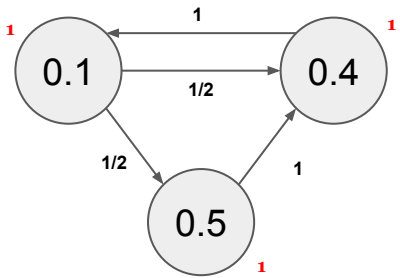
The largest Eigenvalue of a Stochastic Matrix is 1

1. If A is a stochastic matrix, then A**1** = **1**, since each row of A sums to 1. Therefore, 1 is an eigenvalue of A.
2. Suppose there exists $\lambda > 1$ and nonzero **x** such that **x**A = λ**x**, and let x_i be the largest element of **x**.
   a. Since any scalar multiple of **x** will also satisfy this equation we can assume, without loss of generality, that $x_i > 0$.
   b. Since the rows of A are nonnegative and sum to 1, each entry in λ**x** is a convex combination of the elements of **x**. Thus no entry in λ**x** can be larger than x_i.
   But since $\lambda > 1$, $\lambda x_i > x_i$, which is a contradiction.

Therefore, the largest eigenvalue of A is 1.

PageRank as an EigenProblem

- $u_m = u_{m-1}A$
  - $u = uA \Rightarrow 1u = uA$
- The stable distribution u is an (left) eigenvector of A
- In fact, A being stochastic, and positive definite has all real eigenvalues $1 = \boldsymbol{\lambda}_1 > \boldsymbol{\lambda}_2 > \ldots > \boldsymbol{\lambda}_k$
  - 1 is the largest eigenvalues
- So, the stable distribution is the principal eigenvector of A

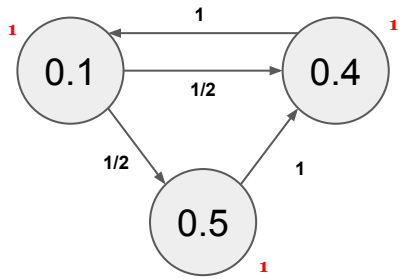


Observation: The “Shape” of A^k

$$A = \begin{pmatrix} 0, & 0.5, & 0.5 \\ 1, & 0, & 1 \\ 0, & 1, & 0 \end{pmatrix}$$

$$A^5 = \begin{pmatrix} 0.5, & 0.375, & 0.125 \\ 0.25, & 0.5, & 0.25 \\ 0.5, & 0.25, & 0.25 \end{pmatrix}$$

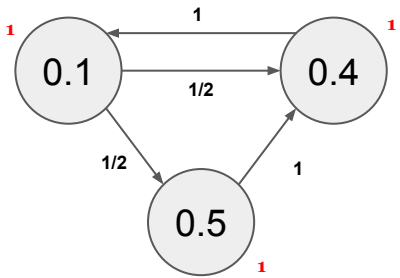
$$A^{60} = \begin{pmatrix} 0.4, & 0.4, & 0.2 \\ 0.4, & 0.4, & 0.2 \\ 0.4, & 0.4, & 0.2 \end{pmatrix}$$



How Do You Find It? **The Power Method**

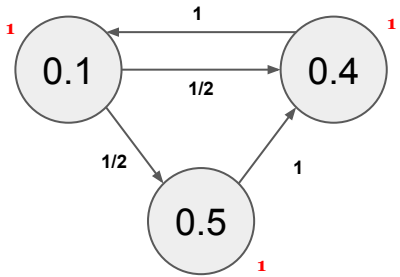
- First suppose A is diagonalizable
  - $A = P\Lambda P^{-1}$
  - $\Lambda = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$
- $\mathbf{x_i}$ is the eigenvector corresponding to λ_i
- Let's consider $\mathbf{u_0} = \alpha_1 \mathbf{x_1} + \alpha_2 \mathbf{x_2} + \ldots + \alpha_n \mathbf{x_n}$
- $u_0 A^k = \Sigma \alpha_j \mathbf{x_j} A^k = \Sigma \alpha_j \lambda_j^k \mathbf{x_j} = \lambda_1^k (\alpha_1 \mathbf{x_1} + \Sigma \alpha_{j>1} (\lambda_j/\lambda_1)^k \mathbf{x_j})$

- Therefore $\lim_{k \to \infty} (\mathbf{u_0} A^k / \lambda_1^k) = \alpha_1 \mathbf{x_1}$



How Do You Find It? **The Power Method**

1. Choose a starting vector $\mathbf{u}_0 \in \mathbf{R}^n$ with $||\mathbf{u}_0||_1 = 1$
2. k = 0;
3. repeat
   - k := k + 1;
   - $\mathbf{y}$:= $\mathbf{x}_{k-1}A$;
   - $\mathbf{x}_k$:= $\mathbf{y} / ||\mathbf{y}||_1$
4. until a convergence criterion is satisfied



The rate of convergence is directly proportional to the ratio λ_2 / λ_1

How Do You Find It? **The Power Method**

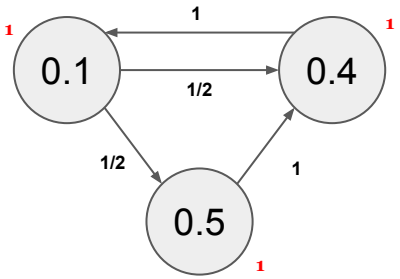
```
[ ]  1 import numpy as np
     2
     3 A = np.array(
     4     [
     5      [0, 0.5, 0.5],
     6      [1, 0, 0],
     7      [0, 1, 0],
     8     ]
     9 )
```

```
[ ]  1 u = np.array([1, 0, 0])
     2 k = 0
     3 max_iter = 100
     4 converged = False
     5 epsilon = 1e-10
     6 while not converged:
     7   k += 1
     8   old_u = np.copy(u)
     9   y = np.dot(u, A)
    10   u = y / np.linalg.norm(y, 1)
    11   if np.linalg.norm(old_u - u) < epsilon or k > max_iter:
    12     converged = True
    13
    14 print("The method converged to {} in {} iterations.".format(u, k))
```

```
The method converged to [0.4 0.4 0.2] in 67 iterations.
```

```
  1 print(np.dot(u, A))
```

```
[0.4 0.4 0.2]
```



Accelerating PageRank Computation

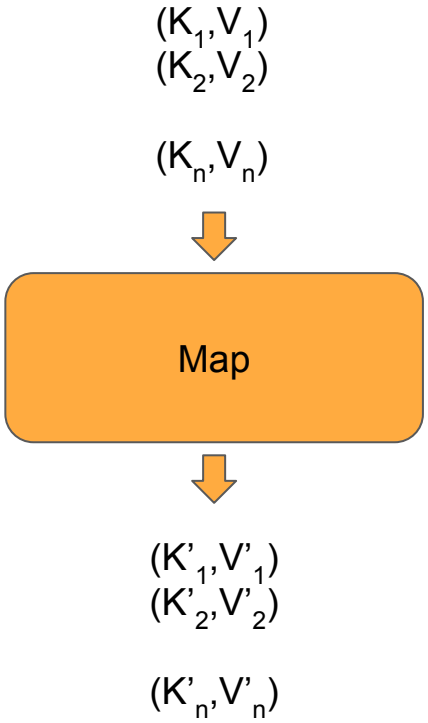
- Web Graph Compression to fit in internal memory
- Efficient external-memory implementation
- Distributed PageRank Computation
- Mathematical approaches
- Combination of the above strategies

MapReduce

- What is it?
  - Programming model originally proposed by Google
  - A combination of the Map and Reduce models with an associated implementation
  - Used for processing and generating large data sets
- MapReduce is highly scalable and can be used across many computers.
- Many small machines can be used to process jobs that normally could not be processed by a large machine.

Map Abstraction

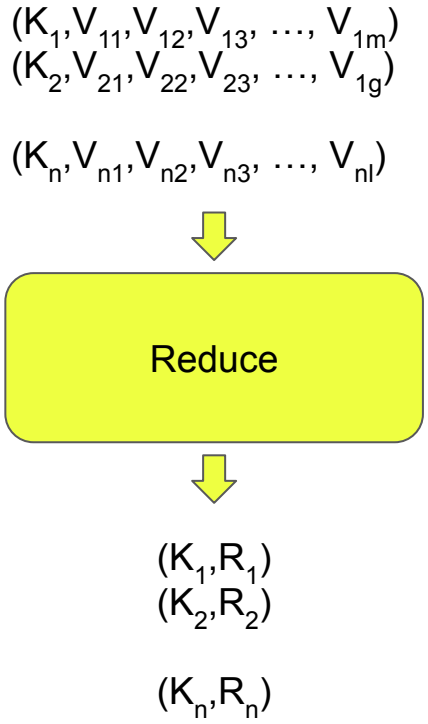
- Inputs a key/value pair
  - Key is a reference to the input value
  - Value is the data set on which to operate
- Evaluation
  - Function defined by user
  - Applies to every value in value input
    - Might need to parse input
- Produces a new list of key/value pairs
  - Can be different type from input pair



Reduce Abstraction

- Starts with intermediate Key / Value pairs
- Ends with finalized Key / Result (Value) pairs

- Starting pairs are sorted by key
- Iterator supplies the values for a given key to the Reduce function.
- Typically a function that:
  - Starts with a large number of key/value pairs
    - One key/value for each word in all files being grepped (including multiple entries for the same word)
  - Ends with very few key/value pairs
    - One key/value for each unique word across all the files with the number of instances summed into this entry
- Broken up so a given worker works with input of the same key.



Reduce Example

```
def reduce(key, listOfValues):
    result = 0
    for x in listOfValues:
        result += x
    return (key, result)
```

PageRank: Map

```
map(key: [url, pagerank], value: outlink_list)
   for each outlink in outlink_list
      emit(key: outlink, value: pagerank/len(outlink_list))
   emit(key: url, value: outlink_list)
```

PageRank: Reduce

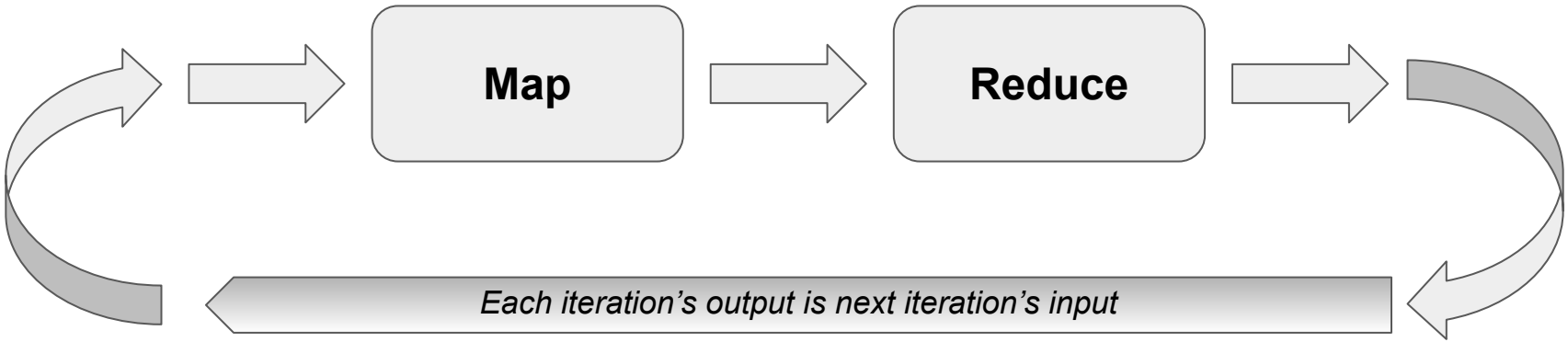
```
reduce(key: url, value: list_pr_or_urls)
     outlink_list = []
     pagerank = 0

     for each pr_or_urls in list_pr_or_urls
          if is_list(pr_or_urls)
               outlink_list = pr_or_urls
          else
               pagerank += pr_or_urls

     pagerank = 1 - DAMPING_FACTOR + ( DAMPING_FACTOR * pagerank )

     emit(key: [url, pagerank], value: outlink_list)
```

PageRank: MapReduce



MapReduce’s PageRank Issue

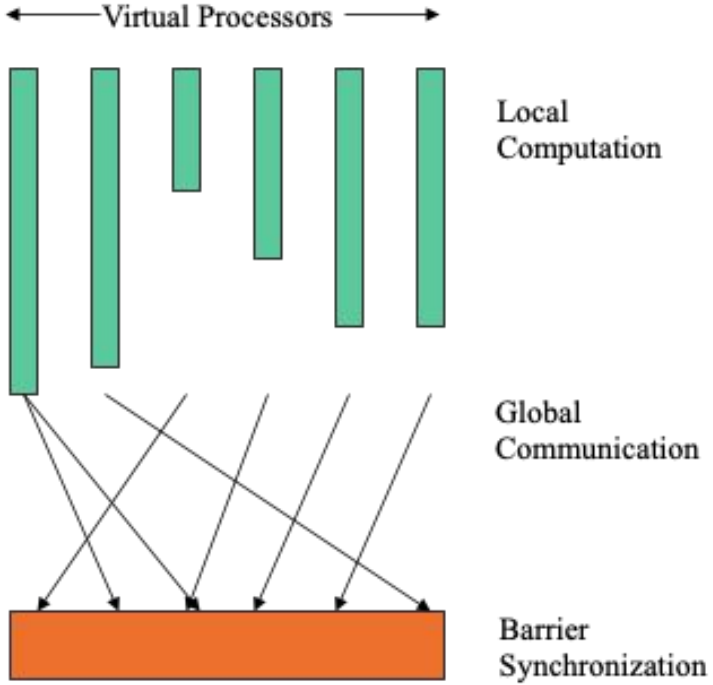
- The problem of stragglers in real world’s graphs
- Number of iterations unknown in advance, how do you decide how to spawn Mappers and Reducers?

Bulk Synchronous Parallelism

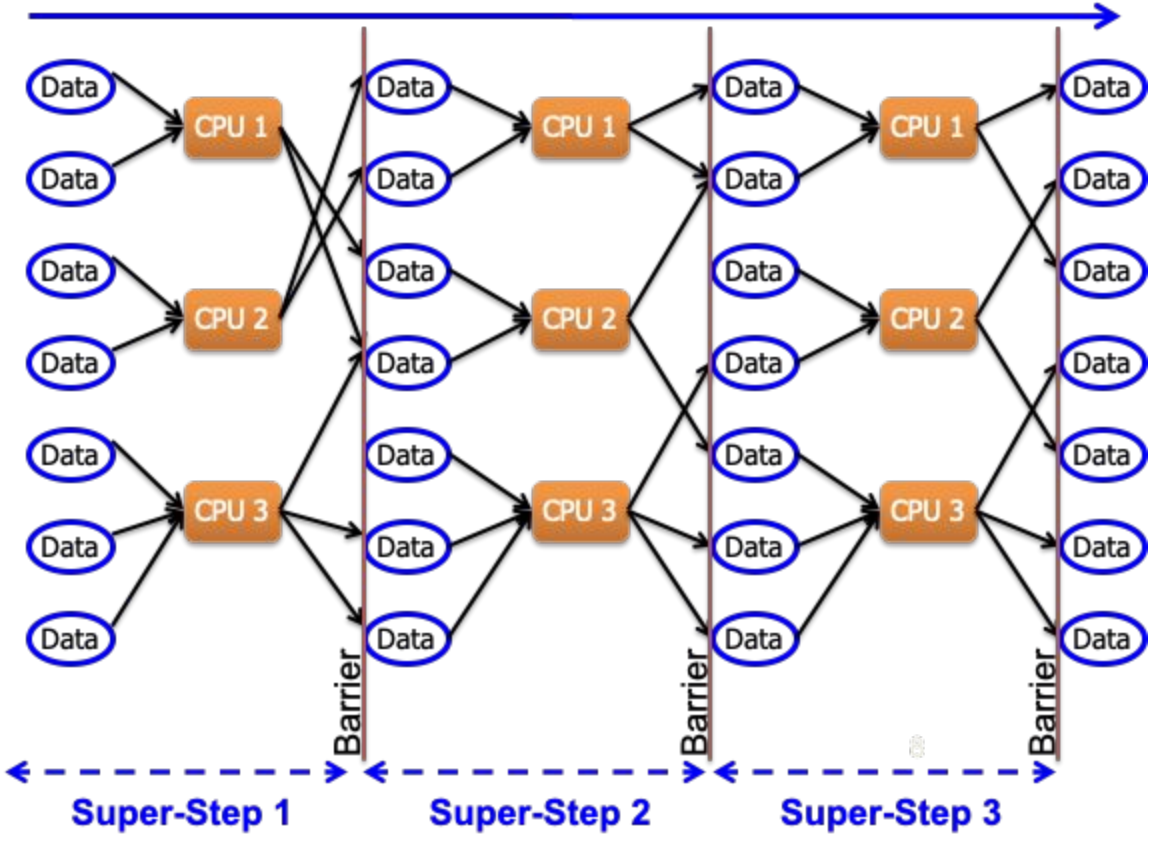
- Computational model of parallel computation consisting of:
  - A set of processor-memory pairs.
  - A communications network that delivers messages in a point-to-point manner.
  - A mechanism for the efficient barrier synchronization for all or a subset of the processes.
  - There are no special combining, replicating, or broadcasting facilities.

What does the BSP Programming Style Look Like?

- Vertical Structure
  - Sequential composition of “supersteps”.
  - Local computation
  - Process Communication
  - Barrier Synchronization
- Horizontal Structure
  - Concurrency among a fixed number of virtual processors.
  - Processes do not have a particular order.
  - Locality plays no role in the placement of processes on processors.
  - p = number of processors.



What does the BSP Programming Style Look Like?



BSP Programming Style

- Properties:
  - Simple to write programs.
  - Independent of target architecture.
  - Performance of the model is predictable.
- Considers computation and communication at the level of the entire program and executing computer instead of considering individual processes and individual communications.
- Renounces locality as a performance optimization.
  - Good and bad
  - BSP may not be the best choice for which locality is critical i.e. low-level image processing.

Google’s Pregel

- Pregel is a large-scale graph-parallel distributed analytics engine inspired by BSP

- Some Characteristics:
  - In-Memory (opposite to MapReduce)
  - High scalability
  - Automatic fault-tolerance
  - Flexibility in expressing graph algorithms
  - Message-Passing programming model
  - Tree-style, master-slave architecture
  - Synchronous

Entities and Super-Steps

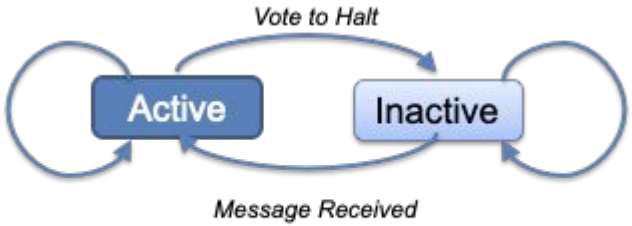
- The computation is described in terms of vertices, edges and a sequence of super-steps

- You give Pregel a directed graph consisting of vertices and edges
  - Each vertex is associated with a modifiable user-defined value
  - Each edge is associated with a source vertex, value and a destination vertex

- During a super-step:
  - A user-defined function F is executed at each vertex V
  - F can read messages sent to V in superset S – 1 and send messages to other vertices that will be received at superset S + 1
  - F can modify the state of V and its outgoing edges
  - F can alter the topology of the graph

Algorithm Termination

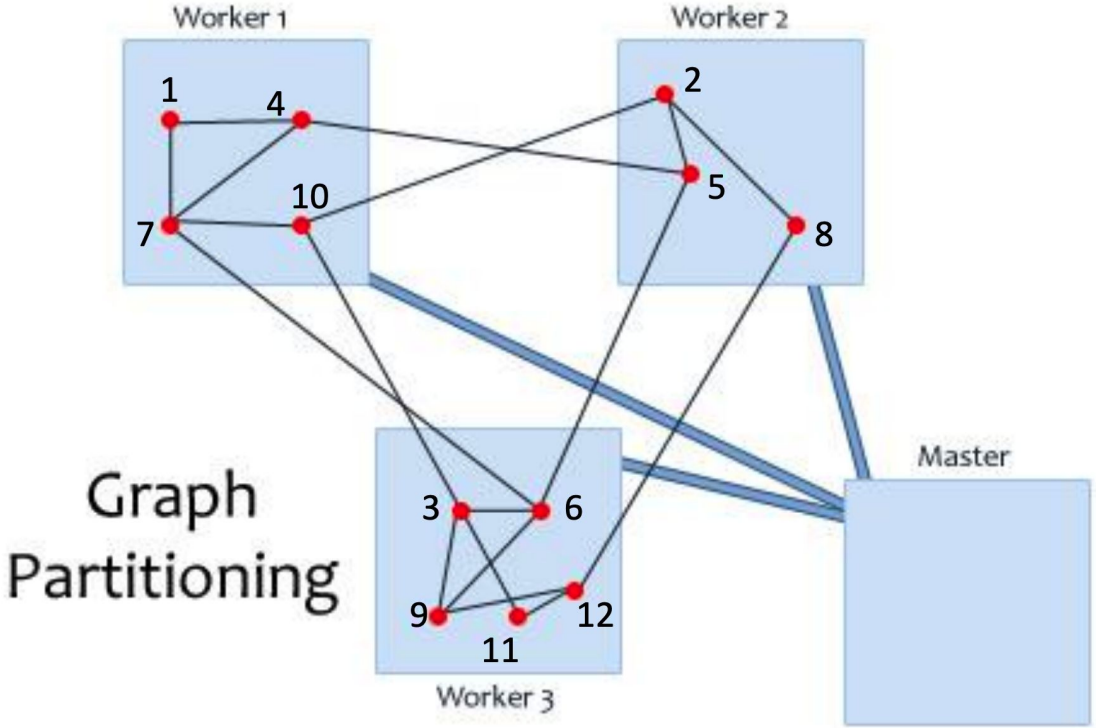
- Algorithm termination is based on every vertex voting to halt
  - In super-step 0, every vertex is active
  - All active vertices participate in the computation of any given super-step
  - A vertex deactivates itself by voting to halt and enters an inactive state
  - A vertex can return to active state if it receives an external message



Vertex State Machine

- A Pregel program terminates when all vertices are simultaneously inactive and there are no messages in transit

Graph Partitioning



The Pregel API in C++

You subclass the Vertex Class

```
template <typename VertexValue,
typename EdgeValue,
typename MessageValue>

class Vertex {
public:
    virtual void Compute(MessageIterator* msgs) = 0;

    const string& vertex_id() const;
    int64 superstep() const;
    const VertexValue& GetValue();
    VertexValue* MutableValue();
    OutEdgeIterator GetOutEdgeIterator();

    void SendMessageTo(const string& dest_vertex,
    const MessageValue& message);

    void VoteToHalt();
};
```

To define the types for vertices, edges and messages

Override the compute function to define the computation at each superstep

To get the value of the current vertex

To modify the value of the vertex

To pass messages to other vertices

SAPIENZA
UNIVERSITÀ DI ROMA

PageRank in Pregel

```
class PageRankVertex:public Vertex<double, void, double> {
 public:
    virtual void Compute (MessageIterator * msgs)
    {
        const double DAMPING_FACTOR = 0.85;
        const int64 MAX_ITER = 30;
        if (superstep() >= 1) {
            double sum = 0;
            for (; !msgs->done(); msgs->Next())
                sum += msgs->Value();
            *MutableValue() = (1 - DAMPING_FACTOR) + (DAMPING_FACTOR * sum);
        }
        if (supersteps() < MAX_ITER) {
            const int64 n = GetOutEdgeIterator().size();
            SendMessageToAllNeighbors(GetValue() / n);
        } else {
            VoteToHalt();
        }
    }
};
```

Using Pregel in Apache Spark (PageRank implementation)

```scala
import org.apache.spark.graphx.GraphLoader

// Load the edges as a graph
val graph = GraphLoader.edgeListFile(sc, "data/graphx/followers.txt")
// Run PageRank
val ranks = graph.pageRank(0.0001).vertices
// Join the ranks with the usernames
val users = sc.textFile("data/graphx/users.txt").map { line =>
  val fields = line.split(",")
  (fields(0).toLong, fields(1))
}
val ranksByUsername = users.join(ranks).map {
  case (id, (username, rank)) => (username, rank)
}
// Print the result
println(ranksByUsername.collect().mkString("\n"))
```

HITS - Kleinberg 1999

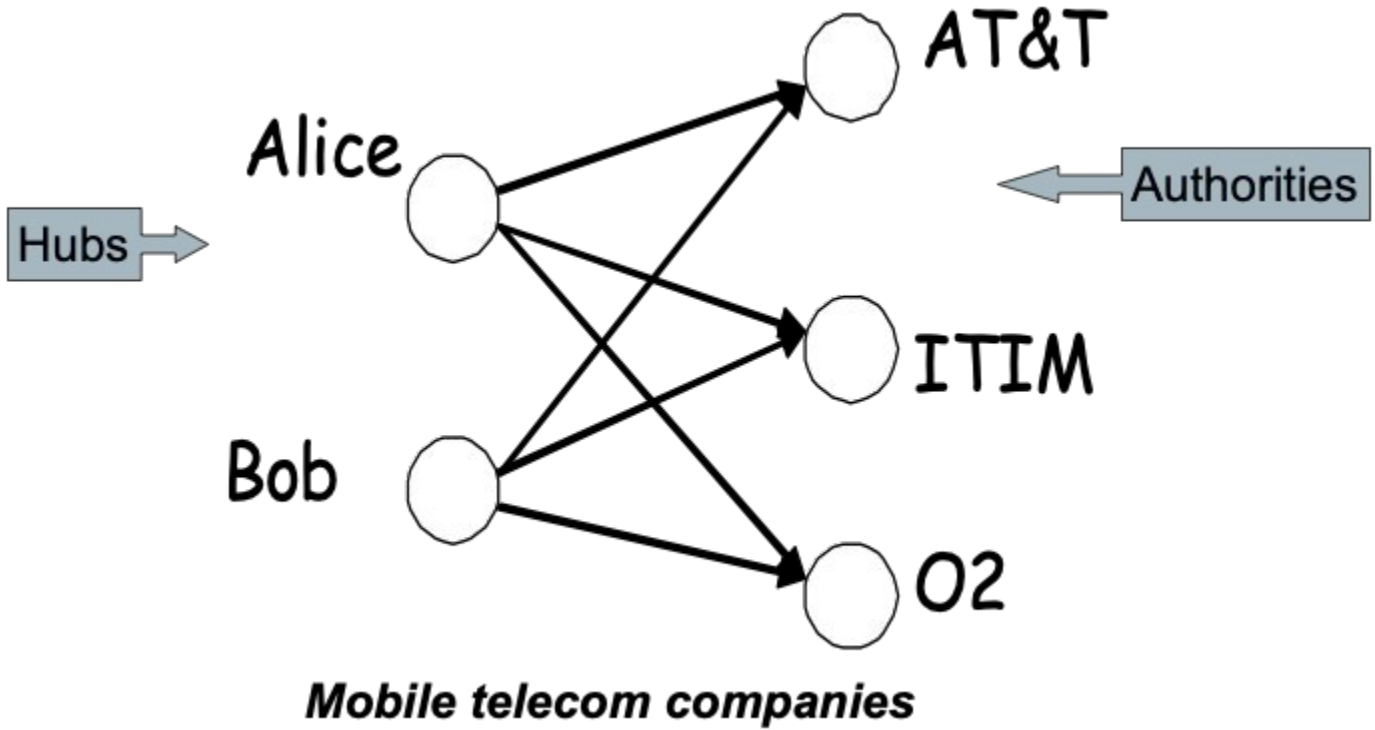
- **H**yperlink-**I**nduced **T**opic **S**earch (HITS)
- In response to a query, instead of an ordered list of pages each meeting the query, find two sets of inter-related pages:
  - Hub pages are good lists of links on a subject
    - e.g., “Bob’s list of cancer-related links.”
  - Authority pages occur recurrently on good hubs for the subject
- Best suited for “broad topic” queries rather than for page-finding queries
- Gets at a broader slice of common opinion

Hubs and Authorities

- Thus, a good hub page for a topic points to many authoritative pages for that topic.
- A good authority page for a topic is pointed to by many good hubs for that topic.

- Circular definition – will turn this into an iterative computation.

Ideally



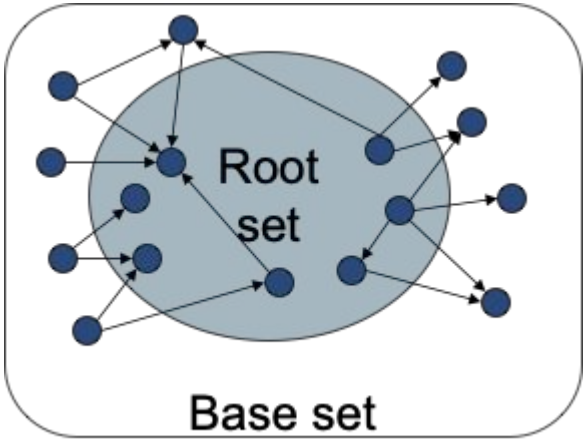
The High Level Algorithm

- Extract from the web a **base set** of pages that *could* be good hubs or authorities.
- From these, identify a small set of top hub and authority pages;

  → iterative algorithm.

The Base Set

- Given text query (say browser), use a text index to get all pages containing browser.
  - Call this the **root set** of pages.
- Add in any page that either
  - points to a page in the root set, or
  - is pointed to by a page in the root set.
- Call this the **base set**.



Get in-links (and out-links) from a *connectivity server* (a.k.a. Graph DB)

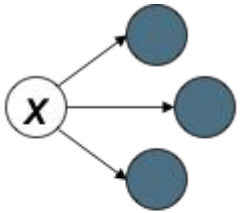
The High Level Algorithm

- Compute, for each page x in the base set, a hub score *h(x)* and an authority score *a(x)*.
- Initialize: for all x, *h(x)* ← 1; *a(x)* ← 1;
- Iteratively update all *h(x)*, *a(x)*; Key
- After iterations
  - output pages with highest *h()* scores as top **hubs**
  - highest *a()* scores as top **authorities**.

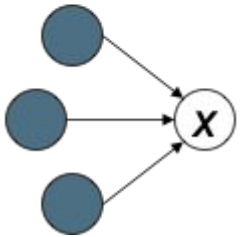
Iterative Update of *h(x)* and *a(x)*

- Repeat the following updates, for all x:

$$h(x) \leftarrow \sum_{x \mapsto y} a(y)$$



$$a(x) \leftarrow \sum_{y \mapsto x} h(y)$$



Scaling

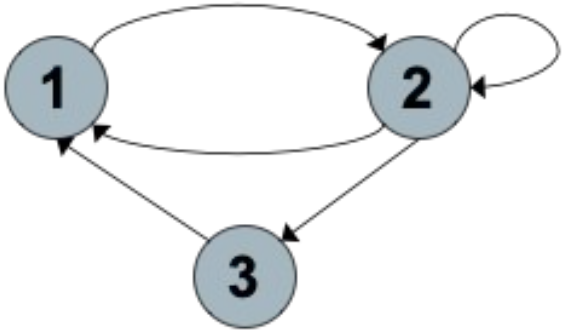
- To prevent the *h()* and *a()* values from getting too big, can scale down after each iteration.
- Scaling factor doesn’t really matter:
  - we only care about the relative values of the scores.

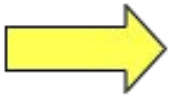
How many Iterations

- Claim: relative values of scores will converge after a few iterations:
  - in fact, suitably scaled, h() and a() scores settle into a steady state!
  - proof of this comes later.
- In practice, ~5 iterations get you close to stability.

Proof of Convergence

- nxn adjacency matrix **A**:
  - each of the n pages in the base set has a row and column in the matrix.
  - Entry A_{ij} = 1 if page *i* links to page *j*, else = 0.





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

Hub/Authority Vectors

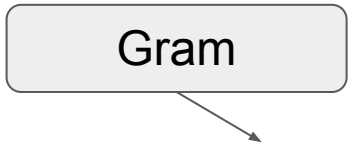
- View the hub scores **h()** and the authority scores **a()** as vectors with n components.
- Recall the iterative updates

$$h(x) \leftarrow \sum_{x \mapsto y} a(y)$$

$$a(x) \leftarrow \sum_{y \mapsto x} h(y)$$

In Matrix Form

- $\mathbf{h} = A\mathbf{a}$
- $\mathbf{a} = A^T\mathbf{h}$



- $\mathbf{h} = AA^T\mathbf{h}$
- $\mathbf{h}$ is the principal eigenvector of AA^T

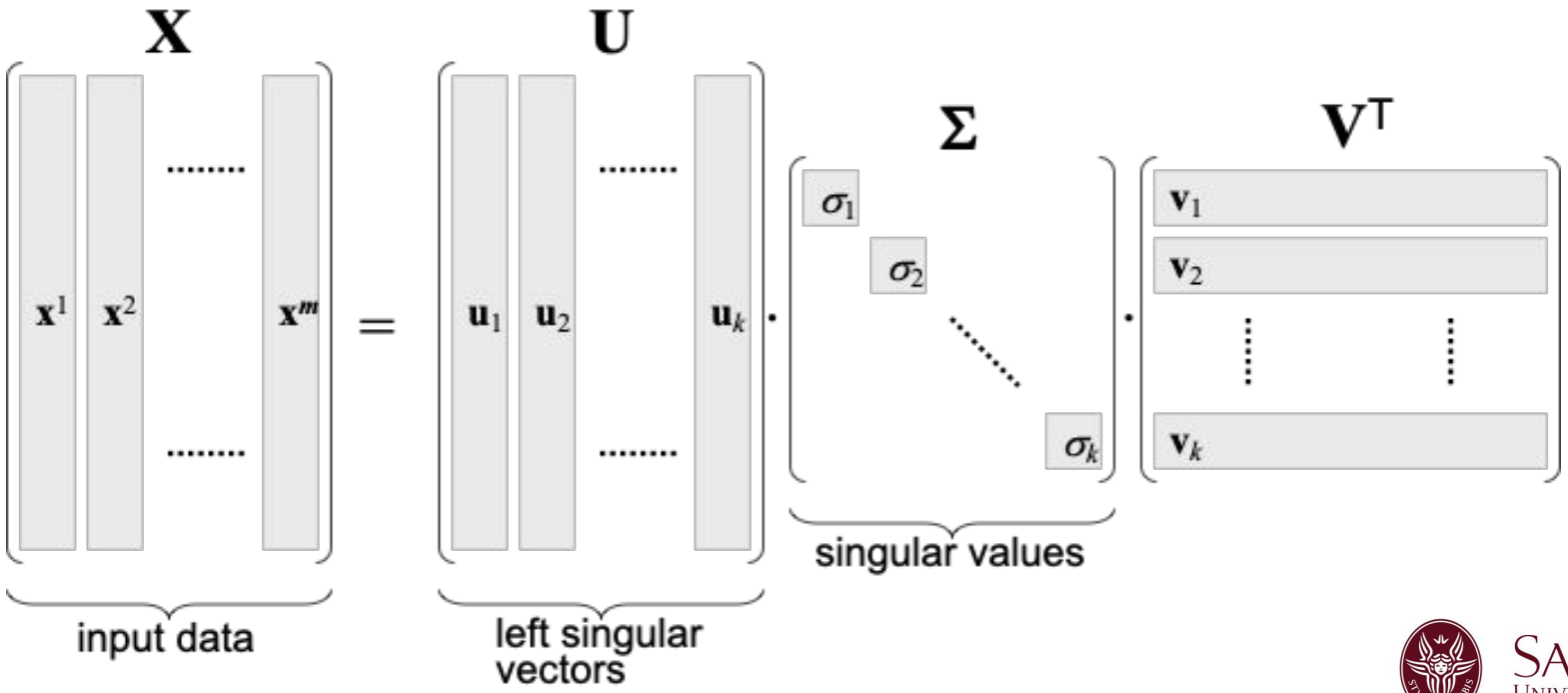
$\mathbf{a} = A^TA\mathbf{a}$

$\mathbf{a}$ is the principal eigenvector of A^TA

Can be computed using Power Method, in this case guaranteed to converge

Singular Value Decomposition

$$\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^{\mathsf{T}}$$



SVD and HITS

- V are the eigenvectors of the covariance matrix A^TA
- U are the eigenvectors of the Gram (inner-product) matrix AA^T

- Therefore, to compute Hub and Authorities score we could compute the SVD of the matrix A
  - SVD is not as scalable as the Power Method

Example of Authorities… found in 1999

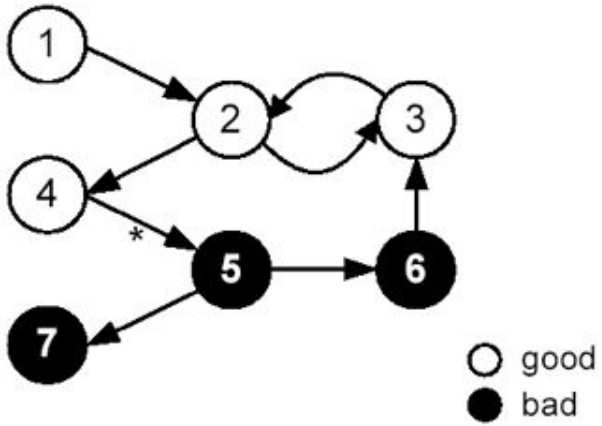
- (java) Authorities
  - .328 http://www.gamelan.com/ Gamelan
  - .251 http://java.sun.com/ JavaSoft Home Page
  - .190 http://www.digitalfocus.com/... The Java Developer: How Do I ...
  - .190 http://lightyear.ncsa.uiuc.edu/;srp/java/ javabooks.html
  - .183 http://sunsite.unc.edu/javafaq/javafaq.html comp.lang.java FAQ
- (censorship) Authorities
  - .378 http://www.eff.org/ EFFweb—The Electronic Frontier Foundation
  - .344 http://www.eff.org/blueribbon.html The Blue Ribbon Campaign for Online Free Speech
  - .238 http://www.cdt.org/ The Center for Democracy and Technology
  - .235 http://www.vtw.org/ Voters Telecommunications Watch
  - .218 http://www.aclu.org/ ACLU: American Civil Liberties Union

Issues

- Topic Drift
  - Off-topic pages can cause off-topic “authorities” to be returned
    - E.g., the neighborhood graph can be about a “super topic”
- Mutually Reinforcing Affiliates
  - Affiliated pages/sites can boost each others’ scores
    - Linkage between affiliated pages is not a useful signal

TrustRank

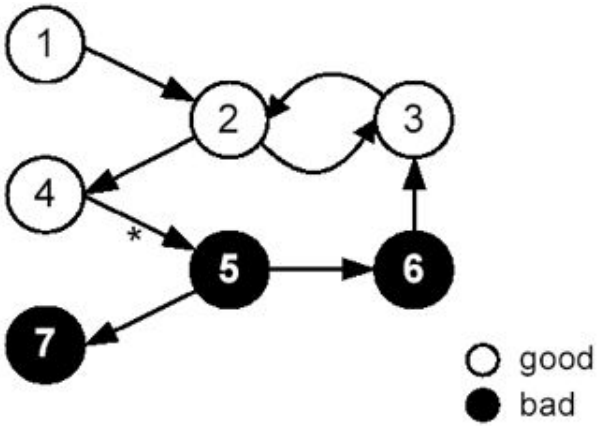
- Spamming in Web
  - Good and bad pages
- TrustRank is used to overcome Spamming
  - It proposes techniques to semi-automatically separate reputable, good pages from spam.
  - It first selects a small set of seed pages to be evaluated by an expert.
  - Once it manually identifies the reputable seed pages, then it uses the link structure of the web to discover other pages that are likely to be good



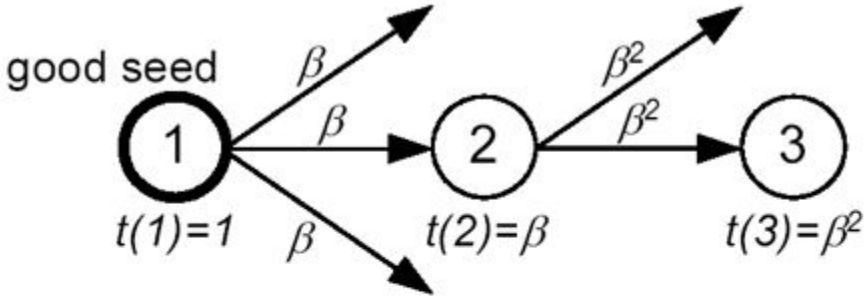
TrustRank

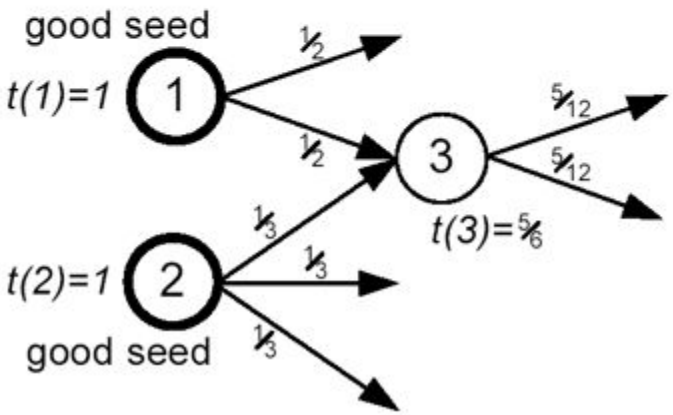
- It formalizes the notion of a human checking a page for spam by a binary oracle function O over all pages p:

$$O(p) = \begin{cases} 0 & \text{if } p \text{ is bad,} \\ 1 & \text{if } p \text{ is good.} \end{cases}$$



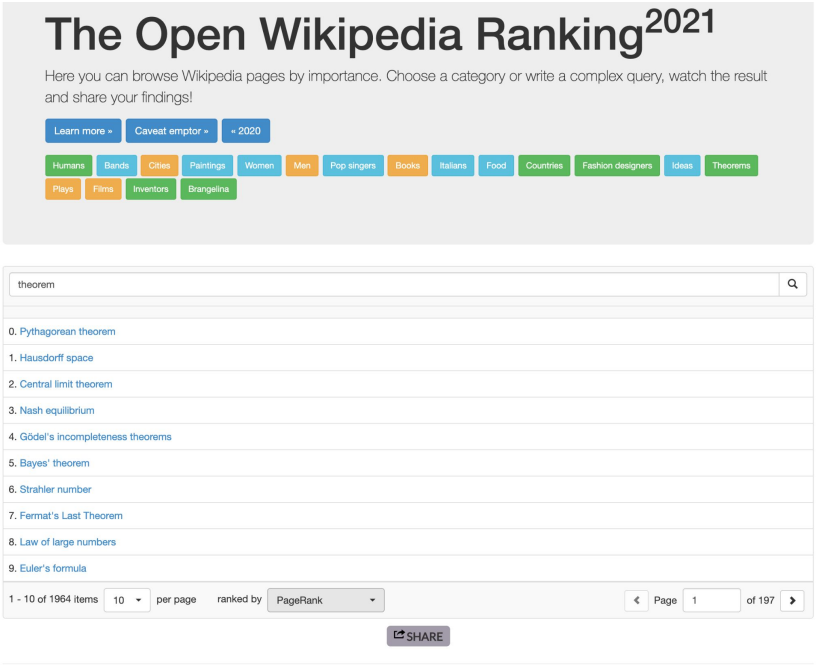
TrustRank: Trust Damping and Trust Splitting





Other Applications of Random Walk With Restart

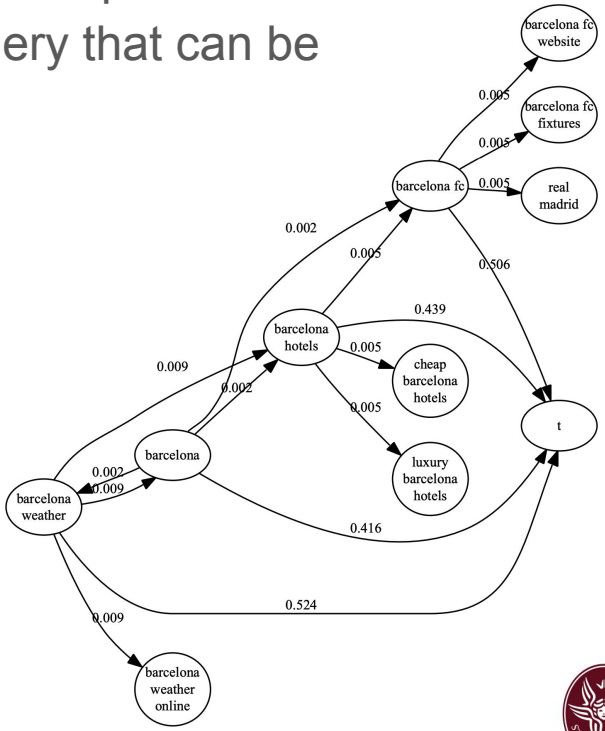
- wikirank-2021.di.unimi.it



SAPIENZA UNIVERSITÀ DI ROMA

Other Applications of Random Walk With Restart

- Query Suggestions and the Query-Flow Graph
- Given a query Q, find the most likely query that can be reached from Q.
    - RWR from Q



SAPIENZA
UNIVERSITÀ DI ROMA

Other Applications of Random Walk With Restart

- Tourism Recommender Systems
  - A person visiting two Points of Interest (PoI) u, v induces an edge in a graph where vertices are PoIs in a city
- RWR from a starting PoI can be used to recommend places to visit from that PoI.

Other Applications of Random Walk With Restart

- Tourism Recommender Systems

	Florence	Glasgow	San Francisco
Number of PoIs	1,022	353	550
Images gathered from Flickr	124,223	176,981	937,389
Number of distinct albums (at least two photos)	2,919	1,971	4,411
Average distinct PoIs per album	3.71	4.97	3.61
Number of edges	131,238	25,486	39,372
Edges from Flickr	22,164	19,150	26,752
Edges from Wikipedia's Categories	111,778	8,644	16,038
Maximum (out)degree	415	103	263
Average (out)degree	121.86	72.20	71.59

Other Applications of Random Walk With Restart

- Tourism Recommender Systems

Starting PoIs in U
Palazzo Vecchio
Piazza della Signoria

Top-10 ranked PoIs

PoI	Probability
Ponte Vecchio	$5.9 \cdot e^{-4}$
Piazzale Michelangelo	$2.1 \cdot e^{-4}$
Palazzo Pitti	$1.9 \cdot e^{-4}$
Giotto's Campanile	$6.8 \cdot e^{-5}$
Boboli Gardens	$4.9 \cdot e^{-5}$
Loggia dei Lanzi	$4.6 \cdot e^{-5}$
Piazza Santa Croce	$4.2 \cdot e^{-5}$
Uffizi	$4.1 \cdot e^{-5}$
Basilica of Santa Croce	$3.9 \cdot e^{-5}$
Ponte alle Grazie	$3.4 \cdot e^{-5}$

Starting PoIs in U
La Specola
Museo Fiorentino di Preistoria
Museo Horne
Bargello

Top-10 ranked PoIs

PoI	Probability
Uffizi	$1.4 \cdot e^{-10}$
Giotto's Campanile	$1.2 \cdot e^{-10}$
Palazzo Medici Riccardi	$9.8 \cdot e^{-11}$
Vasari Corridor	$7.4 \cdot e^{-11}$
Medici Chapel	$6.5 \cdot e^{-11}$
Basilica of Santa Croce	$5.3 \cdot e^{-11}$
San Marco's National Museum	$1.3 \cdot e^{-11}$
Dante Alighieri's House	$9.6 \cdot e^{-12}$
Modern Art Gallery	$9.3 \cdot e^{-12}$
Museo Stibbert	$8.0 \cdot e^{-12}$

Other Applications of Random Walk With Restart

- Tourism Recommender Systems

Starting PoIs in U
Clyde Tunnel
Govan Subway Station
Hillhead Subway Station
Renfrew Airport

Top-10 ranked PoIs	
PoI	Probability
Glasgow International Airport	$1.2 \cdot e^{-8}$
Buchanan Street Subway Station	$4.2 \cdot e^{-9}$
Kelvinbridge	$6.8 \cdot e^{-10}$
Glasgow Seaplane Terminal	$2.4 \cdot e^{-10}$
St Enoch Subway Station	$2.0 \cdot e^{-10}$
Glasgow City Heliport	$2.0 \cdot e^{-10}$
Buchanan Bus Station	$9.5 \cdot e^{-11}$
Ibrox Subway Station	$9.5 \cdot e^{-11}$
Kelvinhall Subway Station	$8.3 \cdot e^{-11}$
Cowcaddens Subway Station	$9.5 \cdot e^{-12}$

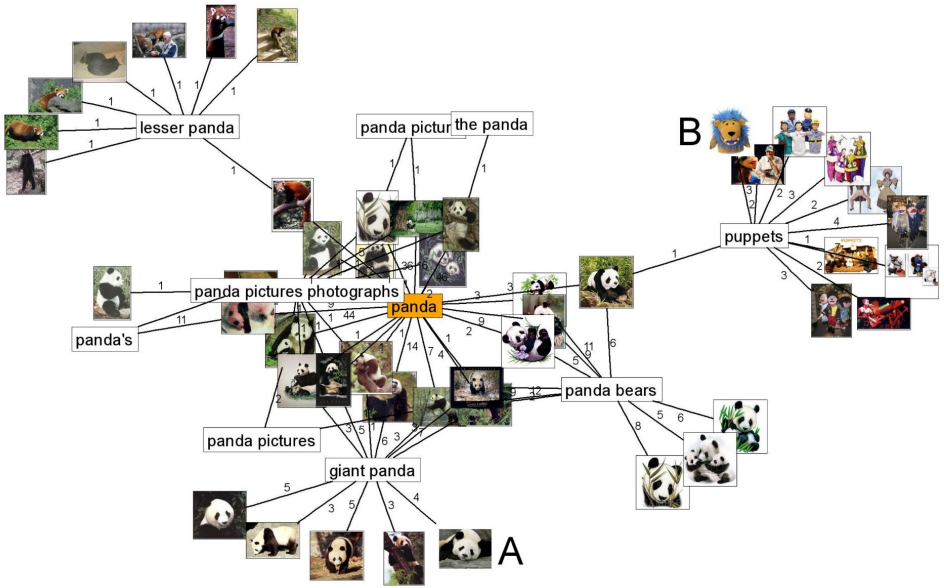
Starting PoIs in U
Golden Gate Theatre
San Francisco Conservatory of Music

Top-10 ranked PoIs	
PoI	Probability
War Memorial Opera House	$1.1 \cdot e^{-5}$
Dolores Park	$1.0 \cdot e^{-5}$
Castro Theatre	$8.1 \cdot e^{-6}$
Yerba Buena Gardens	$7.8 \cdot e^{-6}$
Embarcadero Center	$7.3 \cdot e^{-6}$
Metreon	$6.3 \cdot e^{-6}$
Golden Gate Bridge	$5.5 \cdot e^{-6}$
Pacific-union Club	$4.2 \cdot e^{-6}$
Lake Merritt	$4.1 \cdot e^{-6}$
American Conservatory Theater	$3.9 \cdot e^{-6}$

SAPIENZA
UNIVERSITÀ DI ROMA

Random Walking a Bipartite Graph

- Intuition: queries and clicked docs induce a bipartite graph that can be used to detect relevant documents for a given query based on co-clicks.

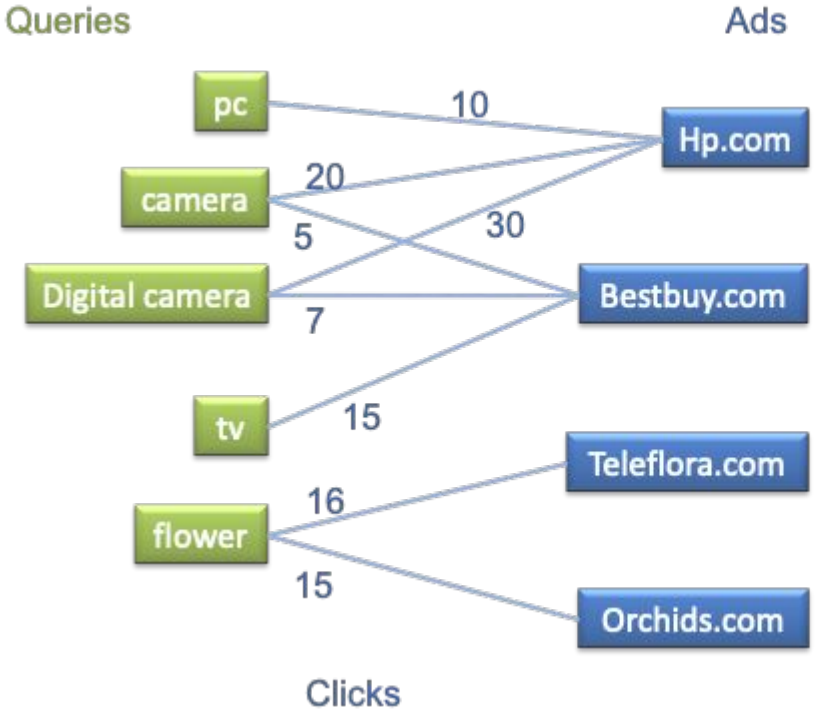


Random Walking a Bipartite Graph

- We can start a random walk from either documents or queries.
- We perform two steps at a time, this way the Markov Chain is irreducible and aperiodic.
- When we start from queries, we find similar queries on the basis of co-clicks
- When we start from documents, we find similar documents on the basis of “co-queries”.

SimRank

- Click Graph from Sponsored Search



SimRank

- Intuition:
  - “Two queries are similar if they are connected to similar ads”
  - “Two ads are similar if they are connected to similar queries”
- Iterative procedure
  - at each iteration similarity propagates through the graph

SimRank

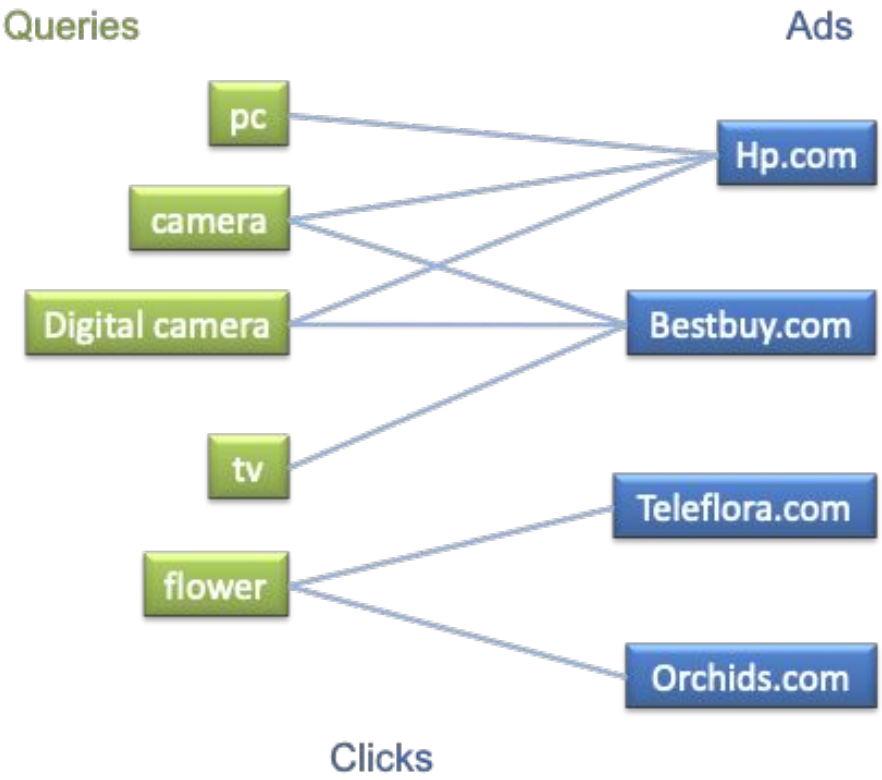
- N(q): # of ads connected to q
- E(q): set of ads connected to q
- simk(q,q'): q-q' similarity at k-th iteration
- Initially sim(q,q) = 1, sim(q,q') = 0, sim(a,a) = 1, sim(a,a') = 0

$$s_k(q,q') = \frac{C}{N(q)N(q')} \sum_{i \in E(q)} \sum_{j \in E(q')} s_{k-1}(i,j)$$

$$s_k(a,a') = \frac{C}{N(a)N(a')} \sum_{i \in E(a)} \sum_{j \in E(a')} s_{k-1}(i,j)$$

- Time: $O(n^4)$

SimRank



$$s_k(q,q') = \frac{C}{N(q)N(q')} \sum_{i \in E(q)} \sum_{j \in E(q')} s_{k-1}(i,j)$$

$$s_k(a,a') = \frac{C}{N(a)N(a')} \sum_{i \in E(a)} \sum_{j \in E(a')} s_{k-1}(i,j)$$

Two random surfers model