软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2015,26(6):1421−1437 [doi: 10.13328/j.cnki.jos.004610]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于实体的相似性连接算法*

刘雪莉, 王宏志, 李建中, 高 宏

(哈尔滨工业大学 计算机科学与技术学院,黑龙江 哈尔滨 150001)

通讯作者: 刘雪莉, E-mail: xueli.hit@gmail.com

摘 要: 按照元组描述的实体对其进行组织和查询处理,是一种管理劣质数据的有效方法.考虑到同一个实体的同一属性存在多个描述的值,因此,基于实体的数据库上的连接是支持多个值的相似性连接.与字符串的相似性连接相比较,实体的相似性连接在数据清洗、信息集成、模糊关键字查询、诈骗检测和文本聚集等领域有着更好的应用效果.通过建立双层索引结构,提出了实体数据库上相似性连接算法ES-JOIN.同时,该方法适用于解决集合中字符串模糊匹配的相似性连接问题,而传统的集合相似性连接只针对集合中元素精确匹配的情况.为了加速连接,还提出了过滤措施对算法进行优化,进一步给出了优化算法 OPT_ES-JOIN.实验验证了 ES-JOIN 算法和 OPT_ES-JOIN 算法具有很好的效率和可扩展性.实验结果表明,过滤措施具有很好的过滤效果.

关键词: 实体;相似性连接;劣质数据

中图法分类号: TP311

中文引用格式: 刘雪莉,王宏志,李建中,高宏.基于实体的相似性连接算法.软件学报,2015,26(6):1421−1437. http://www.jos.org.cn/1000-9825/4610.htm

英文引用格式: Liu XL, Wang HZ, Li JZ, Gao H. Similarity join algorithm based on entity. Ruan Jian Xue Bao/Journal of Software, 2015,26(6):1421−1437 (in Chinese). http://www.jos.org.cn/1000-9825/4610.htm

Similarity Join Algorithm Based on Entity

LIU Xue-Li, WANG Hong-Zhi, LI Jian-Zhong, GAO Hong

(School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China)

Abstract: Taking entity as the basic unit to organize the tuples in query processing is an effective way of managing low-quality data. As many descriptions of attribute value in an entity, join operator must support similarity join over multiple values. Entity similarity join is more effective than traditional similarity join in data cleaning, information integration, fuzzy keyword search, fraud detection, and text aggregation. In this paper, an entity similarity join algorithm, ES-JOIN, is designed by adopting the structure of the double layer prefix index. The presented method is suitable for solving set similarity join problem based on fuzzy elements matching, and thus is a better choice than the traditional set similarity join which only considers exact element match. In order to accelerate the join process, a new filtering measures is proposed to optimize the algorithm, and an optimization algorithm, OPT_ES-join, is also obtained. Experiments demonstrate that the ES-JOIN algorithm has good efficiency and scalability, and the filter measures is very effective.

Key words: entity; similarity join; poor-quality data

近年来,随着信息生产能力的提高和信息采集技术的进步,数据的质量问题引起了人们的重视.在许多现实应用中,例如在经济、电信、物流、金融、农业等领域中,数据的不一致、不完备、过时、错误、冗余、不精确等问题普遍存在,我们称这些数据为劣质数据.将劣质数据存储在传统的关系数据库中进行查询操作从而进行信息决策,将会造成重大的损失.而对劣质数据的处理,已经引起了学术界和工业界的广泛关注.现在,许多大

* 基金项目: 国家自然科学基金(61003046, 6111113089, 61033015, 60831160525, 61173022); 国家重点基础研究发展计划(973)(2012CB316200, 2012CB316202); 国家高技术研究发展计划(863)(2012AA011004); 海量图数据上实体识别(KF2011003)

收稿时间: 2012-11-25; 修改时间: 2013-03-07, 2013-10-30; 定稿时间: 2014-03-28

型公司大都使用两种方法处理劣质数据:

- 一是采用数据清洗的方法.虽然数据清洗在很多情况下可以解决数据质量问题,然而,其在一些应用中存在着局限性:第一,有些劣质数据(例如传感器数据、RFID 数据等)是难以被清洗干净的;第二,对劣质数据的清洗可能会造成信息的丢失(例如,在数据清洗的过程中,可能把正确的信息误认为劣质的处理掉);第三,对这些劣质数据进行清洗(例如数据整合)代价非常大;最后,在信息更新频率非常高的系统中,需要频繁地执行数据清洗过程,这将极大地降低系统效率;
- 二是采用最简单的方法:将劣质数据丢掉[3].显然,在劣质数据量比较大的情况下,将会造成信息的严重缺失.

从上述分析中可以看出:对劣质数据进行数据清洗或是直接丢掉劣质数据,并不能彻底有效地解决劣质数据带来的数据质量问题.然而,现实应用要求必须处理这些劣质数据.因此,需要寻找一种有效的方法处理劣质数据来保证查询结果的准确性和可靠性.很多情况下,应用能够容忍一定程度的劣质数据.查询处理的目标是,从包含劣质数据的数据库中查询得到满足一定清洁度的结果.这样,如何组织和管理劣质数据,是亟待解决的问题.

随着数据清洗和实体识别技术的成熟和完善,我们发现:将实体识别出的元组组织起来,组成新的数据类型,能够很好地描述各种劣质信息.下面举例介绍实体数据模型.见表 1,其中的每条元组表示一个实体数据.实体数据的属性值具有多值的特征,每个值都能够描述这个实体,值的概率不同于概率数据中的概率,不再表示值的存在性信息,而是表示属性值的质量度.质量度是衡量数据质量的指标,它可以是属性值在数据中出现的频率,也可以通过领域知识等其他的方法计算或是由专家指定.例如,表 1 的第 1 条实体数据,Name 属性中,Catherine,Kate,Kitty 都能够同时表示一个人,只是在 40%的情况下都用 Catherine 表示.

Table 1 A simple entity table

表 1 一个简单的实体表

Id	Name	City
1	{Catherine,0.4;Kate,0.1;Kitty,0.5}	{NewYork,0.8;NY,0.2}
2	{Alexander,0.7;Ada,0.3}	{LasVegas,0.8;LV,0.2}
3	{Charles,0.8;Chunk,0.2}	{Chicago,0.8;CHI,0.2}

实体数据模型形式上与概率模型相似,但是本质上不同于已有的概率数据模型.在概率模型中,各元组的任意合法组合均可构成一个可能世界实例,实例的概率值表示该实例的存在概率.从数据的角度看,概率数据只能够描述数据的不确定信息,这只是劣质数据的一种类型.而实体数据模型能够表示各种类型的劣质信息:对于不一致信息,如果关系数据库中有两条元组代表同一个实体,实体数据类型综合这两条元组,消除了不一致性;对于不确定信息,实体数据模型能像概率模型那样表示不确定信息;对于不完整信息,实体数据在组织实体识别的结果时,能够补充一些元组的数据缺失.其他的一些类型也能通过相似的处理进行描述.从数据操作上看,概率数据库上的查询操作需考虑所有可能世界,返回所有可能的结果,这将导致结果规模的指数增长,降低了查询处理的效率.而现有的一些解决方法主要是排序、剪枝等启发式策略,并不能从根本上解决数据规模的膨胀问题.此外,查询时并没有考虑到操作过程对结果准确度的影响,整个查询过程都是精确的查询,最后返回这个查询结果存在的概率,并没有考虑到查询中数据的质量问题.并且,假设用户给定一个质量度的阈值,概率模型并不能据此阈值进行过滤,得到用户满意的结果.最后,概率数据库的更新操作将会造成可能世界实例的数据库大小倍数的增长,这些可能世界实例的概率也需要重新计算.而对实体进行查询处理,返回的是一条实体元组,有效地解决了直接处理时信息的质量问题,使得查询结果更准确.例如在表 1 中,假设一个查询要求返回 Catherine 的电话号码,如果在传统的数据库中,只存储了名字为 Kate 的个人信息,将返回空的查询结果;但在表 1 所在的实体数据库中,查询结果能够被正确返回.此外,在更新实体表时,我们只需更新单个实体数据.

考虑到实体数据模型的特点,每个属性值均可以由多个值描述,且每个值都有对应的质量度,现有的查询方法不再适用.本文考虑基于实体数据模型的相似性连接操作.给定两个实体表 R、S,连接属性 A、相似度阈值 τ,基于实体的相似性连接返回在属性 A 上满足相似度阈值的实体对.由于实体属性值是所有能够描述实体属性

值的多个值的集合,因此在相似性连接的应用领域,比如数据清洗[1]、信息集成[2]、模糊关键字查询[3]、诈骗检测[4]和文本聚集[5]等,实体的相似性连接也同样适用.此外,基于实体的相似性连接解决了现有的相似性连接不能解决的问题:由于信息的不一致表述,在现有的关系数据模型层次上进行查询操作并不能保证结果的正确性.例如在信用卡欺诈检测中,名字为 Robert 和 Bob 的人同时在两个不同的地方进行信用卡消费,在传统的关系数据库中进行相似性查询时,我们并不能断定 Robert 和 Bob 是信用卡欺诈用户,事实上,Bob 是 Robert 的别名,它们代表的是同一个人.显然,若{Bob,Robert}是一个实体属性,则基于此实体做相似性连接能够很好地解决这个问题.类似的,在其他广泛应用相似性连接的领域里,基于实体层次的相似性连接使得返回的结果更加全面和准确,查询的信息也更加地完整.所以,对实体进行相似性连接的研究具有更大的价值、更广泛的应用前景.

然而,现有的相似性连接算法并不能高效解决实体关系数据库上的相似性连接问题.基于字符串的相似性,只能返回满足相似度阈值的字符串对,基于实体的相似性连接需要返回满足相似性阈值的实体对的集合,当实体属性值是字符串集合时,采用基于字符串的相似性连接方法时,需要在计算出满足相似度的字符串对的基础上,按实体对字符串聚合,重新计算实体的相似性.由于实体中属性的每个值都具有清洁度,即使字符串的相似性很大,对整个实体的相似度贡献也不一定很大.这样就造成了大量的冗余计算,在实体关系表很大时,此方法效率过低.因此,基于字符串相似性的近似连接算法在效率上不适用于解决本文的问题.基于集合的相似性连接需考虑集合内部元素的精确匹配.而基于实体的相似性连接考虑的是集合中元素的相似匹配,即模糊匹配,同时还需要考虑集合中元素质量度对实体相似性的影响.

针对这个问题,本文提出了 ES-JOIN 算法,该算法是解决集合相似性连接的一个新方法,适用于基于实体的相似性连接.传统的基于集合的相似性连接都认为:只有当集合中两元素完全匹配时,两个集合的相似性才有所增加.显然,在很多应用中,若集合中两个元素相似,则这两个集合也是很相似的.比如说,两个集合{Microsoft, IBM,Intel}和{Microsft,IBM,intel},若给定 Jaccard 相似性阈值为 0.8.则依据传统的做法,这两个集合不相似;而事实上,这两个集合表示同一信息.ES-JOIN 算法则认为集合中 Microsoft 和 Microsft,Intel 和 intel 是近似匹配的,返回正确的连接结果.

本文主要有如下贡献:

(1) 形式化定义了实体模型,定义了在实体数据库中基于实体的相似性连接问题;

(2) 基于通用的字符串相似性连接的 filter-and-refine 框架,提出了新的索引结构和新的过滤措施,给出了基于实体的相似性连接算法 ES-JOIN 算法,并对算法进行了优化;

(3) 用实验验证了 ES-JOIN 及其优化算法具有很好的效率和扩展性,过滤措施也有很好的过滤效果.

本文第 1 节形式化定义实体相似性连接并给出预备知识.第 2 节描述基于实体的相似性连接算法 ES-JOIN 算法及其优化算法 OPT_ES-JOIN.第 3 节给出优化的基于实体的相似性连接算法 OPT_ES-JOIN 算法.第 4 节通过实验说明 ES-JOIN 算法和 OPT_ES-JOIN 算法的性能和过滤措施的过滤效果.第 5 节综述相关工作.第 6 节总结全文.

1 预备知识及相关定义

由于实体属性值可以看作字符串集合,因此,字符串相似性连接的一些基本方法也可以作为实体相似性连接的一部分来使用.

1.1 预备知识简介

现有的字符串相似性连接操作的一个基本框架是 filter-and-refine 框架.它的基本做法是:在 filter 步,为每个字符串生成 signature,使用 signature 以及过滤措施产生候选集.在 refine 步,确认候选集生成最终结果.其中,signature 可以有多种类型,比如 token,q-gram 等.q-gram 由于其简单、格式统一的特性,被广泛采用在相似性连接的算法中.此外,基于 q-gram 的前缀过滤在产生候选集的过程中有很好的过滤效果.

1.1.1 q-gram 简介

对于一个字符串 σ,它的 q-gram 可以通过采用长度为 q、滑动距离为 l 的窗口对其进行切分得到.每个 q-gram

包含 q 个字符,同时,q-gram 还包含位置信息.

定义 1(q-gram). 给定一个字符串 s 和正整数 q,s 的 q-gram 是一个(l,g)对,其中,g 是 s 的从第 l 个字符开始长度为 q 的子串.

例 1:字符串 *Kate* 的 2-gram 包含{(1,*ka*),(2,*at*),(3,*te*)}.

1.1.2 前缀过滤

基于前缀的过滤方法[8]的基本思想是:首先,将字符串集合中出现的 q-gram 按 q-gram 频率从小到大排列,以此作为 q-gram 的全局序;再将每个字符串的 q-gram 集合按照 q-gram 的全局序排列.设编辑距离阈值为τ,若两个字符串 s,t 相似,则 s,t 的公共 q-gram 至少为 $LB_{\sigma_1;\sigma_2}=(\max(|s|,|t|)-q+1)-q\cdot\tau$.由鸽巢原理,这两个字符串对的前 $q\cdot\tau$+1 个 q-gram 必有一个是相同的.具体如图 1 所示.

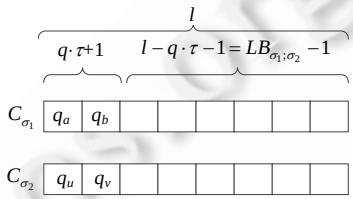

Fig.1 Basic idea of prefix filtering

图 1 前缀过滤的基本思想

图 1 中:C_{σ_1},C_{σ_2} 分别为字符串 s,t 的 q-gram 集合;l 为 q-gram 集合的大小,取字符串 s,t 对应的 q-gram 集合元素个数较大的值.如果 s,t 在前缀的 q-gram 集合前 $q\cdot\tau$+1 中没有一个相同的 q-gram,则 s,t 的公共 q-gram 不会大于 $LB_{\sigma_1;\sigma_2}-1$,则这两个字符串一定不满足编辑距离阈值,即,s,t 不相似.

1.2 实体数据模型

引言中,通过举例简单描述了实体数据模型,本节通过定义实体属性值及实体形式化定义实体数据模型.

从表 1 中可以看到:实体属性是一个字符串集合,这个集合里的所有元素有着相同的语义,每个元素都能代表这个实体所表示的对象.例如,charles 和 chunk 都能代表 3 条元组的 Name.基于上述描述,定义 2 给出了实体属性的概念.

定义 2(实体属性值). 给定字母表Σ,带有质量度的实体属性值记为:$s=\{\sigma_1,p_1;\sigma_2,p_2;\ldots;\sigma_m,p_m\}$,其中,$\sigma_i\in\Sigma^*$,$p_i\in(0,1]$.$\sigma_i$ 为 s 的第 i 种表示,它的质量度为 p_i.

在表 1 中,id,Name,City,Phone 都是一个实体属性.

不同于概率模型,字符串的概率表示该字符串存在的可能性大小,字符串之间的关系是互斥的.在实体关系模型中,字符串的概率表示实体属性值用此字符串表示的可能性大小,实体属性中,每一个字符串都能够表示实体属性,它们是可以同时存在的,也可以是互斥的,概率的不同只是说明字符串表示该实体的程度不同.事实上,概率模型是其属性值取值的一种特例.例如一本书的销售,亚马逊卖 50 元,当当卖 30 元,在价格属性上这两个值是同时存在的.也可以存在这样一种情况:当当上曾经卖 50 元,促销时卖 30 元,此时,价格属性的两个值之间的关系是互斥的.此外,概率模型基于可能世界模型,而在实体模型中不存在可能世界.

仅仅使用实体属性并不能描述一个实体,因为在数据库中每个实体必须是唯一的,因此还需要一个键值来唯一确定这个实体,这个键值可以是实体中的一个属性,例如表 1 中的 id 属性,也可以是多个属性的集合.

定义 3(实体). 一个实体 $E=(K,A)$,其中,A 是实体属性的集合,$K\subseteq A$ 是实体的键值,K 唯一确定实体 A.

表 1 给出了 3 条实体元组,每条元组由 id 唯一确定,包含 Name,City 两个实体属性,id 为 1 的元组中,Name 属性值可用 3 个值来描述.

有了实体数据模型,我们可以定义基于此模型的相关操作.

1.3 基于实体的相似性连接

本文解决了实体属性值类型为字符串时的相似性连接问题.在介绍实体相似性连接之前,我们首先介绍实体的相似性,实体的相似性可以用实体属性的相似性来描述.由于实体属性值中的每个元素都能够代表实体属性值所表示的语义,因此,若两个实体属性值中的元素相似,则这两个实体属性值具有一定的相似性.此外,实体属性中元素权重信息表示实体属性值为此元素值时的质量度,显然,质量度越大的元素越能代表这个实体属性值的真实值,质量度越大的元素对两个实体属性值的相似性贡献越大.本文解决了实体属性值类型为字符串时的相似性连接问题.基于上述两个方面,我们定义实体属性值的相似性定义.

定义 4(实体属性值相似性). 给定属性值 $s=\{(\sigma_1,p_1),(\sigma_2,p_2),\ldots,(\sigma_m,p_m)\}$,$r=\{(\delta_1,q_1),(\delta_2,q_2),\ldots,(\delta_n,q_n)\}$,$s$ 和 r 的相似性为 $\hat{d}(s,r)=\sum_{d(s_i,r_j)\leqslant\tau} p_i\cdot p_j\cdot sim(s_i,r_j)$,$\tau$为用户指定的编辑距离阈值:

$$sim(s_i,r_j)=1-\frac{d(s_i,r_j)}{\max\{|s_i|,|r_j|\}},$$

其中,$d(s_i,r_j)$是字符串 s_i,r_j 的编辑距离.

有了实体属性值相似的概念,接下来定义实体的相似性连接.

定义 5(实体属性相似连接). 给定实体关系表 R 和 T、连接的实体属性 S,R 和 T 在 S 上的相似连接操作返回所有满足以下条件的记录对(r_i,t_j):

(1) $r_i\in R,t_j\in T$;

(2) $\hat{d}(r_i.S,t_j.S)\leqslant\theta$,$\theta$为用户指定的相似性阈值.

在连接属性 S 明确的情况下,可以简称实体相似连接.

例 2:设表 1 是一个银行系统中信用卡用户的基本信息;表 2 是一个交易记录表,记录了用户刷卡购物的信息.

因为同一用户不可能同时出现在两个不同的地方消费,因此,Name 属性上连接表 1 和表 2,可以检测出信用卡欺诈用户.设定编辑距离阈值为 2,用户给定的相似性阈值为 0.1,则相似性连接返回的结果为{(1,1),(1,2),(3,3)},可得表 1 中 ID 为 1 的信用卡持有者是一个信用卡欺诈用户.显然,在传统的关系数据库中,我们并不能通过现有相似性连接算法检测出 ID 为 1 的用户是一个欺诈用户.

Table 2 Transaction record table

表 2 交易记录表

ID	Name	City	Date	Money
1	{Kate,0.8;Kitty,0.2}	Beijing	2012-3-15	2 000
2	{Katherin,1.0}	London	2012-3-15	5 000
3	{chunk,0.4;chales,0.6}	London	2012-3-15	3 000

2 ES-JOIN 算法与分析

2.1 ES-JOIN算法的框架结构

实体相似性连接的一个基本方法是采用基于字符串相似性连接方法,具体方法是:考虑实体属性值 s,r 的相似性,首先使用动态规划算法计算 s 和 r 笛卡尔积组成的所有字符串对的编辑距离值;之后,按照定义 4 计算 s 和 r 的相似性,若 s 和 r 的相似性满足定义 5,则 s 和 r 所在的实体可以连接.显然,这个方法的代价是非常高的,它需要计算连接属性所有可能值之间的编辑距离.而动态规划算法计算两个字符串之间的编辑距离的时间复杂度为 $O(n^2)$,空间复杂度为 $O(n)$.我们的目标是:设计好的过滤措施,尽可能地减少不匹配的字符串对和不匹配的实体对.

由定义 4 可知,实体属性值 s 和 r 的相似性为 $\hat{d}(s,r)=\sum_{d(s_i,r_j)\leqslant\tau} p_i\cdot p_j\cdot sim(s_i,r_j)$,此相似性值包含两个方面:编辑距离 $d(s_i,r_j)\leqslant\tau$的字符串的相似性及其质量度.存在这样一种情况:即使质量度很小的字符串对,若其编辑距

离满足编辑距离阈值时,同样对整个实体属性值的相似性有贡献,因此,本文分开处理字符串对的相似性及其质量度对实体属性值相似性的影响.首先,判断出实体对中可能满足相似性阈值的字符串对;之后,根据其质量度进行过滤;最后,将最终候选集进行确认操作,挑选出满足定义实体相似性阈值的实体对.

现有的研究表明:使用 q-gram 建立倒排索引,采取过滤措施能够很大程度上剪枝不匹配的字符串对,从而减少编辑距离的计算次数,提高算法的执行效率.

基于上述描述,本节提出的 ES-JOIN 算法采用传统的 filter-and-verify 框架,具体流程如图 2 所示.

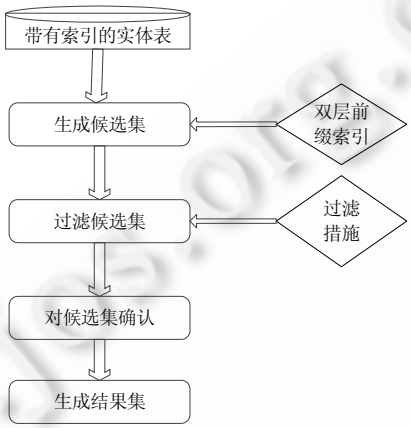

Fig.2 Framework of ES-JOIN algorithm

图 2 ES-JOIN 算法的框架

如图 2 所示,ES-JOIN 算法首先合并实体表中的双层前缀索引得到初始的候选集,第 2 步采用质量度对候选集进行过滤,最后一步对生成的候选集计算相似性,进行确认操作,返回最终结果.此过程不同于字符串相似性连接的方法,双层前缀索引不以字符串作为索引的基本单位,而以实体作为基本单位,在确认操作时不需要合并字符串对.此外,在过滤阶段过滤掉相似性很高但是质量度很低的实体对,节省了大量的冗余计算.

根据 ES-JOIN 算法的框架,算法 1 描述了算法的具体实现过程.

算法 1. ES-JOIN 算法.

输入:连接的实体表 R,S,连接的实体属性 A 上的双层前缀索引 T_r,T_s,编辑距离阈值 τ,相似性阈值 θ;

输出:满足连接条件的实体对集合 *Result*.

(1) $Result=\varnothing$

(2) 初始候选集 $Cans=Generate\_candstruct(T_r,T_s)$;

(3) 候选集 $Canf=Generate\_cand(Cans)$;

(4) $Result=verify(Canf)$

(5) 返回 *Result*.

ES-JOIN 算法采用双层前缀索引结构产生候选集:第(1)步,*Generate_candstruct* 首先合并双层前缀索引,生成可能相似的实体候选对;第(2)步,*Generate_cand* 利用质量度再次过滤第(1)步中生成的候选集,生成最终候选集;最后,第(3)步使用迭代算法进行确认操作,最终返回实体相似性连接的结果.每步的具体实现算法见第 2.2 节.

2.2 ES-JOIN算法及分析

2.2.1 双层前缀索引结构

由定义 4 可知,实体属性值的相似性可以转化为字符串之间的相似性.然而,现有的字符串相似性连接的前缀索引只能返回满足相似度条件的字符串对,并不能反映字符串所在实体对的一些相似性信息.为了解决这个问题,此节定义新的双层前缀索引框架产生实体中相似字符串对的候选.

双层前缀索引以字符串相似性连接中的前缀索引为基础.在字符串相似性连接中,前缀索引是一种倒排索引结构,它的构建方法如下:

- 首先,将要连接的字符串生成 *q*-gram;
- 其次,将所有的 *q*-gram 排序.本文按 IDF 排序,即,出现频率低的 *q*-gram 被排在序列的前边.它的直观思想是:出现频率低的 *q*-gram 倒排表比较小,经过前缀过滤后产生的候选集也相应较小[13];
- 再次,按照全局 *q*-gram 顺序将每个字符串的 *q*-gram 集合排序;
- 最后,前缀索引为出现在所有字符串前缀长度中的 *q*-gram 建立倒排表.

不同于字符串的gram集合,由于实体属性值的多值特征,实体属性值中的字符串生成的*q*-gram不再是一个二元组形式(*l*,*g*),我们用四元组(*eid*,*cid*,*l*,*g*)表示一个 *q*-gram.其中:*eid* 代表实体号,一个实体属性值的所有 *q*-gram 共享唯一的实体号;*cid* 表示 *q*-gram 是实体的第 *cid* 个字符串所成成的 *q*-gram,即,实体属性值的一个字符串的 *q*-gram 集合共享唯一的 *cid*;*l* 表示 *q*-gram 的位置信息;*g* 是 *q*-gram 的值.

不同于已有的基于字符串的前缀索引,基于实体的相似性连接在建立双层前缀索引时,将 *q*-gram 的值作为索引项,倒排表中存储 *q*-gram 所在的实体号 *eid*,第 2 层以 *eid* 为索引项,存储(*cid*,*pro*)的元组结构,其中二元组的各项和 *q*-gram 对应的各项表示相同的含义.采用双层前缀索引,能够在合并索引时反映出实体的相似性信息.

例 3:表 3 实体表中,设 τ=1,*q*=3,*value* 属性中 3-gram 的总序为(bce,bcd,abc).

Table 3 Entity table

表 3 实体表

eid	value
1	{abc,0.9;abcd,0.1}
2	{abce,0.5;bcd,0.5}

双层前缀索引如图 3 所示.

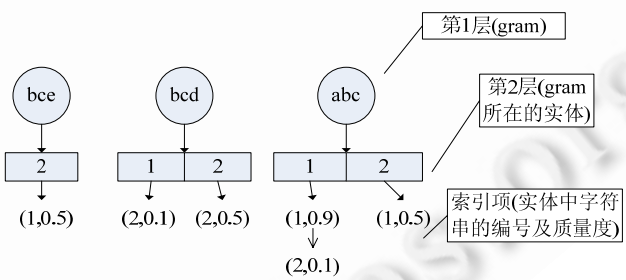

Fig.3 Bi-Layer prefix index

图 3 双层前缀索引

2.2.2 候选集生成

候选集经过两步生成:第 1 步合并前缀索引(第 2.2.2.1 节);第 2 步使用过滤措施过滤掉不满足相似性阈值的实体对,得到实体相似对的候选(第 2.2.2.2 节).

2.2.2.1 合并前缀索引算法及分析

基于实体的相似性连接产生候选集,需要合并要连接的两实体表的索引.在合并前缀之前,我们引入一个新的结构:实体候选结构.

定义 6(实体候选结构 Can_e). 实体对的相似性特征: $(eid_1,eid_2)\rightarrow\{(eid_1.cid,eid_2.cid)\,|\,(C_{eid_1.cid}\cap C_{eid_2.cid})\neq\varnothing\}$. 其中:$(eid_1,eid_2)$是一个实体 id 对,表示连接对中满足前缀过滤条件的所有实体对;*C* 为字符串的前缀 *q*-gram 集合.*Cans* 为实体候选结构集合,记 $Can_e.eid$ 为(eid_1,eid_2),$Can_e.cid$ 为满足 $(C_{eid_1.cid}\cap C_{eid_2.cid})\neq\varnothing$ 的$(eid_1.cid,eid_2.cid)$链表,$|Can_e.cid|$为链表的长度,$(eid_1.cid,eid_2.cid)$的权重为对中两个字符串权重之积.

在合并双层前缀时,首先合并 q-gram 相同的实体号,再次合并实体号中具有相同前缀的字符串,产生实体候选结构.合并之后得到的实体候选集直接以实体作为基本单位,在之后的过滤和确认操作中节省了大量的字符串相似性的重复计算.

例 4:表 1 与表 2 在 Name 属性上做相似性连接操作,设 τ=2,θ=0.1,实体候选结构 Can_e 如图 4 所示.算法 2 给出了生成初步实体候选结构的具体做法.

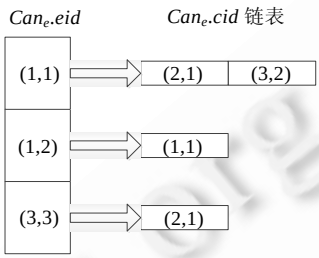

Fig.4 Entity candidate structure Can_e

图 4 实体候选结构 Can_e

算法 2. Generate_candstruct.

输入:实体表 R,S,连接属性上的前缀索引 T_r,T_s;

输出:实体对候选结构集合 $Cans$.

(1) $Cans=\varnothing$

(2) 合并 T_r,T_s 中具有相同 q-gram 的倒排表,组成实体候选结构 Can_e.

(3) For 所有的 Can_e

a) If $Can_e.eid \in Cans.eid$,将 $Can_e.cid$ 按权重由大到小插入到 $Cans$ 中的 $Can_e.cid$ 链表中;

b) If $Can_e.eid \notin Cans.eid$,将 Can_e 插入到 $Cans$ 中;

(4) 返回 $Cans$.

在生成实体候选结构集合 $Cans$ 的过程中:

- 首先,初始化 $Cans$,将其设为空(见算法第(1)步);
- 然后,合并连接属性上的前缀索引,在合并的过程中,由于每个 q-gram 都有唯一的全局序,因此可以采用哈希的方法,把具有共同 q-gram 的倒排表哈希到一个 Can_e 结构中,完成算法第(2)步;
- 之后,判断生成的 Can_e 结构中的实体号对是否已经在 $Cans$ 中出现,若是,则合并两个 Can_e 结构(第(3)步).在合并过程中,将 cid 对相同的去重,按 cid 对权重由大到小插入到 $Cans$ 中.

算法第(3)步的目的是为了第(2)步中生成的 Can_e 集合按照 cid 对聚集,将具有相同 cid 对(两个 cid 相加之和相同)的 Can_e 合并在一起,使得合并之后,$Cans$ 中的每个实体对只出现一次,简化下一步的过滤操作;将 cid 对排序的目的也是为了下一步过滤时无需遍历整个 cid 对链表,减少时间复杂度.

算法时间复杂度分析:设 $P(s)$,$P(r)$为实体表 S 和 R 中要连接的实体属性值的前缀集合,$l_s(e)$为表 S 中 q-gram 为 e 的倒排索引的大小,实体属性值的平均个数设为 c.为了得到候选实体候选结构集合,Generate_candstruct 需要遍历所有 q-gram$\in P(s)\cap P(s)$的倒排表,时间代价为 $\sum_{e\in P(s)\cap P(r)}(l_s(e)+l_r(e))$.设倒排表的平均长度为 l,公共 q-gram 个数为 n,设算法最后返回的结果大小为 C_Q,则算法第(3)步排序的平均复杂度为 $O(c\log c\cdot C_Q)$.算法总的复杂度为两项相加,即 $O(n\cdot l)+O(c\log c\cdot C_Q)$.

2.2.2.2 过滤生成候选集算法及分析

若将合并后的 eid 对全部加入候选集中,则候选集中的字符串对的数量是很庞大的,计算字符串对之间的编辑距离将会严重影响到算法的性能.因此,考虑使用过滤措施过滤合并之后的索引,进一步减少候选集的数量.

通过观察可知,实体中权重大的字符串对实体对相似性贡献较大.如果能够尽早确定权重大的字符串对之

间是否匹配,就能过滤掉更多的候选,使得最终候选集尽可能的小.

基于上述观察,我们给出了两个过滤措施:

性质 1. 给定属性值 $s=\{(p_1,\sigma_1),(p_2,\sigma_2),\ldots,(p_m,\sigma_m)\}$,$r=\{(q_1,\delta_1),(q_2,\delta_2),\ldots,(q_n,\delta_n)\}$,相似性阈值$\theta$,设 s,r 已经按权重由大到小排序,相似性阈值θ.对于实体候选结构 $Can_e.cid$ 链表的第 1 个 cid 对(cid_1,cid_2),若:

$$p_{cid_1}\cdot p_{cid_2}<\frac{\theta}{|Cane.cid|},$$

则 $\hat{d}(Can_e.eid)<\theta$.

证明:若 s 与 r 相似,则有 $\sum_{d(s_i,r_j)\leqslant\tau} p_i\cdot q_j\cdot sim(s_i,r_j)\geqslant\theta$;

因为 $0\leqslant sim(s_i,r_j)\leqslant 1$,所以 $\sum_{d(s_i,r_j)\leqslant\tau} p_i\cdot q_j\geqslant\theta$;

又由 $d(s_i,r_j)\leqslant\tau$的字符串对的个数为 $Can_e.cid$.由鸽巢原理可得:若 $\sum_{d(s_i,r_j)\leqslant\tau} p_i\cdot q_j\geqslant\theta$,则必有:

$$p_i\cdot p_j<\frac{\theta}{|Cane.cid|};$$

又 $p_{cid_1}\cdot p_{cid_2}\geqslant p_i\cdot p_j$,因此,若 $p_{cid_1}\cdot p_{cid_2}<\frac{\theta}{|Cane.cid|}$,则 $\sum_{d(s_i,r_j)\leqslant\tau} p_i\cdot q_j<\theta$.

即,$\sum_{d(s_i,r_j)\leqslant\tau} p_i\cdot p_j\cdot sim(s_i,r_j)<\theta$. □

性质 2. 实体候选结构 $Can_e.cid$ 链表的 cid 对记为$(cid_{11},cid_{21}),(cid_{12},cid_{22}),\ldots,(cid_{1m},cid_{2m})$,其中,$m$ 为链表的长度.令 $k=\min\{j\mid\sum_{i=1}^{j} p_i\cdot q_i\geqslant\theta\}$,若 $k>m$,则 $\hat{d}(Can_e.eid)<\theta$.

证明:若 s 与 r 相似,则有 $\sum_{d(s_i,r_j)\leqslant\tau} p_i\cdot q_j\cdot sim(s_i,r_j)\geqslant\theta$;

因为 $0\leqslant sim(s_i,r_j)\leqslant 1$,所以 $\sum_{d(s_i,r_j)\leqslant\tau} p_i\cdot q_j\cdot sim(s_i,r_j)\leqslant\sum p_i\cdot q_j$.

若 $k>m$,则有 $\sum_{i=1}^{m} p_{cid_{1i}}\cdot p_{cid_{2i}}<\theta$,即,$\sum p_i\cdot q_j<\theta$.所以,$\sum_{d(s_i,r_j)\leqslant\tau} p_i\cdot p_j\cdot sim(s_i,r_j)\leqslant\theta$. □

性质 1 只需判断实体候选结构 $Can_e.cid$ 链表的第 1 个 cid 对是否满足相似性阈值;性质 2 对满足性质 1 的候选进行再次过滤,进一步减小候选集的产生.

例 5:表 1 与表 2 在 Name 实行上做相似性连接操作,设τ=2,θ=0.5.观察例 3 的实体候选结构集合,实体对(1,1)中,cid 对(2,1)的权重为 0.08,链表的长度为 2.由 0.08<0.5/2,可由性质 1 剪掉实体对(1,1)所在的实体候选结构.又由实体对(1,2),其第 1 个 cid 对为(1,2),其权重 0.32>0.5/2,满足第 1 个过滤条件.然而(1,2)和(2,1)的概率相加为 0.32+0.12=0.44<0.5,根据性质 2,从候选集中被剪掉.

从性质 1 可知:在 Generate_candstruct 算法产生的实体候选结构集合中,若候选结构 eid 对第 1 个 cid 权重过小,则此实体对一定不满足相似性阈值,可以直接被过滤掉.性质 2 综合考虑了实体对的概率,进一步过滤掉满足性质 1 但是不满足相似度阈值的候选.算法 3 给出了使用性质 1、性质 2 过滤产生候选的具体过程.

算法 3. Generate_cand.

输入:实体表 R,S,实体对候选结构集合 $Cans$;

输出:实体候选结构集合 $Canf$.

(1) $Canf=\varnothing$

(2) For $Can_e\in Cans$,检索 $Can_e.eid$ 的 cid 链表.

  a) 采用性质 1、性质 2 进行两步过滤

  b) 若满足两个过滤条件,将 $Can_e.eid$ 插入到 $Cans$ 中

(3) 返回 $Cans$.

Generate_cand 算法首先将候选集设为空;其次遍历 Generate_candstruct 算法产生的每个实体候选结构,对于每一个 Can_e,使用性质 1 和性质 2 过滤,返回最终候选集.

算法的时间复杂度分析:此算法通过遍历第 2.2.2.2 节中产生的实体候选结构,可确定时间复杂度为 C_Q,即

为 $O(C_Q)$.

2.2.3 确认候选集算法及分析

对于生成的实体候选结构集合,需要计算其 *cid* 对之间的编辑距离.通过编辑距离计算出相似度和相似度阈值 θ 比较可知:若大于 θ,则表示这两个实体是相似的,能够进行连接.遍历所有的实体候选结构,返回两个实体表相似性连接的最终结果.确认算法见算法 4.

算法 4. Verify.

输入:实体对候选结构集合 *Cans*,编辑距离阈值 τ,相似性阈值 θ;

输出:满足连接条件的实体对集合 *Result*.

(1) *Result*=∅

(2) For $Can_e \in Cans$,检索 $Can_e.eid$ 的 *cid* 对链表.

a) 计算链表中每个 *cid* 对的编辑距离,得到其相似性,累加值赋给对应 eid 的质量度

b) 若 *eid* 对的相似性大于 θ,则插入到 *Result* 集合中.

(3) 返回 *Result*.

算法 4 用迭代的方法计算实体候选对的相似度.由 Generate_cand 算法可知,$Can_e.eid$ 的 *cid* 对链表中包含了所有可能满足编辑距离阈值的字符串对.确认算法用定义 5 定义的实体相似性计算相似性,迭代累加实体对的相似性.

时间复杂度分析:设最终候选集 $Can_e.eid$ 的 *cid* 对链表的平均长度为 *len*(*eid*),总的候选集个数为 *n*,*cost* 为计算每一对字符串相似性的平均值,则验证算法的时间复杂度为 $O(cost \cdot len(eid))$.

2.2.4 ES-JOIN 算法分析

ES-JOIN 算法的正确性分析:算法 Generate_candstruct 和 Generate_cand 产生的候选结果包含了所有可能连接的实体对.在 verify 算法中,$Can_e.eid$ 的 *cid* 对链表中包含了所有可能满足编辑距离阈值的字符串对,因此,所有满足相似度阈值的实体对都会被返回,由此可知算法的正确性.

ES-JOIN 算法的时间复杂度分析:ES-JOIN 算法的时间复杂度为第(2)步~第(4)步时间复杂度之和.其中:由第 2.2.2.1 节得到算法第(2)步合并前缀索引的时间复杂度为 $O(n \cdot l)+O(c\log c \cdot C_Q)$,由第 2.2.2.2 节得到算法第(3)步过滤生成候选集的时间复杂度为 $O(C_Q)$,由第 2.2.3 节得到确认算法第(4)步确认操作的时间复杂度为 $O(cost \cdot len(eid))$.因此,ES-JOIN 算法总的时间复杂度为 $O(n \cdot l)+O(c\log c \cdot C_Q)+O(C_Q)+O(cost \cdot len(eid))$.其中,*l* 为倒排表的平均长度,*n* 为公共 *q*-gram 个数,C_Q 为合并双层前缀索引后的结果大小,*len*(*eid*)为候选集 $Can_e.eid$ 的 *cid* 对链表的平均长度.

2.3 ES-JOIN算法的优化

本节讨论进一步优化 ES-JOIN 算法的策略.

给定实体属性值 $s=\{(p_1,\sigma_1),(p_2,\sigma_2),\ldots,(p_m,\sigma_m)\}$,$r=\{(q_1,\delta_1),(q_2,\delta_2),\ldots,(q_n,\delta_n)\}$,设 *s*,*r* 已经按权重由大到小排序.观察性质 1,若合并双层前缀索引之后第 1 对 *cid* 代表的字符串对的权重积不够大,则这两个实体一定不会相似.也就是在过滤过程中,相似性很小的字符串对于产生候选的贡献很小(只有在性质 2 中用到).基于此,应该在产生候选集时尽可能多且尽可能早地过滤掉那些权重比较大的 *cid* 对,进一步减少候选集的个数,提高算法的执行效率.

性质 3. 给定字符串 σ_1,σ_2,编辑距离阈值 τ,若 $d(\sigma_1,\sigma_2)\leqslant\tau$,则 σ_1,σ_2 的前 $q\tau+k$ 个前缀 *q*-gram 至少有 *k* 个公共 *q*-gram.其中,*k* 的大小采用 FIXPREFIXSCHEME 框架[17]确定,而 FIXPREFIXSCHEME 框架使用固定长度的前缀机制剪枝不相似的实体对.

为了尽可能多地过滤掉权重比较大的 *cid* 对,我们可以修改前缀的长度.然而,前缀长度的增加使得双层前缀索引变大,合并双层前缀的 Generate_candstruct 算法开销增大.为了平衡这两个因素,优化算法只对实体中权重最大的字符串增加其前缀长度.具体做法是:对于实体中最大权重的字符串,增加其前缀长度,将增加的前缀在 *q*-gram 倒排索引中对其所在的 *cid* 标记,在合并双层前缀索引时,若 *q*-gram 倒排索引中有一个 *cid* 是标记的,

则只合并同样标记的 *cid*;之后,对重复的 *cid* 对计数,若个数小于应该满足的公共 *q*-gram 个数,具体见性质 3,则将 *cid* 对从 *eid* 链表中删除.此做法减少了 C_Q 的个数.OPT_ES-JOIN 算法与 ES-JOIN 算法流程相同.Generate_candstruct 和 Generate_cand 算法有所改变.OPT_Generate_candstruct 算法在第(2)步合并双层前缀索引时,只将都没有标记或是都有标记的倒排表中索引项合并;再在第(3)步的步骤 a)中对两个 *cid* 均被标记的重复 *cid* 对计数.由性质 3,当 *cid* 对的计数不小于 *k* 时,*cid* 对才成为满足编辑距离阈值的字符串对.因此在 OPT_Generate_cand 算法中,我们在过滤之前先检查计数数组,将不满足计数条件的 *cid* 对删除,若权重比较大的 *cid* 对被删除,则在第(2)步进行过滤时将更容易过滤到不满足相似性阈值的实体对.

算法 5. OPT_Generate_cand 算法.

输入:实体表 *R*,*S*,实体对候选结构集合 *Cans*,前缀过滤长度 *k*;

输出:实体候选结构集合 *Can*.

(1) *Can*=∅,

(2) For $Can_e \in Cans$.

a) while $Can_e.eid$ 的 *cid* 对被标记且计数小于 *k*,则将 *cid* 对从链表中删除.

b) 采用性质 1、性质 2 进行两步过滤

c) 若满足两个过滤条件,将 $Can_e.eid$ 插入到 *Can* 中

(3) 返回 *Can*

例 6:设 τ=1,*q*=2.有两实体属性值{Microsoft,1.0}和{Macrofilt,1.0},其前缀分别为{Mi,ic,cr}和{Ma,ac,cr}.经过前缀索引合并,这两个实体被加入实体候选集中,且性质 1 和性质 2 均不能过滤掉.使用优化算法,设 *k*=2,由性质 3,公共前缀至少应有 3 个.显然,当前缀长度增加 2 时,其前缀的公共 *q*-gram 为 2<3,被性质 3 过滤掉.

此外,在合并双层前缀索引的算法中,我们可以采用现有的过滤方法,如后缀过滤等,进一步减少候选集的产生.

3 实验结果和分析

3.1 实验配置和数据集

本文的实验在 PC 机上运行,其内存 4G,硬盘 500G,CPU 为 Pentium@2.93GHz,操作系统为 Windows XP.所有算法用 C++实现.

- 真实数据集.

本次实验使用的真实数据集为电子商务数据,数据集分别为从 eBay(http://www.ebay.com)网站和 Amazon(http://www.amazon.com)网站上爬取 Computer Science 类的书籍信息.之后,采用现有的方法对爬来的数据集进行实体识别,将实体识别的结果,组织成实体类型的关系表.其中,实体属性值中字符串的概率值是该字符串在表示其实体属性值的数据集中出现的频率,频率大小代表此字符串表示相应实体的质量度.经过这一系列处理后,得到了两个 1.2M 条实体元组和两个 0.5M 条实体元组的关系表,其中,每条元组包含 4 个实体属性:书名、作者、出版社和价格,作者的属性值平均个数 3 个.我们将这两个实体表在书名属性上进行相似性连接,得到相似名称的书籍的价格是否也是相似的、作者是否有共同特征等信息.其中,书名的字符串长度平均为 100 个字符.

- 人工数据集.

实验使用的合成数据集分为两组,两组均由数据产生器随机产生具有多个属性的元组,元组的属性值的平均长度分别为 100,20 个字符.我们对产生的数据进行实体识别,并将实体识别的结果抽取出来,组成一个实体元组.在抽取的过程中,一个实体属性值限定 1~5 个值.其中,随机产生的元组大小为 10k,50k,100k,500k,1M.

由于 ES-JOIN 算法框架在合并双层前缀索引产生候选集时可以使用现有的字符串相似性连接的各种过滤措施,本文的主要工作是如何解决实体类型的相似性连接问题.仅基于字符串相似性连接的操作不适用于本文的问题,而基于集合的相似性连接不考虑集合内部字符串的相似性,因此,这两类与本文算法都不具有可比性.然而,我们可以修改基于字符串相似性连接的 ED-Join 算法[8]与本文提出的算法进行比较.ED-Join 修改如下:将

两个实体表拆分成两个普通的关系表,用 ED-Join 实现连接操作,再将得出的结果合并,计算出实体连接的最终结果.我们将修改后的算法记为 ED-JOIN 算法.

3.2 实验结果和分析

实验从 3 个方面来评估算法的特点:第 3.2.1 节使用真实数据集和合成数据验证算法的效率和扩展性;第 3.2.2 节使用合成数据分析各种参数对算法效率的影响;第 3.3.3 节分析文中所提过滤措施的过滤效率问题.

实验的时间效率使用 *Time* 来描述,它的值 CPU 时间和 I/O 时间之和.

3.2.1 算法的效率和扩展性分析

首先,我们在真实数据集上验证 ED-JOIN,ES-JOIN 算法和 OPT_ES-JOIN 算法的有效性.此次实验最小编辑距离阈值设为 3,相似性阈值设为 0.8,*q*-gram 设为 2-gram.在 OPT_ES-JOIN 算法中,*k* 的大小设为 2.实验结果如图 5 所示.

图 5 表明:与 ED-JOIN 算法相比较,ES-JOIN 算法和 OPT_ES-JOIN 算法具有较好的效率.这是因为 ED-JOIN 需要计算出所有具有高相似度的字符串对,而 ES-JOIN 算法和 OPT_ES-JOIN 算法则采用过滤措施过滤掉了相似度很高但是质量度很低的字符串对.

其次,我们在合成数据集上更进一步分析算法的特性.为了分析的方便,令连接的两表大小相等,分别为 10k,50k,100k,500k,1M.参数设置和真实数据集上的设置相同.

比较实验比较 ES-JOIN 算法和 OPT_ES-JOIN 算法效率和可扩展性,实验结果如图 6 所示.

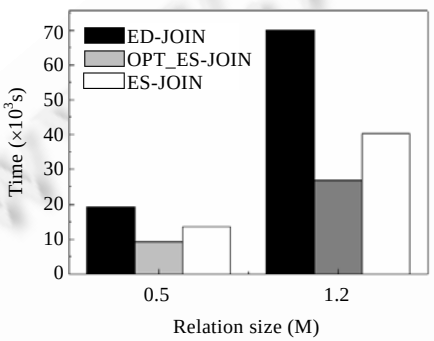

Fig.5 Efficiency of algorithms in real data

图 5 真实数据上算法的效率

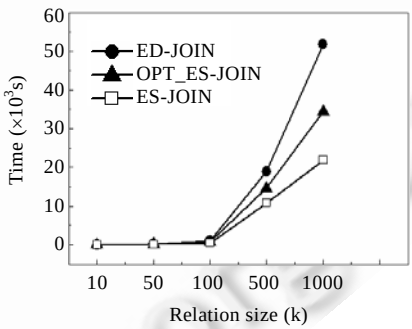

Fig.6 Efficiency and scalability of algorithms

图 6 算法的效率和可扩展性

从图 6 中可以看出:OPT_ES-JOIN 算法较 ES-JOIN 算法的时间效率更好一些,且都好于 ED-JOIN 算法.其中,OPT_ES-JOIN 算法在产生最终候选集的过程中过滤掉了更多不满足相似性阈值的实体对,使得确认算法的输入规模变小,总的效率提高,执行时间变少.此外,OPT_ES-JOIN 算法和 ES-JOIN 算法都具有较好的可扩展性.

3.2.2 参数对算法效率的影响

本节使用合成数据集通过实验检验编辑距离阈值 τ、相似性阈值 θ、OPT_ES-JOIN 算法中 k 以及字符串平均长度对算法效率的影响.两表大小均为 10k.

3.2.2.1 参数 τ 对算法效率的影响

当 τ 作为参数时,θ 设为 0.7,τ 取 2,3,4,5,6,7.实验结果如图 7 所示.

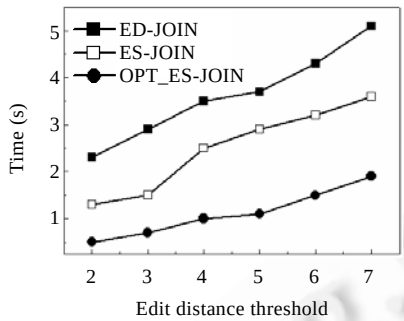

Fig.7 Effect of edit distance

图 7 编辑距离阈值对算法效率的影响

图 7 表明:随着编辑距离阈值的增大,算法的执行时间在总体趋势上增大.这是因为编辑距离阈值增大时,前缀的长度增大,倒排索引变大,查找倒排索引的时间变长.此外,编辑距离阈值增大后,满足条件的字符串对变多,从而算法的执行时间变长.

3.2.2.2 参数θ对算法效率的影响

当参数为θ时,固定最小编辑距离为 3,k 大小为 3,θ取 0.5,0.6,0.7,0.8.实验结果如图 8 所示.

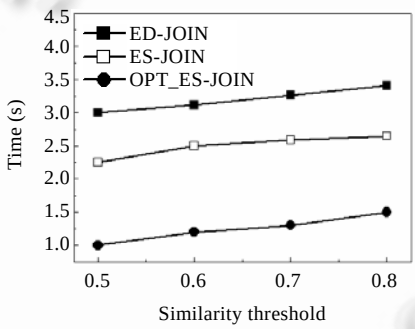

Fig.8 Effect of similarity threshold

图 8 相似性阈值对算法效率的影响

直观上讲,θ越大,最终返回的结果越少,算法在合并前缀索引以及使用过滤措施过滤时过滤掉的实体对越多,确认所消耗的时间越少.实际运行结果验证了我们的直观思想(如图 8 所示).

3.2.2.3 参数 k 对 OPT_ES-JOIN 算法效率的影响

当 k 为参数时,θ设为 0.7,τ取 3,k 取 1,2,3,4.实验结果如图 9 所示.

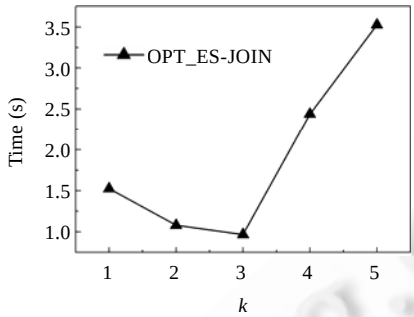

Fig.9 Effect of k in OPT_ES-JOIN algorithm

图 9 OPT_ES-JOIN 中 k 的大小对算法效率的影响

从图 9 中可以看出:随着 k 的增大,算法所耗时间降低;然而当增大到一定值时,算法效率开始下降.这是因为 k 的增大虽然减小了候选集合,但是合并索引的时间相应地增大了,k 的选取需要平衡这两个因素.在本次实验中,k 的最佳取值为 3.

3.2.2.4 字符串长度对算法效率的影响

本组实验比较字符串平均长度为 20 及 100 时算法的效率.实验参数设置如下:θ设为 0.7,τ取 3,k 取 1,2.实验结果如图 10 所示.

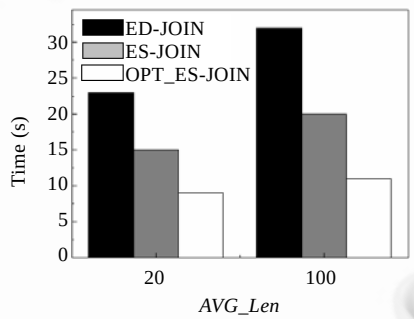

Fig.10 Efficiency of algorithms by average length of string

图 10 字符串平均长度对算法效率的影响

图 10 表明:字符串平均长度大时,算法所需处理的 q-gram 集合变大,执行时间变长.然而,由于基于 q-gram 的方法在处理长字符串时较之短字符串有更好的效率(短字符串产生大量的需要进一步确认的候选集合),因此,其效率不会随着数据量的增加(长字符串数据量大)而减小得过快.从图 10 可以看出:当字符串平均长度从 20 增加到 100 时,执行时间并没有变得过长.

3.2.3 过滤措施的实验效果

此次实验参数设置与第 3.2.1 节中真实数据集上验证 ES-JOIN 算法和 OPT_ES-JOIN 算法有效性的实验设置相同.令 filter 1,filter 2,filter 3 分别为由性质 3、性质 1、性质 2 过滤掉的实体对的数量大小.no filter 表示不采用本文提出的过滤措施,使用长度过滤后候选集的大小.实验结果如图 11 所示.

从图 11 中可以看出:首先,利用性质 3 能够过滤掉 20%的候选;性质 1 能够过滤掉剩余候选集的 55%;性质 2 用于性质 1 过滤之后,能够过滤掉剩余候选集的 45%.实验表明,我们提出的过滤措施具有很好的过滤效果.

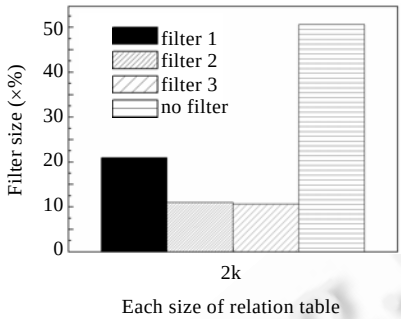

Fig.11 Result of filtering measures

图 11 过滤措施的实验效果

4 相关工作

现有的能够描述劣质信息的数据模型是概率数据模型,概率模型是描述不确定信息的有效数据模型,基于概率模型的不确定数据质量问题已经被广泛研究.概率模型自从 1988 年被文献[19]提出后,经过几十年的发展已经日趋完善.概率数据模型上的数据操作也已经有了多种类型,如范围查询、top-k、skyline 查询等.从数据的角度看,概率数据能够很好地描述数据的不确定信息;然而从数据操作上看,概率数据库上的查询操作考虑所有可能世界,返回所有可能的结果,因此,查询时并没有考虑到操作过程对结果准确度的影响.并且,假设用户给定一个清洁度的阈值,概率模型并不能据此阈值进行过滤得到用户满意的结果.文献[22]使用期望编辑距离为相似性度量措施,提出了概率字符串的相似性连接,使用过滤和确认的方法计算满足相似性阈值的字符串对集合.然而,若两个实体对相似,对相似值有贡献的是编辑距离小于给定编辑距离阈值的字符串对,基于此,本文定义的相似性度量假设足够相似的字符串对才对整个结果的质量度产生影响.给定编辑距离阈值,文献[22]提出的方法并不能返回同时满足相似性阈值和编辑距离阈值的结果.

目前,针对字符串相似性连接操作有着丰富的研究成果.

基于字符串的相似性连接主要以编辑距离作为相似性度量:All-Pairs-ED[9]首先将每个字符串拆成 q-gram 集合,然后按事先排好的 q-gram 顺序选择前 $q\tau$+1 个 q-gram.根据鸽巢原理,若两个字符串相似,则前 $q\tau$+1 个 q-gram 必有一个相同.基于此,此方法过滤掉没有公共 q-gram 的字符串对,然后确认剩余的候选集.ED-Join[8]采用基于位置和内容的不匹配过滤策略,减少了候选集的大小.Trie-Join[7]使用 trie 树结构,采用前缀过滤的方法解决相似性连接问题.Part-Enum[13]针对基于海明距离的相似性连接,提出了一种有效的 signature scheme.PP-join[21]通过引进位置过滤和后缀过滤,改进了 all-pair 算法.Pass-Join[6]提出了基于划分的方法,将每个字符串划分成段,对每个段建立索引,根据相似的字符串对必有一个段相同来产生候选集.

基于字符串集合的相似性连接多以 Jaccard 距离作为度量相似性函数,这些方法[13−15]的基本框架都是采用前缀索引,过滤掉不满足条件的集合对,之后进行确认操作.

为了提高字符串(集合)相似性连接算法的效率,文献[16]提出了将定长的 gram 修改为变长的 gram.文献[17]优化了固定长度前缀,对每个字符串,选择其最优的前缀长度.本文优化算法中,前缀长度参照了文献[17]的研究成果.

5 结 论

本文提出了基于实体的相似性连接算法 ES-JOIN,介绍了新的索引结构——双层前缀索引,并基于权重(指亮度)提出了两种新的过滤措施;在理论上分析了 ES-JOIN 算法的时间复杂度,证明了 ES-JOIN 算法的正确性,并在建立前缀索引时优化了算法;最后,用实验验证了 ES-JOIN 算法和优化算法具有很好的效率和扩展性,过滤措施也有很好的过滤效果.未来的工作包括如何确定两个实体表连接的属性以及实体数据库上的查询操作.

References:

[1] Bertossi L, Kolahi S, Lakshmanan L. Data cleaning and query answering with matching dependencies and matching functions. In: Abiteboul S, Böhm K, Koch C, Tan KL , eds. Proc. of the 27th Int'l Conf. on Data Engineering. Hannover: IEEE Computer Society, 2011. 268−279. [doi: 10.1145/1938551.1938585]

[2] Dong X, Halevy AY, Yu C. Data integration with uncertainty. In: Koch C, Gehrke J, Garofalakis MN, Srivastava D, Aberer K, Deshpande A, Florescu D, Chan CY, Ganti V, Kanne CC, Klas WJ, Neuhold E, eds. Proc. of the 33rd Int'l Conf. on Very Large Data Bases. Vienna: ACM Press, 2007. 687−698.

[3] Ji S, Li G, Li C, Feng JH. Efficient interactive fuzzy keyword search. In: Proc. of the 18th Int'l Conf. on World Wide Web. Madrid: ACM Press, 2009. 371−380. [doi: 10.1145/1526709.1526760]

[4] Timothy C, Justin Z. Methods for identifying versioned and plagiarized documents. Journal of the American Society for Information Science and Technology, 2003,54(3):203−215. [doi: 10.1002/asi.10170]

[5] Broder AZ, Glassman SC, Manasse MS, Zweig G. Syntactic clustering of the Web. Computer Networks and ISDN Systems, 1997, 29(8):1157−1166. [doi: 10.1016/S0169-7552(97)00031-7]

[6] Li G, Deng D, Wang J, Feng JH. Pass-Join: A partition-based method for similarity joins. VLDB Endowment, 2011,5(3):253−264. [doi: 10.14778/2078331.2078340]

[7] Wang J, Feng J, Li G. Trie-Join: Efficient trie-based string similarity joins with edit-distance constraints. VLDB Endowment, 2010, 3(1-2):1219−1230. [doi: 10.14778/1920841.1920992]

[8] Xiao C, Wang W, Lin X. Ed-Join: An efficient algorithm for similarity joins with edit distance constraints. VLDB Endowment, 2008,1(1):933−944. [doi: 10.14778/1453856.1453957]

[9] Bayardo J, Ma Y, Srikant R. Scaling up all pairs similarity search. In: Proc. of the 16th Int'l Conf. on World Wide Web. Banff: ACM Press, 2007. 131−140. [doi: 10.1145/1242572.1242591]

[10] Chaudhuri S, Ganti V, Kaushik R. A primitive operator for similarity joins in data cleaning. In: Liu L, Reuter A, Whang KY, Zhang JJ, eds. Proc. of the 22nd Int'l Conf. on Data Engineering. Atlanta: IEEE Computer Society, 2006. 1−5. [doi: 10.1109/ICDE.2006.9]

[11] Li GL, Deng D, Feng J. Faerie: Efficient filtering algorithms for approximate dictionary-based entity extraction. In: Sellis TK, Miller RJ, Kementsietsidis A, Velegrakis Y, eds. Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. Athens: ACM Press, 2011. 529−540. [doi: 10.1145/1989323.1989379]

[12] Navarro G. A guided tour to approximate string matching. ACM Computing Surveys, 2011,33(1):31−88. [doi: 10.1145/375360.375365]

[13] Arasu A, Ganti V, Kaushik R. Efficient exact set-similarity joins. In: Dayal U, Whang KY, Lomet DB, Alonso G, Lohman GM, Kersten ML, Cha SK, Kim YK, eds. Proc. of the 32nd Int'l Conf. on Very Large Data Bases. Seoul: ACM Press, 2006. 918−929.

[14] Sarawagi S, Kirpal A. Efficient set joins on similarity predicates. In: Weikum G, König AC, Deßloch S, eds. Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. Paris: ACM Press, 200. 743−754. [doi: 10.1145/1007568.1007652]

[15] Vernica R, Carey MJ, Li C. Efficient parallel set-similarity joins using MapReduce. In: Elmagarmid AK, Agrawal D, eds. Proc. of the ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD 2010). Indianapolis: ACM Press, 2010. 495−506. [doi: 10.1145/1807167.1807222]

[16] Li C, Wang B, Yang XC. Vgram: Improving performance of approximate queries on string collections using variable-length grams. In: Koch C, Gehrke J, Garofalakis MN, Srivastava D, Aberer K, Deshpande A, Florescu D, Chan CC, Ganti V, Kanne C, Klas W, Neuhold EJ, eds. Proc. of the 33rd Int'l Conf. on Very Large Data Bases. Vienna: ACM Press, 2007. 303−314.

[17] Wang J, Li G, Feng J. Can we beat the prefix filtering: An adaptive framework for similarity join and search. In: Candan KS, Chen Y, Snodgrass RT, Gravano L, Fuxman A, eds. Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. Scottsdale: ACM Press, 2012. 85−96. [doi: 10.1145/2213836.2213847]

[18] Wang J, Li G, Feng J. Fast-Join: An efficient method for fuzzy token matching based string similarity join. In: Abiteboul S, Böhm K, Koch C, Tan KL, eds. Proc. of the 27th Int'l Conf. on Data Engineering. Hannover: IEEE Computer Society, 2011. 458−469. [doi: 10.1109/ICDE.2011.5767865]

[19] Green T, Tannen V. Models for incomplete and probabilistic information. Current Trends in Database Technology—EDBT, 2006, 4254:278−296. [doi: 10.1007/11896548_24]

[20] Wang HZ, Li JZ, Gao H. Data model for dirty databases. Ruan Jian Xue Bao/Journal of Software, 2012,23(3):539−549 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4042.htm [doi: 10.3724/SP.J.1001.2012.04042]

[21] Xiao C, Wang W, Lin XM, Yu JX. Efficient similarity joins for near duplicate detection. In: Huai JP, Chen R, Hon HW, LiuYH, Ma WY, Tomkins A, Zhang XD, eds. Proc. of the 17th Int'l Conf. on World Wide Web. Beijing: ACM Press, 2008. 131−140. [doi: 10.1145/1367497.1367516]

[22] Jestes J, Li FF, Yan ZP, Yi K. Probabilistic string similarity joins. In: Elmagarmid AK, Agrawal D, eds. Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. Indianapolis: ACM Press, 2010. [doi: 10.1145/1807167.1807204]

附中文参考文献:

[20] 王宏志,李建中,高宏.一种非清洁数据库的数据模型.软件学报,2012,23(3):539−549. http://www.jos.org.cn/1000-9825/4042.htm [doi: 10.3724/SP.J.1001.2012.04042]

刘雪莉(1987－),女,河南新乡人,博士生,CCF 学生会员,主要研究领域为数据质量.

王宏志(1978－),男,博士,副教授,CCF 高级会员,主要研究领域为 XML 数据管理,信息集成.

李建中(1950－),男,教授,博士生导师,CCF 高级会员,主要研究领域为数据库系统实现技术,数据仓库,半结构化数据,传感器网络,压缩数据库技术,Web 数据集成,数据挖掘,计算生物学.

高宏(1966－),女,博士,教授,博士生导师,CCF 高级会员,主要研究领域为复杂结构数据管理.