Web-page ranking:

HITS (Kleinberg's Hub and Authority algorithm)

PageRank (Google's original algorithm)

Disclaimer: *These abbreviated notes DO NOT substitute the textbook for this class. They should be used IN CONJUNCTION with the textbook and the material presented in class. If there is a discrepancy between these notes and the textbook, ALWAYS consider the textbook to be correct. Report such a discrepancy to the instructor so that he resolves it. These notes are only distributed to the students taking this class with A. Gerbessiotis in Spring 2012 ; distribution outside this group of students is NOT allowed.*

The relevance of a query to a document (e.g. a web-page) is established by first computing the function $s(q, d_j)$ that determines the similarity between query q and document d_j.

This similarity measure is based solely on information retrieval-based knowledge (e.g. the searchable contents of the document). Search engines (or web-searching methods in general) do not rely on this measure exclusively to rank a web-page (e.g. document d_j).

They also use additional information such as the link structure of d_j. The link structure relates to information such as:

- (a) what documents/pages are pointed by d_j and are these pages important (e.g. authoritative),
- (b) what documents/pages point to d_j and are these pages important (e.g. hubs of information such as a directory page).

All such link information can be made to contribute a separate component to the rank of d_j or a web-page in general.

Thus the **rank** of d_j relative to query q denoted by $R(q, d_j)$ is a function that will represent the rank of a document/web-page. One component of $R(q, d_j)$ is $s(q, d_j)$.

One of the first attempts to use linkage information was the **vector spread activation model** by Yuwono.

Vector spread activation model. The ranking score $R(q, d_j)$ is the sum of the similarity $s(q, d_j)$ of d_j plus a portion of the similarity of each document that points to d_j. This is under the assumption that if relevant documents point to d_j, then d_j should also be relevant. Let $l(i, j)$ be 1 or 0 based on whether document d_i points to d_j or not. Then

$$R(q, d_j) = s(q, d_j) + \beta \sum_i R(q, d_i) l(i, j)$$

where β is a user defined parameter (e.g. $\beta = 0.2$).

References:

1. M. Cutler, Y. Shih, and W. Meng: Using the Structures of HTML Documents to Improve Retrieval. Usenix Symposium on Internet Technologies and Systems (USITS'97) . pp.241-251, Monterey, California, December 1997.

2. M. Cutler, H. Deng, S. Manicaan, and W. Meng: A New Study on Using HTML Structures to Improve Retrieval. Eleventh IEEE Conference on Tools with Artificial Intelligence (ICTAI'99) , Chicago, November 1999.

HITS: Authority and Hub page rank (Kleinberg 1998)

Kleinberg (1998) introduced a model that assigns two ranks to each web-page/document. One rank signifies how important the document is as a hub of information, and the other how important it is as an authoritative source of information. The resulting ranking algorithm is sometimes referred to as **HITS** i.e. a Hypertext Induced Topic Search.

The ideas behind the model can be summarized as follows.

(1) A page is an **authoritative** page if it is referenced by many **hub** pages that are **relevant** to the query.

(2) A page is a **hub** page for a query if it points to many authoritative pages for that query.

(3) Good **authoritative** and **hub** pages reinforce one another.

Finding pages for such a query. How can one find such authoritative and hub pages for a query q?

Step 1. Submit query q to a similarity-based engine and record the top n i.e. the **root set** $RS(q)$ pages based on $s(q, d_j)$ results.

Step 2. Expand set $RS(q)$ into the **base set** $BS(q)$ to include pages pointed by $RS(q)$ pages.

Step 3. Also include into $BS(q)$ pages pointing to $RS(q)$ pages (if possible, but restrict the number for each $RS(q)$ page).

For each document d_j compute the authority and hub score of d_j i.e. compute $a(d_j)$ and $h(d_j)$ by performing two operations A (for authority) and H (for hub) that update vectors $a(.)$ and $h(.)$ respectively. (We are also going to include superscripts to the vectors and their elements to denote, time or orders of update.)

Let E be the set of links in $BS(q)$ and a link from document d_i to d_j is denoted by the pair (i, j).

A: Authority update step. For every link $(i, j) \in E$ that points to document d_j

$$a(d_j) = \sum_{(i,j)\in E} h(d_i). \tag{1}$$

H: Hub update step. For every link $(j, k) \in E$ of documents d_k pointed by d_j

$$h(d_j) = \sum_{(j,k)\in E} a(d_k). \tag{2}$$

HITS algorithm: A and H, authority and hub operations

The two steps of Eq. 1 and Eq. 2 are interwined: one uses the values computed from the other. There are two ways to perform this computation: (a) synchronous, (b) asynchronous.

In a **synchronous computation**, prior values are kept in an auxiliary vector and those are used in the right hand side of Eq. 1 and Eq. 2. After both steps are performed then the new vectors overwrite the previous vector values and the computation proceeds to the next iteration.

In the **asynchronous computation** the right hand-side of Eq. 2 uses the newly computed values of the A step. Thus values get used as soon as possible. This way the order of computation becomes important: the A step is executed first followed by the H step.

Let L be the adjacency matrix over the set of links E of $BS(q)$. With this we mean that $L(i,j) = 1$ if $(i,j) \in E$. Otherwise, $L(i,j) = 0$ for $(i,j) \notin E$.

Let L^t be the transpose matrix of L i.e. one that results if the rows of L become columns in L^t and the columns respectively rows.

Let $h^{(i)}$, $a^{(i)}$ be the hub, authority vectors after i iterations of the A,H steps, for the n documents of the corpus.

$$h^{(i)} = (h^{(i)}(d_1), h^{(i)}(d_2), \ldots, h^{(i)}(d_n)), \qquad a^{(i)} = (a^{(i)}(d_1), a^{(i)}(d_2), \ldots, a^{(i)}(d_n)),$$

Then by rewriting Eq. 1 we have

$$a^{(i)} = L^t h^{(i-1)}$$

and by rewriting Eq. 2 we obtain

$$h^{(i)} = L a^{(i)}.$$

Note that the asynchronous mode is used in this notation in the sense that the hub values at iteration i are using the authorities values just computed in iteration i (rather than the values of iteration $i-1$).

Note also that the order of execution matters. First the authority values are updated for a given iteration i and then the hub values get updated (and not the other way around).

(c) Copyright A. Gerbessiotis. All rights reserved.

HITS algorithm: Solving for a and h

These recursive formulae are interwined. If we unfold them and unwind them we get the following. (This is also known as the **power method**.)

$$\begin{aligned} a^{(i)} = L^t h^{(i-1)} \quad &\Longrightarrow \quad a^{(i)} = L^t L a^{(i-1)} \Longrightarrow \ldots \Longrightarrow a^{(i)} = (L^t L)^i a^{(0)} \\ h^{(i)} = L a^{(i)} \quad &\Longrightarrow \quad h^{(i)} = L L^t h^{(i-1)} \Longrightarrow \ldots \Longrightarrow h^{(i)} = (L L^t)^i h^{(0)} \end{aligned}$$

One step is missing in this computation. The values $a^{(i)}, h^{(i)}$ can grow large as the are dependent on $(L^t L)^i$ and $(LL^t)^i$. Thus a scaling needs to be performed immediately after a round of A and H updates gets completed. This scaling is shown below

$$a^{(i)}(d_j) = \frac{a^{(i)}(d_j)}{\sqrt{\sum_k (a^{(i)}(d_k))^2}}, \quad h^{(i)}(d_j) = \frac{h^{(i)}(d_j)}{\sqrt{\sum_k (h^{(i)}(d_k))^2}},$$

This computation (with the scaling) is expressed in Algorithm `HubAuthority-Rank` that is shown on the following page. For the computation to work properly and Algorithm `HubAuthority-Rank` to be complete we need to explain certain two missing parts of this expositions

Issue 1. What are the initialization vectors $a^{(0)}$ and $h^{(0)}$? (The answer is in line 4 of Algorithm `HubAuthority-Rank`.)

Issue 2. Why does the algorithm converge? How fast is convergence? What are the preconditions for convergence?

(c) Copyright A. Gerbessiotis. CS 667-101 : Spring 2012 . All rights reserved.

```
  Algorithm HubAuthority-Rank
1.  Obtain RS(q) for query q.
2.  Expand RS(q) into BS(q) as described.
3.  Use BS(q) to obtain a graph G=(V,E) of links with adjacency matrix L
4.  For each dj initialize hub and authority values to 1 i.e.   a(dj) = h(dj ) = 1;
5.  for every document dj component of a,h  repeat {
                                              (i)            (i-1)
6.    Run the authority update step A to get a    (dj)  from h    (...)
                                              (i)             (i)
7.    Run the hub       update step H to get h    (dj)  from a    (...)
             (i)      (i)
8.    Scale a   (dj) h  (dj) as explained.
    } until convergence is reached
9.  Return documents  ordered by authority scores (high first)
```

Figure 1: Kleinberg's HITS algorithm computation

In the scaling step one can remove the superscripts that indicate iteration and have no other significance in the computation to obtain a simpler

$$a(d_j) = \frac{a(d_j)}{\sqrt{\sum_i a^2(d_i)}}, \quad h(d_j) = \frac{h(d_j)}{\sqrt{\sum_i h^2(d_i)}},$$

(Note that the denominator is computed partially and only once by the end of step 7, then inverted so that multiplication rather than the more expensive division is performed.)

Both matrices LL^t and L^tL are well-defined and by matrix analysis (e.g. Perron-Frobenius theorem) vectors $a^{(i)}$ and $h^{(i)}$ will converge to the dominant eigenvector of one or the other matrix.

For this however to occur the initial values $a^{(0)}$ and $h^{(0)}$ must be chosen in such a way that they are not orthogonal to the solution. A choice $a^{(0)} = h^{(0)} = e$, where e is the unit vector (i.e. all its components are ones) suffices.

The rate of convergence however depends on the difference (i.e. eigengap) between the largest λ_1 and second largest λ_2 eigenvalues of the two matrices. HITS usually converges to a solution after 10-20 iterations.

Answer to Issue 1. Initial values are all ones for the hub and authority vector elements as indicated in step 4 of the algorithm.

Answer to Issue 2. Linear algebra and the all-one initialization takes care of convergence. Expect (but no guarantee is offered) convergence in 10-20 iterations.

One problem with HITS is the existence of multiple connected components in the graph represented by L. Then the algorithm might ignore all but one of these components when it converges.

We are brief at this point on the mathematical properties of HITS; a more thorough discussion that also relates to HITS will deal with Google's PageRank algorithm.

(c) Copyright A. Gerbessiotis. All rights reserved.

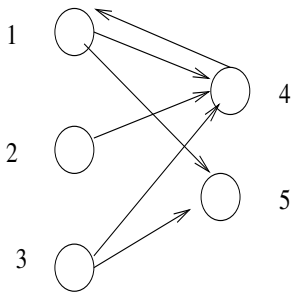


Figure 2: HITS algorithm: An example graph

Example 1. For the graph in Figure 2 we first determine its adjacency matrix L (sometimes also denoted by A). The two vectors a, h are initialized to ones.

```
    |1  2  3  4  5    Iteration  0       a=(1   1   1   1   1)     h =(  1   1   1   1   1)
   ---------------
A= 1|0  0  0  1  1    Iteration  1  A:   a=(1   0   0   3   2) <---[use h of iteration 0]
L= 2|0  0  0  1  0                  H:   [use a of iteration 1]--->h =(  5   3   5   1   0)
   3|0  0  0  1  1              Scale:   a=(.26 0   0  .80 .53)    h =(.64 .38 .64 .12   0)
   4|1  0  0  0  0    [Note sqrt(14)=3.74 and sqrt(60)=7.74]
   5|0  0  0  0  0    after few more iterations
                                       a=(0   0   0  .78 .61)    h =(.65  .36 .65  0   0)
```

As a result of those computations the ranking of the 5 pages based on authority values is $4, 5, 1, 2, 3$ since page 4 has authority rank 0.78.

(c) Copyright A. Gerbessiotis. All rights reserved.

How to fool HITS: Catching conspirators

Question 1. What if someone creates hundreds of pages each one pointing to all the other ones with each one of those pages including lots of "hot" keywords so that they are included in the initial $BS(q)$ set?

Question 2. What if some one does the same over a set of web-sites rather than web-pages of the same site?

Transverse link information: Links between pages with different domain names.

Intrinsic link information: Links within the same domain.

Answer to Questions 1 and 2. One can use different weights for the two types of transverse and intrinsic links or ignore the second type (e.g. weight becomes 0).

(c) Copyright A. Gerbessiotis. All rights reserved.

How to fool HITS: The tyranny of majority

Question 3. Say x web-pages (or web-sites if we got wised up by the prior page discussion) point to page/URL X and y pages point to URL Y where $x \gg y$. (Think of $x = 100$ and $y = 10$ and thus $x/y = 10 \gg 1$.) What happens with the hubs and authority values of X and Y respectively?

Answer to Question 3.
Example 2. We show what happens with the HITS algorithm after a single iteration of it for this case. For page X its authority value becomes x and for page Y it becomes y after the A step. Since $x \gg y$ X's authority is larger than Y's. When the hub step H is performed, the hub values of the pages pointing to X get upped to x and those of Y get (less) upped to y. After scaling, whatever the scaled values will be, we still expect $x' \gg y'$ and of course $d' \gg f'$.

```
                      X  Y  x-pages   y-pages          X  Y  x-pages  y-pages
Iteration 0 :      a=( 1  1  1 ... 1   1 ... 1)    h=( 1  1  1 ... 1  1 ... 1 )

Iteration 1 :  A:  a=( x  y  0     0   0     0)
               H:                                  h=( 0  0  x     x  y     y )
          Scale :  a=( x' y' 0     0   0     0)    h=( 0  0  d'    d' f'    f')
```

In fact we can show that $a(X) = x' = x/\sqrt{x^2 + y^2}$, and $a(Y) = y' = y/\sqrt{x^2 + y^2}$. The important observation is that $a(X)/a(Y) = x/y \gg 1$. After another iteration, $a(X)/a(Y) = (x/y)^2 \gg 1$ and so on. Thus after i iterations we expect $a(X)/a(Y) = (x/y)^i \gg 1$ and thus $a(X)$ would be much larger than $a(Y)$. Thus the more popular X will dominate the less popular Y completely. (The same would apply to hub values as well of the x and y pages.)

So if $x/y = 10$ after 3-4 iterations the X-related values will grow to become 1000 to 10000 larger than the Y-related values. X will completely dominate Y.

(c) Copyright A. Gerbessiotis. All rights reserved.

The HITS algorithm has certain limitations. To start with we need access to a similarity search engine to obtain RS plus the additional steps to formulate BS. Subsequent ranking depends on the BS that depends on the given query q. For this reason we wrote $BS(q)$. The algorithm itself uses two vectors a, h. If the web-graph related to $BS(q)$ has isolated components the HITS algorithm may focus on one such component and miss the other ones (midredirection).

The algorithm used to ranking Google query results is now presented. The original algorithm, named `PageRank` was introduced in 1998. How much it is still being used in Google is open to discussion.

We will be introducing the mathematics behind `PageRank` through some linear and matrix algebra considerations related to adjacency-matrix/adjacency-list properties of graphs. A more thorough and mathematical oriented discussion starts with page 20. (One can skip directly this latter discussion that starts on page 20.)

For an $n \times n$ matrix A, vector x is called the (right) eigenvector of eigenvalue λ of A if and only if $Ax = \lambda x$.

The spectral theorem for matrices states that if there are n eigenvectors of A that form a basis (i.e. they are linearly independent) of dimension n, then A can be written as $A = Q\Lambda Q^{-1}$ where Q is a matrix whose i-th column is the i-th eigenvector and Λ is a diagonal matrix whose i-th diagonal entry is λ_i (i.e. the eigenvalue corresponding to the i-th eigenvector).

From matrix theory we also have that the determinant of A is the products of its eigenvalues, i.e. $\mid A \mid = \lambda_1 \ldots \lambda_n$.

From now on we will assume that the eigenvalues are relabeled and numbered so that $\lambda_1 > \lambda_2 > \ldots$ i.e. we list the eigenvalues from the largest to the smallest one.

This might work for the positive and real eigenvalues; such a listing can also be extended for negative and/or complex eigenvalues if we use $\mid \lambda_1 \mid$ where $\mid . \mid$ represents the measure of a complex number or the absolute value of a real number.

Vector e is the unit vector i.e. it is an all-one vector i.e. $e^t = (1, 1, \ldots, 1)$.

We now introduce (or repeat) some definitions related to adjacency matrices of directed or undirected graphs. Let $G = (V, E)$ be a graph on a set of vertices V and a set of edges E. This graph may correspond to the web and its edges to the links of web-page. Let n be the number of vertices of V and m be the number of edges in E. It is $m \leq n^2$.

Adjacency matrix L. L is the adjacency matrix of a graph whose (i, j)-th entry is 1 if $(i, j) \in E$ and 0 otherwise.

Transpose of adjacency matrix L. L^t is the transpose of L i.e. its (i, j)-th entry is 1 if $(j, i) \in E$ and 0 otherwise.

Weighed adjacency matrix C. The weighed adjacency matrix C can be obtained from L if its (i, j)-th entry is normalized so that $c_{ij} = 1/d(i)$ for $(i, j) \in E$ and 0 otherwise. $d(i)$ is the degree (out-degree for directed graphs) for vertex i.

Thus if the i-th row of L looks like $(0 \quad 1 \quad 1)$, then the corresponding row of C would look like $(0 \quad 1/2 \quad 1/2)$.

Matrix C obtained from the adjacency matrix of a graph has a nice property: $Ce = 1e$, i.e. the unit vector is an eigenvector for eigenvalue 1. This is because the rows of C add up to 1.

We have told half the truth however or hid a minor point. Matrix C might fail to hold this property if it contains a sink, i.e. a vertex with no outgoing edges.

We can fix this by setting sink rows in C with e^t/n where n is the number of vertices of the graph.

This "corrected" matrix has the specified property (and its all zero rows have been replaced by all $1/n$ rows).

Transpose C^t of C. Transposing matrix C to get C^t we obtain a matrix that has an (i, j)-th entry that is $1/d(j)$ for $(j, i) \in E$.

(c) Copyright A. Gerbessiotis. All rights reserved.

Idea behind Google's view of the Web. View the web as a directed graph $G = (V, E)$. The HITS algorithm views as a directed graph not the web but only the base set $BS(q)$; to obtain $BS(q)$ one had to submit q to a similarity-based search engine, obtain $RS(q)$ and then enhance it into $BS(q)$. Google's view of the web is a direct view skipping all those intermediate steps. Each page p has outgoing/forward links and incoming/backlinks. Each backlink is a citation of the named page p; each outgoing link cites the page pointed by p. Google's original ranking algorithm named **PageRank** is a measure of global web-page importance not of an importance based on the search query results of a similarity search-engine.

1998 PageRank and Google Toolbar PageRank. With respect to that measure Google's rank is (naturally) very high, MIT's is a bit less, and NJIT scores a little less than both. Note that this `PageRank` algorithm that we will be discussing in the remainder of this Subject reflects Google's state in 1998. Google is currently using a slightly different "PageRank" measure and makes it available to Google users through what is known as Toolbar PageRank displayed in the Google Toolbar; this is a number between 0 and 10 (Google itself used to be a 10, MIT has a 9 rank, and NJIT an 8).

Idea behind browsing. The idea behind PageRank is simple. Consider the web to be a graph and do the following random walk on it by following the protocol described by the following two rules.

Rule 1. For $d \times 100\%$ (say 85%) of the time, a user follows (outgoing) links (uniformly) at random. (For example, if a web-page has 5 the user pick one at random.) Follow such an outgoing link and browse it.

Rule 2. For $(1 - d) \times 100\%$ of the time, a web-page might become a sink or a user might get bored browsing the current web-page. So instead of using again Rule 1 choose a random page (not pointed by the current page) to visit next. (The choice of a random page is one among the n web-pages of the whole Web.)

Parameter d is a number between 0 and 1. For example one might choose $d = 0.85$.
Fundamental Question. What is the probability that a page A will be visited?

Idea of ranking web-pages. If a page p is linked by **many** pages then p is important. If a page p is linked by **few but important pages** then p should also be important. The importance or not of a page is divided evenly and also inherited to the pages pointed by it. Suppose we have pages $T_1, \ldots, T_m$ pointing to page A. The PageRank $PR(A)$ of A is then defined by

$$PR(A) = (1-d) + d\,(PR(T_1)/C(T_1) + \ldots + PR(T_m)/C(T_m))\,, \tag{3}$$

where $PR(T_i)$ is the page rank of T_i and $C(T_i)$ is the number of outgoing links of T_i.

There are some limitations (matrix algebra-related) to this scheme however. A slight modification of the formula is required to guarantee convergence (same way as HITS).

Technical issues to resolve. Each page has its own PageRank (eg. $PR(A), PR(T_i)$). Each page is recording its number of outgoing links i.e. $C(T_i)$. The RageRank of a page T_i is evenly distributed among the pages pointed by T_i. Each such page get a rank contribution of $PR(T_i)/C(T_i)$. Even if a page has no citations (i.e. incoming links) it will still have a rank of $1-d$. (A choice of $d = 0.85$ would assign a small non-zero rank of $1-d = 0.15$ to such pages.) To be fair, to every other page, all pages get pumped up by $(1-d)$. The choice of d is important in controlling the rate of convergence in a PageRank computation.

The existence of the term $(1-d)$ cause the rank values to be arbitrary real numbers that might grow large. In order to avoid instability problems and keep ranks small (e.g. less that one) we need to modify the PageRank formula accordingly. Again, n denotes the total number of web-pages to be ranked.

Formulation of PageRank $PR(A)$ for A. The PageRank $PR(A)$ of web-page A is given by

$$PR(A) = (1-d)/n + d\,(PR(T_1)/C(T_1) + \ldots + PR(T_m)/C(T_m)) = \frac{1-d}{n} + d \cdot \sum_{i=1}^{m} \frac{PR(T_i)}{C(T_i)} \tag{4}$$

The normalized rank of Eq. (4) could be interpreted to signify probabilities. Thus one might interpret $PR(A)$ as the probability of visiting web-page A during the random walk obtained through Rule 1 and Rule 2.

Computing $PR(A)$

1. In order to compute $PR(A)$ we need to know $PR(T_i)$. To evaluate the rank $PR(A)$ of A one needs to know the ranks $PR(T_i)$ of all pages T_i pointing to A.

2. In order to obtain $PR(T_i)$ we need to know T_i. And this requires that we know in advance all the pages that point to A which is a little bit more elaborate than finding the pages pointed by A. The latter is easy : scan document A and locate all anchor fields to determine pages pointed by A. But finding T_i has not yet been resolved.

3. Even if we know T_i to find $PR(T_i)$ we need to find the pages pointing to T_i! Not only that but in order to find these ranks $PR(T_i)$ of T_i's one needs to know the ranks of the pages pointing to the T_i's.

4. Chicken and egg problem? What if we have a cycle or a very long back-chain?.

5. Matrix theory to the rescue. Matrix theory and the power method alluded in the HITS algorithm (see pages 4 and 5) can be used to show that using a simple iterative algorithm (that was also employed in a different form in HITS), PR(A) can be computed in a few iterations and the rank vector (of all n web-pages) computed corresponds to the principal eigenvector of the normalized (and weighed) link matrix of the web. The term `normalized/weighed` means that all sinks have been eliminated and the graph is used by employing the C matrix rather than the original L form and thus rank is expressed through Eq. 4.

(c) Copyright A. Gerbessiotis. All rights reserved.

1. Rank vector. If we have pages $A, B, C, \ldots$ we can find individually the PageRanks $PR(A), PR(B), PR(C), \ldots$ or we can collectively compute a PageRank vector PR which is then the vector $PR = (PR(A), PR(B), PR(C), \ldots,)$ of PageRanks of the n pages of the Web.

2. Rank vector initialization. In any rank computation we need to determine the initial values of $PR(.)$. Are they zero? One (similarly to HITS) ? Or something else?

3. Is $(1-d)/n$ vs $1-d$ really important? Does it really make a difference to use $(1-d)/n$ rather than $1-d$?

In the first example below we use formula 3 with two sets of initial conditions. We also use a very asynchronous approach in updating ranks (i.e. new ranks overwrite older values immediately). This is just for convenience only as shown by the following case.

Example 3. *A points B and B points to A ranks.* Two pages A and B each one pointing to the other. $C(A) = C(B) = 1$. Consider starting first with $PR(A) = PR(B) = 1$ and then $PR(A) = PR(B) = 0$. We use Eq. 3.

```
Case 1. PR(A)=PR(B)=1 [A, B updated in any order, synchronously or not]
  d=0.85       PR(A) = 0.15 + 0.85*PR(B) =1        , PR(B)= 0.15 + 0.85*PR(A) = 1
Case 2a. PR(A)=PR(B)=0 [A,B updated synchronously]
 Iteration 1.
  d=0.85       PR(A) = 0.15 + 0.85*PR(B) =0.15     , PR(B)= 0.15 + 0.85*PR(A) = 0.15
  and so on
Case 2b. PR(A)=PR(B)=0 [A updated, then B updated]
 Iteration 1.
  d=0.85       PR(A) = 0.15 + 0.85*PR(B) =0.15     , PR(B)= 0.15 + 0.85*0.15 = 0.2775
 Iteration 2.  PR(A) = 0.15 + 0.85*0.2775=0.385875 , PR(B)= 0.15 + 0.85*0.385875= 0.47799375
 Iteration 3.  PR(A) = 0.15 + 0.85*0.4779=0.556294 , PR(B)= 0.15 + 0.85*0.556294= 0.62285048
 Iteration 4.  PR(A) = 0.15 + 0.85*0.6228=0.679422 , PR(B)= 0.15 + 0.85*0.679422= 0.7275094
 Iteration 5.  PR(A) = 0.76 PR(B) = 0.80
 Iteration 6.  PR(A) = 0.83 PR(B) = 0.82    In practice 20-40 iterations are needed!
            ...
 Iteration 16. PR(A) = 0.99 PR(B) = 0.99
 Iteration 17. PR(A) = 0.99 PR(B) = 0.99 [Almost same as Case 1]
```

Figure 3: Example 3: Rank computations for two mutually pointing documents

Example 4

Example 4. This example also follows the asynchronous approach in computing PageRanks and initializes the ranks to zero and updates A then B then C then D and so on. Say page A points to B,C,D and B,C,D all point to A. Using the PageRank formulas through the computational pseudocode listed at the bottom, we get. Final PageRanks are not in $[0, 1]$ interval.

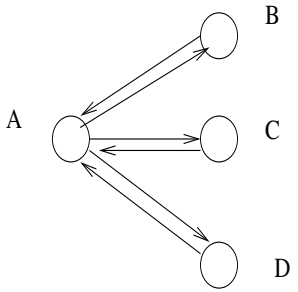


Figure 4: Simple Example 1

```
   PR(A)                  PR(B) = PR(C)= PR(D)
 .15000000000000000000    .19250000000000000000
 .64087500000000000000    .33158125000000000000
 .99553218750000000000    .43206745312500000000
1.25177200546875000000    .50466873488281250000
1.43690527395117187500    .55712316095283203125
1.57066406042972167968    .59502148378842114257
1.66730478366047391355    .62240302203713427550
                      .....
1.91382808120797485032    .68636717492691429826
1.91524078867276182933    .68840028388469558049
i.e. PR(A) ~ 1.92         PR(B)=PR(C)=PR(D) ~ 0.69

a=b=c=d=0;
for(i=1;i<=Iterations;i++){ //Formula used for this non-faithful PageRank computation
  a= 0.15/1+0.85*(b+c+d) // NOTE: This PSEUDOCODE DOES NOT UPDATE
  b= 0.15/1+0.85*(a/3.0) //       Rank values in "waves" but one at
  c= 0.15/1+0.85*(a/3.0) //       a time. It can be considered
  d= 0.15/1+0.85*(a/3.0) //       an ASYNCHRONOUS  approach.
 }
```

(c) Copyright A. Gerbessiotis. All rights reserved.

Example 5: Effects of collusion

Example 5. Say page A points to B and B points to C and C also points to A. Using again an asynchronous approach (A then B then C), with initial values 0, but both Eq. (3) and Eq. (4) we get.

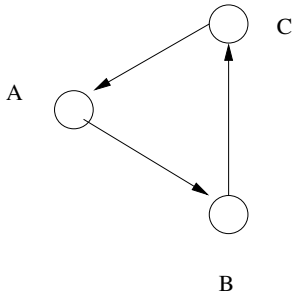


Figure 5: Collusion graph

	Initially all PR(.)= 0 Use Equation (4) : (1-d)/N			Initially all PR(.)=0 Use Equation (3) : 1-d			Initial Values 0 Asynchronous Approach
n	PR(A)	PR(B)	PR(C)	PR(A)	PR(B)	PR(C)	
0	0.050000	0.092500	0.128625	0.150000	0.277500	0.385875	
1	0.159331	0.185432	0.207617	0.477994	0.556295	0.622850	
2	0.226474	0.242503	0.256128	0.679423	0.727509	0.768383	
3	0.267709	0.277552	0.285919	0.803126	0.832657	0.857758	
	....			....			
25	0.333332	0.333332	0.333332	0.999996	0.999996	0.999997	
26	0.333332	0.333333	0.333333	0.999997	0.999998	0.999998	
27	0.333333	0.333333	0.333333	0.999998	0.999999	0.999999	
28	0.333333	0.333333	0.333333	0.999999	0.999999	0.999999	

If examples 3-5 tells us something, it will be that no matter what approach we follow the end-result seems to be the same if not in value but at least qualitatively. Of course the number of steps to obtain this result might vary.

(c) Copyright A. Gerbessiotis. All rights reserved.

The PageRank algorithm (final version). We present below algorithm `PageRank` that uses Eq. (4) to compute the rank of web-pages. The initialization vector is e^t/n i.e. a vector whose all entries are $1/n$. This might work slightly better than e^t (an all-one vector) and is always preferable to an all-zero vector.

For the algorithm to be efficient a proper representation of the Web-graph must be available. A compact form of it is through an adjacency list representation of the link structure i.e. for every vertex we maintain its out-degree and a list of end-points for outgoing edges as shown below

```
Source-vertex Out-degree  Endpoints-of-outgoing edges
       u        m        k1 k2 k3 ...  km
```

```
   PageRank(G,V,E)  //The code performs a SYNCHRONOUS Rank update
1. for all vertices u in V     /* Initialization Step */
2.    Src[u] = 1/n;
3. small = something-small;
4. while (convergence-distance > small) {
5.   for all v in V
6.     D[v]=0;
7.   for(i=0;i<|V|;i++) {
8.     Read-Adjacency-List(u,m,k1,k2,...,km);
9.     for(j=1;j<=m;j++)
10.      D[kj] = D[kj] + Src[u]/m
11.  }
12.  for all v in V
13.    D[v] = d * D[v] + (1-d)/n
14.  convergence-distance = ||Src-D|| /* Euclidean distance */
15.  Src=D;
16. }
```

Figure 6: PageRank

Practical issues

Practical Issue 1. What if we run out of space ? Split Vectors `Src` and `D` into B blocks.

Practical Issue 2. Split adjacency list (endpoint-list) into B blocks with the i-th block containing only edges with endpoints the set of vertices of the i-th block of `Src/D`. Run Algorithm above for i-th block, for all $i = 1, \ldots, B$.

Practical Issue 3. Parallelization of this scheme is possible if we assign each block or groups of blocks of size B/p to the p processors of a system. Details omitted.

The theory behind PageRank

In order to simplify notation, let us define the rank of page i as r_i. To simplify things we refer to the definitions of L, L^t, C, C^t given earlier. We also define $A = C^t$.

$$r_i = \sum_{(i,j)\in E} \frac{r_j}{d(j)} \tag{5}$$

$$\Longrightarrow \quad r = C^t r = Ar \Longrightarrow Ar = 1\, r \tag{6}$$

$$\Longrightarrow \quad r^t = r^t A^t = r^t C \tag{7}$$

The matrix related to Eq. (5) is thus A. From Eq. (6) we have that $Ar = 1r$ that r is the eigenvector of $A = C^t$ corresponding to eigenvalue 1. Why does A have eigenvalue 1? From a prior discussion we have established that C has eigenpair $(e, 1)$ i.e. the unit as an eigenvector for eigenvalue 1. Then $A = C^t$ has eigenpair $(r, 1)$.

The eigenvector r is the rank that needs to be computed from Eq. (5).

Iterative Convergence and number of iterations. How can we find r? We can use iterative methods such as the power method $r^{(i)} = Ar^{(i-1)}$. The iterative process stops when $\mid r^{(k)} - r \mid <$ `error`. The Google author claimed that $k = 50$ works for most cases.

Initial value.

What is r_0? Any vector $r^{(0)}$ is OK as long as it is orthogonal to the solution r. For example the unit vector e or e/n should work fine. A requirement for convergence is scaling. This would require the use of e/n, a modification of Eq. (5), and definite elimination of sinks.

Graph Connectivity issues

What if the graph used is not connected? The only problem that might arise is when the eigenvector r is not in the component we are looking for. If the graph is connected everything would work fine. Is the graph representing WWW connected? No and it also contains sinks. A combined solution that resolves sinks and such connectivity problem is the introduction of "random wanderer". If you reach a web-page with no outgoing links then choose from that point a (uniformly at) random web-page to proceed.

Alternatives to random wanderers.

Modify that last "choose a random page to visit" with a "choose randomly from a smaller subset of pages instead". What is the effect of having or allowing a `BACK` button? Google modeling does not like `BACK` buttons. Markov chains on which the theory of Google's PageRank is based is a memoryless model; a `BACK` button adds memory to the model.

(c) Copyright A. Gerbessiotis. All rights reserved.

Example. A1. Effect of sinks in Eq. (5) is depicted in the two examples of Figure 7.

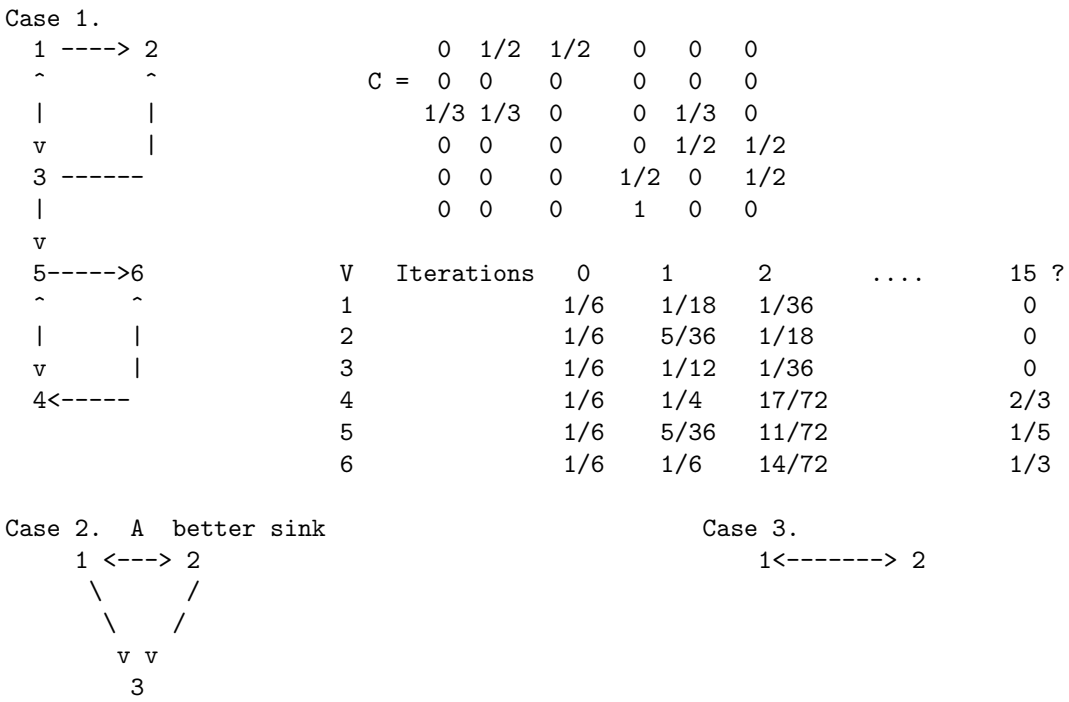


Figure 7: Sinks and their effects

`Rank sinks` such as the component $(4, 5, 6)$ in Case 1, or vertex 3 in Case 2 of Figure 7 cause problem with the ranks of vertices that contribute to the ranks of the vertices of the sink (eg. 1,2,3 in Case 1 or 1,2 in Case 2) but they get no benefit at all for that.

$$r_i = \frac{(1-d)}{n} + \sum_{(i,j)\in E} \frac{r_j}{d(j)}, \quad 0 < d < 1 \tag{8}$$

Rank sink problems can be avoided by modifying the rank formula according to Eq. (8). The choice of $d = 0.85$ was suggested by Page and Brin. If d is chosen to be very small then ranks will not be very sensitive to fluctuations of d. A d around 0.85 suffices for convergence within 50-70 iterations. The existence of d "makes the graph connected" and eliminateds eigenvalue related problems. Why are we using the eigenpair $(r, 1)$ in this analysis to determine the rank?

Linear algebra and $(r, 1)$ pair.

Eigenvalue 1 is the largest eigenvalue of non-negative matrix C (or A). Every other eigenvalue is smaller. Therefore $\lambda_1 = 1 > \lambda_2 > \ldots$. Let the corresponding eigenvectors be $x_1, \ldots, x_n$. From matrix theory we have that $Ax = \lambda x$ for eigenpair (x, λ). Therefore $A^2x = A(Ax) = A\lambda x = \lambda(Ax) = \lambda^2 x$. Let the starting vector $r^{(0)}$ be a linear combination of the eigenvectors i.e. $r^{(0)} = x_1 + a_1x_2 + \ldots + a_nx_n$. Then using the fact that $A^kx_i = \lambda_i^k x_i$ we obtain the following.

$$\begin{aligned} r^{(0)} &= x_1 + a_1x_2 + \ldots + a_nx_n \\ r^{(1)} = Ar^{(0)} &= Ax_1 + Aa_2x_2 + \ldots + Aa_nx_n \\ &= 1x_1 + \ldots + a_n\lambda_n x_n \\ r^{(k)} = Ar^{(k-1)} &= 1x_1 + a_2\lambda_2^k x_2 + \ldots + a_n\lambda_n^k x_n \end{aligned} \tag{9}$$

Therefore

$$r^{(k)} = x_1\left(1 + a_2(\lambda_2/1)^k x_2 + \ldots + a_n(\lambda_n/1)^k x_n\right) \tag{10}$$

From the latter we obtain that $r^{(k)}$ converges to x_1 as long as $\lambda_2^k, \ldots, \lambda_n^k \to 0$. Given that $\lambda_1 = 1 > \lambda_2 > \ldots > \lambda_n$ the rate of convergence depends on λ_2^k.

Resolving rank sink problems (continued)

What is the effect of Eq. (8)? Matrix theory shows that the modified rank formula in matrix form derives a matrix whose largest eigenvalue is still 1 yet the second largest eigenvalue is d smaller i.e. at most d.

Convergence and d.

Since $0.85^{50} = 0.00029576466\ldots$ it means that after 50 iteration we gain only 2-3 places of accuracy in rank calculations. In order to have accuracy of α digits we need to perform $-\alpha/\log_{10} 0.85$ iterations. Note that slight differences in rank are not very reflective of the actual (or relative) rank of two web-pages. Therefore not much will be gained if we increase the number of iterations.

Matrix Formulation.

We can rewrite Eq. (8) in matrix form as follows. Note that Eq. (5) in matrix form was written as $r = Ar$. G stands for `Google matrix`.

$$\begin{aligned} r_i &= \frac{(1-d)}{n} + \sum_{(i,j)\in E} \frac{r_j}{d(j)} \Longrightarrow \\ G &= \frac{1-d}{n} ee^t + dA \Longrightarrow \\ Gx &= \frac{1-d}{n} ee^t x + dAx \end{aligned} \tag{11}$$

An alternative definition is the one that takes into consideration the "random wanderer" choices of visiting a specific set of pages rather than a uniformly at random web-page among all the web-pages. This can lead to a replacement of the e^t vector by a v^t vector that reflects the choices of the wanderer. Then we have

$$\begin{aligned} G &= \frac{1-d}{n} ev^t + dA \Longrightarrow \\ Gx &= \frac{1-d}{n} ev^t x + dAx \end{aligned} \tag{12}$$

Some call vector v^t the `Google teleportation vector`.

Properties of the G matrix

Theorem. If matrix A has eigenvalues $\langle 1 \geq \lambda_2 \geq \ldots \geq \lambda_n \rangle$, then matrix G has eigenvalues $\langle 1 \geq d\lambda_2 \geq \ldots \geq d\lambda_n \rangle$.
Proof. Consider $G = ((1-d)/n)ee^t + dA$. For A, an eigenpair is $(e, 1)$ i.e. $Ae = e$.

Consider matrix Q formed by the eigenvectors of A listed so that e appears first, i.e. $Q = (e \quad X)$ then $Q^{-1} = \begin{pmatrix} z^t \\ Z^t \end{pmatrix}$.

Since $Q^{-1}Q = I$ we obtain that $Q^{-1}Q = \begin{pmatrix} z^te & z^tX \\ Z^te & Z^tX \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & I \end{pmatrix}$. From the last equality we conclude that

$$z^tX = 0, \quad z^te = 1 \quad , \quad Z^te = 0, \quad Z^tX = I \tag{13}$$

Consider now the product $Q^{-1}AQ$. We have

$$Q^{-1}AQ = \begin{pmatrix} z^t \\ Z^t \end{pmatrix} \cdot \begin{pmatrix} Ae & AX \end{pmatrix} = \begin{pmatrix} z^tAe & z^tAX \\ Z^tAe & Z^tAX \end{pmatrix} = \begin{pmatrix} 1 & z^tAX \\ 0 & Z^tAX \end{pmatrix} \tag{14}$$

The last equality was derived because $Ae = e$ implies $z^tAe = z^te = 1$ and the fact that $z^te = 1$ by Eq. (13). In addition $Z^tAe = 0$ by $Z^te = 0$ again from Eq. (13).

$$Q^{-1}GQ = Q^{-1}(dA + ((1-d)/n)ee^t)Q = dQ^{-1}AQ + (1-d)/nQ^{-1}ee^tQ \tag{15}$$

$$= \begin{pmatrix} d & dz^tAX \\ 0 & dZ^tAX \end{pmatrix} + \frac{(1-d)}{N} \cdot \begin{pmatrix} z^te \\ Z^te \end{pmatrix} \cdot \begin{pmatrix} e^te & e^tX \end{pmatrix}. \tag{16}$$

$$\tag{17}$$

(c) Copyright A. Gerbessiotis. All rights reserved.

The last two matrices are the expanded form of $Q^{-1}e$ and e^tQ respectively. Therefore we get after observing again from Eq. (13) that $z^te = 1$, $Z^te = 0$.

$$\begin{aligned} Q^{-1}GQ &= \begin{pmatrix} d & dz^tAX \\ 0 & dZ^tAX \end{pmatrix} + \frac{(1-d)}{N} \cdot \begin{pmatrix} z^te(e^te) & z^te(e^tX) \\ Z^te(e^te) & Z^te(e^tX) \end{pmatrix} \\ &= \begin{pmatrix} d & dz^tAX \\ 0 & dZ^tAX \end{pmatrix} + \frac{(1-d)}{N} \cdot \begin{pmatrix} (e^te) & (e^tX) \\ 0 & 0 \end{pmatrix} \\ &= \begin{pmatrix} d + ((1-d)/N) * N & dz^tAX + ((1-d)/N)e^tX \\ 0 & dZ^tAX \end{pmatrix} \\ &= \begin{pmatrix} 1 & dz^tAX + ((1-d)/N)e^tX \\ 0 & dZ^tAX \end{pmatrix} \end{aligned} \tag{18}$$

Since $Z^tX = I$ from Eq. (13), and from the rightmost side of Eq. (14) we have Z^tAX has eigenvalues the vector $(\lambda_2, \ldots, \lambda_n)$, and thus we can conclude that dZ^tAX will have as eigenvalues the d multiples of them, i.e. $(d\lambda_2, \ldots, d\lambda_n)$.

(c) Copyright A. Gerbessiotis. CS 667-101 : Spring 2012 . All rights reserved.