3 How does Google rank webpages?

3.1 A Short Answer

Now we turn to the other links you see on a search result webpage, not the ads or sponsored search results, but the actual ranking of webpages by search engines such as Google. We will see that Google solves a very big linear equation to rank the webpages each time you search on `www.google.com`.

More important webpages should be ranked higher. But how do you quantify *how* imporant a webpage is? Well, if there are many other important webpages pointing towards a webpage A, probably A is important. This argument implicitly assumes two ideas:

- Webpages form a network, where a webpage is a node and a hyperlink is a directed link in the network.
- We can turn the seemingly circular logic of "important webpages pointed to you means you are important" into a set of equations that characterize the *equilibrium* of this *recursive definition* of "importance".

Suppose there are N webpages. Each webpage i has O_i number of **outgoing links** and I_i number of **incoming links**. The "importance score" of each webpage is π_i. This importance score is evenly spread across all the outgoing links, each of the outgoing neighbors receiving π_i/O_i importance score. Therefore, each node's importance score can also be written as the sum of the importance scores received from all the incoming neighbors. If this sum is indeed also π_i, we have *consistency* of the scores. But it is not clear if we can readily compute these scores, or if they exist in the first place.

It turns out that, with a couple of modifications to the process above, there is always a unique set of consistent scores, denoted as $\{\pi_i^*\}$, and those scores determine the ranking of the webpages: higher the score, higher is the webpage ranked.

For example, consider a very small graph with just four webpages and six hyperlinks, shown in Figure 3.1. This is a directed graph where each node is a webpage and each link a hyperlink. A consistent set of importance scores turns out to be: [0.125, 0.125, 0.375, 0.375]: webpages 3 and 4 are more important than webpages 1 and 2. In this toy example, it so happens that webpages 3 and 4, linking each other, push both webpages' rankings higher.

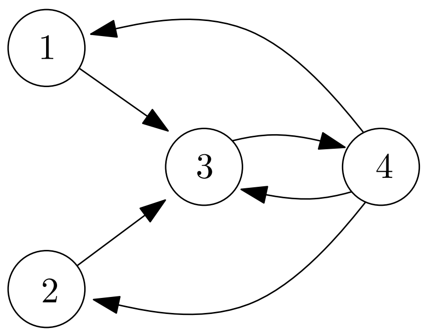

Figure 3.1 A simple example of importance score with 4 webpages and 6 hyperlinks. It is small with much symmetry, leading to a simple calculation of importance scores of the nodes.

Intuitively, the scores make sense. First, by symmetry of the graph, webpages 1 and 2 should have the same importance score. We can view webpages 3 and 4 as if they form one webpage first, a supernode 3+4. Since node 3+4 has two incoming links, and each of nodes 1 and 2 only one incoming link, node 3+4 should have higher importance score. Since node 3 points to node 4 and vice versa, these two nodes' importance scores mix into an equal division at equilibrium. This line of reasoning qualitatively explains the actual scores we see.

But how do we calculate the exact scores? In this small example, it boils down to two simple linear equations. Let the score for node 1 (and 2) be x, and that for node 3 (and 4) be y. Looking at node 1's incoming links. There is only one, and that comes from node 4, which points to three nodes, so we know $x = y/3$. Since $2x + 2y = 1$, we have $x = 0.125$ and $y = 0.375$.

Now how do we compute this set of consistent scores in a large, sparse, general graph of hyperlink connectivity?

3.2 A Long Answer

In any search engine, there are two main activities going on continuously behind the scene: (a) crawling the hyperlinked web space to get the webpage information, (b) indexing this information into concise representations and storing the indices.

When you search in Google, it triggers a ranking procedure that takes into account two main ingredients:

- How relevant is the content on each webpage, the **relevance score**.
- How important is the webpage, the **importance score**.

It is the composite score of these two factors that determine the ranking. We focus on the importance score ranking, since that usually determines the order of the top few webpages in any reasonably popular search, with its tremendous impact on how people obtain information and how online businesses generate traffic.

We will be constructing several related matrices: $\mathbf{H}, \hat{\mathbf{H}}$, and $\mathbf{G}$, step by step. Eventually we will be computing an eigenvector of $\mathbf{G}$ as the importance score vector. Each matrix is $N \times N$, where N is the number of webpages. These are extremely large matrices, considering that there are about $N = 40$ billion webpages out there in 2011.

The first matrix we define is $\mathbf{H}$: its (i, j)th entry is $1/O_i$ if there's a hyperlink from webpage i to webpage j, and 0 otherwise. This matrix describes the network topology: which webpage points to which webpage. It also evenly spread the importance of each webpage among its outgoing neighbors, the webpages that it points to.

Let π be an $N \times 1$ column vector denoting the importance scores of the N webpages. If we start with guessing that the consistent score vector is $\mathbf{1}$: simply a vector of 1s as each webpage is equally important, we have an initial vector $\pi[0]$, where 0 denotes the 0th iteration, *i.e.*, the initial condition.

Then keep multiplying π^T (the vector flipped to be a row vector) on the right by matrix $\mathbf{H}$. You can write out this matrix multiplication, and see this is spreading the importance score from the last iteration evenly among the outgoing links, and re-calculating the importance score of each webpage in this iteration by summing up the importance score from incoming links. For example, $\pi_1[2]$ (for webpage 1 in the second iteration) can be expressed as the following sum of importance scores from the first iteration:

$$\pi_1[2] = \sum_{j \text{ linked to } 1} \frac{\pi_j[1]}{O_j},$$

i.e., the π vector from the previous iteration inner-producting the first column of $\mathbf{H}$.

If we index the iterations by k, the update at each iteration is simply:

$$\pi^T[k] = \pi^T[k-1]\mathbf{H}. \tag{3.1}$$

We followed the (visually a little clumsy) convention in this research field that defined $\mathbf{H}$ such that the update is a mutiplication of row vector π^T by $\mathbf{H}$ from the right. We could also normalize the resulting π vector so that its entries add up to 1.

Does the iterations in (3.1) converge, *i.e.*, is there a k sufficient large such that $\pi[k]$ vector is arbitrarily close to $\pi[k-1]$ (no matter what is the initial guess $\pi[0]$)? If so, we have a way to compute consistent score vector as accurately as we want.

But the answer is "not quite yet". We need two adjustments to $\mathbf{H}$.

First, some webpages do not point to any other webpages. These are "dangling nodes" in the hyperlink graph . For example, in Figure 3.2, node 4 is a dangling node, and its row is all 0s in the $\mathbf{H}$ matrix:

$$\mathbf{H} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1/3 & 0 & 1/3 & 1/3 \\ 1/3 & 1/3 & 0 & 1/3 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

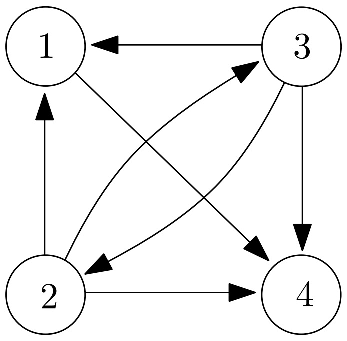

Figure 3.2 A network of hyperlinked webpages with a dangling node 4.

and there is no consistent scores. To see this, write out the system of linear equations $\pi = \pi\mathbf{H}$:

$$\begin{cases} \frac{1}{3}(\pi_2 + \pi_3) = \pi_1 \\ \frac{1}{3}\pi_3 = \pi_2 \\ \frac{1}{3}\pi_2 = \pi_3 \\ \pi_1 + \frac{1}{3}(\pi_2 + \pi_3) = \pi_4 \end{cases}$$

and solving them gives $\pi_1 = \pi_2 = \pi_3 = \pi_4 = 0$, which violates the normalization requirement $\sum_i \pi_i = 1$.

One solution is to replace each row of 0, like the last row in $\mathbf{H}$ above, by a row of $1/N$. Intuitively, this is saying that even if a webpage does not point to any other webpage, we will force it to spread its importance score evenly among all the webpages out there.

Mathematically, this amounts to adding a matrix of $\frac{1}{N}(\mathbf{w}\mathbf{1}^T)$ to $\mathbf{H}$, where $\mathbf{1}$ is simply a vector of 1s, and $\mathbf{w}$ is a vector with the ith entry being 1 if webpage i points to no other webpages (a dangling node) and 0 otherwise (not a dangling node). This is an *outer product* between two N-dimensional vectors, which leads to an $N \times N$ matrix. For example, if $N = 2$ and $\mathbf{w} = [1\ 0]^T$, we have

$$\frac{1}{2}\begin{pmatrix} 1 \\ 0 \end{pmatrix}(1\ 1) = \begin{pmatrix} 1/2 & 1/2 \\ 0 & 0 \end{pmatrix}.$$

This new matrix we add to $\mathbf{H}$ is clearly simple. Even though it is big: $N \times N$, it is actually the same vector $\mathbf{w}$ repeated N times. We call it a **rank-1 matrix**.

The resulting matrix:

$$\hat{\mathbf{H}} = \mathbf{H} + \frac{1}{N}(\mathbf{w}\mathbf{1}^T),$$

has all the entries non-negative and each row adds up to 1. So we can think of each row as a probabiliy vector, with the (i, j)th entry of $\hat{\mathbf{H}}$ indicating the probability that, if you are currently on webpage i, you will click on a link and go to webpage j.

Well, the structure of the matrix says that you are equally likely to click on any links shown on a webpage, and if there's no link at all, you will be equally

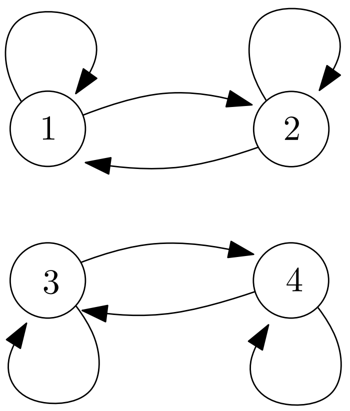

Figure 3.3 A network of hyperlinked webpages with multiple consistent score vectors.

likely to visit any other webpages. Such behavior is called a **random walk on graphs** and can be studied as **Markov chains** in probability theory. Clearly this does not model web browsing behavior exactly, but turns out it strikes a pretty effective balance between *simplicity* of the model and *usefulness* of the resulting webpage ranking. We will see a similar model for influence in social networks in Chapter 8.

Second, there might be many consistent score vectors all compatible with a given $\hat{\mathbf{H}}$. For example, for the graph in Figure 3.3, we have

$$\mathbf{H} = \begin{bmatrix} 1/2 & 1/2 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 \\ 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 1/2 & 1/2 \end{bmatrix}.$$

Different choices of $\pi[0]$ result in different π^*, which are all consistent. For example, if $\pi[0] = [1\ 0\ 0\ 0]^T$, then $\pi^* = [0.5\ 0.5\ 0\ 0]^T$. If $\pi[0] = [0\ 0.3\ 0.7\ 0]^T$, then $\pi^* = [0.15\ 0.15\ 0.35\ 0.35]^T$.

One solution to this problem is to add a little *randomization* to the iterative procedure and the recursive definition of importance. Intuitively, we say there is a chance of $(1-\theta)\%$ that you will be jumping to some other random webpage, without clicking on any of the links on the current webpage.

Mathematically, we add yet another matrix $\frac{1}{N}\mathbf{1}\mathbf{1}^T$, a matrix of 1s scaled by $1/N$ (and clearly a rank-1 matrix), to $\hat{\mathbf{H}}$. But this time a *weighted* sum, with a weight $\theta \in [0,1]$. $(1-\theta)$ describes how likely you will randomly jump to some other webpage. The resulting matrix is called the **Google matrix**:

$$\mathbf{G} = \theta\hat{\mathbf{H}} + (1-\theta)\frac{1}{N}\mathbf{1}\mathbf{1}^T. \tag{3.2}$$

Now we can show that no matter what is the initialization vector $\pi[0]$, the iterative procedure below:

$$\pi^T[k] = \pi^T[k-1]\mathbf{G} \tag{3.3}$$

will converge as $k \to \infty$, and converge to the unique vector π^* representing the consistent set of importance scores. Obviously, π^* is the left eigenvector of $\mathbf{G}$ corresponding to the eigenvalue of 1:

$$\pi^{*T} = \pi^{*T}\mathbf{G}. \tag{3.4}$$

One can then normalize π^*: take $\pi_i^* / \sum_j \pi_j^*$ as the new value of π_i^*, and rank the entries in descending order, before outputting them on the search result webpage in that order. The matrix $\mathbf{G}$ is designed such that there is a solution to 3.4 and that 3.3 converges from any initialization.

Whichever way you compute π^*, taking (the normalized and ordered version of) π^* as the basis of ranking is called the **pagerank algorithm**. Compared to DPC for wireless networks in Chapter 1, the matrix $\mathbf{G}$ in pagerank is much larger, but we can afford a centralized computation.

3.3 An Example

Consider the network in Figure 3.4 with 8 nodes and 16 directional links, we have

$$\mathbf{H} = \begin{bmatrix} 0 & 1/2 & 1/2 & 0 & 0 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & 1/2 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 0 & 0 & 0 & 0 & 1/2 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 0 & 0 & 0 & 1/2 \\ 0 & 0 & 0 & 1/2 & 1/2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 0 & 1/2 & 0 & 0 \\ 1/3 & 0 & 0 & 1/3 & 0 & 0 & 1/3 & 0 \end{bmatrix}.$$

Here $\hat{\mathbf{H}} = \mathbf{H}$ since there is no dangling node. Taking $\theta = 0.85$, we have

$$\mathbf{G} = \begin{bmatrix} 0.0188 & 0.4437 & 0.4437 & 0.0188 & 0.0188 & 0.0188 & 0.0188 & 0.0188 \\ 0.4437 & 0.0188 & 0.0188 & 0.0188 & 0.4437 & 0.0188 & 0.0188 & 0.0188 \\ 0.0188 & 0.4437 & 0.0188 & 0.0188 & 0.0188 & 0.0188 & 0.0188 & 0.4437 \\ 0.0188 & 0.0188 & 0.8688 & 0.0188 & 0.0188 & 0.0188 & 0.0188 & 0.0188 \\ 0.0188 & 0.0188 & 0.0188 & 0.4437 & 0.0188 & 0.0188 & 0.0188 & 0.4437 \\ 0.0188 & 0.0188 & 0.0188 & 0.4437 & 0.4437 & 0.0188 & 0.0188 & 0.0188 \\ 0.0188 & 0.0188 & 0.0188 & 0.4437 & 0.0188 & 0.4437 & 0.0188 & 0.0188 \\ 0.3021 & 0.0188 & 0.0188 & 0.3021 & 0.0188 & 0.0188 & 0.3021 & 0.0188 \end{bmatrix}.$$

Initializing $\pi[0] = [1/8 \ 1/8 \ \cdots 1/8]^T$ (by convention, a vector is a column vector, so when we write a vector horizontally on a line, we put transpose symbol

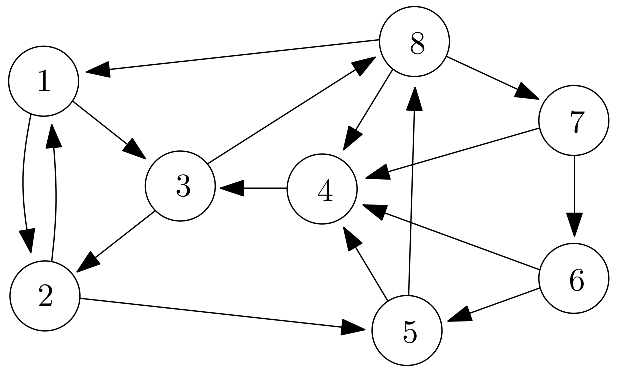

Figure 3.4 An example of the pagerank algorithm with 8 webpages and 16 hyperlinks. Webpage 3 is ranked the highest even though webpage 4 has the largest in-degree.

on top of the vector), iteration (3.3) gives

$$\begin{aligned}
\pi[1] &= [0.1073\ 0.1250\ 0.1781\ 0.2135\ 0.1250\ 0.0719\ 0.0542\ 0.1250]^T\\
\pi[2] &= [0.1073\ 0.1401\ 0.2459\ 0.1609\ 0.1024\ 0.0418\ 0.0542\ 0.1476]^T\\
\pi[3] &= [0.1201\ 0.1688\ 0.2011\ 0.1449\ 0.0960\ 0.0418\ 0.0606\ 0.1668]^T\\
\pi[4] &= [0.1378\ 0.1552\ 0.1929\ 0.1503\ 0.1083\ 0.0445\ 0.0660\ 0.1450]^T\\
\pi[5] &= [0.1258\ 0.1593\ 0.2051\ 0.1528\ 0.1036\ 0.0468\ 0.0598\ 0.1468]^T\\
\pi[6] &= [0.1280\ 0.1594\ 0.2021\ 0.1497\ 0.1063\ 0.0442\ 0.0603\ 0.1499]^T\\
&\vdots
\end{aligned}$$

and $\pi^* = [0.1286\ 0.1590\ 0.2015\ 0.1507\ 0.1053\ 0.0447\ 0.0610\ 0.1492]^T$. This means the ranked order of the webpages are: 3, 2, 4, 8, 1, 5, 7, 6.

The node with the largest in-degree, *i.e.*, the number of links pointing to a node, is node 4, which is *not* ranked the highest. This is in part because its importance score is spread exclusively to node 3. As we will see again in Chapter 8, there are many more useful metrics measuring node importance than just the degree, pagerank being one of them.

3.4 Advanced Material

3.4.1 Generalized pagerank and some basic properties

The Google matrix $\mathbf{G}$ can be generalized if the randomization ingredient is more refined. Instead of the matrix $\frac{1}{N}\mathbf{1}\mathbf{1}^T$, we can add the matrix $\mathbf{1}\mathbf{v}^T$ (again, the outer product of two vectors), where $\mathbf{v}$ can be *any* probability distribution, with $\frac{1}{N}\mathbf{1}^T$ being a special case of that. We can also generalize the dangling node treatment: instead of adding $\frac{1}{N}\mathbf{w}\mathbf{1}^T$ to $\mathbf{H}$, where $\mathbf{w}$ is the indicator vector of dangling nodes, we can add $\mathbf{w}\mathbf{v}^T$.

Now, the Google update equation can be written in the long form (not using the shorthand notation $\mathbf{G}$) as a function of the given webpage connectivity matrix $\mathbf{H}$, vector $\mathbf{w}$ indicating the dangling webpages, and the two algorithmic parameters: scalar θ and vector $\mathbf{v}$:

$$\pi^T\mathbf{G} = \theta\pi^T\mathbf{H} + \pi^T(\theta\mathbf{w} + (1-\theta)\mathbf{1})\mathbf{v}^T. \tag{3.5}$$

You should verify that the above equation is indeed the same as (3.3).

There are many viewpoints to further interpret (3.3) and connect it to matrix theory, Markov chain theory, and linear systems theory. For example:

- π^* is the left eigenvector corresponding to the dominant eigenvalue of a positive matrix.
- It also represents the stationary distribution of a Markov chain whose transition probabililities are in $\mathbf{G}$.
- And it represents the equilibrium of an economic growth model according to $\mathbf{G}$ (more on this viewpoint later in this section).

The major operational challenges of running the seemingly simple update (3.3) are *scale* and *speed*: there are billions of webpages and Google needs to return the results almost instantaneously.

Still, the power method (3.3) offers many numerical advantages compared to a direct computation of the dominant eigenvector of $\mathbf{G}$. First, (3.3) can be carried out by multiplying vector by the sum of $\mathbf{H}$ and two rank-1 matrices. This is numerically simple: $\mathbf{H}$ is very lage but also very *sparse*: each webpage usually links to just a few other webpages, so almost all the entries in $\mathbf{H}$ are zero. Multiplying by rank-1 matrices is also easy. Furthermore, at each iteration, we only need to store the current π vector.

While we have not discussed the speed of convergence, it is clearly important to speed up the computation of π^*. It turns out that the convergence speed in this case is governed by the second largest eigenvalue $\lambda_2(\mathbf{G})$ of $\mathbf{G}$, which can be shown to be, approximately, θ. So this parameter θ controls the tradeoff between convergence speed and the relevance of hyperlink graph in computing the importance scores: smaller θ (closer to 0) drives the convergence faster, but also de-emphasizes the relevance of hyperlink graph structure more. This is hardly surprising: if you view the webpage importance scores more like random objects, it is easier to compute the equilibrium. Usually $\theta = 0.85$ is believed to be a pretty good choice. It gives convergence in about 50 iterations while still giving most of the weight to the actual hyperlink graph structure rather than the randomization component in $\mathbf{G}$.

3.4.2 Pagerank as solution to a linear equation

Pagerank is similar to the distributed power control in Chapter 1. They both apply the power method to solve a system of linear equations. The solution to those equations capture the right engineering configuration in the network, whether that is the relative importance of webpages in a hyperlink graph, or the best transmit power vector in a wireless interference environment. This conceptual connection can be sharpened to an exact, formal parallelism below.

First, we can rewrite the characterization of π^* as the solution to the following linear equation (rather than as the dominant left eigenvector of matrix $\mathbf{G}$ (3.4), the viewpoint we have been taking so far):

$$(\mathbf{I} - \theta \mathbf{H})^T \pi = \mathbf{v}. \tag{3.6}$$

Compare (3.6) with the characterization of optimal power vector in distributed power control algorithm in Chapter 1:

$$(\mathbf{I} - \mathbf{DF})\mathbf{p} = \mathbf{v}.$$

Of course, the vectors $\mathbf{v}$ are defined differently in these two cases: based on webpage viewing behavior in pagerank and receiver noise in power control. But we see a striking parallelism: the consistent score vector π, and the optimal tansmit power vector $\mathbf{p}$ are both solutions to a linear equation with the following structure: identity matrix minus a scaled version of the network connectivity matrix.

In pagerank, the scaling is done by one scalar θ, and the network connectivity is represented by the hyperlink matrix $\mathbf{H}$. This makes sense since the key factor here is the hyperlink connectivity pattern among the webpages.

In power control, the scaling is done by many scalars in the diagonal matrix $\mathbf{D}$: the target SIR for each user, and the network connectivity is represented by the normalized channel gain matrix $\mathbf{F}$. This makes sense since the key factor here is the strength of interference channels.

To make the parallelism exact, we can also think of a generalization of Google matrix $\mathbf{G}$ where each webpage has its own scaling factor θ.

The general theme for solving these two linear equations can be stated as follows. Suppose you want to solve a system of linear equations $\mathbf{Ax} = \mathbf{b}$ but do not want to directly invert the square matrix $\mathbf{A}$. You might be able to split $\mathbf{A} = \mathbf{M} - \mathbf{N}$, where $\mathbf{M}$ is invertible and its inverse $\mathbf{M}^{-1}$ can be much more easily computed than $\mathbf{A}^{-1}$.

The following **linear stationary iteration** over times denoted by k:

$$\mathbf{x}[k] = \mathbf{M}^{-1}\mathbf{N}\mathbf{x}[k-1] + \mathbf{M}^{-1}\mathbf{b}$$

will converge to the desired solution:

$$\lim_{k \to \infty} \mathbf{x}[k] = \mathbf{A}^{-1}\mathbf{b},$$

from any initialization $\mathbf{x}[0]$, provided that the largest eigenvalue of $\mathbf{M^{-1}N}$ is smaller than 1. Both DPC and pagerank are special cases of this general algorithm.

But we still need to show that (3.6) is indeed equivalent to (3.4): a π that solves (3.6) also solves (3.4), and vice versa. First, starting with a π that solves (3.6), we can easily show the following string of equalities:

$$\begin{aligned}\mathbf{1}^T\mathbf{v} &= \mathbf{1}^T(\mathbf{I}-\theta\mathbf{H})^T\pi\\ &= \mathbf{1}^T\pi - \theta(\mathbf{H}\mathbf{1})^T\pi\\ &= \mathbf{1}^T\pi - \theta(\mathbf{1}-\mathbf{w})^T\pi\\ &= \pi^T(\theta\mathbf{w}+(1-\theta)\mathbf{1}),\end{aligned}$$

where the first equality uses (3.6) and the third equality uses the fact that summing each row of $\mathbf{H}$ gives a vector of 1s (except those rows corresponding to dangling webpages). The other two equalities are based on simple algebraic manipulations.

But $\mathbf{1}^T\mathbf{v} = 1$ by design, so we know

$$\pi^T(\theta\mathbf{w}+(1-\theta)\mathbf{1}) = 1.$$

Now we can readily check that $\pi^T\mathbf{G}$, using its definition in (3.5) and the above equation, equals $\theta\pi^T\mathbf{H}+\mathbf{v}$.

Finally, using one more time the assumption that π satisfies (3.6), *i.e.*, $\mathbf{v} = (\mathbf{I}-\theta\mathbf{H})^T\pi$, we complete the argument:

$$\pi^T\mathbf{G} = \theta\pi^T\mathbf{H} + (\mathbf{I}-\theta\mathbf{H})^T\pi = \theta\pi^T\mathbf{H} - \theta\pi^T\mathbf{H} + \pi = \pi.$$

Therefore, any π solving the linear equation (3.6) is also a dominant left eigenvector of $\mathbf{G}$ that solves (3.4). And vice versa can be similarly shown.

3.4.3 Scaling up and speeding up

It is not easy to adjust the parameters in pagerank computation. We discussed the role of θ before, and we know that when θ is close to 1, pagerank results become very sensitive to small changes in θ, since the importance matrix $(\mathbf{I} - \theta\hat{\mathbf{H}})^{-1}$ approach infinity.

There is also substantial research going into designing the right randomization vector $\mathbf{v}$. Even the entries of the $\mathbf{H}$ matrix: a web surfer likely will not pick all of the hyperlinked webpages equally likely, and their actual behavior can be recorded to adjust the entries of $\mathbf{H}$.

But the biggest challenge to running pagerank is *scale*: how to scale up to really large matrices? How to quickly compute and update the rankings? There are both storage and computation challenges. And there are many interesting approaches developed over the years, including the following five. The first one is a computational acceleration method. The other four are *approximations*, two in changing the notion of optimality and two in restructuing the graph of the hyperlinked webpages.

1. *Decomposition of* $\mathbf{H}$. A standard trianglular decomposition gives $\mathbf{H} = \mathbf{DL}$ where $\mathbf{D}$ is a diagonal matrix with entries being 1 over the number of links from webpage i, and $\mathbf{L}$ is a binary adjacency matrix. So only integers, instead of real numbers, need to be stored to describe $\mathbf{H}$. Now suppose there are N webpages, and on average each webpage points to M webpages. N is huge: tens of billions, and M is very small: often 10 or less. Instead of NM multiplications, this matrix decomposition reduces it to just M multiplications.
2. *Relax the meaning of convergence.* It is not the values of importance scores that matter to most people, it is just the *order* of the webpages, especially the top ones. So once the computation of pagerank is sure about the order, there is no need to further improve the accuracy of computing π towards convergence.
3. *Differentiate among the webpages.* Most webpages' pageranks quickly converge, and can be locked while the other webpages' pageranks are refined. This is an approximation that works particularly well when the pageranks follow the power law that we'll discuss later.
4. *Leave the dangling nodes out.* There are many dangling nodes in the billions of webpages out there, and their behavior in the matrices and computations involved are pretty similar. So they might be grouped together to speed up the computation.
5. *Aggregation of webpages.* When many nodes are lumped together into a cluster, then *hierarchical* computation of pageranks can be recursively computed, first treating each cluster as one webpage, then distributing the pagerank of that cluster among the actual webpages within that cluster. We will visit an example of this important principle of building hierarcy to reduce computation (or communication) load in a homework problem.

3.4.4 Beyond the basic search

There is another player to the game of search: companies that specialize in increasing a webpage's pagerank, possibly pushing it to the top few search results, or even to the very top spot. This obviously important industry is called SEO: **Search Engine Optimization**. There are many proprietary techniques used by SEO companies. Some techniques enhance content relevance scores, sometimes by adding bogus tags in the html files. Other techniques increase the importance score, sometimes by adding links pointing to the customers sites, and sometimes by creating several truly important webpages and then attaching many other webpages as its outgoing neighbors.

Google is also playing this game by detecting SEO techniques and then updating its ranking algorithm so that the artificial help from SEO techniques is minimized. For example, in early 2011, Google had a major update of its ranking algorithm to counter the SEO effects.

There are also many important variants to the basic type of search we discussed. For example, personalized search based on user's feedback on how she likes the usefulness of the top webpages in the search result webpage. Multimedia search is another challenging area: searching through images, audios, and video clips require very different ways of indexing, storing, and ranking the content than text-based search.

Further reading

The pagerank algorithm is covered in almost every single book on network science these days. Some particularly useful references are as follows.

The Google founders wrote the following seminal paper explaining pagerank algorithm back in 1998:

[BP98] S. Brin and L. Page, "The anatomy of a large-scale hypertextual Web search engine", *Computer Netowrks and ISDN Systems* vol. 33, pp. 107-117, 1998.

The standard reference book devoted to pagerank is

[LM06] A. N. Langville and C. D. Meyer, *Google's Pagerank and Beyond*, Princeton University Press, 2006.

A well written website explaining pagerank is this one:

[R] C. Ridings, "Pagerank explained: Everything you've always wanted to know about Pagerank", `http://www.rankwrite.com`.

Dealing with non-negative matrices like the three we saw in this chapter is well documented, *e.g.*, in the following textbook:

[BP79] A. Berman and R. J. Plemmons, *Nonnegative Matrices in the Mathematical Sciences*, Acadmeic Press, 1979.

Computational issues in matrix multiplication is treated in textbooks like this one:

[GV96] G. Golub and C. F. van Van Loan, *Matrix Computations*, 3rd Ed., The Johns Hopkins University Press, 1996.