一种基于 LCS 的相似网页检测算法

黄连恩，王磊，李晓明

北京大学 网络与分布式系统实验室， 100871

报告编号 **PKU_CS_NCIS_TR2007012**
提交时间 2007-12-20
北京大学 信息科学技术学院
网络与信息系统研究所，100871

一种基于 LCS 的相似网页检测算法.

黄连恩，王磊，李晓明
（北京大学 信息科学与技术学院， 100871）

摘要：网页的相似性检测长期以来是一个研究的热点，shingling 和 simhash 被认为是当前最好的两个算法，然而它们存在着一定的不足：一方面，高的分数意味着高的相似概率，但并不必然意味着高的相似度；另一方面，哈希代码的长度和多样性之间的冲突决定了难以同时获得高的准确率和覆盖率。本文提出了一种新型的相似网页检测算法，同时具备高准确率与高覆盖率的优点。该算法采用基于 LCS（longest common subsequence）的相似性度量方法，它可以更好地度量网页之间的相似度和包含关系，并获得高的准确度。同时，本文设计了一个包含了三个步骤的检测过程框架，以保证算法的效率。综合实验表明本文的算法同时获得了高准确率与高覆盖率并具有较好的效率。该算法成功应用于 4.3 亿文章型网页集合，将其分割为 0.68 亿个相似网页子集，整个过程使用 6 台 Linux 服务器仅花费了 5 天的时间。

关键词： 相似性检测; 消重; 最长公共子序列; 相似性度量

Achieving both High Precision and High Recall in Near-duplicate Detection

Huang Lian’en, Wang Lei, Li Xiaoming
Institute of Network Computing and Information Systems, Peking University

Abstract: To Find near-duplicate documents, fingerprint-based paradigms such as shingling and simhash have been recognized as effective approaches and are considered the state-of-the-art. Nevertheless, we see two aspects of these approaches which may be improved. First, high score under these algorithms' similarity measurement implies high probability of similarity between documents, which is different from high similarity of the documents. Second, there has to be a tradeoff between hash-code length and hash-code multiplicity in fingerprint paradigms, which makes it hard to maintain a satisfactory recall level while improving precision. In this paper we propose a new approache of near-duplicate detection for web pages, which has both merits of high precision and high recall. Technically, our approache is based on LCS (longest common subsequence) measurement, together with a 3-step framework, which is carefully designed to ensure high efficiency. A comprehensive experiment was conducted, which shows our method achieves both high precision and high recall. Also, the method has been successfully used to partition a set of 430 million web pages into 68 million subsets of similar pages. The process of partition took only 5 days to complete, using a cluster of 6 linux boxes, which demonstrates its effectiveness.

Key words: near-duplicate detection; replica detection; longest common subsequence; similarity measurement

1. 引言

Web 上相似网页的检测方法长期以来一直都是一个研究的热点。它在很多与 Web 信息相关的应用中扮演着重要的角色，包括：检索结果的聚类和排序、Web 搜集、信息提取、Spam 检测等等。

* 此项工作得到国家自然科学基金项目（60573166）与国家 863 计划项目（2007AA01Z100）支持

正是因为它的重要性，近年来，有很多的方法被提出来并得到评估，其中，Broder 提出的 shingling 算法[1]和 Charikar 的 simhash 算法[2]被认为是当今最好的算法。不过这两种算法存在着一定的不足：它们都是基于哈希匹配的方法来评估两篇文档相似度，而不是基于文档内容的匹配，高的评估分数意味着高的相似概率，但并不必然意味着高的相似度。两个完全不同内容的网页，仍然可能被判定为相似的。同时，文献[3]指出，在基于哈希的相似检测算法中，哈希代码的长度决定了基于哈希方法的准确度，而多样性（multiplicity）则决定了召回率，两者之间的冲突决定了无法同时获得高的准确度和召回率。

本文提出了一种基于 LCS（Largest Common Subsequence）的相似网页检测算法，它不同于基于哈希的方法，能够有效地克服上述两种不足。本文在从 24 亿历史网页中筛选出的 4.3 亿篇文章型网页上进行了综合的实验，结果表明，本文提出的方法同时获得了高的准确度和召回率。

2. 相关研究

国际上对文档相似性检测的研究最初主要针对大型文件系统，后来由被扩展应用于搜索引擎等领域的相似网页检测，如 Stanford 大学的 SCAM 系统。到目前为止许多研究人员在该领域提出了许多方法，这些方法主要可以从两个方面进行区分：从文档提取文档特征的策略，以及由这些特征计算文档签名的策略。在第一个方面，shingles 和文档向量是最常采用的特征。Brin 等[4]在 COPS 系统中以句子为单位来获得 shingles，Broder 等[1]采用单词粒度上的滑动窗口来获得 shingles。Hoad [5]和 Chowdhury [6]等分别基于文档向量开发了相似性衡量标准以及利用词典来改进文档向量；在第二个方面，基于 Shingles 的方法[1,7]通常利用 方法或者 方法来从它们的特征中获得文档签名。而 Charikar 的 Simhash 算法和 Chowdhury 等人的 I-match [6]算法在文档向量上计算文档签名。在这些技术中，Broder 的 shingling 算法和 Charikar 的 simhash 算法被认为是代表了当前的技术发展水平，并已经被用于实际的搜索引擎系统中。

Shingling 算法的相关研究

Broder 的 Shingling 算法[1]用两个网页的 resemblance 和 containment 来衡量其相似性。网页 *A* 和网页 *B* 的 resemblance 是位于区间[0,1]中的一个数值，越接近于 1，这两个网页越相似；类似的，containment 也是位于区间[0,1]中的一个数值，越接近于 1，说明网页 *A* 被网页 *B* 包含的程度越高。

Schleimer 的 Winnowing 算法[8]对经典的 shingling 算法做了改进，以期能够检测出两篇网页的匹配部分，其主要流程是：将文章划分成长度为 *k* 的片段，每 *w* 个片段为一个窗口，在每个窗口中取最小的片段哈希值作为该窗口的哈希值。所有被选择的哈希值作为文章的指纹。该算法能够保证检测出长度大于等于 *w*+*k*-1 的相匹配字符串。

Fetterly 等人[7]在研究相似网页的群体演进的过程中对 shingling 算法作了改进。

Simhash 算法的相关研究

Charikar 的 simhash(相似性哈希)算法[2]通过比较两个网页的哈希值中相同的位所占的比例来衡量相似性，其主要思想是：将网页中的每个 token 映射到一个 *b* 维的向量空间，每一维的值为 1 或者-1。把网页中所有 token 的映射相加得到该网页的一个 *b* 维矩阵。该矩阵中每个非负的元素都置为 1，否则置为 0，如此得到该网页的唯一哈希值。该哈希值所具有的性质是，两个网页的相似程度与这两个网页的哈希值中相同的位的个数成正比。

Manku 等[9]设计了解决 Hamming 距离问题的方法，即从一个 位的指纹集合中快速找到与一个给定的指纹相异的位数最多为 的所有指纹，从而使得相似性哈希在海量网页的相似性检测中成

为实用的技术。

相关的评测工作

Henzinger [10]对 shingling 算法和 simhash 算法的效果进行了大规模评测，通过调整参数来获得近似相等的召回率，在此基础上比较两者的查准率。其实验显示出由于模板的影响，这两种算法在查找相同网站上的相似网页时都不能获得理想的效果，在查找位于不同的网站上的相似网页时查准率较高。

近年来的其他技术

在 Yang 和 Callan [11]的工作中，他们将相似网页的检测问题视为实例层面的受限聚类问题，并开发了半监督性的聚类算法来解决此问题。他们考虑了三种类型的限制：must-link（被确认是属于同一类别的），cannot-link（被确认是属于不同类别的）和 family-link（可能是属于同一类别的）。Huffman 等人[12]将相似性检测问题作为搜索评测的一个部分来进行研究，通过为每一对文档计算多种类型的信号来提高精度。

3. 基于 LCS 的相似性度量

3.1. 背景知识

求解两个序列 *A* 和 *B* 的 LCS 和求解从 *A* 转变到 *B* 的最短编辑脚本（SES，shortest edit script）被认为是同一个问题的两个方面。这个问题的最早算法见于文献[13]，它是一个 $O(N^2)$ 的算法。后来，这个算法经过了很多的改进。其中 Myers 提出的 $O(ND)$ 算法[14]在当参数 *D* 较小时表现得最好。

Myers 的算法基于编辑图（Edit Graph）。在编辑图中，求解 LCS 的问题等价于在编辑图中求解包含最多斜线的路径。图 1 演示一个实际的例子见于文献[14]，两个序列分别为，*A* = abcabba，*B* = cbabac。计算得到 *A* 和 *B* 的 LCS 为：caba。从 *A* 到 *B* 的最短编辑脚本（SES）为“1D 2D 3Ib 6D 7Ic”，也就是说删除第 1、2、6 个字符，在第 3 个字符后插入‘b’，在第 7 个字符后插入‘c’。

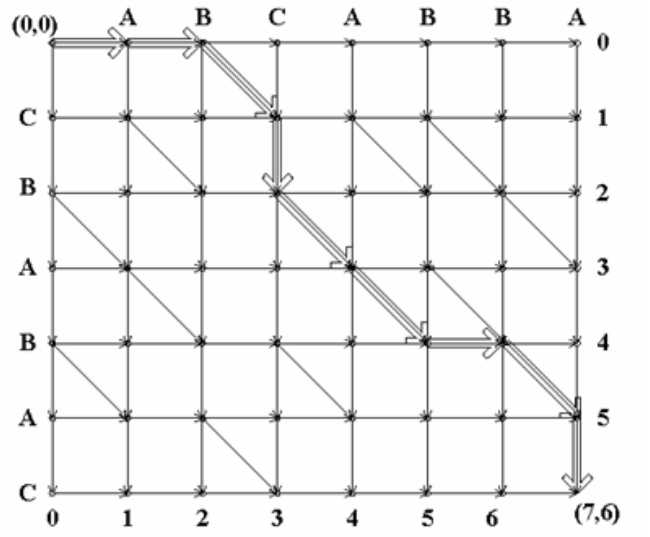

图 1 一个 Edit Graph 例子

3.2. 度量方法

我们定义一个文档为一系列的字符组成。给定两个文档 *A* 和 *B*，|LCS| 为它们的 LCS 的长度，|SES| 为最短编辑脚本的长度。以上述例子为例，设 *A* = abcabba，*B* = cbabac，*A* 和 *B* 的长度分别表示为 |*A*| 和 |*B*|，则有，

$$|A| + |B| = 2|LCS| + |SES|$$

定义 resemble rate 为：

$$r(A,B) = |LCS| / (|A| + |B| - |LCS|)$$

定义 contain rate 为：

$$c(A,B) = |LCS| / |B|$$

这两参数分别表示了两篇文档 *A* 和 *B* 的相似度和包含率。

在本文中，我们判断符合下面条件的两篇文档为相似文档：如果 r(*A*,*B*) 或者 c(*A*, *B*) 超过了一定阈值。

3.3. 相似性度量的一般过程

LCS 是一种两两比较文档的相似性检测方法。假设对于一个文档集合 *R*，经相似性检测后将得到分割为若干相似文档子集 a, b, c, d, … 共 *n* 个。那么，相似性检测的一般过程可描述如下：

1）按一定方法对文档进行排序，原则是尽可能使形成最大子集的文档排在前面。

2）读取第一个文档，放入第一个子集 a。

3）顺序读取其它文档，与已有子集的第一个文档比较，若不相似，新生成一个子集。

分析可知，在这个过程中，每个文档最多将比较 *n*－1 次。因此，每篇文档的比较次数，取决于文档集合 *R* 的不相似文章数量。

3.4. 和 COS 测度的比较

当前，一种常见的相似性度量方法是 Cos（余弦）相似度量，它将两篇文档看作是词的向量，如果 *x*、*y* 为两篇文档的向量，则：

$$Cos(x,y) = \frac{x \cdot y}{\|x\| \|y\|}$$

Cos 测度实际上是 *x* 和 *y* 之间夹角（余弦）的度量。这样，如果余弦相似度为 1，则 *x* 和 *y* 之间夹角为 0°，并且除大小（长度）之外，*x* 和 *y* 是相同的；如果余弦相似度为 0，则 *x* 和 *y* 之间夹角为 90°，并且它们不包含任何相同的词。

简单地比较，LCS 与 Cos 相比，具有下面三点重要的优势：

LCS 体现了词的顺序，而 Cos 没有。显然，词的顺序在网页文档的相似性比较中本身就是一种重要的信息，一个由若干词语按顺序组成的句子和若干没有顺序的词语组成的集合有着完全不同的意义。完全有可能，两篇文档根本不同，但是 Cos 值却很接近 1，特别是当数规模很大的时候。

LCS 可以评估两篇文档之间的包容关系，而 Cos 不能。一般地，如果文档 *A* 仅是文档 *B* 的一个部分，我们仍然认为它们是相似的，这在很多应用中是很重要的，例如搜索引擎结果的消重。

网页上有很多噪音信息，例如网站的模板，广告信息等，它们通常存在于网页的头尾部分，使用 LCS 可以对不同位置文字赋予不同的权值，将这些信息和网页正文信息区别开来，从而提高检测精度。

另一方面，一般认为，Cos 的方法要比 LCS 更快。不过，注意到，对于中文文档来说，这一优势并不是那么明显。因为中文文档应用 Cos 方法时需要进行切词，而这一过程是很耗时的，以我们

的经验为例：我们曾经使用过的一个切词程序，1 秒种仅能切 10 个网页。

由此可见，与 Cos 测度相比，LCS 方法具有一些显而易见的潜在好处。其中最重要的是可以获得更高的准确度，这也是我们研究基于 LCS 方法的一个主要动机。同时本文提出的算法也较好地解决了效率问题。

4. 算法与实现

4.1. 系统框架

给定上述度量方法，我们有三个问题需要解决。前两问题是关于如何有效地计算文档的相似性。考虑到 Myers 的 $O(ND)$ 算法在相似的文档之间的计算很快，这两个问题可以表述为：1）如何在计算 LCS 之前先尽可能排除掉那些完全不需要进行判断的文档，仅当两篇文档具有相似可能性时再进行 LCS 计算；2）如何在计算 LCS 之前进一步减少两者之间的差异以提高计算速度。第 3 个问题则是关于如何在该度量的基础上进一步提高准确度。主要的问题是存在于 LCS 的相同网站的模板可能会造成错误的肯定（false positive），因此第三个问题可以表述为：如何从 LCS 中提取出真正可信的部分（排除了网站模板的干扰）来用于计算文档相似性。

本文算法的系统框架划分为三个步骤：1）将网页文档集合分割为可能相似文档子集，仅在每个可能相似文档子集中计算 LCS；2）在计算两篇网页文档的 LCS 之前，对网页文档进行过滤生成文档过滤框架，在该框架而不是原始文档上进行 LCS 计算；3）计算 LCS 并选取它的可信部分（称为 TLCS）来计算相似度。下面的小节将详细介绍这三个步骤。

4.2. 可能相似子集的计算

本文算法的第一个步骤是将网页文档集合分割为可能相似文档子集，以大大减少每个文档平均需要进行的比较次数，并将相似比较限制在仅在可能相似的文档之间进行，而在这种情况下使 Myers 的 $O(ND)$ 算法会运行得很快。本部分算法包含了三个阶段：网页文档预处理、为每篇文档计算指纹、基于指纹将可能相似文档聚集在一起。

在第一阶段，我们将原始网页文档分割为句子的集合：将所有的 HTML 标签、格式化信息去掉，并按照标点符号将剩下的文本内容分割为句子。

在第二阶段，计算文档的指纹，方法如下：

仅选择那些长度大于一定值 X 的句子，每个对句子计算 MD5 摘要，对 m 取模，不同值分成 m 个组；

对每个组，取其中 MD5 值最小的个 Y 不同句子（并且要求 Y 个句子分布在不同位置，即其覆盖的文字长度达到一定值 Z）计算一个哈希值（128 位），称为指纹，这样共得到 m 个指纹；

对 MD5 值（共 16 字节）循环移位一个字节，重复上面 2 步，这样，最多可以得到 $16\times m$ 个指纹。

因此，在第二步中，对于每个文档，最多可以得到 $16\times m$ 个指纹。调节 X、Y、Z 三个参数，我们的一般取值情况是，$X > 10$，$Y = 4$，$Z > 100$。显然，当参数值越大时，可以认为条件越严格，而当条件严格到一定程度时，观察表明，只要两篇文档存在着同一个指纹，则它们为相似文档的概率就已经很高了。因此，只要存在一个指纹相同，本文就认为它们是可能相似的。然而，由于哈希的缘故，仅使用一个指纹往往会漏掉很多可能相似的文档，因此应当使用多个指纹，将每个指纹找到可能相似文档合并，这样可以达到较好的召回率。本文一般取 $m = 7$，这样最多可以有 112 个指纹。

同时，取决于参数的严格程度，并不是所有文档都可以生成全部的指纹，特别是一些文字内容

很短的文档，有可能一个指纹也没有。参数越严格，这种可能性越高。在我们的网络文章集合仅有1.03%的文章一个指纹也没有，观察表明，这些都是属于很短的没有价值的网页的，因此我们将这类文章全部删除（另一种选择是将它们单独处理，用其它检测方法或使用宽松一点的条件）。

在第三阶段，我们对文档按指纹进行索引，然后将它们聚集到可能相似的子集里。除了两篇文档只要存在一个指纹相同就认为它们可能相似以外，这里我们还考虑到，文档的相似存在着传递性。例如，*A* 与 *B* 可能相似（包含了至少一个相同的指纹），而 *B* 与 *C* 可能相似，那么 *A* 与 *C* 就可能也是相似的。显然，利用传递性可以有效地提高相似检测的召回率，但是不能无限地传递下去，否则会造成巨大的相似子集。我们的办法是，对文档进行排序，相似性只能从大到小单向传递，在上面的例子中，如果 $A > B$，$B > C$，那么通过传递性我们将 *C* 聚集到 *A* 的相似子集中。

因此我们的聚集算法描述如下：

对文档按它可以生产的指纹数从大到小进行排序，显然，指纹越多它就可能找到越多的相似文档。

按顺序从文档集合中逐个读取文档，对于每个文档，根据指纹索引从剩下的文档集合中读取和它可能相似的文档以及根据传递性得到的可能相似文档聚集到一起，形成一个可能相似子集。

在这里我们还使用了一个技巧，它可以有效地消除相同网站模板影响并用于 Spam 内容检测：我们删除了那些出现频度非常高的指纹（> 10000 次）以及那些出现频度（> 400 次）高并且集中在少数网站的指纹。这种类型的指纹绝大部分来源于相同网站模板或 Spam 网页。

4.3. 文档过滤框架的计算

计算文档过滤框架的目的是使 LCS 计算在减小了差异部分的框架上进行，以大大提高 LCS 的计算速度。这里我们采用了和文献[15]类似的方法来计算文档的过滤框架。算法描述如下：

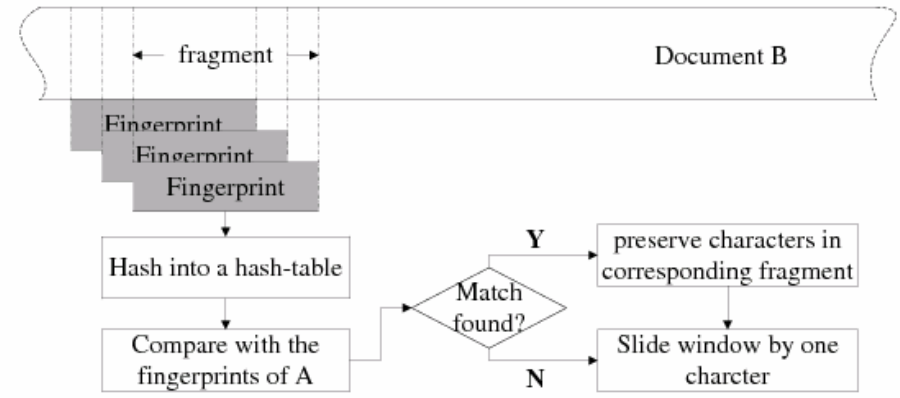

图 2 文档过滤框架的计算过程

如图 2 所示，给定将要进行比较的两篇文档 *A* 和 *B*，

对每篇文档，使用一个长度为 *S* 的滑动窗口来顺序读取一个文字片断，滑动窗口相互是交叉的；对该窗口的文字片断使用 Rabin 的方法[16]计算一个哈希值。

每篇文档将哈希值记录在自己的一个哈希表里（另一种办法是记录在位图里，位图的大小应该远大于文档长度）。

对于 *A* 中的每个哈希值，检查它是否在 *B* 中也存在，如果是，则它所对应的文字片断被保留下来。文档 *A* 的过滤框架即为这些被保留下来的文字片断的交集。

使用同样的方法得到文档 *B* 的过滤框架。

可以看出，本方法的时间代价和空间代价都是 $O(N)$，*N* 为两篇为文档的长度之和。

本方法具有两个特性：1）如果在一篇文档中，它和另一篇文档的两个不同之间隔小于 *S*，那么这两个不同之处中间的文字内容将会被去除，虽然这些文字内容原本是属于 LCS 的部分的，但由于长度不足 *S* 而被过滤掉。这看起来是一种信息上的损失，但实际上它把很多没有意义的孤立的短文字片断给去掉了，例如“的”、“地”、“是”等孤立的常见词语，它们对于后面我们计算 LCS 的可信部分（TLCS）没有帮助。2）由于滑动窗口的缘故，本方法对于少量的文字改动是健壮的，而这在网页上是很常见的，比如因为格式化在每行插入一个换行符、手工录入出现的少量错误、从 PDF 转化成为 HTML 文档等，只要两个文字改动的地点间隔超过 *S*，就仅有那些不同的文字被过滤掉，不会对 LCS 造成损失。因此本方法对 TLCS 计算的损失很少。

计算文档过滤框架还存在着一个好处，即：对于两篇很不相似的文档，在很多情况下根本不需要进行较为费时的 LCS 计算即可判断它们就是不相似的。显然，本文算法后面部分的计算得到的 TLCS 的长度不可能大于过滤框架的长度（取两个中最小的一个），因此，如果使用文档过滤框架的长度代替 TLCS 长度来计算文档相似度，resemble rate 和 contain rate 仍然不能超过规定阈值，即可判断它们是不相似的。这显然可以节省大量的计算时间。

4.4. LCS 可信部分（TLCS）的计算

本文算法的第三步是计算 LCS 并提取出的它的可信部分（TLCS，即 trustable LCS）用于计算文档相似度。LCS 计算采用的是 Myers 的 O(ND)算法，因此本文这里描述的是如何从 LCS 计算得到 TLCS。不过，本文这里所得到的 LCS 并不是两篇文档的严格意义上的 LCS，而是采用了一定的近似：对于文档 *A* 和 *B*，我们首先生成它们的过滤框架，然后在过滤框架上计算 LCS 来近似地代替 *A* 和 *B* 的 LCS；*A* 和 *B* 的 SES 则由过滤框架的 SES 与生成过滤框架的编辑脚本合并（该过程为线性时间复杂度）来近似代替。本文下面的 LCS 和 SES 指的都是近似值。

我们对 TLCS 的定义是基于这样的认识：在网页中的噪音信息例如广告、版权信息等通常都出现在网页正文内容的头尾两端。同一网站的两篇网页，往往在它的正文内容的前后混杂有网站的模板信息，而在正文内容内部，则一般是连续的。因此我们定义 TLCS 为 LCS 在原始网页文档中央的连续内容部分，可以用三个属性来表述：1）连续性：这部分内容在原始文档中是较为连续的，2）中央性：它属于网页文档的中央部分，3）扩展性：在满足前两个特征的前提下应当包含尽可能长的文字内容。

图 3 描述了一个文档的最短编辑脚本（SES）曲线图。图中 *d* 轴表示 SES 的编辑次数，也即文档与 LCS 的差异长度，*x* 轴表示文档的位置。一个区域（region）的斜率定义为连接区域两端点的线段的斜率，它反映了这个区域 LCS 部分的连续程度。本文使用一个参数斜率阈值 θ 来求解 TLCS，即 TLCS 为位于图中中央部分斜率恰好小于 θ 的最大区域文字内容所对应的 LCS 部分。图中斜率阈值设定为 0.2，TLCS 为可信区域（trustable region）对应的 LCS 部分。

图 4 描述了 TLCS 的提取算法，它实际计算得到的是图 3 的扩展区域（extented region），可以看作是 TLCS 的近似。通过将文档分割为 *k* 个块得到 $(k^2+k)/2$ 个可能的基础区域，因此计算基础区域的时间复杂度为 $O(k^2)$，在最后的 R_B 中最多可能得到两个权值最大而覆盖块数最小的基础区域。扩展区域的左右端点各需要最多 *s*-1 次计算，但实际上我们不需要在所有的 *s*-1 位置进行计算而只需要在发生了编辑操作的位置（即 *d* 发生了改变）进行计算，最坏情况下是 *D*（SES 的长度），平均值是 *D*/*k*。因此整个过程在最坏情况下的计算复杂度是 $O(k^2+D)$。

该算法其它的设计考虑还包括：1）对于具有相同权值的区域，我们选择覆盖了较少的块数的区域，这意味着它们更加靠近中央；2）我们计算出来的实际上不是严格的 TLCS 部分，而是一个近似，严格计算需要更复杂的计算方法，但并没有很大的意义。

有的时候，一个文档内部部分段落的位置发生改变后，仍然被认为是相似的。有一些特定的应用会有这样的需求。这个问题可以通过一个迭代的策略来解决：在从文档 *A* 中提取出可信区域后，

将它从文档 *A* 中删除，并将 TLCS 从文档 *B* 中删除。然后使用剩下的部分来计算第二个可信区域，如此迭代进行。对于两个包含了 *n* 个段落的文档来说，最坏的情况是文档 *A* 的段落顺序恰好是文档 *B* 的反序，这就需要有 *n* 次的迭代。考虑到 *n* 通常是很小的，整个计算的复杂度仍然是 *O*(*ND*)。而且，每次迭代所得到的 *LCS* 是单调递减的，这就可以设定一个阈值来控制迭代的深度。

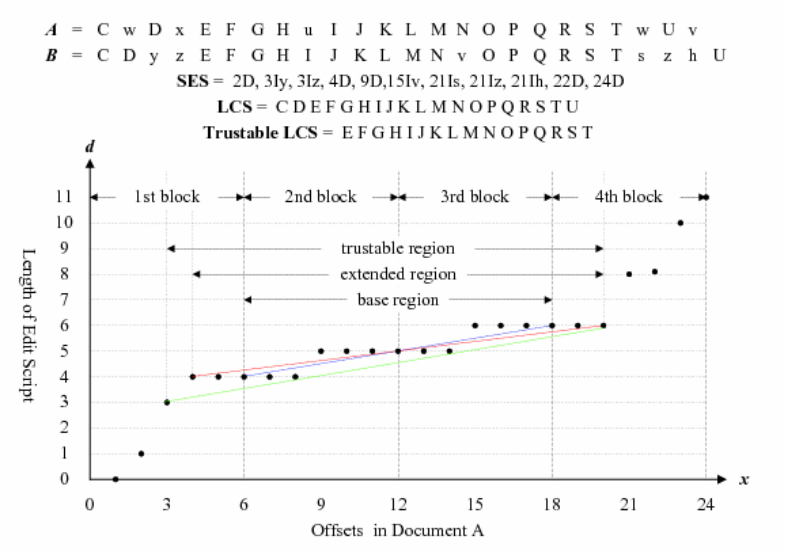

图 3 TLCS 的计算示例

计算基础区域 (base regions)：

1. 将文档 A 分割为 $k = \lceil size(A)/s \rceil$ 个大小为 s 的块，第 i 个块赋予权值 $w_i = \lceil k/2 \rceil - |\lceil k/2 \rceil - i|$.
2. Set $R_B \leftarrow \{\}$. R_B 为基础区域的集合.
3. **For** $0 < i < k$
   For $i < j \leq k$, 第 i 到第 j 块组成一个基础区域 r_{ij}
   If r_{ij} 的斜率 $\leq$ 阈值 θ **then**
   $R_B \leftarrow R_B + \{r_{ij}\}$
   计算 r_{ij} 的权值 $W_{ij} = \sum_{t=i}^{j} W_t$.
4. 过滤 R_B 仅留下最大权值的 W_{ij} (可能有多个).
5. 进一步过滤 R_B 仅留下覆盖块数最少的 W_{ij}.

计算可信区域 (trustable region)：

1. **For** R_B 中的每个基础区域 r_{ij}
   For 文档 A 中从 $(i\text{-}1) \times s + 1$ 到 $i \times s - 1$ 的每个位置
   找到第一个 (最小) 位置 x_l 使得将区域的左端点扩展到 x_l 时区域的斜率刚好小于阈值 θ
   用同样的方法将区域右端点扩展到最大位置 x_r 使得区域的斜率刚好小于阈值 θ.
2. 找到最大 x_r-x_l 即最长扩展区域作为可信区域的近似.

图 4 TLCS 的提取算法

5. 实验评测

一般地，我们认为相似网页检测比较有意义的应用是用于有主题的网页而不是导航型网页。对于导航型网页，我们认为检测相似网页的意义并不大，而且，如何确定两个导航型网页是否相似本身就不好界定。例如，以一个典型的导航型网页新浪首页为例，它不断地在改变，今天的新浪首页和昨天的新浪首页相比较，其内容已经有了一定的改变了，这个时候应该认为它们是相似的还是不相似？其内容改变要到达什么程度才算不相似？这本身是很难界定的，因此也并不适合用来做实验评测。

因此，我们的实验评测主要在一种有主题的网页上进行，但我们相信本文提出的算法同样适用于其它类型的网页和其它类型的文档。我们称新闻报道、分析评论、论坛帖子、博客日记等由人创作完成的作品，具有标题以及一段逻辑上连续的正文，为网络文章。文章型网页指的是包含网络文章的网页。我们的评测数据来源于 WebInfomall（http://www.infomall.cn）的 24 亿历史网页集合，从中筛选得到 4.3 亿个文章型网页。本文的实验便是在 4.3 亿篇文章型网页上进行的，本节下面所说的文档指的就是文章型网页。

为了进行人工评测，我们将两篇文档（或称为文档对）之间的相似关系划分为三种情况：相似、不相似、未知。我们定义如果发现两篇文档的主要内容部分是相同的，则两篇文档是相似的。特别地，本文参考文献[11]的办法，制定了判断两篇文档为相似文档的指导性标准如下：

1) 段落的增加、删除：从一篇文档增加或删除一些段落（小于文字长度的 40%）可以得到另一篇文档。
2) 关键段落：一篇文档包含另一篇文档的关键段落。因此，两篇文档即使包含少量的相同段落，它们仍然可能被认为是相似的文档。
3) 少量的文字出入：在一个段落内只有少量的文字出入（10%以内）。
4) 段落的重排序：少量段落的位置发生改变。

而判断两篇文档为不相似文档的指导性标准同文献[10]所描述的，如果两篇文档所描述的主要对象是不同的，则认为它们是不相似的。举个例子，两个关于商品出售的网页，它们可能包含了很多相同的样板文字，但所描述的商品（在页面中央部分）是不同的，则仍然认为它们是不同的。

除此之外，还有一些情况下，两篇文档之间无法通过上述指导性标准明确地判别出来，我们则将这样的文档对标识为“未知”。在本文中，我们主要评测对象是相似文档，它不包括未知文档。因此也可以将未知文档视为不相似文档。

本文的实验部分是这样组织的：首先，我们评估了指纹数量对生成可能相似文档子集的影响，以及滑动窗口长度对计算文档过滤框架的影响。其次，我们评估了可信 LCS 区域斜率阈值对精度和召回率的影响。然后，我们和 simhash 算法作了一个比较。最后，评估了算法的运行效率。

5.1. 指纹数量的影响

在算法的第一步生成可能相似文档子集时，我们有两个目标：1）给定一个文档，所有它的相似文档应该被尽可能完整地聚集到同一个子集；2）聚集到同一个子集地文档应该尽可能相似。我们使用覆盖率（coverage rate）和可信率（trustable rate）来分别评估这两个目标。覆盖率（coverage rate）定义为：任意给定一个文档，使用我们提出的 TLCS 判断方法（即本文算法除去第一步），仅从它所属的可能相似子集中找到的相似文档数 x 与从整个文档集合中所能找到的相似文档数 X 的比值：x/X。可信率（trustable rate）定义为：任意给定一个文档，在它所属的可能相似子集中，和它相似的文档数 x 与该子集的文档总数 n 的比值：x/n。

我们随机从整个文档集合中选出 150 个文档进行评估。首先，对于每篇选中的文档，我们对整个集合运行一次相似文档检测，使用了 TLCS 判断方法（即本文算法除去第一步），也就是说，对

于每一篇选中的文档，我们将它与所有 4.3 亿篇文档都进行了比较，计算出它们的 TLCS 判断它们是否相似。然后我们对计算机找到的相似文档（称为 similar）进行人工检查以确定它们是否与选中的文档真的相似（称为 true-similar），其中，少数选中文档找到了大量的相似文档（similar），对此我们仅人工检查其中 100 个。其次，对于每篇选中的文档，我们仅在生成后的可能相似子集中，运行同样的相似文档检测。

在算法的第一步中，设置指纹数量为不同的值将得到不同的可能相似子集。因此我们改变指纹数量从 7 到 112（以 7 为增加值）进行实验，共得到 16 组不同数据。TLCS 判断方法的其它参数选择则根据本文下面 5.2 小节和 5.3 小节的实验结果取最优值。（本文为方便阐述，对顺序作了更改）

实验结果如图 5 所示，随着指纹数的增加，覆盖率（coverage rate）不断上升而可信率（trustable rate）不断下降。根据 3.3 小节的分析可知，可信率越低，意味着平均每篇文档需要的比较次数越多，也就意味着更高的计算代价。因而，这意味着可以通过增加计算资源（例如更长的计算时间）来获得更高的覆盖率。

图中，覆盖率（coverage rate）和可信率（trustable rate）看起来在当指纹数为 14 时取得一个平衡。不过，由于在我们的需求中，对于覆盖率的要求要比计算时间上的要求更为重要，因此我们选择了 112 我们系统的指纹数。另一方面，112 也是当前我们可以用的最大值。在这种情况下 4.3 亿篇文章得到的指纹数据占用了 1.2TB 的磁盘空间并需要花费 33 个小时进行索引（indexing）和聚集（clustering）操作。

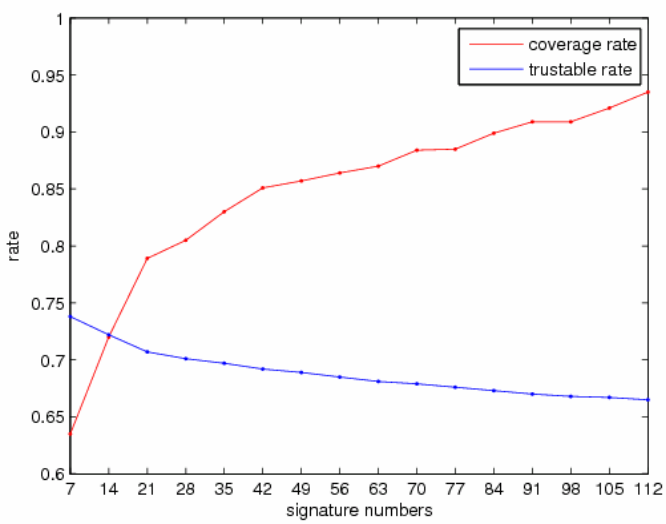

图 5 指纹数量的影响

5.2. 滑动窗口长度的影响

在算法的第二步生成过滤文档框架时，我们同样有两个目标：对于将要进行比较的两篇文档，1）由于大部分 LCS 计算（复杂度为 O(ND)）来说，文档之间差异越小，计算速度越快，因此两篇文档的差异部分在 LCS 计算以前应该被尽可能被过滤掉；2）对于两篇真正相似（true-similar）的文档，其中属于 TLCS 的部分应当被尽可能保留下来。我们分别使用去除率（strainaway rate）和保留率（reserved rate）来评估这两个目标。去除率（strainaway rate）定义为：当两篇文档不是真正相似的文档（not-true-similar）时，它们中被过滤掉的文字内容的长度和文字内容总长度（过滤前）的比值。保留率（reserved rate）定义为：对于两篇真正相似的文档（true-similar），对过滤文档框架计算得到的 TLCS 长度和对原始文档计算得到的 TLCS 长度的比值。

首先，我们在第一步中使用 112 指纹数对 4.3 亿文档集合计算可能相似集，得到 0.46 亿个可能相似子集。然后，我们从全部 4.3 亿文档集合中随机选择 100 个文档，并找到它们所对应的可能相似子集，作为评估对象。对于每篇选中的文档，我们将它的可能相似子集中其它文档经人工检查标识为真正相似（true-similar）或不是真正相似（not-true-similar）。这两类文档然后分别用来计算保留率（reserved rate）和去除率（strainaway rate）。

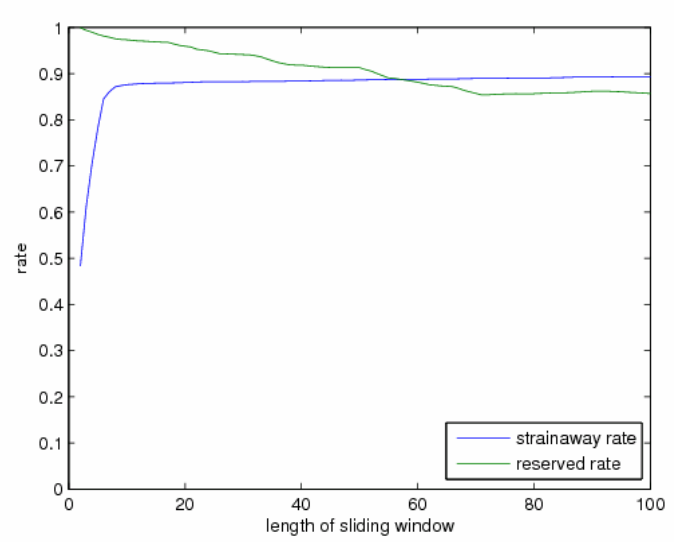

图 6 滑动窗口长度的影响

实验结果如图 6 所示，去除率（strainaway rate）在长度为 2 到 6 时迅速上升，当长度为 6 时达到 0.8。之后维持在一个平滑的状态（非常缓慢上升）。这个现象的原因可能是大多数的汉语词汇都是在 3 个汉字（6 个字节）以下。保留率（reserved rate）则缓慢地下降到达最低点 0.85。

基于这个实验结果，我们的系统采用 16 字节作为滑动窗口长度，此时去除率达到了 0.87 而保留率达到 0.975，这样既在进行 LCS 计算之前显著去除了不相似文档的差异部分，从而大大降低了计算时间，而同时对于真正相似文档的 TLCS 内容，只有很小的影响。

5.3. 相似度阈值的影响

本小节我们对算法第三步提取 LCS 的可信部分（TLCS）中斜率阈值参数的影响进行评估。算法的前两步主要解决的是效率问题，目的是在提高效率的同时实现尽可能少的数据或信息损失。第三步的目的则是要在前两步的基础上，实现尽可能好的效果，这就需要综合评估斜率阈值参数对算法的准确率（precision）和召回率（recall）的影响。我们的方法是维持召回率（recall）在一定值，来评估斜率阈值参数对准确率的影响。

实验方法如下：

一般地，定义准确率（precision）为通过算法找到的相似文档（similar）中真正相似文档（true similar）所占的比例；定义召回率（recall）为我们算法所找到的真正相似文档（true similar）所占全部文档集合中真正相似文档（true similar）的比例。显然，召回率（recall）是无法准确评估的：对于一个文档，我们不可能人工去穷尽所有的文档去找到它的所有真正相似文档。因此，本小节使用如下近似：使用可能相似子集来代替全部文档集合，即假设在可能相似子集中包含了全部的真正相似文档。由于只需要维持召回率不变，这样的近似是合理的。

我们随机选择 100 个文档并取得它们对应的可能相似子集来进行评估（实验数据与上一小节相同）。对每一个选中文档，在它的可能相似子集中使用算法的第三步寻找相似文档，并计算准确率

和近似的召回率，最后取平均值。

实验过程为：改变斜率阈值（threshold of slope）从 0.02 到 0.2，在这个过程中调节相似度量参数使近似的召回率保持在 0.90，计算相应的准确率。相似度量参数如**错误！未找到引用源。**小节所述，有 resemble rate 和 contain rate 两个参数，其中 resemble rate 是主要参数，contain rate 是次要参数，起的作用较小。为简化问题，本次实验仅使用 resemble rate 是否大于一定阈值来判断文档是否相似。

实验结果如图 7 所示，它显示当斜率阈值（threshold of slope）取 0.10 时，准确率（precision）达到最高值 0.948。因此基于此实验，我们的系统采用 0.10 作为斜率阈值。相应地，取 resemble rate 阈值为 0.28。

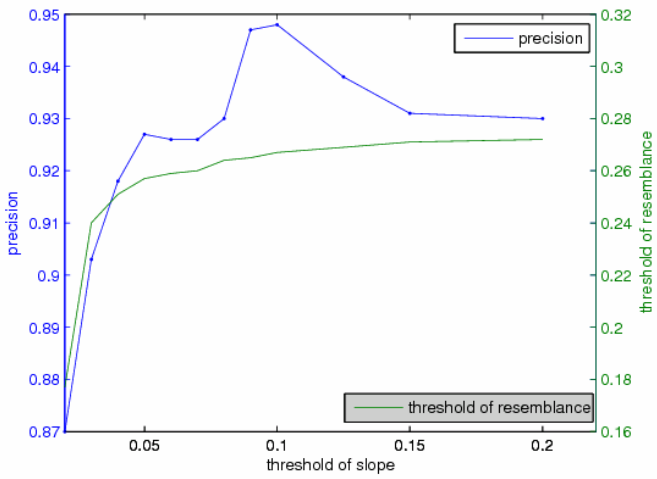

图 7 斜率阈值的影响

这个实验同时表明，本文算法的第三步在一定程度上提高了准确度。如果不采用第三步，则相当于取斜率阈值为无限大值。图中最大取值 0.2 已经足够大，它和取无限大的差别已经很小。可以看到采用第三步后将准确率从大约 0.93 提高到了 0.948。考虑到准确率最大可能值为 1.0，这个提高幅度还是有意义的。

5.4. 和 SimHash 算法的比较

在文献[10]的评估中，在几乎相同的召回率（recall）上，Charikar 的 simhash 算法取得了比 shingling 算法更好的准确率（precision）。因而本文参照 Henzinger 的做法做了一些实验来比较我们的算法和 simhash 算法。我们在两个基准上比较了算法：1）整体的准确率和召回率，2）仅针对相同网站的准确率和召回率。

我们所实现的 simhash 算法和 Henzinger 实现的有所区别，这是因为我们采用了不同的人工评测的指导性标准。我们为每篇文档生成一个 128 位（bit）的向量，并将取得一致的阈值位数设为 97，这在设定的召回率（recall）值下，取得了最高的准确率（precision）。对于我们的算法，取指纹数为 112，滑动窗口长度为 16，斜率阈值为 0.10，resemble rate 的阈值取 0.28，contain rate 的阈值取 0.7。准确率（precision）的计算方法同上一小节所述。召回率（recall）的评估如下所述：我们从 4.3 亿个文档集合中随机选择 1000 个文档，取出它们所在的可能相似子集，合并作为测试集，约 1.2 万个文档。对于选中的每一篇文档，在测试集中计算我们算法所获得的真正相似文档（true-similar）与 simhash 算法所获得的真正相似文档的比值。该比值的平均值被认为是我们算法

召回率和 simhash 算法召回率的比值。

实验结果表明我们算法在准确率（precision）和召回率（recall）上都表现得更好：1）我们算法在整体准确率上达到了 0.95 而同时 simhash 算法是 0.72，而我们算法和 simhash 算法得召回率比值是 1.86；2）仅评估相同网站的文档时，我们算法和 simhash 找到得真正相似文档（true-similar）数几乎是一致的，而精度则分别为 0.91 和 0.52。我们算法能够取得更好准确率的原因在于：我们算法基于“文档内容的匹配”而不是“文档哈希指纹的匹配”，而且相同网站模板的影响得到降低。我们算法能够取得更好召回率的原因在于：1）我们算法能够评估两篇文档之间的包容关系并判断它们是相似的如果满足了条件；2）我们算法在面对真正相似的文档但包含不同网站模板的情况时更加健壮，这也是为什么我们的方法能找到更多的存在于不同网站的真正相似文档。

观察表明，我们算法所造成的错误判断（false-positive）主要来源于 3 个方面：1）那些描述了相似但不相同的网页，例如两个销售计算机的网页，描述了同一系列但不同型号的计算机；2）网页模板占据了网页内容的很大比重；3）网页内容中存在少量差异，但却很关键。

本小节将我们算法与 simhash 算法在小规模测试集上对准确率和召回率进行了比较。不过，由于我们并没有实现在大规模数据条件下的 simhash 算法，因此没有对我们算法和 simhash 算法的效率进行比较，这将是我们未来需要做的一项工作。

5.5. 运行效率

当前，我们已经实现一个相似网页检测系统。我们使用了 6 台安装 linux 系统的服务器，每台机器配备有 6GB 内存、2 个 CPU（Intel Xeon 3.0GHz），可同时运行 4 个进程。使用本文的方法，对 4.3 亿文章型网页集合进行了一次相似性检测，最终将上述网页集合划分为 6800 万个相似网页的子集。整个过程耗时 5 天。

1) 计算可能相似网页子集花费了 2 天，其中计算指纹花费了 14 个小时，索引并将可能相似网页集聚在一起花费了 33 个小时。绝大部分时间花费在磁盘访问上。
2) 计算文档过滤框架和可信 LCS 花费了 3 天，大部分时间花费在 CPU 的运算上。

我们的运行结果表明，本文提出的算法是可以应用于大规模的数据集的，它所需要的计算代价是可以接受的。在这之前，LCS 计算常常被认为是非常耗时的而不适合于网页的相似性检测，因而本文在这一方面的尝试是很有意义的。

6. 总结与下一步工作

本文描述了一种基于 LCS 的相似网页检测算法，同时具有高准确率和高覆盖率的优点。我们设计了一个三步骤的框架来保证效率：第 1 步，使用一种指纹方法将大的文档集合划分为可能相似子集，如果每个子集的的文档足够相似，平均每个文档需要进行的 LCS 比较次数就会很少；第 2 步，在 LCS 比较之前，过滤生成文档的框架，进一步消除文档的差异，对该框架而不是完整的内容进行 LCS 计算；第 3 步，计算可信 LCS，清除噪音信息，再计算文档的相似度。我们并且将本文的算法与 simhash 算法在小规模数据集上进行了比较，实验表明本文的算法在准确率和召回率上都表现得更好。

本文的工作同时也存在一些不足。由于我们并没有实现大规模数据条件下的 simhash 算法，因此没有对两者在大规模数据处理情况下的效率和效果进行比较，这将是我们未来需要做的一项工作。

参考文献

1. A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. *Computer Networks*, 29(1157-1166):8-13, 1997.
2. M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of 34th Annual ACM Symposium on Theory of Computing*, (Montreal, Quebec, Canada, May 19-21, 2002), pages 380-388, 2002.
3. B. Stein. Principles of hash-based text retrieval. In *SIGIR'07*, (July 23-27. 2007, Amsterdam, The Netherlands), pages 527-534, 2007.
4. S. Brin, J. Davis, and H. Garcia-Molina. Copy detection mechanisms for digital documents. *Proc. ACM SIGMOD Annual Conference*, pages 398-409,May 1995.
5. T. C. Hoad and J. Zobel. Methods for identifying versioned and plagiarized documents. *Journal of the American society for information science and technology*, 54(3):203-214, February 2003.
6. A. Kolcz, A. Chowdhury, and J. Alspector. Improved robustness of signature-based near-replica detection via lexicon randomization. In *SIGKDD*, pages 605-610, 2004.
7. D. Fetterly, M. Manasse, and M. Najork. On the evolution of clusters of near-duplicate web pages. In *Proceedings of First Latin American Web Congress*, (LA-WEB, 2003), pages 37-45, 2003.
8. S. Schleimer, D. Wilkerson, and A. Aiken. Winnowing: Local algorithms for document fingerprinting. In *Proc. of the 2003 ACM SIGMOD Int. Conf. on Management of Data*, pages 76–85, 2003.
9. G. S. Manku, A. Jain, and A. D. Sarma. Detecting near-duplicates for web crawling. In *WWW 2007*, (May 8-12, 2007, Ban®, Alberta, Canada), pages 141-149, 2007.
10. M. Henzinger. Finding near-duplicate web pages: A large-scale evaluation of algorithms. In *SIGIR'06*, (August 6-11, 2006, Seattle, Washington, USA), pages 284-291, 2006.
11. H. Yang and J. Callan. Near-duplicate detection by instance-level constrained clustering. In *SIGIR'06*, (Auguest 6-11, 2006, Seattle, Washington, USA), pages 421-428.
12. S. Huffman, A. Lehman, and A. Stolboushkin. Multiple-signal duplicate detection for search evaluation. In *SIGIR'07*, (July 23-27, 2007, Amsterdam, The Netherlands), pages 223-230, 2007.
13. N. Nakatsu, Y. Kambayashi, and S. Yajima. The string-to-string correction problem. *Journal of ACM*, 21(1):168-173, 1974.
14. E. Myers. An O(ND) difference algorithm and its variations. *Algorithmica*, 1(2):251–266, 1986.
15. T.E. Denehy and W.W. Hsu. Duplicate Management for Reference Data. Technical Report RJ 10305, *IBM Research*, October 2003.
16. M. O. Rabin. Fingerprinting by Random Polynomials. Technical Report TR-15-81, Center for Research in Computing Technology, Harvard University, 1981.