0.1 PageRank

- *Algorithm:* PageRank (algo. 1)
- *Input:* A network/graph G with N nodes
- *Complexity:* $\mathcal{O}(k * N)$, where k is #iteration
- *Data structure compatibility:* Graph
- *Common applications:* Web pages ranking.

Problem. PageRank

In a network, we measure the importance of a node with *rank*. The *PageRank* algorithm is a Monte-Carlo method to estimate the rank of all nodes in a network. It sets a uniform prior distribution for ranks and recursively updates it with a network transition matrix obtained from node connections.

Description

A network is composed of nodes, and each of them has its own portion of significance. In many cases, we would prefer to visit the more important nodes. For example, among millions of entries in the result of search engines, those pages with higher weights will be shown in the first few pages, which are considered of higher quality. We measure the importance of a node with *rank*. A page with higher rank has more importance in the network.

PageRank is developed by Google company to rank web pages in their search engine results. Is is named after *Larry Page*, one of the co-founder of the company [3]. PageRank estimates the importance of an web page by observing the number and quality of links to it. The idea is based on two assumptions:

- A page if of more importance if it has more links pointing in.
- A link from an important page add more to a page's own importance.

Based on the idea, we may have the following way to calculate the rank of a page. Denote r_i the rank of node i in a network, and we have

$$r_i = \sum_{j \in \mathcal{N}(i)} \frac{r_j}{d_j}$$

where $\mathcal{N}(i)$ means the nodes linking into i and d_j is the out degree of node j. The equations for all nodes form a linear equation system, and we can express it in the way of

$$\mathbf{r} = M \cdot \mathbf{r}$$

where matrix M is defined as

$$M_{ij} = \begin{cases} 1/d_j, \text{ if } j \to i \\ 0, \text{ otherwise} \end{cases}$$

Therefore, to find the page ranks is to find the eigenvector of the matrix M. However, to efficiently solve for $\mathbf{r}$, we would adopt power iteration. As a Monte-Carlo Markov Chain (MCMC), the ranks is the stationary distribution of the matrix M, and the answer will approach it no matter what initial value we assign to $\mathbf{r}$. We will follow the scheme below:

1. Initialize $\mathbf{r}^(0) = [r_1, \cdots, r_N]$ with $r_i = 1/N$ for all $i \in [1, N]$.
2. Update $\mathbf{r}^{(t+1)} = M \cdot \mathbf{r}^{(t)}$
3. Repeat step 2 until $|r^{(t+1)} - r^{(t)}| < \eta$ a small positive number.

Algorithm 1: PageRank

Input : Network with N nodes, small positive number η
Output: ranks $\mathbf{r}$ for all nodes

for $i \leftarrow 1$ *to* N **do**
  for $j \leftarrow 1$ *to* N **do**
    if j *points to* i **then** $M_{ij} = 1/degree_{j,out}$;
    else $M_{ij} = 0$;
  end for
end for
for $i \leftarrow 1$ *to* N **do**
  $r_i^0 \leftarrow 1/N$
end for
while $|r^{(t+1)} - r^{(t)}| < \eta$ **do**
  $\mathbf{r}^{(t+1)} = M \cdot \mathbf{r}^{(t)}$
end while
return $\mathbf{r}$

There are two main problem with this simple form. Firstly, if the out-degree of node j is 0, i.e. it is linked to no other pages, then all $M_{.j} = 0$, causing the chain to enter a dead end. To deal with it, simply assign a uniform distribution to the j-th row.

Another problem is when a group of one or more pages have no links pointing out of the group, most of the ranks would be adsorbed by that group. To avoid it, we introduce random jump. Instead of always following the current ranks, the random walk jump to a node according to a uniform distribution with a small probability $1 - \beta$. This help jump out of such rank "drains". The update rule now written as [1]

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

and this leads to the Google Matrix that Google adopts

$$A = \beta M + (1 - \beta) + \frac{1}{N} I_{1/N}.$$

The complexity of PageRank algorithm is $\mathcal{O}(kN)$, where k is the number of iterations. We have known that a matrix-vector multiplication costs $(O)(n^2)$, then the total complexity is originally larger. However, as a sparse network, the matrix M have few non-zero elements, and sparse matrix multiplication requires $\mathcal{O}(nonZero(N))$. What's more, WWW is found to follow the power-law degree distribution [2], which means that the average web pages have 10 links [4], and $nonZero(M) \approx 10N$. Therefore, the total time complexity is reduce to $\mathcal{O}(kN)$.

References.

[1] Krishna Bharat and Monika R Henzinger. "Improved algorithms for topic distillation in a hyperlinked environment". In: *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*. 1998, pp. 104–111 (cit. on p. 2).

[2] Bernardo A Huberman and Lada A Adamic. "Growth dynamics of the world-wide web". In: *Nature* 401.6749 (1999), pp. 131–131 (cit. on p. 2).

[3] Google Inc. *Google Press Center: Fun Facts*. July 2001. URL: https://www.google.com/competition/howgooglesearchworks.html (cit. on p. 1).

[4] Jure Leskovec, Kevin J Lang, Anirban Dasgupta, and Michael W Mahoney. "Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters". In: *Internet Mathematics* 6.1 (2009), pp. 29–123 (cit. on p. 2).