《数据仓库与数据挖掘技术》项目报告
多种页面链接分析算法实现

杨东升

1400012898

yangds@pku.edu.cn

信息科学技术学院

2017 年 6 月 15 日

目录

摘要

我完成了多种页面链接分析算法（**亮点 1**），包括 PageRank 算法，加权 PageRank 算法，以及 HITS 算法。我实现了 PageRank 算法的两个版本（**亮点 2**），其一是最基础的矩阵幂乘法版本，其二是对稀疏矩阵的优化版本。另外，我实现了三种加速收敛的技术（**亮点 3**），分别是初始值设计，二次外推法，以及固定已收敛点技术。我在报告中给出了数学推导，并详细展示了实验结果（**亮点 4**），实验结果体现了我的优化算法的正确性和高效性。

第一章　PageRank 算法

1.1　算法概述

PageRank 来源于网络检索问题。最早的网络黄页采用的是分类目录的方法，即通过人工进行网页分类并整理出高质量的网站。那时 Yahoo 和国内的 hao123 就是使用的这种方法。后来网页越来越多，人工分类已经不现实了。搜索引擎进入了文本检索的时代，即计算用户查询关键词与网页内容的相关程度来返回搜索结果。但是仅仅根据关键词的匹配度对网页进行排序是不合理的，网页的内容质量和网页的相关度同等重要，所以需要一些方法来评价网页的价值，以把价值高的且相关度高的网页排在前面。

这个时期有一些算法致力于解决网页价值评价的问题，其中最有影响力的是 PageRank 算法。PageRank 由谷歌的两位创始人佩奇 (Larry Page) 和布林 (Sergey Brin) 最早提出 [3]。谷歌和百度曾经都实用 PageRank 算法作为搜索引擎核心算法，并获得了巨大成功。如图1.1所示，PageRank 的核心思想是基于链接分析来评价网页的价值：

- 如果一个网页被很多其他网页链接到，说明这个网页比较重要，因此 PageRank 值会相对较高
- 如果一个 PageRank 值很高的网页链接到一个其他的网页，那么被链接到的网页的 PageRank 值会相应地因此而提高

PageRank 不仅可以用于搜索引擎，也可以用在其他链接分析的应用场景中，比如对期刊的排序，对发表过论文的研究者的排序，对网络社区的发现和聚集等等。

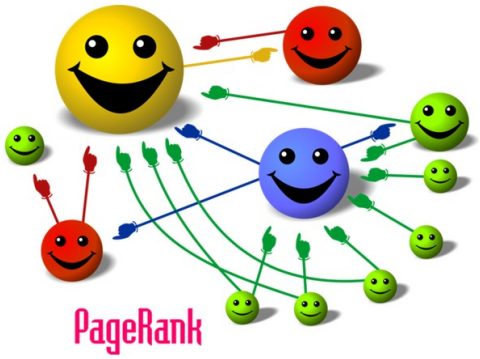

图 1.1: PageRank 原理示意图

1.2　算法原理

为了形式化讨论的方便，首先定义一些符号：$ntotal$ 代表总点数，$cntOut[.]$ 代表出边数，$cntIn[.]$ 代表入边数，$inLinks[.]$ 代表入边的集合，$outLinks[.]$ 代表出边的集合，$PR[.]$ 代表 PageRank 值，$A[.][.]$ 代表关联矩阵，c 代表随机游走概率，ε 代表收敛的阈值。

PageRank 的思想是模拟用户的点击行为。假设有一些用户在浏览一些页面，在某个时刻，他们同时地随机点击当前页面上的一个链接，跳转到另一个页面，当时间足够长以后，每个页面上的用户数量应该是稳定的，这个用户数量就代表页面的 PR 值。前述的 PageRank 思想的公式表达如下：

$$PR[i] = \sum_{j \in inLinks[i]} \frac{PR[j]}{cntOut[j]} \tag{1.1}$$

我们可以定义关联矩阵 A，代表一个点到另一个点 PR 值转移的比例：

$$A[j][i] = \begin{cases} \frac{1}{cntOut[j]} & \text{if } j \in inLinks[i] \\ 0 & \text{else} \end{cases} \tag{1.2}$$

通过上述公式，我们可以发现，每个点原来的 PR 值公平地转移到它的所有出边中，每个点新的 PR 值由所有入边转移来的值累加得到。尽管点的值在每一次更新时会变化，但是在大部分情况下变化的幅度会越来越小，趋于稳定，也就是收敛。可以证明，当矩阵 A 满足如下条件时，每个点的 PR 值都会收敛，且与初始值无关：

- A 是马尔科夫矩阵
- A 对应的图是强连通的
- A 是非周期的

在现实世界中，很多时候关联矩阵 A 是不满足上述条件的，为了让 PageRank 算法得到一个稳定的有意义的结果，就必须对 A 进行改造，使之满足这三个条件。

首先看第一个条件，马尔科夫矩阵也叫随机矩阵，定义是所有元素大于等于 0，并且任意行的元素和等于 1。在我们的问题场景中，A 是马尔科夫矩阵的等价条件是所有点都有出边。对于这个收敛条件我们可以这样理解：如果第 i 个点没有出边，即 A 的第 i 行和就是 0，那么所有转移到这个网页的 PR 值都会被丢失，不能转移给其他网页。进而，整个系统的 PR 值之和会“泄露”，即不断减小，直到减为 0。因此所有页面的 PR 值的收敛值是 0，没有实际意义。为了让 A 成为马尔科夫矩阵，我们要让所有没有出边的网页有出边。最公平的修改方法就是让这些没出边的网页向所有网页连边。因此，我们在之前的基础上修改 A 如下：

$$A[j][i] = 1/ntotal \qquad \text{if } j \text{ has no out links} \tag{1.3}$$

然后观察第二个和第三个条件，强连通性意为任何两个点之间都有路径可以相互到达，非周期性意为图不能只由一些环路组成。如果图不满足强连通性，最后可能会只有 0 和 1 两种 PR 值，没有 0 和 1 的中间状态，这是没有意义的。如果图不满足非周期性，那么点的 PR 值会沿着环不停地传递，永远不会停止在固定的状态上。为了使 A 满足这两个条件，需要加入随机游走策略。随机游走的意思是用户关闭当前网页，随机打开一个新网页。加入随机游走策略后，每次用户以 c 的概率进行随机游

走，以 $1-c$ 的概率点开当前网页上的链接。用公式表达，即 A 变成了所有元素都不为 0 的全连通矩阵，在之前的基础上，修改 A 如下：

$$A[j][i] = A[j][i] \times (1-c) + c \tag{1.4}$$

至此，利用 A，PR 向量从第 $k-1$ 代到第 k 代的转移公式可以表示如下：

$$PR_k = A \times PR_{k-1} \tag{1.5}$$

利用线性代数，我们可以证明，PR 向量的收敛值是 A 特征值为 1 的特征向量。但是，计算特征向量的时间复杂度是 $O(n^3)$，时间代价比较大。因此，现在主要的 PageRank 算法仍然使用迭代计算的方法，目标是快速地求出 PR 向量的近似收敛值。

1.3　矩阵幂迭代算法

在本小节中，我介绍基于矩阵幂实现的 PageRank 算法。本算法的原理来自1.5，是该式的一个直接实现。

首先依据公式1.2 1.3 1.4，从输入数据构建 A 矩阵，然后初始化均分 PR 向量的值，PR 的和为 1；然后开始迭代，让 A 与 PR 做乘法，直到 PR 的变化量小于阈值。见下面的伪代码：

Algorithm 1 矩阵幂迭代算法

Calculate A
for all node i **do**
  $PR_0[i] = 1/ntotal$
end for
$k = 1$
repeat
  $PR_k = A \times PR_{k-1}$
  $\delta = \max_i(|PR_k[i] - PR_{k-1}[i]|)$
  $k = k + 1$
until $\delta < \varepsilon$

矩阵幂迭代的算法复杂度比较高，是 $O(ntotal^2 \times iter)$，其中 $iter$ 是迭代次数。由于矩阵 A 的每一个元素都不为零，所以 A 是稠密矩阵，乘法的时间占据了大部分的运行时间。

1.4　稀疏矩阵幂迭代算法

经过观察发现，A 中的大部分元素都是相同的，它们可以被当作常数抽取出来，A 去掉这些元素后会变成稀疏矩阵。稀疏矩阵相比稠密矩阵可以大大减少乘法次数。

A 中的每个元素的值可能有三种成分：

- 随机游走
- 无出边点均分给所有点

- 一条指向其他点的边

A 中每个元素都包含第一种成分，因此这个成分可以当作常数从每个元素中提取出来，再加到所有点上；A 中很多点包含第二种成分，这些成分贡献的 PR 值会均分给所有点，因此也可以当作常数提取出来，加到所有点上。经过上述处理后，A 只剩下第三种成分，变成稀疏矩阵 A'。现在，每轮更新 PR 向量的公式如下：

$$PR_k = A' \times PR_{k-1} + sum0_{k-1} \times (1-c) + c \tag{1.6}$$

其中 $sum0_{k-1} = \sum_{i \in \text{noOutLinks}} PR_{k-1}[i]$

为了进一步加快稀疏矩阵的乘法，我新建了一个不定长的二维数组 B 来代替 A' 矩阵，$B[i][j]$ $(1 \leqslant j \leqslant cntIn[i])$ 代表第 i 个点的第 j 条入边是从哪个点连过来的，每轮迭代中，只需要用 $B[i]$ 这一行的 $cntIn[i]$ 条信息，而不是 $ntotal$ 条信息，来更新 $PR[i]$，因此可以加快每轮迭代的速度，并且可以节省空间。伪代码展示如下：

Algorithm 2 稀疏矩阵幂迭代算法

```
Calculate B
for all node i do
    PR_0[i] = 1/ntotal
end for
k = 1
repeat
    sum0 = 0
    for all node i with no out links do
        sum0 += PR_{k-1}[i]
    end for
    for all node i do
        for 1 ⩽ j ⩽ cntIn[i] do
            PR_k[i] += PR_{k-1}[ B[i][j] ] * (1 - c) / cntOut[ B[i][j] ]
            PR_k[i] += sum0 * (1 - c) / ntotal
            PR_k[i] += c
        end for
    end for
    δ = max_i(|PR_k[i] - PR_{k-1}[i]|)
    k = k + 1
until δ < ε
```

算法的复杂度分析如下：设总边数为 $etotal$，初始化时间 $O(ntotal)$，每次迭代时，更新 PR 值时间为 $O(etotal)$，检查是否收敛的时间是 O(ntotal)，设总迭代数是 $iter$，则时间复杂度为 $O((ntotal + etotal) \times iter$，稀疏矩阵 $etotal \ll ntotal^2$，因此稀疏矩阵幂算法复杂度显著低于矩阵幂算法。

1.5　加速收敛技术

为了进一步加快 PageRank 的收敛速度，我采用了三种不同的技术，分别是初始值设计，二次外推法和固定已收敛点技术。它们可以分开使用，也可以一起使用。

1.5.1　初始值设计技术

我们已经证明，当关联矩阵 A 满足三个条件，初始值不会影响收敛的结果。但是，好的初始值可以减少迭代次数，加快收敛速度。初始值的总和必须是 1，在此基础上，初始值的分布应该尽量与最后的收敛值接近。为了达到这个目标，我根据点的入边的数量来分配初始值。初始值大小与入边数量成正比，同时，为了防止无入边的点初始值为 0，我把每个点的入边数都当作实际的入边数加 1：

$$PR_0[i] = \frac{cntIn[i]+1}{\sum_{1\leqslant j\leqslant ntotal} cntIn[j] + ntotal} \tag{1.7}$$

这个改进的复杂度分析如下：对所有点的入边数求和需要时间 $O(ntotal)$，对所有点赋初值需要时间 $O(ntotal)$，总时间复杂度 $O(ntotal)$，小于一次迭代的复杂度 $O(ntotal + etotal)$，因此，只要初始值设计能够节省一次迭代，就可以达到加速收敛的目的。

1.5.2　二次外推法

二次外推法的本质是通过 PR_{k-2}, PR_{k-1}, PR_k，可以用较低的时间复杂度推导出一个更接近收敛的新向量 PR_k。即通过代数计算代替矩阵乘法，减少矩阵乘法迭代的次数。我从一篇论文中学习到这个方法 [1]，其理论依据如下：

假设 PR_{k-2} 是第一第二特征的特征向量的线性组合，在这个假设下，可以通过对 PR_{k-1}, PR_k 进行计算直接得到需要的收敛向量 (第一特征向量)。然而，事实上，PR_{k-2} 只是近似于这两个特征的特征向量的线性组合，而计算出来的新 PR_k 也只是对真正的收敛向量的一个近似估计。但由于这个估计的近似度是比原来的 PR_k 更好的，所以可以采用新 PR_k 代替老 PR_k，来加快收敛速度。

使用二维外推法首先假设 PR_{k-2} 是第一第二特征对应的特征向量的线性组合：

$$PR_{k-2} = u_1 + \alpha_2 u_2 \tag{1.8}$$

由于最大特征值 $\lambda = 1$，从而对后两次迭代有

$$\begin{aligned} PR_{k-1} &= A \times PR_{k-2} = u_1 + \alpha_2\lambda_2 u_2 \\ PR_k &= A \times PR_{k-2} = u_1 + \alpha_2\lambda_2^2 u_2 \end{aligned} \tag{1.9}$$

令

$$\begin{aligned} g &= PR_{k-1} - PR_{k-2} = \alpha_2(\lambda_2 - 1)u_2 \\ h &= PR_k - PR_{k-1} = \alpha_2(\lambda_2 - 1)\lambda_2 u_2 \end{aligned} \tag{1.10}$$

使用最小二乘法可以得到 λ_2 的计算公式

$$\lambda_2 = \frac{h^T \times g}{g^T \times g} \tag{1.11}$$

把1.9的两个公式作差得到下式，再把公式1.11代入，可以得到第一特征向量的近似值，最后令 PR_k 的新值等于第一特征向量的近似值，可以加速收敛。

$$PR_k = u_1 = \frac{\lambda_2 PR_{k-1} - PR_k}{\lambda_2 - 1} \tag{1.12}$$

这个改进的时间复杂度为 $O(ntotal)$，因为所有计算都是在向量上进行的。一般情况下，$O(ntotal)$ 的时间相比一次乘法迭代的时间 $O(etotal + ntotal)$ 小很多，所以这个改进只要能减少差不多 $\frac{ntotal}{etotal}$ 比例的乘法迭代次数，就能起到一定的加速效果。

1.5.3　固定已收敛值技术

在 PageRank 计算中，不同的点收敛的速度差异很大。经过观察发现，算法后期大部分点的值变化很小，少部分点的值还在变化，但是由于我的算法需要等到所有点收敛才能停止，所以大部分的收敛点还要跟着少部分的变化点一起被计算，浪费了计算资源。

受此现象的启发，我发现可以通过不再计算已收敛点来减少乘法次数，加快收敛速度。如果一个点的 PR 值变化已经远小于收敛阈值，那么我们可以认为这个点已经几乎收敛，因此算法可以不再更新这个点的值。

判断一个点是否可以不被计算的阈值记为不计算阈值，不计算阈值需要仔细设计。如果它太大，那么没有收敛的点被固定了，导致 PR 值出现误差，且 PR 总和不为 1；如果它太小，那么所有的点都无法固定，进而无法节省计算资源。经过实验测试，我认为把不计算阈值设为收敛阈值的十分之一比较合适，不会影响准确性，且加速效果好。

算法的时间复杂度是 $O(ntotal)$，小于一次迭代的时间复杂度 $O(etotal + ntotal)$，所以时间代价是很小的，空间上需要 $O(ntotal)$ 位，也可以忽略不计。

1.5.4　伪代码实现

综合前面的三种加速技术，现给出伪代码实现;

Algorithm 3 稀疏矩阵幂迭代加速算法

Calculate B
for all node i **do**
  $PR_0[i] = (cntIn[i] + 1)/(sumCntIn + ntotal)$
  $fix[i] = 0$
end for
$k = 1$
repeat
  $sum0 = 0$
  for all node i with no out links **do**
    $sum0$ += $PR_{k-1}[i]$
  end for
  for all node i **do**
    for $1 \leqslant j \leqslant cntIn[i]$ **do**
      $PR_k[i]$ += $PR_{k-1}[\ B[i][j]\ ] * (1 - c)\ /\ cntOut[\ B[i][j]\ ]$
      $PR_k[i]$ += $sum0 * (1 - c)\ /\ ntotal$
      $PR_k[i]$ += c
    end for
    if $|PR_k[i] - PR_{k-1}[i]| < \varepsilon/10$ **then**
      $fix[i] = 1$
    end if
  end for
  $g = PR_{k-1} - PR_{k-2}$
  $h = PR_k - PR_{k-1}$
  $\lambda = (h^T \times g)/(g^T \times g)$
  $PR_k = (\lambda PR_{k-1} - PR_k)/(\lambda - 1)$
  $\delta = \max_i(|PR_k[i] - PR_{k-1}[i]|)$
  $k = k + 1$
until $\delta < \varepsilon$

第二章　其他链接分析算法

2.1　Weighted PageRank 算法

Weighted PageRank 算法 [4] 于 2004 年被提出，引用量近千次，这说明它是 PageRank 算法很好地改进。Weighted PageRank 算法与 PageRank 算法的思想十分类似，唯一的不同之处在于 Weighted PageRank 算法中起始点到目标点的边是有权重的，权重与目标点的入边数和出边数成正比。

在论文 [4] 中，给出的边权计算公式可以表达如下：

$$weight[i][j] = \frac{cntIn[i]}{\sum_{s \in outLinks[j]} cntIn[s]} \times \frac{cntOut[i]}{\sum_{s \in outLinks[j]} cntOut[s]} \tag{2.1}$$

第一个分式意为目标点 i 的入边数，除以起始点 j 的所有出边关联的点的入边数之和，第二个分式同理。

我在实现这个算法时发现一个问题，那就是这个算法的关联矩阵的每一行的和不是 1，也就是说一个点的所有出边的权重之和不是 1，因此这个矩阵不是马尔科夫矩阵。从第 1.2 节的分析可知，这样的矩阵不会收敛到非零。因此，我认为这个算法有问题。除此之外，这个算法考虑一个点的出边数，出边越多点的 PR 值就可以越大，这为垃圾页面恶意增加出边数量留下了漏洞，使得很多垃圾页面的 PR 值偏大。

基于上述两点考虑，我最终采用的边权是如下定义的：

$$weight[i][j] = \frac{cntIn[i]}{\sum_{s \in outLinks[j]} cntIn[s]} \tag{2.2}$$

即只保留原来边权中关于入边数量的项。

Weighted PageRank 算法得到的 PR 向量与 PageRank 的 PR 向量是有差异的，Weighted PageRank 算法的两极分化更严重，即高分更高，低分更低。这是因为高分者的入边本来就多，而且入边的权重还都很大。两个算法得到的结果在排名上相差不大。一个中排名靠前的点在另一个中排名也靠前。由于缺少评价标准，很难评价哪个算法的效果更好。

2.2 HITS 算法

HITS 算法 [2] 于 1998 年被提出，引用量上万次，是一个思想类似 PageRank，但又实现上有很大不同的算法。

首先 HITS 算法是动态计算的，以搜索引擎按照关键词匹配到的页面作为输入；而不是预先计算的，以全网页面作为输入。其次 HITS 算法会考虑网页之间的横向关系，一个主题下的子网站不会与另一个主题的子网站发生联系。

在网页价值的问题上，HITS 算法为每个网页计算两个变量，一个是 authority，意为网页的价值，另一个是 hub，意为网页的指示性，指示性越强，说明它指向的网页质量越高。authority 和 hub 是相互作用的，一个网页指向的网页的 authority 值越大，它自己的 hub 值就越大；一个网页被越多 hub 值高的网页指向，它自己的 authority 值就越大。

在计算之前，需要对矩阵进行预处理，使其满足收敛条件。在计算时，同样采用迭代的办法，先计算所有网页的 authority 值，再计算所有网页的 hub 值，直到这两个值收敛。实际上，计算过程可以视为两个 PageRank 计算过程交替进行。两个 PageRank 的图边的结构是相同的，但是边的方向是恰好相反的。一个简化形式的伪代码示意如下：

Algorithm 4 HITS 简单矩阵幂算法

Calculate A
for all node i **do**
  $AUTH_0[i] = HUB_0[i] = 1/ntotal$
end for
$k = 1$
repeat
  $AUTH_k = A \times HUB_{k-1}$
  $HUB_k = A^T \times AUTH_{k-1}$
  $\delta = \max_i(|AUTH_k[i] - AUTH_{k-1}[i]|)$
  $k = k + 1$
until $\delta < \varepsilon$

所有用在 PageRank 算法上的优化都可以用在 HITS 算法上。

第三章　实验测试

3.1　代码实现

我在这里简单描述一下 pagerank.cpp 代码的结构，HITS.cpp 与 pagerank.cpp 的结构相同，故不赘述。

在主函数中，首先从命令行读取参数，决定以哪种方式运行。

然后调用 BuildIndex 函数进行数据读取和预处理：先从作者名字文件读入名字，进行离散化，即建立字符串到数字编号的双射。然后从作者引用网络文件读入引用关系，填入 matrix 矩阵（即第一章中的 A）和 citedBy 二维数组（即第一章中的 B），并更新 cntIn, cntOut 等数组。

接下来主函数调用 MakeStatistics 函数，建立其他相关的数据结构，比如计算边权矩阵 weight，计算符合三个收敛条件的转移矩阵 matrix，统计无出边的点。

然后调用 Init 函数对 PR 值进行初始化

之后调用 pagerank 核心函数进行计算，核心函数有稠密矩阵乘法版本（pagerank_naive) 和稀疏矩阵乘法版本（pagerank_sparse)，核心函数逻辑与第一章的伪代码相同。每轮迭代之后，调用 CheckChange 函数，检查是否收敛，记录时间和差值等信息。

最后调用 MakeResult 函数对 PR 值进行排序，并输出结果到相应的文件。

3.2　程序运行环境和操作说明

3.2.1　操作说明

代码文件夹下有两个.cpp 源代码，分别实现的是 PageRank 相关的算法和 HITS 相关的算法。代码开发环境是 Windows，IDE 是 DEV-CPP，在 Windows 下可以使用任意 C++ 编译器直接编译，在 Linux 下需要改用 Linux 的计时函数。编译生成可执行文件后，请在命令行中运行.exe 文件，并按提示附加相应参数。运行流程会打印在控制台上，详细信息输出到当前路径下的 output 文件夹中。每次运行程序会产生两个输出文件：PageRank_ 或 HITS 文件的内容是页面按价值排序后的结果，statistics_ 文件的内容是运行时间，迭代次数，每一代的最大差值等统计信息。

pagerank.exe 的命令行参数列表如下：

-n　　矩阵幂乘法

-s　　稀疏矩阵幂乘法

-i　　设计初始值

-e　　二次外推法

-f　　固定已收敛的点的值

-w　　边带权值（weighted PageRank）

需要说明的是，-n 和 -s 不能一起设置，-n 不能和 -e 一起设置，-n 不能和 -f 一起设置，-n 和 -s 至少设置一个，满足上述约束的情况下所有参数可以随意组合，参数的组合和顺序不同，生成的输出文件就不同。

3.2.2　数据集

本实验采用自然语言处理领域的论文引用关系数据，源文件包括所有作者的姓名和作者间的引用关系，一共有 16152 个作者，554439 条引用关系；作者中最多的引用次数为 2921，最多的被引用次数为 7460。引用量基本符合齐夫分布。

3.2.3　运行环境

所有实验在 DELL VOSTRO 5560 台式笔记本上进行。电脑参数如下：处理器为 Core i5 3230M 2.6MHz；内存大小为 4M；操作系统为 Windows 10 32 位

应当包括：程序性能、计算出的结果展示等，宜使用图表进行展示。

3.3　实验结果

我在实验中测试之前所有的算法和优化策略，给出运行时间，迭代次数和收敛速度。对于不同的算法，我还给出它们的页面评价结果的准确性的分析。

3.3.1　运行时间

我测试了九种算法的运行时间，分别是：矩阵幂乘法 n，稀疏矩阵幂 s，设计初值的稀疏矩阵幂 si，二次外推法的系数矩阵幂 se，固定已收敛值的稀疏矩阵幂 sf，带边权的次数矩阵幂 sw，全家福优化 PageRank 算法 siefw，HITS 算法 h。由于稠密矩阵和稀疏矩阵的时间差异巨大，所以全体的结果需要用表格展示。去掉稠密矩阵的结果可以用图展示。

表 3.1: 运行时间

accuracy	si	se	s	sf	siefw	sw	n	nw	h
3	0.017867	0.025707	0.035918	0.03543	0.030427	0.044154	8.642724	5.372923	0.039527
4	0.044908	0.042502	0.053914	0.054161	0.046944	0.063265	12.259364	7.994627	0.064724
5	0.070873	0.076859	0.084556	0.081537	0.070565	0.08705	15.812389	12.367312	0.099722
6	0.136129	0.120567	0.15979	0.160797	0.13799	0.148886	25.557247	24.686784	0.133917
7	0.220048	0.186019	0.237609	0.234128	0.175909	0.22317	39.039193	37.541971	0.170293
8	0.299658	0.26086	0.311536	0.253999	0.183104	0.304605	52.689267	50.28216	0.253081
9	0.371335	0.344993	0.392375	0.2805	0.18601	0.379507	65.583294	64.113265	0.359786
10	0.449237	0.389475	0.464323	0.292831	0.196057	0.45746	78.993351	77.804197	0.44925

在表3.1中展示了九种算法的运行时间，稠密矩阵幂 n 收敛到精度 10 的总时间在 78 秒左右，而最快的优化算法 siefw 收敛到精度 10 需要 0.196 秒，加速了近 400 倍。

表中第一列 accuracy 代表收敛阈值的指数的相反数。实验中改变收敛阈值从 10^{-3} 到 10^{-10}，对应的 accuracy 变化区间是 3 到 10。对每个阈值，分别测试上述 8 个算法达到收敛所需要的时间。需要说明的是，这个阈值范围已经基本可以满足所有需求，如果阈值小于 10^{-10}，精度提高的十分有限，而所需的时间会变得很长，因此意义不大。

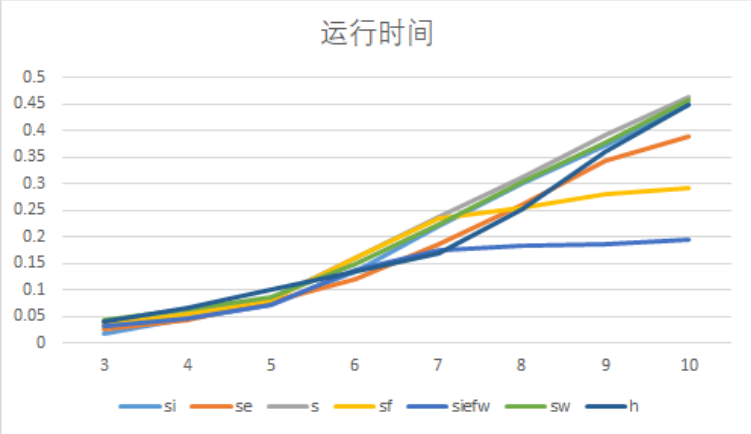

图 3.1: 运行时间

我把 n 和 ns 之外的 7 个算法的运行时间画在图3.1中。从图中可以看出，所有算法的运行时间都随阈值减小而单调上升，近似于线性变化。在 PageRank 算法的优化算法之中，si 的曲线一直在 s 的下方，但是差异不大，说明设计初始值对 s 算法的加速有限。se 的曲线中期斜率较小，表明二次外推法在收敛中期最有效。sf 的曲线后期斜率很小，几乎为零，这表明固定已收敛值算法在后期十分有效，因为只有少部分点未收敛，所以只需计算少部分点的乘法。全家福优化算法 siefw 在前期时间偏大，因为有一定的开销，但是中期和后期的表现十分优秀，达到了超过 1 倍的加速效果。HITS 算法的运行时间与 PageRank 相仿，中期时间略短，不过后期时间又一样了，这说明 HITS 和 PageRank 的核心算法是十分相似的。

3.3.2　迭代次数

在图3.2中展示了九种算法的迭代次数，横轴是收敛阈值的指数的相反数，纵轴是迭代次数。最高的灰线是三条重合的线，分别是 s, sf, n，原因是 s 和 n 的算法逻辑完全相同，而 sf 不影响收敛最慢的点，所以它们的迭代次数是一样的；略低一点的棕线是两条重合的线，分别是 sw, nw，原因是 sw 和 nw 的算法逻辑完全相同；浅蓝色的线是 si，说明设置好初始值会稍微减少迭代次数；橘黄色的线是 se，说明二次外推法可以减少迭代次数；天蓝色线是 siefw，说明所有优化共存可以更好地减少迭代次数；深蓝色的是 HITS 算法，它每次迭代要进行一正一反两次更新，因此它每一次迭代的实际运算量应该是 PageRank 类算法的两倍，所以在总运算量差不多的情况下，它的迭代次数是 PageRank 的一半。总体来说，各个算法的迭代次数都比较少，收敛到 10^{-10} 最多只需要 86 代。

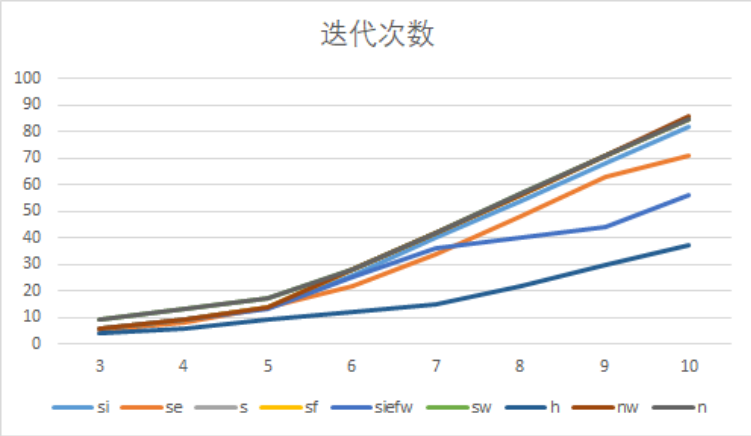

图 3.2: 迭代次数

3.3.3　收敛速度

在图3.2中展示了九种算法的收敛速度，横轴是迭代次数，纵轴是达到的精度，定义为这一代的点与前一代的对应点的最大差值的负对数。在精度达到 10 以后，该折线会回落到 0，因为后面的数据就没有了。

最左面的灰线是 HITS 算法，由于每代做的运算是两倍的量，所以它收敛最快；其次是 siefw 算法，它有波动是由于二次外推法导致的；然后的天蓝线是 se，二次外推法导致它在波动；然后是 si，设置初始值技术使它的截距更小；接下来是两条重合的线，sw 和 nw；最后是三条重合的线，分别是 s, sf, n，原因是 s 和 n 的算法逻辑完全相同，而 sf 不影响收敛最慢的点，所以它们的收敛速度是一样的；

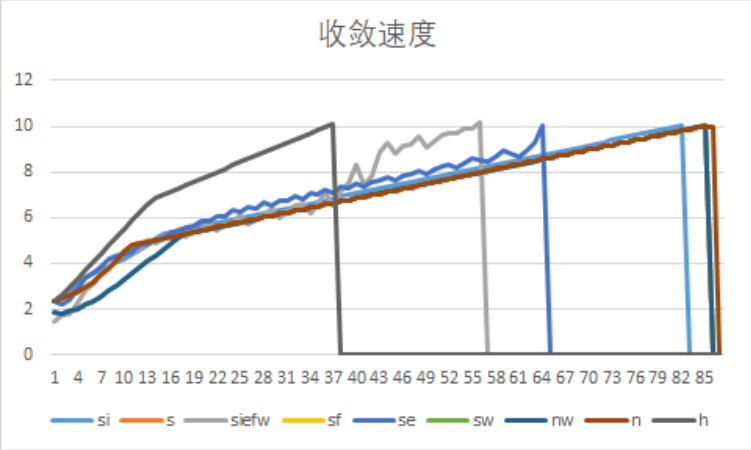

图 3.3: 收敛速度

3.3.4　结果评价

在表3.1中展示了 PageRank 算法，加权 PageRank 算法和 HITS 算法的运行结果的前十名及对应的值。

表 3.2: 排序结果

author(PR)	value	author(PRW)	value	author(HITS)	value
Mercer, Robert L.	0.011848	Mercer, Robert L.	0.058544	Och, Franz Josef	0.007704
Della Pietra, Vincent J.	0.010697	Della Pietra, Vincent J.	0.057083	Ney, Hermann	0.005702
Della Pietra, Stephen A.	0.009956	Della Pietra, Stephen A.	0.049481	Della Pietra, Vincent J.	0.004846
Brown, Peter F.	0.009932	Brown, Peter F.	0.047885	Koehn, Philipp	0.00482
Church, Kenneth Ward	0.009576	Och, Franz Josef	0.034386	Manning, Christopher D.	0.00471
Sampson, Geoffrey	0.007043	Church, Kenneth Ward	0.023735	Marcus, Mitchell P.	0.004503
Marcus, Mitchell P.	0.006773	Ney, Hermann	0.023397	Collins, Michael John	0.004463
Och, Franz Josef	0.006386	Marcus, Mitchell P.	0.020976	Marcu, Daniel	0.00439
Jelinek, Frederick	0.006264	Collins, Michael John	0.014101	Church, Kenneth Ward	0.00431
Ney, Hermann	0.005058	Roukos, Salim	0.012853	Della Pietra, Stephen A.	0.004271

观察 PR 和 PRW 的结果可以发现它们的排名十分相似，并且有 8 个项是共有的。区别在于 PRW 的值更大。

观察 PR 和 HITS 的结果可以发现它们的排名有一定的差异，并且只有 5 个项是共有的。PR 的项值比 HITS 的项值更大一些。

我通过搜索谷歌学术发现这些作者都很有名，说明这些算法的结果都比较准确。

参考文献

[1] 吴家麒 and 谭永基. Pagerank 算法的优化和改进. 计算机工程与应用, 45(16):56–59, 2009.

[2] Jon M Kleinberg. *Authoritative sources in a hyperlinked environment*. ACM, 1999.

[3] L Page. The pagerank citation ranking : Bringing order to the web. *Stanford Digital Libraries Working Paper*, 9(1):1–14, 1998.

[4] Wenpu Xing and Ali Ghorbani. Weighted pagerank algorithm. In *Communication Networks and Services Research, 2004. Proceedings. Second Conference on*, pages 305–314, 2004.

网络资料

- PageRank 算法——从原理到实现 `http://www.cnblogs.com/rubinorth/p/5799848.html`
- PageRank 背后的数学 `http://blog.csdn.net/golden1314521/article/details/41597605`