ISSN 1673-9418 CODEN JKYTA8
Journal of Frontiers of Computer Science and Technology
1673-9418/2013/07(10)-0924-09
doi: 10.3778/j.issn.1673-9418.1305047

E-mail: fcst@vip.163.com
http://www.ceaj.org
Tel: +86-10-89056056

Pass-Join-K：多分段匹配的相似性连接算法*

余海洋[1]，林 琛[1,2]，陈 珂[3]，江 弋[1]，邹 权[1,2+]
1. 厦门大学 信息科学与技术学院，福建 厦门 361005
2. 厦门大学 深圳研究院，广东 深圳 518057
3. 广东石油化工学院 计算机科学与技术系，广东 茂名 525000

Pass-Join-K: Similarity Join Method Based on Multi-Match Partition*

YU Haiyang[1], LIN Chen[1,2], CHEN Ke[3], JIANG Yi[1], ZOU Quan[1,2+]
1. School of Information Science and Technology, Xiamen University, Xiamen, Fujian 361005, China
2. Shenzhen Research Institute of Xiamen University, Shenzhen, Guangdong 518057, China
3. Department of Computer Science and Technology, Guangdong University of Petrochemical Technology, Maoming, Guangdong 525000, China
+ Corresponding author: E-mail: zouquan@xmu.edu.cn

YU Haiyang, LIN Chen, CHEN Ke, et al. Pass-Join-K: similarity join method based on multi-match partition. Journal of Frontiers of Computer Science and Technology, 2013, 7(10)：924-932.

Abstract: Similarity join is the basic model of data cleaning in the database research and has attracted lots of attention from the database community. This paper studies the edit distance based similarity join, which finds similar strings from two large sets of strings whose edit distance is less than a given threshold, and proposes an improved Pass-Join algorithm, named Pass-Join-K. Pass-Join-K is efficient both for short strings and long strings. The main idea of Pass-Join-K is to divide the query string into more parts based on the partition strategy of Pass-Join, and filter the candidate string pairs more strictly by multi-match. The experimental results show that Pass-Join-K can decrease the candidate pairs, and run 2-5 times more quickly than the origin algorithm which outperforms state-of-the-art methods on real datasets.

Key words: edit distance; similarity join; multi-match; data cleaning; Pass-Join-K

---

* The National Natural Science Foundation of China under Grant Nos. 61102136, 61001013 (国家自然科学基金); the Natural Science Foundation of Fujian Province of China under Grant No. 2011J05158 (福建省自然科学基金); the Basic Research Project of Shenzhen Bureau of Science, Technology and Information under Grant No. JCYJ20120618155655087 (深圳市科技创新基础研究).
Received 2013-04, Accepted 2013-06.
CNKI网络优先出版：2013-07-02, http://www.cnki.net/kcms/detail/11.5602.TP.20130702.1419.002.html.

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

摘 要:相似性连接是数据清理工作的基本模型,获得了大量数据库工作者的关注。研究了基于编辑距离的相似性连接问题,即在两个字符串集合中寻找编辑距离小于一个阈值的字符串对,并在Pass-Join算法的基础上,提出了一个新的Pass-Join-K算法。Pass-Join-K算法在长短字符串上都有很好的表现。该算法的主要思想是利用Pass-Join算法的划分原理,以多次匹配的方式,达到更加严格地选取候选配对的目的。实验结果显示,Pass-Join-K算法减少了候选对的数量,在实际数据集上相比元算法在运行时间上有2~5倍的提升。

关键词:编辑距离;相似性连接;多次匹配;数据清理;Pass-Join-K算法
文献标志码:A 中图分类号:TP391

1 引言

数据清理是数据库领域研究中的一项重要内容,相似性连接问题是数据清理的基本模型,是在两个字符串集合中找出具有高相似度的字符串对。给定两个字符串集合R、S和距离阈值,相似性连接问题即找出所有的字符串对$(r, s)\in R\times S$,使得r和s的距离小于τ。

按照对相似度的定义,可以分为基于Jaccard距离和基于编辑距离(edit distance,ED)的相似度连接[1]。在处理长文本时,多使用Jaccard距离来衡量相似性,处理短文本时则多使用编辑距离。本文主要研究基于编辑距离的相似性连接,主要因为编辑距离拥有以下特点:

⑴ 它反映了字符串中字符出现的次序;

⑵ 它在对齐后,能够保证两个字符串是完全相同的。

这两个特点使得编辑距离能够很好地展示两个字符串的相异性。在生物信息学研究中,使用编辑距离也能够更好地匹配DNA序列。

目前处理相似性连接的方法主要有两种:首先是Filter-And-Refine方法,即先过滤后验证。在过滤阶段,经筛选留下可能的候选配对,其数量远小于总的配对数目,然后再验证这些候选对是否符合要求。其大多是利用Q-Gram[2]来进行相似配对的选择,相关工作有Part-Enum[3]、All-Pairs-Ed[4]、PP-Join[5]、ED-Join[6]。它们在Filter阶段,通过一系列手段来选取有可能相似的字符串对,然后在Refine阶段再对这些可能相似的配对进行详细验证。另一种方式是运用树结构来达到选择相似对的目的,如Trie-Join[7],这种方法利用字符串的前缀进行匹配,去除掉那些在前缀匹配中就已经使得两个字符串不相似的配对。对于Filter-And-Refine方法,当字符串集合中字符串长度不长时,候选对数目会很多,使得在验证阶段要花费大量时间;而树结构的方式不适合处理长字符串,因为长的相似字符串通常不会有很长的相同前缀。

Pass-Join[8]算法基于Partition-Based框架,在长短字符串中,都拥有良好的表现。在计算编辑长度时,设阈值为τ,其核心思想是把字符串分割为$\tau+1$段,由于鸽巢原理,如果两个字符串相似,必定会有一段被匹配上。基于这个思想,本文提出增加划分对的对数,通过更严格的比对,达到减少验证环节时间的目的。

本文的工作包括:改进了Pass-Join的划分字符串算法,并提出了一种多比较的Pass-Join-K算法来改进Partition-Based框架算法。在长短字符串集合中进行了多组实验,验证了本文算法的优越性,并总结了该算法的若干规律。

2 问题描述

Similar Join问题是在两个字符串集合中,寻找相似配对。本文用编辑距离来衡量字符串之间的相似度。编辑距离的定义是,对两个字符串,通过对一个字符串进行最少的插入、删除、替换操作,使得此字符串转换为另一个字符串,如ED(“ABCDEFG”,“ABDCFG”)=3。设编辑距离阈值为τ,即在两个字符串集合R和S中,选择相似字符串配对$<r, s>$,其中每对字符串间编辑距离小于等于τ。

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

为了简化，选取一个数据集，使得在本集合中寻找相似配对，即 $S=R$。例如，在表1中，设 $\tau=3$，那么{<Similarity，Similar>，<Careful，Creful>}就是其中的编辑距离不超过 τ 的相似对。

Table 1 A set of strings

表1 字符串集合

编号	字符串
1	Similarity
2	Similar
3	Hope
4	Careful
5	Creful

3 基于字符串划分的算法框架

首先提出了划分字符串方法，在这种划分方法下，证明了其正确性，并给出了一些简单想法。

3.1 子符串划分方法

给定字符串 s，需要把其划分为 $\tau+k$ 段子串，其中，$length(s)>\tau+k$。如果两个字符串 s 与 r 相似，那么 s 中必有 k 段子串能够与 r 中的一个子串匹配。那么应该如何划分片段呢?很容易就知道，如果子字符串长度越短，就越容易成为长字符串的子串，而拥有相同 k 个子串代表这两个字符串有可能是相似的，因此选择子字符串时，子字符串应该尽量长一点，这样选出来的候选配对会更少。对于字符串 s，划分其为 $\tau+k$ 段子字符串，可以将长度定为 $\left\lfloor\frac{|s|}{\tau+k}\right\rfloor$ 和 $\left\lfloor\frac{|s|}{\tau+k}\right\rfloor+1$，其中 $|s|$ 表示 $length(s)$。令 $P=|s|-\left\lfloor\frac{|s|}{\tau+k}\right\rfloor\times(\tau+k)$，所划分的后 P 段子字符串长度为 $\left\lfloor\frac{|s|}{\tau+k}\right\rfloor+1$，剩下的子字符串长度为 $\left\lfloor\frac{|s|}{\tau+k}\right\rfloor$。例如对字符串 $s=$“similarity”，其中 $\tau=3$，计算得到 $P=2$，那么将 s 划分为4段子字符串为{si，mi，lar，ity}。

3.2 基于字符串子串划分的算法思想

将字符串分为 $\tau+k$ 段子串，如果两个字符串相似，那么它们必然有 k 个子串能够匹配上。

引理1 给定两个字符串 r 和 s，把 r 划分为 $\tau+k$ 段，如果 r 和 s 相似，那么必然存在一种把 s 分为 $\tau+k$ 段的划分，使得有 k 段和 r 中的 k 段匹配上。

证明 假设在任意一种划分中，都没有一种划分能够使得其中至少有 k 段能够匹配，那么由于每个对应子段里，编辑距离至少有一个差异，那么大于 k 个子段不匹配，说明至少有大于 k 个编辑距离的差异。□

为了方便描述，下面给出一些符号的定义。L 表示倒排索引；L_l 表示长度为 l 的字符串的倒排索引；L_l^i 表示长度为 l 的字符串所划分的第 i 个子字符串所对应的倒排索引。

由于两个相似字符串长度不可能大于 τ，在匹配时，对一个长度为 l 的字符串，只需要检查 $L_l^i(|s|-\tau\leqslant l\leqslant|s|+\tau,1\leqslant i\leqslant\tau+k)$，Pass-Join-K算法在 $L_l^i(|s|-\tau\leqslant l\leqslant|s|+\tau,1\leqslant i\leqslant\tau+k)$ 中寻找相似配对，过程如下：

(1) 子串选择

如果 s 串相似，那么 s 必有 k 段子串匹配 L_l^i 的字符串。一个直接的作法就是枚举字符串 s 中所有子串，然后看是否出现在 L_l^i 中。事实上，并不需要枚举所有的字符串，只需要根据 L_l^i 选取一些字符串，然后用这些字符串来寻找相似配对。对每个根据 L_l^i 选取的字符串，检查它是否出现在 L_l^i 中，如果出现了，那么它们就是候选的相似配对。

(2) 验证

验证候选配对是否真正相似，就是计算两个配对间的编辑距离，通常可以使用 $O(n^2)$ 的时间与 $O(n)$ 的空间[9]。文献[10-11]对计算编辑距离的方法进行了优化，可以减少时间复杂度。这里通过文献[8]所介绍的验证方法来计算两个字符串是否相似。

4 Pass-Join-K算法

Pass-Join-K算法是对Pass-Join[8]算法的改进。当 $k=1$ 时，Pass-Join-K就和Pass-Join算法相同；当 $\tau+k$ 等于字符串长度时，该算法就是把一个字符串拆成单个字符进行比较，这样虽然能够保证验证的数目

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

最小，但是由于拆分得太多，从而使倒排索引树很大，查找时间就更多。因此该算法从时间复杂性上来讲，以原算法为基础，随着 k 值的增加，算法执行时间应是先减少后增加。当验证减少的时间小于建立和检索倒排索引的时间时，算法效率将不会增加。总而言之，该算法通过增加 k 值，使得必须匹配上的子串数目增加，从而更加严格地限制了进行动态规划验证的字符串数量。因为通常情况下，由于动态规划复杂度较高，验证算法的时间要高于划分子串与匹配所用时间。

4.1 算法介绍

Pass-Join-K 算法首先对字符串集合中的字符串根据它们的长度进行排序，如果它们长度相同，则按照字符表排序。对目前要比较的字符串 s，Pass-Join-K 算法通过倒排索引，找到与 s 相似的字符串，并在 Multi-Match 计数上加 1，当 Multi-Match 计数大于等于 k 时，则通过优化过的动态规划算法进行比较。算法伪代码如下：

Algorithm Pass-Join-K (S, τ, k)

1. Input: S: a collection of strings
   τ: a given edit-distance threshold
   k: the number of addition segment
2. Output: $A=\{(s \in S, r \in S) | ED(s, r) \leqslant \tau\}$
3. Begin
4. Sort S by string length and second in alphabetical order;
5. for $s \in S$
6. for $L_l^i (|s| - \tau \leqslant l \leqslant |s|)$
7. for $L_l^i (1 \leqslant i \leqslant \tau + k)$
8. $W(s, L_l^i) = SUBSTRINGSELECTION(s, L_l^i)$;
9. for $w \in W(s, L_l^i)$
10. if w is in L_l^i, then
11. Add $L_l^i(w)$ into $C(s)$;
12. $VERIFICATION(s, S, C)$;
13. End

Function $SUBSTRINGSELECTION(s, L_l^i)$

1. Input: s: a string
   L_l^i: inverted index
2. Output: $W(s, L_l^i)$ which contains selected substrings
3. Begin
4. $W(s, L_l^i) = \{w | w$ is a substring of $s\}$;
5. End

Function $VERIFICATION(s, S, C)$

1. Input: s: a string
   S: a collection of strings
   C: candidate numbers
2. Output: $A=\{(s \in S, r \in S) | ED(s, r) \leqslant \tau\}$
3. Begin
4. for $c \in C$ do
5. if the number of c is bigger than k then
6. if $ED(S(s), s) \leqslant \tau$ then
7. $A \leftarrow < S(c), s >$;
8. End

该算法首先介绍了输入输出（第1、2行），接着在第4行对输入的字符串按其长度从小到大进行排序，对于一样长的字符串，以字典序进行排序。然后，对字符串集合里所有的字符串（第5行），选择长度在 $|s| - \tau$ 到 $|s|$ 之间的字符串的分割倒排索引（第6行），对该字符串的第1到 $\tau + k$ 段子串（第7、8行）进行抽取，并把抽取出来的子串与倒排索引中的相应位置进行比较，如果与相应位置匹配，则把 $L_l^i(w)$ 在 $C(s)$ 中的计数加1（第9~11行）。最后，进行验证，并将串划分为 $\tau + k$ 段后加入倒排索引中（第12行）。

下面对函数 $SUBSTRINGSELECTION(s, L_l^i)$ 的选择进行探讨。至于验证过程，请参考文献[8]的第五部分。

4.2 子串选择

对于子字符串的选择，必须保证所得到的结果具有完备性，即选出来的配对都是相似的，同时所有相似配对，都能够通过这个方法选择出来。一个简单的方法就是把所有 s 的子串都提取出来。对长度为 l 的子串，s 共有 $|s| - l + 1$ 个子串，因此子字符串的总数将会非常大。通常所选择出来的候选配对字符串数量越少，则这个算法性能就越好，因此需要减少提取出来的候选配对子串的数目。

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

因为在 L_l^i 中，其分段拥有相同的长度，将第 i 段子串长度表示为 l_i，一种最简单的方法是寻找其他字符串中所有长度为 l_i 的子串。对每段长度为 l_i 的子串，选择出来的数目是 $|s|-l+1$，那么总数目则是 $(\tau+k)(|s|+1)-l$。

虽然上面的方法能够产生满足完备性的子串，但是所产生的候选配对数目太多，同时在匹配时，没有考虑每段的相应信息。对 L_l^i，设起始位置为 p_i，则 $p_1=1, p_i=p_1+\sum_{k=1}^{i-1} l_k$。由于在匹配两个字符串时，子串匹配时的位置相差不能太多，比如对于字符串对<a**bc**def，ade**bc**f>，匹配时如果前串的‘ bc ’和后串的‘ bc ’相匹配，那么必然会导致前串的‘ a ’与后串的“ ade ’匹配，那么由于它们间的长度相差为 2，那么它们之间的编辑距离至少为 2。从而在选取子串时，只需要考虑子串的起始点在 $[p_i-\tau, p_i+\tau]$ 之间的子串，那么子串总数目为 $(\tau+k)(2\tau+1)$。同时，很容易发现，如果两个字符串同样的长度，左边相应的线段相差为 τ，右边也为 τ，那么加起来至少为 2τ，因此需要加起来为 τ，从而可以把匹配的数目再减少一点，这样很容易得到如下算法。

4.3 基于位置信息的子串选择

由于已经排好序了，从而对新加入的字符串，其长度肯定大于等于前面已经比较过的字符串长度。令 r 为加入倒排索引的字符串，s 为新加入的字符串，如图 1。

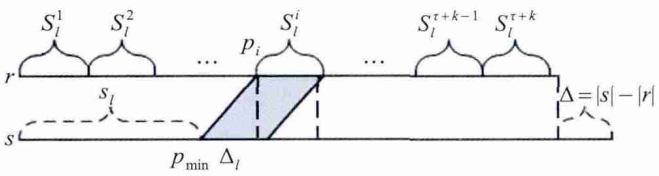

Fig.1 Substrings selection based on position information (left)

图 1 基于位置信息的子串选择(左)

首先考虑 r 中，S_l^i 相对于匹配上 s 的位置 $p_{\min}<p_i$，从而有 $\Delta_l+(\Delta_l+\Delta)\leqslant d_l+d_r\leqslant\tau$，$\Delta_l\leqslant\left\lfloor\frac{\tau-\Delta}{2}\right\rfloor$，因此：

$$p_{\min}=p_i-\Delta_l\geqslant p_i-\left\lfloor\frac{\tau-\Delta}{2}\right\rfloor$$

又因为 $p_{\min}\geqslant 1$，所以：

$$p_{\min}=\max\left(1, p_i-\left\lfloor\frac{\tau-\Delta}{2}\right\rfloor\right)$$

同理，参考图 2，根据对称性可得最大距离为：

$$p_{\max}=\min\left(|s|-l_i+1, p_i+\left\lfloor\frac{\tau-\Delta}{2}\right\rfloor\right)$$

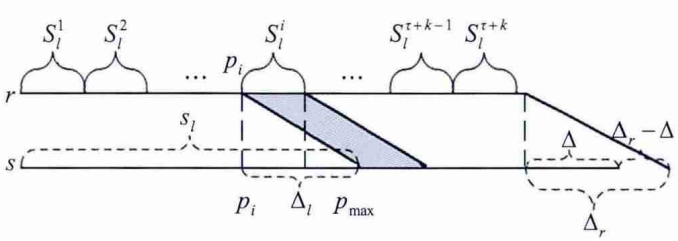

Fig.2 Substrings selection based on position information (right)

图 2 基于位置信息的子串选择(右)

4.4 基于子字符串划分信息的子串选择

如图 3，先从最左边的 S_l^1 开始匹配。当匹配到 S_l^1 时，如果 r 与 s 能够匹配，同时 S_l^1 右边的 $\tau+k-1$ 个线段至少有 k 个线段能够匹配上，那么可以放弃匹配 S_l^i。因为就算这个不匹配，那么在后面的工作中，也能够把足够的匹配段，即大于等于 k 个匹配段拿出来。如果在 S_l^i 左边的 $i-1$ 个线段中，得到的编辑距离大于等于 i，那么说明 S_l^i 右边最多能够匹配的编辑距离为 $\tau-i$，而右边所得到的段数为 $\tau+k-1$，说明如果 r 与 s 匹配，由于子串不会不被匹配，则两个对应子串的编辑距离至少会产生 1 个不同，那么右边至少有 k 个线段是能够匹配上的，因此这个线段就不用匹配了。令 $ED(r,s)$ 表示字符串 r 和 s 的编辑距离，r_l 表示在字符串 r 中 S_l^i 左边的部分，r_r 表示在字符串 r 中 S_l^i

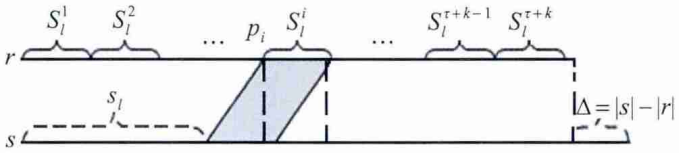

Fig.3 Substrings selection based on substrings partition information (left)

图 3 基于子字符串划分信息的子串选择(左)

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

右边的部分。从而当 $ED(r,s)\geqslant i$ 时，如果知道 r_r 与 s_r 至少有 k 段匹配，S_l^i 就可以不用再去匹配了。因此只有当 $ED(r,s)<i$ 时，才需要把 S_l^i 拿去匹配，从而可以得到：

$$p^l_{\min_i}=\max(1,p_i-(i-1))$$

$$p^l_{\max_i}=\min(|s|-l_i+1,p_i+(i-1))$$

同理，当从右边匹配时，如图4，为了保证 S_l^i 左边有 k 个相似的线段，那么 S_l^i 右边的编辑距离不能超过 $\tau+k-i$。因此由对称性可得：

$$p^r_{\min_i}=\max(1,p_i+\Delta-(\tau+k-i))$$

$$p^r_{\max_i}=\min(|s|-l_i+1,p_i+\Delta+(\tau+k-i))$$

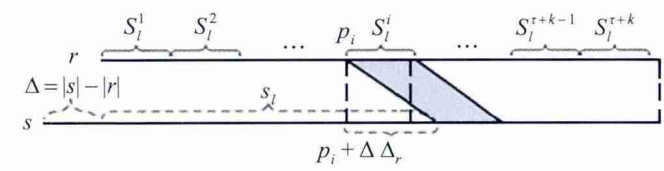

Fig.4 Substrings selection based on substrings partition information (right)

图4 基于子字符串划分信息的子串选择(右)

综合上面两组讨论：

$$p_{\min_i}=\max(p^l_{\min_i},p^r_{\min_i})$$

$$p_{\max_i}=\min(p^l_{\max_i},p^r_{\max_i})$$

4.5 复杂度分析

4.5.1 空间复杂度

由于本文算法涉及到建立倒排索引，因此先分析空间复杂性。因为在对字符串进行匹配时，可以用Hash映射的方式编码字符串，所以在倒排索引中，每个子串所用位置应该是一个整数值，占用空间为1，从而整个空间复杂度为：

$$O(\sum_{l=l_{\min}}^{l_{\max}}(\tau+k)\times|S_l|)$$

其中，$l_{\min}$ 和 $l_{\max}$ 分别指字符串中最短长度与最大长度；$|S_l|$ 表示长为 l 的字符串数目。算法中匹配子串时，l 的取值范围为 $|s|-\tau\leqslant l\leqslant|s|$，因此在 $|s|-\tau$ 之前的倒排索引将不会在以后的算法中用到，从而可以删除掉，于是空间复杂度可以进一步优化为：

$$O(\max_{l_{\min}\leqslant j\leqslant l_{\max}}\sum_{l=j-\tau}^{j}(\tau+k)\times|S_l|)$$

4.5.2 时间复杂度

下面讨论时间复杂性，首先进行排序，可以先把相同长度的字符串放在一起，然后对相同长度的字符串进行排序，排序时间复杂度为 $O(\sum_{l=l_{\min}}^{l_{\max}}|S_l|\times\mathrm{lb}\,|S_l|)$。由算法5~8行可以看到，选取子串的时间复杂度为：

$$O(\sum_{s\in S}\sum_{l=|s|-\tau}^{|s|}\sum_{i=1}^{\tau+k}\Gamma(s,L_l^i))$$

其中，$\Gamma(s,L_l^i)$ 是子字符串集合 $W(s,L_l^i)$ 的大小。由后面的讨论可以看到，该复杂度为 $O(\tau)$，因此选择子串的时间复杂度为 $O(\tau^2\times(\tau+k)\times|S|)$。而验证的时间复杂度为：

$$O(\sum_{s\in S}\sum_{l=|s|-\tau}^{|s|}\sum_{i=1}^{\tau+k}\sum_{w\in W(s,L_l^i)}\sum_{r\in L_l^i(w)\&\&C(r)>k}V(s,r))$$

其中，$V(s,r)$ 是验证复杂度，为 $O(\tau\times\min(|s|,|r|))$ [8]。由于 τ 值通常较小，会比字符串长度小很多，当 k 值增加时，选择子串的时间会增加，但是验证时间会减小。在后面的实验部分，可以看到，当 k 值为2或3时，在数据集中提升最为明显，当 k 继续增大后，验证所减少的时间已经不能够抵消掉增加选择子串部分的时间，从而使得时间增加。

5 实验分析

本文实验中采用与原算法Pass-Join相同的数据集[8]，即DBLP (Digital Bibliography & Library Project)上的Author+Title字段与美国在线(American Online，AOL)上的查询日志字段。由于原算法Pass-Join已经在时间效率上优于最新的算法如ED-JOIN[6]与TRIE-JOIN[7]，为了公平地比较，本文采用与Pass-Join算法同样的配置，即使用C++进行编码，并用GCC编译器所提供的O3参数生成应用文件。

5.1 与原算法Pass-Join时间效率上的比较

原算法Pass-Join在时间效率上优于ED-JOIN与TRIE-JOIN，因此本文只比较了Pass-Join-K算法和

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

Pass-Join算法的时间。从图5中可以看到，Pass-Join-K算法的时间效率相比Pass-Join算法有较大提升。在数据集Log中，使用Pass-Join-K算法，当$\tau=10$时，所使用的时间从原算法的1 502.79 s降低到374.08 s，时间效率提升了4倍多，这正是在大规模数据上所需要的特性。同时观察到，τ值越大，效果提升也就越明显。

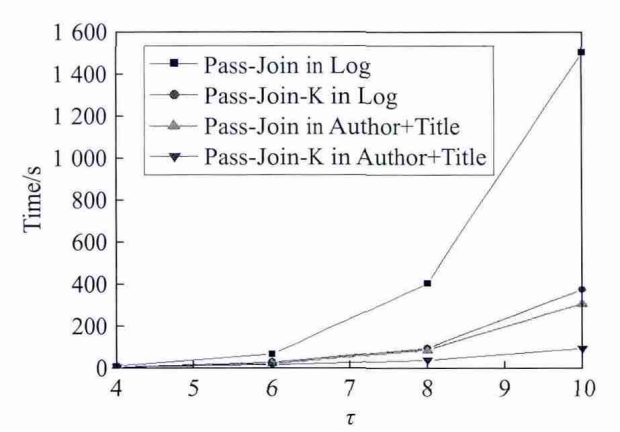

Fig.5 The comparison of time between Pass-Join and Pass-Join-K

图5 Pass-Join与Pass-Join-K的时间对比

5.2 k值对时间效率的影响

在两个数据集中，测试了k的选取对时间效率的影响。首先，在Log数据集上测试了k值的影响，结果如图6所示。发现随着k的增长，时间效率先提升后下降。经过分析发现，这是因为原算法Pass-Join中，把每个字符串划分为$\tau+1$个子串，而本文Pass-Join-K算法中，把字符串划分为$\tau+k$个子串，从而在倒排索引中会加入更多的项，所以虽然随着τ值的增加，需要验证的限制条件越来越严格，进行验证的项目越来越少，但是随着k值的增加，倒排索引也就越来越大，在检索时，所用时间也就越来越多。从而并不是随着k值的增大，算法所用时间越来越少。不过几乎在所有情况下，$k=2$时所用时间都要比原算法少1倍以上。

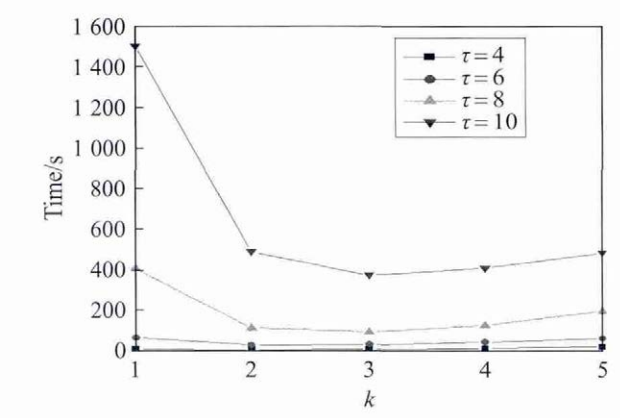

Fig.6 Elapsed time with different k

图6 对应不同k值所用时间

5.3 k值的选取

由于用户并不能确定得到最优的k值，本文设计了自适应算法，即从源数据集中提取1%的数据来比较选择最佳的k值，然后再总体采用这个k值，这样就比最佳的k值算法需要更多的时间，但是其效果介于原算法与最佳算法之间。不过本文推荐选取k值为2或者3，因为经过多次实验发现，k值为2或者3几乎能达到最佳的提升效果。至于更加精确的k值选取，则是未来的研究工作。

6 结束语

本文选取划分$\tau+k$个子串，并通过更加严格的限制验证条件减少了比较次数，大大提升了算法的效率。在下一步工作中，将继续深入研究其内部规律，找到具体的k值选取规律，同时将本文算法应用到其他的如余弦相似度等相似函数上。由于其可分割的特性，文献[12-15]提出了一系列方法把Similarity Join问题应用到MapReduce框架下。而如何将本文算法应用到MapReduce框架下，将是下一步要研究的工作。

References:

[1] Chaudhuri S, Ganti V, Kaushik R. A primitive operator for similarity joins in data cleaning[C]//Proceedings of the 22nd International Conference on Data Engineering (ICDE '06). Washington, DC, USA: IEEE Computer Society, 2006: 5.

[2] Gravano L, Ipeirotis P G, Jagadish H V, et al. Approximate string joins in a database (almost) for free[C]//Proceedings of the 27th International Conference on Very Large Data Bases (VLDB '01), Rome, Italy, 2001. San Francisco, CA, USA:

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

Morgan Kaufmann Publishers Inc, 2001: 491-500.

[3] Arasu A, Ganti V, Kaushik R. Efficient exact set-similarity joins[C]//Proceedings of the 32nd International Conference on Very large data bases (VLDB '06), Seoul, Korea, 2006: 918-929.

[4] Bayardo R J, Ma Yiming, Srikant R. Scaling up all pairs similarity search[C]//Proceedings of the 16th International Conference on World Wide Web (WWW '07), Alberta, Canada, 2007. New York, NY, USA: ACM, 2007: 131-140.

[5] Xiao Chuan, Wang Wei, Lin Xuemin, et al. Efficient similarity joins for near duplicate detection[C]//Proceedings of the 17th International Conference on World Wide Web (WWW '08), Beijing, China, 2008. New York, NY, USA: ACM, 2008: 131-140.

[6] Xiao Chuan, Wang Wei, Lin Xuemin. Ed-join: an efficient algorithm for similarity joins with edit distance constraints[J]. Proceedings of the VLDB Endowment, 2008, 1(1): 933-944.

[7] Wang Jiannan, Feng Jianhua, Li Guoliang. Trie-join: efficient Trie-based string similarity joins with edit-distance constraints[J]. Proceedings of the VLDB Endowment, 2010, 3(1/2): 1219-1230.

[8] Li Guoliang, Deng Dong, Wang Jiannan, et al. Pass-join: a partition-based method for similarity joins[J]. Proceedings of the VLDB Endowment, 2011, 5(3): 253-264.

[9] Wagner R A, Fischer M J. The string-to-string correction problem[J]. Journal of the ACM, 1974, 21(1): 168-173.

[10] Masek W J, Paterson M. A faster algorithm computing string edit distances[J]. Journal of Computer and System Sciences, 1980, 20(1): 18-31.

[11] Myers G. A fast bit-vector algorithm for approximate string matching based on dynamic programming[J]. Journal of the ACM, 1999, 46(3): 395-415.

[12] Vernica R, Carey M J, Li Chen. Efficient parallel set-similarity joins using MapReduce[C]//Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data (SIGMOD '10). New York, NY, USA: ACM, 2010: 495-506.

[13] Baraglia R, De Francisci Morales G, Lucchese C. Document similarity self-join with MapReduce[C]//Proceedings of the 2010 IEEE 10th International Conference on Data Mining (ICDM '10). Washington, DC, USA: IEEE Computer Society, 2010: 731-736.

[14] Kim Y, Shim K. Parallel top-k similarity join algorithms using MapReduce[C]//Proceedings of the 2012 IEEE 28th International Conference on Data Engineering (ICDE '12). Washington, DC, USA: IEEE Computer Society, 2012: 510-521.

[15] Blanas S, Patel J M, Ercegovac V, et al. A comparison of join algorithms for log processing in MapReduce[C]//Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data (SIGMOD '10). New York, NY, USA: ACM, 2010: 975-986.

YU Haiyang was born in 1990. He is a master candidate at Xiamen University. His research interests include data mining and parallel computing, etc.

余海洋(1990—),男,重庆合川人,厦门大学硕士研究生,主要研究领域为数据挖掘,并行计算等。

LIN Chen was born in 1982. She received her Ph.D. degree in computer science from Fudan University in 2010. Now she is a lecturer at Xiamen University, and the member of CCF. Her research interests include Web data mining, the intelligent management of structured and unstructured data.

林琛(1982—),女,福建厦门人,2010年于复旦大学计算机科学专业获得博士学位,现为厦门大学讲师,CCF会员,主要研究领域为Web数据挖掘,大规模结构化与非结构化数据的智能管理等。

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

CHEN Ke was born in 1964. He is an associate professor at Guangdong University of Petrochemical Technology. His research interests include data mining and bioinformatics.

陈珂(1964—),男,黑龙江牡丹江人,广东石油化工学院副教授,主要研究领域为数据挖掘,生物信息学。

JIANG Yi was born in 1960. He is an associate professor at Xiamen University. His research interests include database, data mining and bioinformatics.

江弋(1960—),男,福建福州人,厦门大学副教授,主要研究领域为数据库,数据挖掘,生物信息学。

ZOU Quan was born in 1982. He received his Ph.D. degree in artificial intelligence and information processing from Harbin Institute of Technology in 2009. Now he is an assistant professor and master supervisor at Xiamen University, and the member of CCF. His research interests include bioinformatics and data mining.

邹权(1982—),男,黑龙江佳木斯人,2009年于哈尔滨工业大学人工智能与信息处理专业获得博士学位,现为厦门大学计算机科学系助理教授、硕士生导师,CCF会员,主要研究领域为生物信息学,数据挖掘。

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net