PageRank - Page, Brin (1998)

View web as directed graph $G$
- nodes of $G$ represent web pages $\{P_1, \ldots, P_n\}; n \approx 1$ trillion $= 10^{12}$.
- edges of $G$ denote links from one web page to another.

Example 1: $n = 4$, edges: $\{12, 24, 31, 32, 34, 43\}$

“Random walk” experiment
Choose a starting node from $\{1, \ldots, n\}$
for $k = 1 : N$
Choose at random an outlink from current node; click on it.
end

The PageRank of node $i$ is defined by:

$$r_i = \lim_{N \to \infty} \frac{\text{no. of times node } i \text{ visited}}{N}$$

Note: For example 1, a limit $r = \begin{pmatrix} .111 \\ .222 \\ .333 \\ .333 \end{pmatrix}$ is approached, regardless of starting node.

Note: Our definition of PageRank makes sense only if limit is always the same, independent of starting node. This is guaranteed to be the case if $G$ is strongly connected, i.e., there is a path between every pair of nodes.

Probabilistic model of random walk

Definition: A probability vector is a vector of nonnegative components whose sum is 1.

Notation: $\mathcal{P} = \{\text{probability vectors } p\}$

For our random walk experiment, we define a vector $p^{(k)} \in \mathcal{P}$ by:
$p^{(k)}_i \equiv$ probability that random walk is at node $i$ after $k$ steps.

$$p^{(0)}_i = \begin{cases} 1, & \text{if walk starts at node } i \\ 0, & \text{otherwise} \end{cases}$$

$$p^{(k+1)}_i = \sum_{j=1}^{n} A_{i,j} p^{(k)}_j \text{ where}$$
$$A_{i,j} = \text{probab of transition from node } j \text{ to node } i \text{ assuming we’re at node } j.$$
The matrix $A = \{A_{i,j}\}_{i,j=1}^n$ is a “transition matrix.” Its $j$-th column, call it $A_j$, shows the transition probabilities away from node $j$.

The reason for constructing this mathematical model is so we can apply the iteration

Choose $p^{(0)} \in \mathcal{P}$

for $k = 0, 1, 2, \ldots$

$p^{(k+1)} = Ap^{(k)}$

and compute PageRank $r$ as $\lim_{k \to \infty} p^{(k)}$.

**Back to example 1:** Here the transition matrix is

$$A = \begin{pmatrix} 0 & 0 & .333 & 0 \\ 1 & 0 & .333 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & .333 & 0 \end{pmatrix}$$

For any $p^{(0)} \in \mathcal{P}$, the iteration $p^{(k+1)} = Ap^{(k)}$ converges to $r = \begin{pmatrix} .111 \\ .222 \\ .333 \\ .333 \end{pmatrix}$ as $k \to \infty$.

**Definition:** A column stochastic matrix $A_{n \times n} = [A_1, A_2, \ldots, A_n]$ is one whose columns $A_j$ are probability vectors.

**Claim:** If $y = Ax$ where $A$ is column stochastic, then

(i) $x \in \mathcal{P} \implies y \in \mathcal{P}$ (so $\lim_{k \to \infty} p^{(k)} \in \mathcal{P}$ assuming this limit exists.)

(ii) For any $x$, $\sum_i |y_i| \leq \sum_i |x_i|$.

**Proof:**

(i) Since $x \in \mathcal{P}$, $x_i \geq 0$ for all $i$. Likewise, all $A_{i,j} \geq 0$. So $y_i = \sum_j A_{i,j} x_j \geq 0$ for all $i$. Also,$$
\sum_i y_i = \sum_i (Ax)_i = \sum_i \sum_j A_{i,j} x_j = \sum_j x_j \sum_i A_{i,j} = \sum_j x_j = 1.
$$

(ii) $\sum_i |y_i| = \sum_i |(Ax)_i| = \sum_i |\sum_j A_{i,j} x_j| \leq \sum_i \sum_j A_{i,j} |x_j| = \sum_j |x_j| \sum_i A_{i,j} = \sum_j |x_j| \sum_i A_{i,j}$

Claim: $(A^2)_{i,j}$ is the probability of a 2-step transition from node $j$ to node $i$ (assuming we’re at node $j$ to begin with). More generally, the $i,j$-th element of $A^k$ is the probability of a $k$-step transition from node $j$ to $i$.

Proof left to reader.
But there are complications...

Complication 1. **dangling nodes** ... nodes with no outlinks (≈ \( \frac{1}{4} \) of web) ... act as sinks for random walk; columns \( A_j \) corresponding to dangling nodes are zero vectors. In fact, our random walk experiment just gets stuck if it visits a dangling node.

Google’s remedy: If at a dangling node, in next step jump to node chosen at random from all of \( G \).

new transition matrix \( \tilde{A} \)...

\[
\tilde{A}_j = \begin{cases} 
\frac{1}{n} e, & \text{if node } j \text{ has outdegree 0} \\
A_j, & \text{otherwise}
\end{cases}
\]

where \( e = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \)

equivalent representation for \( \tilde{A} \):

Let \( \chi \) be the “characteristic vector” for the dangling nodes, i.e.,

\[
\chi_j = \begin{cases} 
1, & \text{if node } j \text{ has outdegree 0} \\
0, & \text{otherwise}
\end{cases}
\]

Then

\[
\tilde{A} = A + \frac{1}{n} e \chi^T.
\]

Note that \( \tilde{A} \) is guaranteed to be a “column stochastic” matrix.

**Example 2:** \( n = 4 \), links: \( \{21, 23, 24, 31, 32, 34\} \)

\[
A = \begin{pmatrix} 
0 & .333 & .333 & 0 \\
0 & 0 & .333 & 0 \\
0 & .333 & 0 & 0 \\
0 & .333 & .333 & 0
\end{pmatrix}
\]

\[
\tilde{A} = \begin{pmatrix} 
.25 & .333 & .333 & .25 \\
.25 & 0 & .333 & .25 \\
.25 & .333 & 0 & .25 \\
.25 & .333 & .333 & .25
\end{pmatrix} = A + \frac{1}{4} \begin{pmatrix} 
1 \\
1 \\
1 \\
1
\end{pmatrix} \begin{pmatrix} 
1 \\ 0 \\ 0 \\ 1 
\end{pmatrix}; \quad r = \begin{pmatrix} 
.2857 \\
.2143 \\
.2143 \\
.2857
\end{pmatrix}
\]
Complication 2. *unreachable nodes* ... subgraphs of $G$ act as sinks.

Google’s remedy: Fix $\alpha \in (0,1)$. For each node... follow prescription of $\tilde{A}$ with probability $\alpha$, jump to randomly chosen node in all of $G$ with probability $1 - \alpha$. Typical value for $\alpha$: .85

new transition matrix $\tilde{A}$...

$$\tilde{A} = \alpha \tilde{A} + \frac{1 - \alpha}{n} ee^T$$

**Example 3:** $n = 6$, links: \{12, 13, 31, 32, 35, 45, 46, 54, 56, 64\}, $\alpha = .9$

$$\tilde{A} = \begin{pmatrix} 0 & .167 & .333 & 0 & 0 & 0 \\ .5 & .167 & .333 & 0 & 0 & 0 \\ .5 & .167 & 0 & 0 & 0 & 0 \\ 0 & .167 & 0 & 0 & .5 & 1 \\ 0 & .167 & .333 & .5 & 0 & 0 \\ 0 & .167 & 0 & .5 & .5 & 0 \end{pmatrix}, \quad \tilde{A} = \begin{pmatrix} .017 & .167 & .317 & .017 & .017 & .017 \\ .467 & .167 & .317 & .017 & .017 & .017 \\ .467 & .167 & .017 & .017 & .017 & .017 \\ .017 & .167 & .017 & .467 & .467 & .917 \\ .017 & .167 & .317 & .467 & .017 & .017 \\ .017 & .167 & .017 & .467 & .467 & .017 \end{pmatrix}$$

$$p^{(k)} \rightarrow \begin{pmatrix} .037 \\ .054 \\ .042 \\ .375 \\ .206 \\ .286 \end{pmatrix}$$

Now suppose a particular query is entered, and keyword search indicates that pages 1,3,4,6 have potential relevance. These pages are then sorted by PageRank and reported back to user in the order 4,6,3,1.
Computation of PageRank

Method 1. Choose initial vector \( p^{(0)} \in \mathcal{P} \) and iterate:

\[
(1) \quad p^{(k+1)} = \bar{A}p^{(k)}, \quad k = 0, 1, 2, \ldots
\]

until a limit – the PageRank \( r \) – is reached (to within a prescribed tolerance). Substituting for \( \bar{A} \) in the above iteration, we get

\[
(2) \quad p^{(k+1)} = \alpha \bar{A}p^{(k)} + \frac{1 - \alpha}{n} e e^T p^{(k)}
\]

Thus if \( p^{(k)} \to r \) as \( k \to \infty \), then \( r \) must satisfy

\[
(3) \quad r = \alpha \bar{A}r + \frac{1 - \alpha}{n} e,
\]
equivalently,

\[
(4) \quad (I - \alpha \bar{A})r = \frac{1 - \alpha}{n} e.
\]

It’s not hard to show that this linear system has a unique solution \( r \) and that \( r \in \mathcal{P} \). We will assume this and show that each iteration of (2) reduces the error

\[
\Delta^{(k)} = p^{(k)} - r
\]

by a factor \( \leq \alpha \). Subtracting (3) from (2), we have

\[
\Delta^{(k+1)} = \alpha \bar{A}\Delta^{(k)}
\]

From part (ii) of the ‘claim’ on p. 2:

\[
\sum_i |\Delta_i^{(k+1)}| \leq \alpha \sum_i |\Delta_i^{(k)}| \text{ for all } k.
\]

Thus

\[
\sum_i |\Delta_i^{(k)}| \leq \alpha^k \sum_i |\Delta_i^{(0)}| \leq \alpha^k \sum_i (p_i^{(0)} + r_i) \leq 2\alpha^k \to 0 \text{ as } k \to \infty.
\]

Thus the error in \( p^{(k)} \) is reduced by a factor \( \leq \alpha \) per iteration. Indications are that Google uses about 50 iterations with a value for \( \alpha \) in the neighborhood of .85. This gives an overall
error reduction of \(0.8550 \approx 0.003\), implying roughly three digit accuracy, enough evidently for Google’s purposes.

Note that \(\bar{A}\) in iteration (2) will in general contain many nonzeros, assuming the presence of dangling nodes. It would be much better to have the very sparse matrix \(A\) to deal with - for the real web \(G\), it’s \(99.999999\ldots\%\) zeros! Accordingly, we replace \(\bar{A}\) in (2) by \(A + \frac{1}{n}e\chi^T\) to get our final iterative algorithm for computing \(r\):

Choose \(p^{(0)} \in \mathcal{P}\)

for \(k = 0, 1, 2, \ldots\)

\[
d_k = \chi^T p^{(k)} = \sum_{\text{dangling nodes} i} p_i^{(k)}
\]

\[
p^{(k+1)} = \alpha Ap^{(k)} + \frac{1 - \alpha + \alpha d_k}{n} e
\]

The quantity \(d_k\) amounts to the probability of winding up at a dangling node after \(k\) steps of the algorithm.

**Method 2.** Note that the preceding algorithms (1), (2), or (5) amount to iterative methods for solving the linear system (4). Another obvious option would be to solve (4) directly, using Gaussian elimination.

As noted already, the coefficient matrix of this system is not particularly sparse, as a result of dangling nodes. It would be much better, from the standpoint of applying Gaussian elimination efficiently, to have the very sparse matrix \(I - \alpha A\) as coefficient matrix for our system. We can accomplish this by means of the following “trick”...

Write (4) in the equivalent form:

\[
(I - \alpha A) r = [(1 - \alpha) + \alpha \chi^T r] \frac{e}{n},
\]

The solution \(r\) of (6) must have the form

\[
r = \gamma s
\]

where \(s\) is the solution of

\[
(I - \alpha A) s = \frac{e}{n},
\]

since (6) and (8) have the same coefficient matrix and their right-hand side vectors are multiples of each other. Moreover, (8) has the desired sparse coefficient matrix. Here’s
how we can get the value of $\gamma$, thus $r$, from the solution of (8): Substituting $r = \gamma s$ in (6), we get $\gamma_n = [(1 - \alpha) + \alpha \chi^T s]_n$. Thus

\[
\gamma = \frac{1 - \alpha}{1 - \alpha \chi^T s}.
\]

For relatively small subsets of the web, applying Gaussian elimination as in (7)-(9) is potentially feasible, or perhaps even applying it to the nonsparse system (4). But for $A$ representing the entire web, Gaussian elimination isn’t a viable option for solving the corresponding systems - they’re way too large. In addition, Gaussian elimination typically does not work efficiently on a “sparse” system with lots of zeros in the coefficient matrix, as is the case for $I - \alpha A$. Zeros in the coefficient matrix get replaced by nonzeros as the elimination proceeds. In general, iterative methods are more efficient for solving such a system.

We mention one final approach to computing $r$...

Recall from linear algebra that if $Ax = \lambda x$ where $x$ is a nonzero vector and $\lambda$ is a scalar, then $x$ is an eigenvector of $A$ with corresponding eigenvalue $\lambda$. Now $r$ satisfies

$$r = \bar{A}r$$

since it’s the limit of (1) as $k \to \infty$. Thus $r$ is an eigenvector of $\bar{A}$ and the corresponding eigenvalue is $\lambda = 1$. (It can be shown that the other eigenvalues of $\bar{A}$ all have magnitude less than 1.) We conclude that PageRank is the eigenvector corresponding to the dominant eigenvalue $\lambda = 1$ of $\bar{A}$. This observation leads to other possibilities for computing $r$ which we won’t discuss here.

**Problem:** For the (disconnected) 6-node web with links $\{12, 23, 24, 31, 41, 43, 56, 65\}$...

(i) What is the corresponding directed graph $G$?

(ii) Take $\alpha = .85$ and use Matlab to compute the corresponding PageRank vector $r$:

(a) iteratively, by (1), (2), or (5).

(b) via Gaussian elimination applied to (4), or as in (7)-(9).

(iii) What happens to the definition of PageRank in the limit $\alpha \to 1$? What happens to the computational algorithms you used in (ii)?