Journal of Computer Applications
计算机应用,2016,36(1):27-32
ISSN 1001-9081
CODEN JYIIDU
2016-01-10
http://www.joca.cn

文章编号:1001-9081(2016)01-0027-06
DOI:10.11772/j.issn.1001-9081.2016.01.0027

# 基于划分的增量式字符串相似性连接方法

燕彩蓉*,朱　斌,王　健,黄永锋
(东华大学 计算机科学与技术学院,上海 201620)
(*通信作者电子邮箱 cryan@dhu.edu.cn)

**摘　要**:字符串相似性连接是数据质量管理的基本操作,也是数据价值发现的关键步骤。针对目前已有的方法不能满足面向大数据的增量式处理需求的问题,提出一种面向流式数据的增量式字符串相似性连接方法——Inc-Join,并对方法的索引技术进行了优化。该方法以 Pass-Join 字符串连接算法为基础,首先,采用字符串划分技术将字符串划分成多个互不相交的子串;然后,建立字符串的反向索引列表并将其作为状态;最后,新增数据只需根据状态进行相似性计算,每次连接操作结束后都对状态进行更新。实验结果表明,Inc-Join 方法在不影响连接准确率的同时,有效将长、短字符串重复匹配次数减少为$\sqrt{n}$($n$ 是批处理方式的匹配次数)。实验对 3 种数据集进行处理,发现使用批处理方式进行相似性连接的响应时间是 Inc-Join 的 1 至 4.7 倍,并呈现急剧递增的趋势;而且优化后 Inc-Join 方法的响应时间最小只占优化前的 3/4,并随处理数据的增多所占比例越来越小。同时优化后的 Inc-Join 不需要保存状态,再一次减小了算法执行的时间和空间开销。

**关键词**:字符串相似性连接;增量处理;划分;字符串匹配;反向索引
**中图分类号**:TP311　　**文献标志码**:A

# Partition-based incremental processing method for string similarity join

YAN Cairong*, ZHU Bin, WANG Jian, HUANG Yongfeng
(*School of Computer Science and Technology, Donghua University, Shanghai* 201620, *China*)

**Abstract:** String similarity join is an essential operation of data quality management and a key step to find the value of data. Now in the era of big data, since the existing methods can not meet the demands of incremental processing, an incremental string similarity join method oriented streaming data, called Inc-Join, was proposed. And the string index technique was optimized. Firstly, based on the Pass-Join string join algorithm, strings were divided into some disjoint substrings by utilizing partition technique; secondly, the inverted index of strings was created and acted as a state; finally, the similarity calculation was done according to the state when new data came, and the state would be updated after each operation of string similarity join. The experimental results show that Inc-Join method can reduce the number of reduplicate matching between short or long strings to $\sqrt{n}$ ($n$ is the number of matching with batching processing model) without affecting the join accuracy. The elapsed time of string similarity join with batching processing model was 1 to 4.7 times the time Inc-Join needs when three different datasets were processed, and it tended to increase sharply. And the minimum elapsed time of optimized Inc-Join only accounted for 3/4 of original elapsed time of Inc-Join. With the increasing number of strings, the elapsed time of optimized Inc-Join would account for less and less of proportion in original elapsed time. The state need not to be saved, so the optimized Inc-Join further reduces time and space cost of Inc-Join.

**Key words:** string similarity join; incremental processing; partition; string matching; inverted index

## 0　引言

字符串相似性连接是数据质量管理的基本操作,引起学术界广泛关注[1]。字符串相似性连接的目的是选定一种算法计算字符串之间的相似度,并在数据集中找出相似度在给定阈值范围内的字符串对。计算字符串相似度的算法主要有编辑距离、杰卡德距离和 Cosine 距离及其变型[1-4],特点是根据字符串本身的结构来计算其相似度;也有算法在计算相似度时考虑了字符串之间的同义词关系[5]。

实际应用时通常会选择一些简单且高效的算法进行相似性连接,这些算法大致可以分为两类:第 1 类算法能高效处理长字符串,但不适用于短字符串,例如 Part-Enum[6] 和 ED-Join[7];第 2 类算法能高效处理短字符串,但不适用于长字符串,例如 Trie-Join[8]。Pass-Join[1-2] 则能同时高效地处理长、短字符串相似性连接,它是一种基于划分的算法。基于划分算法的思想是根据一种划分规则将字符串 $s$ 划分成多个子串,如果字符串 $r$ 与 $s$ 相似则 $r$ 至少包含 $s$ 中的一个子串[3]。基于划分算法的划分规则有很多,包括基于字符串长度的划分

**收稿日期**:2015-07-12;**修回日期**:2015-08-08。
**基金项目**:国家自然科学基金资助项目(61402100);中央高校基本科研业务费专项(2232013D3-15)。
**作者简介**:燕彩蓉(1978-),女,湖北仙桃人,副教授,博士,主要研究方向:并行计算、分布式计算、大数据处理;　朱斌(1990-),男,江西吉安人,硕士研究生,主要研究方向:并行计算、分布式计算、大数据处理;　王健(1989-),男,河南信阳人,硕士研究生,主要研究方向:并行计算、分布式计算、大数据处理;　黄永锋(1971-),男,山东泰安人,副教授,博士,主要研究方向:数据挖掘、机器学习、图像处理。

和基于字符串频率向量[4]的划分。

随着互联网和移动技术的发展,各种在线应用系统不断增加[9],大数据给现有应用系统带来挑战[10-11],流式数据使得相关批量处理方法和技术亟待改进[12],但是已有的字符串相似性连接算法都是基于有限内存的批量处理[4],即要求数据必须一次性读入内存。大数据时代这种做法非常不可行,增量式处理技术已越来越广泛地运用于实际应用中[13]。联机聚集和增量的多维数据集成能实时、高效地得到 OLAP (Online Analytical Processing)的分析结果[14];增量式的机器学习法经常被用于解决大规模的知识发现问题;增量式的网页爬取和索引建立对于搜索引擎来说是非常重要的[15-16];文献[17]实现了一个增量式的处理系统,并将其用来建立 Google 网页查询的索引。随着学术论文数量的递增,如何从众多的文献中快速检索出重复率较高的文章是增量式字符串相似性连接的重要应用之一。研究发现 Pass-Join 虽然能高效地过滤字符串[18],但在处理增量数据上存在一定的缺陷,因为它会导致很多冗余的计算,特别是处理需要密集型计算的大规模数据会严重影响执行效率[19]。

本文在 Pass-Join 的基础上提出了一种处理流式数据的增量式字符串相似性连接方法——Inc-Join,主要思想是把字符串的反向索引列表作为状态,迭代式地更新处理历史数据得到的状态,并运用新的状态对增量数据进行相似性连接。状态由字符串的 $\Gamma+1$ 个子串以及每个子串后面保存的包含当前子串的字符串组成。综上所述,本文的贡献点如下。

1)采用划分技术,以 Pass-Join 字符串连接算法为基础,提出了一种面向流式数据的增量式处理方法 Inc-Join,提高了连接效率。

2)对字符串索引技术进行优化,减少了算法运行带来的时间和空间消耗。

3)通过实验证明,Inc-Join 方法能有效、便捷地处理增量数据,极大地减少了字符串之间的重复计算,提高了字符串相似性连接的效率。

## 1 相关概念

为了详细阐述 Inc-Join 的执行流程,先作一些定义和说明。

**定义 1** 字符串相似性连接。给定两个字符串集合 $R$ 和 $S$,以及一个编辑距离阈值 $\Gamma$,字符串相似性连接是指找出两个数据集中所有相似的字符串组合$\langle r,s\rangle \in R\times S$,且满足 $ED(r,s)\leqslant\Gamma$。$ED(r,s)$ 是两个字符串之间的编辑距离,即 $r$ 转变成 $s$ 所需要经过的最少编辑操作次数。编辑操作包括插入、删除和替换。

**定义 2** $S(n)$、$S(n)_l$、$S(n)_l^i$、$\zeta(n)_l^i$。$S(n)$ 表示 $S$ 被分割成的较小数据集,$S$ 是较大数据且 $0\leqslant n<\lceil|S|/size\rceil$,$|S|$ 是 $S$ 中字符串的数量,$size$ 是 $S(n)$ 中字符串数量。$S(n)_l$ 表示 $S(n)$ 中字符串长度为 $l$ 的集合。$S(n)_l^i$ 表示将 $S(n)_l$ 中的字符串划分成子串之后的第 $i$ 个子串。$\zeta(n)_l^i$ 表示为每一个 $S(n)_l^i$ 建立的索引,索引内容包括当前子串及长度为 $l$ 且包含当前子串的字符串。

**定义 3** 基于长度的字符串划分。给定一个字符串 $s$,基于长度的字符串划分是指将 $s$ 划分成 $\Gamma+1$ 个互不相交的子串,$\Gamma$ 与定义 1 中的 $\Gamma$ 是同一个编辑距离阈值,且每个子串的长度大于或等于 1。例如字符串 $s$ = “vankatesh”,如果 $\Gamma=3$,则将 $s$ 划分成 $\Gamma+1$ 个子串的方法将有多种,如{“va”,“nk”,“at”,“esh”}。

**引理 1** 给定字符串 $s$ 和 $r$,将 $r$ 划分成 $\Gamma+1$ 个互不相交的子串,如果 $s$ 在阈值 $\Gamma$ 内和 $r$ 相似,则 $s$ 必定包含 $r$ 的一个子串[20]。

根据引理 1,如果字符串 $s$ 不包含字符串 $r$ 的一个子串,则 $s$ 和 $r$ 一定不相似。本文将利用引理 1 来减少候选字符串的数量,进而提高相似性连接的效率。由于字符串 $r$ 的子串长度划分得越小,则子串出现在其他字符串中的概率越大,候选字符串的数量也就越多,导致相似性连接的效率下降,因此在划分字符串的时候尽量不要使其长度太短,即在划分字符串的时候应尽量使得各个子串的长度相等,所以本文使用平均长度划分的方法。

## 2 基于划分的增量式处理框架

大数据时代,面对不断产生的庞大文本数据,字符串相似性连接的批量处理方式效率低下。为了应对大数据给字符串相似性连接带来的挑战,本文提出一种基于划分的增量式处理方法 Inc-Join,它既能找出新增数据中相似的字符串对,也能识别新增数据与历史数据之间相似的字符串对。Inc-Join 每次连接计算操作都将对状态进行更新,根据新状态进行下一次的字符串相似性连接操作,避免了冗余操作的产生。当现有的连接算法不能一次性处理较大的数据集或者需要处理的数据是在线的流式数据,Inc-Join 算法会先将数据分割成多个小的数据集,并进行增量式的处理。

图 1 所示为基于划分的增量式处理框架,主要分成 3 部分。

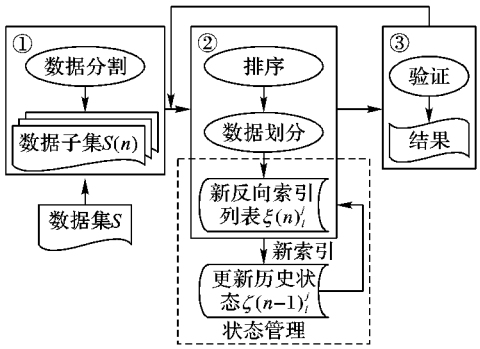


图 1 基于划分的增量式处理框架

1) 数据预处理。数据集 $S$ 被分割成多个较小的数据集 $S(n)$,见图 1 的 ①。如果 $S$ 不是很大,可以不用分割。

2) 数据筛选。数据筛选主要包括 3 个操作:对数据集 $S(n)$ 排序,将字符串划分成子串以及建立子串的反向索引列表。筛选字符串是以反向索引列表即状态为依据,所以虚线框所示的状态管理部分维持了状态的更新。状态管理要求在进行筛选之前必须先读入历史状态,并将新增数据的反向索引列表插入到历史状态中,通过这种迭代式更新状态的方式进行增量数据的处理,见图 1 的 ②。

3)数据验证。筛选出的数据并不是真正的结果,必须经过最后的验证才能判断两个字符串是否相似。每验证完一个数据集,字符串必须将更新的状态进行保存,以便下一次的增量式处理,见图 1 的③。

基于划分的增量式处理框架进行字符串相似性连接的效率比批量处理的方式高。通过减少冗余比较降低了时间的消耗,提高了相似性连接的响应速率。假设输入数据 $D_i=\{d_1,d_2,\cdots,d_i\}$,$d_i$ 是第 $i$ 个以增量方式输入的数据集,$D_i$ 是总共输入的数据集,包括 $d_1$ 到 $d_i$,则 $D_i$ 包含字符串的个数为:

$$|D_i|=\sum_{j=1}^{i}|d_j| \tag{1}$$

其中 $|d_j|$ 表示 $d_j$ 含有的字符串个数。如果任意两条字符串进行相似性连接操作花费的平均时间是 $t$,则以批量的方式处理数据 $D_i$ 花费的时间为:

$$B_i=|D_i|\times|D_i|\times t=\left(\sum_{j=1}^{i}|d_j|\right)\times\left(\sum_{j=1}^{i}|d_j|\right)\times t \tag{2}$$

以增量式方法处理数据 $D_i$ 花费的时间为:

$$I_i=|d_i|\times\left(\sum_{j=1}^{i}|d_j|\right)\times t+T(i) \tag{3}$$

其中 $T(i)$ 是指使用数据 $d_i$ 生成的反向索引列表对历史状态进行更新所花费的时间,当 $D_{i-1}$ 中的字符串长度都不相等时,$T(i)$ 的时间复杂度最大为 $O\left(\left(\sum_{j=1}^{i-1}|d_j|\right)\cdot|d_i|\right)$,远小于 $O\left(\left(\sum_{j=1}^{i}|d_j|\right)\cdot\left(\sum_{j=1}^{i-1}|d_j|\right)\right)$。从式(2)和(3)可以看出 $B_i$ 的时间复杂度大于 $I_i$,并且随着 $i$ 的增大 $B_i$ 会远远大于 $I_i$。

## 3　字符串相似性连接算法

### 3.1　增量式字符串连接

Algorithm 1 是字符串相似性连接算法,实现了字符串的增量式处理,相比批量式处理它能高效地计算出字符串相似性连接的最后结果。

Algorithm 1: Inc-Join($S$, $\Gamma$, $\zeta(n)$)。

```
Input: S: A collection of strings
       Γ: A given edit-distance threshold
       ζ(n): the historical inverted indexes
Output: λ = {(s ∈ S(n), r ∈ S(n)) ∥ (s ∈ S(n), r ∈ R(n-1))
        | ED(s,r) ≤ Γ}
begin
1)    split(S,size);
2)    foreach(S(n) ∈ S) do
        //Sort S(n) first by string length and second in
        //alphabetical order
3)      sort(S(n));
        //Partition each s∈S(n) and create inverted indexes
        //and 1 ≤ j ≤ Γ + 1
4)      partition(s∈S(n));
        //insert ξ(n) into ζ(n-1)
5)      insert(ξ(n), ζ(n-1));
6)      foreach(s∈S(n))
          //1 ≤ j ≤ Γ + 1 and *
7)        foreach(ζ(n)_l^j)
8)          verification(s, Γ, ζ(n)_l^j);
        //Save the updated inverted state
9)      save(ζ(n));
end
```

该算法先经过 1)将数据分割再通过 3)、4)对每一个 $S(n)$ 执行数据的排序和划分操作,并建立反向索引列表,具体规则已在本文第 1 部分描述。然后执行状态更新操作 5),这是一个很重要的步骤。在进行第一次相似性连接时,由于历史状态 $\zeta(n-1)$ 为空,所以直接写入新的状态。当新增数据到来,Inc-Join 算法首先要做的是将新增数据生成的反向索引 $\xi(n)$ 插入到 $\zeta(n-1)$ 中,插入的规则是将长度相同的字符串生成的反向索引合并。如果新增数据集中存在长度为 $t$ 的字符串且历史数据中不存在,则将反向索引 $\xi(n)_t^i$ 直接加入到 $\zeta(n-1)_t^i$ 中。再依次遍历新增数据集 $S(n)$ 的每一条字符串,并在更新的状态中选择长度在一定范围内的字符串 $r$ 产生的反向索引,通过反向索引进行字符串的筛选。

为保证 Algorithm 1 正确、高效地执行,在此作两点说明:

1)当 $\zeta(n-1)$ 为空,$*$ 表示 $|s|-\Gamma\leqslant l\leqslant|s|$,当 $\zeta(n-1)$ 不为空,$*$ 表示 $|s|-\Gamma\leqslant l\leqslant|s|+\Gamma$,这样是为了防止小范围内的冗余比较和错漏比较。如果两条字符串的长度相差大于 $\Gamma$,则编辑距离一定大于 $\Gamma$,即一定不相似,所以范围的规定保证了算法的正确性。

2)算法最后必须保存更新的状态,以便后续增量处理。

由于索引结构包括子串以及子串所对应的索引,该算法的执行需要使用所有字符串生成的索引列表且一个字符串被分成 $\Gamma+1$ 个子串。如果以一个整数编码一个子串或字符串,则子串的空间复杂度是 $O((\Gamma+1)\cdot|S|)$,其中 $|S|$ 为数据集 $S$ 中字符串的总个数。因为每一个子串对应一个索引列表,所以子串对应的所有索引列表的空间复杂度是 $O(\max((\Gamma+1)\cdot|S|\cdot|\zeta(n)_l^j|))$,其中 $|\zeta(n)_l^j|$ 是一个子串对应的索引长度即包含此子串的字符串个数。设 $v$ 是 Function 2 的时间复杂度,则该算法的时间复杂度是 $O\left(\max\left(\sum_{str\in S(n)}\sum_{l=|str|-\Gamma}^{|str|+\Gamma}\sum_{j=1}^{\Gamma+1}(|S|/size)\cdot|\zeta(n)_l^j|\cdot v\right)\right)$。

### 3.2　数据预处理

Function 1 的输入参数包括数据集 $S$ 和阈值 $size$,$size$ 是数据子集的大小,输出结果是 $N$ 个数据子集。

该函数的功能是对输入数据 $S$ 进行预处理,当处理大数据集或者流式数据时,先将数据分割成多个小的数据子集。数据子集的大小 $size$ 满足 $size\leqslant T_{\max}$,$T_{\max}$ 是运行设备能一次性处理的最大数据集的大小。

数据预处理是一个可选操作,当 $S$ 的大小未超过运行设备的处理能力时该函数可以不被 Algorithm 1 调用。由于 Function 1 需循环 $N$ 次将 $S$ 进行分割,所以其空间复杂度是 $O(|S(n)|)$,时间复杂度是 $O(N)$。

Function 1: split($S$,$size$)。

```
Input: S: A collection of strings
       size: The size of S(n)
Output: R = { ⋃_{n=1}^{N} S(n) | ⋃_{n=1}^{N} S(n) = S, S(n) ∩ S(m) = ∅,
```

$N = \lceil |S|/size \rceil\}$

begin:
1) while ($n \leqslant N$)
2) $R \leftarrow S(n)$
3) $n++$;
end

### 3.3 数据验证

Function 2 的输入包括当前字符串 $s$,编辑距离阈值 $\Gamma$ 和长度是 $l$ 的字符串第 $j$ 个子串生成的索引列表且 $1 \leqslant j \leqslant \Gamma + 1$,输出结果是两个字符串$\langle s,r\rangle$ 是否相似。

计算两个字符串真正的相似度是一个比较耗时的过程,Function 2 之前的操作是过滤掉明显不相似的字符串并筛选出可能相似的字符串。该函数的作用是对筛选的可能相似的字符串进行最后的验证。

该函数根据反向索引列表、当前字符串 $s$ 及引理 1 对字符串进行验证,若 $s$ 包含反向索引中的子串 $w$,则 $s$ 和 $r$ 可能相似,并且如果计算出 $s$ 与 $r$ 的实际编辑距离在阈值 $\Gamma$ 的范围内则确定字符串$\langle s,r\rangle$ 相似。

$|w|$ 是第 $j$ 个位置子串的个数,$|\zeta(n)_l^j(w)|$ 是子串 $w$ 对应的索引的长度,因为子串数量决定其对应索引数量,所以 Function 2 的空间复杂度和时间复杂度分别是 $O(\max(|w| \cdot |\zeta(n)_l^j(w)|))$ 和 $O(\max(|w| \cdot |\zeta(n)_l^j(w)| \cdot |s| \cdot |r|))$。

如何更快速、有效地验证 $s$ 和 $r$ 是否真正相似也是字符串相似性连接的关键步骤,这将在以后的研究中进一步探索。

Function 2: verification($s$, $\Gamma$,$\zeta(n)_l^j$)。

Input: $s$: The current string
$\zeta(n)_l^j$: Inverted index of strings with length of $l$
$\Gamma$: A given edit-distance threshold
Output: $\lambda = \{(s \in S(n), r \in S(n)) \| (s \in S(n), r \in R(n-1))\}$
begin:
1) foreach($w \in \zeta(n)_l^j$) do
2) foreach($r \in \zeta(n)_l^j(w)$ && $ED(s,r) \leqslant \Gamma$ && $r \notin S(n)$)
3) if ($ED(s,r) \leqslant \Gamma$)
4) $\lambda \leftarrow \langle s,r\rangle$
end

## 4 索引优化策略

### 4.1 问题描述

Inc-Join 方法虽然能有效地进行字符串相似性连接,但是在执行过程中该方法需要保存不断更新的状态。这在实际应用中需要花费大量存储空间,特别是在大数据时代,维持这种方法执行的代价会随着时间的推移而增大。

所以本文针对 Inc-Join 方法的索引部分作了进一步的优化,采用动态地建立索引的方式极大地减少了算法运行过程中所需要的存储,其中也包括内存的消耗。采用动态建立索引方法后,内存中仅保存在一定范围内的索引列表,这不但降低了内存的消耗,提高了运算速率而且算法执行过程中不需要耗费额外的存储空间。

### 4.2 优化方法

动态建立索引是指不一次性地划分所有字符串并建立好反向索引,而是在遍历字符串的时候,对于当前字符串 $s$ 只划分长度在一定范围内的字符串,并根据这些字符串建立反向索引。Algorithm 2 是优化后的相似性连接方法,此方法将动态建立的索引列表作为状态,以状态为依据进行相似性连接,索引列表的动态生成就是状态的动态更新。

与 Algorithm 1 相比,该算法最大的特点就是划分字符串并建立反向索引只针对部分字符串。为了减少内存的消耗,在划分字符串之前先删除内存中的部分索引。如果当前处理的字符串是 $str_1$,$str_0$ 是 $str_1$ 之前处理的字符串,划分字符串之前先删除由字符串 $s_i$ 生成的反向索引列表,且 $|s_i| \in [|str_0| - \Gamma, |str_1| - \Gamma]$。对 $str_1$ 只需要划分字符串 $s_j$,且 $|s_j| \in U - T, U = [|str_1| - \Gamma, |str_1| + \Gamma], T = [|str_1| - T, |str_0| + T]$,并对对 $s_j$ 的子串建立反向索引列表。这样可以在一定范围内防止同一个字符串重复划分。

Algorithm 2: Inc-Join($S$,$\Gamma$)。

Input: $S$: A collection of strings
$\Gamma$: A given edit-distance threshold
Output: $\lambda = \{(s \in S(n), r \in S(n)) \| (s \in S(n), r \in R(n-1)) | ED(s,r) \leqslant \Gamma\}$
begin:
1) split($S$,$size$);
2) foreach($S(n) \in S$) do
3) $\zeta(n) \leftarrow$ null;
4) $temp \leftarrow S(n)$;
5) sort($temp$);
6) foreach($s \in S(n)$)
7) remove($\zeta(n)_l^j$);
8) partition($str \in temp$);
9) foreach($\zeta(n)_l^j$)
10) verification($s$,$\Gamma$, $\zeta(n)_l^j$);
end

Algorithm 2 在 Algorithm 1 的基础上最大的改进是索引的建立方式,所以 Algorithm 2 的时间复杂度与 Algorithm 1 相同,但索引结构的空间复杂度得到了极大的改善,由于 Algorithm 2 动态地建立索引,不需要保存全部的索引信息,避免了所有状态读入内存,所以 Algorithm 2 索引结构中子串的空间复杂度是 $O\left(\max\left(\sum_{l=j-\Gamma}^{j+\Gamma}(\Gamma+1) \cdot |S(n)_l|\right)\right)$,其中 $j$ 是任一字符串的长度,且子串对应索引的空间复杂度是 $O\left(\max\left(\sum_{l=j-\Gamma}^{j+\Gamma}(\Gamma+1) \cdot |S(n)_l|\right) \cdot |\zeta(n)_l^j|\right)$,其中 $|\zeta(n)_l^j|$ 是一个子串下索引的长度。

## 5 实验结果与分析

### 5.1 实验条件

实验平台配置:Intel CoreTM i7-4770CPU 3.4 GHz × 8,15.6 GB内存,976 GB 硬盘,64 位 Ubuntu 12.04 操作系统,实验程序用 Java 编写,编译软件为 Eclipse。实验使用了 3 个真实数据集:DBLP Author(http://www.informatik.uni-trier.de/~ley/db)、DBLP Author + Title 和 AOL Query Log(http://www.gregsadetsky.com/aol-data/)。表 1 是 3 种数据的详细信息。由于 AOL Query Log 包含少于 500 000 条的数据,所以算法执行时从 3 个数据集分别随机抽取 400 000 条字符串作为增量的总数据集,即每次输入 80 000 条字符串,分 5 次输入。

表 1　数据集详细信息

| 数据集 | 数量 | 平均长度 | 最大长度 | 最小长度 |
|---|---|---|---|---|
| Author | 612 781 | 14.3 | 46 | 6 |
| Query Log | 464 189 | 44.8 | 522 | 30 |
| Author + Title | 863 073 | 105.8 | 866 | 21 |

### 5.2　实验结果

本文实现了 3 种字符串相似性连接算法:批量式字符串连接算法(Batch-Join)、Inc-Join 和采用动态索引技术的 Inc-Join(Dynamic Inc-Join, DInc-Join)。实验针对不同阈值 $\Gamma$,在输入 3 种不同数据的情况下统计每种方法处理相同数据的时间,并描绘出相应曲线。由于 3 个数据集字符串平均长度相差较大,字符串的阈值 $\Gamma$ 太小或者太大会严重影响相似性连接的结果,所以根据文献[1],处理 DBLP Author、AOL Query Log 和 DBLP Author + Title 的阈值范围分别设置为[1,4]、[4,7]和[7,10]内的整数。

图 2(包括后续图表)中横轴表示输入数据的总量,每次新增数据量为 80 000,纵轴为算法执行时间,并以算法执行时间来衡量相似性连接的响应速率。从图 2 中可以看出当输入数据是 DBLP Author 时,3 种方法在不同阈值下的运行时间都随着数据的增加而增多。最明显的特征是当数据量一定,Batch-Join 方法消耗的时间远远大于其他两种方法。算法首次执行时 3 种方法的执行时间大致相同,因为此时状态为空,3 种方法都类似于批处理方式。在阈值不同的 4 种情况下,Batch-Join 的执行时间都是随着数据的增大呈现指数增长的趋势,而 Inc-Join 的执行时间变化比较缓慢,呈现线性增长。因为 DBLP Author 在 3 个数据集中字符串的平均长度最小为 6,证明与 Batch-Join 相比 Inc-Join 处理短字符串时具有很高的效率。

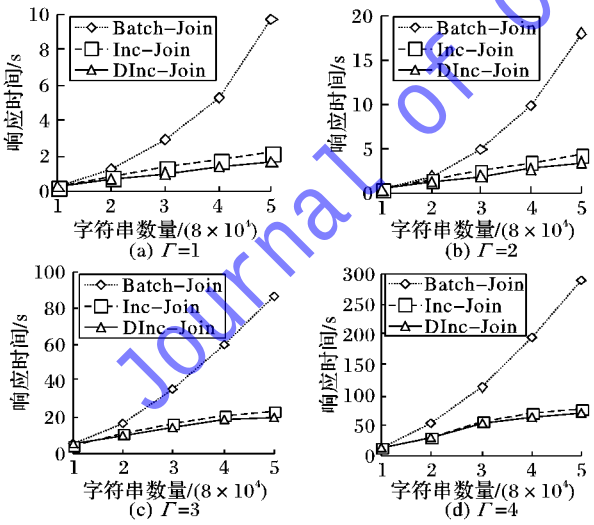


图 2　3 种方法处理 DBLP Author 数据集的时间对比

当采用动态划分建立索引时,DInc-Join 既节省了空间开销,也提高了相似性连接响应速率。因为动态索引技术使得状态不再作为输入参数,也不需要保存,既节省了读写文件的时间,又降低了内存的消耗。在处理大数据时,文件的读写是一个比较耗时的操作。图 2 中这种优势表现得不是很明显,因为 DBLP Author 数据集包含的是短字符串,生成的索引文件较小,实验平台内存为 15.6 GB,导致状态的读写时间短且内存的占用率低,但从图 2 可以看出动态索引方法节省时间的趋势会越来越明显。

当输入数据是 AOL Query Log 时,图 3 中阈值的增大也会导致执行时间的增多,因为阈值增大候选集的数量就会增大,验证需要的时间就增多,且 Batch-Join 的执行时间也远远超过 Inc-Join 的执行时间,同样 DInc-Join 的动态索引技术使得它的相似性连接效率在 Inc-Join 的基础上再一次提高。

图 4 中处理的是 Author + Title 数据集,其中 DInc-Join 的连接效率提高得较为明显。因为 Author + Title 是长字符串集,反向索引包含字符串的子串,及子串对应的反向索引列表。每一个字符串分成 $\Gamma+1$ 个子串,$\Gamma$ 越大,子串对应索引列表的数量越多。所以 DInc-Join 避免了大规模状态的读写及内存的占用,提高了相似性连接的效率。证明与 Batch-Join 相比 Inc-Join 对长字符串进行相似性连接也有很高的效率。

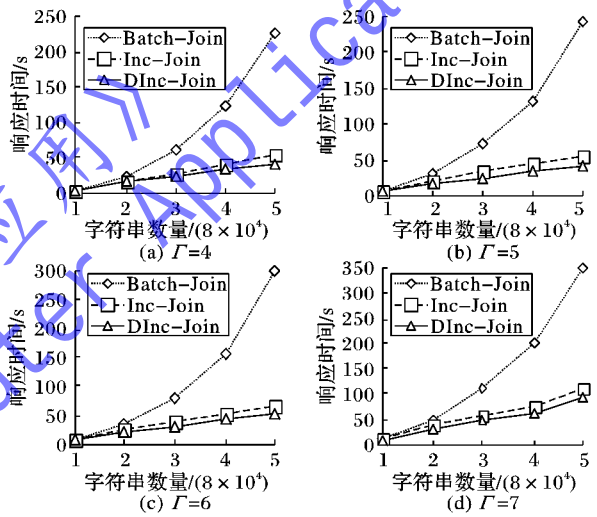


图 3　3 种方法处理 AOL Query Log 数据集的时间对比

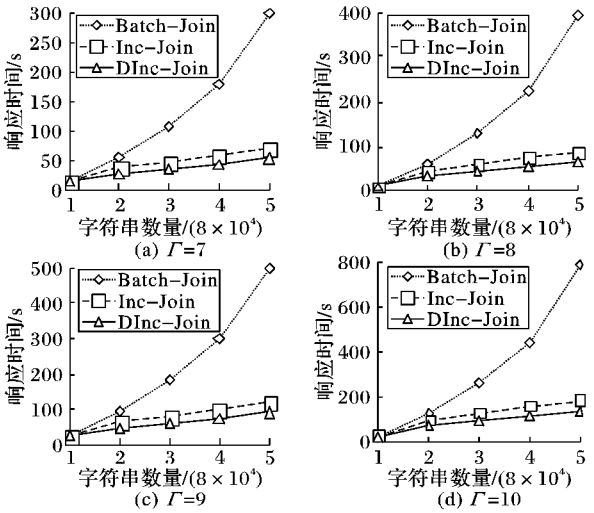


图 4　3 种方法处理 DBLP Author + Title 数据集的时间对比

实验表明 Inc-Join 的增量式处理方法适用于任何长度的字符串,极大地提高了相似性连接的效率。冗余比较的减少使得方法的执行时间呈线性增长,相似性连接的响应时间得到了改善,而且阈值 $\Gamma$ 越大,Inc-Join 的优越性越明显,但是 Inc-Join 能增量式地高效处理流式数据并不表明任何数据以增量式方式处理的效率一定高于批量处理方式。当需处理的

数据集能一次性获得,且数据量未超过设备的处理能力则批处理方式的效率高于增量式处理。

## 6 结语

本文提出了一种面向流式数据的增量式字符串相似性连接方法——Inc-Join,它以状态为依据进行相似性连接,并不断更新历史状态。通过使用 3 个真实数据集进行测试,证明 Inc-Join 对长、短字符串都有很高的效率,避免了批处理方式时间的指数增长,并对 Inc-Join 的索引技术进行了优化,采用动态索引技术降低了算法运行的消耗,减少了大量读写文件的时间,进一步提高了 Inc-Join 的执行效率。字符串相似性连接是一个很耗时的操作,面对流式数据应有更高效的连接方法来提高字符串相似性连接的效率。Inc-Join 虽然能高效地进行相似性连接,但是它的效率仍然可以再次提高。未来工作的重点是探索出一种高效的验证方法以及使用内存并行处理技术,并将其运用在 Inc-Join 方法中。

**参考文献:**


[1] LI G, DENG D, WANG J, et al. Pass-Join: a partition-based method for similarity joins [J]. Proceedings of the VLDB endowment, 2011, 5(3): 253 – 264.

[2] JIANG Y, DEND D, WANG J, et al. Efficient parallel partition-based algorithms for similarity search and join with edit distance constraints [C]// Proceedings of the Joint EDBT/ICDT 2013 Workshops. New York: ACM, 2013: 341 – 348.

[3] 荣垂田,徐天任,杜小勇. 基于划分的集合相似连接[J]. 计算机研究与发展, 2012, 49(10): 2066 – 2076. (RONG C T, XU T R, DU X Y. Partition-based set similarity join [J]. Journal of computer research and development, 2012, 49(10): 2066 – 2076.)

[4] 曹海,骆吉洲,陈懿诚. 一种基于数据划分的字符串相似性连接外存算法[J]. 智能计算机与应用, 2012, 2(5): 31 – 34. (CAO H, LUO J Z, CHEN Y C. A data-partition based disk algorithm for string join [J]. Intelligent computer and applications, 2012, 2(5): 31 – 34.)

[5] LU J, LIN C, WANG W, et al. String similarity measures and joins with synonyms [C]// Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. New York: ACM, 2013: 373 – 384.

[6] ARASU A, GANTI V, KAUSHIK R. Efficient exact set-similarity joins [C]// Proceedings of the 32nd International Conference on Very Large Data Bases. [S.l.]: VLDB Endowment, 2006: 918 – 929.

[7] XIAO C, WANG W, LIN X. Ed-Join: an efficient algorithm for similarity joins with edit distance constraints [J]. Proceedings of the VLDB endowment, 2008, 1(1): 933 – 944.

[8] WANG J, FENG J, LI G. Trie-Join: efficient trie-based string similarity joins with edit-distance constraints [J]. Proceedings of the VLDB endowment, 2010, 3(1/2): 1219 – 1230.

[9] METWALLY A, FALOUTSOS C. V-SMART-Join: a scalable MapReduce framework for all-pair similarity joins of multisets and vectors [J]. Proceedings of the VLDB endowment, 2012, 5(8): 704 – 715.

[10] DONG X, SRIVASTAVA D. Big data integration [C]// ICDE 2013: Proceedings of the 2013 IEEE 29th International Conference on Data Engineering. Piscataway, NJ: IEEE, 2013: 1245 – 1248.

[11] CHRISTEN P. A survey of indexing techniques for scalable record linkage and deduplication [J]. IEEE transactions on knowledge and data engineering, 2012, 24(9): 1537 – 1555.

[12] CHEN Q, HSU M. Continuous MapReduce for In-DB stream analytics [C]// OTM 2010: Proceedings of the 2010 International Conference on the Move to Meaningful Internet Systems. Berlin: Springer, 2010: 16 – 34.

[13] YAN C, YANG X, YU Z, et al. IncMR: incremental data processing based on MapReduce [C]// CLOUD 2012: Proceedings of the 2012 IEEE 5th International Conference on Cloud Computing. Piscataway, NJ: IEEE, 2012: 534 – 541.

[14] LOGOTHETIS D, YOCUM K. Ad-Hoc data processing in the cloud [J]. Proceedings of the VLDB endowment, 2008, 1(2): 1472 – 1475.

[15] DE FRANCISCI MORALES G, GIONIS A, SOZIO M. Social content matching in MapReduce [J]. Proceedings of the VLDB endowment, 2011, 4(7): 460 – 469.

[16] THUSOO A, SARMA J S, JAIN N, et al. Hive—a petabyte scale data warehouse using Hadoop [C]// ICDE 2010: Proceedings of the 2010 IEEE 26th International Conference on Data Engineering. Piscataway, NJ: IEEE, 2010: 996 – 1005.

[17] PEND D, DABEK F. Large-scale incremental processing using distributed transactions and notifications [C]// OSDI'10: Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation. Berkeley, CA: USENIX Association, 2010: 1 – 15.

[18] XIAO C, WANG W, LIN X, et al. Efficient similarity joins for near-duplicate detection [J]. ACM transactions on database systems, 2011, 36(3): 15.

[19] HE Q, DU C, WANG Q, et al. A parallel incremental extreme SVM classifier [J]. Neurocomputing, 2011, 74(16): 2532 – 2540.

[20] 李璐, 王宏志, 李建中, 等. Ed-Sjoin: 一种优化的字符串相似性连接算法[J]. 计算机研究与发展, 2009, 46(z2): 319 – 325. (LI L, WANG H Z, LI J Z, et al. Ed-Sjoin: an optimal algorithm for similarity joins with edit distance constraints [J]. Journal of computer research and development, 2009, 46(z2): 319 – 325.)



**Background**

This work is partially supported by the National Natural Science Foundation of China (61402100), the Fundamental Research Funds for the Central Universities (2232013D3-15).



**YAN Cairong**, born in 1978, Ph. D., associate professor. Her research interests include parallel computing, distributed computation, big data processing.

**ZHU Bin**, born in 1990, M. S. candidate. His research interests include parallel computing, distributed computation, big data processing.

**WANG Jian**, born in 1989, M. S. candidate. His research interests include parallel computing, distributed computation, big data processing.

**HUANG Yongfeng**, born in 1971, Ph. D., associate professor. His research interests include data mining, machine learning, image processing.