The PageRank Algorithm (Chapter 5)

Wojtek Kowalczyk

plus slides

http://www.mmds.org/mmds/v2.1/ch05-linkanalysis2.pptx

Some history

- Internet in early 80’: small, slow, local,
- Internet in early 90’: growing, global, slow
  - newsgroups
  - mailing lists
  - bulletin boards
- “primitive” search engines:
  - Lycos
  - Excite
  - Netscape
  - Yahoo!
  - MSN
  - “hybrid”: combinations of existing engines
- **1997: google.stanford.edu; 15 September 1997: domain google.com registered**

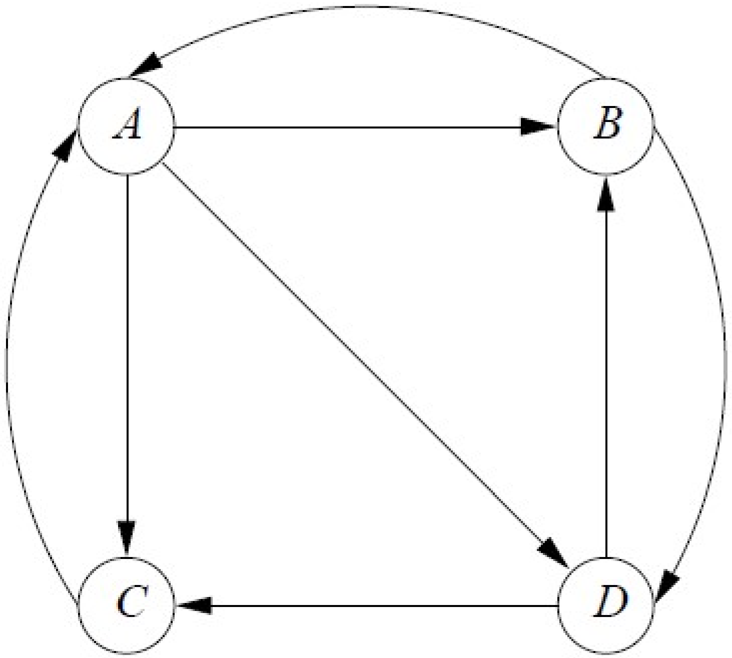
The key to success: the PageRank algorithm

- Invented by Sergiey Brin and Larry Page in 1996 (Stanford University)
- Estimates “page importance” by analyzing the structure of links
- Some Rough Statistics (from August 29th, 1996)
  - Total indexable HTML urls: 75 Million
  - Total content downloaded: 207 gigabytes ...
  - BackRub is written in Java and Python and runs on several Sun Ultras and Intel Pentiums running Linux. The primary database is kept on a Sun Ultra II with 28GB of disk. Scott Hassan and Alan Steremberg have provided a great deal of very talented implementation help. **Sergey Brin** has also been very involved and deserves many thanks.
    -**Larry Page** pagecs.stanford.edu
- By the end of 1998, Google had an index of about 60 million pages
- *http://en.wikipedia.org/wiki/History_of_Google*

The PageRank Algorithm (Ch. 5)

- How Google determines the importance of a page?
- A “random surfer” model:
  - visitors start their session at random pages
  - visitors walk along links at random
  - choices are made uniformly (each outgoing link has the same chance)
  - “page importance” = “frequency the surfer visits the page”
- It can be modeled by a Markov Process
  - transition matrix
  - iterative calculation of page probability distribution
  - solution = principal eigen vector of the transition matrix
- Nowadays a much more sophisticated model is used …

The transition matrix



$$
\begin{array}{cc}
 & \begin{array}{cccc} A & B & C & D \end{array} \\
M = & \left[\begin{array}{cccc} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{array}\right] \begin{array}{c} A \\ B \\ C \\ D \end{array}
\end{array}
$$

Each column X lists probabilities of going to Y

Entries in each column sum up to 1

v=[1/4, 1/4, 1/4, 1/4]’
What is Mv? What is M(Mv)? M(M(v))?…

The transition matrix: iterating Mv

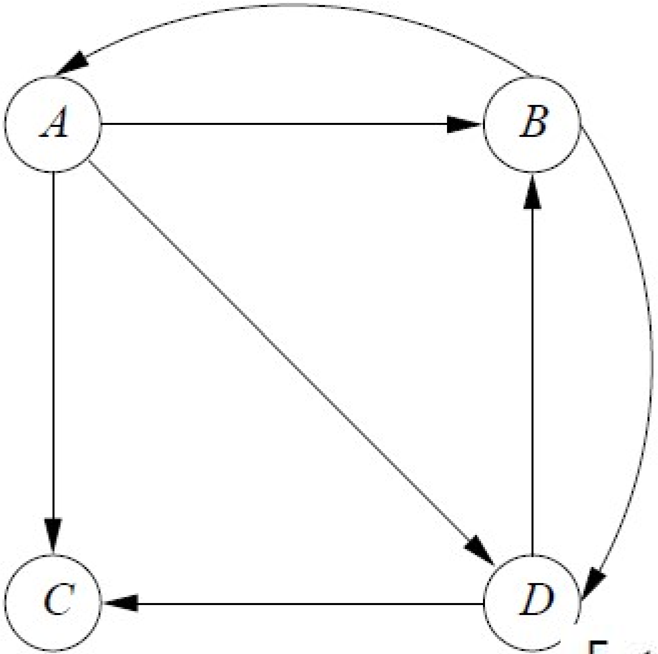
$$M = \begin{array}{c} \\ \\ \\ \\ \\ \end{array}\begin{array}{cc} \begin{array}{cccc} \;\;A & \;\;\;\;B & \;\;C & \;\;\;D \end{array} & \\ \begin{bmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix} & \begin{array}{c} A \\ B \\ C \\ D \end{array} \end{array}$$

$$\begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix} \begin{bmatrix} 9/24 \\ 5/24 \\ 5/24 \\ 5/24 \end{bmatrix} \begin{bmatrix} 15/48 \\ 11/48 \\ 11/48 \\ 11/48 \end{bmatrix} \begin{bmatrix} 11/32 \\ 7/32 \\ 7/32 \\ 7/32 \end{bmatrix} \cdots \begin{bmatrix} 3/9 \\ 2/9 \\ 2/9 \\ 2/9 \end{bmatrix}$$

Convergence of the Markov process

- If:
  - the graph is strongly connected
    (i.e., there is a path between any two nodes)
  - there are no “dead ends” (nodes with no links going out)
- then
  - the sequence v, Mv, M(Mv), … converges to v’ such that v’=Mv’
  - v’ is the principal eigen vector of matrix M
    (principal = biggest eigen value)
- In practice, 50-70 iterations are sufficient
- … even for huge M (billions x billions , sparse) …
- For small M, v=Mv can be solved directly…

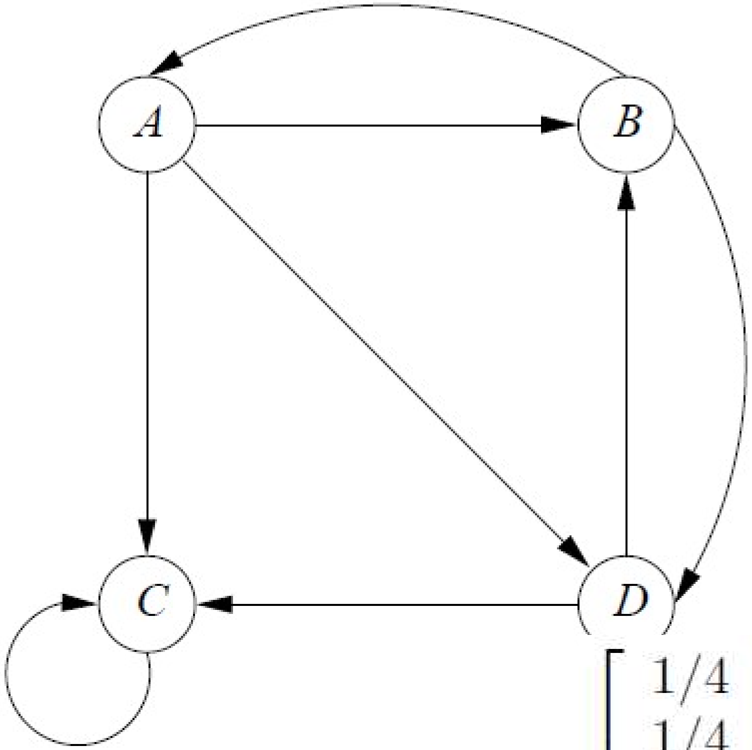
Dead-ends



$$\begin{array}{cc} & \begin{array}{cccc} A & B & C & D \end{array} \\ M = & \begin{bmatrix} 0 & 1/2 & 0 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix} \begin{array}{c} A \\ B \\ C \\ D \end{array} \end{array}$$

$$\begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix} \begin{bmatrix} 3/24 \\ 5/24 \\ 5/24 \\ 5/24 \end{bmatrix} \begin{bmatrix} 5/48 \\ 7/48 \\ 7/48 \\ 7/48 \end{bmatrix} \begin{bmatrix} 21/288 \\ 31/288 \\ 31/288 \\ 31/288 \end{bmatrix} \cdots \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Spider-traps



$$M = \begin{array}{c@{}c} \begin{array}{cccc} A & B & C & D \end{array} & \\ \begin{bmatrix} 0 & 1/2 & 0 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 1 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix} & \begin{array}{c} A \\ B \\ C \\ D \end{array} \end{array}$$

$$\begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix} \begin{bmatrix} 3/24 \\ 5/24 \\ 11/24 \\ 5/24 \end{bmatrix} \begin{bmatrix} 5/48 \\ 7/48 \\ 29/48 \\ 7/48 \end{bmatrix} \begin{bmatrix} 21/288 \\ 31/288 \\ 205/288 \\ 31/288 \end{bmatrix} \cdots \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

Teleporting

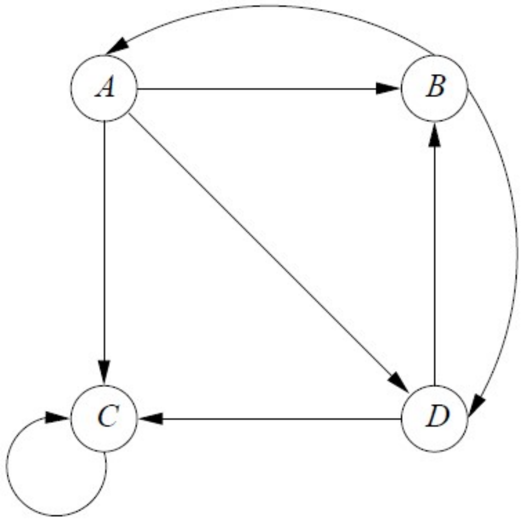
- At each step, decide:
  - with probability β ($0 < \beta \leq 1$) continue random walk
  - with probability $(1 - \beta)$ jump to any other node
- The corresponding update rule:

  $$v' = \beta M v + (1 - \beta) e/n$$ (*e* is a vector of *n* 1's)

- β is usually 0.8 or 0.9
- "adding extra links": all problems solved (*really?*)

Teleporting: example (slide 9 continued...)

β = 0.8 = 4/5



$$\mathbf{v}' = \begin{bmatrix} 0 & 2/5 & 0 & 0 \\ 4/15 & 0 & 0 & 2/5 \\ 4/15 & 0 & 4/5 & 2/5 \\ 4/15 & 2/5 & 0 & 0 \end{bmatrix} \mathbf{v} + \begin{bmatrix} 1/20 \\ 1/20 \\ 1/20 \\ 1/20 \end{bmatrix}$$

$$\begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix} \begin{bmatrix} 9/60 \\ 13/60 \\ 25/60 \\ 13/60 \end{bmatrix} \begin{bmatrix} 41/300 \\ 53/300 \\ 153/300 \\ 53/300 \end{bmatrix} \begin{bmatrix} 543/4500 \\ 707/4500 \\ 2543/4500 \\ 707/4500 \end{bmatrix} \cdots \begin{bmatrix} 15/148 \\ 19/148 \\ 95/148 \\ 19/148 \end{bmatrix}$$

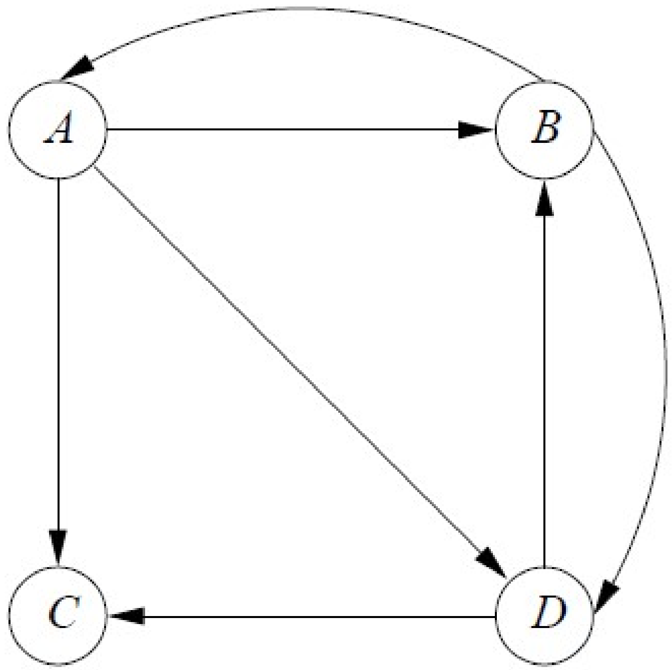
Teleporting: dead ends (slide 8 continued…)

```
Beta=0.8

M=[0,    1/2,  0,   0;
   1/3,   0,    0,  1/2;
   1/3,   0,    0,  1/2;
   1/3, 1/2,   0,   0]

v=[1/4, 1/4, 1/4, 1/4]'


for i=1:20
   v=Beta*M*v +(1-Beta)*ones(size(v))/length(v)
end
```



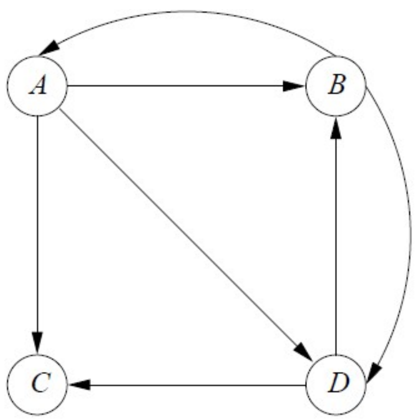
Teleporting: dead ends (slide 8 continued…)

Convergence after 20 iterations:

Iter:	0	1	2	3	10	20
	0.2500	0.1500	0.1367	0.1207	0.1018	0.1014
	0.2500	0.2167	0.1767	0.1571	0.1290	0.1284
	0.2500	0.2167	0.1767	0.1571	0.1290	0.1284
	0.2500	0.2167	0.1767	0.1571	0.1290	0.1284

Does it make sense?

- **Why is sum(v) < 1**? Surfers "die" at dead ends? Not exactly…
- Is sum(v) always >0 ? Yes: it is at least (1-Beta)
- What should be the case:
    - p(A)=0.5*p(B)?
    - p(A)= 0.5*p(B) + (1-Beta)/4?
    - **p(A)= Beta*0.5*p(B) + (1-Beta)/4? Check it!**



Teleporting: dead-ends and spider traps

For graphs with no dead-ends "teleporting" works fine:
the process converges to a vector v, such that sum(v)=1
and can be interpreted as "probabilities".

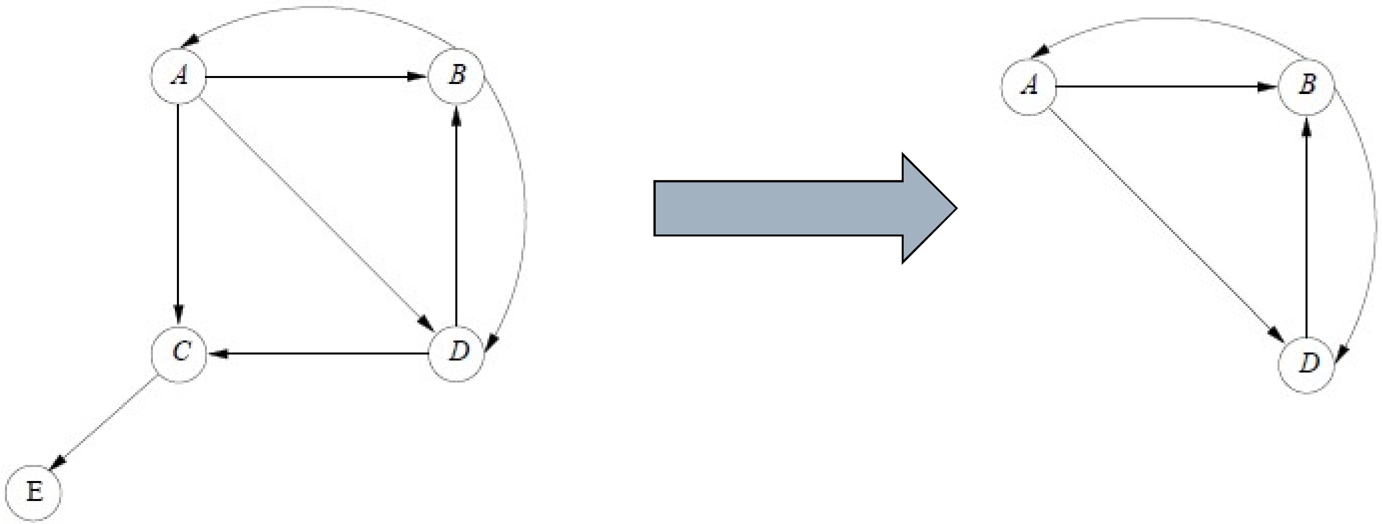
For graphs with dead-ends, the "teleporting" still works,
but the sum(v) may be smaller than 1 (no probabilistic interpretation).
Still, useful in practice (used in the original "Google" paper).

An alternative algorithm for handling dead-ends (read **Section 5.1.4**!)

- recursively remove dead ends and corresponding links,
- calculate of PageRank for the nodes of the remaining fragment of the graph,
- propagate computed values to the removed nodes.

Another alternative: when reaching a dead end jump the next node
"uniformly at random with probability 1"

Example from 5.1.4



After removal of nodes E and C, calculate PageRank of A, B, D and set:

PageRank(C) = 1/2*PageRank(A)+1/2*PageRank(D)

PageRank(E) = PageRank(C)

Note that sum of PageRanks is now bigger than 1!

Implementation of PageRank

- Key ideas:
  - M is very sparse: say 10 links per page => 10 non-zeros in a column
  - Use “inverted indexing” to represent M: a list of <node, outdegree, children of the node>
  - keep on harddisk M and vector v_old
  - keep in RAM only v_new
  - update v_new in a single scan of M and v_old
- Some numbers:
  - n=1.000.000.000 (1 billion nodes)
  - RAM needed: 4GB (32bits per node)
  - harddisk: about 40GB (10xRAM)
  - a single scan: about 2-3 minutes
  - 50 iterations => 2-3 hours

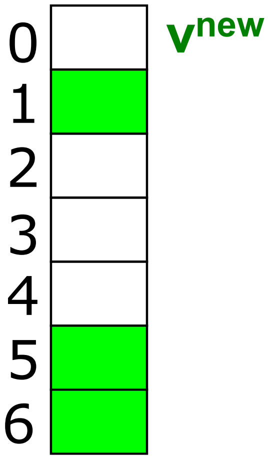
Sparse Matrix Encoding

- **Encode sparse matrix using only nonzero entries**
  - Space proportional roughly to number of links
  - Say 10N, or 4*10*1 billion = 40GB
  - **Still won't fit in memory, but will fit on disk**

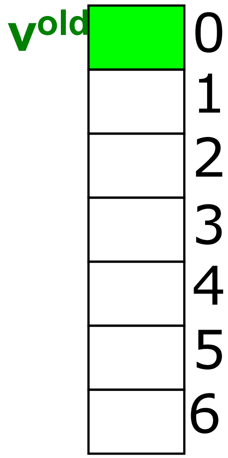
source node	degree	destination nodes
0	3	1, 5, 7
1	5	17, 64, 113, 117, 245
2	2	13, 23

- **Assume enough RAM to fit v^{new} into memory**
  - Store v^{old} and matrix **M** on disk
- **1 step of power-iteration is:**

> **Initialize** all entries of $\mathbf{v^{new}} = (1-\beta) / N$
> For each page i (of out-degree d_i):
>   Read into memory: i, d_i, $dest_1, \ldots, dest_{d_i}$, $v^{old}(i)$
>   For $\mathbf{j = 1 \ldots d_i}$
> $\mathbf{v^{new}(dest_j) \mathrel{+}= \beta\, v^{old}(i) / d_i}$



source	degree	destination
0	3	1, 5, 6
1	4	17, 64, 113, 117
2	2	13, 23



In short: "column-wise" computations

When multiplying **M** by **v** we organize computations
"by columns": "a single, linear scan through the harddisk"

- data is already organized in this way,
- we access elements of "**old v**" one by one (a, b, c),
- outdegrees (one per column) are easy to find.

$$\begin{bmatrix} A & B & C \\ D & E & F \\ G & H & I \end{bmatrix} * \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} \boxed{Aa} + \boxed{Bb} + \boxed{Cc} \\ \boxed{Da} + \boxed{Eb} + \boxed{Fc} \\ \boxed{Ga} + \boxed{Hb} + \boxed{Ic} \end{bmatrix}$$

A3: Structure of wikipedia links

Go to https://zenodo.org/record/2539424 and fetch the file:

https://zenodo.org/record/2539424/files/enwiki.wikilink_graph.2004-03-01.csv.gz?download=1

Investigate the graph:

- Dead ends
- Distribution of in-degrees
- Distribution of out-degrees of nodes
- Implement the page rank algorithm from slide 18
- Implement direct (sparse) matrix multiplication
- Compare results
- *Is this graph strongly connected?*

Data preprocessing (prep)

The data has the following layout:

page_id_from	page_title_from	page_id_to	page_title_to
12	Anarchism	34568	16th century
12	Anarchism	35416	1793
…	…	…	…

- Extract only the 1st and the 3rd column
- Convert page_id's into consecutive integers, in such a way that you can return back to the original numbering
- Both columns should use the same coding!
- Save the prepared data on HD

Exploratory data analysis (eda)

- Dead ends: find nodes with no outgoing edges. How many have you found?
- Distribution of in-degrees: for every node compute the number of incoming edges
- Distribution of out-degrees: for every node compute the number of outgoing edges
- Make nice & informative plots of both distributions
- What is the average out-degree and the average in-degree of the graph?

Estimate RAM requirements: (eda)

1. How much RAM would you need to store the transition matrix M and the initial vector v in RAM? Assume double precision (64 bits per number).

2. The same question assuming that you store M in a sparse matrix (in RAM)?

3. The same question, assuming that you use data structures as described on slide 17.

embed your answers in the notebook eda.ipynb

Implement PageRank algorithm (sparse)

1. Store both M (as a sparse matrix) and v (in RAM).
2. Run 25 iterations of the “classical” update rule from slide 10, with Beta=0.8.
3. Plot the MSE of the differences (25 numbers): v – Mv
4. Assume that your computer has 1GB RAM and the average out-degree of a graph G is 15.

   What is the maximal number of nodes of G such that your algorithm could be executed on your computer?

Implement PageRank algorithm from slide 18

1. Store both M and v^{old} and v^{new} in RAM.
   (Normally M and v^{old} stored on the hard disk)
2. Implement the algorithm from slide 18. Run 25 iterations of this algorithm with Beta=0.8.
3. Plot the MSE of the differences (25 numbers): $v - Mv$
4. How much time is needed to run a single iteration?
5. Have you obtained similar/identical results as in the previous task? What might be the source of eventual differences?

What to deliver?

Four Jupyter notebooks:

- prep.ipynb
- eda.ipynb
- sparse.ipynb
- PageRank.ipynb

that correspond to all the subtasks.

Your notebooks should read/write files from/to the same directory as your notebooks. We will test your programs in a directory which contains wikilink_graph.2004-03-01.csv