PageRank 算法和 MapReduce 框架

王家蔚

2020 年 7 月 27 日

摘要

对于一个特定的查询，搜索结果的排名取决于两方面的信息，第一个是关键字与网页的相关性，第二个是网页本身的质量，本文着重于第二点，也就是介绍一种被证明行之有效的评估网页质量的办法及其并行计算策略。

如今的全球搜索引擎巨头 Google，最早的革命性发明是一款名为 PageRank 的算法，它是用来衡量网页本身的重要性的，核心思想并不复杂，发明人将互整个互联网想象成一张相互关联的有向图，如果图里的每个结点如果一个网页被其他很多网页链接，那么就说明该网页很重要，是受到普遍认可的，那么它的排名就要高。[1]

算法具体的实现方法涉及到图论和线性代数，为了更快地进行稀疏矩阵的运算，Google 还专门开发了并行计算工具 MapReduce，本文旨在先清晰地说明其中的技术细节，然后用 Python 和 C++ 程序重现这一经典算法，并且针对常见关键字进行测试。

1 引言

在上世纪末，互联网上的网页数量快速增多，传统的搜索引擎，例如 AltaVista 和 Yahoo，疲于应付新的趋势，主要原因是他们的算法是根据关键词分类所收录网站的。[2] 但是对于应对大规模的查询业务，很少有学术研究涉及到相关工程领域。Google 的这项技术在 1998 年前后使得搜索引擎的结果在相关性方面取得了质的飞跃，具体表现在改善了网页的排序上，同时也完美地支持了高并发状态，从而一举超过 Yahoo 成为行业领头羊。

互联网发明最初是为了更高效地传递学术信息，在发展过程中，商业信息逐渐变成了主流内容，商业网站的占比从 1993 年的 1.5% 上升到 1997 年的超过 60% [3]，但是学术内容和商业网站之间存在着明显的差别，商业网站可以通过程序自动创建大量指向特定网站的链接，使自己的网站排名提高以谋求利润。所以 Google 在满足学术研究需求的同时，兼顾商业运营，以一种截然不同的方式对商业网站排名。Google 这个项目的最终目的是要建立起一个覆盖全球互联网的网页索引大全。

2 文献综述

PageRank 这方面的文章和工作主要由谢尔盖 · 布林和拉里 · 佩奇为代表的谷歌公司工程师完成，引用来介绍这个算法的文章大多也由他们亲自撰写。

不过在他们之前，已经有了一些开发文本链接结构系统的工作，Pitkow 已经尝试用网页链接分析万维网的环境 [4]，Spertus 把拥有链接结构的系统用在了数据挖掘上 [5]，此外还有人从不同的角度出发，也在对网页质量进行评估 [6]。当时的搜索引擎巨头也是把链接结构当作重要指标来优化搜索结果，但是这些项目都是简单地以链接数目作为评判标准，忽略了不同网站的链接权重并不相等这一事实，后面会仔细介绍这点。

3 研究方法

3.1 PageRank 算法

3.1.1 核心思想

我们可以假设现在所有的网页都是图上的节点，网页之间的链接是有向边，每个节点都有一定数量的入链接 (inedges) 和出链接 (outedges)，被链接数大的网页很大概率就是更加重要的网页。[7]

此外也要考虑到这样一种情况，某一个网页被

连接数不多，但都是被比较重要的网页所链接，比如如下图所示有一个网页 C，只有两个网页 A 和 B 链接了它，但是这个网页 B 相当重要，收到了 1000 个其他网页的链接，那么网页 A 的重要性就不是单纯一个被链接数量能够描述的。

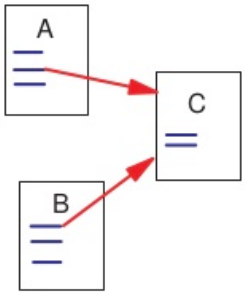

图 1: 特殊文件链接情况

为了通过链接更加合理地度量网页的重要性从而解决这个问题，不同网页被赋予了不同的权重，换句话说，如果一个网页想要得到高的排名，那么链接它的网页排名也需要尽可能地高。

3.1.2 随机浏览者假设

首先我们假设有一个网页的浏览者，他随机点击网页上的一个链接，一直持续下去。但是有可能网页上的链接是循环往复的，所以按照假设，他会一直持续点击这些链接。事实上，并不会有人真的这样做，如果碰到循环的情况，真实世界的浏览者肯定会重新打开一个新的网页，而且每次用户不一定会一直点击到没有链接为止，每当打开一个新网页，浏览者的下一步行为都有可能是打开一个不被现在网页直接链接的随机网页 [8]

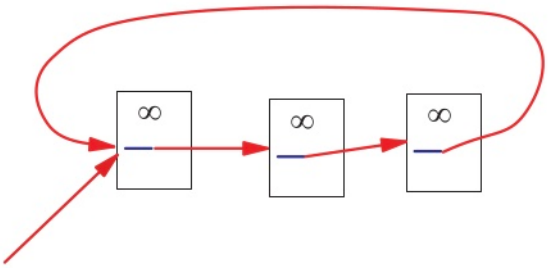

图 2: 循环的网页链接

为了在算法中模拟出在这个情况下的用户行为，我们需要一个 N 维向量 E 来存储随机浏览者转跳到不同网页的概率，其中的 N 是收录的网页数量。一般来说，我们继续假定每个网页被点击的概率都是相等的，当然这是一个可以个性化的参数，而且直接影响到收敛速度。关于向量 E 的进一步说明，我把它放在了第四部分。这小节假设的情况是为了给接下来介绍计算得分的方法作铺垫。

3.1.3 计算方法

假设网页 A 被 n 个网页所链接，这些网页分别记为 $T_1, T_2, \cdots, T_n$，令 $C(T_i)$ 表示网页 T_i 的出链接数量，接着我们还需要一个衰减系数 $d \in (0, 1)$，这是为了让每个网页在开始迭代的时候都有一定的保底分数。那么网页 A 的重要性得分可以按照以下的公式得出

$$PR(A) = (1-d) + d(\frac{PR(T_1)}{C(T_1)} + ... + \frac{PR(T_n)}{C(T_n)})$$

这里的 d 一般取 0.85。我们可以借助下图来直观的理解不同链接权重的含义

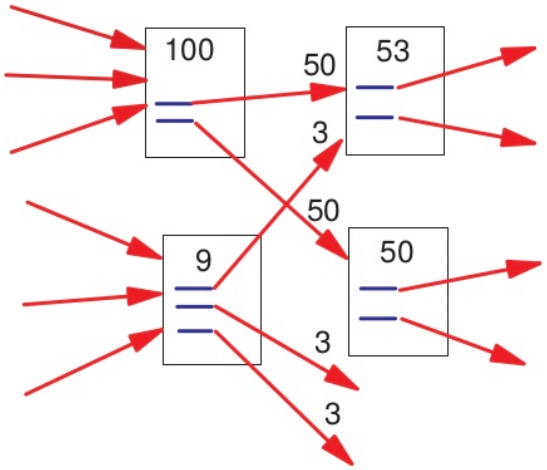

图 3: 带权重的链接

之后为了求出网页 A 的得分，我们要解决的问题是获取 $T_1, T_2, \cdots, T_n$ 得分，这些网页又会有链接到它的其他网页。如果不存在循环链接的情况，那么从最底层的网页分数依次网上累加；如果存在如随机浏览者假设中循环链接的情况，那么就不存在最底层的网页，无法自底向上加出总分。为了解决这个问题，PageRank 算法把问题转变成了一个二维矩阵相乘的问题，并用迭代的方法最终解决了它。

具体来看这个迭代的过程，首先我们把所有网页都给予从 1 到 N 的唯一编号，把它们的对应的最终得分记录在 N 维向量

$$B = [b_1, b_2, \cdots, b_N]^T$$

中，然后把链接结构用如下的矩阵

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ \vdots & & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mN} \\ \vdots & & & \vdots \\ a_{N1} & a_{N2} & \cdots & a_{NN} \end{bmatrix}$$

记录下来，其中的 a_{mn} 表示第 m 个网页指向第 n 个网页的链接数，迭代起点设置为

$$B_0 = [\frac{1}{N}, \frac{1}{N}, \cdots, \frac{1}{N}]^T$$

迭代公式设置为

$$B_{n+1} = AB_n + (||AB_n||_1 - ||B_n||_1)E$$

完整过程我用伪代码的形式给出

Algorithm 1 PageRank 伪代码

Input: 迭代收敛误差的容忍极限 ϵ
存储网页链接结构的矩阵 A
存储网页初始得分的向量 B_0
浏览者跳转概率向量 E
Output: 存储网页最终得分的向量 B

function PageRank(A, B_0, E, ϵ)
  $d = ||AB_0||_1 - ||B_0||_1$
  $B_1 = AB_0 + dE$
  $\delta = ||B_1 - B_0||_1$
  while $\delta > \epsilon$ **do**
    $B_{i+1} = AB_i$
    $d = ||B_{i+1}||_1 - ||B_i||_1$
    $B_{i+1} = B_{i+1} + dE$
    $\delta = ||B_{i+1} - B_i||_1$
  end while
  $B = B_{i+1}$
  return B
end function

简明扼要说明以下思路，首先要明确的输入和输出，既然是企图用迭代操作获取收敛值，那么就不能期望得到精确解，必须要设置一个误差限 ϵ, 其他的输入输出量前面都已经提到，这里不再赘述。之后先在循环外进行一次迭代，把 δ 初始化, 之后进入 while 循环，特别注意为了简化计算复杂度，向量间的距离用 1 范数给出。最后当 $\delta < \epsilon$ 时，就进行输出。

3.1.4 收敛证明

取 e 为所有分量都为 1 的列向量，定义矩阵

$$M = \alpha E + \frac{1-\alpha}{N} ee^T$$

那么上述的迭代过程就是 $B_n = MB_{n-1}$, 于是该问题就转为了一个 Markov 过程了。马尔科夫链的收敛条件如下 [9]

- M 为随机矩阵
- M 是不可约的
- M 是非周期的

以上条件均满足，所以 pagerank 是收敛的，且与初始值无关，即无论 B_0 取值如何，都不会影响最后各个网页的得分情况。

3.1.5 部分源代码

Listing 1: **生成评分的函数模块**

```
def pageRankGenerator(At = [array((), int64)],
    numLinks = array((), int64),ln = array((),
    int64),alpha = 0.85, convergence = 0.0001,
    checkSteps = 10):
   N =  len(At)
   M = ln.shape[0]
   iNew = ones((N,), float64) / N
   iOld = ones((N,), float64) / N
   done = False
    while  not done:
       iNew /=  sum(iNew)
        for step  in  range(checkSteps):
           iOld, iNew = iNew, iOld
           oneIv = (1 - alpha) *  sum(iOld) / N
           oneAv = 0.0
            if M > 0:
               oneAv = alpha *  sum(iOld.take(ln
                    , axis = 0)) / N
```

```
            ii = 0
             while ii < N:
                page = At[ii]
                h = 0
                 if page.shape[0]:
                    h = alpha * dot(
                            iOld.take(page, axis
                                = 0),
                            1. / numLinks.take(
                                page, axis = 0)
                            )
                iNew[ii] = h + oneAv + oneIv
                ii += 1
        diff =  sum( abs(iNew - iOld))
        done = (diff < convergence)
        yield iNew


def pageRank(linkMatrix = [[]],alpha = 0.85,
    convergence = 0.0001, checkSteps = 10):
   incomingLinks, numLinks, leafNodes =
        transposeLinkMatrix(linkMatrix)
    for gr  in pageRankGenerator(incomingLinks,
        numLinks, leafNodes,alpha = alpha,
        convergence = convergence,checkSteps =
        checkSteps):
       final = gr
    return final
```

3.2 MapReduce 框架

3.2.1 核心思想

从前面的算法可以看出，为了给新收录的网页评分，PageRank 算法反复做大规模的矩阵运算，而且计算量每次都会增加，很紧迫的需求就是切分矩阵实现并行计算。针对挑战，Google 给出的解决方案是一个叫做 MapReduce 的程序，它的核心设计思想是分治算法。[10]

将一个大任务拆分成若干个子任务，并且完成子任务的计算，这个过程叫做 *Map*，将中间结果合并成最终要的结果，这个过程叫做 *Reduce*。还有很多更加具体的问题需要考虑，比如平均分配负载，分布式存储数据，错误容忍。[11]

MapReduce 框架给出的是一个并行计算的通用解决方案，文件首先要输入 (input) 程序，然后进行切分 (split)，把每一个片段分配 (map) 到不同的工作站 (worker) 中，计算的结果分布的存储在本地，之后由远端的工作站 (worker) 逐个读取结果并且重新排序 (shuffle)，最后一步才是合并 (reduce) 输出最终的文件。具体的过程见下图

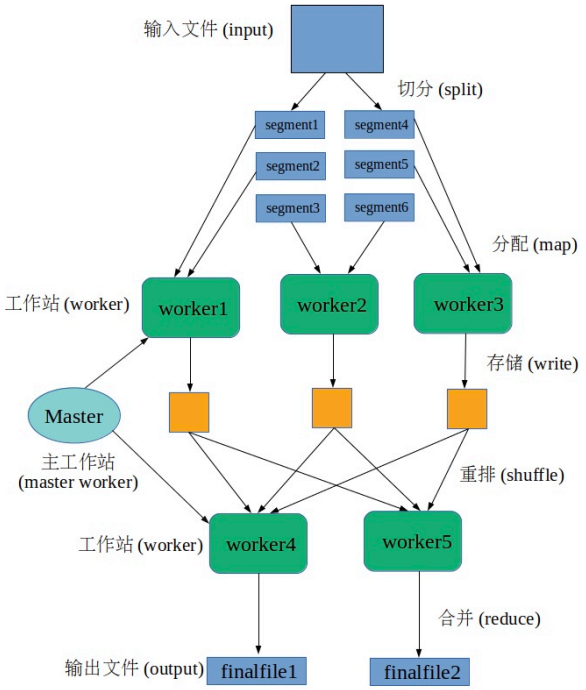

图 4: MapReduce 架构

此外还有一个需要强调的部分是，过程中还会产生一个主工作站 (master worker)，这个和其他工作站等级是一样的，只不过会做一些特殊事情，它会作为用户的代理来协调整个过程，用户就可以做其他事情。主工作站会让一个工作站去拿 1 号数据，另一个工作站负责拿 2 号数据等等，这就是分配数据的过程。抽象地说，主工作站是将 map 任务产的中间数据位置传送到 reduce 任务的管道，所以主工作站保存了 map 任务完成后产生的若干中间文件的位置信息包含大小，当然，map 任务结束的时候会更新位置和大小信息。这个信息会被逐步的推送到正在处理 reduce 任务的工作站。

3.2.2 矩阵运算

由于 PageRank 算法涉及到大规模矩阵并行计算，所以这里特别说明一下 MapReduce 在处理矩阵乘法上的应用。我们先讲理论抽象，假设我们有

规模为 $m \times n$ 的矩阵 A 和规模为 $n \times p$ 的矩阵 B，两者相乘得到矩阵 C. 就以 a_{11} 为例，它将会在 $c_{11}, c_{12}, \cdots, c_{1p}$ 的计算中使用。也就是说，在 Map 阶段，当我们从切分完毕的文件中取出一行记录时，如果该记录是 A 的元素，则需要存储成 p 个 $<key, value>$ 对，并且这 p 个 key 互不相同；如果该记录是 B 的元素，则需要存储成 m 个 $<key, value>$ 对，同样的，m 个 key 也应互不相同；但同时，用于存放计算 c_{ij} 的 $a_{i1}, a_{i2}, \cdots, a_{in}$ 和 $b_{1j}\ b_{2j}, \cdots, b_{nj}$ 的 $<key, value>$ 对的 key 应该都是相同的，这样才能被传递到同一个 Reduce 中。

举个例子，我们有规模为 3×2 的矩阵 A 和规模为 2×4 的矩阵 B

$$C = AB = \begin{bmatrix} 5 & 3 \\ 10 & 2 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} 3 & 0 & 7 & 11 \\ 9 & 2 & 6 & 8 \end{bmatrix}$$

在计算的时候，我们需要将矩阵 A 转换成下面这张键值表

key	value	key	value	key	value
1,1	a,1,5	2,1	a,1,10	3,1	a,1,1
1,1	a,2,3	2,1	a,2,2	3,1	a,2,1
1,2	a,1,5	2,2	a,1,10	3,2	a,1,1
1,2	a,2,3	2,2	a,2,2	3,2	a,2,1
1,3	a,1,5	2,3	a,1,10	3,3	a,1,1
1,3	a,2,3	2,3	a,2,2	3,3	a,2,1
1,4	a,1,5	2,4	a,1,10	3,4	a,1,1
1,4	a,2,3	2,4	a,2,2	3,4	a,2,1

这里的 key 是指这个数据用来计算矩阵 C 中哪个位置的值，比如第一行第一列这个位置对应 $key = (1,1)$，计算过程涉及到矩阵 A 中的两个值，所以上表中同样的 key 也只对应两个 $value$。$value$ 第一部分表示它属于哪个矩阵，第二部分表示它是原矩阵这一行/列第几个元素，只有对应的元素才能相乘，第三部分才是用来计算的数值，其余部分都是为了用来定位的。同样的办法可以得到矩阵 B 的键值表，如下所示

key	value	key	value	key	value
1,1	b,1,3	2,1	b,1,3	3,1	b,1,3
1,1	b,2,9	2,1	b,2,9	3,1	b,2,9
1,2	b,1,0	2,2	b,1,0	3,2	b,1,0
1,2	b,2,2	2,2	b,2,2	3,2	b,2,2
1,3	b,1,7	2,3	b,1,7	3,3	b,1,7
1,3	b,2,6	2,3	b,2,6	3,3	b,2,6
1,4	b,1,11	2,4	b,1,11	3,4	b,1,11
1,4	b,2,8	2,4	b,2,8	3,4	b,2,8

后面重排 (shuffle) 这步将分布式存储的数据按照 key 值进行归类

key	values		key	values	
1,1	a,1,5	b,1,3	1,2	a,1,5	b,1,0
	a,2,3	b,2,9		a,2,3	b,2,9
1,3	a,1,5	b,1,7	1,4	a,1,5	b,1,11
	a,2,3	b,2,6		a,2,3	b,2,8
2,1	a,1,10	b,1,3	2,2	a,1,10	b,1,0
	a,2,2	b,2,9		a,2,2	b,2,2
2,3	a,1,10	b,1,7	2,4	a,1,10	b,1,11
	a,2,2	b,2,6		a,2,2	b,2,8
3,1	a,1,1	b,1,3	3,2	a,1,1	b,1,0
	a,2,1	b,2,9		a,2,1	b,2,2
3,3	a,1,1	b,1,7	3,4	a,1,1	b,1,11
	a,2,1	b,2,6		a,2,1	b,2,8

最后一步是归并 (reduce)，依次对每一个 key 进行加总输出结果，这里就不列表进行展示了。

3.2.3 部分源代码

Listing 2: **Map 过程模块**

```
 class Mapper {
public:
  virtual void Map(const string& key, const
      string& value) = 0;
  virtual  int Shard(const string& key,
      int num_reduce_shards);
protected:
  virtual void Output(const string& key, const
      string& value);
  virtual void OutputToShard( int reduce_shard,
      const string& key, const string& value);
  virtual void OutputToAllShards(const string&
      key, const string& value);
  const string& CurrentInputFilename() const;
  const string& GetInputFormat() const;
```

```
const string& GetOutputFormat() const;
 int GetNumReduceShards() const;
 bool IsMapOnly() const;};
```

4 讨论

PageRank 算法为了解决循环链接的问题，3.1.2 节引入了一个描述转跳概率的矩阵 E，之前我们假设所有网页被转跳到的概率是一模一样的，实际上数值是被重新设置过的。在测试中发现，一些写有著作权，免责申明的网页被广泛链接，还有邮件服务器等没有实际内容的网页评分也很高。解决方案是只给各大网站的根目录给予一定的概率分布，另外对用户设置的家网页更高的概率。

5 结论

个人认为数学软件的最大作用，是根据你的想法，以最快速度做出一个模型，用来检验结果的有效性。使用者不需要重复造轮子，Matlab 已经将绝大多数现在证明可行的办法，就拿课堂上和我作业里的例子来说，比如多层卷积神经网络和 KDTree，线性回归等都已经完全封装在一个函数里，任何人只要理解了算法的原理，读一下自带文档的接口参数，就可以迅速得到想要的结果。完善友善的内置文档，让使用者不需要到处寻找技术博客查询某个特定功能的函数，大大提高了。

Matlab 以接近自然语言的语法和丰富的扩展包为特点，配合上高效的矩阵计算，能解决大多数数学研究设计到的编程需求。控制系统设计与分析、金融工业建模和仿真模拟等领域都是它极度擅长的。高度封装性优点在于只要看文档明确输入输出的格式，完全不需要花费额外的时间去调试中间繁琐的实现过程，不仅如此，像 Matlab 这样成熟的商业软件在结果的正确性上也完全可以信任。总而言之，可以让数学研究者更多地专注于数学本身，尽可能少地与编程技巧较劲。

Matlab 这样大型的商业软件的优点很明显，局限性也不小。首先它的解释器就已经超过了 500M，如果要加上一些常用的扩展包，起码需要 10G 的存储空间，此外运行效率比较低，所以几乎只能将.m 程序放在个人计算机上运行，不能放在一些小型的嵌入式系统上。由于不是开源的软件，所以也不能才剪掉某个特定程序不需要的部分，更不能做一些底层的优化。此外，对于软件工程师来说，长时间使用可能会导致变成水平下降；同样也不适合计算机编程领域的初学者，因为它几乎没有需要进行硬件操作的地方，就连并行的实现也轻而易举，gpu 工具箱是把整个 cuda 打包了。

综上，Matlab 语法简单，文档易懂，封装性好，扩展包多，所以是应用数学建模领域和学术研究的一个良好测试工具，相比于开源的 Octave 在运行稳定性上有明显的优势。但是运行效率上存在劣势，在大型商业项目的开发上，肯定无法代替编译型语言。

参考文献

[1] The Anatomy of a Large-Scale Hypertextual Web Search Engine, Sergey Brin and Lawrence Page

[2] On Top of Tides，Jun Wu

[3] Web Growth Summary, Matthew Gray

[4] Characterizing World Wide Web Ecologies, James E. Pitkow

[5] Parasite: Mining structural information on the web, Ellen Spertus

[6] Evaluating quality on the net, Hope N. Tillman

[7] The PageRank Citation Ranking: Bringing Order to the Web, Lawrence Page, Sergey Brin, Rajeev Motwani, Terry Winograd

[8] PageRank and The Random Surfer Model, Prasad Chebolu and Páll Melsted

[9] Convergence of Markov Chains in Information Divergence, Peter Harremoës and Klaus Kähler Holst

[10] Beauty of Mathematics, Jun Wu

[11] MapReduce: Simplified Data Processing on Large Clusters, Jeffrey Dean and Sanjay Ghemawat