A Review on PageRank Algorithms

Mahsa Kazemi

Department of Computer Science, University of Western Ontario

Abstract

As the amount of information on the web grows exponentially, we need effective tools in order to extract relevant ones. This has introduced an active research area over the last few decades and powerful approaches have been proposed for treating relative problems. The aim of this review paper is to represent the pioneering ranking algorithms such as PageRank [4] and HITS algorithms [12], their advantages and drawbacks, as well as approaches for improving them. In particular, we address some of the existing computational issues and explain how to handle them.

Keywords: HITS, PageRank.

---

1 Introduction

The massive amount of information on the web and the vital need for retrieving relevant ones highlight the role of search engines. Google, the most popular search engine [14], relies on PageRank algorithm introduced by Sergey Brin and Lawrence Page in 1998 [4,17] with the aim of measuring the popularity of each hyperlinked document (web page). This algorithm has been extensively utilized in various areas, such as distributed computing [6], clustering [1], ranking sports teams [9, 13], and network alignment [18]. HITS, standing for Hypertext Induced Topic Search, was proposed by Jon Kleinberg in 1998 [12]. Although it uses the hyperlink structure (the same as PageRank algorithm) to assign scores to web pages, there exists some differences. PageRank algorithm creates one rank for each page whereas HITS computes two. PageRank algorithm is query-independent but HITS is query-dependent. The idea behind HITS follows two concepts *authority* and *hub*. The former refers to a page with many in-links and the latter is a page with many out-links. The rest of this paper is organized as follows

Email: mkazemin@uwo.ca.

2

2.1 PageRank Algorithm

The idea behind PageRank algorithm is to crawl internet pages and give each page a score based on its importance. Now we mention a number of drawbacks associated with this algorithm. First of all, the algorithm fails to represent up-to-date events which have been of utmost importance amongst users searching information through internet. One reason is that these scores are not computed at the query time as it is both expensive and slow. This, in turn, is done while indexing, a process in Google that looks for relevant pages containing keywords and/or phrases. Therefore, recently updated pages will not be regarded as authorities on a specific subject unless they have obtained exposure and paths from other pages. Because of this, Google News is devoted to news articles, or the result of Google News is listed at the top of the general search results. Next, this algorithm cannot respond to queries containing natural language and/or information rather than keywords. For instance, if a user searches "how many hours longer does a bear sleep than a human", the algorithm looks for "years", "hibernating animals" or "human" but may not understand the logical meaning of the question. Furthermore, it cannot recognize which keyword is more/less important to a user; see [19]. According to [20], ambiguous queries are answered by an AI system called RankBrain which is the third-most important *signal* in Google.

2.1.1 Classical Power Iteration Method

[23] Suppose that we have a web graph of N nods, where nodes and edges represent pages and hyperlinks, respectively.

Definition 2.1. Suppose that node i has d_i out-links. The column stochastic adjacency matrix M is defined as follows

$$M_{ji} = \begin{cases} \frac{1}{d_i} & \text{if node i points to node j} \\ 0 & \text{otherwise} \end{cases}$$

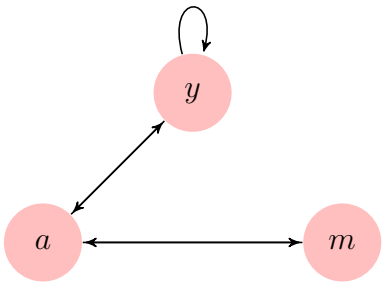
The power iteration method is given by the following steps:

- Start with the initial vector $r^{(0)} = [\frac{1}{N}, \ldots, \frac{1}{N}]$,
- Do iteration $r^{(t+1)} = Mr^{(t)}, \quad r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^t}{d_i}$, d_i is the set of edges pointing to node i,
- Stop when $\mid r^{(t+1)} - r^{(t)} \mid_1 < \epsilon$

Here, we can use any norm such as L_1 norm defined by $\mid \mathbf{x} \mid_{\mathbf{1}} = \sum_{\mathbf{1}}^{\mathbf{N}} \mid \mathbf{x_i} \mid$. Let us describe some observations about this algorithm. The first point is that, one matrix multiplication is required for each iteration which needs $O(N^2)$ computation. Sparsity is regarded as a useful property of matrix M. This arises from the fact that we are able to store merely non-zero entries of sparse matrices via

sparse storage schemes. In terms of matrix multiplication, we need O(number of non-zero entries) which is less than that of dense matrices.

Example 2.2. Consider the following 3-node web graph



with corresponding matrix M

$$M = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & 1 \\ 0 & \frac{1}{2} & 0 \end{bmatrix}$$

Applying the above method on the graph gives rise to iterations

$$r^{(0)} = \begin{bmatrix} \frac{1}{3} \\ \frac{1}{3} \\ \frac{1}{3} \end{bmatrix}, r^{(1)} = \begin{bmatrix} \frac{1}{3} \\ \frac{3}{6} \\ \frac{1}{6} \end{bmatrix}, r^{(2)} = \begin{bmatrix} \frac{5}{12} \\ \frac{1}{3} \\ \frac{3}{12} \end{bmatrix}, \ldots$$

for $\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix}$ which finally converges to $\begin{bmatrix} \frac{6}{15} \\ \frac{6}{15} \\ \frac{3}{15} \end{bmatrix}$.

2.1.2 Random Walk Interpretation

Now we describe an interpretation of page rank scores that are equivalent to probability distribution of a random walker in a graph. For a random walker assume that:

- At time t, the random walker is on page i
- At time $t+1$, the random walker moves along an out-link from i uniformly at random
- The random walker reaches page j linked from i
- Process repeats indefinitely

Furthermore,

- The probability that the random walker is on page i at time t is represented by the i-th coordinate of the vector $p(t)$

- The vector $p(t)$ is a probability distribution over pages

Now the question is where the random walker is going to be at time $t+1$. The answer follows

$$p(t+1) = Mp(t) \tag{2.1}$$

Definition 2.3. A probability distribution $p(t)$ is called *stationary distribution* if

$$p(t+1) = Mp(t) = p(t).$$

Remark 2.4. The rank vector r is a stationary distribution for the random walk process.

The fundamental theorem of Markov chains [10] states that under certain conditions, the stationary distribution is unique independent of the initial probability distribution.

2.1.3 Two issues with PageRank algorithm

In this section we first illustrate issues concerning page rank algorithm and then propose ideas on how to handle them. For more information see [4, 15, 24, 25].

The dead end problem. A node is called dead end when it has no out links. This property triggers equal-to-zero PageRank for other nodes vsiting a dead end one.

Solution: Always teleport. Now we go back to the 3-node graph

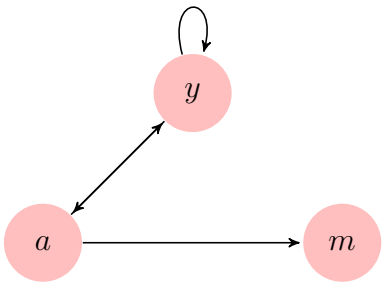


Figure 1: A graph with a dead end node.

The structure of matrix

$$M = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 \end{bmatrix}.$$

shows that this is not stochastic. Applying power iteration method results in

$$r^{(0)} = \begin{bmatrix} \frac{1}{3} \\ \frac{1}{3} \\ \frac{1}{3} \end{bmatrix}, r^{(1)} = \begin{bmatrix} \frac{2}{6} \\ \frac{1}{6} \\ \frac{1}{6} \end{bmatrix}, r^{(2)} = \begin{bmatrix} \frac{3}{12} \\ \frac{2}{12} \\ \frac{1}{12} \end{bmatrix}, r^{(3)} = \begin{bmatrix} \frac{5}{24} \\ \frac{3}{24} \\ \frac{2}{24} \end{bmatrix}, \dots$$

for $\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix}$ which finally converges to $\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$. The idea is to teleport with probability 1 as soon as we reach a node that has no out-links. To be more precise, when a random walker visits node m, it jumps out and lands in any other node with probability $\frac{1}{3}$.

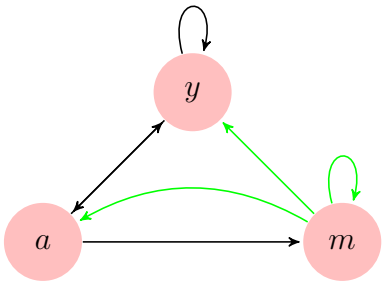


Figure 2: The issue of dead end is resolved.

As a result, the matrix M is modified as follows

$$M_{modified} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & 0 & \frac{1}{3} \\ 0 & \frac{1}{2} & \frac{1}{3} \end{bmatrix}.$$

A non-stochastic matrix M is transferred to the stochastic matrix A through the following equality

$$A = M + a^T(\frac{1}{N}e)$$

where the i-th entry of vector a is defined as follows

$$\begin{cases} 1 & \text{if node i has no out-links} \\ 0 & \text{otherwise} \end{cases}$$

The spider trap problem. A spider trap refers to a set of nodes that all have out-links but are not connected to any other nodes. This happens on the web and causes all the computed PageRank to store in the spider trap.

Solution: Random teleports. No we explain how to handle the issue of spider trap through the following 3-node graph

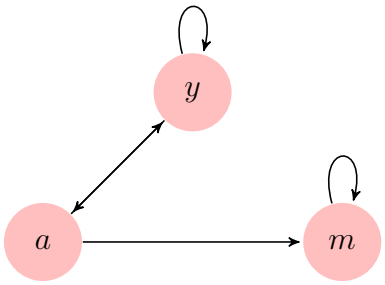


Figure 3: A graph with a one-node spider trap.

when a random walker arrives in m, it gets trapped because m has no out-link. Thus m as a spider trap. The stochastic matrix is given by

$$M = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 1 \end{bmatrix}.$$

Power iteration method gives

$$r^{(0)} = \begin{bmatrix} \frac{1}{3} \\ \frac{1}{3} \\ \frac{1}{3} \end{bmatrix}, r^{(1)} = \begin{bmatrix} \frac{2}{6} \\ \frac{1}{6} \\ \frac{3}{6} \end{bmatrix}, r^{(2)} = \begin{bmatrix} \frac{3}{12} \\ \frac{2}{12} \\ \frac{7}{12} \end{bmatrix}, r^{(3)} = \begin{bmatrix} \frac{5}{24} \\ \frac{3}{24} \\ \frac{16}{24} \end{bmatrix}, \ldots$$

for $\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix}$ which finally converges to $\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$. As can be seen, the highest importance is assigned to node m meaning that as soon as the random walker moves along the link toward m it will be stucked there. Suppose that β is a parameter with $0.8 < \beta < 0.9$. Google gives the random walker two choices at each time step

- Follow a link at random with probability β
- Jump to a random page with probability $1 - \beta$

This guarantees that the random walker will teleport out of the spider trap within a few time steps. Hence, the corresponding PageRank equation [4] is

$$r_j = \sum_{i \to j} \beta \frac{r_i}{d_i} + (1 - \beta) \frac{1}{N}$$

and the Google matrix A is given by

$$A = \beta M + (1 - \beta) \frac{1}{N} e \cdot e^T \tag{2.2}$$

where e is a column vector of all 1's. The vector $\frac{1}{N}e$ is called the *personalization* vector. Google uses $\beta = 0.85$ [17] and the uniform vector for v [14].

Remark 2.5. The PageRank solution is

$$r^{(t+1)} = A \cdot r^{(t)}$$

where A is the Google matrix.

We continue the above example and explain how to deal with the issue of spider trap using the proposed approach.

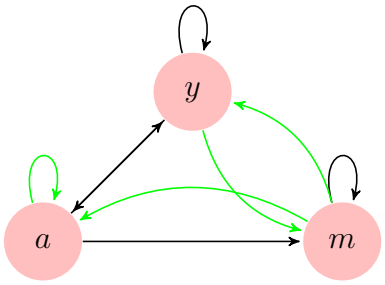


Figure 4: The issue of spider trap is resolved.

The Google matrix (2.2), for $N = 3$ and matrix M, is

$$A = 0.8 \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 1 \end{bmatrix} + 0.2 \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{bmatrix}$$

Applying power iteration method yields

$$r^{(0)} = \begin{bmatrix} \frac{1}{3} \\ \frac{1}{3} \\ \frac{1}{3} \end{bmatrix}, r^{(1)} = \begin{bmatrix} 0.33 \\ 0.20 \\ 0.46 \end{bmatrix}, r^{(2)} = \begin{bmatrix} 0.24 \\ 0.20 \\ 0.52 \end{bmatrix}, r^{(3)} = \begin{bmatrix} 0.26 \\ 0.18 \\ 0.56 \end{bmatrix}, \ldots$$

for $\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix}$ which finally converges to $\begin{bmatrix} \frac{7}{33} \\ \frac{5}{33} \\ \frac{21}{33} \end{bmatrix}$. Note that the node m has still received the highest PageRank. The impact, however, has been diminished and other nodes get some of the PageRank.

3 Computation of the PageRank for the web-scale graphs: improvements of PageRank algorithm

A naive approach. To be completed soon! [2, 26]

3.1 Block structure method

To accelerate the computation of PageRank algorithm, a group of researchers from Stanford University proposed an algorithm called `BlockRank` algorithm [2, 11, 14]. Consider a web graph (where web pages are denoted by nodes) and look at this as a host graph (where nodes describe hosts). Web pages like `www.princeton.edu` containing lots of pages are called hosts. There are many links between pages in a host (intralinks) but this is decreased when it comes to links between different hosts (interhost links). The idea behind `BlockRank` algorithm is to first compute *local PageRank* vectors, the PageRank vector assigned to pages in a host. Note that we utilize intralinks and ignore interhost links. Therefore, the `PageRank` algorithm is applied on each host containing less than a few thousands pages. This process produces the $1\times \mid H \mid$ `HostRank` vector (when $\mid H \mid$ represents the number of hosts) and $\mid H \mid$ local PageRank vectors of size $1\times \mid H_i \mid$, where the number of pages in host H_i is denoted by $\mid H_i \mid$. The next step, called expansion step, multiplies the local PageRank vector for host $\mid H_i \mid$ by the i-th coordinate of HostRank vector, which is the probability of being in that host. This algorithm estimate the global PageRank algorithm because at each step we ignore some links. This estimation is improved through repeating the expanding step untill it converges.

In [11] authors redefined the personalization vector v to be nonuniform and employed the following algorithm for computing PageRank.

Algorithm 1 Compute the PageRank for a graph G, initial vector $r^{(0)}$ and personalization vector v.

procedure PAGERANK$(G, r^{(0)}, v)$
  Construct M from G following Definition 2.1
  repeat
    $r^{(t+1)} = 0.85Mr^{(t)}$
    $w = \mid r^{(t)} - r^{(t+1)} \mid_1$
    $r^{(t+1)} = r^{(t+1)} + wv$
    $\delta = \mid r^{(t+1)} - r^{(t)} \mid_1$
  until $\delta < \epsilon$
  return $r^{(t+1)}$
end procedure

Notations. We use lower (i.e., i, j) and upper case (i.e., I, J) indices in order to represent hosts and blocks, respectively. We further use the notation $i \in I$ to state that page i belongs to block I. The number of elements in block J is given by N_J. We correspond a graph to a given block J which is a $N_J \times N_J$ submatrix of the matrix G and denoted by G_{JJ}.

The suggested approach on how to derive the BlockRank is summarized through the following steps

- Decompose the web graph (based on domain) into blocks
- For each block, compute the PageRank; see 1
- Approximate the BlockRank of each block
- In each block, weight the local PageRank by the BlockRank of that block. Next, aggregate the weighted local PageRanks to estimate global PageRank z
- Use z as a starting vector for `pageRank` algorithm 1

Local PageRanks. The *local PageRank vector* l_J of a block J (G_{JJ}) is the output of PageRank algorithm on block J. Thus,

$$l_J = \texttt{pageRank}(G_{JJ}, r_J, v_J) \tag{3.1}$$

where r_J is the initial vector $[\frac{1}{N_J}]_{N_J \times 1}$. The entries of $N_J \times 1$ personalization vector v_J are all zero except that the one corresponding to the root node of block J is 1.

Approximating the BlockRank of each block. Suppose that the web graph consists of k blocks. We build the *block graph* B whose vertices are blocks. An edge linking two pages in the web is regarded as an edge between corresponding blocks (a self-edge means both pages are in the same block). The sum of the edge-weights from pages in block I to pages in block J that are weighted by the local PageRanks of the connecting pages in block I gives the edge weight between block I and block J. For the web graph A and the local PageRank of page i in block I, the edge weight B_{IJ} (between two blocks I and J) is defined by

$$B_{IJ} = \sum_{i \in I,\ j \in J} A_{i,j} l_i$$

The local PageRank vectors l_J form the local PageRank matrix L as follows

$$L = \begin{bmatrix} l_1 & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & l_2 & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & l_k \end{bmatrix}$$

Setting $l_i = 1, i = 1, \dots, k$ in L gives a new matrix say S. We define the block matrix B to be

$$B = L^t A S \tag{3.2}$$

Note that B is a transition matrix and B_{IJ} refers to the transition probability of block I to block J. Now we can compute the BlockRank vector b through

$$b = \texttt{pageRank}(B, v_k, v_k) \tag{3.3}$$

where v_k is the $k \times 1$ matrix of all $\frac{1}{k}$'s.

The BlockRank algorithm and its advantages. A complete description of the BlockRank algorithm is given as follows

1. For each block J, compute the local PageRank vector l_J; see (3.1)
2. Following (3.2) and (3.3), construct block transition matrix B and find BlockRanks b
3. Weight the local PageRanks of pages in block J (matrix L) by the BlockRank of J (b) in order to obtain an approximation $r^{(0)}$ to the global PageRank vector r
4. Use $r^{(0)}$ as an initial vector in Algorithm 1

Authors in [11] has compared their approach against the standard (classical) PageRank algorithm and addressed four main advantages

- The speedup of BlockRank algorithm is attributed to caching effects. Block graphs and BlockRank vectors fit in the main memory and CPU cache, respectively. Less disk i/o is devoted to local PageRank iteration since the entire graph does not fit in the main memory. Relying on the sorted link structure, they enhance the memory access patterns to the rank vector
- Only a few iterations are required for PageRank vectors computations to terminate as they converge fast. This hilights the role of PageRank vectors computation in that they allocate more computations on slowly converging blocks and less on faster ones. In contrast to standard PageRank algorithm, the iterations are applied to the whole graph causing the slowest blocks to generate convergence bottleneck
- The local PageRank algorithm (item 1 of the BlockRank algorithm) can be performed in parallel. Furthermore, it is feasible to reuse the computation of this algorithm during the future applications of BlockRank algorithm.

3.1.1 Why teleports solve the problem?

Let X be a set of states and P be the transition matrix with $P_{ij} = P(X_t = i | X_{t-1} = j)$

3.2 Weighted Page Rank Algorithm

3.3 HITS Algorithm

Suppose that page i has an authority score x_i and a hub score y_i. We denote the set of all directed edges in a graph by E. Let e_{ij} describe the directed edge from node i to node j. Assume that

each page has initial authority and hub scores x_i and y_i, respectively. These scores are successively refined through

$$x_i^{(k)} = \sum_{j:e_{ji}\in E} y_j^{(k-1)} \tag{3.4}$$

$$y_i^{(k)} = \sum_{j:e_{ij}\in E} x_j^{(k)} \tag{3.5}$$

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

The equations (3.4) can be expressed in terms of matrices as follows

$$x^{(k)} = L^t y^{(k-1)} \quad y^{(k)} = Lx^{(k)} \tag{3.6}$$

Here $x^{(k)}$ and $y^{(k)}$ are approximations for authority and hub scores at each iteration while matrix L is given by

$$L_{ij} = \begin{cases} 1 & \text{if node i points to node j} \\ 0 & \text{otherwise} \end{cases}$$

The HITS algorithm is given by

1. For a column vector m of all 1's let $y^{(0)} = m$. Other positive vectors can be used
2. Repeat the following until $x^{(k)}$ and $y^{(k)}$ converge

$$\begin{aligned} x^{(k)} &= L^t y^{(k-1)} \\ y^{(k)} &= Lx^{(k)} \\ k &= k+1 \\ \text{Normalize} \quad x^{(k)} &\text{and} \quad y^{(k)} \end{aligned}$$

See [14, Page 116]

Remark 3.1. We obtain equivalent version of Equations (3.6) that is

$$x^{(k)} = L^t L x^{(k-1)} \quad y^{(k)} = LL^t y^{(k-1)} \tag{3.7}$$

through

$$x^{(k)} \to L^t y^{(k-1)} \quad y^{(k-1)} \to Lx^{(k-1)}$$

in the second and first Equations, respectively. Deriving the *dominant eigenvector* () for L^tL and LL^t is done via Equations (3.7). This process is called *iterative power method*. Indeed, the dominant eigenvectors of L^tL and LL^t give the authority and hub vectors, respectively. Compared to PageRank power method, two approaches are the same except that in (3.7) the coefficient matrices are different from that of PageRank algorithm. Note that L^tL is called the *authority matrix* and LL^t is called the *hub matrix*.

How HITS works in practice [14, Page 117]. HITS algorithm revolves around two main stages. The first stage focuses on creating a *neighborhood graph*, denoted by NG, for a query search. The next stage is to compute the authority and hub scores for each page in NG and return two lists of ranks associated with pages the of highest authority and hub. In the following, we explain how to detect pages in NG carrying references to the query. One way of treating this is through a method called *inverted file index*. As an example consider

- term 1 (aardvark):3, 117, 3961
  ⋮
- term 10 (aztec): 3, 15, 19, 101, 673, 1199
  ⋮

The above numbers represent pages containing each term. When a query carrying term 1 and term 10 is searched, all numbers (pages) are moved into NG. Then, we look at node (pages) pointing to/from nodes in NG in order to expand a graph of a subset of NG. This type of expansion may fix the issue of synonym that is when a query containing the term "car" is searched , pages carrying "automobile" might be moved into NG. As this process might increase the size of NG, an upper bound of 100 to the number of in-links and out-links that a node can experience in NG has been assigned. Corresponding to each NG, the adjacency matrix L is built which is of smaller dimension compared to the size of the whole graph. The authority and hub vectors are computed using the dominant eigenvectors of L^tL and LL^t that requires small computational burdon. Note that the computational cost can be still reduced. To be more precise, computing dominant eigenvectors of either L^tL or LL^t is enough. If we compute the dominant eigenvector of L^tL (which is the authority vector x), through the equations $y = Lx$ the hub vector y is obtained.

Lemma 3.2 (See also [8, 12])**.** *Suppose that M is a symmetric square matrix and v is a vector not orthogonal to the principal eigenvector of M, $w_1(M)$. By increasing k without bound, The unit vector in the direction of M^kv converges to $w_1(M)$.*

Corollary 3.3 (See also [8,12])**.** *The principal eigenvector of a matrix M with non-negative entries has only non-negative entries.*

Lemma 3.4. *Two matrices LL^t and L^tL have similar eigenvalue. Moreover, L^tx is an eigenvector for L^tL if x is an eigenvector for LL^t.*

Theorem 3.5. *The sequences $x^{(1)}, x^{(2)}, x^{(3)}, \ldots$ and $y^{(0)}, y^{(2)}, y^{(3)}, \ldots$ converge.*

Proof. After k iteration in step 2 of the algorithm, $x^{(k)}$ is a unit vector in the direction of $(L^tL)^{k-1}L^tm$ and $y^{(k)}$ is a unit vector in the direction of $(LL^t)^km$. Note that both LL^t and L^tL are symmetric square matrix. Since vector m is not orthogonal to $w_1(LL^t)$, $y^{(k)}$ converges to $w_1(LL^t)$. Similar reasoning together with Lemma 3.4 prove that $x^{(k)}$ converges to $w_1(L^tL)$. □

Definition 3.6. We call a square matrix B reducible if for a permutation matrix Q we derive the following upper-triangular matrix

$$Q^t B Q = \begin{bmatrix} X & Y \\ 0 & Z \end{bmatrix}$$

where X and Z are square matrices. Otherwise, it is called irreducible.

Nonunique solutions.

According to [14, Page 120], different initial vectors can lead to different authority and hub vectors. This is justified by the example from [7]. Let

$$L = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix}, \qquad L^t L = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Starting with $x^{(0)} = (\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4})$ the authority vector converges to $x = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3}, 0)$. But for $x^{(0)} = (\frac{1}{4}, \frac{1}{8}, \frac{1}{8}, \frac{1}{2})$, we get $x = (\frac{1}{2}, \frac{1}{4}, \frac{1}{4}, 0)$. The Perron Frobenius Theorem [16, Page 673] states if a matrix is irreducible and non-negative, it contains a unique normalized positive dominant eigenvector which is called *Perron vector*. Hence, different results is due to this fact that $L^t L$ is reducible. This is handled via a modified version of the authority and hub matrices forcing them to be irreducible; see [**?**].

Example 3.7. Now we consider the following example to clarify the algorithm. First query terms should be introduced to the algorithm. This can be done through different schemes; see [14, Page 120] for more information. Suppose that we look at a subset of nodes maintaining the query terms (nodes 1 and 6). Assume that Figure 3.7 represents the corresponding neighborhood graph NG. The adjacency matrix L, the authority matrix $L^t L$, and the hub matrix $L L^t$ are

$$L = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}, \quad L^t L = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad L L^t = \begin{bmatrix} 2 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 2 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

The authority and hub vectors are

$$\begin{aligned} x &= (0, 0, 0.3660, 0.1340, 0.5, 0) \\ y &= (0.3660, 0, 0.2113, 0, 0.2113, 0.2113) \end{aligned}$$

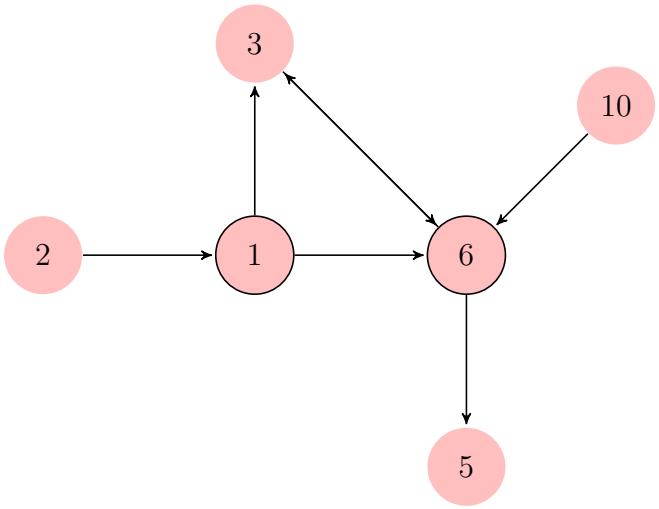


Figure 5: Neighborhood graph NG for nodes 1 and 6.

There are two issues (the occurrence of zero values and identical values) with respect to these vectors that can be fixed via several strategies [14, Pages 121-122]. After applying a strategy the authority and hub scores are sorted (in decreasing order) and their corresponding pages are given as

$$\begin{aligned} \text{authority ranking} &= (6, 3, 5, 1, 2, 10) \\ \text{hub ranking} &= (1, 3, 6, 10, 2, 5) \end{aligned}$$

From the first and second vectors, it is concluded that page 6 and page 1 are the most important authority and hub pages.

Strengths and Weaknesses of HITS. HITS algorithm provides users with two ranked lists: one associated with authoritative documents of the query and another with hubby documents. This can benefit users as they either go deep into a research query or do a broad research; therefore, authoritative or hub pages are needed, respectively. Another advantage is that, HITS reduces the problem into finding dominant eigenvectors of small matrices [14]. Compared to the number of pages on the web, size of these matrices is small.

The query-dependent property of HITS algorithm results in an expensive query time. According to [14], when a user searches a query, a neighborhood graph must be produced and atleast one matrix eignevector problem handled. This process is required for each query. This algorithm is susceptible to spamming. Generally, authority and hub scores can be affected if we increase a web page's inlinks and/or out-links. These changes might cause the ranked list to give a higher position to this web page. This is important since users prefer to find the result on top of the ranked list. without a doubt, adding out-links (influencing hubs) from the web page is simpler than in-links (influencing authorities). Following Equations 3.4 hubs and authorities are computationally dependent to each other. And since the neighborhood graph is smaller than the whole web, small changes to the link

structure may lead to a huge impac. This issue of link spamming is, however, controlled in [3] relying on L_1 normalization step. HITS algorithm suffers from the problem of topic drift. In fact, an authoritative page might be connected to a page carrying the query terms. This happens when a neighborhood graph is constructed during the query search process. As a result, a wrong page receives too much weight and dominate a ranked list. Again in [3], authors propose an approach to fix this issue.

Query-independent HITS. The impact of link spamming can be mitigated via compelling HITS algorithm to be query-independent. To this end, we compute a global authority and a global hub vector. Convergence to unique vectors for both authority and hub vectors is achieved via the following algorithm; see [14, Page 124]

1. Let $x^{(0)} = \frac{m}{N}$
2. Repeat the following $(0 < \zeta < 1)$ until $x^{(k)}$ and $y^{(k)}$ converge

$$\begin{aligned} x^{(k)} &= \zeta L^{(t)} L x^{(k-1)} + \frac{1-\zeta}{N} m \qquad (3.8) \\ x^{(k)} &= \frac{x^{(k)}}{|x^{(k)}|_1} \\ y^{(k)} &= \zeta L L^{(t)} y^{(k-1)} + \frac{1-\zeta}{N} m \qquad (3.9) \\ y^{(k)} &= \frac{y^{(k)}}{|y^{(k)}|_1} \\ k &= k+1 \end{aligned}$$

References

[1] R. Andersen, F. Chung, and K. Lang. Local graph partitioning using PageRank vectors. *In Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, 2006.

[2] P. Berkhin. A survey on PageRank computing. *Internet Mathematics*, 2:73–120, 2005.

[3] K. Bharat and M. R. Henzinger. Improved algorithms for topic distillation in hyperlinked environments. *In 21st International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, 104–111, 1998.

[4] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30:107–117, 1998.

[5] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Weiner. Graph structure in the web. *Computer Networks*, 33:309–320, 2000.

[6] M. Cook. Calculation of pagerank over a peer-to-peer network. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.123.9069, 2004.

[7] A. Farahat, T. Lofaro, J. C. Miller, G. Rae, and L. A. Ward. Authority rankings from HITS, PageRank, and SALSA: Existence, uniqueness, and effect of initialization. *SIAM Journal on Scientific Computing*, 27:1181–201, 2006.

[8] G. H. Golub and C. F. Van Loan. Matrix Computations Johns Hopkins University Press, 1989.

[9] A. Y. Govan, C. D. Meyer, and R. Albright. Generalizing Google PageRank to rank national football leagueteams. *In SAS Global Forum*, 2008.

[10] G. Grimmet and D. Stirzaker. Probability and random processes. Cambridge University Press, 2014.

[11] S. D. Kamvar, T. H. Haveliwala, C. D. Manning, and G. H. Golub. Exploiting the block structure of the web for computing PageRank. *Technical Report*, Stanford University, 2003.

[12] J.M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46:604–632, 1999.

[13] A. N. Langville. Sports ranking, 2009. From a draft of a forthcoming book.

[14] A. N. Langville and C. D. Meyer. Google PageRank and beyond: The science of search engine rankings. Princeton University Press, 2011.

[15] J. Leskovec, A. Rajaraman, and J. D. Ullman. Mining of massive datasets. Cambridge University Press, 2014.

[16] C. D. Meyer. Matrix Analysis and Applied Linear Algebra. SIAM, Philadelphia, 2000.

[17] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. *Stanford Digital Libraries Working Paper*, 1998.

[18] R. Singh, J. Xu, and B. Berger. Pairwise global alignment of protein interaction networks by matching neighborhood topology. *In Proceedings of the 11th Annual International Conference on Research in Computational Molecular Biology (RECOMB)*, 16–31, 2007

[19] https://blogs.cornell.edu/info2040/2015/10/28/limitations-of-pagerank/

[20] https://www.theverge.com/2015/10/26/9614836/google-search-ai-rankbrain

[21] https://blogs.cornell.edu/info2040/2017/10/19/fast-distributed-pagerank-computation/

[22] http://snap.stanford.edu/class/cs246-2012/slides/10-hits.pdf

[23] https://www.youtube.com/watch?v=VpiyOxiVmCg

[24] https://www.youtube.com/watch?v=ytjf6zYDd4s

[25] https://www.youtube.com/watch?v=UZePPh340sU

[26] https://www.youtube.com/watch?v=E9aoTVmQvok