doi：10.3969/j.issn.1001−2400.2018.06.0

PageRank 算法发展历程

孙晖[1]

(1. 西安电子科技大学 软件学院 ，陕西 西安 710071)

摘要：Stanford 大学博士研究生 Sergey Brin 和 Lawrence Page 在 1998 年提出了网络搜索排名算法 PageRank[1]，不同于以网页访问量来进行网页排名的其它算法。PageRank 算法是基于被其它网页链接的次数（RP 值）来决定一个网页的重要程度。在此基础上，很多学者相继提出基于 PageRank 的改进算法，包括斯坦福大学 Taher Haveliwalat[2] 等人提出的主题敏感的 PageRank 算法。如今，PageRank 算法开始在图聚类 Hao Yin[3] 中大放异彩。

关键词：PangRank，网页，主题感知，图聚类

中图分类号: **文献标识码：**A **文章编号：**1001-2400(2018)06-0-0

The Development Process of PageRank Algorithm

Hui Sun[1]

(1. School of Software Engineering, Xidian Univ., Xi’an 710071)

Abstract: The doctors of Stanford University Sergey Brin and Lawrence Page introduced a new citation ranking algorithm PageRank[1] in 1998, which is different from other algorithms that base on page views rank. PageRank was used to decide a page's importance based on the times of being linked by other web pages (PR). And this basis, many scholars introduced some improved algorithms of PageRank, such as the Stanford University's Taher Haveliwalat[2] introduced a PageRank algorithm based on topic-sensitive. At present, PageRank has been used in graph clustering——Hao Yin[3]。

Key Words: PageRank, web, topic-sensitive, graph clustering

收稿日期：2018.6.28 **网络出版日期：**
基金项目：Web 工程作业
作者简介：孙晖（1997-），男，本科，西安电子科技大学软件学院。
网络出版地址：

1 PageRank 算法

PageRank 算法是 Stanford 大学博士生 Sergey Brin 和 Lawrence Page 在 1998 年提出的网络搜索排名算法，可以毫不夸张地说，Google 公司就是基于这个算法而创立的。PageRank 算法是依据其它网页链接到该网页的次数来决定该网页的重要性，在此之前，搜索引擎经历了基于分类目录的方法和基于文本检索的方法。Google 搜索引擎之前，网页排名算法大多按照网页被访问量排名，但这种排名方法受抽样的样本数据集的差异性而显著变化，同时这种排名方法需要不断的统计被访问量，而 Google 公司只需要每一个季度更新 PR 值并排名即可，更加准确方便。

1.1 PageRank 算法模型

在 PageRank 算法模型中，S.Brin 和 L.Page 两人将整个网络看成一个有向图，每一个网页看成是一个结点，而结点与结点之间的有向边代表从一个页面有一个链接指向另一个界面。

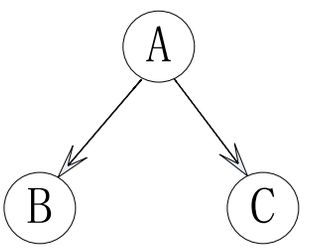

图 1 网页链接示意图

图 1 展示了由三个网页组成的网络图，其中 ABC 三个节点代表三个不同的网页，A 网页内含有一个指向 B 网页的链接和一个指向 C 网页的链接，B 和 C 网页均无向外指向的链接，PageRank 算法就是通过这样一张图来表示整个网络中的网页与网页之间的关系。

下面，将以图 2 为例讲解 PageRank 计算 PR 值的全部过程，以四个网页节点为例。

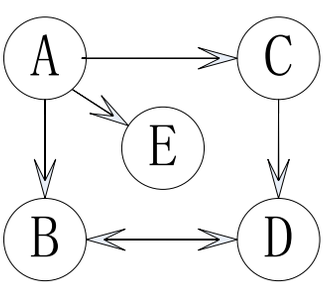

图 2 网页链接示意图

1. 将图 2 中，从链接页面到被链接页面的指向在 N×N 的矩阵 H 中表示为 1，其它情况表示为 0，例如节点 A 到节点 B 和 C 在矩阵中的表示均为 1，而节点 C 到节点 A 的矩阵表示为 0。规定 h_{ij} 表示矩阵 H 中第 i 行第 j 列的值，其中，

$$\begin{cases} h_{ij}=1, \text{如果节点i 指向节点j} \\ h_{ij}=0, \text{其它情况} \end{cases}.$$

依此得出图 2 的 N×N 邻接矩阵 H：

$$H=\begin{pmatrix} 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

2. 在现实情况中，如果用户访问一个页面，那么他可能存在的两种操作为，关闭该界面或者访问界面中某一个链接从而进入其它界面，在图 2 中，对于界面 A 而言，用户在进行跳转操作时，存在可能性去访问界面 B 也存在可能性去访问 C，因此界面 A 去访问 B 和 C 的概率各为 $1/2$。因此构建概率矩阵 S，其中元素 s_{ij} 表示从网页 i 跳转到网页 j 的概率。由此得到概率矩阵 S

$$S=\begin{pmatrix} 0 & 1/3 & 1/3 & 0 & 1/3 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

经观察发现，存在一个网页 E 并没有向外链接的接口，邻接矩阵 H 和概率矩阵 S 中 E 节点那一行均为 0. 这个网页在现实中可能是一个人写的一篇原创博客，没有向外的链接或者其它情况。因此对 S 矩阵做如下处理

$$\bar{S}=S+de^T/n$$

其中 e^T 是一个全 1 的行向量，n 为页面个数，d_i 在页面 i 无向外链接的时候为 1，有向外链接时为 0. 由此得到随机矩阵 $\bar{S}$

$$\bar{S}=\begin{pmatrix} 0 & 1/3 & 1/3 & 0 & 1/3 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \end{pmatrix}.$$

由 Google 矩阵

$$G=\alpha\bar{S}+(1-\alpha)/nee^T$$

其中 α 为阻尼系数，一般规定为 0.85，计算得出

$$G=\begin{pmatrix} 3/100 & 47/150 & 47/150 & 3/100 & 47/150 \\ 3/100 & 3/100 & 3/100 & 22/25 & 3/100 \\ 3/100 & 3/100 & 3/100 & 22/25 & 3/100 \\ 3/100 & 22/25 & 3/100 & 3/100 & 3/100 \\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \end{pmatrix}.$$

3. 通过处理[10]，使得 PR 值在 10 以内收敛，得到 PR 值为：

$$PR=(0.19186, 2.11598, 0.24622, 2.19973, 0.24622).$$

由此得到网页的重要性排序为：

网页	A	B	C	D	E
排名	5	2	3	1	3

表 1 网页排名

2 PageRank 算法的改进

在 PageRank 算法应用后的短短一年里，就出现了很多对 PageRank 算法的改进，其中比较著名的就是斯坦福大学的 Taher Haveliwala，他提出了主题敏感的 PageRank 算法。

PageRank 算法在实际应用的过程中暴露了一些问题，当用户搜索一个名词时，搜索引擎返回给用户的是一个按照 PR 值排序的网页结果，而这些网页结果中包含着用户不需要的网页。举个例子，当用户搜"C"这个字母时，对于程序员而言他想得到的结果可能是和“C 语言”相关的网页；对于关注体育赛事的用户而言他想得到的可能是关于“C 罗”的相关消息；对于一个爱车的发烧友而言，他想得知的可能是“奔驰 C 级”的相关信息。

图 3 搜索关键字'C'出现的结果

通过图 3 的搜索结果发现，作为一名软件工程的学生，我在百度搜索关键字’C’时，除了第一个为广告以外，紧接着就是和 C 语言相关的网页，这说明国内著名搜索引擎是采用了主题敏感的搜索算法的。

2.1 主题敏感的 PageRank 算法模型

主题敏感的 PageRank（以下简称 TS-PageRank），是参考网站（www.dmoz.org）中给出的 16 个大的主题类别来实现的，这 16 个大的主题类别包括艺术、计算机、游戏、健康、运动等，每一个大的主题下又划分有很多个小的主题。

算法通过给予一个网页关于 16 个主题的分向量来决定用户搜索时返回给用户的页面有哪些，例如，一

个界面的一个向量$\vec{R}=\left(a_1,a_{2,}a_3,a_4\cdots\right)$其中每一个$a_i$对应着一个主题，那么分量越大，就说明该网页和分量对应的主题相关程度越高，当用户之前一直在搜索跟 C 语言相关的页面时，搜索引擎就明白用户想要的结果是和计算机相关的，那么算法在返回结果时，就会从页面排名中抽取和计算机相关的页面重新组成一个结果返回给用户，而这一现象在 web 中叫做防止“主题漂移”。这样做的好处也是显而易见的，用户将搜索到自己希望得到的结果，而不是一个单单按照排名顺序而展现出来的搜索结果。

2.2 其它 PageRank 改进算法

在之后的几年里，华盛顿大学的 Matthew Richardson 和 PedroDominggos[5]提出结合链接和内容信息的 PageRank 算法（以下简称 MP-PageRank）。国内上海交大的宋聚等人提出结合网页中的 HTML 语言内容进行改进，通过为不同的 HTML 标签赋予不同的权重，依此来改进 PageRank 算法。北京大学也提出了基于 HTTP 协议，通过记录页面最近一次被修改的时间，从而将老旧界面沉淀下去。再到 06 年浙江工业大学的黄德才和戚华春两人在 MP-PageRank[6] 算法之上，提出了基于对两个网页相似程度的改进算法，通过描述相似程度，构造相似度矩阵，给出一个改进的 PR 值算法，有效的遏制了主题漂移现象。

2.3 Personalized PageRank

2003 年斯坦福大学的 Glen Jeh 和 Jennifer Widom[7] 提出了著名的个性化 PageRank 算法（简称 PPR），这个算法在随后的十多年里随着推荐系统以及图数据挖掘的兴起而被广泛引用。基于 PPR 算法的各种改进数不胜数，直到今天，PPR 还是一个主要的算法来衡量推荐系统和数据挖掘的一个重要算法。

简单来说，PPR 算法和 PR 算法不同的地方就在于，PR 算法是假设所有从 A 界面链接出去的每一个界面平分概率 1，也就是说，如果 A 界面向外链接 10 个界面，那么每一个界面被链接到的概率为 0.1。这样的概率只是理想情况下的概率，假设用户对于这 10 个链接没有偏好，而实际情况是，用户对于界面有自己的偏好。因此 PPR 算法让用户来指定权重，例如从 10 个界面中选出 5 个用户认为重要的界面去平分概率 1，那么这 5 个界面被访问的概率分别为 0.2，其余五个界面被访问的概率设置为 0。

2.4 Personalized PageRank on MapReduce

最近正在清华参加清华-谷歌人工智能学院开幕式的 Jeff Dean 在开发了 MapReduce 框架之后，斯坦福的 Bahman Bahmani 和微软的 Kaushik Chakrabarti[8] 提出了基于 MapReduce 的 PageRank 算法，使得 PPR 算法和 MapReduce 结合起来，更加快速的计算结果。将 PPR 算法应用在图数据挖掘的最大限制就在于，图数据的庞大性，全球超过 1 亿个网站包含难以计数的网页整合为一个图，Facebook 过 20 亿的用户整合为一个图，用 PPR 算法去挖掘这么庞大的数据是难以实现的，MapReduce 的出现使得 PPR 算法在数据挖掘领域成为了可能。

3 PageRank 算法在图数据挖掘中的应用

3.1 近似的个性化 PageRank 算法

近似的个性化 PageRank 算法（Approximate Personalized PageRank，以下简称 APPR 算法），是在 PPR 算法遇到困难后，一系列学者提出的算法。前文已经提到，PPR 算法就是在 PageRank 算法之上，有用户规定一些页面的重要性，然后基于这些页面分配概率。Google 公司在此之前只需要每过四个月重新的 PR 向量进行排序即可，这是一种离线的计算，对于实时性无要求。而现在需要实时的根据用户的喜好和行为来实时的计算 PR 值，这意味着需要大量的计算量，因此学者提出 APPR 算法用来减轻计算 PPR 向量的难度，包括 Jef 等人提出的基于 hub 的算法，Kamvar[9] 等人提出的基于块的算法，以及通过降低 PPR 向量计算的精度的近似算法，最终经过时间的沉淀，基于降低 PPR 向量计算精度的算法被人们广泛使用。

3.2 APPR 算法在图数据挖掘领域的应用

17 年，斯坦福大学的 SNAP 小组的 Hao Yin 博士提出基于 motif 的全局图聚类算法，将生物领域的 motif 使用在 APPR 算法上，结合南加州大学滕尚华教授（时为波士顿大学博士生）在 2008 年提出的局部图聚类算法，提出了 MAPPR 算法来实现超大规模图的局部聚类的实现，这也是 APPR 算法在图数据挖掘领域中的一个新的方向，不同于以往的将每条边每个点看成一个独立个体，算法借用生物学上的概念，将 motif

这个概念引入进来，把每三个点看成一个整体进行处理。通过数学上的严谨证明了算法的正确性，进而引出 APPR 算法在数据挖掘领域中的重要性。

4 总结

从 1998 年提出的 PageRank 算法在网页排名上应用的风靡，到如今 PageRank 算法早已不再受限于网页排名这一个应用。20 年的时间里，PageRank 就像是一棵树的结点，不断地开枝散叶，经过时间的沉淀，高效的改进算法最终成长地粗壮，低效的算法最终被淘汰，最终衍生出一条粗壮的根茎，从 PageRank 算法到个性化 PageRank 算法（PPR），再到近似的个性化 PageRank 算法（APPR），再到近期的基于 motif 的近似个性化算法（MAPPR），PageRank 算法正逐渐从 web 应用领域向推荐系统和图数据挖掘领域发展。可以想象，在未来 PageRank 算法将有更广阔的应用前景。

参考文献：

[1] S Brain, L Page. The PageRank Citation Ranking: Bringing Order to the Web. Standford Digital Libraries Working Paper, 1998.

[2] Haveliwala T. Topic-Sensitive PageRank. Proceeding of the Eleventh International. World Wide Web Conference, 2002.

[3] Hao Yin. Local Higher-Order Graph Clustering. KDD, 2017.

[4] D. A Spielman and ShangH Teng. A Local Clustering Algorithm for Massive Graphs and its Application to Nearly-Linear Time Graph Partitioning. Siam Journal on Computing, 2008.

[5] M Richardson, P Dominggos. The Intelligent Surfer: Probabilistic Combination of Link and Content Information in PageRank. Neural information Processing System. 2002.

[6] 黄德才，戚华春. PageRank 算法研究. 计算机工程. 2006.

[7] G Jeh, J Widom. Scaling Personalized Web Search. World Wide Web. 2003.

[8] B Bahmani, K Chakrabarti. Fast Personalized PageRank on MapReduce. Acm Sigmod International Conference on Manageme of Data. 2011.

[9] Kamvar S, Haveliwala T. Manning C. Exploiting the Block Structure of the Web for Computing PageRank. Stanford University. 2003.

附录：

[10] PageRank 计算 PR 值的 java 源码在 https://github.com/hacksun/PageRank. 下载。

```
public class PageRank {
/**
 * 矩阵 g 乘以矩阵 p
 * @param g
 * @param p
 * @return 矩阵 g 乘以矩阵 p 的结果矩阵
 */
private static double[] multiMatrix(double[][] g, double[] p){
    double[] multiResult = new double[p.length];
    for(int i=0; i<g.length; i++){
        double rowResult = 0.0f;
        for(int j=0; j<g.length; j++){
            rowResult+=g[i][j]*p[j];
        }
```

```
            multiResult[i] = rowResult;
        }
        return multiResult;
    }
    /**
     * 根据初始矩阵计算真正的 Google 矩阵
     * @param 初始矩阵
     * @param weight
     * @param oneMatrix
     * @return 真正的 Google 矩阵
     */
    private static void getGoogleMatrix(double[][] transitionMatrix, double weight){

        for(int i=0; i<transitionMatrix.length; i++){
            for(int j=0; j<transitionMatrix.length; j++){
                transitionMatrix[i][j] *= weight;
                transitionMatrix[i][j] += (1-weight)/transitionMatrix.length;
            }
        }
    }
    /**
     * 如果 pageRankN=pageRankN_1,返回 true；否则，返回 false

     * @param pageRankN
     * @param pageRankN_1
     * @return
     */
    private static boolean compareMatrix(double[] pageRankN, double[] pageRankN_1){
        for(int i=0; i<pageRankN.length; i++){
            if(pageRankN[i]-pageRankN_1[i]>0.0000001){
                return false;
            }
        }
        return true;
    }
    /**
     *
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
         double[][]
         transitionMatrix={{0,0,0,0,1/5f},{1/3f,0,0,1,1/5f},{1/3f,0,0,0,1/5f},{0,1,1,0,1/5f}
         ,{1/3f,0,0,0,1/5f}};
```

```
        double[] p={1,1,1,1,1};
        double weight = 0.85f;
        getGoogleMatrix(transitionMatrix, weight);
        double[] pageRank = multiMatrix(transitionMatrix, p);
        while(!compareMatrix(pageRank, p)){
            p = pageRank;
            pageRank = multiMatrix(transitionMatrix, p);
        }
        for(int i=0; i<pageRank.length; i++){
            System.out.println(pageRank[i]);
             }
          }
    }
```