分类号	
UDC	
密　　级	
学　　号	1208090509

西安理工大学

硕士学位论文

搜索引擎 PageRank 算法研究及其改进

金明心

学 科 门 类：工 学

学 科 名 称：电路与系统

指 导 教 师：王林　教授

申 请 日 期：2015 年 6 月

独 创 性 声 明

本人所呈交的学位论文是在导师指导下进行的研究工作及取得的成果。尽我所知，除特别加以标注的地方外，论文中不包含其他人的研究成果。与我一同工作的同志对本文的研究工作和成果的任何贡献均已在论文中作了明确的说明并已致谢。

本论文及其相关资料若有不实之处，由本人承担一切相关责任。

论文作者签名：金明心　　2015 年 6 月 24 日

学位论文使用授权

本人作为学位论文作者了解并愿意遵守学校有关保留、使用学位论文的规定，即：在导师指导下创作完成的学位论文的知识产权归西安理工大学所有，本人今后在使用或发表该论文涉及的研究内容时，会注明西安理工大学。西安理工大学拥有学位论文的如下使用权，包括：学校可以保存学位论文；可以采用影印、缩印或其他复制手段保存论文；可以查阅或借阅。本人授权西安理工大学对学位论文全部内容编入公开的数据库进行检索。本学位论文全部或部分内容的公布（包括刊登）授权西安理工大学研究生院办理。

经过学校保密办公室确定密级的涉密学位论文，按照相关保密规定执行；需要进行技术保密的学位论文，按照《西安理工大学学位论文技术保密申请表》内容进行保密（附《西安理工大学学位论文技术保密申请表》）。

保密的学位论文在解密后，适用本授权。

论文作者签名：金明心　导师签名：　　2015 年 6 月 24 日

论文题目：搜索引擎 PageRank 算法研究及其改进

学科专业：电路与系统

研 究 生：金明心　　签 名：金明心

指导教师：王　林 教授　　签 名：

摘 要

近年来，随着 Internet 的不断发展，互联网已经成为人们重要的信息来源，并为人们提供了丰富的信息资源。同时，互联网的迅速发展，也使得现有搜索引擎技术面临着巨大的挑战，面对如此海量的信息，如何快速准确检索到用户所需要的信息，便显得十分重要。而排序算法，作为一个搜索引擎的核心，自然也就成为了人们关注的重点。

作为世界上最经典的页面排名算法，PageRank 算法于 1998 年由 Google 的两位创始人 Larry Page 和 Sergey Brin 开发，并被成功应用于 Google 搜索引擎。本文围绕 PageRank 算法，首先详细介绍了传统 PageRank 算法的体系结构、形成思路和计算方法，同时，通过对传统 PageRank 算法进行深入的剖析，指出了传统算法存在主题漂移和偏重旧页面这两大不足，并针对这两个不足介绍了多种现有的经典改进算法。接着，从主题漂移角度出发，本文提出了一种优化改进算法，即改进传统算法在分配 PageRank 值时没有针对性，单纯将自己权值平均分配给各个页面的思路，而采用根据各个页面重要性不同去分配 PageRank 值的方法。

最后，本文基于 Nutch 搜索引擎设计原型系统，对改进算法进行验证，通过在 Nutch 中分别实现 PageRank 原型算法和改进算法，将两种算法进行比较，实验结果表明，不论从查准率角度还是相关度角度，改进算法均有一定的优越性，即改进算法确实可以优化页面的排序质量，提升搜索的准确性，表明该算法的有效性。

关键词：PageRank；搜索引擎；非平均分配；Nutch

Title: Research Of PageRank Algorithm In Search Engine And Its Improvement

Major: Circuit and Systems

Name: Mingxin Jin **Signature:** Mingxin Jin

Supervisor: Prof. Lin Wang **Signature:** Lin Wang

Abstract

With the quick popularization and development of the Internet technology, the Internet becomes people's important source of information and provides people with abundant information over past decades. Meanwhile, the rapid development of the Internet provides huge challenges to the search engine technology. Facing of such vast amounts of information, how to search the useful information quickly and accurately is very important for search engines. Ranking algorithm, as the core of a search engine, naturally draws peoples' attention.

As the most classical web page ranking algorithm, the PageRank algorithm was developed by Brin and Page in 1998, and has been successfully applied in the Google search engine.This paper attaches importance to the PageRank algorithm. First, introduces the architecture, develop ideas and calculation method of the traditional PageRank algorithm, by diving deep into it, this paper points out the topic drift and put emphasis on the old page are two big problems of the traditional algorithm and introduces a variety of classical existing algorithms. Then, the author puts forward an improved algorithm from the topic drift perspective, improves the traditional algorithm in the distribution of PageRank value no pertinence, simply to assign the PageRank value on the average and adopts the new method which distributes the PageRank value according to each page importance.

In the last of this paper，the author designs the prototype system based on Nutch search engine to verify this improved algorithm, by implementing the traditional PageRank algorithm and improved algorithm in the Nutch respectively, comparing the two algorithms, the results show that the improved algorithm has certain advantages no matter from the perspective of precision or relevance perspective, mean that the improved algorithm can optimize the quality of the sorting, improve the accuracy of search engine, and demonstrate the effectiveness of the algorithm.

Key words: PageRank; Search Engine; Non-Average Distribution; Nutch

目 录

1 绪论

自 20 世纪 90 年代以来，随着计算机技术的普及和 Internet 网络技术的发展，信息数字化和数据网络化已成为现代经济社会发展的客观要求和必然趋势。中国互联网信息中心（CNNIC）2015 年 2 月 3 日在京发布《第 35 次中国互联网发展状况统计报告》[1]，报告指出，截止 2014 年 12 月底，我国网民数量达 6.49 亿，较 2013 年底增加 5672 万人，互联网普及率为 47.9%，较 2013 年底提升了 1.1 个百分点。同时，包括政治、经济、科学、文化、艺术等多个社会领域也都不同程度地实现了其资源信息的数字化和共享化，Internet 网络已名副其实的成为世界上最大的信息中心。

1.1 研究背景及意义

现代人类已经处在一个信息极度丰富的时代，网络可以为用户提供所能想到的各种信息资源。作为一个 Internet 用户，自然希望能够最大程度地在如此庞大而全面的信息资源中找到对自己有价值的信息。搜索引擎（Search Engine）正是在这种需求下出现的技术。当用户以关键词查找信息时，搜索引擎会在数据库中进行搜寻，如果找到一个与用户需求相符的网站，便采用特殊的算法，根据关联度高低，按顺序将这些页面链接返回给用户。

由于互联网的发展迅速，互联网上的信息呈现出信息量大且分散、自治性强、资源多样的特点。然而，现有搜索引擎检索出来的结果往往存在不一致性和不完整性，为此，面对互联网的种种问题，如何快速、准确地从互联网上获取有价值的信息，就成为评价搜索引擎的重要指标，而排序算法，作为一个搜索引擎的核心，自然也就成为了人们关注的重点。

1.2 国内外研究现状

页面排序算法主要有两大类，一种是基于页面内容分析进行排序，一种是基于互联网超链接结构分析进行排序。基于页面内容的排序源于传统的信息检索，但由于互联网中的数据过于庞大，当今的页面检索已对传统信息检索提出了巨大难题。

基于超链接结构分析的排序将互联网搜索与传统的信息检索区分开来，Google 创始人 Sergey Brin 和 Lawrence Page 于 1998 年提出的 PageRank 算法[2]是目前被认为最为成功的一种超链接结构分析方法。该算法通过分析网络的链接结构来获取网络中的权威页面，并在商用搜索引擎 Google 中大获成功，掀起了网络链接分析的热潮。

但由于仅仅利用了网络的链接结构，且链接信息相对静态，没有考虑页面使用的动态信息，因此该算法还存在不少缺点，如主题漂移，偏重旧页面，无法判断页面内容上的相似性等问题。为此，继 Sergey Brin 和 Lawrence Page 提出 PageRank 算法以后，国内外很多学者、研究机构纷纷对 PageRank 进行了改进，主要有以下几个方面。

1.2.1 国外研究现状

为了提高检索结果的主题相关性，斯坦福大学的 Taher H.Haveliwala 在 2002 年提出

了一种主题敏感（Topic-Sensitive）的 PageRank 算法[3]。该算法考虑到一些页面在某些领域被认为是重要的，但并不表示它在其它领域也是重要的。所以算法首先根据 ODP（Open Directory Project）[4] 列出 16 个基本主题向量，对页面进行分类，并针对每个页面的 ODP 分类，给出一个主题敏感值，然后将此值引入算法，离线计算出页面在各个 ODP 分类下的 PageRank 值。再根据用户输入的查询请求及查询上下文，计算页面的综合得分，进而对页面进行排序。

华盛顿大学的 Matthew Richardson 和 Pedro Domingos 认为用户在浏览页面时，会受到当前页面内容和查询主题的影响，而从一个页面跳转到另一个页面，为此提出了一种结合链接结构和内容信息的 PageRank 算法（简称 MP-PageRank 算法）[5]，该算法除去 PageRank 算法假设用户完全随机访问网络这一前提外，增加考虑了用户从一个页面直接跳转到另一个与之内容相关却非直接相邻页面的情况。

康奈尔大学的 Kleinberg 博士认为 PageRank 算法中对于页面的所有出链接是同等对待的，并没有考虑页面之间的重要性差异。然而，在实际的 Web 链接中，不同链接其重要性是不同的。为此他提出了一种 HITS（Hyperlink Induced Topic Search）算法[6]，该算法通过引入中心页面（Hub）和权威页面（Authority），让其互相加强，从而找到 Authority 页面，实现 Web 结构及资源的自动发现。

随着页面数量的快速增长，集中式的网络搜索引擎已经不能在性能上满足需求。于是，Yamamoto 和 Asahara 等人认为 PageRank 只有采用分布式的算法，才能和目前的网络搜索引擎结构相结合，并在性能上满足快速更新的要求。为此，他们提出了一种基于 P2P 网络的分布式 PageRank 算法[7]，各结点采用异步方式进行通信，实验结果表明该算法解决了链接失效问题，具有很好的收敛性。

在传统 PageRank 算法中，大多数采用的都是串行离线计算整个网络的 PageRank 值，这样不仅消耗系统资源而且耗时。为了解决这一问题，Rungsawang 和 Manaskasemsak 提出了一种并行计算 PageRank 值的算法[8]。在实验中他们采用 32 台服务器并行计算 2800 万个网页，使收敛速度达到了原来的 6 倍多。

紧接着为了使 PageRank 算法拥有更快的收敛速度，一批优秀的加速收敛算法被设计出来。斯坦福大学的 Sepandar D. Kamvar，Taher H. Haveliwala，Gene H. Golub 提出了一种自适应法（Adaptive Method）[9]，其基本思想是基于迭代过程中某些页面的 PageRank 向量比其他页面的 PageRank 向量收敛快这一事实，进而通过锁止这类页面的迭代计算来减少运算的规模和时间。

美国布鲁克林工艺大学 CIS 部 YenYu Chen，Qingqing Gan，Torsten Suel 提出了一种基于 I/O 技术的 PageRank 加速算法[10]。

为了使收敛速度进一步加快，Sepandar D. Kamvar，Taher H. Haveliwala，Christopher D. Manning，Gene H. Golub 提出了多种外推法（Extrapolation Method）[11]。其基本思想是在迭代计算过程中进行加速，充分利用网络链接矩阵最大特征值是 1 这一实际情况去求得主

特征向量的近似值。

除了上述从特征值和特征向量的角度在迭代计算过程中进行加速外，Sepandar D. Kamvar，Taher H. Haveliwala，Christopher D. Manning，Gene H. Golub 还提出了一种从网络图结构角度去加速的方法：块结构法（Block Structure Method）[12]，也称为分块矩阵排序算法（BlockRank Algorithm）。它充分利用 Web 页面间链接结构呈现出块状结构这一特征来改进算法效率。

1.2.2 国内研究现状

在 MP-PageRank 算法的基础上，黄德才，戚华春提出一个针对两个页面进行相似度描述的二阶相似度概念，进而提出了一种二阶相似度算法[13]。该算法吸收了 MP-PageRank 算法的基本思想，即认为用户从一个页面跳转到另一个页面会受到当前页面内容和查询主题的影响。并在此基础上，进一步计算两个页面之间的相似度，使得页面的 PageRank 值主要在具有相似主题的页面间传播，从而减少在主题无关页面上的扩散。

PageRank 算法的主要缺陷就是其向外链接平均分配权值的思想不符合实际情况。平均分配的思想，往往使得主题不相关页面获得与主题相关页面同样的重视度。基于这一思想，高琪，张永平通过引入经典的 *tf -idf* 策略用一个超链接文本权重除以页面中所有超链接文本权重之和来计算页面中的超链接文本在所有页面中所占权重比例，提出了一种与查询主题相关的 Query-Based PageRank 算法[14]。

白似雪，刘华斌针对传统 PageRank 算法中平均分配策略的不足，提出了一种基于分块主题的 PageRank 算法[15]，该算法通过对页面进行分块操作，给不同分块的出链接以相应的权重。

页面之间的链接分析有这样一种特征：一个与教育相关的 A 网站指向另一个与教育相关的 B 网站较一个与娱乐相关的 C 网站指向另一个与教育相关的 B 网站更为权威、可信。原始的 PageRank 算法中，在进行页面排序时，没能很好地考虑页面的分类特性和页面之间的相关性。根据上述特征，李绍华，高文宇提出了一种基于层次分类（Hierarchic Classification）的 PageRank 算法[16]。

为了让高质量的新页面能快速地在网络上传播，上海交通大学的张玲博士提出了一种基于时间序列分析的加速评估算法（Accelerated Ranking，AR）[17]，该算法使得网络上有价值的页面能够以更快的速度传播；相反，一些含有陈旧数据的页面将被加速掩盖。这种算法保证了信息检索中的优胜劣汰，确保向用户提供高质量的网络链接。

戚华春等人提出将页面的存在时间通过搜索引擎搜索的周期数来表示，即用一个页面被搜索引擎访问的周期数来代替时间反馈权值，从而提出了一种 PageRank-Time 算法[18]。

由于页面采用的是半结构化的 HTML 语言，其中包含了丰富的结构信息，上海交通大学的宋聚平博士等人认为对于这些结构信息应该给予充分利用，为此，他们提出了一种结合页面 HTML 语言的 PageRank 算法[19]。

北京大学计算机系提出了一种结合 HTTP 协议的 PageRank 算法[20]，该算法通过记录

页面最近一次被修改时间，在运行分析算法时把页面修改时间作为控制参数，从而给予新修改的页面以较高的权值，老页面以较低的权值，以此计算页面的排名。

随着计算机的普及和互联网的迅速发展，大量的数据以网页的形式呈现在人们面前，如何使网络上的数据及时准确的提取、过滤和归纳成便于人们查询检索的形式已成为研究开发的热点，为此，西安理工大学的舒鹏同学在《非结构数据化提取方法研究》一文中，合理运用 PageRank 算法，在提高了数据提取准确率和查全率的同时，也使提取时间大大缩短，显著提高了提取效率[21]。

1.3 本文的研究意义及主要工作

本文重点是对 Google 搜索引擎的 PageRank 排序算法进行研究，详细介绍了该算法的体系结构和实现细节，并在此基础上，结合多种现有的经典改进算法，从 PageRank 算法存在主题漂移问题的角度出发，提出了一种新的基于 PageRank 算法的优化策略与改进算法，旨在提高搜索引擎排名的可靠性和公平性，保证搜索引擎的性能。

本文的主要工作可以概括为以下几个方面：

（1）对 Google 搜索引擎传统 PageRank 算法的体系结构、形成思路和计算方法进行了详细介绍，分析其存在的优缺点，并针对其缺点介绍了国内外现有的多种经典改进算法。

（2）针对传统 PageRank 算法的不足提出改进方案，改进了传统算法在分配 PageRank 值时没有针对性，单纯将自己权值平均分配给各个页面的思路，而采用根据各个页面重要性不同去分配 PageRank 值的方法。

（3）Nutch 原型系统设计和研究，通过在 Nutch 上搭建搜索引擎，并在 Nutch 中分别实现 PageRank 原型算法和改进算法，证明改进算法的有效性和合理性。

1.4 本文的组织安排

本文各章节的组织结构：

第一章介绍了本文的研究背景和国内外对该主题的研究现状，在此基础上提出了本文的主要研究内容。

第二章对 Google 搜索引擎的 PageRank 算法进行了详细介绍，着重剖析了该算法的体系结构、形成思路和计算方法，接着介绍了多种有关 PageRank 算法的加速算法。并在此基础上，分析了传统 PageRank 算法的优缺点，并主要针对主题漂移问题和偏重旧页面问题详细介绍了国内外现有的多种经典改进算法。

第三章主要针对传统 PageRank 算法平均分配权值这一缺陷，提出了一种新的基于传统 PageRank 算法的优化策略，并详细介绍了改进算法的基本思想。最后通过一个简单的试验网络说明了改进算法的理论可靠性。

第四章主要分析了 Nutch 搜索引擎的工作流程和排序机制，通过融入 PageRank 计算因子对 Nutch 评分公式进行了改进，得到了结合 PageRank 算法的 Nutch 综合排序算法。

第五章在 Nutch 上构建搜索引擎，通过对传统算法和改进算法的分析比较，证明了改

进后的 PageRank 算法确实可以提高搜索结果的查准率和相关度，优化排序质量，提升搜索的准确性。

第六章总结和展望，对本文的工作做一总结，同时介绍了今后需要进一步完善的工作。

1.5 本章小结

本章首先介绍了有关本文研究主题的研究背景和国内外研究现状，随后说明了本文的研究意义和本文所要做的主要工作，最后对本文的结构给出了总体框架。

2 PageRank 算法综述

搜索引擎 Google 最初是斯坦福大学的博士研究生 Sergey Brin 和 Lawrence Page 实现的一个原型系统，现已发展成为 WWW 上最好的搜索引擎之一。PageRank 算法作为 Google 搜索引擎的核心技术，是一种完全基于网络链接结构的页面排序算法，因此它只与网络的拓扑结构有关，可以将其视为网络的一个属性。Google 通过 PageRank 排序算法计算出页面的 PageRank 值，从而决定页面在检索结果中出现的位置，即 PageRank 值越高，在结果中出现的位置越靠前。

2.1 PageRank 算法的基本思想

PageRank 算法的基本思想源自传统文献计量学中的文献引文分析。主要基于以下两个前提：

（1）一个页面被多次引用，则它可能是重要的；一个页面虽然没有被多次引用，但是被重要的页面引用，则它也可能是重要的；一个页面的重要性被平均传递到它所引用的所有页面。

（2）基于“随机冲浪者”（Random Surfer）模型[22]，即用户一开始随机访问页面集合中的任一页面，随后按照该页面的向外链接向前不回退重复浏览页面，浏览下一个页面的概率就是被浏览页面的 PageRank 值。

2.2 PageRank 算法描述

假设 u 是一个页面，$B(u)$ 是指向 u 的页面集合，那么 u 的 PR 值（PR 值为 PageRank 值的简称，以下统一用 PR 值代替 PageRank 值）为：

$$PR(u)=\sum_{v\in B(u)}PR(v) \tag{2.1}$$

但因为指向 u 的页面同时也可能指向其他页面，该页面会将它的 PR 值分配给自身指向的所有页面，这里，假设 $N(u)$ 是页面 u 所指向的外链接总数，对公式(2.1)进行修改得：

$$PR(u)=\sum_{v\in B(u)}\frac{PR(v)}{N(v)} \tag{2.2}$$

用户在浏览页面时，有时并不是严格按照页面的链接结构进行浏览，常常因无聊而随机跳入另一个页面。这个假设相当符合实际情况，也就是说虽然 A 页面没有链接到 B 页面，但实际上当用户浏览到 A 时还是会有可能跳转到 B。为了满足上述情况，这里引入一个参数 d，d 值越高，继续点击链接进行浏览的概率就越大，而将用户停止点击并随机跳转至另一页面的概率记为 $(1-d)$。我们将这个概率 d 称为阻尼系数（Damping Factor）[23]，它取决于用户点击的次数，被设定为 0-1 之间。为此，我们将式(2-2)改进如下：

$$PR(u)=d\sum_{v\in B(u)}\frac{PR(v)}{N(v)}+(1-d) \tag{2.3}$$

从式(2.3)看出，页面 *PR* 值仅 *d* 部分在它所链接到的页面中分配，剩余部分用在整个网络的所有页面中分配，*d* 值通常为 0.85 左右。换句话说，假设用户在当前时刻浏览任一页面，下一步，他要么以概率 *d* 从当前页面的超链接中选择一个页面作为下一个将要浏览的对象，要么以概率 $(1-d)$ 从整个网络中选择一个页面进行浏览。

这样每计算一次，便可以得到页面的 *PR* 值，经过多次递归迭代计算直到 *PR* 值达到收敛，即求得页面最终的 *PR* 值。

阻尼系数 *d* 的引入，不仅仅解决了随机浏览的问题，还间接的解决了实际网络中广泛存在的另外两个问题：等级沉淀（Rank Sink）和等级泄露（Rank Leak）[24]。

等级沉淀（Rank Sink）是指在庞大的互联网中，有时一些链接只是在一个集合内部互相链接而不会向外进行任何链接，如图 2-1 所示：页面 B，C，D 内部相互链接，无向外链接，此时页面 A 链接到页面 D，则在迭代过程中，页面 B，C，D 的 *PR* 值因不能分布出去而循环累计。最后导致循环外的页面 *PR* 值为零，并使循环内的页面 *PR* 值分布出现震荡。

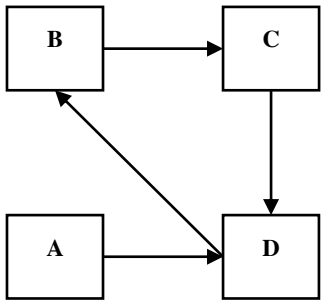

图 2-1 PageRank 中的等级沉淀现象

Fig.2-1 Phenomenon of Rank Sink in PageRank

等级泄露（Rank Leak）是由于现实中页面之间并不是强连通而造成的，即存在一些没有向外链接的页面，可理解为这些页面的出度（Outdegree）为 0，如图 2-2 所示：页面 A 没有向外链接，则在迭代过程中，最终所有页面的 *PR* 值将趋于 0。这种无向外链接的页面也被称为悬挂页面（Dangling Page）。

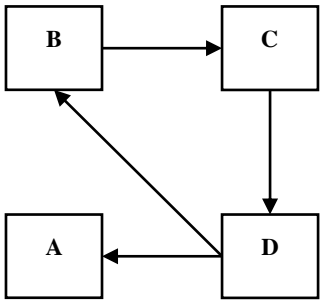

图 2-2 PageRank 中的等级泄露现象

Fig.2-2 Phenomenon of Rank Leak in PageRank

上述两种情况在实际网络中是广泛存在的，但我们不能因为网络的局限性而终止我们的浏览行为。当用户发现这两种现象时，可以通过重新输入网址或其他方法进行下一步的浏览。

针对上述问题，目前也存在很多其它解决方法，对于 Rank Leak，Kamver[12] 等人提出可以删除网络中的悬挂页面及相关的边，因为这些页面并不会影响到其它页面的 *PR* 值，但要注意的是当悬挂页面被拿掉后，可能会出现一批新的悬挂页面。当移除所有悬挂页面后，对剩下部分计算 *PR* 值，然后以逆向顺序反推悬挂页面的 *PR* 值。另外一种方法是由 Haveliwala[25] 等人提出的，他们假设悬挂页面对所有指向它的链接都有相应的返回链接。对于 Rank Sink，Sergey Brin 和 Lawrence Page[26] 提出通过引进一个等级源（Rank Source）概念来不断补充每个页面的 *PR* 值，以使得 *PR* 值的分配不完全依赖着链接。

下面我们利用数学工具，按照其与链接关系相关的数据，通过构建数学模型来分析如何计算 *PR* 值。

在此之前首先需要构建代表网络链接结构的链接矩阵。将 Web 网看做一个由点集 W 和边集 L 组成的有向图 G（W,L），其中 W 代表页面，L 代表页面之间的超链接。当两个页面之间存在链接时，L 取值 1；不存在链接时，L 取值 0，即

$$L_{ij}=\begin{cases}1 & \text{网页}i\text{指向}j\\ 0 & \text{不存在链接}\end{cases} \tag{2.4}$$

设一个节点的出链接数（即出度）为 $deg(i)$，则 $deg(i)=\sum_j L_{ij}$。我们定义矩阵 $P_{ij}=L_{ij}\big/deg(i),(deg(i)>0)$，当 $deg(i)=0$ 时，$P_{ij}=0$，这里的矩阵 $\boldsymbol{P}$ 我们称之为转移概率矩阵（Transition Probability Matrix）[27]。

为了保证上述矩阵 $\boldsymbol{P}$ 是一个有效转移概率矩阵，即为了确保矩阵 $\boldsymbol{P}$ 有唯一稳定的概率分布，或者说保证迭代过程的收敛，矩阵 $\boldsymbol{P}$ 必须满足两个条件：一是矩阵 $\boldsymbol{P}$ 必须是非循环的（Aperiodic）；二是矩阵 $\boldsymbol{P}$ 必须是不可约的(Irreducible)。前者可以由网络结构来保证，后者需要保证网络图 G 是强连通的[28]。

在实际数据集中，很多网络图都是非强连通的，那是因为摇摆页面的存在，即网络图中存在一些没有出链的节点，反映在矩阵 $\boldsymbol{P}$ 中即存在某一行元素全为 0 的情况。为了解决这一问题，我们可以通过对所有节点加入一个小的转移概率，即通过构造一个新矩阵 $\boldsymbol{P}'$ 来确保建立一个强连通的网络图。假设 n 为网络图中的节点数（即页面数），定义 $\boldsymbol{f}$ 为 n 维列向量，用来识别出度为 0 的节点，即

$$f_i=\begin{cases}1 \text{ if } \deg(i)=0\\ 0 \text{ otherwise}\end{cases} \tag{2.5}$$

定义 $\boldsymbol{v}$ 是一个 n 维列向量，代表着一个统一的概率分布，即

$$\boldsymbol{v}=\left[\frac{1}{n}\right]_{n\times 1} \tag{2.6}$$

那么可以按如下方式构造矩阵 $\boldsymbol{P}^{'}$：

$$\begin{cases}\boldsymbol{F}=\boldsymbol{f}\cdot\boldsymbol{v}^{T}\\ \boldsymbol{P}^{'}=\boldsymbol{P}+\boldsymbol{F}\end{cases} \tag{2.7}$$

这样变化后，相当于如果矩阵中的某一行元素全为 0，则将其全部替换为$1/n$，即相当于人工的给摇摆页面添加指向网络所有页面的链接【29】。这样的方法很实用，而且计算起来很方便，即使网络中不存在摇摆页面，这样的变换也是有效的。

最后，为了满足前面所介绍的随机跳转原则，在转移概率矩阵中引入阻尼系数 d，即构造矩阵 $\boldsymbol{P}^{''}$：

$$\begin{cases}\boldsymbol{E}=\boldsymbol{e}\cdot\boldsymbol{v}^{T}\\ \boldsymbol{P}^{''}=d\cdot\boldsymbol{P}^{'}+(1-d)\cdot\boldsymbol{E}\end{cases} \tag{2.8}$$

其中，$\boldsymbol{e}=[1]_{n\times 1}$。

从随机行走模型的角度看，上式中，$\boldsymbol{F}$ 的作用是修正转移概率，当随机冲浪者进入一个摇摆页面时，下一步它可以使用给定的概率分布 $\boldsymbol{v}$ 随机地跳转到网络图中任意一个其它页面。$\boldsymbol{E}$ 的作用是，在任一时间，一个冲浪者在访问某些节点时，可以以概率 $(1-d)$ 随机跳转至任一页面而不用按照链接结构来访问。

值得注意的是，从 $\boldsymbol{F}$ 和 $\boldsymbol{E}$ 中人为引入的边并不需要具体化。因此引入 $\boldsymbol{F}$ 和 $\boldsymbol{E}$ 对于矩阵在计算过程中的使用并不会造成太大影响（其影响可忽略不计）【30】。

根据页面重要性的两点原则，指向页面链接数量的多少和质量的高低是衡量页面重要性的核心标准，即 PageRank 算法强调的是被多少地方链接而不是链接到多少地方，反应在矩阵中体现为矩阵中的某个数值应该为行数 i 所代表的结点被列数 j 所代表的结点所链接的概率，即 j 指向 i 的概率。可上述矩阵 $\boldsymbol{P}^{''}$ 的意义正好相反，因此需要将 $\boldsymbol{P}^{''}$ 变换一下【31】，这样也是为了后面的计算方便，取 $\boldsymbol{P}^{''}$ 的转置，得到

$$\boldsymbol{A}=(\boldsymbol{P}^{''})^{T} \tag{2.9}$$

在实际网络中，因为页面数目巨大，而每个页面的链接数目又很有限，由此可以断定这是一个维数巨大并且及其稀疏的矩阵。

如果将整个网络页面的 PR 值看成一个向量 $\boldsymbol{p}$，设初始向量为 $\boldsymbol{p}^{(0)}$（这个值可以任意选取，它对最终结果无任何影响），那么点击一次链接后停留在每个页面的概率为：

$$\boldsymbol{p}^{(1)}=\boldsymbol{A}\boldsymbol{p}^{(0)} \tag{2.10}$$

之后一直迭代下去，有

$$\boldsymbol{p}^{(k)}=\boldsymbol{A}^{k}\boldsymbol{p}^{(0)} \tag{2.11}$$

最终，在经过 k 次点击后，每个页面停留的概率会趋于稳定，即无论如何迭代，都存在 $\boldsymbol{p}^{(k)} = \boldsymbol{p}^{(k-1)}$，这样我们可以得到：

$$\boldsymbol{p}^{(k)} = \boldsymbol{p}^{(k-1)} = \boldsymbol{A}\boldsymbol{p}^{(k-1)} = \boldsymbol{A}\boldsymbol{p}^{(k)} \tag{2.12}$$

而整个迭代的过程，就是在寻求方程 $\boldsymbol{p} = \boldsymbol{A}\boldsymbol{p}$ 的解。即无论 $\boldsymbol{p}^{(0)}$ 是多少，在迭代无限多次之后，一定会取得令 $\boldsymbol{p} = \boldsymbol{A}\boldsymbol{p}$ 成立的 $\boldsymbol{p}$ 值。综上所述，我们可以得到：

$$\boldsymbol{p} = \boldsymbol{A}\boldsymbol{p} \tag{2.13}$$

由公式(2.13)可知，通过数学模型解决 PageRank 算法的关键是求解矩阵的特征向量，同时可以发现该矩阵存在一个特征值为 1。由文献【26】,【32】进一步可知 1 就是该矩阵的最大特征值，于是问题最终演化为如何求解矩阵对应于最大特征值为 1 时的特征向量。

传统求特征向量的方法是首先求解特征方程的全部特征值，然后对每个特征值依次求解其对应齐次线性方程组的基础解系。但因为高次多项式求特征值和基础解系是极其困难的，所以在这里，我们采用另一种方法计算特征向量：幂法（Power Method）。实际中，Google 也是这么做的。

幂法是一种计算矩阵主特征值（矩阵按模最大的特征值）及对应特征向量的经典迭代算法，特别适用于大型稀疏矩阵。首先我们从数学角度介绍一下幂法：

设实矩阵 $\boldsymbol{A}$ 有一个完全的特征向量组，其特征值为 λ_1， λ_2，…， λ_n，且满足条件 $|\lambda_1| \geq |\lambda_2| \geq \ldots \geq |\lambda_n|$，对应特征向量为 $\boldsymbol{x_1}$， $\boldsymbol{x_2}$，…， $\boldsymbol{x_n}$。

幂法的基本思想是任取一个非零的初始向量 $\boldsymbol{v}^{(0)}$，由矩阵 $\boldsymbol{A}$ 构造一向量序列

$$\begin{cases} \boldsymbol{v}^{(1)} = \boldsymbol{A}\boldsymbol{v}^{(0)} \\ \boldsymbol{v}^{(2)} = \boldsymbol{A}\boldsymbol{v}^{(1)} = \boldsymbol{A}^2\boldsymbol{v}^{(0)} \\ \boldsymbol{v}^{(k+1)} = \boldsymbol{A}\boldsymbol{v}^{(k)} = \boldsymbol{A}^{k+1}\boldsymbol{v}^{(0)} \end{cases} \tag{2.14}$$

称为迭代向量。由假设， $\boldsymbol{v}^{(0)}$ 可表示为

$$\boldsymbol{v}^{(0)} = a_1\boldsymbol{x_1} + a_2\boldsymbol{x_2} + \ldots + a_n\boldsymbol{x_n}(a_1 \neq 0) \tag{2.15}$$

于是

$$\begin{cases} \boldsymbol{v}^{(1)} = \boldsymbol{A}\boldsymbol{v}^{(0)} = a_1\lambda_1\boldsymbol{x_1} + a_2\lambda_2\boldsymbol{x_2} + \ldots + a_n\lambda_n\boldsymbol{x_n} \\ \boldsymbol{v}^{(2)} = \boldsymbol{A}\boldsymbol{v}^{(1)} = a_1\lambda_1^2\boldsymbol{x_1} + a_2\lambda_2^2\boldsymbol{x_2} + \ldots + a_n\lambda_n^2\boldsymbol{x_n} \\ \ldots\ldots \\ \boldsymbol{v}^{(k)} = \boldsymbol{A}\boldsymbol{v}^{(k-1)} = a_1\lambda_1^k\boldsymbol{x_1} + \ldots + a_n\lambda_n^k\boldsymbol{x_n} = \lambda_1^k[a_1\boldsymbol{x_1} + \sum_{i=2}^{n} a_i(\frac{\lambda_i}{\lambda_1})^k\boldsymbol{x_i}] = \lambda_1^k(a_1\boldsymbol{x_1} + \varepsilon_k) \end{cases} \tag{2.16}$$

其中

$$\varepsilon_k = \sum_{i=2}^{n} a_i(\frac{\lambda_i}{\lambda_1})^k\boldsymbol{x_i} \tag{2.17}$$

由假设

$$\left|\frac{\lambda_i}{\lambda_1}\right| < 1(i = 2,3,...,n) \Rightarrow \lim_{k\to\infty} \varepsilon_k = 0 \tag{2.18}$$

从而

$$\lim_{k\to\infty} \frac{\boldsymbol{v}^{(k)}}{\lambda_1^k} = a_1 \boldsymbol{x}_1 \tag{2.19}$$

这说明当 k 充分大时，有

$$\begin{cases} \boldsymbol{v}^{(k)} \approx a_1 \lambda_1^k \boldsymbol{x}_1 \\ \boldsymbol{v}^{(k+1)} = \boldsymbol{A}\boldsymbol{v}^{(k)} \approx a_1 \lambda_1^k \boldsymbol{A}\boldsymbol{x}_1 = \lambda_1 a_1 \lambda_1^k \boldsymbol{x}_1 = \lambda_1 \boldsymbol{v}^{(k)} \end{cases} \tag{2.20}$$

由上式发现，矩阵 $\boldsymbol{A}$ 特征值为 λ_1 对应的特征向量，也等于向量 $\boldsymbol{v}^{(k)}$（由 $\boldsymbol{A}\boldsymbol{v}^{(k)} = \lambda_1 \boldsymbol{v}^{(k)}$ 可知），即在 k 充分大时，有

$$\lim_{k\to\infty} \boldsymbol{v}^{(k)} = \boldsymbol{x}_1 \tag{2.21}$$

现在我们回到 PageRank 算法的计算中，前面已经提到，求页面的 *PR* 值即相当于求解矩阵 $\boldsymbol{A}$ 对应特征值为 1 的特征向量，而特征值 1 又是矩阵 $\boldsymbol{A}$ 的最大特征值，我们可以等效理解为上述推导过程中的 λ_1，这样，就把一个计算多项式根的问题通过幂法的引入简化为了连续迭代求矩阵和向量乘积的问题，从而减少了计算的复杂度。所以，标准 PageRank 算法计算主特征向量的方法就是不断重复地计算 $\boldsymbol{v}^{(k)} = \boldsymbol{A}^k \boldsymbol{v}^{(0)}$，直至收敛。

从理论上讲，要想最真实的反应出页面重要性的 *PR* 值，那么应该计算出上式的极限值。但是计算极限值显然需要经过大量的迭代计算，这必然会影响搜索引擎的总体运行速度。另一方面算法所需要的计算结果只要能够正确反应出页面的重要性差别即可，并不需要精确计算出每个页面的 *PR* 值为多少。因此我们在计算中，设定一个相对误差（Relative Tolerance），当最后两次迭代结果的向量差小于该误差时，便可认为已经得到最终向量，也就是说向量已经完全收敛了。

我们还应注意到，迭代次数的多少反映着幂法收敛速度的快慢。影响幂法收敛速度的因素有很多，首先，选取的起始向量 $\boldsymbol{v}^{(0)}$ 可以影响收敛速度；另外，根据文献【33】，d（阻尼系数）等于次大特征值 λ_2，所以 $d/1$（实际为 λ_2/λ_1，即 次极大值/极大值，而在这里，$\lambda_2 = d$，$\lambda_1 = 1$）就约等于收敛速度，d 也就当然会影响收敛速度。当随机转移概率 $(1-d)$ 很大时，幂法的可行性很高。然而，对于一个大的转移概率 $(1-d)$，很多非相关信息链接的影响也在增加，因此页面将获得不合理的高等级值。所以，对于 d 的取值要结合实际慎重而定。

下面我们通过一个简单的例子来具体化上述内容，如图 2-3 所示：

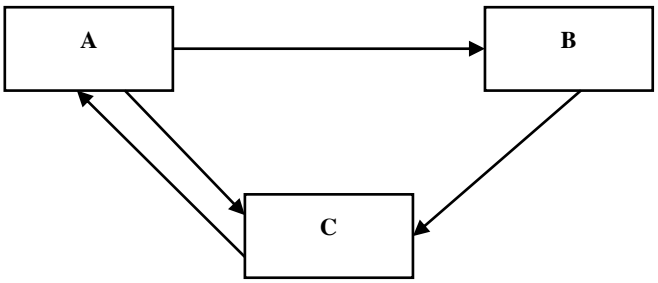

图 2-3 PageRank 算法思想示意图

Fig.2-3 Schematic diagram of PageRank algorithm

这是一个由三个页面 A，B，C 组成的小网络，A 链接到 B 和 C，B 链接到 C，C 链接到 A。为方便计算，这里取阻尼系数 $d=0.5$。我们可以得到如下计算 *PR* 值的方程：

$$\begin{cases} PR(A)=0.5PR(C)+0.5 \\ PR(B)=0.5(PR(A)/2)+0.5 \\ PR(C)=0.5(PR(A)/2+PR(B))+0.5 \end{cases}$$

求解，得：

$$\begin{cases} PR(A)=1.0769 \\ PR(B)=0.7692 \\ PR(C)=1.1538 \end{cases}$$

对于这个只有三个页面的简单范例来说，通过方程组很容易求得 *PR* 值。但实际上，互联网包含数以亿计的页面，是不可能解方程组的，实际中仍需使用幂法通过迭代来计算页面的 *PR* 值。再次使用上例来说明迭代计算，如表 2-1 所示，这里设每个页面的初始值为 1。

表 2-1 PageRank 迭代计算过程

Table2-1 Iterative calculation process of PageRank

迭代次数	$PR(A)$	$PR(B)$	$PR(C)$
0	1	1	1
1	1	0.75	1.25
2	1.125	0.75	1.125
3	1.0625	0.78125	1.15625
4	1.07812	0.76525	1.15625
5	1.07812	0.769531	1.15234
6	1.07617	0.769531	1.15543
7	1.07715	0.769043	1.15381
8	1.0769	0.769287	1.15381
9	1.0769	0.769226	1.15387
10	1.07693	0.769226	1.15384

由上表发现，到第 7 次迭代时，迭代结果和第 6 次迭代的结果基本相同，我们可认为

此时已收敛，迭代可以停止。一般情况下，50 到 100 次的迭代计算才能得到令人满意的页面 *PR* 值[34]。

2.3 PageRank 算法的加速

由于互联网规模巨大，PageRank 算法只能在页面搜集完成之后，搜索查询提交之前花费几天甚至几周的时间进行离线计算。而在目前应用广泛的主题敏感（Topic-Sensitive）PageRank 算法和个性化（Personalized）PageRank 算法[35-37]中，因为存在多种类型的页面，而每一类型的页面都需要一个 PageRank 向量来表示，所以需要进行大量运算，为了提高这些算法的效率，我们不得不加速。但是，对 PageRank 算法进行加速又面临着许多挑战：

（1）幂法的收敛率（Convergence Rate）已经很快（通常情况下，$|\lambda_2|/|\lambda_1|=0.85$[38]），在一个含有八千万页面的网络数据集中，幂法经过 60 次迭代便可收敛，加速一个收敛率已经很快的算法的确很难；

（2）由于网络矩阵规模和稀疏度的限制，导致很多其他传统特征值快速求解方法（如逆迭代法）并不适用解决此类问题。

下面我们主要介绍 2 种加速算法：

（1）自适应法

我们发现，在一个庞大的页面数据集中，页面 *PR* 值呈现明显的不均匀分布（Nonuniform Distribution），即各个页面分值有着很大的差别，除此之外，大多数页面有着较快的收敛速度，而其余小部分页面的收敛速度较慢，进一步分析得到，通常收敛速度慢的页面有着较高的 *PR* 值。根据文献【26】，对于一个有着 280000 个页面的网络来说，当阻尼系数为 0.85，相对误差为 0.001 时，有 80%的页面在约 15 次迭代后就会收敛，而其余页面需要将近 30 次迭代才能收敛。于是，Kamvar[9]等人根据上述情况提出了一种自适应法（Adaptive Method），其基本思想是在幂法的基础上进行改进，基于迭代过程中某些页面的 PageRank 向量比其它页面的 PageRank 向量收敛快的事实，通过锁止这类页面的迭代计算来减少运算的规模和时间。

宏观上，我们需要构建两个页面集，一个存放未收敛的页面，一个存放已收敛的页面，如下：

$$PR'(u)=\begin{cases} PR(u), & PR(u)\text{收敛} \\ d\sum\limits_{v\in B(u)}\dfrac{PR(v)}{N(v)}+(1-d), & \text{其他} \end{cases} \tag{2.22}$$

假设我们通过幂法经过 k 次迭代后，得到了整个网络页面的 PageRank 向量 $\boldsymbol{p}^{(k)}$，此时，在这个总共包含 n 个节点的网络中，m 个节点未收敛，用节点集 N 表示；$n-m$ 个节点已收敛，用节点集 C 表示。现在我们希望通过再一次迭代得到 $\boldsymbol{p}^{(k+1)}$。

为此，我们需要将 $\boldsymbol{p}^{(k)}$ 分解为两部分，即 $\boldsymbol{p}^{(k)}=\left(\boldsymbol{p}_N^{(k)},\boldsymbol{p}_C^{(k)}\right)$，其中 $\boldsymbol{p}_N^{(k)}$ 是一个 m 维向量对应着 m 个未收敛的节点，$\boldsymbol{p}_C^{(k)}$ 是一个 $n-m$ 维向量对应着 $n-m$ 个已收敛的节点。类似的

将概率转移矩阵 $\boldsymbol{A}$ 分解为两部分，即 $\boldsymbol{A}=(\boldsymbol{A}_N,\boldsymbol{A}_C)$，其中 $\boldsymbol{A}_N$ 是一个 $m\times n$ 阶矩阵对应着 m 个未收敛的节点， $\boldsymbol{A}_C$ 是一个 $(n-m)\times n$ 阶矩阵对应着 $n-m$ 个已收敛的节点。那么，在下一次迭代求 $\boldsymbol{p}^{(k+1)}$ 时我们可以采用如下公式：

$$\begin{pmatrix}\boldsymbol{p}_N^{(k+1)}\\ \boldsymbol{p}_C^{(k+1)}\end{pmatrix}=\begin{pmatrix}\boldsymbol{A}_N\\ \boldsymbol{A}_C\end{pmatrix}\cdot\begin{pmatrix}\boldsymbol{p}_N^{(k)}\\ \boldsymbol{p}_C^{(k)}\end{pmatrix} \tag{2.23}$$

当然，因为 $\boldsymbol{p}_C^{(k)}$ 已经收敛，我们不需要通过迭代计算 $\boldsymbol{p}_C^{(k+1)}$，因此，我们可以将上式简写为：

$$\begin{cases}\boldsymbol{p}_N^{(k+1)}=\boldsymbol{A}_N\,\boldsymbol{p}^{(k)}\\ \boldsymbol{p}_C^{(k+1)}=\boldsymbol{p}_C^{(k)}\end{cases} \tag{2.24}$$

通过上述公式，我们发现只需迭代计算未收敛的部分 $\boldsymbol{p}_N$ 即可。随着 m 越来越小，矩阵 $\boldsymbol{A}_N$ 的规模也相应减小，这样很多计算便可以逐渐省略，大大节省了时间。

自适应法可以通过避免大量重复的迭代计算来显著减少运算时间，文献【9】指出，该方法在实际应用中可以减少 30%的运算时间。当然，自适应法也有一些不足之处，例如，一个页面的 *PR* 值可能在多次迭代后趋于稳定，但在后来某一次迭代计算后又会显著变化，而自适应法往往将这类节点视为已收敛节点，将其趋于稳定的值视为该页面的最终 *PR* 值，从而对最终结果产生影响。

（2）外推法

如果在幂法中，能让每次迭代后的 $\boldsymbol{v}^{(k)}$ 更加逼近 $\boldsymbol{x}_1$，将会使幂法收敛更快。根据公式(2.21)可以看出，当 k 足够大时， $\boldsymbol{v}^{(k)}$ 退化为 $\boldsymbol{x}_1$，而 $\boldsymbol{v}^{(0)}$ 又可以表示为 $\boldsymbol{x}_1$ 到 $\boldsymbol{x}_n$ 的线性组合。可以看出， k 逐渐增大的过程，实际上是逐步舍弃 $(\lambda_i/\lambda_1)^k$ 的影响过程。根据这一思想，斯坦福大学的 Kamvar 等人【11】提出了一种外推法（Extrapolation Method），他们作如下假设， $\boldsymbol{v}^{(k-2)}$ 能被最大的两个特征向量组合表示，即忽略了 $\boldsymbol{x}_2$ 以后向量的影响：

$$\boldsymbol{v}^{(k-2)}=\boldsymbol{x}_1+a_2\boldsymbol{x}_2 \tag{2.25}$$

由于矩阵最大特征值是 1，因此有：

$$\begin{cases}\boldsymbol{v}^{(k-1)}=\boldsymbol{A}\boldsymbol{v}^{(k-2)}=\boldsymbol{x}_1+\alpha_2\lambda_2\boldsymbol{x}_2\\ \boldsymbol{v}^{(k)}=\boldsymbol{A}\boldsymbol{v}^{(k-1)}=\boldsymbol{x}_1+a_2\lambda_2^2\boldsymbol{x}_2\end{cases} \tag{2.26}$$

定义：

$$\begin{cases}g_i=\left(v_i^{(k-1)}-v_i^{(k-2)}\right)^2\\ h_i=v_i^{(k)}-2v_i^{(k-1)}+v_i^{(k-2)}\end{cases} \tag{2.27}$$

这里 v_i 代表向量 $\boldsymbol{v}$ 的第 i 个元素，利用公式(2.26)和(2.27)得到：

$$\begin{cases}g_i=a_2^2(\lambda_2-1)^2(x_2)_i^2\\ h_i=a_2(\lambda_2-1)^2(x_2)_i\end{cases} \tag{2.28}$$

接下来定义 f_i 为等式 g_i/h_i：

$$f_i = \frac{g_i}{h_i} = \frac{a_2^2(\lambda_2 - 1)^2(x_2)_i^2}{a_2(\lambda_2 - 1)^2(x_2)_i} = a_2(x_2)_i \tag{2.29}$$

因此：

$$\boldsymbol{f} = a_2\boldsymbol{x_2} \tag{2.30}$$

从公式(2.25)可得到 $\boldsymbol{x_1}$ 的一个闭合解（Closed-Form Solution）：

$$\boldsymbol{x_1} = \boldsymbol{v}^{(k-2)} - a_2\boldsymbol{x_2} = \boldsymbol{v}^{(k-2)} - \boldsymbol{f} \tag{2.31}$$

然而，由于这个解是基于 $\boldsymbol{v}^{(k-2)}$ 能够被最大两个特征向量组合表示的假定，所以等式(2.31)只是给出了向量 $\boldsymbol{x_1}$ 的近似值。当然，随着 k 值逐渐变大，这个近似值将会越来越接近向量 $\boldsymbol{x_1}$ 的真实值。

由于上述外推法相当于应用了著名的 Aitken（埃特金）线性加速法[39]，所以该外推法也被称为 Aitken 外推法。该算法的新颖之处在于 Aitken 加速法的推导过程以及在计算主特征向量时充分利用 $\boldsymbol{v}^{(k-2)}$ 能被最大的两个特征向量组合表示这个假设。

除此之外，我们还可以对上述推导进行一些修改，例如，通过执行一个二阶 ε 加速算法[40]，引入 ε 外推法（Epsilon Extrapolation）。Kamvar 等人基于马尔科夫矩阵 $\boldsymbol{A}$ 仅有 3 个特征向量 $\boldsymbol{x_1}$，$\boldsymbol{x_2}$，$\boldsymbol{x_3}$，且 $\boldsymbol{v}^{(k-3)}$ 可以近似表示为这 3 个特征向量的线性组合的假设，提出了二次外推（Quadratic Extrapolation）[11]。

外推法是从特征值和特征向量的关系出发，在迭代计算的过程中进行加速，虽然文献【41】指出外推法能显著减少运算时间，但是加速环节需要额外的计算时间和存储空间，所以该算法只能周期性的进行加速。

除了上述从特征值和特征向量的角度在迭代计算过程中进行加速外，Sepandar D. Kamvar，Taher H. Haveliwala 等人[12]还提出了一种从网络图结构角度去加速的方法：块结构法（Block Structure Method），还有美国布鲁克林工艺大学 CIS 部 YenYu Chen，Qingqing Gan，Torsten Suel[10]提出了基于 I/O 技术的 PageRank 加速算法。

2.4 PageRank 算法分析

PageRank 算法作为目前最成功的一种链接分析方法，自其诞生以来，一直受到国内外学者和研究机构的广泛关注，该算法通过分析网络链接结构来获得网络中的权威页面，并在商用搜索引擎 Google 中大获成功，掀起网络链接分析的高潮。但由于该算法仅仅利用了网络的链接结构，其缺点也不可忽视，如主题漂移，偏重旧页面，无法判断页面内容上的相似性等问题。

2.4.1 PageRank 算法的优点

PageRank 算法通过页面间的链接结构来评价页面的重要性，在一定程度上避免和减少了人为因素对排序结果的影响；PageRank 算法离线计算页面 *PR* 值，在用户查询时仅根

据关键字匹配来获得页面集合，因此具有很快的响应速度；一个页面只能通过别的页面对其引用来增加自身的 *PR* 值，且算法的均分策略使得一个页面的引用越多，被引用页面所获得的 *PR* 值就越少，因此，算法可以有效避免那些为了提高网站搜索排名而故意使用链接的行为。

2.4.2 PageRank 算法的缺点

PageRank 算法在 Google 搜索引擎中的成功运用，说明其是高效、可行的。但由于其完全基于链接分析，且链接信息相对静态，没有考虑页面使用的动态信息，因此该算法还存在一些缺陷，主要可归纳为[4]：

（1）主题漂移问题

PageRank 算法仅利用网络的链接结构，无法区分页面中的链接与该页面的主题是否相关，即无法判断页面之间内容的相似相关性，且算法根据向外链接平均分配权值的策略使得主题不相关页面获得与主题相关页面同样的重视度，这样容易导致在最后的检索结果中出现大量与查询主题无关却又具有很高 *PR* 值的页面，即出现主题漂移现象。

（2）偏重旧页面问题

决定页面 *PR* 值的一个主要因素是指向该页面链接个数的多少。一些刚放到互联网上的新页面，因为存在时间短暂，导致很多页面还没有引用它，因此由于链接数目的限制，使得这样一些包含更新更有价值信息的新页面很难出现在检索结果的前面，从而不能获得与其实际价值相符的排名，这正好与用户需求相反。而一些旧页面因为存在时间长，被引用的可能性也较高，从某种程度来讲，其 *PR* 值较高，即出现偏重旧页面现象。

（3）忽视用户个性化问题

PageRank 算法在设计之初，没有考虑到用户的个性化需要。但一个页面能否被用户再次浏览，很大程度上取决于用户的兴趣度。个性化搜索引擎的兴起，对 PageRank 排序算法提出新的挑战。

（4）依赖大型，综合性页面

以.com 结尾的网站，由于这类网站通常是大型、综合性网站，可以比其他类型的网站获得更多链接。而事实上，这类页面通常涉及的面多而不专，相比之下，某些专业网站对问题的阐述更有权威性且与检索的主题更贴切。

2.5 PageRank 算法改进的研究现状

算法的改进可归纳为两类：其一，基于算法理论的改进，可转化为求解矩阵特征向量和大型稀疏矩阵线性方程组的问题，如前面介绍的各种加速算法。其二，针对互联网实际特点的改进。下面我们主要针对第二种改进思路进行研究阐述。

2.5.1 主题漂移问题

（1）TS-PageRank 算法

斯坦福大学计算机科学系的 Taher H.Haveliwa[3] 于 2002 年提出了一种基于主题敏感

（Topic-Sensitive）的 PageRank 算法。算法的核心思想是根据链接到页面主题相关性的高低来传递 *PR* 值，而不采用平均传递策略。

TS-PageRank（Topic-Sensitive PageRank）算法考虑到一些页面在某些领域被认为是重要的，但并不表示它在其它领域也同样是重要的。该算法首先根据 ODP（Open Directory Project）[4] 提出的 16 个基本分类标准对页面进行分类，并针对每个页面的 ODP 分类，给出一个主题敏感值，然后离线计算出每个页面在各个分类下的 *PR* 值。最后根据用户输入的查询请求及上下文，计算页面的综合得分。

1. 离线计算每个页面的 *PR* 值

根据 ODP 的 16 个基本分类标准对所有页面进行分类，设主题敏感值 $A_j = 1/|F_j|$，其中 F_j 为当前页面 u 在某个 ODP 分类 C_j $(j = 1,2,\cdots 16)$ 下的链出页面集合，j 为 16 个基本分类标准。得：

$$PR_j(u) = d \sum_{v \in B(u)} \frac{PR(v)}{N(v)} + (1-d) \cdot A_j \tag{2.32}$$

$PR_j(u)$ 表示页面 u 在 ODP 分类 C_j 下的 *PR* 值。

2. 在线计算页面综合得分

当用户输入某查询信息后，该算法可以根据用户的输入信息及查询上下文来确定页面的最终 *PR* 值，公式如下：

$$PR(u) = \sum_{j=1}^{16} q_j \cdot PR_j(u) \tag{2.33}$$

其中 q_j 是当前查询 u 属于某个 ODP 分类的可能性大小。

该算法基于页面分类思想，根据用户的查询请求和相关上下文来判断与用户查询相关的主题，返回与查询主题相关的页面，从而提高排序结果的准确性。

在知道查询 u 的上下文时，该算法可以有效避免一些明显的主题漂移现象，比如在查询关键字为“天堂”时，当有了上下文的指引，算法能明确的区分用户查询的是①天堂电影下载软件；②天堂网络游戏公司；③杭州天堂伞制造商；④汉语词语。但是，一旦缺少上下文的指引，该算法会把查询 u 当做上下文，此时便无法有效的对查询主题进行归类。

（2）MP-PageRank 算法

华盛顿大学计算机科学与工程系的 Matthew Richardson 和 Pedro Dominggos [5] 认为用户从一个页面跳到另一个页面会受到当前页面内容和查询主题的的影响，所以提出了一种结合链接和内容信息的 PageRank 算法（简称 MP- PageRank 算法）。算法的计算形式为：

$$PR_q(u) = d \sum_{v \in B(u)} PR_q(v) \cdot p(v \to u) + (1-d) p_q'(u) \tag{2.34}$$

式中，$p(v \to u)$ 表示用户在查询主题 q 下从页面 v 跳转到页面 u 的可能性。$p_q'(u)$ 表示用户正在访问的当前页面没有链出链接指向页面 u 时，下一步浏览直接跳转到页面 u 的

可能性。$PR_q(u)$表示页面u在查询主题q下的PR值。$p(v \to u)$和$p_q^{'}(u)$采用一个查询q和页面u的相关函数$R_q(u)$来计算，公式如下：

$$\begin{cases} p(v \to u) = \dfrac{R_q(u)}{\sum_{k \in W} R_q(k)} \\ p_q^{'}(u) = \dfrac{R_q(u)}{\sum_{k \in F_v} R_q(k)} \end{cases} \tag{2.35}$$

其中，W表示页面v的链出页面集，F_v表示网络中的所有页面集。相关函数$R_q(u)$理论上可以是任意的，但一般取值为查询主题q的关键字在页面u中出现的次数。

与 PageRank 算法相比，该算法增加考虑了用户从一个页面直接跳转到一个与之非直接相邻但内容相关页面的情况。同时，在传递页面PR值时，除了充分利用页面之间的链接结构外，还考虑到页面之间的主题相关性，使得PR值的传递更加精确。

（3）Query-Based PageRank 算法

在页面链接结构中，一个页面的出链接能很好的代表此页面的主题。所以我们可以根据向外链接文本在页面中所占权重比例来分配权值，权重比例越大，则分配获得的权重越大，反之则越小。基于这一思想，高琪，张永平[14]通过采用$tf\text{-}idf$策略计算主题的相关权重，提出了一种与查询主题相关的 Query-Based PageRank 算法。

$tf\text{-}idf$（Term Frequency-Inverse Document Frequency）[42]是一种用于信息搜索和信息挖掘的常用加权技术。在搜索、文献分类和其他相关领域有广泛的应用。$tf\text{-}idf$实际上是$tf*idf$，tf代表词频（Term Frequency），idf代表逆文档频率（Inverse Document Frequency）。tf表示目标词条在文档d中出现的频率。idf的主要思想是如果包含目标词条的文档越少，则idf越大，以此说明目标词条具有很好的类别区分能力。

令u为一页面，$B(u)$是指向u的页面集合。v是$B(u)$中的任一页面。则v中必有一链接指向u。用A代表这一超链接的文本，D表示v中的词条（即所有超链接文本），则超链接文本A在页面v中所占权重比例$W(A|v)$的计算公式如下：

$$W(A|v) = \frac{\sum_{t \in A} tf(t,v) \times idf(t)}{\sum_{t \in D} tf(t,v) \times idf(t)} \tag{2.36}$$

其中$tf(t,v)$为词条t在页面v中所有词条的频数，$idf(t) = \log(N/n_t)$表示反文档频率，N为页面集合中的页面总数，n_t为页面集合中含有词条t的页面数量。

由公式可知，$\sum_{t \in A} tf(t,v) \times idf(t)$为超链接$A$的权重，$\sum_{t \in D} tf(t,v) \times idf(t)$为页面中所有超链接权重之和，即公式用一个超链接文本权重除以页面中所有超链接文本权重之和来计算页面中的各个超链接文本在页面中所占的权重比例，这样便可以合理的分配权重。

接下来用$W(A|v)$代替原公式中的$N(v)$，得到：

$$PR(u)=d\sum_{v\in B(u)}PR(v)\cdot W(A|v)+(1-d) \tag{2.37}$$

（4）基于分块主题的 PageRank 算法

网络中的信息包罗万象，大多数情况下，一个页面中的内容会包含不止一类的主题，比如在搜狐的页面中，可能包含时政类信息，体育类信息，也可能含有广告等信息，这些信息一般分布在同一页面的不同位置。日常检索中，我们通常想得到的只是这某一具体信息，试想一下，在浏览页面时，用户点击从页面 A 到页面 B 的链接一般意味的仅仅是页面 A 的某部分与页面 B 的某部分之间存在一定的关系。因此将整个页面作为一个基本的信息获取单位并不合理，所以要想获取准确的页面信息，必须对给定的页面进行进一步细分。

为此，白似雪，刘华斌[15]针对传统 PageRank 算法中平均分配策略的不足，提出了一种基于分块主题的 PageRank 算法，算法的基本思想是考虑到同一页面内属于不同分块的出链有着不同的权重，首先对页面进行分块操作，然后给同一页面不同分块的出链接以相应的权重，即将链接关系从两个完整的页面之间转移到两个页面的某部分之间。该算法主要由以下 3 个步骤组成：

1. 页面分块重要模型的实现；

2. 利用该模型对页面集的所有页面进行处理；

3. *PR* 值的计算。

对页面分块的实现我们利用基于视觉特征的 VIPS（Vision-Based Page Segmentation）算法，具体的方法可参考文献【43-45】。模型实现后，首先利用 VIPS 算法对所有页面进行划分，再用重要性模型对划分后的页面分块进行处理，得到各块的重要性等级，据此计算出引用重要性权值，用$W(v,u)$表示，最后根据此权值来计算页面的 *PR* 值。改进后 PageRank 算法如下：

$$PR(u)=d\sum_{v\in B(u)}PR(v)\cdot f(v,u)+(1-d) \tag{2.38}$$

式中，$f(v,u)$是页面u分到页面v *PR* 值的权重，其形式为：

$$f(v,u)=\frac{W(v,u)}{W(v,u)+\sum_{i=1}^{n}W(v,u_i)} \tag{2.39}$$

式中，$W(v,u)$表示页面v引用页面u的重要性权值，u - u_n表示被页面v所引用的所有页面。

（5）基于层次分类的 PageRank 算法

在传统的链接分析中有这样一种特征：一个与教育相关的 A 页面指向另一个与教育相关的 B 页面较一个与娱乐相关的 C 页面指向与教育相关的 B 页面更为权威、可信。传统 PageRank 算法中，在进行页面排序时，没能很好地考虑页面的分类特性和页面间的相

关性。根据上述特征，李绍华，高文宇[16]提出了一种基于层次分类（Hierarchic Classification）的 PageRank 算法。该算法主要分为以下几步骤：

1. 对页面进行层次分类

这里层次化分类体系仍采用前面提到的 ODP 分类体系，当根据 ODP 分类体系列出其层次结构时，便得到了图 2-4 所示的树状层次结构，图中每一节点的子节点代表子类。页面的特征向量并不与所有的类别特征向量进行比较，而只与部分类别特征向量进行比较，从而大大减少比较次数，提高分类效率。

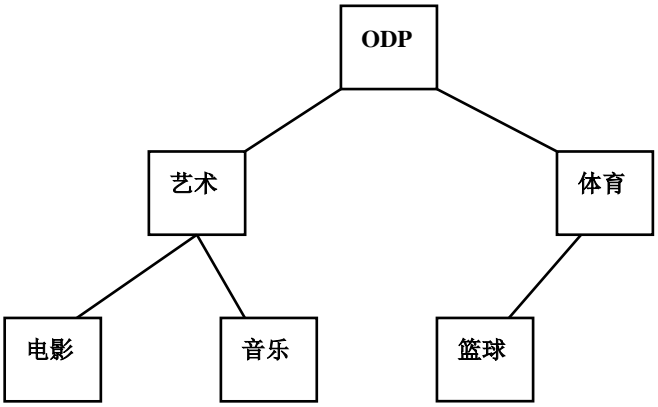

图 2-4 树状层次结构

Fig.2-4 Tree-structured hierachy

2. 计算页面之间的相关性

对任意两个页面，其相关性 $R(A,B)$ 计算如下：

$$R(A,B)=\begin{cases}0.2, & \min(A,B)=1\\ 0.4, & \min(A,B)=2\\ 0.8, & \min(A,B)=3\\ 1, & \min(A,B)=4\end{cases} \tag{2.40}$$

式中，$\min(A,B)$ 表示 A 和 B 的最小公共祖先。即根据两个页面各自所属的类别在层次分类体系中的最小公共祖先不同，相关性值也不同。当然，$R(A,B)$ 的值在实际中是可以灵活变动的。

3. *PR* 值的计算

根据上述页面层次分类和页面相关性计算结果，结合原始公式，得到改进后的公式为：

$$PR(u)=d(PR(v_1)\cdot R(u,v_1)+PR(v_2)\cdot R(u,v_2)+...+PR(v_n)\cdot R(u,v_n))+(1-d) \tag{2.41}$$

其中，v_1 - v_n 是指向 u 的页面，从上式可以看出，页面 u 的重要性取决于指向 u 的页面 $(v_1,v_2,...,v_n)$ 的重要性，而这些页面中与 u 相关性较大者其影响也越大。

除上述所列算法外，还有 Diligenti[46] 等人提出的将访问某链接的概率分成两部分（页面主题相关和链接主题相关）来计算的 Double Focused PageRank 算法；黄德才，戚华春[47] 提出的基于主题相似度模型的 TS-PageRank 算法等等。

2.5.2 偏重旧页面问题

（1）加速评估算法

为了让高质量的新页面能快速地在网络上传播，上海交通大学的张玲博士[17]提出了一种基于时间序列分析的加速评估算法（Accelerated Ranking，AR)。算法的核心思想是通过分析页面基于一个时间序列的 *PR* 值变化情况，预测页面在未来一段时间内的期望值。当用户检索时，搜索引擎将此预测值作为一个页面在检索结果中排名的有效参数，并根据该值的高低决定一个页面在检索结果中的位置。

该算法首先定义一个页面的加速因子 *AR* 为：

$$AR = PR \cdot sizeof(D) \tag{2.42}$$

其中，$sizeof(D)$ 为整个数据集的页面数量。

下面，结合线性回归的方法得到加速后的 PageRank 算法表达式为：

$$PR_{accelerate} = \frac{AR_{last} + B \cdot D}{M_{last}} \tag{2.43}$$

其中，AR_{last} 是页面最近一次的 *AR* 值，B 是该页面在一段时间内 *PR* 值的二次拟合曲线的斜率，对于 $B<0$ 的页面，表明该页面的重要性在降低，其 *PR* 值实质上是被加速削减了；对于 $B>0$ 的页面，表示该页面重要性在增加，其 *PR* 值实质上是被加速提高了。D 为离最近一次页面下载的时间间隔天数（一般可取多次下载间隔的平均值），M_{last} 是最近一次下载的数据集内页面的数目。 *AR* 算法的实质就是给予对 *AR* 值呈上升趋势的页面以额外的奖励，对 *AR* 值呈下降趋势的页面已额外的惩罚。

该加速算法使得网络上有价值和良好发展趋势的页面将以更快的速度传播，而那些陈旧页面的 *PR* 值将加速下滑，保证信息检索中的优胜劣汰。它克服了原始 PageRank 算法对新内容的迟钝性，使一些具有高价值的新页面能第一时间出现在检索结果中，从而帮助用户更好的发掘出有价值的页面，确保向用户提供高质量的网络链接，有效提高了搜索引擎的检索质量。

（2）具有时间反馈的 PageRank 算法

为使在网络上存在较长时间的页面慢慢沉下去，新的页面迅速浮上来，可以考虑在计算页面 *PR* 值时，引入其他因素，比如页面的发布日期等等。为此，戚华春[18]等人提出将页面的存在时间通过搜索引擎搜索的周期数来表示，用一个页面被搜索引擎访问的周期次数来代替时间反馈权值，提出 PageRank-Time 算法。

该算法的核心思想是：一般而言，搜索引擎的搜索周期为半个月到一个月，如果一个页面存在的时间较长，那它将在每个搜索周期里都被搜索到，即页面的存在时间正比于搜索引擎搜索到该页面的次数。为此引入一个时间反馈因子，即

$$W(t) = \frac{e}{T} \tag{2.44}$$

其中，$W(t)$ 为页面的时间反馈因子，T 为一个页面被搜索引擎访问的周期次数，e 为一个比例常数，它的取值不影响最后的 PR 值排名，但在计算过程中，e 的大小影响会迭代收敛的快慢，一般取 $e=(0.15/n)$（n 为页面总数）。

改进后的 PageRank 算法的公式为：

$$PR=d\sum_{i=1}^{n}\frac{PR(T_i)}{C(T_i)}+(1-d)+W_t \tag{2.45}$$

算法通过引入时间反馈因子，克服了提取页面发布时间的困难，能更准确地判断页面日期。同时，页面 PR 值的大小受页面发布时间长短的影响，有利于旧页面的下沉，新页面的迅速上浮。

2.6 本章小结

本章是本文的基础，首先对Google搜索引擎的PageRank排序算法进行了详尽的剖析，详细介绍了该算法的体系结构、形成思路和计算方法。并在此基础上，介绍了多种针对传统 PageRank 算法的加速算法。接着指出了传统 PageRank 算法的优缺点，并针对其不足之处，主要从主题漂移和偏重旧页面两个方面介绍了多种国内外现存经典改进方法。

3 改进的 PageRank 算法

传统 PageRank 算法对页面 *PR* 值的分配是没有针对性的，它把自己的权值平均分配给各个页面，这种平均分配 *PR* 值的策略，在一定程度上影响了页面的排序质量。基于此，本章主要介绍一种根据各个页面重要性不同去分配 *PR* 值的方法。

3.1 改进 PageRank 算法基本思想

通过前几章的分析，我们得到这样一个事实：被引用越多的页面，其 *PR* 值越高。换句话说，被引用越多的页面，其重要性也越高。在传统 PageRank 算法中，它将当前页面的 *PR* 值平均分配给它链接的所有页面。事实上由于某页面各个出链页面的重要性存在差别，所以这种平均分配 *PR* 值的方法，在一定程度上影响了页面的排序质量。基于这一事实，我们改进传统 PageRank 算法在分配 *PR* 值时没有针对性，单纯将自己权值平分的思路，而将 *PR* 值采用非平均分配，即重要性越高的页面，分配的权值将越高；重要性越低的页面，分配的权值将越低。

以图 3-1 来说明改进算法的基本思想：

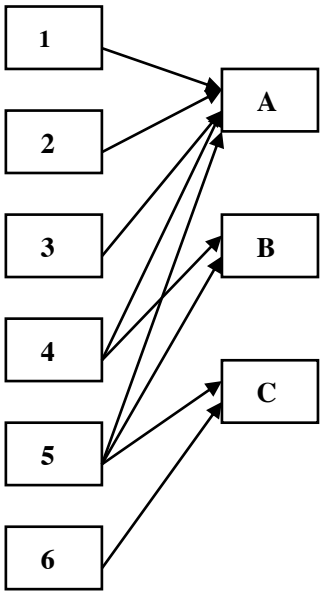

图 3-1 改进算法思想示意图

Fig.3-1 Schematic diagram of improved algorithm

在传统 PageRank 算法中，页面 4 的 *PR* 值将平均分给页面 A 和页面 B，由上述内容可知，由于页面 A 被引用的次数明显多于页面 B，即页面 A 的重要性明显高于页面 B，因此，将页面 4 的 *PR* 值平均分给页面 A 和页面 B 是显然不合理的，在一定程度上影响了页面的排序质量。

那么如何分配页面 4 的 *PR* 值才合理呢？由图可知，页面 A 有 5 个入链，即有 5 个页面（分别为页面 1，页面 2，页面 3，页面 4，页面 5）引用了页面 A 的内容；页面 B 有 2 个入链，即有 2 个页面（分别为页面 4，页面 5）引用了页面 B 的内容，因此，我们可以

将入链数等价于被引用次数来反应页面的重要程度，即页面的入链数越多，该页面的重要性程度越高。那么在图 3-1 中，页面 A 将分得 $\frac{5}{5+2}$ 的 $PR(4)$，页面 B 将分得 $\frac{2}{5+2}$ 的 $PR(4)$，这样的分配才显得比较合理。

3.2 改进 PageRank 算法描述

结合上节提出的改进思想，参照 PageRank 算法计算公式，改进算法可描述为：

$$PR(u)=\sum_{v\in B(u)} f_v \cdot PR(v) \tag{3.1}$$

式中，f_v 表示页面 v 分配给页面 u 的 PR 值权重，通过前面内容讨论可知：

$$f_v=\frac{IN_u}{\sum_{j=1}^{m} IN_j} \tag{3.2}$$

其中，IN_u 代表页面 u 的入链数，$\sum_{j=1}^{m} IN_j$ 代表页面 v 指向的所有页面的链入页面数之和。

对于改进的 PageRank 算法，由于同样存在随机跳转问题，因此，经过修正后的改进 PageRank 算法表达式为：

$$PR(u)=d\sum_{v\in B(u)} f_v \cdot PR(v)+(1-d) \tag{3.3}$$

式中，各参数的含义同 PageRank 算法相同。由于需要计算入链数，因此改进算法的计算复杂度稍大于传统 PageRank 算法。

根据上述公式，在图 3-1 中，页面 PR 值的分配情况应为：

$$\begin{cases} PR(A)=d(PR(1)+PR(2)+PR(3)+\frac{5}{5+2}PR(4)+\frac{5}{5+2+2}PR(5))+(1-d) \\ PR(B)=d(\frac{2}{5+2}PR(4)+\frac{2}{5+2+2}PR(5))+(1-d) \\ PR(C)=d(\frac{2}{5+2+2}PR(5)+PR(6))+(1-d) \end{cases}$$

3.3 对比实验

下面我们通过一个由 5 个页面构成的简单试验网络（如图 3-2）来对比传统 PageRank 算法和改进 PageRank 算法的优劣。

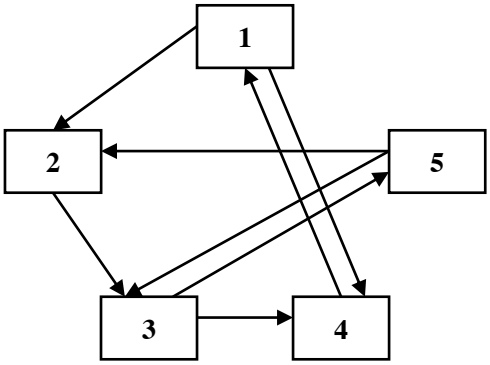

图 3-2 算法对比实验图

Fig.3-2 Diagram of algorithm contrast

假设在两算法中，5 个页面的初始 *PR* 值为 $PR=[0.2,0.2,0.2,0.2,0.2]$，迭代稳定后（迭代次数 20 次左右）的结果如表 3-1 所示。

表 3-1 传统算法与改进算法的对比

Table3-1 Comparsion of traditional algorithm and improved algorithm

页面编号	预计排名	传统算法 *PR* 值	传统算法排名	改进算法 *PR* 值	改进算法排名
1	4	0.2222	3	0.2518	2
2	3	0.1805	4	0.1762	4
3	2	0.2393	1	0.2188	3
4	1	0.2262	2	0.2611	1
5	5	0.1318	5	0.0921	5

3.4 实验结果及分析

在图 3-2 的例子中，页面 2，页面 3，页面 4 均有 2 个入链，但页面 4 的一个链接是从页面 3 链入的，页面 3 的一个链接是从页面 2 链入的，而反过来，页面 4 没有链入页面 3，页面 3 没有链入页面 2，这种单方面的链接关系必然使页面 4 具有更高的排名，其次为页面 3，最后为页面 2。因为页面 1 和页面 5 只有一个链接，且链入页面 1 的页面 4 的重要性比链入页面 5 的页面 3 的重要性要高，所以页面 1 的重要性高于页面 5。相对于传统 PageRank 算法，改进算法的排名比较符合上述分析。

在理论分析时，我们认为页面 3 的重要性应高于页面 1，因为页面 3 有 2 个入链，而页面 1 只有 1 个入链。但在改进的 PageRank 算法中，页面 1 的排名却高于页面 3。只有一种原因可以合理的解释这种情况：即虽然页面 3 有 2 个入链，但这两个入链均是普通页面（*PR* 值较低），而页面 1 虽然只有 1 个入链，但这个入链却是重要页面（*PR* 值较高）。实际上也正是这种原因造成了上述结果，链入页面 1 的是图中 *PR* 值最高的页面 4，而链入页面 3 的是图中 *PR* 值较低的页面 2 和页面 5。

在图 3-1 中，也存在上述情况，页面 5 和页面 6 指向页面 C，页面 1-5 指向页面 A。在分配 *PR* 值时，页面 A 得到了 $\frac{5}{7}PR(5)$，页面 C 得到了 $\frac{2}{7}PR(5)$，即页面 A 分得的 *PR* 值

是页面 C 的 2.5 倍，显然，从新算法的角度（仅用入链数来衡量页面的重要性），我们认为页面 A 的重要性要高于页面 C。但是，如果页面 6 是一个极其重要的页面，而页面 1-5 只是一些普通页面，那么如何衡量两个页面的重要性呢？即究竟是被多个普通页面指向的页面 A 重要，还是仅被一个重要页面指向的页面 C 重要？

这里，由于上述两个例子都只是包含个位数页面的小网络，恰巧遇到了这种情况，而在实际中，由于互联网包含数以亿计的页面，这种情况是极其罕见的，即很少存在一个重要性页面指向一个入链数很少（重要性较低）的普通页面，这种情况也是与人们思想相背离的。

最后，我们发现改进算法明显提高了排名第一和第二页面的 *PR* 值，降低了排名第四和第五页面的 *PR* 值，即提升了重要页面的 *PR* 值，降低了普通页面的 *PR* 值，确实较好地区分了页面的重要程度。

3.5 本章小结

本章是本文的核心，在分析了传统 PageRank 算法和多种经典改进算法的基础上，提出了一种新的 PageRank 排序算法。新算法改进了传统算法在分配 *PR* 值时没有针对性，单纯将自己权值平均分配给各个页面的思路，而采用根据各个页面重要性不同去分配 *PR* 值的方法，并通过一个简单的试验网络从理论上说明了改进算法的优越性。

4 Nutch 相关技术

4.1 Nutch 简介

Nutch 是 Apache 软件基金会 Jakarta 项目组的一个子项目，它是以 Lucene 为基础，使用 Java 语言开发的一种开放源代码（Open-Source）的 Web 搜索引擎[48]。Nutch 提供了我们运行自己搜索引擎所需的全部工具，包括全文搜索和 Web 爬虫。

之所以选取 Nutch 作为测试平台主要是因为其具有以下几个特点[49]：

（1）透明度，Nutch 是开放源代码的，任何人都可以在遵守开源准则的条件下下载到源代码，查看其具体各个部分是如何工作的；

（2）扩展性，Nutch 是非常灵活的，我们可以将各种不同的算法、分词技术及合理的插件导入 Nutch，根据实际应用需要对 Nutch 的功能进行扩展；

（3）稳定性，Nutch 具有较高的运行稳定性。

4.2 Nutch 工作原理

Nutch 作为完整的搜索引擎，其工作流程和基本组成与其它搜索引擎基本一致，总体上可以分为爬虫抓取（Crawler）和检索查询（Searcher）两个部分[50]。爬虫抓取部分主要是负责从网络上抓取页面并为抓取回来的页面建立索引;检所查询部分主要是利用索引检索用户的查询请求来产生检索结果。各部分工作流程图如图 4-1 所示。

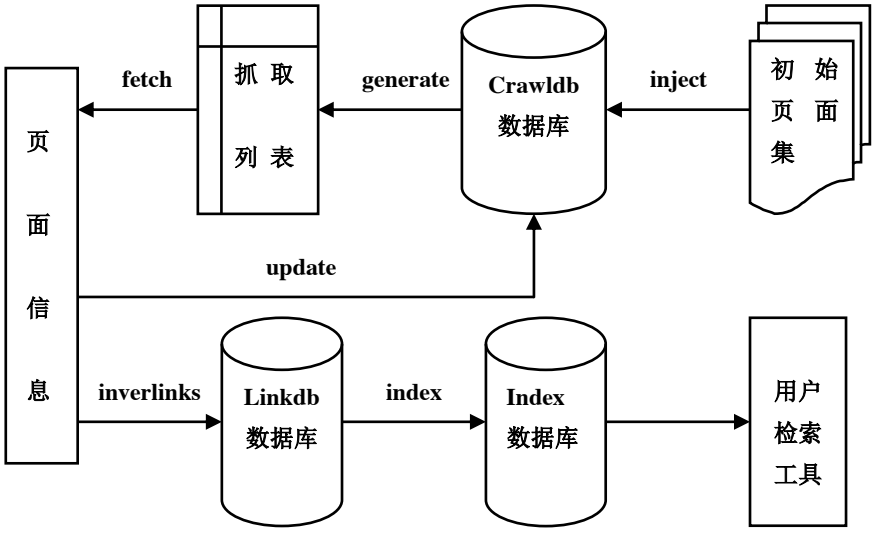

图 4-1 Nutch 工作流程图（取自文献【51】）

Fig.4-1 Working flow chart of Nutch(taken from literature[51])

下面，我们主要分析爬虫 Crawler 的工作流程和相应的数据文件。

（1）Crawler 的工作流程

Nutch 爬虫抓取系统的主要任务就是采集互联网上的 Web 数据并为其建立索引，由于 Nutch 自身架构设计的特点及其采用的特殊实现机制，我们可以清晰地分离出每个阶段所要执行的任务[52]，工作流程如图 4-2 所示。

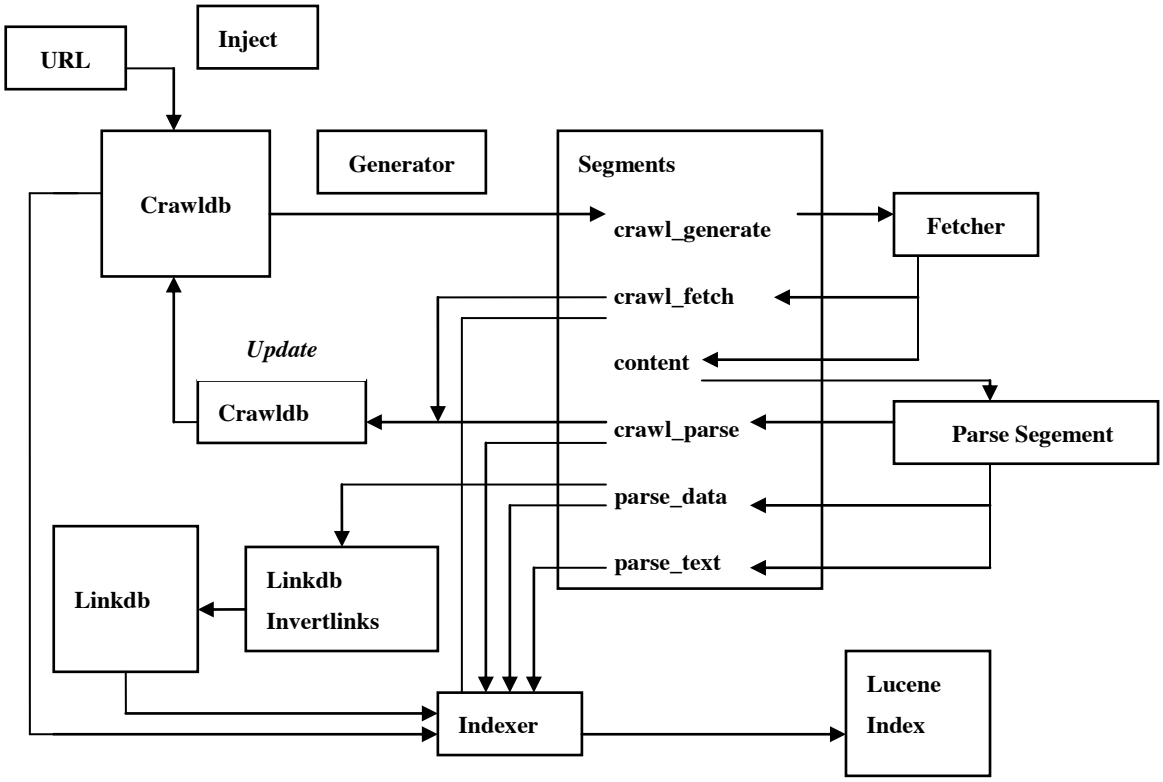

图 4-2 抓取过程流程图（取自文献【51】）

Fig.4-2 Flow chart of fetching process(taken from literature[51])

假设预先设定抓取深度为 3，则抓取过程可表示为：

```
Injector>
    Generator > Fetcher > Parse Segment > Crawldb Update    depth=1
    Generator > Fetcher > Parse Segment > Crawldb Update    depth=2
    Generator > Fetcher > Parse Segment > Crawldb Update > Linkdb    depth=3
```

可以看出，抓取过程的核心是根据抓取深度重复 Generator > Fetcher > Parse Segment > Crawldb Update 这个循环。总结如下：

1) 建立一个初始 URL 数据集；
2) 执行 Inject 操作将初始 URL 数据集导入 Crawldb 数据库；
3) 执行 Generate 操作创建页面抓取列表；
4) 执行 Fetch 操作对页面进行抓取，获取页面信息；
5) 执行 Parse 操作解析 Segments 中由 Fetch 操作抓取到的页面数据；
6) 执行 Updatedb 操作更新原始 URL 数据库，把抓取到的相关页面信息导入 Crawldb 数据库中，并为下一次启动抓取任务做准备；
7) 重复进行步骤 3-5，直到达到预先设定的抓取深度；
8) 执行 Inverlinks 操作并根据 Segments 的内容更新 Linkdb 数据库；
9) 执行 Index 命令对抓取的页面建立索引。

（2）Crawler 的数据文件

Nutch 爬虫抓取结束后，抓取到的结果将保存在预先设定的爬行目录文件夹内（此处设文件名为 crawl），在爬行结果目录中含有 crawldb，segments，linkdb，indexes，index 共 5 个文件夹，如图 4-3 所示，其含义依次如下：

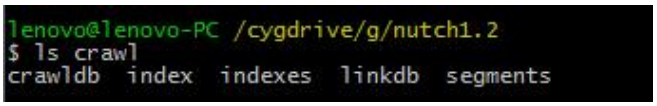

图 4-3 爬行结果目录

Fig.4-3 Crawl results directory

Crawldb：爬行数据库，即 URL 库，用来存储所要爬行的 URL 地址。保存了页面的抓取状态、抓取时间、修改时间、有效期、分值等详细关于抓取的内容。

Linkdb：链接数据库，保存的是页面的链接情况。用来存储每个 URL 的自身地址和源链接。

Segments：存放的是爬行抓取的页面，Segments 文件包含 *n* 个 Segment 子文件（*n* 等于预先设定的抓取深度），一个 Segment 子文件的名称是对应的抓取时间，里面对应存储的是爬虫进行一组 Generate/Fetch/Update 循环时抓取到的页面以及这些页面的索引。一个 Segment 包含多个子目录，如图 4-4 所示，其含义依次如下：

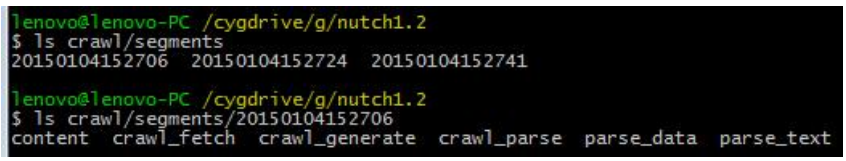

图 4-4 Segment 子目录

Fig.4-4 Subdirectory of Segment

content：包含了从每个页面获取回来的原始内容；

crawl_generate：包含了一组将要被抓取页面的基本信息；

crawl_fetch：包含了每个被抓取页面的状态（是否被成功抓取等）；

parse_text：包含了每个页面被解析之后的文本内容；

parse_data：包含了每个页面被解析之后的基本数据（页面的标题，作者，日期等）和输出链接；

crawl_parse：包含了所有页面的信息，用于更新 Crawldb 数据库。

Indexes：存放每次下载的独立索引目录。

Index：符合 Lucene 格式的索引目录，存储的是 Crawler 抓取的所有页面的索引，是对 Indexes 里所有索引进行合并得到的。

我们可以通过 Nutch 自带的命令，对上述文件内容逐一进行查看，有关索引的内容我们可以结合 Luke 工具进行查看。

4.3 Nutch 排序机制

Nutch 对页面的综合排序算法主要由两部分组成，其一是基于 Lucene 索引检索机制

的关键字匹配算法，主要解决排序过程中内容相关性的问题【53】；其二是页面的重要性权重，通过分析页面之间的超链接得到，类似于 PageRank 算法，在 Nutch 中把这个重要性权重量化为 Boost 值。

4.3.1 Nutch 中基于 Lucene 的页面排序算法

Lucene 排序技术使用了信息检索中的向量空间模型方法【54】，采用基于关键字匹配的方式，通过在计算过程中使用 $tf\text{-}idf$ 策略来计算不同关键词在页面中的权重，并以此权重作为排序依据【55】。$tf\text{-}idf$ 关键词匹配方式示意图如图 4-5 所示，具体公式如式(4.1)所示。

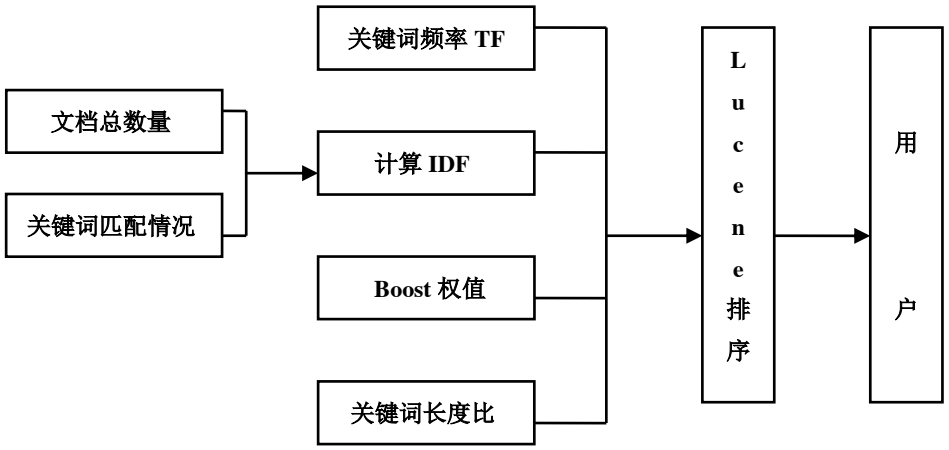

图 4-5 $tf\text{-}idf$ 关键词匹配示意图（取自文献【56】）

Fig.4-5 Keywords matching diagram of tf-idf(taken from literature[56])

$$Score(q,d) = coord(q,d)\cdot queryNorm(q)\cdot \sum_{t\in d}(tf(t\ in\ d)\cdot idf(t)^2\cdot t.getBoost()\cdot norm(t,d)) \tag{4.1}$$

其中，q 表示检索条件，d 表示检索页面，t 表示检索关键词。为了更好的理解公式，下面依次解释公式中各个参数的含义以及是如何计算的。

（1）$Score(q,d)$，表示检索页面 d 相对检索条件 q 的得分。

（2）$tf(t\ in\ d)$，词频率（Term Frequency），表示关键词 t 在页面 d 中出现的次数，出现次数越多，页面的检索得分就越高。计算公式如下：

$$tf = \sqrt{termFreq} \tag{4.2}$$

（3）$idf(t)$，逆文档频率（Inverse Document Frequency），表示查询匹配页面数量的反向函数。可理解为关键词 t 对不同页面的区分能力，这个值越大说明该关键词的作用越大，有利于得到好的排序结果。计算公式如下：

$$idf = 1.0 + \log(\frac{numDocs}{docFreq+1}) \tag{4.3}$$

其中，$numDocs$ 表示索引中的页面总数，$docFreq$ 表示索引中包含关键词 t 的页面总

数。按照信息理论，*docFreq* 出现的次数越少，这类页面的信息量就会越大。所以匹配的页面越少，得分就越高。而索引库中页面总数越多，找到目标页面的难度越困难，相应的信息量也会比较大。但在同一个检索条件下，这些值是相同的。

（4）$coord(q,d)$，这是一个评分因子，表示页面中关键词出现的比例，越多的关键词出现在一个页面中，说明这些页面的匹配程度越高。计算方法是 q 中不同关键词在 d 中出现的次数之和与页面 d 中所有关键词之和的比值。计算公式如下：

$$coord(q,d)=\frac{\sum_{t\in q} tf(t\ in\ d)}{\sum_{t\in d} tf(t\ in\ d)} \tag{4.4}$$

（5）$t.getBoost()$，关键词 t 在页面 d 中的重要性权值（Boost），用来控制页面的重要性。它由两部分组成，页面权值（Document Boost）和页面域权值（Field Boost），页面权值是建立索引时设置的页面总体得分，页面域权值是一个页面域（例如 Url，Title，Content 等）添加到页面时设定的域得分[57]。

（6）$queryNorm(q)$，这个函数是一个归一化因子，在搜索时计算并仅在搜索时起作用，不会影响最后的排序结果。在公式中引入这个因子的作用仅仅是为了使不同的排序结果在不同的检索条件之间具有可比性。计算公式如下：

$$queryNorm(q)=\frac{1}{\sqrt{q.getBoost()^2\cdot\sum_{t\ in\ q}(idf(t)\cdot t.getBoost())^2}} \tag{4.5}$$

其中，$q.getBoost()$ 表示的是一个查询子句被赋予的 Boost 值。

（7）$norm(t,d)$，表示在计算中一些相关的优先权量化数值，是在索引时进行计算并存储的。计算公式如下：

$$norm(t,d)=d.getBoost()\cdot lengthNorm(field)\cdot\prod_{field\ f\ in\ d} f.getBoost() \tag{4.6}$$

其中，$d.getBoost()$ 表示整个页面的 Boost 值，$f.getBoost()$ 表示页面中某个域的 Boost 值，$lengthNorm(field)$ 表示长度因子，一个域中包含的关键词总数越多，即页面越长，此值越小；页面越短，此值越大。计算公式如下：

$$lengthNorm(f)=\frac{1}{\sqrt{num\ of\ terms\ in\ field\ f}} \tag{4.7}$$

注意，上面所有的 $.getBoost()$ 值均可以在索引的时候通过 $setBoost(boost)$ 方法设置。

下面通过一个具体例子来了解上述公式在实际中是如何实现的，图 4-6 是在 Nutch 检索结果评分详解中截取的一部分内容。

page

- title = 爱范品的频道-搜狐视频自媒体
- segment = 20150108165408
- boost = 0.21721295
- digest = 49fb1d6530b4ebaa1f4b12185d184cf8
- tstamp = 20150108085451240
- url = http://my.tv.sohu.com/user/232799889

score for query: 爱

- 0.060514007 = (MATCH) sum of:
  - 9.105228E-4 = (MATCH) weight(content:爱 in 193), product of:
    - 0.049023338 = queryWeight(content:爱), product of:
      - 1.9212102 = idf(docFreq=364, maxDocs=917)
      - 0.025516905 = queryNorm
    - 0.018573252 = (MATCH) fieldWeight(content:爱 in 193), product of:
      - 1.4142135 = tf(termFreq(content:爱)=2)
      - 1.9212102 = idf(docFreq=364, maxDocs=917)
      - 0.0068359375 = fieldNorm(field=content, doc=193)
  - 0.059603482 = (MATCH) weight(title:爱^1.5 in 193), product of:
    - 0.20424499 = queryWeight(title:爱^1.5), product of:
      - 1.5 = boost
      - 5.3362007 = idf(docFreq=11, maxDocs=917)
      - 0.025516905 = queryNorm
    - 0.29182348 = (MATCH) fieldWeight(title:爱 in 193), product of:
      - 1.0 = tf(termFreq(title:爱)=1)
      - 5.3362007 = idf(docFreq=11, maxDocs=917)
      - 0.0546875 = fieldNorm(field=title, doc=193)

图 4-6 Nutch 评分详解

Fig.4-6 Page score of Nutch

在图 4-6 中，包含了页面的基本信息（title，segment，boost 等）和页面得分（Weight）的具体计算过程。可以看出，在检索条件为“爱”时，其得分等于该检索条件在 Content 域和 Title 域的得分之和。而在每个域中，得分又等于查询权重（queryWeight）和页面域权重（fieldWeight）之积。

综上所述，一个检索条件对于一个特定页面的得分，是该检索条件在页面不同域（Url，Title，Content）中的得分之和。对于每一个域，得分主要是下列因素的乘积：*tf*，*idf*，*boost*，*lengthNorm*，*queryNorm*。

4.3.2 Nutch 综合排序算法

为了验证改进后的 PageRank 算法相对传统 PageRank 算法确实拥有更好的检索效果，我们需要在 Nutch 中分别实现 PageRank 原型算法和改进算法。

Lucene 中的 Boost 值是指在建立索引的过程中，给整个页面（Document）或者页面某一特定域（Title，Content，Url 等）设定的权值因子。由公式(4.1)可以看出，在 Nutch 进行页面排序时，会获取每个索引页面中的多个域值参与计算，其中就包括了整个页面的权值。又因为本文提出的 PageRank 改进算法和传统 PageRank 算法只考虑了页面之间的链接结构，并没有考虑页面之间的内容相似相关性。所以，在 Nutch 中实现结合 PageRank 的综合排序算法时，只需要改进整个页面的 Boost 值即可。

本实验中，我们将 *PR* 值作为整个页面的权值因子导入 Nutch 页面评分公式中，即用 *PR* 值来代替原来整个页面 Boost 值，这就是结合了 PageRank 算法的 Nutch 排序综合算法。

结合图 4-6 中的实例，改进后的评分公式如下

$$\begin{cases} Score(q,d) = \sum_{t\ in\ q} \sum_{t\ in\ field} queryWeight(t) \cdot fieldWeigh\,t(t) \\ queryWeight(t) = getBoost(t\ in\ field) \cdot idf(t) \cdot queryNorm(q) \\ fieldWeigh\,t(t) = tf(t) \cdot idf(t) \cdot PR(d) \cdot lengthNorm(field) \end{cases} \tag{4.8}$$

其中，*queryWeigh t* 是查询权重，*fieldWeigh t* 页面域权重，*getBoos t*(*t in field*)表示检索关键词 *t* 出现在页面某个域中的权值，例如页面标题域（Title）权值应比正文域（Content）权值大等，其余参数含义与公式(4.1)中相同。

4.4 本章小结

本章主要介绍了 Nutch 开源搜索引擎的相关知识，对其基本工作流程和排序机制进行了详细剖析。通过融入 PageRank 计算因子对 Nutch 评分公式进行改进，即用 *PR* 值来代替原来整个页面的 Boost 值，以此得到了结合 PageRank 算法的 Nutch 排序综合算法。通过对本章的分析和研究，目的是为后面基于 Nutch 平台验证改进算法与传统 PageRank 算法的优劣性提供技术支持。

5 基于 Nutch 搜索引擎实验平台的测试

本章将搭建一个基于 Nutch 的搜索引擎，通过在这个搜索引擎中应用所提出的改进算法来验证改进后的 PageRank 算法相对传统 PageRank 算法确实拥有更好的检索效果。

5.1 Nutch 实验平台搭建

在开发平台上，我们选择 Nutch1.2 作为搜索引擎基础构建平台。

5.1.1 软件环境

安装环境：Windows 7

安装软件：Nutch1.2

Cygwin

Jdk1.7

Apache Tomcat 6.0.39

在设计上，我们需要借助 Cygwin，Apache Tomcats 6.0.39，Jdk1.7，Luke，Notepad++，Ant 和 Eclipse4.3.2 等多种软件。

Nutch1.2 能够正常在 Windows 操作系统上运行，需要安装 Cygwin 软件，此软件可以在 Windows 操作系统下模拟 Linux 操作系统环境。由于 Nutch 是纯 Java 编写的开源代码，为了正常运行和二次开发 Nutch，必须安装 Java 开发工具包 Jdk，本实验采用 Jdk1.7 版本。Nutch 搜索引擎的检索接口是通过页面发布的，必须要提供发布页面的服务，本实验采用 Tomcat 6.0.39 来提供信息发布服务。

5.1.2 搭建步骤

第一步：安装。

首先安装 Nutch，安装路径：G:\nutch1.2。接着安装 Cygwin，安装路径：G:\Cygwin。随后安装 JDK，安装路径：G:\JDK。最后安装 Tomcat，安装路径：G:_apache-tomcat-6.0.39。

第二步：环境变量配置。

添加 NUTCH_JAVA_HOME，JAVA_HOME，设置为

```
G:\JDK
```

添加 CATALINA_BASE,CATALINA_HOME，设置为

```
G:\apache-tomcat-6.0.39
```

修改 PATH 环境变量，加入

```
G:\cygwin\bin;%JAVA_HOME%\bin;%JAVA_HOME%\gre\bin;%CATALINA_HOME%\bin”
```

添加 CLASSPATH 环境变量，设置为

```
.;%JAVA_HOME%\lib\dt.jar;%JAVA_HOME%\lib\tools.jar;%CATALINA_HOME%\lib\servlet-api.jar
```

第三步：配置 Nutch。

第四步：爬虫部署，抓取页面并建立索引。

第五步：配置 Tomcat，进行检索。

5.2 实验数据抓取

首先在 G:/nutch1.2 目录下新建一个文件夹，命名为 urls，在此文件夹下，新建一个文本文件，命名为 urls.txt，其内容是要抓取的起始网站地址，本实验中，为了保证实验数据的综合性和广泛性，选取 3 个综合型网站作为入口页面地址，分别是：

```
http://www.sina.com.cn/
http://www.sohu.com/
http://www.xinhuanet.com/
```

接下来修改 G:/nutch1.2/conf 目录下的 crawl-urlfilter.txt 文件，将其中的

```
# accept hosts in MY.DOMAIN.NAME
+^http://([a-z0-9]*\.)*MY.DOMAIN.NAME/
# skip everything else
```

修改为

```
# accept hosts in MY.DOMAIN.NAME
+^http://([a-z0-9]*\.)*sina.com.cn/
+^http://([a-z0-9]*\.)*sohu.com/
+^http://([a-z0-9]*\.)*xinhuanet.com/
# skip everything else
```

最后，打开 G:/nutch1.2/conf 目录下的 nutch-site.xml 文件（本实验中选择使用 Notepad++打开.xml 格式的文件），在<configuration></configuration>间添加如下相应的配置内容：

```
<configuration>
<property>
<name>http.agent.name</name>
<value>shiyan</value>
</property>
</configuration>
```

上述配置完成后，打开 Cygwin 窗口，输入如下命令：

```
bin/nutch crawl urls –dir crawl –depth 8 –topN 200
```

- urls，设定起始页面存放的文件夹。
- -dir crawl，设定抓取结果数据存放的文件名为 crawl。
- -depth 8，设定需要抓取的页面深度为 8 层。
- -topN 200，设定爬虫在每层深度上只抓取前 N 个 url，这里指定每一层前 200 个。

命令执行后会调用一系列底层命令，在 Cygwin 窗口输出一系列输出信息，如图 5-1 所示。输出内容包括执行中使用的各种参数以及存储相关的数据段信息。

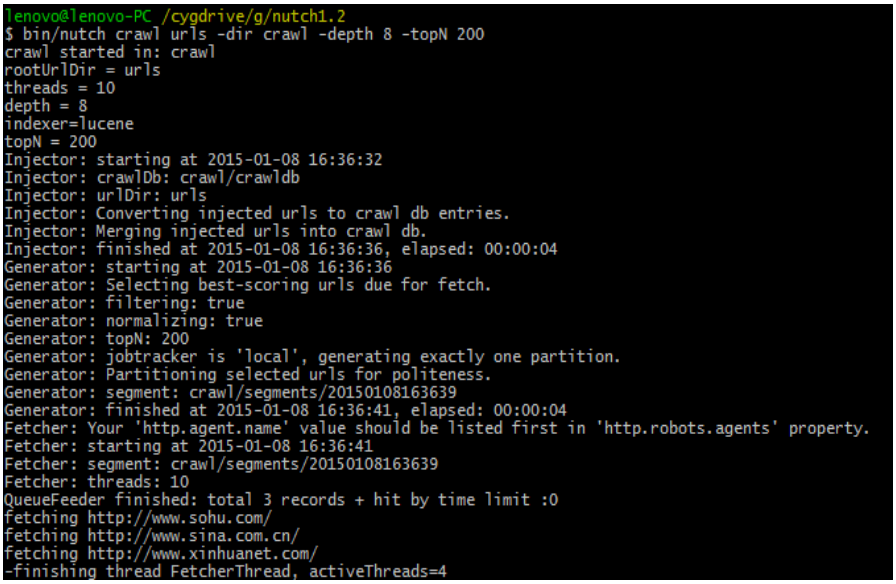

图 5-1 Cygwin 窗口信息片段

Fig.5-1 Part information of Cygwin window

抓取完成后，我们通过执行一个简单的命令来查看页面抓取的大概信息，命令如下：

```
bin/nutch readdb crawl/crawldb -stats
```

得到页面抓取情况如图 5-2。

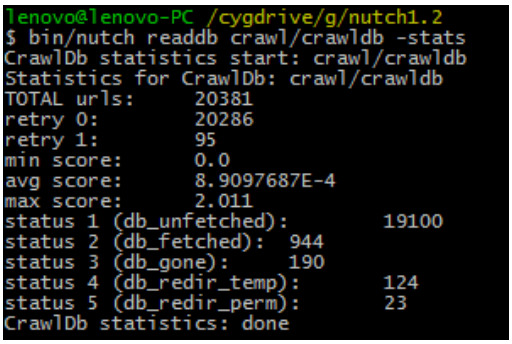

图 5-2 抓取页面信息

Fig.5-2 Information of fetching web

当页面抓取完成后，配置其检索接口 Tomcat。首先复制 G:/nutch1.2 目录下的 nutch-1.2.war 文件到 G:/apache-tomcat-6.0.39 目录下的 webapps 文件夹，启动 Tomcat 的同时，该解压文件会在该目录下自动生成一个名为 nutch-1.2 的文件夹。修改 G:/ apache-tomcat-6.0.39 /webapps /nutch-1.2/WEB-INF/classes 目录下的 nutch-site. xml 文件，在<configuration> </configuration>间添加如下相应的配置内容：

```
<configuration>
<property>
<name>searcher.dir</name>
<value>G:/nutch1.2/crawl</value>
</property>
</configuration>
```

此处，Value 值是 nutch 文件中保存爬取页面信息的文件地址。注意每次修改 nutch-site.xml 文件，都要重新启动 Tomcat。当出现如图 5-3 的界面，Tomcat 便配置成功。

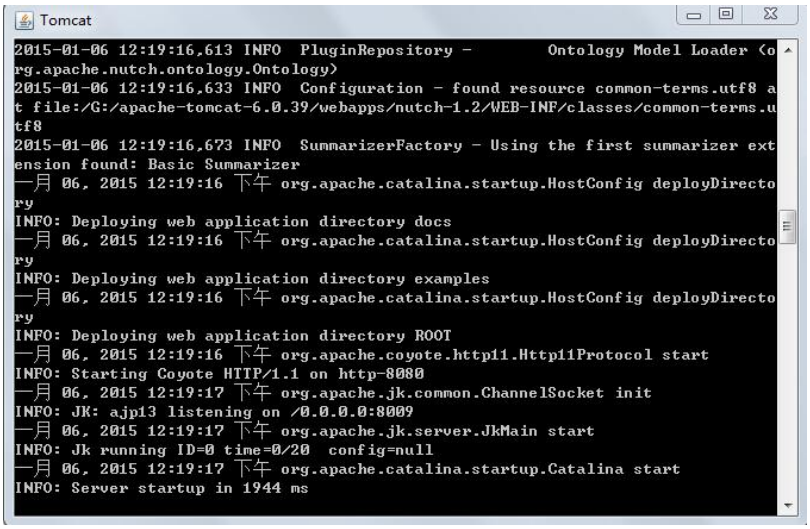

图 5-3 Tomcat 配置成功界面

Fig.5-3 Successful configuration interface of Tomcat

为了解决中文乱码问题。修改 G:/ apache-tomcat-6.0.39/conf 目录下的 server.xml 文件，

将其中的

```
<Connector port="8080" protocol="HTTP/1.1"
               connectionTimeout="20000"
               redirectPort="8443" />
```

修改为

```
<Connector port="8080" protocol="HTTP/1.1"
               connectionTimeout="20000"
               redirectPort="8443"
               URIEncoding="UTF-8"
               useBodyEncodingForURI="true"/>
```

当上述配置全部完成之后，在浏览器中输入 http://localhost:8080/nutch-1.2/，就会得到检索界面，提交检索条件，就会看到排序好的检索结果，检索条件为“体育”的检索结果如图 5-4 所示。

图 5-4 Nutch 用户检索界面

Fig.5-4 User retrieval interface of Nutch

5.3 实验过程及结果分析

本实验通过已建立搜索引擎爬取的 20381 个页面，选取了 15 位来自不同知识领域的测试人员和 10 个比较受关注的检索条件在本地机器上进行对比实验来验证改进算法的优越性，检索条件如表 5-1 所示。

表 5-1 检索条件列表

Table5-1 Retrieval condition list

序号	检索条件	序号	检索条件
1	房价	6	公务员考试
2	医疗	7	食品安全
3	环境保护	8	油价变动
4	大学生就业	9	社会稳定
5	反腐倡廉	10	教育改革

对于一个检索条件，搜索引擎通常会给出海量的检索结果，在这些大量的检索结果中，只有很少一部分才是用户真正需要的。研究发现[58]，用户一般只对搜索引擎检索结果的前 2 页表达出浓厚的兴趣，特别是对排名前 20 位的检索结果最感兴趣。本实验中，为了更好地检验改进算法的优越性，我们选取排名前 30 的检索结果进行测试。

本实验中，我们首先采用搜索结果的查准率来对比两种算法的优劣性，因为我们关注的是排名前 30 的检索结果，因此，查准率计算公式如下：

$$查准率=\frac{前30位与查询主题相关的搜索结果的个数}{30(不足30的取搜索结果总数)} \tag{5.1}$$

针对 10 个检索条件的查询，传统 PageRank 算法与改进 PageRank 算法的检索排序效果如表 5-2 所示。

表 5-2 传统 PageRank 算法与改进 PageRank 算法查准率比较

Table5-2 Comparison of precision ratio between traditional and improved algorithm

检索条件	传统 PageRank 算法查准率%	改进 PageRank 算法查准率%	查准提升率%
房价	80.0	86.7	8.3
医疗	80.0	80.0	0
环境保护	53.3	60.0	12.6
大学生就业	60.0	66.7	11.2
反腐倡廉	50.0	56.7	13.4
公务员考试	56.7	63.3	11.6
食品安全	56.7	73.3	29.3
油价变动	60.0	63.3	5.5
亚洲杯	60.0	66.7	11.2
教育改革	53.3	53.3	0

从表 5-2 可知，相对于传统 PageRank 算法，改进算法明显提高了检索结果的查准率，在 10 个相同主题的检索条件下，除“医疗”和“教育改革”持平外，改进算法比传统算法的查准率分别提高了 8.3%， 12.6%，11.2%，13.4%，11.6%，29.3%，5.5%，11.2%，

平均提高了 10.31%。

为了更好的验证改进算法的优越性，除查准率这个评价标准外，我们还可以借鉴类似于文献【47】,【59】中提出的相关度方法对改进算法进行评价，该方法的基本思想是对于检索结果中的页面，根据用户对页面与检索条件的相关程度分为如下 3 个档次：

$$K=\begin{cases}2 & \text{页面与检索条件密切相关}\\ 1 & \text{页面与检索条件部分相关}\\ 0 & \text{页面与检索条件毫不相关}\end{cases} \tag{5.2}$$

让 15 名参与实验的测试人员对检索结果中的页面进行相关度评价，为保证评价结果的公正性，只有当大多数人认为某页面是某一结果时，才将该页面归为这种结果。如果出现无法断定的情况，则重新评估该页面。对于某个检索条件，由于最终结果的评定是按照检索结果中前 30 个页面相关性之和来确定的，因此，除了页面是否与检索条件相关之外，我们还应该考虑页面在检索结果中出现的位置，综合上述思想，我们可以得到相关度函数的计算公式如下：

$$R=\sum_{i=1}^{30}K_i\times(30-i+1)/30 \tag{5.3}$$

其中，R 为检索结果与检索条件的相关度，K_i 为检索结果中某页面与检索条件的相关度，由 15 名参与实验的人员根据式(5-2)得到，i 为该页面在检索结果中的排名。针对 10 个检索条件的查询，传统 PageRank 算法与改进 PageRank 算法的检索排序效果如表 5-3 所示。

表 5-3 传统 PageRank 算法与改进 PageRank 算法相关度比较

Table5-3 Comparison of relevancy between traditional and improved algorithm

检索条件	传统 PageRank 算法相关度	改进 PageRank 算法相关度	相关度提升率%
房价	8.95	9.2	2.8
医疗	8.34	8.49	1.8
环境保护	7.26	7.56	4.1
大学生就业	7.32	7.36	0.5
反腐倡廉	6.86	7.12	3.8
公务员考试	7.02	7.48	6.6
食品安全	7.68	8.42	9.6
油价变动	8.32	8.46	1.7
亚洲杯	7.65	8.15	6.5
教育改革	7.14	7.69	7.7

从表 5-3 可知，改进 PageRank 算法的排序效果略好于传统 PageRank 算法，在 10 个相同主题的查询条件下，改进算法比传统算法的相关度分别提高了 2.8%，1.8%，4.1%，0.5%，3.8%，6.6%，9.6%，1.7%，6.5%，7.7%，平均提高了 4.5%。

5.4 本章小结

本章是论文的实验部分，首先阐述了实验的软件环境和搭建步骤，并给出了实际数据抓取的详细配置和方法。最后针对实验数据，做了具体实验，通过两种不同的评价标准对结果进行了详细分析。实验结果表明，不论从查准率角度还是相关度角度，改进算法均有一定的优越性，即改进后的 PageRank 算法确实提高了页面的排序质量，提升了检索结果的准确性，说明改进算法的正确性。

6 总结与展望

6.1 总结

搜索引擎技术自互联网问世以来一直备受关注，由于网络互联复杂性，信息庞大性，以及资源多样性的特征，原有的排序技术已经不能满足日益多元化的信息索引需求。本文从经典的 PageRank 算法入手，通过对该算法进行深入研究，在传统 PageRank 算法的基础上从 *PR* 值分配的角度提出了一种相应的改进算法，实验结果表明，改进算法更好的揭示了互联网的特征，提高了用户的查询体验。

本文的主要工作：

（1）对传统 PageRank 算法的分析和评价。

从基本概念出发，详细介绍了传统 PageRank 算法的体系结构、形成思路和计算方法，并在此基础上，介绍了多种针对传统 PageRank 算法的加速算法。同时，通过对传统 PageRank 算法进行深入剖析，分析其优缺点，并针对其不足，主要从主题漂移和偏重旧网页两个方面介绍了多种国内外现存的经典改进算法，为进一步优化现有算法提供了可能。

（2）针对传统 PageRank 算法，提出了一种基于 PageRank 算法的改进算法。

针对传统 PageRank 算法在分配 *PR* 值采用平均分配策略这一缺陷，提出了一种优化改进算法，即改进传统算法在分配 *PR* 值时没有针对性，单纯将自己权值平均分配给各个页面的思路，而提出一种采用根据各个页面重要性不同去分配 *PR* 值的方法，并通过一个简单的实验，从理论上验证了改进算法的可行性，为后面实际网络中的实验提供了理论依据。

（3）架构实验平台，验证改进算法。

构建基于 Nutch 1.2 的搜索引擎，首先对其基本工作流程和排序机制进行了详细剖析。接着通过融入 PageRank 计算因子对 Nutch 评分公式进行改进，即用 *PR* 值来代替原来整个页面的 Boost 值，以此得到了结合 PageRank 算法的 Nutch 排序综合算法。最后将改进后的 PageRank 算法与传统 PageRank 算法进行比较，实验证明，改进算法确实提高了页面的排序质量，提升了搜索结果的准确性，更符合用户需求。

6.2 展望

PageRank 算法作为最成功的一种链接分析方法，长久以来，一直受到国内外学者，研究机构的广泛关注，本文提出的改进算法虽然在检索结果的排序质量上有了一些提高，但依旧存在一些不足，主要包括以下几个方面：

（1）本文仅仅针对传统 PageRank 算法在分配 *PR* 值采用平均分配策略这一缺陷，提出了有针对性的改进算法。对于其他方面的不足，如偏重旧页面，忽略用户的兴趣要求等，本文暂时没有进行深入研究，在今后的工作中可以逐步展开。

（2）Google 在应用 PageRank 算法时，采用的是离线计算，在线查询的方式，因此所有页面 *PR* 值都是预先，离线处理好的。尽管如此，这样的计算量也是很庞大的。在本文的改进算法中，因为要考虑每个页面的入链总数，所以改进算法的计算复杂度稍大于传统 PageRank 算法。如何在保证排序质量的前提下减少时间复杂度这还有待以后的研究。

（3）本实验结果与实验所选的检索条件，实验测试人员，评价标准均密切相关，而本文所选的测试人员较为单一，检索条件也有一定的局限性，而且仅从查准率和相关度两个角度来验证，没有考虑更多的因素。下一步我们需要在不同背景的人群中收集更多的数据，以便进行更全面的实验验证。

总之，经过本文改进以后的 PageRank 算法较传统算法而言，确实提高了页面排序的准确度，优化了排序质量，具有现实可行性。当然，也存在一些不足，后面仍需进一步的研究和改进。望各位老师批评指正。

致谢

在论文即将完成之际，谨此向我的导师王林教授致以衷心的感谢和崇高的敬意！在接近 3 年的硕士研究生期间，从论文的选题、研究工作的开展到论文的最后完成，王老师均给予了我悉心细致的教诲和无私的帮助。王老师广博的专业知识、深厚的学术素养、严谨的治学精神和奉献的工作态度使我终生受益，这些教诲必将激励我在今后的人生道路上奋勇向前。

同时，我还要感谢自动化与信息工程学院的各位老师，感谢他们多年来给我学习生活上的教导和关心。感谢教研室的各位同学，三年来的共同学习和帮助，让大家在各方面都有了很大的提高，特别是在论文的写作过程中，得到了他们宝贵的建议和帮助，在此一并致以诚挚的谢意。

我还要感谢我的家人和朋友多年来对我的关爱、理解和支持。正是他们的寄托和支持激励我不断地拼搏进取，在科研的道路上坚定前行。

最后，衷心感谢在百忙中抽出时间对本文进行评审并提出宝贵意见的各位专家和学者！

由于作者水平有限，文中如有疏漏，还恳请各位老师同学批评指正。

参考文献

【1】 CNNIC 第 35 次互联网报告[R],中国互联网信息中心. 2015.

【2】 Page L, Brin S, et al. The PageRank Citation Ranking: Bringing Order to the Web [C]. Computer Science Department, Stanford University.1998.

【3】 Haveliwala T H. Topic-Sensitive PageRank[C].Proceedings of the Eleventh International World Wide Web Conference.Hoholulu Hawaii.2002.

【4】 李稚楹,杨武,谢治军.PageRank 算法研究综述[J].计算机科学.2011.38(10A):185-188.

【5】 Richardson M, Domingos P. The Intelligent Surfer: Probabilistic Combination of Link and Content Information[J]. Advances in Neural Information Processing Systems.2002,14:673-680.1441-1448.

【6】 Kleinberg J.Authoritative Sources in a Hyperlinked Environment[C].9th Annual ACM-SIAM Symposium on Discrete Algorithms,1998:668-677.

【7】 Yamamoto A, Asahara D, Tanaka S, et al. Distributed PageRank：A Distributed Reputation Model for Open Peer-to-Peer Networks [C].International Symposium,2004.

【8】 Manaskasemsak, B. Rungsawang, A. Parallel PageRank computation on a gigabit PC cluster Advanced Information Networking and Applications, AINA,18th International Conference,2004.

【9】 Kamvar S D, Haveliwala T H, Golub G H. Adaptive Methods for the Computation of PageRank[R]. Technical Report, Stanford University.2003.

【10】 Chen Yenyu,Gan Qingqing,Suel T.IO-efficient Techniques for Computing PageRank[R]. New York City College of Technology (Brooklyn).2002.

【11】 Kamvar S D, Haveliwala T H, et al.Extrapolation Methods for Accelerating PageRank Computations[C].Proceedings of the Twelveth International World Wide Web Conference. Budapest, 2003.

【12】 Kamvar S D,Haveliwala T H,et al.Exploiting the Block Structure of the Web for Computing PageRank[C]. Proceedings of the Twelveth International World Wide Web Conference.Budapest 2003.

【13】 黄德才,戚华春. PageRank 算法研究[J].计算机工程.2006,32(4):145-147.

【14】 高琪,张永平.PageRank 算法中主题漂移的研究[J].网络与通信.2010,26(3):117-119.

【15】 白似雪,刘华斌.基于页面分块模型的 PageRank 算法研究[J].南昌大学学报.2008,30(2):179-183.

【16】 李绍华,高文宇.基于层次分类的页面排序算法[J].计算机工程.2007,33(11):56-58.

【17】 张岭,马范援.加速评估算法:一种提高 Web 结构挖掘质量的新方法[J].计算机研究与发展.2004, 41(1): 98-103.

【18】 戚华春,黄德才,郑月锋.具有时间反馈的 PageRank 改进算法[J].浙江工业大学学报.2005,33(3): 272-275.

【19】 宋聚平,王永成,腾伟，等.对网页 PageRank 算法的改进[J].上海交通大学学报,2003,37(3):397-400.

【20】 Chakrabarti S,Dom B,Gibson D,et al.Automatic Resource Compilation by Analyzing Hyperlink

Structure and Associated Text[C].Proceeding of the Seventh International World Wide Web Conference.Brisbane,1998.

【21】 舒鹏.非机构化数据提取方法研究[D].西安,西安理工大学,2014.

【22】 Brin S, Page L. The Anatomy of a Large-Scale Hypertextual Web Search Engine[C].Computer Science Department, Stanford University.1998.

【23】 邵晶晶. PageRank 算法的阻尼因子值[J].华东师范大学学报(自然科学版).2011,45(4):534-537.

【24】 Langville A N, Meyer C D. Deeper Inside Pagerank[J]. Internet Mathematics.2005,1(3):335-380.

【25】 Haveliwala T H, Kamvar S D,et al. Computing PageRank Using Power Extrapolation[R].Technical Report, Standford University.2003.

【26】 Berkhin P. A Survey on Pagerank Computing[R]. Internet Mathematics.2005,2(I):73-120.

【27】 Attardi G, Gulli A, Sebastiani, F. Automatic Web Page Categorization By Link and Context Analysis[R]. Proceedings of THAI-99, European Symposium on Telematics, Hypermedia and Artificial Intelligence.1999.

【28】 潘浩,谭龙远.邻域相关自适应的 PageRank 算法搜索策略[J].计算机应用.2008,28(9):2192-2194.

【29】 周卉. PageRank 算法的存储问题研究[J].中国科技论文在线. 2009.

【30】 Haveliwala T H. Efficient Computation of PageRank[R].Stanford University: Standford C A.1999.

【31】 孔娟,马亨冰. PageRank 算法的原理与解析[J].福建电脑.2007(1):95-97.

【32】 许南山,丛磊. Pagerank 二维线性收敛方法[J].计算机应用.2006,26(6):1415-1417.

【33】 Haveliwala T H, Kamvar S D. The Second Eigenvale of the Google Matrix [R]. Technical Report, Stanford University.2003.

【34】 汪大明. 复杂网络社团模型与结构研究[D].北京,国防科技大学,2010.

【35】 李凯,郝枫铃,左万利.PageRank-Pro——一种改进的网页排序算法[J].吉林大学学报(理学版). 2003,41(2):175-179.

【36】 Jeh G, Widom J. Scaling Personalized Web Search[C].Proceedings of the twelveth International World Wide Web Conference.Budapest,2003.

【37】 Langville A N, Meyer C D. A Reording for the PageRank Problem. SIAM Journal on Scientific Computing.2006,27(6):2112-2120.

【38】 黄婵,刘发升. PageRank 外推插值法[J].计算机时代.2009(1):51-53.

【39】 熊世春,阿不都热西提.讨论埃特金加速法的收敛性[J].新疆大学学报.2005,22(4):412-415.

【40】 Sidi A. Survey of Numerical Stability Issues in Convergence Acceleration[J].Applied Numerical Mathematics.2010,60(12):1395-1410.

【41】 Langville A N, Meyer C D. Google's PageRank and Beyond:The Science of Search Engine Rankings[M].Princeton University Press.2006.

【42】 曲卫民,张俊林,孙乐,等.基于记忆的自适应汉语语言模型模型的研究[J].中文信息学报.2003, 17(5):13-18.

【43】 Cai Deng, Yu Shipeng, Wen Jirong, et al. VIPS: A Vision-Based Page Segmentation [R]. Microsoft Technical Report,MSR-TR-2003-79,2003.

【44】 高乐,张健,田贤忠.基于视觉的 Web 页面分块算法的改进与实现[J].计算机系统应用.2009(4):65-69.

【45】 Song Ruihua,Liu Haifeng,Wen Jirong,et al.Learning Block Importance Models for Web Pages[C].Proceedings of The Thirteenth Internaltional World Wide Web Conference.New York.2004.

【46】 Diligenti M, Gori M, Maggini M. Web Page Scoring Systems for Horizontal and Vertical Search[C].Proceedings of the Eleventh International World WideWeb Conference.HohoLulu Hawaii.2002.

【47】 黄德才,戚华春,钱能.基于主题相似度模型的 TS-PageRank 算法[J].小型微型计算机系统.2007,28(3):510-514.

【48】 Cafarella M, Cutting D. Building Nutch: Open Source Search[J]. ACM Queue,2004,2(2):56-61.

【49】 李村合,吕克强.Nutch 搜索引擎的页面排序修改方法研究[J].计算机工程与设计.2009,30(6):1343-1345.

【50】 詹恒飞,杨岳湘,方宏.Nutch 分布式网络爬虫研究与优化[J].计算机科学与探索,2011,5(1):68-74.

【51】 王春花.基于 Nutch 的农业搜索引擎检索结果排序策略的研究[D].西安,西北农林科技大学.2010.

【52】 鱼健榕.基于 Nutch 的搜索引擎系统的研究与实现[D].北京,北京邮电大学.2010.

【53】 郎小伟,王申康.基于 Lucene 的全文检索系统与开发[J].计算机工程,2006,32(4):94-96.

【54】 吴玮.基于空间向量模型的垃圾文本过滤方法[J].湖南科技大学学报.2014,29(1):78-83.

【55】 蔡峰.Lucene 排序算法的研究和改进[J].中国新技术新产品.2011(4):15-16.

【56】 袁威,薛安荣,周小梅.基于 Nutch 的分布式爬虫的优化研究[J].无线通信技术.2014(3):44-47.

【57】 王学松. Lucene.Nutch 搜索引擎开发[M].北京,人民邮电出版社. 2008.

【58】 王建勇,单松巍,雷鸣,等.海量 Web 搜索引擎系统中用户行为的分布特征及其启示[J].中国科学(E辑).2001,31(4):372-384.

【59】 王冬,雷景生.一种基于 PageRank 的页面排序算法[J].微电子学与计算机.2009,26(4):210-213.