COMP336 — Big Data

Week 9 Lecture 1: Link Analysis

Diego Mollá

COMP336 2018H1

Abstract

In this lecture we will focus on a particular type of data that is often used in the analysis of webpages and social media: graphs. We will cover approaches that can be used for large graphs such as those encountered on Web applications. Among other methods, you will learn about PageRank as a way to determine the importance of a node in the graph.

Update May 6, 2018

Contents

Reading

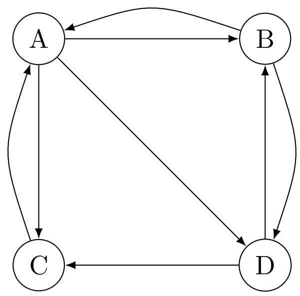
- Leskovec, Rajaraman, Ullman (2014): Mining of Massive Datasets, Chapter 5. `http://www.mmds.org/`

1 PageRank

1.1 Definition of PageRank

The Web as a Graph

- You can image the Web as a large *directed* graph.
- The webpages are the nodes of the graph.
- If there is a hyperlink from page A to page B, then the corresponding graph has a link from node A to node B.

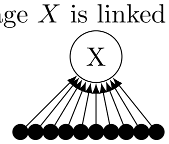


Defining the Importance of a Webpage

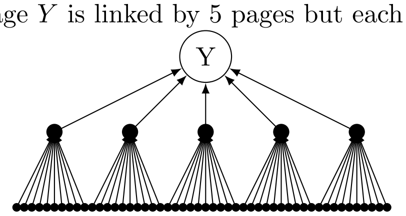
The importance of a webpage depends on two factors:

1. How many pages are linking to the page; and
2. How *important* are the pages that are linking to the page.

Scenario 1
Page X is linked by 10 pages but nobody is linking to any of these pages.



Scenario 2
Page Y is linked by 5 pages but each of them is linked by 10 other pages.



PageRank and Random Surfers

- PageRank computes the importance of a webpage in function of the importance of the pages that link to it.
- PageRank computes the importance *independently* of how relevant the page might be to the user query.
- So, a webpage that is slightly irrelevant to the query might appear in the top list just because it's important.
- The PageRank of a page A models the probability that a *random surfer* is in page A at a given time.
  - Random surfer: A web surfer that follows hyperlinks randomly.

PageRank Formula (take 1)

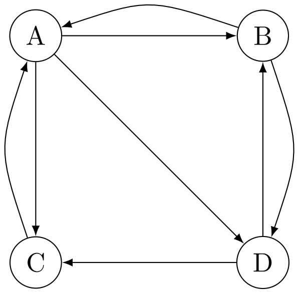
The following formula computes the importance of a page based on the importance of the other pages linking to it:

PageRank (take 1)

$$PR(A) = \frac{PR(T_1)}{C(T_1)} + \cdots + \frac{PR(T_n)}{C(T_n)}$$

T_i = page that links to A
$C(T_i)$ = number of outgoing links from page T_i

Example of Computing PageRank

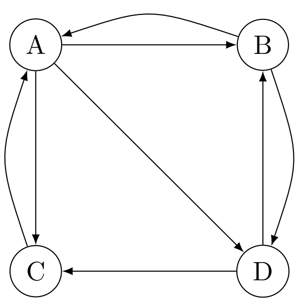


$$
\begin{aligned}
PR(A) &= \tfrac{PR(B)}{2} + \tfrac{PR(C)}{1} \\
PR(B) &= \tfrac{PR(A)}{3} + \tfrac{PR(D)}{2} \\
PR(C) &= \tfrac{PR(A)}{3} + \tfrac{PR(D)}{2} \\
PR(D) &= \tfrac{PR(A)}{3} + \tfrac{PR(B)}{2}
\end{aligned}
$$

- The idea is that the PageRank of a page (say, B) is spread equally among all the pages that it links to (in our example, A and D).
- In other words, if random surfer starts at page B, then it will next be at page A with probability 0.5, and at page D with probability 0.5.

The Transition Matrix

- We can model a step of the random surfer with the help of a *transition matrix*.
- The rows and columns of the transition matrix M represent the nodes of the network.
- The cell value at M_{ij} is the probability of the random surfer moving from j to i.



$$
M = \begin{pmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{pmatrix}
$$

Using the Transition Matrix

- We can use the transition matrix to compute the probability of being in each node given that we know the random user is in a particular node.

- For example, if the user is in node B, we apply the following matrix multiplication:

$$\begin{pmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0.5 \\ 0 \\ 0 \\ 0.5 \end{pmatrix}$$

Computing PageRank

- To compute PageRank of all nodes, we assume that a surfer begins from any node with equal probability.
- We then apply the transition matrix to determine where the surfer is next.

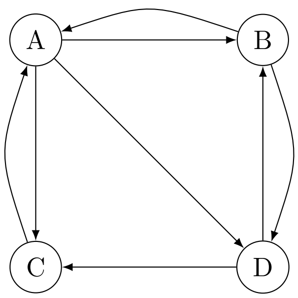
$$\begin{pmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \end{pmatrix} = \begin{pmatrix} 1/2 \times 0.25 + 1 \times 0.25 \\ 1/3 \times 0.25 + 1/2 \times 0.25 \\ 1/3 \times 0.25 + 1/2 \times 0.25 \\ 1/3 \times 0.25 + 1/2 \times 0.25 \end{pmatrix}$$

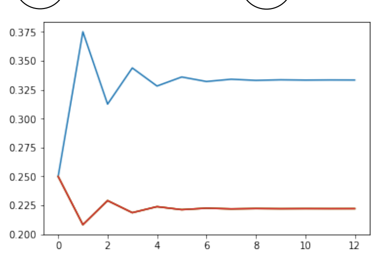
- And keep applying the transition matrix until we reach a *stationary state* (when the probabilities do not change).
- It can be shown that we will always reach a stationary state.
  - (This is connectected with the concept of matrix eigenvectors)
- In practice, the stationary state is reached after applying the transition matrix a small number of times.

Algorithm

Algorithm

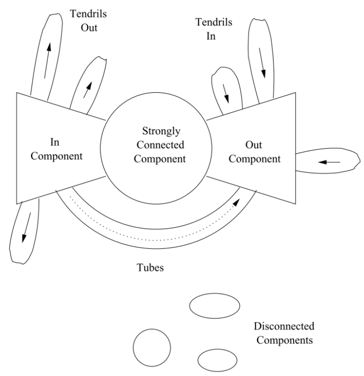
1. $PR \leftarrow$ column vector with values $1/N$
2. WHILE PR changes:
3. $\quad PR \leftarrow M \cdot PR$





The Bowtie Picture of the Web

- The Web is not as strongly connected as in our example above.
- It has a large part that is strongly connected but others that are not.



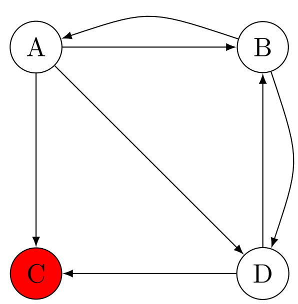
- *In-component*: Can reach SCC, but not reachable from SCC.
- *Out-component*: Reachable from SCC but unable to reach SCC.
- *Tendrils*: One-way connections to either the in-component or the out-component.
- *Tubes:* From the in-component to the out-component.
- Small isolated components.

The Problem with Dead Ends

- The parts from the network that are not part of the strongly connected component create problems with our first version of PageRank.
- Our first version assumes that transition matrix is *stochastic*:
  - For every column, the sum of values is 1.
- If there is a dead end, the sum of values in some columns is zero: the matrix is *substochastic*.
- In a stochastic matrix, at every iteration of PageRank the sum of PageRank values is 1.
- In a substochastic matrix, the sum of PageRank values will decrease at every iteration.

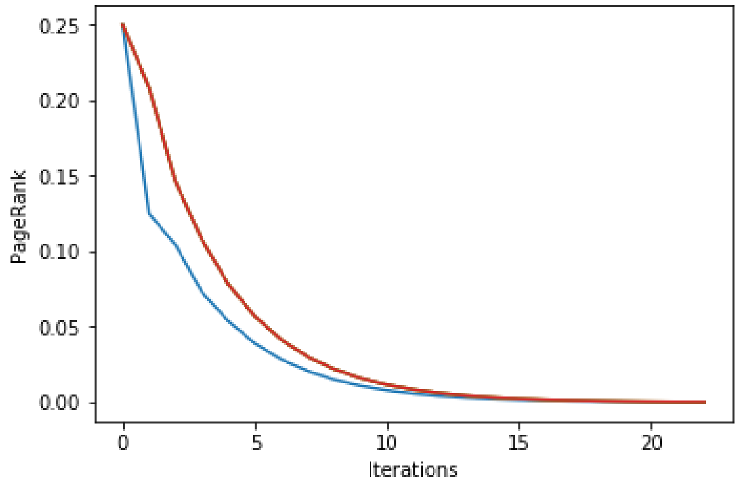
Dead End: Example

In the following network, node C is a dead end:



$$M = \begin{pmatrix} 0 & 1/2 & 0 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{pmatrix}$$

We can see that the third column of the transition matrix does not sum to 1.



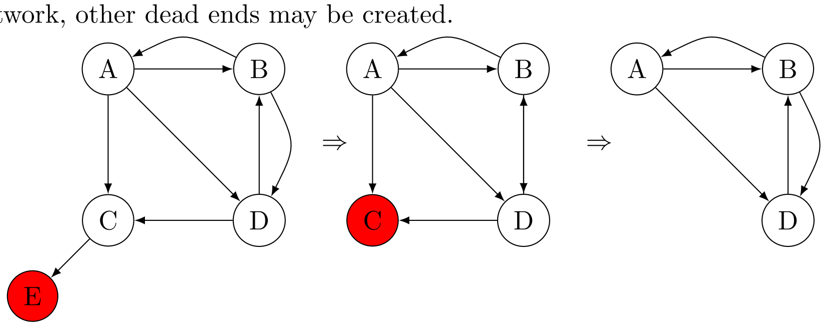
Treating Dead Ends

There are two main ways to tread dead ends:

1. Remove nodes that are dead ends and edges leading to them until there are no dead ends, and apply PageRank on the resulting strongly connected graph.
2. Apply teleporting (we will cover this later).

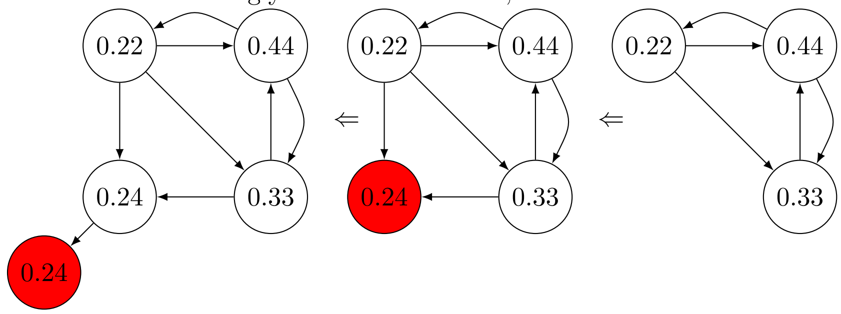
Removing Dead Ends: Example

We may need to apply several iterations to remove dead ends, since after removing the dead ends of a network, other dead ends may be created.



Restoring Dead Ends

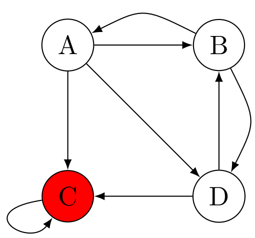
After computing PageRank on the reduced strongly connected network, we need to restore the dead ends

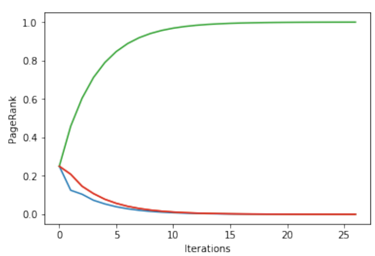


and compute their PageRank. $PR(E) = PR(C)$ $\quad$ $PR(C) = PR(A)/3 + PR(D)/2$

Spider Traps

- Spider traps are regions of the network that are strongly connected but which have no links out.
- Our simple formula for PageRank will place all PageRank scores inside spider traps.
- Below is an example with a one-node spider trap.





1.2 Teleporting

PageRank with Teleporting

- Teleporting (also called *taxation*) solves the problem of spider traps up to some extent.
- It modifies the model of the random surfer.
- When the random surfer is at node A, it has two choices:
    1. With probability β, follow one of the links randomly (as in the previous version of PageRank).
    2. With probability $1 - \beta$, teleport to a random page.
- Teleporting solves the problem of spider traps.

PageRank with Teleporting

$$PR(A) = \beta(\frac{PR(T_1)}{C(T_1)} + \cdots + \frac{PR(T_n)}{C(T_n)}) + (1-\beta)\frac{1}{N}$$

T_i = page that links to A
$C(T_i)$ = number of outgoing links from page T_i
N = total number of nodes in the network.

Computing PageRank with Teleporting

To compute PageRank with teleporting, at each iteration of the computation we need to add a term for teleporting:

$$PR \leftarrow \beta M \cdot PR + (1-\beta)E\frac{1}{N}$$

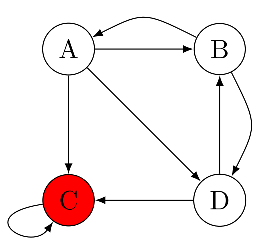
Where E is a column vector with all ones.

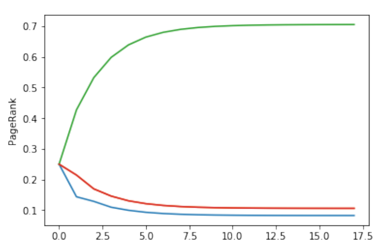
Algorithm

1. $PR \leftarrow$ column vector with values $1/N$
2. WHILE PR changes:
3. $\quad PR \leftarrow \beta M \cdot PR + (1-\beta)E\frac{1}{N}$

Exampe of a Spider Trap with Teleporting

- We can see that teleporting does not solve the problem of spider traps completely, as node C has a much higher PageRank than the others.
- To remove the effect of spider traps is quite complicated.
- Spammers try to create complex spider traps to fool the search engine.





2 Efficient Computation of PageRank

2.1 Efficient Representation of the Transition Matrix

Representing Transition Matrices

- Transition matrices are very sparse.
  - Many of the elements of the transition matrix are zero.
- Also, all non-zero terms in a column are equal and their sum is 1.
- So, we only need to represent, for every column, the list of non-zero rows.

Transition Matrix

$$M = \begin{pmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{pmatrix}$$

Efficient Representation

Source	Degree	Destination
A	3	B, C, D
B	2	A, D
C	1	A
D	2	B, C

PageRank with Efficient Matrix Representation

Algorithm

1. $PR^{old} \leftarrow$ column vector with values $1/N$
2. WHILE PR changes:
3. $PR^{new} \leftarrow$ matrix initialised to $(1-\beta)/N$
4.   FOR each page $i = 1, 2, \cdots, N$:
5.   Read into memory: $i, d_i, dest_1, \cdots dest_{di}, PR_i^{old}$
6.   FOR $j = 1, 2, \cdots, d_i$:
7. $PR_{dest_j}^{new} \leftarrow PR_{dest_j}^{new} + \beta PR_i^{old}/d_i$
8. $PR^{old} \leftarrow PR^{new}$

2.2 Using MapReduce

PageRank Iteration Using MapReduce

- We can use MapReduce to implement each iteration of the computation of PageRank:

$$PR \leftarrow \beta M \cdot PR + (1-\beta)E\frac{1}{N}$$

- If N is small enough that the PR column vector fits in main memory, the PageRank of each node can be computed with simple MapReduce operations.
- Often N is not large enough and we would have to resort to striping (see textbook, section 2.3.2).

MapReduce Iteration if N is not Too Large

Here we will focus on the matrix-vector computation, since all other parts of the MapReduce computation are straightforward.

$$X_i = \sum_{j=1}^{N} M_{ij} PR_j$$

Map

- The map function is written to apply to one element of M.
- The compute node performing the map task reads PR entirely in memory (if it hasn't done it in a previous map task), to avoid *thrashing*.
- Return the key-value pair $(i, M_{ij}PR_j)$.

Reduce

- The reduce function sums all values associated with a given key i and returns the pair (i, X_i).

Use of Combiners to Consolidate the Result Vector

- We can process a block of the matrix in one node.
- This can be more efficient, and allows us to process large PR vectors.
- We partition M into k^2 square blocks, and PR into k stripes.
- We then use k^2 map tasks.

$$\begin{array}{|c|}\hline X_1 \\ \hline X_2 \\ \hline X_3 \\ \hline X_4 \\ \hline \end{array} \quad \leftarrow \quad \begin{array}{|c|c|c|c|}\hline M_{11} & M_{12} & M_{13} & M_{14} \\ \hline M_{21} & M_{22} & M_{23} & M_{24} \\ \hline M_{31} & M_{32} & M_{33} & M_{34} \\ \hline M_{41} & M_{42} & M_{43} & M_{44} \\ \hline \end{array} \quad \begin{array}{|c|}\hline PR_1 \\ \hline PR_2 \\ \hline PR_3 \\ \hline PR_4 \\ \hline \end{array}$$

MapReduce Tasks Using Combiners

Map

- Read one square block M_{ij}.
- Read PR_j (j is the same as the second index of M_{ij}).
- Return key-value pair $(i, M_{ij} \cdot PR_j)$ (this is a matrix-vector multiplication).

Reduce

- Read all values associated with a given key i and compute the vector sum X_i.
- Return the pair (i, X_i).

Take-home Messages

- The Web as a graph.
- PageRank to compute the importance of a webpage.
- PageRank and random surfers.
- Implementing PageRank with a transition matrix.
- Dead ends and spider traps.
- Incorporating teleporting.
- Efficient representations of the transition matrix.
- Using MapReduce to compute PageRank.

What's Next

Week 10

- Frequent Itemsets