软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2016,27(7):1671−1684 [doi: 10.13328/j.cnki.jos.005037]

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

基于函数依赖与条件约束的数据修复方法*

金澈清, 刘辉平, 周傲英

(华东师范大学 计算机科学与软件工程学院 数据科学与工程研究院,上海 200062)

通信作者: 刘辉平, E-mail: hpliu@stu.ecnu.edu.cn

摘 要: 随着经济与信息技术的发展,在许多应用中均产生大量数据.然而,受硬件设备、人工操作、多源数据集成等诸多因素的影响,在这些应用之中往往存在较为严重的数据质量问题,特别是不一致性问题,从而无法有效管理数据.因此,首要的任务就是开发新型数据清洗技术来提升数据质量,以支持后续的数据管理与分析.现有工作主要研究基于函数依赖的数据修复技术,即以函数依赖来描述数据一致性约束,通过变更数据库中部分元组的属性值(而非增加/删除元组)来使得整个数据库遵循函数依赖集合.从一致性约束描述的角度来看,函数依赖并非是唯一的表达方式,还存在其他表达方式,例如硬约束、数量约束、等值约束、非等值约束等.然而,随着一致性约束种类的增加,其处理难度也远比仅有函数依赖的场景要困难.考虑以函数依赖与其他一致性约束共同表述数据库的一致性约束,并在此基础上设计数据修复算法,从而提升数据质量.实验结果表明,所提方法的执行效率较高.

关键词: 数据质量;数据修复;函数依赖;条件约束;等价类

中图法分类号: TP311

中文引用格式: 金澈清,刘辉平,周傲英.基于函数依赖与条件约束的数据修复方法.软件学报,2016,27(7):1671−1684. http://www.jos.org.cn/1000-9825/5037.htm

英文引用格式: Jin CQ, Liu HP, Zhou AY. Functional dependency and conditional constraint based data repair. Ruan Jian Xue Bao/Journal of Software, 2016,27(7):1671−1684 (in Chinese). http://www.jos.org.cn/1000-9825/5037.htm

Functional Dependency and Conditional Constraint Based Data Repair

JIN Che-Qing, LIU Hui-Ping, ZHOU Ao-Ying

(Institute for Data Science and Engineering, School of Computer Science and Software Engineering, East China Normal University, Shanghai 200062, China)

Abstract: Along with the development of economy and information technology, a large amount of data are produced in many applications. However, due to the influence of some factors, such as hardware equipments, manual operations, and multi-source data integration, serious data quality issues sunch as data inconsistency arise, which makes it more challenging to manage data effectively. Hence, it is crucial to develop new data cleaning technology to improve data quality to better support data management and analysis. Existing work in this area mainly focuses on the situation where functional dependencies are used to describe data inconsistency. Once some violations are detected, some tuples must be changed to suit for the functional dependency set via update (instead of insert or delete). Besides functional dependency, there also exist other types of constraints, such as the hard constraint, quantity constraint, equivalent constraint, and non-equivalent constraint. However, it becomes more difficult when more inconsistent conditions are involved. This paper addresses the general scenario where functional dependencies and other constraints co-exist. Corresponding data repair algorithm is designed to improve the data quality effectively. Experimental results show that the proposed method performs effectively and efficiently.

* 基金项目: 国家重点基础研究发展计划(973)(2012CB316203); 国家自然科学基金(61370101, U1501252, 61532021); 上海市教委科研创新重点项目(14ZZ045)

Foundation item: National Basic Research Program of China (973) (2012CB316203); National Natural Science Foundation of China (61370101, U1501252, 61532021); Innovation Program of Shanghai Municipal Education Commission (14ZZ045)

收稿时间: 2015-10-09; 修改时间: 2016-01-12; 采用时间: 2016-02-22; jos 在线出版时间: 2016-03-22

CNKI 网络优先出版: 2016-03-22 13:23:32, http://www.cnki.net/kcms/detail/11.2560.TP.20160322.1323.006.html

Key words: data quality; data repair; functional dependency; conditional constraint; equivalence class

信息技术极大地推动了国民经济与社会的发展.随着数据采集技术的不断进步以及新型应用的不断涌现,待管理数据的规模也飞速增长.能否高效地管理、分析数据业已成为衡量一个国家综合国力的重要指标.2010年底,美国总统科技顾问委员会(PCAST)提出一份报告,建议"由于数据规模呈指数增长,...,各联邦机构都需要制定应对'大数据'的策略"[1].2008 年 9 月和 2011 年 2 月,《自然》和《科学》分别登载专刊,阐述大数据管理的迫切性和重要性,并展望发展前景[2,3].学界惯以"3V"来描述大数据的特征,即:量大(volume)、多样(variety)、实时(velocity).数据由多源产生,数据形态可以不同,数据总量庞大.以海洋监控应用为例,需要在不同地点布置传感器节点(工业机器人)来采集数据,并最终进行综合分析.

然而,"量大"并不意味着"质高";从某种意义上讲,量大往往意味着出现质量问题的几率增加了[4].因为数据可以从多个来源生成,数据之间往往具有关联性.在局部看来是高质量的数据,放到全局视角来看,就会出现新的问题.据报道,大型机构所产生的数据中 60%都是冗余的[5].此外,RFID 技术现已广泛应用到物流等多个行业之中,而 RFID 数据的准确度仅达到 60%~70%[6].国外权威机构统计表明,除非经过非常精心的设计与对待,企业里面的数据错误率会在 1%~5%之间[7].质量低下的信息会带来严重的社会问题.例如,医生基于低质量的医疗信息无法准确、迅速地为病人就诊,甚至可能引发医疗纠纷.

因此,定义数据集合的质量高低并且在知悉数据质量不高时能够执行操作来提升数据集合的质量,就显得非常重要.函数依赖能够表达一个关系中两组属性(集合)之间的映射关系,已经被广泛应用于数据库规范化设计当中,它也是一种重要的描述数据一致性的手段.令 R 表示一个关系,A 和 B 分别是 R 上的两个属性集合,且 $A\rightarrow B$ 表示一个函数依赖,则对于任意两个 R 中的元组,当其在属性集合 A 上的值相等时,则在属性 B 上也会相等.事实上,可以在一个关系 R 上定义一个函数依赖集合,包含多个函数依赖.如果对于关系 R 来说,该集合中的所有函数依赖都能够被满足,则认定该关系的数据质量达到要求;反之,则认为存在数据质量问题,需要进一步提升.

典型的数据修复策略可以分为两类.一类是直接删除不符合要求的记录,使得剩余记录能够符合所有给定的函数依赖.由于删除记录时剩余数据集合中存在的错误不会增加,因此总可以不断迭代执行以完成数据修复,即:首先找到违背函数依赖约束的元组集合,然后随机删除一条元组,再检测剩余数据集合.这种方法尽管简单,但却会丢失很多信息.例如,一条包含数十个属性的元组,可能仅仅由于某一个属性值不符合规范而被整体删除,一些有益信息就丢失了.另一种方式是并不增加/删除任何记录,而是仅仅修改某些字段,使得整个集合能够符合预设的一致性要求.这种方式能够最大限度地保留数据集合的原始信息,也是目前的研究重点.但是,后一种处理方式做起来并不容易,因为修改某条元组的属性值,有可能又与现有的其他元组之间产生新的不一致性.所以,增加了技术难度.

例 1:图 1 的上部表格是一个包含 4 条学生记录的小型关系实例.在该实例上共有 3 个函数依赖,例如:邮编→城市,省份.然而,这个实例中存在数据不一致性现象.例如,t_1 和 t_2 的邮编相同,根据该条函数依赖关系,则这两个元组的城市和省份也应该相同.但在本实例中却并不相同.因此,可以修改 t_2 中的相关字段,使之与 t_1 相同.如左下方的修复策略 1 所示.

尽管函数依赖非常重要也非常有效,但是仍然存在一些无法由函数依赖表达的重要约束条件,包括硬约束、与数量相关的约束、等值约束、非等值约束等.

- 硬约束.硬性指定某一个单元格的值,例如"学号为 001 的同学来自杭州".
- 数量约束.此类型中最为典型的约束就是唯一性约束.即某个属性或者属性值中的信息都是唯一的.除此之外,也可以包括大于 1 的约束条件,例如:这个学校中的杭州人不超过 2 人.
- 等值约束.说明特定条件之下的属性值是相等的.例如,"李四和王五是一个城市的".
- 非等值约束.除了相等之外的二元操作,包括大于、小于等这些不等于条件下属性值一定不相等.例如:"这个学校有两个张三,但是邮编不同".

可以将这些函数依赖之外的约束条件存放于外部知识库.在数据修复过程中,不仅要使得这些修复符合预先设定的函数依赖集合,也必须满足在外部知识库之中的约束条件.

原始的输入实例

	学号	姓名	城市	省份	邮编
t_1	001	张三	杭州	浙江	310027
t_2	002	李四	宁波	安徽	310027
t_3	003	张三	杭州	浙江	310014
t_4	004	王五	温州	浙江	325000

函数依赖集合:
学号→姓名,城市,省份,邮编
邮编→城市,省份
姓名→城市
外部约束条件:
学号“002”和“004”的同学的城市相同
这个学校的杭州人不超过 2 人

修复策略 1

	学号	姓名	城市	省份	邮编
t_1	001	张三	杭州	浙江	310027
t_2	002	李四	杭州	浙江	310027
t_3	003	张三	杭州	浙江	310014
t_4	004	王五	温州	浙江	325000

修复策略 2

	学号	姓名	城市	省份	邮编
t_1	001	张三	杭州	浙江	310027
t_2	002	李四	温州	浙江	325000
t_3	003	张三	杭州	浙江	310014
t_4	004	王五	温州	浙江	325000

Fig.1 An example of data repair
图 1 数据修复样例

例 2:图 1 所示的修复策略 1 能够满足所有函数依赖,但却不能满足外部约束条件.该约束表明,杭州的学生不超过 2 人,但是修复策略 1 中却出现了 3 个杭州.因此,修复策略 1 违背了外部约束条件.可以修改 t_2 中的字段,从而既满足函数依赖集合,又满足外部知识库中的约束,参见修复策略 2.

然而,针对兼具函数依赖集合与外部知识库的关系数据库的数据修复工作并不容易.当前的研究工作主要集中于如何利用函数依赖进行修复.这些工作研究各种各样的优化目标,分析其复杂度,并且设计相应的算法[8−12].也有学者研究修复的多样性问题,提出了一种返回 k 种修复方案的方法,且修复方案之间的差异性比较大,从而使得整体修复方案具有多样性,有利于项目实施者进行挑选[13,14].仅有少数工作考虑到了除函数依赖之外的其他一致性约束条件,例如,文献[8]考虑了一种特殊的包含关系 IND,指明某一个关系中某一个属性的所有值均需在另外一个关系的某个属性值空间之中,类似于关系数据库中的外键关系;文献[12]考虑到了硬约束,即某些单元格的值是强制设定的.然而,在这些工作中所考虑的一致性约束条件仍然比较有局限性,无法简单扩展以解决其他一致性约束条件.在本文中,我们研究一种数据质量提升方法,既能够应对函数依赖,也能够应对包括硬约束、数量约束、等值约束和非等值约束等多种一致性约束条件.

本文的主要贡献如下.

首先,鉴于传统的基于函数依赖集合的数据修复方案在数据一致性表达上具有局限性,因而需要引入新的约束条件,丰富其表达能力,并在此基础上研究数据修复问题.

其次,本文提出了一种新颖的数据修复算法,该算法采用增量式的修复策略,在初始化阶段首先产生部分正确的单元格,在后续过程中再不断地修复其他单元格.

最后,我们在模拟数据集合上进行了验证,实验结果表明,本文所提算法具有较好的可行性以及较佳的执行效率.

本文第 1 节回顾相关工作.第 2 节正式定义问题.第 3 节给出解决方案.第 4 节通过实验进行验证.最后在第 5 节总结全文.

1 相关工作

近年来,数据质量问题得到了广泛的关注.李建中等人意识到劣质数据随着大数据的爆炸性增长而来,认为数据可用性是大数据的一个重要方面,需要从战略高度进行思考[15].当待处理的数据质量被检测出来存在较大问题时,必须及时进行修复.数据的修复主要针对实体同一性和一致性问题而来.所谓的实体同一性错误是指在数据库中存在多条描述同一实体的记录,但是不同记录所记载的信息存在冲突,可以采取数据融合技术进行修复[16,17].一致性错误是指数据库违背了预先定义好的一组规则.本文主要针对数据的一致性错误.一般来说,数

据一致性的自动修复工作可以分为基于统计学的方法和基于语义规则的方法.

基于统计学的修复方法充分考虑了概率分布等因素,并以此作为依据进行修复.文献[18]提出了一种基于概率的数据一致性错误修复方法.该方法首先假定存在一个满足一致性约束的合理修复的空间,然后依据合理修复的概率分布进行抽样,最后会用抽取出来的合理修复完成数据的修复.文献[6,19,20]分别以 RFID 和传感器网络为背景,针对利用统计学定义的数据一致性错误,提出了修复数据一致性错误的方法.

基于语义规则的方法得到了更多的研究.文献[8,9]的目标是通过最少的更新操作来修复数据质量,其优化目标就是基数最小策略.文献[8]分析了计算复杂度,提出了一种贪心算法进行处理.文献[9]则设计了一种近似最优的策略来解决这个问题.此外,另有一组学者则尝试利用集合最小化策略来进行修复[10,11].文献[12]提出了基数集合最小化策略,平衡了基数最小化策略的"最少修改"和集合最小化策略的"必要修改"两点.同时,文献[12]中还研究了硬约束,即部分单元格的值是预先设定好的情况.文献[21]也研究了基于函数依赖的一致性修复问题.文献[22]研究针对数据和约束修复的统一模型,不仅考虑在修复过程中最小化数据库更新,也考虑约束条件随时间演化的情况.这些约束条件也是由函数依赖所决定的.文献[13]同时考虑了修复的多样性问题,针对一个低质数据库实例,力图返回多个修复,而且这些修复之间具有较大的差异性.文献[14]则尝试在分布式数据库上对数据进行修复.

从以上分析可以看出,基于语义规则的方法得到了更为广泛的研究.在这些工作中,绝大多数均将函数依赖作为一致性约束的唯一方式.仅有文献[8,12]考虑到了其他依赖关系,即包含关系和硬约束关系.但是硬约束关系相对来说比较简单,无法表达更为复杂的语意;包含关系主要适用于多表场景.因此,由于约束条件和优化目标不一致,文献[8,12]的方法很难直接应用到本文的工作中.鉴于此,本文扩充了其他的约束语义,并基于此进行修复.

2 模型与定义

2.1 函数依赖

令 R 表示一个关系,它包含 m 个属性;$Attrs(R) = (A_1,...,A_m)$表示 R 上的属性集合,$Dom(A)$表示一个给定属性 A 的域.令 I 表示关系 R 的一个实例,包含若干元组,各元组均属于域 $Dom(A_1)\times...\times Dom(A_m)$.令 $Dom_I(A)$表示属性 A 的空间,它包括所有出现在实例 I 中的 A 属性值.假设 I 中的每个元组均有一个标识符,即使元组的其他属性都发生变更,该标识符也不会改变.令 $TIDs(I)$表示在实例 I 中的所有元组的标识符的集合.令 $t[A]$表示元组 t 的一个单元,其中,$A\in Attrs(R)$,$t\in TIDs(I)$.每一个单元 $t[A]$由元组 t 以及属性 A 所确定.

在 R 上定义一个函数依赖集合,包含多个函数依赖.对于两个属性集合 X 和 Y,它们均属于 $Attrs(R)$.基于实例 I 的一个函数依赖 $X\rightarrow Y$ 被表示为 $I\trianglelefteq X\rightarrow Y$.换言之,对于实例 I 中的任意两个元组 t_1 和 t_2,如果 $t_1[X]=t_2[X]$成立,则 $t_1[Y]=t_2[Y]$必然成立.令Σ表示基于关系 R 的函数依赖集合.我们假设Σ是正则最小化的[22].每个函数依赖均可以被描述为如下的形式:$X\rightarrow A$,其中,$X\subset Attrs(R)$,且 $A\in Attrs(R)$.

2.2 外部知识库

函数依赖集合是一种描述数据质量的重要方式,但却无法描述所有情况.因此还需要以其他方式进行描述,包括硬约束、数量约束、等值约束、非等值约束等.

定义 1(硬约束,hard constraint,HC(S,v)). 给定一个单元格集合 S 中各单元格的值必须为 v.

例如,"学号为 001 的同学的城市是杭州"可以表示为 HC(t_1[城市],杭州).

定义 2(数量约束,quantity constraint,QC(S,v,τ)). 给定一个单元格集合 S,约定在这些单元格的值被设置为 v 的数量的上限τ.

例如,"这个学校的杭州人不超过 2 人"可以表示为 QC(I[城市],杭州,2).其中,I[城市]表示数据库实例 I 的城市属性的所有单元格.

定义 3(等值约束,equality constraint,EC(S)). 给定一个单元格集合 S,约定在这些单元格中值相等.

参数S是一个固定的集合.例如,“学号是001和002的同学的城市相同”可以表示为:EC($\{t_1$[城市],t_2[城市]$\}$).在这里,集合$\{t_1$[城市],t_2[城市]$\}$包含两个指定位置的单元格.

定义 4(非等值约束,non-equality constraint,NC(S)). 给定一个单元格集合 S,约定在这些单元格中值均不相等.

与等值约束类似,单元格集合 S 也是固定的集合,例如,“学号是 001 和 002 的同学的邮编不同”可以表示为NC($\{t_1$[邮编],t_2[邮编]$\}$).

定义 5(干净数据库,clean database). 给定一个数据库,如果该数据库中的所有单元格既满足所有的函数依赖也满足所有的条件约束,则该数据库是干净的.

例如,图 1 中的修复策略 2 生成的数据库是一个干净的数据库.

2.3 数据库修复

给定一个函数依赖集合Σ和外部约束集合Ω,如果数据库实例 I 违背了Σ中的任意一个函数依赖或者Ω中的任意一个约束条件,则称 I 相对于(Σ,Ω)不一致.一般来说,针对一个不一致实例 I 的修复会产生另外一个同时满足Σ和Ω的实例 I'.如前所述,本文中所研究的修复是指对于数据集合的修改,而非添加或者删除数据.令$\Delta(I,I')$表示在修复 I'中值发生变更的所有单元格的集合.

事实上,针对一个给定的数据库实例的修复策略往往并非唯一,而是多元化的.目前,典型的修复目标有 3 种:集合最小化修复(set-minimal repair)、基数最小化修复(cardinality-minimal repair)和基数集合最小化修复(cardinality-set-minimal repair).

定义 6(集合最小化修复)[10,11]. 一个针对数据库实例 I 的修复 I'是集合最小化的,当且仅当不存在一个修复 I'',使得$\Delta(I,I'')\subset\Delta(I,I')$,并且对于任一单元格 $C\in\Delta(I,I'')$,$I''(C)=I'(C)$.

此外,基数最小化修复返回一个修复 I',它是针对原始数据库实例 I 的最小修复,换言之,不存在另外一个修复 I'',使得$|\Delta(I,I'')|<|\Delta(I,I')|$.基数集合最小化修复则返回一个修复 I',当且仅当不存在任一修复 I'',使得$\Delta(I,I'')\subset\Delta(I,I')$.本文研究集合最小化修复.

3 解决方案

本节详细介绍数据修复方案.我们首先介绍算法的总体框架,然后依次介绍这个框架中的各个重要函数,最后分析算法的性能.

3.1 算法框架

本节介绍算法的框架.“等价类(equivalence class)”是由文献[8]所提出的一个关键概念,在后续多个文献中延续使用.所谓的等价类是指,在关系数据库实例中位于同一列的若干单元格所构成的集合,且这些单元格中的值相同.此外,在函数依赖集合的作用下,这些等价类还可以进行合并.图 2 给出了一个等价类例子.图 2(a)是原始的数据库实例,它有 3 个属性(A,B,C),包含 3 条元组.它有两条函数依赖$\{A\rightarrow C,B\rightarrow C\}$.检查各列数据,可以将整个数据库实例划分为如下 6 个等价类,如图 2(b)中各粗体矩形框所示.在每个等价类中的值均相等.通过检验各条函数依赖,可以发现等价类{3,3}和{5}必须合并成{3,3,5},如图 2(c)所示.鉴于这个新等价类中的数值并不完全相同,可以认定本数据库实例中存在数据质量问题.

	A	B	C
t_1	1	4	3
t_2	1	2	3
t_3	6	2	5

$\Sigma=\{A\rightarrow C,B\rightarrow C\}$

(a) 原始关系数据库实例

	A	B	C
t_1	1	4	3
t_2	1	2	3
t_3	6	2	5

$\Sigma=\{A\rightarrow C,B\rightarrow C\}$

(b) 检查各列后的等价类集合

	A	B	C
t_1	1	4	3
t_2	1	2	3
t_3	6	2	5

$\Sigma=\{A\rightarrow C,B\rightarrow C\}$

(c) 查验 FD 后的等价类集合

Fig.2 Functional dependency evaluation via equivalence classes

图 2 构建等价类以检测函数依赖

本文不仅研究函数依赖,还研究其他外部约束问题.因此,传统的方法无法胜任.我们提出了一种新的处理算法,其框架如算法 1 所示.给定待修复的数据库实例 I,函数依赖集合Σ以及外部知识库Ω,首先调用Initialize(Σ,Ω)函数根据函数依赖和外部知识库中的硬约束、等值约束以及非等值约束对干净数据库实例 C 和等价类集合 E 进行初始化,即将满足这些约束的单元格先放入 C 使其不可变,则最后 C 必然满足硬约束、等值约束和非等值约束(第 1 行).然后对于待修复数据库实例 I 中的每一个单元格 c,将其依次加入 C,并对该单元格 c 增量构建等价类形成新的等价类集合.为了保证修复的多元性,可随机选取 I 中的单元格(第 2 行~第 5 行).当 I 中所有单元格都被插入到 C 后,C 必然满足硬约束、等值约束、非等值约束、数量约束以及函数依赖.最后对于 C 中的变量值,我们对其进行确定使得 C 是干净的(第 6 行).

可以看出,该算法的特点是在初始化之后的过程中,迭代随机访问各个单元格,但在检测到数据不一致现象时,只会修改该单元格的值,而不去变动原有的存放在 C 中的单元格的值.一旦单元格存在于 C 中,且其是干净的,则最后这些单元格必然也是干净的.

算法 1. 算法总体框架 Framework.

输入:数据库实例 I,函数依赖集合Σ,外部知识库Ω;

输出:干净的数据库实例 C.

1. (C,E)←Initialize(I,Σ,Ω); //初始化阶段,生成单元格集合 C 和等价类集合 E,参见第 3.2 节
2. **for each** 单元格 $c\in I\backslash C$ **do** //c 可以在 $I\backslash C$ 中随机选取
3. E←IncreBuildEquivRel(E,Σ,C,c); //增量式构建新的等价类集合 E,参见第 3.3 节
4. C←$C\cup\{c\}$; //将 c 插入到单元格集合 C 中
5. **end for**
6. VarRepair(C); //变量值修复,参见第 3.4 节
7. **return** C

3.2 初始化阶段

算法 1 的第 1 个步骤就是基于数据库实例 I、函数依赖集合Σ和外部知识库Ω进行初始化,生成初始实例 I' 和等价类集合 E.考虑到外部知识库中存在多种约束类型,在本阶段主要处理硬约束(HC)、等值约束(EC)、非等值约束(NC)和函数依赖(FD).具体处理如下:

- 硬约束 HC.硬约束限定给定单元格的值.如果这个单元格恰巧从属于某一个等价类的话,则需要检查这个等价类中的其他单元格,若值不等于硬约束所给定的值,则需要进行修复.
- 等值约束 EC.对于等值约束而言,给定的若干个单元格的值必须相等.因此,需要将这些单元格划分到一个等价类中去.
- 非等值约束 NC.对于非等值约束来说,给定单元格集合中任意两个单元格的值均不相等.因此,可以将每个单元格分别设定一个等价类.如果存在值相等的单元格,先将其值均设为变量.
- 函数依赖 FD.函数依赖描述属性集合之间的依存关系.在初始化阶段,查验预设的函数依赖集合有助于合并等价类.

具体步骤如算法 2 所示.首先,生成一个空的等价类集合 E(第 1 行).对于Ω中的每一个硬约束,将其单元格放入同一个等价类中并设置相应的值,然后加入到 E 中(第 2 行~第 6 行).对于Ω中的每一个等值约束,将该约束中的所有单元格划分到同一个等价类中,保证其值相等(第 7 行~第 10 行).对于Ω中的每个非等值约束,将该约束中的每个单元格都当作独立的等价类,并将值相等的单元格的值设为变量(第 11 行~第 15 行).对于Σ中的每条函数依赖 F:$B\rightarrow A$,考察每对元组(t,t'),判断是否有必要合并等价类.检查的规则就是判断检查 $t[B]$和 $t'[B]$是否相等,如果相等,则有必要合并 $t[A]$和 $t'[A]$所对应的等价类.在这个过程中,可能会遇到各个参与的等价类值不相等的情况,则需要在不违背硬约束的前提下进行修改.如果无法设置合适的值,则说明约束条件和函数依赖相矛盾,该数据库实例不可修复(第 16 行~第 19 行).之后,生成单元格集合 C,并最终返回 C 和等价类集合 E(第 20 行~第 21 行).

算法 2. 初始化阶段 Initialize.

输入:数据库实例 I,函数依赖集合Σ,外部知识库Ω;

输出:单元格集合 C,等价类集合 E.

1. $E\leftarrow\varnothing$;
2. **for each** 硬约束 HC(S,v)$\in\Omega$ **do** //处理硬约束 HC
3. 将 S 中所有单元格的值设置为 v;
4. 创建一个包含该 S 中所有单元格的等价类 e;
5. $E\leftarrow E\cup\{e\}$; //将 e 插入到 E 中去
6. **end for**
7. **for each** 等值约束 EC(S)$\in\Omega$ **do** //处理等值约束 EC
8. 创建一个包含该等值约束覆盖的所有单元格的等价类 e;
9. $E\leftarrow E\cup\{e\}$; //将 e 插入到 E 中去
10. **end for**
11. **for each** 非等值约束 NC(S)$\in\Omega$ **do** //处理非等值约束 NC
12. 针对该非等值约束所覆盖的各个单元格均创建独立的等价类,标记为 e_1,e_2,...;
13. 将值相等的单元格的值设为变量;
14. $E\leftarrow E\cup\{e_1,e_2,...\}$; //将新创建的等价类添加到 E 中去
15. **end for**
16. **for each** 函数依赖 $F\in\Sigma$,F:$B\rightarrow A$ **do** //处理函数依赖集合,尝试合并等价类
17. 对于 E 中所涉及的任意元组对(t,t'),如果 $t[B]$和 $t'[B]$属于同一等价类但 $t[A]$和 $t'[A]$属于不同的等价类,则合并 E 中包含 $t[A]$和 $t'[A]$的等价类;
18. 如果在等价类合并过程中,各参与方的值不相同,则可以在不违背硬约束的前提下进行修改;
19. **end for**
20. $C\leftarrow E$ 中所有等价类所包含的所有单元格;
21. **return** (C,E);

3.3 等价类增量构建

初始化阶段仅处理、修复数据库实例 I 中的部分单元格,仍然需要进一步操作来修复其余的单元格.在本文所提出的总体框架中,需要迭代式地处理剩余单元格,直到所有单元格均被访问完毕.在这个过程中,增量式地构建等价类集合就是一个关键性操作,其目的在于高效地生成基于 $C\cup\{c\}$的等价类集合,其中 C 是原始的单元格集合,c 是即将加入的单元格.在这个函数中,比较重要的一个步骤是测试数量约束 QC.如果新单元格 c 加入到已有的等价类之中,会导致违背某一条数量约束 QC 时,则需要将该单元格单独设置为一个等价类,且其值为变量.

算法 3 描述了具体步骤.首先,如果在检验等价类集合 E'(即输入等价类集合 E)之后发现存在等价类 e,这个等价类的值与单元格 c 中的值恰巧相等且 c 与 e 处于同一属性,则需要将 c 与 e 进行合并.而这个合并操作又可能会引发后续的等价类合并操作.因此,需要再次检查函数依赖集合进行判断.鉴于潜在的等价类合并操作都是由于新加入的单元格所导致的,因此仅需要检测与这个插入元组所在的等价类相违背的等价类并将其合并直到 E'中所有等价类都满足函数依赖.更新操作与算法 2 中的第 16 行类似(第 2 行~第 5 行).执行上述操作可能导致违反某一个现存的数量约束 QC,此时,则需要将单元格 c 单独作为一个等价类,将其值设置为变量,再加入到输入等价类 E 之中(第 6 行~第 7 行).此外,合并之后的等价类可能不干净,即其中的单元格的值不相等,则说明新插入的单元格 c 不满足条件.此时,如果 c 属于之前的等价类集合 E 中的一个等价类中且该等价类不是一个独立等价类,则将 c 变为该等价类的值;否则为 c 创建一个独立等价类并加入到 E 中.最后将 E 赋值给 E'(第 8 行~第 12 行).如果这个单元格 c 不能够与任何等价类进行合并,则无需进一步检查函数依赖集合,同样为 c 创建一个

独立等价类并加入到 E 且赋值给 E'.最后算法返回 E'(第 13 行~第 15 行).

算法 3. 增量构建等价类集合 IncreBuildEquivRel.

输入:单元格集合 C,等价类集合 E,函数依赖集合Σ,单元格 c;

输出:新的等价类集合 E'.

1. $E'\leftarrow E$;
2. **if** ($\exists e,e\in E'$,c 与 e 中的值相等且属于同一属性) **then**
3. $e\leftarrow e\cup\{c\}$; //将 c 加入到 e 中
4. 令 T 代表 e 所涉及的元组集合;
5. 基于 T 依次检查Σ中所有函数依赖,合并 E'中相关的等价类;
6. **if** ($\exists e,e\in E'$,且 e 违背了某一个数量约束 QC) **then**
7. 创建一个仅包含 c 的等价类 e',且将其值设置为变量;$E'\leftarrow E\cup\{e'\}$;
8. **else if** ($\exists e,e\in E'$,且 e 不干净) **then**
9. **if** ($\exists e,e\in E,c\in e$,且 e 不是单独的独立等价类) **then**
10. 将 c 的值变为 e 中的值;
11. **else** 创建一个仅包含 c 的等价类 e',且将其值设置为变量;$E'\leftarrow E\cup\{e'\}$;
12. **end if**
13. **else**
14. 创建一个仅包含 c 的等价类 e';$E'\leftarrow E\cup\{e'\}$;
15. **end if**
16. **return** E'

3.4 变量值修复

最后,我们确定单元格集合 C 中的变量.如果可以将该变量变回原值而又满足约束条件和函数依赖,则将其变为原值;否则将变量变成满足条件的值(第 1 行~第 6 行).

这里,我们采用两种策略.第一,遍历该单元格所属属性中已出现的所有值,直到找到一个满足条件的值为止.第二,直接随机选取一个在其属性中未出现的值,这种贪心方法由于肯定不会违反函数依赖和约束条件,因此其效率明显很高.

图 3 展示了整个算法的运行过程.假设外部知识库Ω中的硬约束为 HC($t_1(B)$,2);数量约束为 QC(B,2,3),即属性 B 中值为 2 的单元格最多为 3 个;等值约束为 EC($\{t_2[A],t_3[A]\}$),EC($\{t_1[B],t_2[B]\}$);非等值约束为 NC($\{t_3[B],t_4[B]\}$);函数依赖集合为$\{A\rightarrow B\}$.图 3(b)是根据外部知识库中的条件约束生成的等价类集合;图 3(c)在函数依赖的情况下合并等价类;图 3(d)表示等价类赋值后单元格以及等价类的情况.至此,数据初始化完成,其满足硬约束、等值约束和非等值约束.图 3(e)表示插入 $t_1[A]$后的状态,由于其没有违反函数依赖,所以值无需改变.插入 $t_4[A]$=1 时,由于其开始加入到 1 的等价类中,然而此时不满足函数依赖,因为 $t_4[B]$=*Var* 并不属于 2 的等价类.这时需将 $t_4[A]$的值改为变量,如图 3(f)所示.接着插入 $t_5[A]$=1,其会加入到 1 的等价类中,此时 $t_5[B]$会加入到 2 的等价类中.这时 2 的值会有 4 个,违反了数量约束,因此需要将 $t_5[A]$的值改为变量,如图 3(g)所示.最后插入 $t_5[B]$,不违反条件约束和函数依赖,因此不需要更改 $t_5[B]$的值,此时产生了一个满足约束条件和函数依赖的修复.

算法 4. 变量值修复 VarRepair.

输入:单元格集合 C;

输出:无.

1. **for each** (变量 v in C) **do**
2. **if** (将 v 变为原值不满足条件) **then**
3. 将 v 变为满足条件的值;
4. **end if**

5. **end for**

6. **return**;

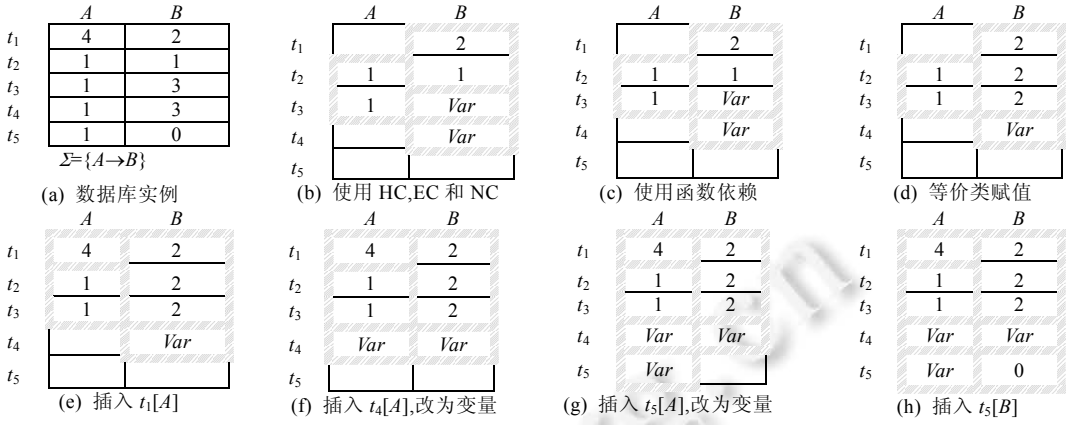


Fig.3 The progress of data repair

图 3 数据修复过程

定理 1. 算法 1 生成的修复都是集合最小化修复.

证明:令原始数据库为 I,修复后的数据库为 I',$\Delta(I,I')$表示 I 和 I'中不同值的单元格集合.要证明 I'是集合最小化修复,即说明$\Delta(I,I')$中的单元格都不能变为原值而使 I 得到修复.初始化阶段算法为了满足条件约束和函数依赖将必要的单元格划分为相同的或不同的等价类,然后修改同一个等价类中单元格的值使其相等.显然,任何一个修改的值变为原值都会使条件约束或函数依赖无法满足,否则算法不会修改.对于非等值约束 EC 中值相同的单元格,我们同时都把它们置为变量.但当其中一个或多个变量的值确定后,最后一个变量的值可以变回原值.为了处理这种情况,我们修复变量时,优先把变量变回原值.综上,初始化后的单元格满足集合最小化修复.

下面考虑增量构建等价类情况.由于从 I 选取单元格到 I'中是随机的,不妨令 $c_1,c_2,\ldots,c_j$ 为从 I 插入到 I'中的单元格的顺序.我们通过归纳法证明,对于任意 $c_i(1<i<j)$,它要么是一个不变的单元格,要么是一个需要变化的单元格.首先初始化过后 I'中单元格满足集合最小化修复,即此时 I'中的单元格要么是不变的单元格,要么是必要变化的单元格.接着我们开始往 I'中插入单元格 c_i,若 c_i 满足条件,则不需要更改值;否则,c_i 必定需要修改,我们用反证法来证明.假设 c_i 保持原值能使 I'保持干净,则算法 1 就不会修改 c_i,因此导致矛盾.所以增量构建等价类的过程同样保证满足集合最小化修复.因此算法 1 生成的修复都是集合最小化修复. □

3.5 性能分析

初始化:假设外部知识库中的单元格个数为 p,函数依赖个数为 k,则条件约束处理的复杂度为 $O(p)$,函数依赖处理的复杂度为 $O(p\times k)$,因此数据初始化的时间复杂度为 $O(p\times k)$.

等价类构建:假设原始等价类集合中包含 n 个等价类.由于需要遍历所有等价类来确定新单元格的所属等价类,所以其时间复杂度为 $O(n)$.假设函数依赖个数为 k,则对于每一个函数依赖,构建依赖等价类相当于检查函数依赖左边的等价类其对应的右边属性是否为等价类,其对应的时间复杂度为 $O(k\times n)$.综上,增量构建等价类算法的时间复杂度为 $O(k\times n)$,由于函数依赖的个数有限,所以增量构建等价类的时间复杂度为等价类个数的线性复杂度.

总体算法:显然对于实例中的每一个单元格,我们都需要进行一次增量的等价类构建.假设实例中单元格的个数为 n,最后形成的等价类的个数为 m,函数依赖的个数为 k,则数据修复算法的时间复杂度为 $O(a+n\times m\times k)$,其中 a 为初始化算法的时间复杂度.结合初始化算法,整个数据修复算法的时间复杂度为 $O(p\times k+n\times m\times k)$,其中,$p$ 为外部知识库中的单元格个数.

4 实 验

本节通过在模拟数据上的实验来展示所提出数据修复算法的性能,主要以运行时间作为度量.除此之外,我们还将展示其他因素对算法运行的影响.

4.1 实验设置

数据描述.为了简便起见,给定关系的所有属性均是整数型类型,且各属性上的数据都遵循均匀分布.在数据生成过程中,我们引入参数 S 来表示每一个属性中每个数据的副本数.例如:假定要生成一个包含 1 000 条元组的关系,且该关系含 10 个属性,则 r=1000,c=10.若设置 S=10,对于其中的每一列中的每一个单元格,我们在(0,r/S)范围内随机生成一个整数(在这里,范围就是(0,100)),则最后在该列中每一个数值大概有 S=10 个副本.

函数依赖.函数依赖关系随机生成.本实验生成形如 $A\rightarrow B$ 的函数依赖,其中,A 和 B 分别为数据库中的两个不相同的属性.同时,为了避免函数依赖之间的矛盾,假设属性间存在排序关系,我们约定所生成的函数依赖左边的属性都要大于(小于)右边的属性.

条件约束.对于给定的随机生成的数据库实例,我们随机生成本文中的 4 种条件约束.同时保证这 4 种约束之间相互不会发生矛盾.例如,如果有等值约束 EC(a,b),就不可能有非等值约束 NE(a,b).

本文的所有实验代码均使用 Java JDK 1.6 编写,运行在 Windows 7 操作系统之下.实验机器为配置主频是 1.60GHz 的 Intel Core i5 处理器,内存为 4GB 的 Thinkpad x240.

4.2 实验结果

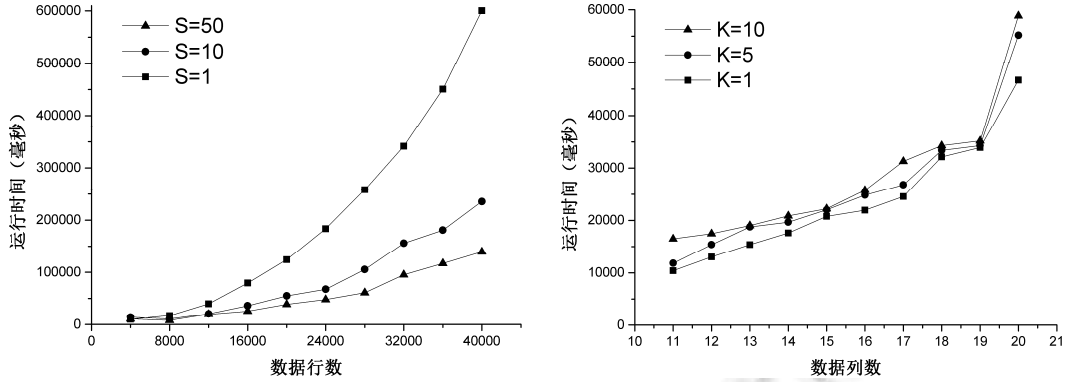
我们从 3 个方面来考察所提出算法的性能,包括总体运行时间、初始化时间以及算法修复过程中发生改变的单元格个数.最后我们考察算法的随机性.

4.2.1 算法总体运行时间

由第 3.5 节可知,本文所提算法的时间复杂度为 $O(p\times k+n\times m\times k)$,其中,$p$ 为外部知识库中的单元格个数,k 为函数依赖个数,n 为数据库中单元格的总数,m 为算法最后形成的等价类的个数.显然,算法的时间效率与多个因素有关,为了充分显示这些因素对算法总体运行时间的影响,我们通过改变数据量、函数依赖个数并控制算法最后生成的等价类来测试算法的运行效率.其中,我们通过改变数据库的行列数来变化总的数据量.而通过改变参数 S,即数据的副本数来达到控制最后生成的等价类个数的目的.显然,S 越大,最后生成的等价类个数越少.反之,最后生成的等价类个数越多.此外,我们还会变化算法中函数依赖的个数 K.由于函数依赖与数据库的列数相关,而副本数与数据库的行数相关.因此,实验中我们将行数与副本数、列数与函数依赖个数一起考量.

图 4 展示了本文所提算法的总体运行时间.其中,通过固定数据列 r=11,函数依赖个数 K=10,我们得到图 4(a).图 4(a)表示不同的数据副本数下,算法的运行时间与元组数量的关系.可以看出,随着元组数量的不断增加,总体运行时间也相应延长.这是因为,数据行数的增加使得单元格个数越来越多,执行增量生成等价类操作也越来越频繁,导致运行时间变长.此外,随着数据副本数的增加,运行时间反而会缩短.这是因为,在数据量固定的情况下,数据副本数越高,则其中包含的等价类个数就会越少.最坏情况下,如果数据副本数为 1,即一列数据中的每一个单元格的值都基本不相等,这时等价类的个数最多为数据的行数.随着等价类个数的增多,等价类的查找就会越来越耗时,最后使得运行时间也相应地变长.当 S=1 时,算法的复杂度变成了 $O(n^2)$(n 为数据量),所以其时间增长趋势呈二次曲线状.另外,图中值得注意的一点是,当数据行数较少时,运行时间会出现微小波动.这是因为,算法本身是一种随机算法,每次都是从数据库中随机取出数据.

通过固定数据行数 r=10000,副本数 S=10,我们得到图 4(b).图 4(b)显示在不同函数依赖个数的条件下,算法的运行时间随着数据列数的变化情况.同样地,随着数据列数的增加,使得单元格个数增多,导致运行时间变长.此外,随着函数依赖个数的增加,算法的运行时间也会相应变长.这是由于,函数依赖个数越多,增量构建等价类时检查的列数也会增多,导致时间变长.显然,函数依赖对运行时间的影响小于副本数对时间的影响.这是因为,当插入新单元格时需要检查同一列中所有等价类来判断它们的值是否相等.而当单元格加入到相应等价类之后,只需要检查当前单元格的等价类所对应列上的相关等价类之间的函数依赖,而不用检查列上其他的等价类.



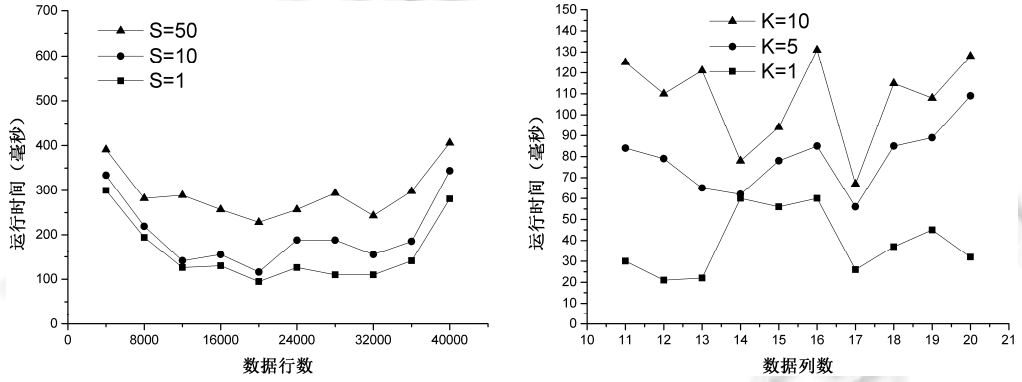
(a) 不同行数和副本数 (b) 不同列数和函数依赖个数

Fig.4 The overall running time

图 4 算法总体运行时间

4.2.2 初始化运行时间

通过固定 4 种条件约束的个数,我们同样考虑行数与副本数、列数与函数依赖个数对初始化的影响.固定数据列 r=11,函数依赖个数 K=10 得到图 5(a).图 5(a)展示在不同副本数的条件下,初始化阶段的运行时间随数据行数的变化情况.由于条件约束的个数固定,所以不同的数据量对初始化的时间影响不大,而且初始化的时间本身较短,所以时间统计的随机性相对较强.另一方面,随着副本数的增加,初始化的运行时间同样也有所延长.



(a) 不同行数和副本数 (b) 不同列数和函数依赖个数

Fig.5 The running time during the initialization phase

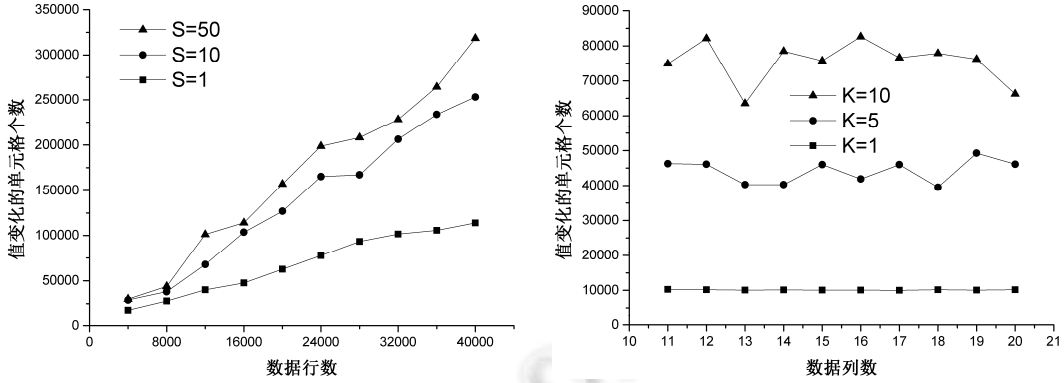
图 5 初始化运行时间

固定数据行 r=10000,副本数 S=10 得到图 5(b).图 5(b)显示在不同的函数依赖条件下,初始化阶段的运行时间随数据列数的变化情况.同样地,由于条件约束个数固定,初始化运行时间表现与列数无关.但是随着函数依赖个数的增加,初始化中函数依赖检查的消耗会增多,从而导致运行时间变长.

4.2.3 改变单元格的个数

改变单元格的个数能从一定程度上反映数据修复的质量,如果改变的单元格越少,则修复后的数据库与元数据库越接近,其质量越高.反之,如果改变的单元格越多,则修复后的数据库与原来的数据库偏差越大,所保留的原数据越少,其质量也越低.我们通过数据量、副本数和函数依赖个数的改变来观察改变单元格的个数.固定数据列 r=11,函数依赖个数 K=10 得到图 6(a).图 6(a)反映在不同的副本数的条件下,改变单元格的个数随数据行数的变化情况.可以看出,随着数据行数的增加,单元格违反函数依赖的可能性增加,需要改变的单元格个数也随之增多.此外,副本数越多,改变的单元格个数也越多.这是因为,副本数越多,值相等的单元格也越多,加入等价类后导致的函数依赖检查也越频繁,违反函数依赖的可能性越大,使得改变的单元格个数也越多.

固定数据行 r=10000,副本数 S=10 得到图 6(b).图 6(b)显示在不同的函数依赖个数的情况下,改变单元格的个数随数据列数的变化情况.可以看出,增加列数并不会明显地改变发生变化的单元格的个数,这是由于,在函数依赖的个数固定的情况下,增加列数并不会导致更多的函数依赖检查,使得改变单元格的个数没有明显的相关变化.然而,随着函数依赖个数的增长,改变单元格的个数会随之变多.这是因为,函数依赖越多,违反函数依赖的可能性越大,发生改变的单元格越多.



(a) 不同元组个数和副本数 (b) 不同列数和函数依赖个数

Fig.6 The number of cells changed

图 6 改变单元格的个数

4.2.4 算法的随机性

由于本文提出的算法是一种随机算法,所以原始数据库中的单元格进行修复过程的顺序不同,需要改变的单元格的个数不确定,产生的修复数据库也不相同,同时算法运行过程中的消耗也会有所差异.为了展示算法的随机性,我们利用同一个随机生成的数据库(行数 r=10000,列数 c=11),同样的条件约束和函数依赖(K=10),在副本数 S=10 的条件下独立地运行 10 次算法得到表 1.可以看出,每次修复的结果都有差异,因为改变单元格的个数都不一样,算法的总体运行时间也有所差异.但是算法总体稳定,时间消耗大约在 11s 左右,改变单元格的个数大约为 70 900.因此上述的性能实验比较是有意义的.另一方面,与第 4.2.2 节类似,由于初始化阶段的用时极短,虽然过程不存在随机性,但是消耗波动性很大,容易受到计算环境的影响.

Table 1 The running time in random

表 1 算法随机运行情况

更改单元格的个数	初始化时间(ms)	总时间(ms)
70 995	343	11 662
70 994	266	11 066
70 898	243	11 141
71 007	158	11 400
70 875	203	11 116
70 927	157	10 867
71 074	141	10 974
71 044	141	11 174
70 954	170	11 112
70 942	157	11 094

5 小 结

数据修复是数据管理领域的重要研究课题,其目的在于提升数据的质量.现有工作主要是采用函数依赖集合进行验证.但是,函数依赖集合仅仅能够部分表达数据质量问题,还需要考虑其他约束条件才能更准确地表达语意.本文提出了多种外部约束条件,包括数量约束、等值约束和非等值约束等.要求修复策略不仅要考虑到函数依赖集合,还应该同时符合这些约束条件.在修复策略目标上,本文采用了基数集合最小化目标,以平衡“最少修复”和“必要修复”二者之间的关系.

外在的约束条件还有更多.因此,在未来我们将沿着两条路线进行探索.首先,我们拟考虑更多的约束定义,或者是约束定义的复杂表达式.其次,我们拟从执行性能上作优化,进一步提升执行效率.

References:

[1] President's Council of Advisors on Science and Technology. Designing a digital future: Federally funded research and development in networking and information technology. 2010. http://www.whitehouse.gov/sites/default/files/microsites/ostp/pcast-nitrd-report-2010.pdf

[2] Big data: Science in the peta-byte era. Nature, 2008,455:1−136. http://www.nature.com/nature/journal/v455/n7209/edsumm/e080904-01.html [doi: 10.1038/455001a]

[3] Dealing with the data. Science, 2011,331(6018):639−806. http://www.sciencemag.org/site/special/data/

[4] Gong XQ, Jin CQ, Wang XL, Zhang R, Zhou AY. Data-Intensive science and engineering: Requirements and challenges. Chinese Journal of Computers, 2012,35(8):1−16 (in Chinese with English abstract).

[5] Ao L, Shu JW, Li MQ. Data deduplication techniques. Ruan Jian Xue Bao/Journal of Software, 2010,21(5):916−929 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3761.htm [doi: 10.3724/SP.J.1001.2010.03761]

[6] Jeffery S, Garofalakis M, Franklin M. Adaptive cleaning for RFID data streams. In: Proc. of the VLDB. Seoul: VLDB Endowment, 2006. 163−174.

[7] Redman TC. The impact of poor data quality on the typical enterprise. Communications of the ACM, 1998,41(2):79−82.

[8] Bohannon P, Flaster M, Fan WF, Rastogi R. A cost-based model and effective heuristic for repairing constraints by value modification. In: Proc. of the SIGMOD. New York: ACM Press, 2005. 143−154. [doi: 10.1145/1066157.1066175]

[9] Kolahi S, Lakshmanan LVS. On approximating optimum repairs for functional dependency violations. In: Proc. of the ICDT. New York: ACM Press, 2009. 53−62. [doi: 10.1145/1514894.1514901]

[10] Arenas M, Bertossi LE, Chomicki J. Consistent query answers in inconsistent databases. In: Proc. of the PODS. New York: ACM, 1999. 68−79. [doi: 10.1145/303976.303983]

[11] Lopatenko A, Bertossi LE. Complexity of consistent query answering in databases under cardinality-based and incremental repair semantics. In: Proc. of the ICDT. Barcelona: Springer-Verlag, 2007. 179−193. [doi: 10.1007/11965893_13]

[12] Bekales G, Ilyas IF, Golab L. Sampling the repairs of functional dependency violations under hard constraints. In: Proc. of the VLDB. Singapore: VLDB Endowment, 2010,3(1):197−207. [doi: 10.14778/1920841.1920870]

[13] He C, Tan ZJ, Chen Q, Sha CF, Wang ZH, Wang W. Repair diversification for functional dependency violation. In: Proc. of the DASFAA. Heidelberg: Springer-Verlag, 2014. 468−482. [doi: 10.1007/978-3-319-05813-9_31]

[14] Chen Q, Tan ZJ, He C, Sha CF, Wang W. Repairing functional dependency violations in distributed data. In: Proc. of the DASFAA. Heidelberg: Springer-Verlag, 2015. 441−457. [doi: 10.1007/978-3-319-18120-2_26]

[15] Li J, Liu X. An important aspect of big data: Data usability. Journal of Computer Research and Development, 2013, 50(6):1147−1162 (in Chinese with English abstract).

[16] Bleiholder J, Szott S, Herschel M, Kaufer F, Naumann F. Subsumption and complementation as data fusion operators. In: Proc. of the EDBT. New York: ACM Press, 2010. 513−524. [doi: 10.1145/1739041.1739103]

[17] Liu HP, Jin CQ, Zhou AY. A pattern-based entity resolution algorithm. Chinese Journal of Computers, 2015,38(9):1796−1808 (in Chinese with English abstract).

[18] Xie JY, Yang J, Chen YG, Wang HX, Yu PS. A sampling-based approach to information recovery. In: Proc. of the ICDE. Iscataway, NJ: IEEE Computer Society, 2008. 476−485. [doi: 10.1109/ICDE.2008.4497456]

[19] Chen HQ, Ku WS, Wang HX, Sun MT. Leveraging spatio-temporal redundancy for RFID data cleansing. In: Proc. of the SIGMOD. New York: ACM Press, 2010. 51−62. [doi: 10.1145/1807167.1807176]

[20] Zhuang YZ, Chen L. In-Network outlier cleaning for data collection in sensor networks. In: Proc. of the CleanDB. New York: VLDB Endowment, 2006. 41−48.

[21] Chiang F, Miller RJ. A unified model for data and constraint repair. In: Proc. of the ICDE. Iscataway, NJ: IEEE Computer Society, 2011. [doi: 10.1109/ICDE.2011.5767833]

[22] Silberschatz A, Korth H, Sudarshan S. Database System Concepts. 6th ed., McGraw-Hill Education, 2010.

附中文参考文献:

[4] 宫学庆,金澈清,王晓玲,张蓉,周傲英.数据密集型科学与工程:需求和挑战.计算机学报,2012,35(8):1−16.

[5] 敖莉,舒继武,李明强.重复数据删除技术.软件学报,2010,21(5):916−929. http://www.jos.org.cn/1000-9825/3761.htm [doi: 10.3724/SP.J.1001.2010.03761]

[15] 李建中,刘显敏.大数据的一个重要方面:数据可用性.计算机研究与发展,2013,50(6):1147−1162.

[17] 刘辉平,金澈清,周傲英.一种基于模式的实体解析算法.计算机学报,2015,38(9):1796−1808.

金澈清(1977－),男,浙江文成人,博士,教授,博士生导师,CCF 会员,主要研究领域为海量数据管理,包括基于位置的服务,数据质量,不确定数据管理.

刘辉平(1990－),男,博士生,主要研究领域为基于位置的服务,数据挖掘,实体解析.

周傲英(1965－),男,博士,教授,博士生导师,CCF杰出会员,主要研究领域为Web数据管理,数据密集型计算,内存集群计算,大数据基准测试和性能优化.