网页排序算法PageRank

2017-02-17 Updated: 2017-02-17 7.6K

计算PageRank，之前直接用NetworkX的API `nx.pagerank` 求的，大概知道PageRank是怎么回事，但对算法细节并不了解。现在想进一步了解PageRank的细节。

目录 [hide]

1. 直观理解

1.1 基本思想

PageRank是以Google创始人Larry Page的姓命名的，于1999被提出来，用于测量网页的相对重要性（对网页进行排序），学术论文如下：

> Page, L., Brin, S., Motwani, R., & Winograd, T. (1999). The PageRank citation ranking: Bringing order to the web. Stanford InfoLab. [PDF]

PageRank的设计受到学术论文引用启发（两人的父亲都是大学教授）。衡量一篇学术论文质量高与否，最重要的一个指标是引用次数，高引用量的论文通常意味着高质量。同理，如果一张网页被引用（以超链接的形式）多了，那么这张网页就比较重要。总结起来，PageRank的核心思想有两点（结合图1说明）：

- 越多的网页链接到一个网页（可以理解成投票，`D --> B`，`D`给`B`投了一票），说明这个网页更加重要，如图1的`B`。(一篇论文被很多论文引用)
- PageRank高的网页链接到一个网页，说明这张网页也很重要。如图1，尽管`C`只有一张网页`B`链接到它，但`C`的重要性高于`E`，尽管`E`有一堆小罗罗给它投票。（论文被大牛引用了，说明这篇论文很有价值）（也可以从话语权角度理解，重要的人说话份量重）

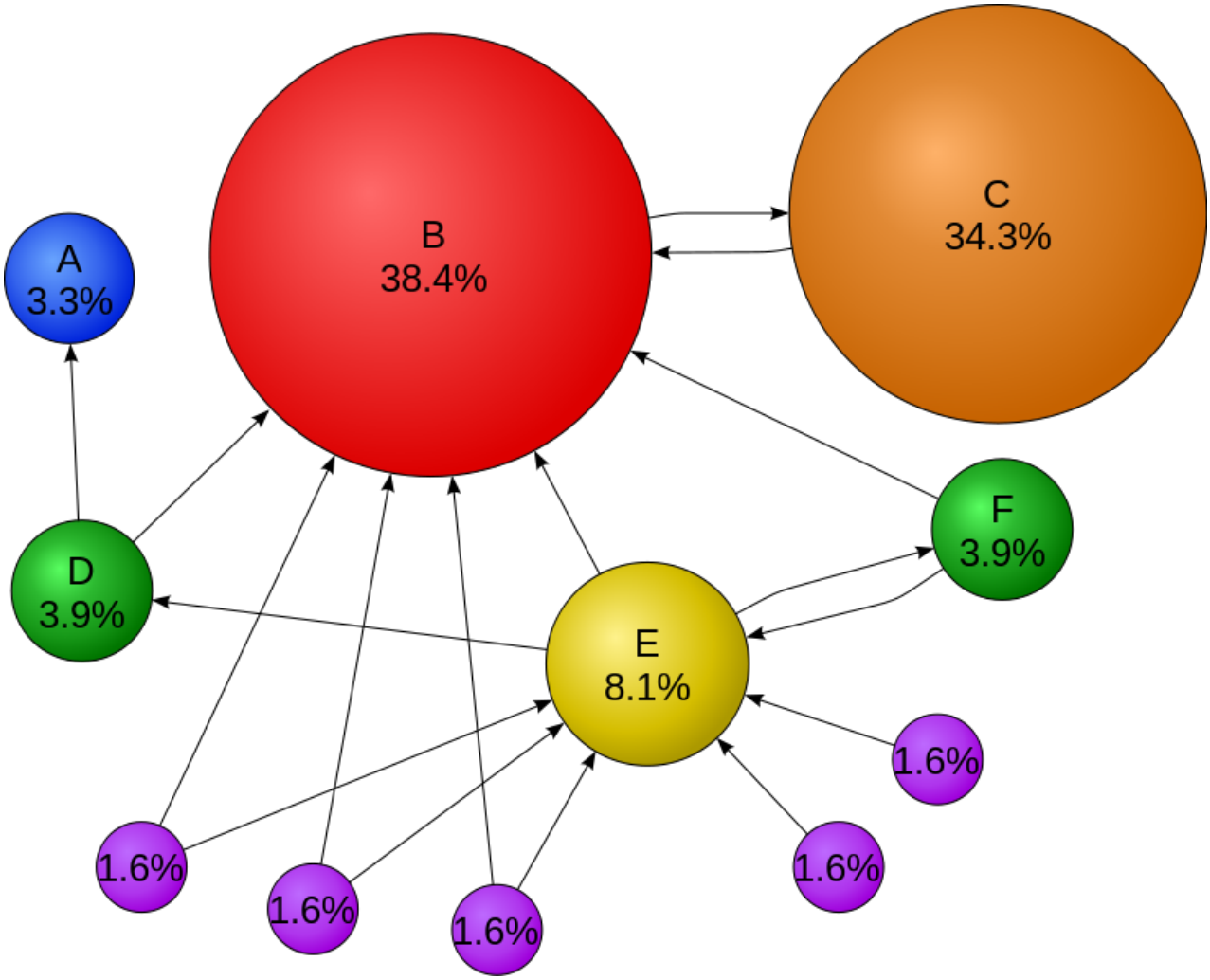

Fig. 1: PageRanks for a simple network (image from Wikipedia)

整个万维网（World Wide Web）可以抽象成一张有向图，节点表示网页，连线$p_i \to p_j$表示网页p_i包含了超链接p_j（p_i指向了p_j）。如果能对图中每个节点重要性量化，那么就能对网页进行排序了。PageRank提出之初就是为了对网页进行排序。

搜索引擎的工作原理可以简化为：输入关键词，返回与该关键词相关的网页（一个集合，相当于得到一张子图），在该子图上计算每个节点的PageRank值，PR值高的网页排在前面，低的就排在后面。

1.2 如何计算

接下来的问题是，如何计算每个节点的PageRank。想要知道一个网页p_i的PR值，需要知道：

- 有多少网页链接到了p_i
- 这些网页的PR值是多少

其他网页的PR值又很可能是依赖于p_i，这就陷入了“先有鸡还是先有蛋”的循环，要想知道p_i的PR值，就得知道链向p_i所有网页的PR值，而要知道其他网页的PR值，又得先知道p_i的PR值。

为了打破这个循环，佩奇和布林采用了一个很巧妙的思路， 即分析一个虚拟用户在互联网上的漫游过程。 他们假定：虚拟用户一旦访问了一个网页，下一步将以相同的概率访问被该网页所链接的任何一个其它网页[3]。比如，网页p_i包含N个超链接，那么虚拟用户访问这N个页面中的任何一个的概率是$1/N$。那么，网页的排序就可以看成一个虚拟用户在万维网漫游了很长时间，页面被访问的概率越大，其PR值就越高，网页排名也越靠前。

先从简化的PageRank说起，以PageRank论文的例子为例，看看PageRank是怎么计算的，如下：

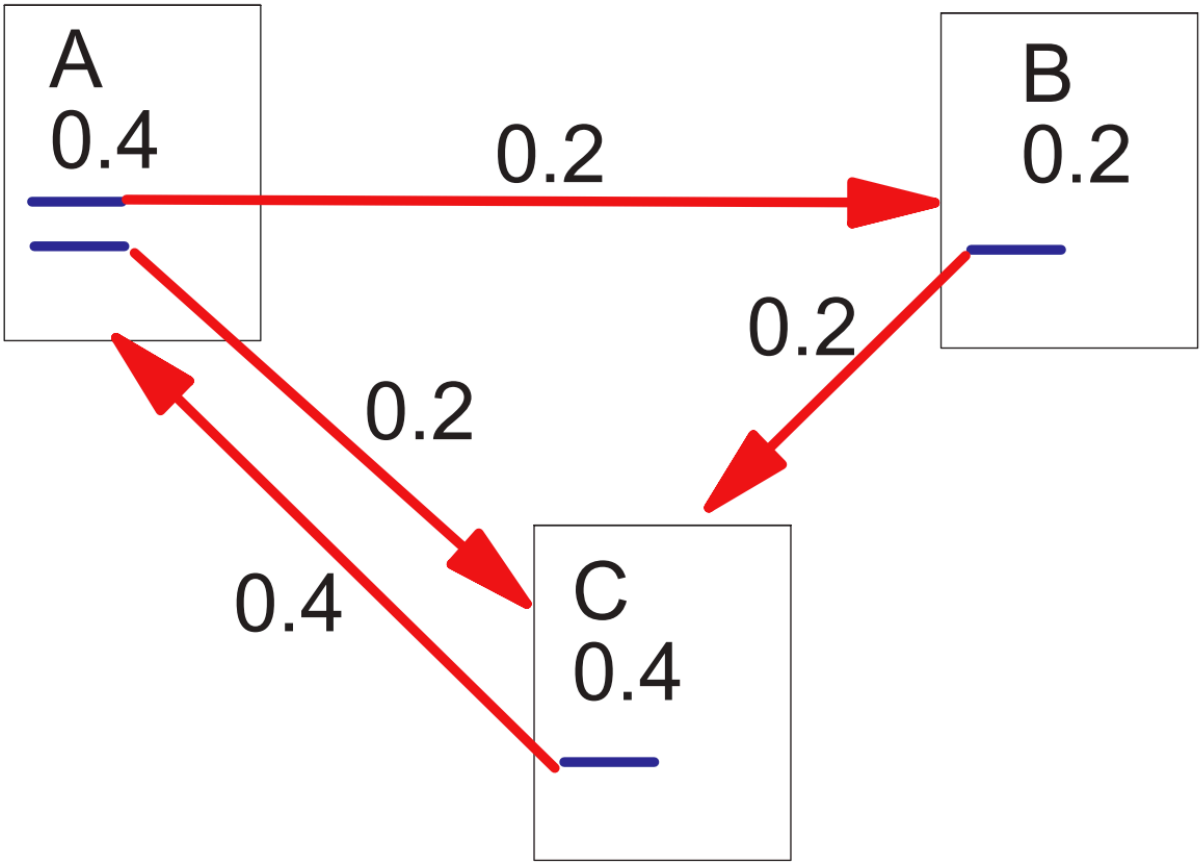

Fig. 2: Simplified PageRank calculation (image from [1])

每个节点初始化或者指定一个PageRank值（如`PR(A)=0.4`），网页`A`包含两个超链接，分别指向`B`和`C`（或者说`A`投票给`B`和`C`），`0.4`拆分成两份，每份`0.2`，所以`PR(B)=0.2`。`A`和`B`同时给`C`投票，所以`PR(C)=0.2+0.2=0.4`。如此，不断地迭代，最后每个节点的值会趋于稳定（或者说收敛），这样就求得了所有节点的PR值。事实上，在这个例子中，PageRank已收俭。

每个页面将其当前的PageRank值平均分配到本页面所有出链上，一个页面将所有入链的PR值累加起来就构成了该页面新的PR值。如此迭代下去，最后得到一个稳定值。用数学公式表达，如下：

$$PR(A) = \frac{PR(B)}{L(B)} + \frac{PR(C)}{L(C)} + \frac{PR(D)}{L(D)} + \cdots$$

更一般化地（$B(p_i)$表示所有链向网页p_i的集合），

$$PR(p_i) = \sum_{p_j \in B(p_i)} \frac{PR(p_j)}{L(p_j)}$$

但这样算存在两个问题：

- 对于没有forward links (outedges)的网页，即只有别人给她投票，她从不给别人投票，那么她的PageRank每次迭代都会增加。
- 对于没有blacklinks (inedges)的网页，即没人给她投票，其PageRank永远等于0。

对于第一个问题，给等式乘以一个小于1的常数d（damping factor，翻译成阻尼因数）；对于第二个问题，给等式加上一个常数。新的等式如下（N表示网页总数，或者节点数目）：

$$PR(p_i) = \frac{1-d}{N} + d \sum_{p_j \in B(p_i)} \frac{PR(p_j)}{L(p_j)}$$

其中，

- $B(p_i)$：链接到网页p_i的集合（a set of pages link to pipi）
- $L(p_j)$：从pjpj链出去的网页数目（the number of outbound links）

这样，就确保所有节点的PR值加起来等于1。

1.3 一个简单实例

以一个很简单的例子（`A < --> B`）来看PageRank是怎么收俭的。

Fig. 2: An illustration of PageRank calculation.

假设他们的初始PR值为1，第一次迭代后，PR(A)和PR(B)的值为：

```
PR(A) = 0.15/2 + 0.85*1.0                   = 0.9249999999999999
PR(B) = 0.15/2 + 0.85*0.9249999999999999    = 0.8612499999999998
```

写个简单的脚本，得到每次迭代后的值，部分如下：

```
1: A=0.925000     B=0.861250
2: A=0.807062     B=0.761003
3: A=0.721853     B=0.688575
4: A=0.660289     B=0.636245
5: A=0.615808     B=0.598437
6: A=0.583672     B=0.571121
7: A=0.560453     B=0.551385
8: A=0.543677     B=0.537126
9: A=0.531557     B=0.526823
10: A=0.522800     B=0.519380
11: A=0.516473     B=0.514002
12: A=0.511902     B=0.510116
13: A=0.508599     B=0.507309
14: A=0.506213     B=0.505281
15: A=0.504489     B=0.503815
16: A=0.503243     B=0.502757
17: A=0.502343     B=0.501992
18: A=0.501693     B=0.501439
19: A=0.501223     B=0.501040
20: A=0.500884     B=0.500751
...
42: A=0.500001     B=0.500001
43: A=0.500001     B=0.500000
44: A=0.500000     B=0.500000
45: A=0.500000     B=0.500000
```

可见，随着迭代次数的增加，PageRank越来越接近收俭值0.5。

1.4 迭代次数

迭代次数越多，结果越准确，但花费时间 也越长。出于效率考虑，在实际应用中，当PR值落在误差允许范围内（PR值跟前一次比较，如$PR'(A) - PR(A) < 1.0e-6$，想想浮点数在计算机的存储），就可以返回结果了。

当然，对于超大型网络来说，有更复杂的计算方法，比如分布式。

1.5 PR初始值

不管节点PR初始值怎么设置，最后节点的PR值都一样，但收俭速度不一样。NetworkX的`pagerank`实现是将PR值初始化为`1/N`。

1.6 Damping factor

跟PR初始值类似，d的取值也会影响算法效率。根据Page的论文，d通常设为0.85。