链接分析–PageRank 算法分析实现及优化

张宇豪 2120190498[*], 周宝航 2120190442[†]
王嘉贤 2120190430[‡]

2019 年 12 月

1 摘要

互联网时代带给人们生活最大的改变是，通过搜索引擎进行高效准确的 Web 搜索。尽管 Google 并不是最早的搜索引擎，但它却史无前例地成功解决了 Web 作弊问题。Google 搜索引擎的核心正是 PageRank 算法。PageRank 算法，是 Google 的创始人拉里 · 佩奇和谢尔盖 · 布林于 1998 年在斯坦福大学发明了一种高效的链接分析方法。该算法基于一种假设：更重要的页面往往更多地被其他页面引用（或称其他页面中会更多地加入通向该页面的超链接）。PageRank 算法用于求解对搜索结果的网页排名，由于其高效性，它至今还在被各大搜索引擎使用。

本文主要进行 PageRank 算法的核心原理探讨以及实现优化。在实现算法基础功能的工作之上，引入随机游走因子，对于算法进行优化，从而解决 Web 数据中存在的 dead ends 和 spider traps 问题。此外，本文进一步实现了 PageRank 算法的 Block-Stripe Update 分块优化，以达到空间复杂度的优化。

关键词：PageRank 算法；链接分析；随机游走；Block-Stripe Update 分块优化

[*]网络空间安全学院 计算机技术
[†]计算机学院 计算机科学与技术
[‡]计算机学院 计算机科学与技术

2 引言

互联网时代，人们越来越依靠网络来获取知识。搜索引擎对于用户给出的查询，在海量 Web 网页寻找与查询匹配的内容，并在尽可能短的时间内给出用户反馈。用户通常关心搜索引擎搜索结果中的排名考前的结果，因此，在此需求下，搜索引擎需要对于海量 Web 数据进行质量分析。Google 公司的拉里 · 佩奇和谢尔盖 · 布林于 1998 年在斯坦福大学提出 PageRank 算法，通过对于 Web 网页间的链接分析，检索出其中与搜索相符的高质量网页，成功地解决了 Web 中广泛存在的垃圾页面和链接作弊的问题，极大地提高了搜索引擎的性能。

本次实验实现了 PageRank 算法基础功能，并在此之上通过随机游走算法解决了 dead ends 和 spider traps 问题，并通过 Block-Stripe Update 分块优化，保证了算法在大规模数据的情况下仍能有较高的性能。

3 核心问题与解决方案

本节将针对 PageRank 算法面对的核心问题以及解决方案给出算法分析。

3.1 连接分析与网页评分

搜索引擎根据用户的查询关键词，给出内容相关的 Web 页面。PageRank 算法通过 Web 页面之间的链接关系，建立相互投票的评分机制，通过计算页面最终的得分来衡量页面的质量。

具体来说，PageRank 让链接来” 投票”。一个页面的 “得票数” 由所有链向它的页面的权重（重要性）来决定，到一个页面的超链接相当于对该页投一票。一个页面的 PageRank 是由所有链向它的页面（“链入页面”）的权重经过递归算法得到的。一个有较多入链的页面会有较高的权重，反之则页面的权重较低。

3.2 网页陷阱与随机游走算法

朴素的 PageRank 算法在 Web 网页结构良好的环境下可以正常运行，通过迭代可以对不同的网页给出合理的打分。然而，研究表明，现实中的 Web 网页结构常常出现网页个体或网页群体没有出向链接，即网络中的

dead ends 和 spider trap。PageRank 算法经过迭代之后，全体系统的权重会被以上两种 Web 网页结构吸收，其余页面的权重会趋于 0，这使得计算得出的结果失去意义。

基于以上的问题，Google 对于朴素的 PageRank 算法提出改进策略。新的算法增加了随机游走因子，对于 Web 网页间的行为进行了更加细致的建模。朴素的 PageRank 认为网页之间的跳转只能通过彼此间的链接来实现；而加入随机游走因子后，算法认为用户会以一定概率随机打开任意一个网页，例如，用户直接在地址栏中输入某 Web 网页的网址。随机游走因子使得流入 dead ends 或 spider trap 中的权重能够以一定概率跳出该结构，保证系统的权重不会困于局部网页结构中，提高了算法的鲁棒性。

3.3 大规模数据与分块优化

互联网的发展带来的是网络中页面的爆炸式增长，这导致在现实情景下，PageRank 算法的性能受到存储介质空间的限制。内存空间，甚至单台机器的硬盘存储空间，难以支持庞大的数据以及 PageRank 的复杂运算，因此，分块化的 PageRank 算法优化显得格外重要。

为了解决存储空间有限的问题，可以采用分块的思想。即把进行解耦合并分块处理。每个块内部先进行一次处理，之后把处理之后的结果，通过整合、处理，得到最终的结果。对于每一块，通常认为是单台计算机可以存储并且处理的，而最终的整合信息，也认为是单台计算机可以综合处理的。这样，就能使大规模数据集被多台计算机配合处理，从而得到最终结果。

工程领域中，常采用 MapReduce 分布式集群的模型架构。本文的实验方案中，对于输入数据进行了划分，使每部分数据适应运行环境的内存大小，已达到 PageRank 算法的分块化处理。

4 PageRank 算法分析

本节针对 PageRank 朴素算法、随机游走算法以及分块化算法给出公式定义。

4.1 朴素 PageRank 算法分析

本小节针对朴素 PageRank 算法给出分析。在理想的 Web 网页结构中，即不存在 dead ends 以及 spider trap 结构的前提下，PageRank 算法可以

可以概括为

- 如果一个网页被大量网页链接到，则该网页质量较高，拥有较高的 PageRank 权重
- 如果一个高 PageRank 权重的网页中包含一条链出至其他页面的链接，则被链接的网页会拥有较高的 PageRank 值。

基于以上的假设，利用 “权重投票 “的方式描述网页间的链接关系，并以此计算网页相应的 PageRank 权重。例如图 1中的网络结构示意图，其中 i, j, k 表示 Web 中的三个网页，r_i, r_j, r_k 分别表示网页的 PageRank 权重, 有向箭头表示网页间的链接。

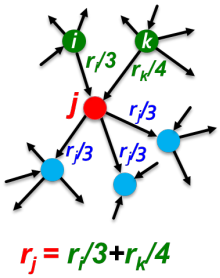

图 1: 网络结构示意图

每个网页的 PageRank 权重值由**指向它的网页的 PageRank 权重值及这些网页的出度**共同决定；同时这个网页的 PageRank 权重值及其出度又会影响它所指向的网页的 PageRank 权重值。由此，可以得出网页 j 的 PageRank 权重值 r_j，其公式化定义为：

$$r_j = \sum_{i \to j} \frac{r_i}{d_i}, \tag{1}$$

其中 d_i 表示网页 i 的出度，i 为所有链接至 j 的网页。

接下来考虑 PageRank 算法的初始化和算法结束的条件。在初始化阶段，假设用户以相等的概率随机访问 N 页面中的任意一个，故为每个网页分配均等的初始 PageRank 值 $\frac{1}{N}$。同时，保系统的总权重值之和为 1，保证了算法的收敛性。初始化完成后，根据公式 1可得到 N 个线性无关的方程，解方程后得到唯一的一组解即为 N 个网页的 PageRank 值。当 N 规模较小时，线性方程组求解的算法代价是可接受的；但 N 的规模增大时，这样

的算法在时间代价上是不可接受的。因此，PageRank 算法通常采用迭代法进行计算。

领 R 为所有 N 个网页的 PageRank 值组成的列向量，令 A 为归一化后的网页间的邻接矩阵。根据定义有：

$$R = cA^T R \Leftrightarrow c^{-1}R = A^T R. \tag{2}$$

根据线性代数中有关特征值和特征向量的理论，R 是矩阵 A^T 的特征值 c^{-1} 对应的特征向量。通过迭代，使 R 向量收敛，即得到最终结果。此时 R 向量中的每一维即为每一个网页的 PageRank 权值。根据计算精度条件的限制，我们认为每次迭代后，向量 R 的更新幅度低于阈值 10^{-10} 时，可认为算法已收敛。

4.2 随机游走算法

朴素 PageRank 算法在网络结构中出现 dead ends 或 spider trap 的情况时，会出现系统的权重值被部分网页全部持有，网络中大部分网页的权重趋向于 0 的情况。如图 2所示，网页 m 没有出向的链路，因此为 dead ends 网页。一种常见的链接作弊手段就是构造类似 dead ends 或 spider trap 的结构，骗取较高的 PageRank 值，已推广自己的网页并从中谋取利益。

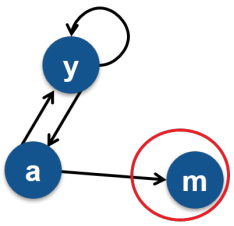

图 2: Dead ends 结点示意图

随机游走（Random Walk）算法的提出，可以有效解决这类问题。算法将到达网页 j 的路径划分成两类，第一种是通过指向该网页的链接访问；，并跳转到该目标网页；第二种是随机访问，即用户在地址栏中直接输入网页的网址。其中第二种访问方式就是随机游走算法的核心思想。

依照概率，随机游走算法将网页 j 的 PageRank 值划分为两部分，即链接传递（与朴素 PageRank 算法相同）和随机获取。修改后结点的 PageRank

计算公式为:

$$r_j = \sum_{i \to j} \beta \frac{r_i}{d_i} + (1-\beta)\frac{1}{N}, \tag{3}$$

其中 β 是进入随机游走的概率，通常为 0.85。加入随机走算法后，可以很大程度上缓解 dead ends 和 spider trap 对于 PageRank 算法权重更新的影响。当到达某个页面后，有 $1-\beta$ 的概率跳转到其他任意网页，使得系统整体的权重不会全部集中于局部的某些网页中。

4.3 分块优化算法

朴素 PageRank 算法还存在一个缺陷，即显示中的网页数量以达到千亿级别，在单台机器上难以其存储链接关系，更无法运行 PageRank 算法。因此，将海量的数据进行划分，使得划分后的任务能够满足处理设备的边界条件。

本文采用分块的思路，将大规模数据解耦为低依赖的数据块，单个处理设备仅需要完成某一块或几块的子任务。子任务全部完成后，在将分块的结果汇总成为最终的结果。如图 3为分块算法的示意图。

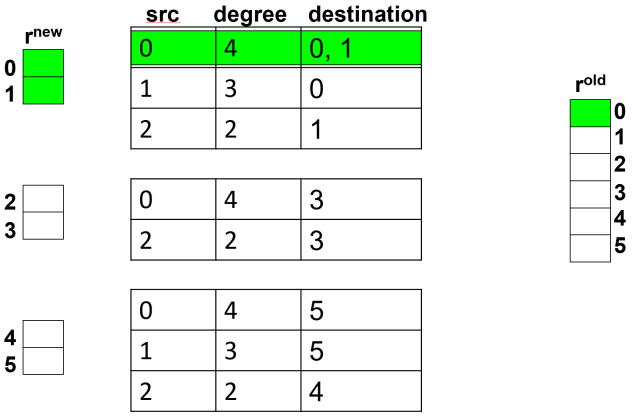

图 3: 分块算法示意图

分块算法解决了处理设备空间不足的问题，但由此带来的问题也十分明显。数据分块后，算法运行过程中需要多次对存储介质进行访问，大大增加了 I/O 开销，并且当块划分的个数越多，每个块的数据逐渐变小，I/O 代价更加不可忽视。因此，快优化算法需要在时间开销和空间开销中寻找平衡。

工程领域通常使用分布式服务器并配合相关框架，如 Hadoop，来实现分块算法。本文进行的实验通过将原始数据文件拆分为多个子文件来模拟数据分块的过程，拆分后的子文件能装载如运行环境中的内存介质。

5 代码分析

本节将对实现 PageRank 的部分核心代码进行分析。

5.1 数据存储结构

本小节分析实现 PageRank 算法使用的数据结构。首先读取文件中的链接信息，保存在矩阵中。

```
# 读取链接信息并保存入matrix字典中
matrix = {}
for s, d in data:
    s = int(s)
    d = int(d)
    if s not in matrix:
        matrix[s] = []
    if d not in matrix:
        matrix[d] = []
    matrix[s].append(d)

self.N = max(matrix.keys()) + 1
```

构建保存网页结点 PageRank 值的向量并初始化，*r_new* 代表当前时刻网页的 PageRank 值，*r_old* 代表上一时刻的 PageRank 值。

```
# 创建r_new向量
index = 0
with open('r_new.txt', 'w') as fp:
    while index <= sme.nodesize:
        fp.write(f"{index} {0.}\n")
        index += 1

# 创建r_old向量
index = 0
with open('r_old.txt', 'w') as fp:
    while index <= sme.nodesize:
        fp.write(f"{index} {1./sme.nodesize}\n")
```

```
        index += 1
        errors += 1. / sme.nodesize
```

5.2 矩阵分块算法实现

本小节分析矩阵分块算法的实现。本次实验中，对于原始的数据文件按照参数 *bloc_size* 进行分块，并将划分后的数据分别成新的文件，以实现数据的分块。

```
def break_stripe(self, block_size):
# block_size 为块大小参数
        index = 0
        while index <= self.nodesize:
        # nodesize 为网络中总的结点数
            block = []
            with open(f"block_{index}.txt", 'w') as fw:
                with open(self.filename, 'r') as fr:
                    for line in fr:
                        src, *dst = line.strip('\n').split(' ')

                        tmp = index
                        flag = False
                        for i in range(tmp, tmp+block_size):
                            if str(i) in dst:
                                flag = True
                                break

                        if flag:
                            block.append(line)
                fw.writelines(block) # 划分后的数据写入新的文件
            index += block_size
```

5.3 随机游走算法实现

本小节分析随机游走算法部分的实现代码。据公式 3引入随机游走因子 β，利用矩阵乘法（公式 2），迭代计算 PageRank 值向量。

```
def matmul(self, r_old, r_new, block, beta=0.85):
    # beta 为随机游走概率参数
        for line in block:
            src, *dst = line.strip('\n').split(' ')
            for j in range(len(dst)):
                if int(dst[j]) in r_new.keys():
```

```
                # 矩阵乘法迭代更新PageRank参数
                r_new[int(dst[j])] += beta * r_old.get(int(src), 0) / len(dst)

        return r_new
```

当迭代更新的误差小于阈值时，则认为算法收敛。

```
while errors > args.threshold:# 判断更新误差是否小于阈值

    # 矩阵乘法更新PageRank值向量
    for r_new, block in zip(load_r('r_new.txt', block_size), load_block()):
        r_new_tmp = r_new
        for r_old in load_r('r_old.txt', block_size):
            r_new_tmp = sme.matmul(r_old, r_new_tmp, block)

    with open('r_new_clip.txt', 'r') as f1:
        with open('r_old.txt', 'r') as f2:
            for l1, l2 in zip(f1, f2):
                new = float(l1.strip('\n').split(' ')[1])
                old = float(l2.strip('\n').split(' ')[1])
                errors += abs(new - old) # 计算更新误差
                index += 1
```

6 实验分析

本节将从开发环境、实验设置、实验数据统计以及实验结果展示三个方面进行实验介绍与分析。

6.1 开发环境

本小节将介绍本次实验所采用的环境。受限于实验条件条件，本次实验仅在本地运行，未使用云服务器及分布式架构。本地使用处理器为 *Intel(R) Core(TM) i7-6700*, 主频 3.40GHz；硬盘参数为 *Samsung SSD 850 EVO 250GB* 固态硬盘，理论读取速度 540MB/S，理论写入速度 520MB/S；RAM 空间为 16GB；未使用 GPU。

6.2 实验参数设置

为保证实验的可复现性，本小节将介绍实验中部分参数的设置情况。参数 threshold 为控制算法收敛的阈值，当算法每轮迭代的更新量小于阈值时，

则算法以收敛；参数 β 为随机游走因子，控制算法随机游走的概率；参数 *block_size* 为分块优化算法中块的尺寸。具体参数取值见表 1。

参数名	取值	备注
threshold	10^{-10}	迭代更新停止阈值
beta	0.85	随机游走因子
block_size	1000	数据分块尺寸

表 1: 实验参数设置详情

6.3 实验数据统计

针对本次实验使用的 Wiki 数据，进行了数据部分参数的统计，统计信息与实验结果可形成验证关系。统计详细信息见表 3

统计量	数值	备注
数据集尺寸	1070KB	数据集所占存储空间
结点数	7115	数据集中网页总数
边数	103689	网页总连接数
Dead ends 数	1005	出链数为 0 的网页数量
边缘结点数	4734	入链数为 0 的网页数量
最大入链节点编号	4037	包含最多入向链接的网页编号
最大出链节点编号	2565	包含最多出向链接的网页编号

表 2: Wiki 数据集统计量详情

Wiki 数据集大小为 1070KB，约为 1MB，远小于运行环境下的内存容量。因此，对于矩阵分块算法，本实验假设数据规模过大，无法一次性装载入内存中。网页总数是 7115，假设使用邻接矩阵的形式来存储网页间的链接关系，共需要保存 7115^2 个矩阵项，占用内存大小为 $7115^2 \times 4$ Byte $=$ 193.11MB。统计网页间的链接数，共 103689 条，计算矩阵的非 0 项比例约为: 0.020。由此可见，邻接矩阵相当稀疏。尽管使用邻接矩阵所占用的存储空间远小于运行环境中的内存空间，本实验还是优化稀疏矩阵的存储方式，以达到进一步节省空间开销的目的。

6.4 实验结果展示

本小节展示算法运行的输出结果。在 PageRank 算法收敛后，选取权重值最大的前 100 个网页写入输出文件。如表所示的为排名在 1-5 以及排名在 96-100 位的部分网页编号以及 PageRank 数值。PageRank 值均保留 6 位小数。

网页排名	编号	PageRank 权重值
1	4037	0.004348
2	15	0.003472
3	6634	0.003385
4	2625	0.003099
5	2398	0.002462
...	...	
96	1726	0.000940
97	3238	0.000935
98	2323	0.000931
99	6784	0.000927
100	3034	0.000924

表 3: Wiki 数据集统计参数详情

值得注意的是，根据 PageRank 权重排名最高的节点编号为 4037，对比 *6.3* 小节中 Wiki 数据集统计量信息，编号为 4037 的网页拥有最多的入向链接。换言之，被链接次数最多的网页拥有最高的权重，实验结果与 PageRank 算法的定义相符。

根据算法计算得到的 PageRank 值，可视化计算的结果。如图 4右图所示，将网页映射至二维平面空间，散点的面积与 PageRank 值成正比。如图 4左图所示，统计网页的 PageRank 值的取值范围，以及在权重范围内网页的数量。对比表 3的统计信息，绝大多数的网页 PageRank 值较小，只有极个别的网页拥有较大的权重。绘制得到的统计柱状图符合“长尾”模型。

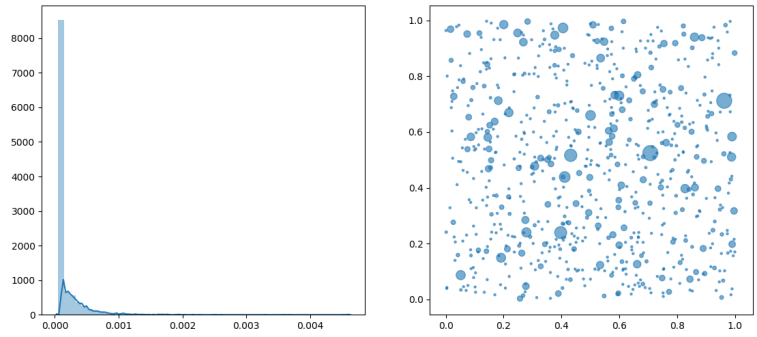

图 4: PageRank 统计信息可视化结果

7 总结

本次实验聚焦于 PageRank 算法分析实现及其优化。在学习并分析了有关 PageRank 算法基础理论后，通过编程对于现实中的案例给出了一个相对原始的解。在此基础上，加入了对于网络结构恶劣情况下算法的改进，掌握了经典的随机游走算法。对于大规模数据，通过分块算法，对于数据进行处理，对于海量数据存储于预算给出了一个相对初级的解决方案。

在这次实验中，加深了对于链接分析与 PageRank 算法的认识，提升了对大规模数据集的处理能力。

8 参考文献

1. Mining of Massive Datasets `http://www.mmds.org/`
2. 互联网大规模数据挖掘与分布式处理(第二版) Jure Leskovec, Anand Rajaraman, Jeff Ullman
3. PageRank 算法：`https://en.wikipedia.org/wiki/PageRank`