How do we really compute the PageRank?

Mining of Massive Datasets
Leskovec, Rajaraman, and Ullman
Stanford University

Computing Page Rank

- **Key step is matrix-vector multiplication**
  - $r^{new} = A \cdot r^{old}$
- Easy if we have enough main memory to hold **A**, $\mathbf{r}^{old}$, $\mathbf{r}^{new}$
- **Say N = 1 billion pages**
  - We need 4 bytes for each entry (say)
  - 2 billion entries for vectors, approx 8GB
  - Matrix **A** has $\mathbf{N}^2$ entries
    - 10^{18} is a large number!

$$\mathbf{A} = \beta \cdot \mathbf{M} + (1-\beta)\ [\mathbf{1/N}]_{\mathbf{N \times N}}$$

$$A = 0.8 \begin{bmatrix} ½ & ½ & 0 \\ ½ & 0 & 0 \\ 0 & ½ & 1 \end{bmatrix} + 0.2 \begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix}$$

$$= \begin{bmatrix} 7/15 & 7/15 & 1/15 \\ 7/15 & 1/15 & 1/15 \\ 1/15 & 7/15 & 13/15 \end{bmatrix}$$

Matrix Formulation

- Suppose there are N pages
- Consider page j, with $\mathbf{d}_j$ out-links
- We have $M_{ij} = 1/|d_j|$ when $j \longrightarrow i$
  and $M_{ij} = 0$ otherwise
- **The random teleport is equivalent to:**
  - Adding a **teleport link** from j to every other page and setting transition probability to $(1-\beta)/N$
  - Reducing the probability of following each out-link from $1/|d_j|$ to $\beta/|d_j|$
  - **Equivalent:** Tax each page a fraction $(1-\beta)$ of its score and redistribute evenly

Rearranging the Equation

- $\boldsymbol{r} = \boldsymbol{A} \cdot \boldsymbol{r}$, **where** $\boldsymbol{A_{ij} = \beta M_{ij} + \frac{1-\beta}{N}}$
- $r_i = \sum_{j=1}^{N} A_{ij} \cdot r_j$
- $r_i = \sum_{j=1}^{N} \left[\beta M_{ij} + \frac{1-\beta}{N}\right] \cdot r_j$

$$= \sum_{j=1}^{N} \beta M_{ij} \cdot r_j + \sum_{j=1}^{N} \frac{1-\beta}{N} r_j$$

$$= \sum_{j=1}^{N} \beta M_{ij} \cdot r_j + \frac{1-\beta}{N} \qquad \text{since } \sum r_j = 1$$

- **So we get:** $\boldsymbol{r = \beta M \cdot r + \left[\frac{1-\beta}{N}\right]_N}$

此处把M提取出来，因为M是稀疏的

Note: Here we assumed **M** has no dead-ends.

$[x]_N$ … a vector of length N with all entries $\boldsymbol{x}$

Sparse Matrix Formulation

- We just rearranged the **PageRank equation**

$$r = \beta M \cdot r + \left[\frac{1-\beta}{N}\right]_N$$

  - where $[(1-\beta)/N]_N$ is a vector with all N entries $(1-\beta)/N$

- ***M*** is a **sparse matrix!** (with no dead-ends)
  - 10 links per node, approx 10N entries
- **So in each iteration, we need to:**
  - Compute $r^{new} = \beta M \cdot r^{old}$
  - Add a constant value $(1-\beta)/N$ to each entry in r^{new}
    - **Note if *M* contains dead-ends then $\sum_i r_i^{new} < 1$ and we also have to renormalize r^{new} so that it sums to 1**

PageRank: The Complete Algorithm

- **Input: Graph G and parameter β**
  - Directed graph G with **spider traps** and **dead ends**
  - Parameter β
- **Output: PageRank vector r**
  - **Set:** $r_j^{(0)} = \frac{1}{N}, \quad t = 1$
  - **do:**
    - $\forall j: \boldsymbol{r'^{(t)}_j = \sum_{i \to j} \beta \frac{r_i^{(t-1)}}{d_i}}$

      $\boldsymbol{r'^{(t)}_j = 0}$ if in-deg. of $\boldsymbol{j}$ is $\mathbf{0}$

      Dead Ends
    - **Now re-insert the leaked PageRank:**

      $\forall \boldsymbol{j}: \boldsymbol{r_j^{(t)} = r'^{(t)}_j + \frac{1-S}{N}}$ **where:** $S = \sum_j r'^{(t)}_j$
    - $\boldsymbol{t = t + 1}$
  - **while** $\sum_j \left| r_j^{(t)} - r_j^{(t-1)} \right| > \varepsilon$