文章编号: 1001-0920(2006)07-0805-04

# 一种改进的相似重复记录检测方法

朱恒民, 王宁生
(南京航空航天大学 CMS 工程中心, 南京 210016)

**摘 要**: 针对当前相似重复记录检测方法中存在的问题, 提出一种改进方法. 该方法根据关系表的决定属性值划分记录集, 并在每个决定属性值类中检测相似重复记录. 在决定属性值聚类时, 提出了动态优先队列聚类算法和合并逆序算法, 尽可能使相似重复的属性值聚为同一类; 在记录聚类时提出了类调整算法, 以提高类的代表记录的代表性. 通过大量的实验分析, 验证了该方法的有效性.

**关键词**: 相似重复记录; 优先队列; 聚类; 数据清洗; 数据预处理
**中图分类号**: TP311 **文献标识码**: A

# Improved Method for Detecting Approximately Duplicate Database Records

*ZHU Heng-min, WANG Ning-sheng*
(Research Center of CIMS Engineering, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China. Correspondent: ZHU Heng-min, E-mail: zhuhengmin@sina.com)

**Abstract**: Problems in current existing methods of detecting approximately duplicate records are discussed, and an improved method is proposed. The proposed method partitions record set according to decided attribute values, and then detects approximately duplicate records in each class of decided attribute value. In order to find duplicates as many as possible, two algorithms for clustering decided attribute values are proposed under the assumption of transitivity. Since representative record of class produced by traditional priority queue algorithm is not representative, an algorithm of class modification is given. The effectiveness of the proposed method is verified through many experiments and analysis.

**Key words**: Approximately duplicate records; Priority queue; Clustering; Data cleaning; Data preprocess

## 1 引 言

数据库中的相似重复记录, 是指那些客观上表示现实世界同一实体, 但由于在格式和拼写上有差异而导致数据库管理系统不能正确识别的记录[1]. 当数据库包含较多的相似重复记录时, 一方面导致数据冗余, 浪费存储空间; 另一方面, 较多的相似重复记录影响了数据的真实分布, 从而降低了数据处理结果的正确性. 因此, 相似重复记录是影响数据质量的一个重要原因.

检测相似重复记录的研究工作较多. 文献[1]给每个记录赋一个 $N$-gram 值, 根据该值对记录排序, 然后采用优先队列技术对记录聚类; [2]将整个记录看成一个字符串, 对记录进行排序, 然后使用包含固定大小的优先队列来顺序扫描所有的排序记录, 并将它们聚类; [3]根据记录中的相关字段或字段的一部分构成键, 按键对记录排序, 并采用滑动窗口技术来限制比较的记录; [4]将字段值分解成多个 tokens, 并对 tokens 进行排序, 然后排序记录, 并采用滑动窗口技术比较邻近范围内两两记录.

上述相关工作都基于"排序+ 合并"的思想. 对记录集排序时, 有的是按字段值的字典序排列, 但字典序对错误位置较敏感; 有的是按构造的字段排序,

**收稿日期**: 2005-03-31; **修回日期**: 2005-07-21.
**基金项目**: 国家科技部基金项目(2002ED691036).
**作者简介**: 朱恒民(1974—), 男, 南京人, 博士生, 从事数据挖掘、数据库技术等研究; 王宁生(1936—), 男, 南京人, 教授, 博士生导师, 从事 FMS, CMS, ERP 等研究.

但要构造能反映整条记录信息的字段是很难的. 合并相似重复记录时, 滑动窗口技术假定相似重复记录具有传递性, 这会产生 false-positive 类型错误; 优先队列技术在产生类的同时, 可以给出类的代表记录, 但代表记录往往缺乏代表性.

针对当前相似重复记录检测方法中存在的问题, 本文提出一种改进方法. 该方法首先对决定属性值初聚类, 然后对产生的每个属性值类进行记录聚类, 最后调整产生的记录类, 得到最终结果.

## 2 相似重复属性值的聚类算法

关系表中的记录和现实世界的实体是一一对应的, 关系表中的属性描述了实体的特征. 至少有一个属性对实体的身份起决定性作用(即只有该属性值相似重复, 两实体才有可能相同), 否则不同实体将无法区分. 例如, 对于关系表中的学生信息(姓名, 家庭住址, 出生日期, …), 属性"姓名"和"家庭住址"都对学生身份起了决定性作用.

### 2.1 选择决定属性

决定属性是指对实体身份起决定性作用的某一属性, 它被用来划分记录集. 决定属性一般由用户根据背景知识来指定. 当关系表中对实体身份起决定性作用的属性较多时, 可从两个方面综合考虑选择决定属性: 1) 不同属性值的数量 $N_v$; 2) 属性值的平均字符数量 $n_c$. $N_v$ 越大, 根据决定属性划分的类平均尺寸越小, 有利于计算记录的聚类; $n_c$ 越小, 属性值聚类的精度越高, 时间开销越小.

### 2.2 属性值的距离

由于编辑距离法求两个字符串距离的时间开销很大, 本文采用文献[5]提出的基于 $q$-gram 算法的两字符串距离指标. 令 $G_1$ 和 $G_2$ 分别表示字符串 $s_1$ 和 $s_2$ 的所有 $q$-gram 集合, 则两字符串的距离

$$\mathrm{dist}(s_1, s_2) = 1 - (|G_1 \cap G_2| / |G_1 \cup G_2|).$$

### 2.3 决定属性值聚类算法

1) 处理决定属性的每个值. 首先将决定属性值按标点符号自动分隔成 tokens, 每个 token 实际上就是一个单词; 然后对每个属性值的多个 tokens 按字典顺序排序.

2) 按处理后的决定属性值的字典序排序记录集.

3) 采用动态优先队列聚类算法 dyn-cluster(), 对排序后的记录集进行属性值聚类.

4) 为了降低字典序对错误位置的敏感性, 算法按决定属性值的字典逆序排序记录集.

5) 采用 union() 算法合并按字典序和字典逆序产生的属性值类.

相对于文献[1, 2]介绍的传统优先队列聚类算法而言, 算法 dyn-cluster() 的动态性体现在类的代表记录的动态调整. 前者类的代表记录指定为被第一次扫描的类的记录, 而后者队列中类的代表记录为属于类的最近一次被扫描的记录. 例如, 属性 $f_1$ 与 $f_2$ 相似, $f_2$ 与 $f_3$ 也相似, 但 $f_1$ 与 $f_3$ 不相似. 如果采用前者聚类, $f_1$ 为代表记录, 则类只包含 $f_1$ 和 $f_2$; 若采用后者聚类, 代表属性值依次调整为 $f_1$ 和 $f_2$, 则类包含 $f_1$, $f_2$ 和 $f_3$. 该算法假设属性值的相似具有传递性, 以便尽可能使相似的属性值聚为一类.

令 $F(f_1)$ 表示属性值 $f_1$ 对应的类, 算法 dyn-cluster() 的具体步骤如下:

**算法 1** dyn-cluster()

输入: 队列 queue 大小 $S$, 相似重复属性值的域值 $k_1$;

输出: 决定属性值类集 $F$(每个元素为一个属性值类).

```
1) to initialize t: = 1
2) for each decided attribute value f_1 do {
3) if (i = 1) then Push(f_i, queue)    // 将第一个属性置入队列
4) else {
5) if exiting q_j to make Distance(f_i, q_j) ≤ k_1 then {    // 判断f_i能否与队列中某个元素q_j相似重复
6) if not exiting F(q_j) then {    // 判断q_j是否不属于任何存在的属性值类
7) CreateNewCls(q_j, f_i, t)    // 为q_j和f_i创建新类, 赋类标号为 t
8) t: = t + 1}    // 类标号递增
9) elseMarkCls(f_i, F(q_j))    // 将属性值f_i判为类F(q_j)的成员
10) ChangeRep(q_j, f_i)}    // 将f_i代替q_j作为类F(q_j)的代表属性值, 并将f_i置队列首位
11) else {
12) Push(f_i, queue)    // 将f_i置队列首位
13) if SizeOfQueue(queue) > s then
14) PopTail(queue)}    // 如果队列元素多于s, 则将最后一个元素去掉
15) } }
```

按字典序排列属性值, 将导致一部分近似属性值排列位置相隔较远. 例如, 属性值 Seattle 和 Ceattle, 如果按字典逆序重排, 这些属性值位置将会相近. 针对这种情况, 本文提出了 union() 算法. 算法的基本思想是: 判断经过逆排序后相邻两条记录的类标号, 若类标号不同或尚不存在的两属性值的距离 $\le k_1$, 则更改它们的类标号或分配以新的类标

号. 具体步骤如下:

**算法 2**　union()

输入: 属性值逆排序结果, 决定属性值类集 $F$, 相似重复属性值的域值 $k_1$;

输出: 决定属性值类集 $F'$.

1) to initialize $t$: = MaxCls()　// 取出属性值类集中的最大类标号

2) for each decided attribute value $f_i$ do {

3) if (both $F(f_i)$ and $F(f_{i+1})$ are not exiting){　// 判断 $f_i$ 和 $f_{i+1}$ 是否都不属于任何已存在的属性值类

4) if Distance$(f_i, f_{i+1}) \leq k_1$ then {　// 判断 $f_i$ 和 $f_{i+1}$ 是否相似重复

5) $t$: = $t$ + 1　// 类标号递增

6) CreateNewCls$(f_i, f_{i+1}, t)$}　// 为 $f_i$ 和 $f_{i+1}$ 创建新类, 赋类标号为 $t$

7) }else if (only one of $F(f_i)$ and $F(f_{i+1})$ is exiting) {　// 判断 $f_i$ 和 $f_{i+1}$ 中是否只有一个属于现存的类

8) if Distance $(f_i, f_{i+1}) \leq k_1$ then{　// 将属于现存类的属性值的类标号赋值给另一属性值

9) if $F(f_i)$ not exiting then $F(f_i)$: = $F(f_{i+1})$

10) else $F(f_{i+1})$: = $F(f_i)$}

11) }else if ((both $F(f_i)$ and $F(f_{i+1})$ are exiting) and $(F(f_i) \neq F(f_{i+1}))$) then {　// 判断 $f_i$ 和 $f_{i+1}$ 是否都属于现存的不同的属性值类

12) if Distance$(f_i, f_{i+1}) \leq k_1$ then

13) SameCls$(F(f_i), F(f_{i+1}))$}}　// 将类标号为 $F(f_{i+1})$ 的所有属性值更改其类标号为 $F(f_i)$

14) }}

算法 dyn-cluster() 和 union() 都假设属性值的相似具有传递性, 可以较大限度地使可能相似的属性值聚为一类, 以便发现尽可能多的相似重复记录. 虽然这会增加属性值类的 false-positive 类型错误, 但并不影响相似重复记录检测的精度, 因为相似重复记录的聚类算法并没有传递性这一假定.

## 3　相似重复记录的取消类算法

相似重复记录检测方法的最终结果是找出相似重复的记录类. 由于相似记录一般不存在传递性, 如果算法在给出记录类时, 还给出类的代表记录(代表记录与类中其他记录都相似重复), 则既避免了 false-positive 类型错误, 又有利于重复记录的合并或清除. 代表记录的选择直接影响着相似重复记录的合并或清除的质量, 选择不当将导致正确描述实体的记录被清除掉, 而错误的记录被保留下来.

本文提出的相似重复记录聚类算法, 是采用传统的优先队列聚类算法对每个决定属性值类包含的所有记录进行聚类. 但传统的优先队列聚类算法产生的类的代表记录, 通常为类中第一次被扫描到的记录, 它并不能保证具有代表性. 针对这一问题, 本文提出了算法 class-modify(), 以便对产生的记录类进行调整.

class-modify() 的基本思想是按一定顺序调整相似重复记录类的代表记录, 使调整后的类包含更多的记录. 这样可以减少类的数量, 有利于相似重复记录的合并或清除, 并且提高了类中代表记录的代表性. 算法的具体步骤如下:

**算法 3**　class-modify()

输入: 类集 $C$, 包含各个记录类 $C_i$ 及其代表记录 $\text{rep}_i$, 迭代次数的上界 $k$, 记录可能相似重复的域值 $k_2$;

输出: 调整后的类集 $C'$, 包含各个记录类 $C_i'$ 及其代表记录 $\text{rep}_i'$.

1) to initialize ite-num, sum, maxSum with 0

2) $C_{\max}$: = GetMaxCls($C$)　// 从初始类中取出包含最多记录的类 $C_{\max}$

3) do {

4) to initialize notBest: = FALSE　// 用来判断当前类集是否需要调整, FALSE 表示不需要调整

5) for (each $r_j$ in $C_{\max}$) and ($r_j \neq \text{rep}_{\max}$) do{　// 从 $C_{\max}$ 中选择一个尚未被计算的可能代表记录 $r_j$

6) sum: = GetDupNumOfRec$(r_j, C_{\max})$　// 求出 $C_{\max}$ 中与 $r_j$ 相似重复的记录数量

7) for each $\text{rep}_i$ and $\text{rep}_i \neq \text{rep}_{\max}$ do{

8) if distance$(r_j, \text{rep}_i) \leq k_2$ then　// 判断 $C_i$ 是否为 $r_j$ 的可能相似重复类

9) sum: = sum + GetDupNumOfRec$(r_j, C_i)$}　// 若是, 则求出 $C_i$ 中与 $r_j$ 相似重复的记录数量

10) if sum > maxSum then {　// 找出有最大数量的记录与其相似重复的 $r_j$, 并保存至 maxId

11) maxSum: = sum; maxId: = $r_j$}}

12) if $|C_{\max}| <$ maxSum then {　// 判断与 maxId 相似重复的记录数量是否大于当前最大类 $C_{\max}$ 的数量

13) $C_{\text{new}}$: = CreateClsWithRep(maxId)　// 若是, 则以 maxId 为代表记录创建一新类 $C_{\text{new}}$

14) $C'$: = Modify($C$)　// 对原类集 $C$ 进行调整, 去掉属于 $C_{\text{new}}$ 的所有记录, 得到类集 $C'$

15) notBest: = TRUE; $C_{max} = C_{new}$} // $C_{new}$ 包含了新记录, 需要重新迭代一次, 以产生更大的类

16) ite_num: = ite_num + 1 // 迭代次数递增

17) }while ((ite_num < $k$) and (notBest)) // 如果迭代次数 = $k$, 或 $C_{max}$ 已最大, 则终止迭代

18) class_modify($C'$) // 对余下类集 $C'$ 递归调整

算法 3 的步骤 3)～ 17) 是对类 $C_{max}$ 的调整, 其中步骤 8) 定义了 $r_j$ 的可能相似类; 步骤 18) 是对余下的类进行递归调整

令所有可能相似类包含的记录总数为 $n$, 调整后的类总数为 $c$, 每个类的记录数为 $a_i$, 则算法 3 的最大记录比较次数为 $\left(n^2 - \sum_{1\le i,j<c}^{i\neq j} a_i a_j\right)$. 其中 $n$ 为某个属性值类的子集大小, 一般较小. 因此算法 3 的时间复杂性是可行的.

## 4 实验分析

本实验从记录比较次数(反映算法的时间开销)、检测精度和产生的类数量 3 个方面来评价算法性能. 由于本算法不存在 false-positive 类型错误, 在评价算法精度时, 着重评价算法的相似重复记录的检出率. 在相同的检出率下, 类数量越少, 越有利于重复记录的清除, 类的质量就越高. 本实验将改进方法与 Priority queue strategy 算法作了比较.

相似重复记录的分布影响着算法性能. 反映相似重复记录分布的指标有覆盖率 $C$ 和集中度 $F$. 其中覆盖率是指数据库中所有相似重复记录的数量与数据库尺寸之比, 集中度是指平均每个重复类包含的记录数量. 实验中首先构造了数据库产生器, 它能随机产生任意数量和任意分布的测试数据. 测试数据包含人名、家庭住址、邮编、email 信箱等常用数据属性.

### 4.1 相似重复记录的分布对算法性能的影响

取实验数据规模为 2 万条记录, 队列大小为 4(文献[2]推荐的队列大小). 图 1 中(a) 和(b) 是固定相似重复记录的覆盖率 $C$ = 20.7%, (c) 和(d) 是固定集中度 $F$ = 6. (a) 和(c) 表明, 本算法的记录比较次数随着集中度或覆盖率的增大而直线增长. 这是因为集中度增大, 类调整算法中的记录比较次数增多; 而覆盖率越大, 算法 dyn_cluster() 产生的属性值子类数量越多, 导致记录聚类和类调整算法的记录比较次数都增加. 由(b) 和(d) 可知, 算法的检出率随着集中度的增大而缓慢下降, 基本上不随重复记录的覆盖率增大而变化. 这主要是因为本实验固定了队列大小的缘故.

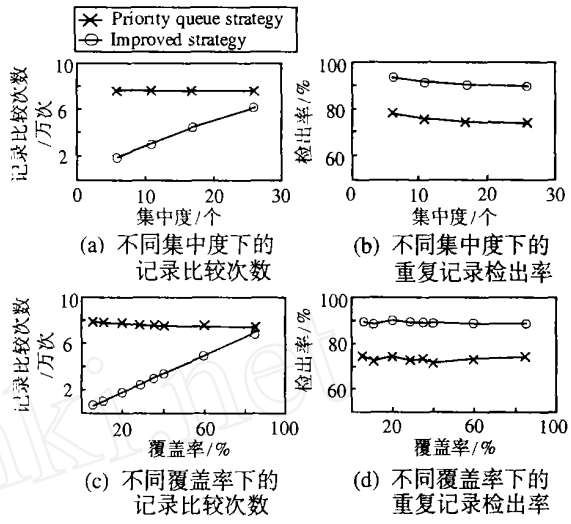


图 1 相似重复记录的分布对算法性能的影响

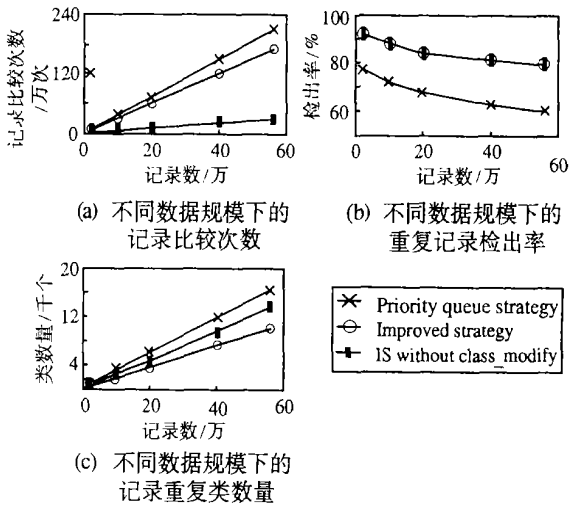


图 2 数据规模对算法性能的影响

### 4.2 数据规模对算法性能的影响

取近似重复记录的分布参数 $C$ = 20.8%, $F$ = 26; 队列大小为 4. 由 4.1 节的实验结果可知, 算法的记录比较次数会随 $F$ 的增大而直线上升. 因此在重复记录分布下的算法性能具有较强的说服力. 图 2 表明, 随着数据规模的增大, 本算法的记录比较次数、检出率和产生的重复类数量都优于 Priority queue strategy 算法.

图 2(b) 显示, 包含 class_modify() 的重复记录聚类算法与不含 class_modify() 的算法的重复记录检出率几乎一致; 从(a) 和(c) 可以看出, 前者产生的类数量小于后者, 但前者的记录比较次数较大.

(下转第 813 页)

*Triennial World Congress* [C]. San Francisco, 1996: 203-208.

[3] 谢永芳, 桂卫华, 吴敏. 基于线性矩阵不等式的分散 $H_2/H_\infty$ 控制的次优设计[J]. *自动化学报*, 2000, 26(2): 263-266.
(Xie Y F, Gui W H, Wu M. The Suboptimal Design of Decentralized $H_2/H_\infty$ Control Based on Linear Matrix Inequality [J]. *Acta Automatica Sinica*, 2000, 26(2): 263-266.)

[4] Ikeda M, Zhai G, Fujisaki Y. Decentralized $H_\infty$ Controller Design for Large Scale Systems: A Matrix Inequality Approach Using Homotopy Method [J]. *Automatica*, 2001, 37(4): 565-573.

[5] 桂卫华, 陈宁, 吴敏. 不确定关联大系统鲁棒分散可靠 $H_\infty$ 控制[J]. *控制理论与应用*, 2002, 19(6): 923-926.
(Gui W H, Chen N, Wu M. Robust Decentralized Reliable Control for Uncertain Large-scale Systems [J]. *Control Theory and Applications*, 2002, 19(6): 923-926.)

[6] 甘永梅, 王兆安. 不确定性关联大系统的分散鲁棒状态反馈 $H_\infty$ 控制[J]. *控制理论与应用*, 2002, 19(2): 297-301.
(Gan Y M, Wang Z A. Decentralized State Feedback Robust $H_\infty$ Control Design for Uncertain Interconnected Large-scale Systems [J]. *Control Theory and Applications*, 2002, 19(2): 297-301.)

[7] Scorletti C, Duc G. An LMI Approach to Decentralized $H_\infty$ Control [J]. *Int J Control*, 2001, 74(3): 211-224.

[8] 程储旺. 不确定性时滞大系统的分散鲁棒 $H_\infty$ 控制[J]. *自动化学报*, 2001, 27(3): 361-366.
(Cheng C W. Decentralized Robust $H_\infty$ Control for Uncertain Delay Large-scale Systems [J]. *Acta Automatica Sinica*, 2001, 27(3): 361-366.)

[9] Cao Y Y, Sun Y X, Mao W J. Output Feedback Decentralized Stabilization: LMI Approach [J]. *Systems and Control Letters*, 1998, 35(3): 183-194.

[10] Boyd S P, Chaou L E, Feron E, et al. *Linear Matrix Inequalities in Systems and Control Theory* [M]. Philadelphia: SIAM, 1994.

[11] Iwasaki T, Skelton R E. All Controllers for the General $H_\infty$ Control Problem: LMI Existence Conditions and State Space Formulas [J]. *Automatica*, 1994, 30(8): 1307-1317.

(上接第 808 页)

## 5 结　论

相对于以往检测相似重复记录的方法, 本文改进方法的优点主要体现在以下 3 个方面: 1) 该方法按决定属性值划分记录集, 再对各个属性值类进行记录聚类, 大大降低了时间开销; 2) 在决定属性值聚类时, 假定属性值的相似具有传递性, 并合并了字典逆排序的聚类结果, 显著提高了算法的检出率; 3) 对聚类结果进行调整, 减少了记录类数量, 提高了聚类质量.

## 参考文献(References)

[1] 邱越峰, 田增平, 季文赟, 等. 一种高效的检测相似重复记录的方法[J]. *计算机学报*, 2001, 24(1): 69-77.
(Qiu Y F, Tian Z P, Ji W Y, et al. An Efficient Approach for Detecting Approximately Duplicate Database Records [J]. *Chinese J of Computers*, 2001, 24(1): 69-77.)

[2] Monge A E, Elkan C P. An Efficient Domain-independent Algorithm for Detecting Approximately Duplicate Database Records [A]. *Proc of DMKD'97* [C]. Tucson Arizona, 1997: 23-29.

[3] Hernandez M A, Stolfo S G. The Merge/Purge Problem for Large Databases [A]. *Proc ACMS IGMOD Int Conf on Management of Data* [C]. California, 1995: 127-138.

[4] Lee M L, Lu H J, Ling T W, et al. Cleansing Data for Mining and Warehousing [A]. *10th Int Conf on Database and Expert Systems Applications* [C]. Florence, 1999: 751-760.

[5] Jin L, Li C, Mehrotra S. Efficient Record Linkage in Large Data Sets [A]. *Proc 8th Int Conf on Database Systems for Advanced Applications* [C]. Kyoto, 2003: 137-148.

[6] Gravano L, Ipeirotis P G, Jagadish H V, et al. Approximate String Joins in a Database (almost) for Free [A]. *Proc of the 27th VLDB Conf* [C]. Roma, 2001: 491-500.