**Efficient Approximation Algorithms for Repairing Inconsistent Database**

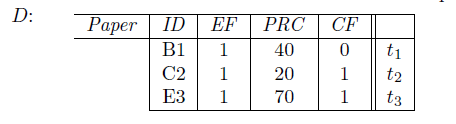
1. **背景和领域介绍**

在修复非一致数据库时一般是修改数据的值。在这篇文章中，我们主要是关注在计算复杂度的分析方面。我们用一种有效的近似算法来修复数据库，并且提出了一种执行时间为O(nlogn)的算法。在本文中，实验表明，即是是对大型数据库，我们的算法也同样有效。

在处理非一致性数据库时，有两种方法：

1. 数据清理
2. 一致性查询问答（CQA）：它保持源数据不改变，在查询时解决非一致性，识别一致性数据。

Example



约束条件：只有当PRC>= 50,且CF=1时，EF才为1；

所以在数据库D中，元组t1和t2不满足约束条件。有两种方法进行修复：

1. 删除最小数量的元组来满足约束条件，即要删除t1和t2，这样只有t3留在数据库D中。
2. 修改最少的的属性值来满足约束条件。例如，我们可以将t2中的EF值从1改为0，这样就可以减少丢失的信息。

由于在找从关于IC的属性更新修复到已知输入事件的距离是MAXSNP-hard问题。通过将这个问题转换为最小权值覆盖问题，这个修复问题可以通过例如greedy或者layer的近似算法来计算。

但是greedy算法的执行时间是O(n^3)，这对于真正的数据库来说是不够高效的。在这篇文章中，提出来一种变形greedy算法，在假设某元组可以参与到多种在恒定范围内的非一致的地方，其执行时间可以降为O(n^2 logn)。通过实验得出，修改后的greedy算法要比greedy算法快。但greedy算法要比layer和修改后的layer算法都要快。

1. **算法的说明与步骤**

Cardinality Repairs

在有不一致的数据时，可删除该数据。但基础原子查询的查询回复的复杂性是。并且在语义条件下获得数据库修复是计算复杂度。所以需要一种近似算法。在每个关系中增加一个属性，它是数据库中唯一可以被修改的属性。在元组中，表示其在数据库中，反之，时，表示是删除的元组。所以数据库中的不一致数据可以将的值从1改为0，就相当于删除了该数据。

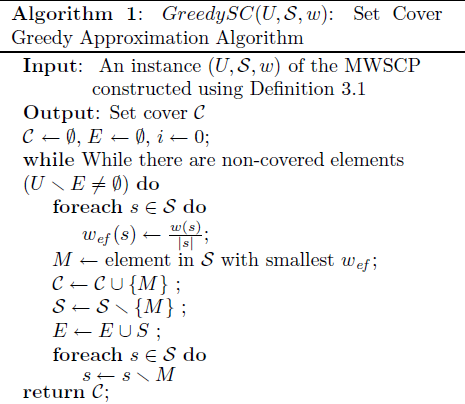


**Attribute-Update Repairs**

**算法背景：**把问题转换为MWSCP，Minimum Weighted Set Cover Optimization Problem，即为最小权值覆盖优化问题。给定集合U，S是U的子集，权值集合w（Si），覆盖集C是S的子集。U中的每个元素都至少属于C中的一个，其中所有权值的和最小即可。

Algorithms 1：GreedySC（U,S,w）

**伪代码：**



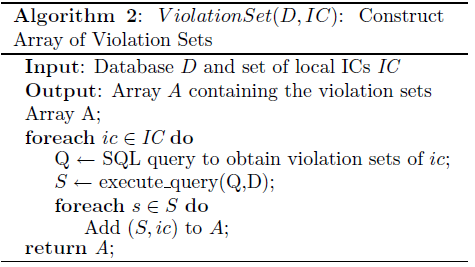
**算法说明和解释：**输入分别是U,S,w，只要还有没被覆盖的元素，就进行循环，选取S中最小的Wef，最终输出C是最优覆盖集。

**算法结论：**当数据库的大小是n时，该算法的执行时间是O（n^3），并且当Deg（D，IC）为常数时，则为O（n^2）。即使当不一致的度是已界定时，其执行时间可控制在平方时间内。但对大型数据库来说代价还是太高，所以我们要修改原greedy算法来获得更高效的方法。

Algorithms 2：

**算法背景：**在原greedy 算法中，在找没有被覆盖元素的最小权值是代价消耗最大的一步；**算法基本思想：**我们将S存储在优先队列P中，在优先队列中每个元素都包含一个非一致元组t，其权值，和指向U中元素的指针，是一个violation sets，他们会存储在数组中。主要是获得包含violation sets的数组A。通过重写每个完整性约束作为SQL视图，即是如果被满足则为空。

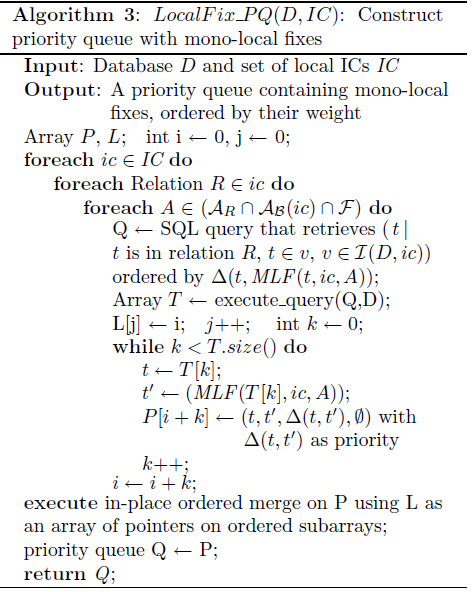
**伪代码：**



Algorithms 3：

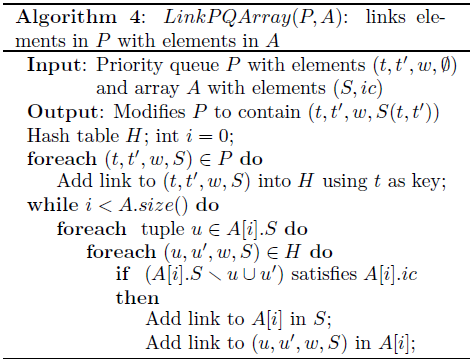
**算法基本思想：**通过计算每个非一致元组t的mono-local fix，t’=MLF（t, ic, A）。并将其对应的权值按顺序存在数组中。这个有序数组可看作是堆优先队列实现的数组。但在此阶段，P中的s（t,t’）是空；

**伪代码**：



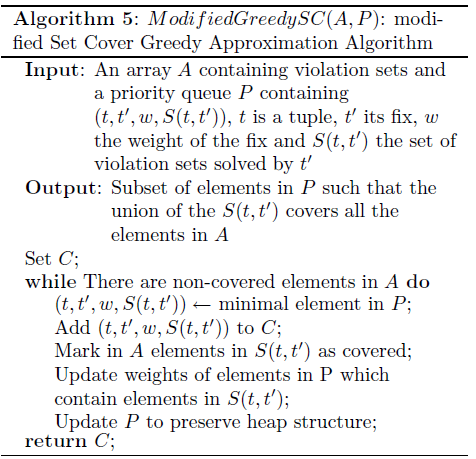
Algorithms 4：

**算法基本思想**：将在数组A存储的violation sets加入到每个s(t,t’)中，这个算法运用了包含P中元素链接的辅助哈希表来实现。



Algorithms 5：ModifiedGreedySC

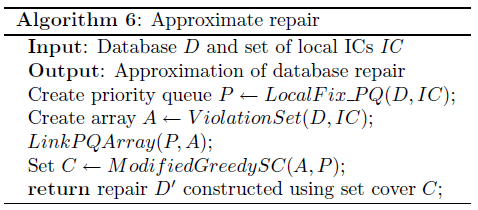
**算法基本思想：**在每次迭代中都将优先队列中最小元素加入到覆盖集中，则每次优先队列的权重也相应更新，由于A中包含了链接P中的指针，所以每次更新的效率都很高效。其次，我们还要存储堆结构，这通过自上而下堆来实现更新元素。



Algorithms 6：

**算法基本思想：**完整的修复算法包括MWSC问题的构建和modified-greedy算法的执行。

**伪代码：**

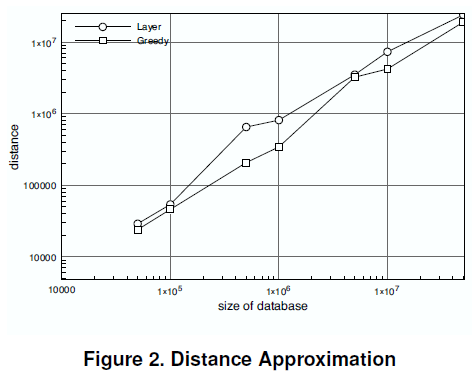


算法结论：其执行时间为O(n^2logn)，若Deg（D,IC）为常数，则为O(nlogn);

1. **实验：**

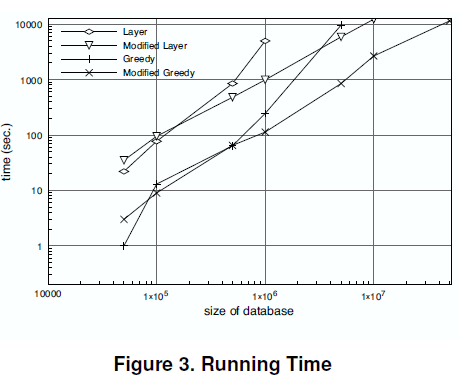
随机产生3个不同大小的数据库，其中30%的元组是不一致的。

1. 比较greedy和layer算法，看哪个能给出一个更好的近似数据库修复方法。如图2；



理论上，我们预期layer要比greedy算法好，但实验结果表明，greedy能产生更好的近似最优MWSCP权重。

1. 比较MWSCP近似算法的执行时间，如图3；



采用优先队列来存储MWSCP对，能提高性能。

* 对大型数据库来说，修改后的greedy算法是最有效的；
* 修改后的greedy算法可有效的用于计算相对于一组本地数据库完整性约束的修复，甚至可用于大型数据库；
* 修改后greedy算法运行速度快，相比于layer算法，能给出更好的近似结果；

1. **结论：**

这篇文章提出了一种优化算法来计算在关于局部线性否定约束的非一致数据库中进行属性更新修复。

这个算法首先将修复问题转化成MWSCP，并用一种有效的修复贪心算法来解决MWSCP，我们实验得出这个算法也可以适用于更大型数据库；

由于greedy和layer算法分别和对数，常数因子接近，所以我们预期从layer算法中会得到更好的估计值。然而，实验结果表明贪心算法结果更好，且修改后的贪心算法比修改前执行时间更短，比layer更快。

运用修改后的greedy算法可以将元组删除修复法转化成属性更新问题。这样，数据库就不用主键约束，也不用要求在线性否定ICs上的局部性设定。

本文的结果同样在经过小的修改后运用到其他语义修复上。例如，将设定为不同的值，这样的值也会随之改变，可以确定从表中删除元组的优先级，表示先从表R中删除元组。



**Interaction Between Record Matching and Data Repairing**

1. **背景和领域介绍**

匹配（Matching）：主要将元组和现实中的实物对应起来；

修复（Repairing）：用约束条件来修复数据库中的错误；

现阶段，两者处于相对独立的过程。所以在这篇文章中，我们研究了一个新的问题，即记录匹配和数据修复之间的交互作用。我们发现修复可以有效地帮助我们找到匹配的实物；同理，匹配也能有效的帮助修复。

为了得到这种交互作用，我们提出了一种统一的，能无缝结合修复，匹配操作的框架。在完整性约束，匹配规则和主数据的基础上来清理数据库。

再通过匹配和修复清理数据问题上，我们给出了详细的分析，包括约束规则共同作用的静态分析，其复杂性，终止性和确定性数据清理分析；但这些问题是NP，和coNP完全，PSPACE完全的难问题。

但我们才用了同时通过匹配和修复清理数据的高效算法。该算法分别通过置信分析和熵分析得到确定性修复和可靠性修复，这比通过启发式产生的可能性修复要精确多了。通过实验证明采用现实数据，我们的方法比作为独立国产记录匹配和数据修复的精确度要高得多。

新问题：有数据库D，主数据Dm，CFD，匹配规则г；找到Dr使得

1. 满足各项CFD条件，且为一致性；
2. Dr中没有能和Dm匹配的元组；
3. Dr与源数据D相比差别是最小的；

所以，我们提出了一个统一的框架来清理数据：将CFDs和MDs都作为清理规则，能告诉我们如何修复错误；利用a.用户设定的数据置信值；b.数据自己产生的熵来计算数据确定性；c.主数据，三者来确保其准确性；

1. **算法说明及步骤**

由于数据清理问题是NP-hard，所以我们提出来三种算法构成的方法：

1. 依据置信值分析和主数据确定精确的确定性修复；
2. 当置信值很低或没有时，采取根据信息熵来计算可靠性修复；
3. 采用启发式算法在脏数据中找到一致性修复；

以此来提高精确度；

我们采用DBLP和US Health & Human Service的数据；实验证明我们的方法比独立采用匹配和修复方法要好得多，且确定性修复和可靠性修复要比启发式方法的修复要精确的多。尽管清理数据库是高复杂性的，但我们的算法也能很好的运行。

数据清理规则：

CFD：Conditional Functional Dependencies 有条件函数依赖；

CFDφ是定义在R上的配对（X—>Y, tp）。当

1. X🡪Y是R上的一个标准函数依赖，相当于FD是嵌入在φ中。
2. Tp是规则模式，X和Y的属性，对于每个XUY中的属性A，tp[A]要么是常数，要么是未命名的变量。

MD：Matching Dependencies 相似谓词的集合г

分别有定义在R和Rm上的positive MD和Negative MD；

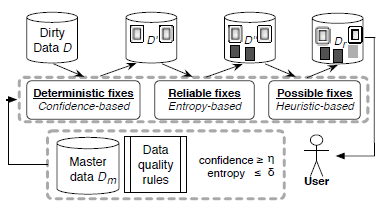
三级数据清理方法（A tri-level Data Cleaning Solution）

* UniClean:

输入：D，Dm，清理规则θ，阈值：ηδ分别从熵和置信值得到；

UniClean是通过这些清理规则，将匹配和修复整合在一起来产生修复；用三级精度值区分这些修复，强调其准确度

如图所示的框架概览，通过三个算法一个接一个的执行来得到修复；



1. 通过置信值得到置信修复：只有当t中某属性的置信度高于阈值η时，才根据置信度用清理规则更新t[A];
2. 通过熵得到可靠性修复：当某些属性置信度很低或没有置信度时，则采用熵来修复；只有当t的某一属性的熵低于阈值δ，则采用清理规则г来更新t[A];
3. 并不是所有的错误都能在前两步修复，所以采用启发式方法产生修复，即possible fix；

数据质量规则的两个主要问题

1. 一致性问题：在给定Dm和θ=∑∪г，确定是否有非空D，使D|=∑，且（D，Dm）|=г。直观的来看，是判断在θ中的规则本身是否是脏数据。（NP-complete）；
2. 蕴含问题（Implication Problem）：给定Dm，∑，和CFD/MD ζ，确定是否有∑|=ζ。帮助我们找到并删除在∑中多余的规则，以此提高性能（coNP-complete）

基于规则的数据清理的两个问题

1. 终止问题（Termination Problem），看整个过程是否终止，到达一个不动点，没有任何清理规则能再运用；
2. 决定性问题（Deterministic Problem），看是否所有终止问题都在相同点不动点结束；

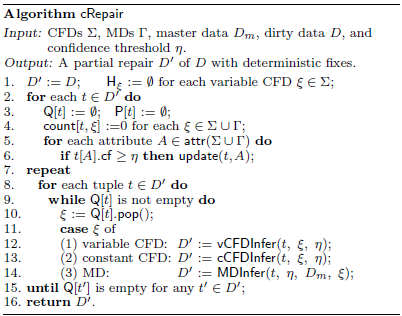
看整个过程的输出是否独立于规则应用的顺序，都是PSPACE-Complete，所以不能再实际中找到有效。

由数据置信值得到的确定性修复：(Deterministic Fixes with Data Confidence)

确定性修复由三种方法产生：a. 从MD；b. 从固定的CFD；c. 从变量CFD

* **基于置信值的数据清理算法cRepair**

**伪代码：**



**算法解释：**

输入：CFD∑，MDг，主数据Dm，脏数据D，置信阈值η

Line1—6：初始化

Line7—15：递归计算确定性修复，分别从变量CFD，常量CFD和MD中得到vCFDinfer，cCFDinfer，MDinfer。直到没有更多的确定性修复能找到；

Line16：返回较干净的数据D’；

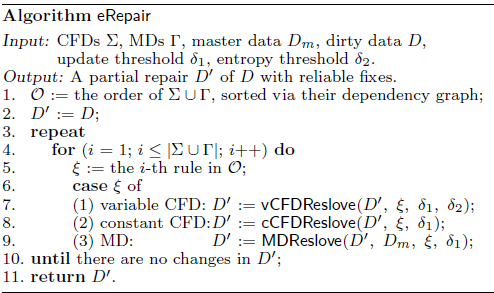
**算法结论：**复杂性：O（|D||Dm|size(∑∪г)）;由于采用MD找确定性修复时，可采用后缀树，则可优化为O（|D|size(∑∪г)）;

* **基于信息熵的可靠性修复（Reliable Fixes with Infromation Entropy）**

**算法基本思想：**离散随机变量χ的熵为：；熵值能描述χ的确信度。越小，预测越准确。



**伪代码：**基于熵的数据清理算法eRepair，如下所示：



**算法解释：**

输入：CEDs∑，MDsг，主数据Dm，脏数据D，δ1更新频率阈值，δ2熵的阈值；

找到D的可靠性修复，且在此过程中deterministic修复不发生变化；

Line1：根据依赖关系图，将∑∪г中得到的清理规则进行排序，有较大影响力的条件则优先应用，依次类推；

Line3—9：依次将排好序的规则解决D中的冲突，根据规则的类型，选取方案，得到vDFDResolve，cCFDResolve，MDResolve。

Line10：直到没有更多的规则可用或所有数据的值都超过δ次改变，则停止。

Line11：返回D’，是一个干净的带有可靠性修复的数据库；

1. **实验**

**Exp1：匹配能帮助修复**

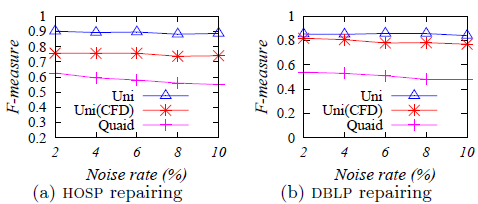
分别将Uni，Uni（CFD），quaid得到的F-measure值进行对比，Noi%从2%~10%；

结论：A. Uni比Uni（CFD）和quaid都要好，所以匹配能够帮助修复；

B. 随着noi%的上升，F-measure下降，但对Uni 的影响较小；

C．即使只采用CFD，Uni（CFD）比quaid好，因为quaid只产生possible fixes，而Uni（CFD）可靠性和确定性修复都能找到；

如下图所示：



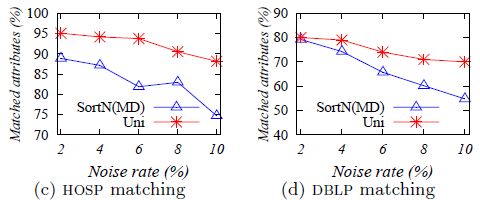
**Exp2：修复能帮助匹配**

对比Uni和采用MD的SortN（MD）的匹配属性所占的比例；

结论：A. Uni比SorN（MD）的效果好，所以修复能帮助到匹配；

B．随着noi%的升高，F-measure的值下降，但Uni影响较小；和Exp1中的结论一样；

如下图所示：



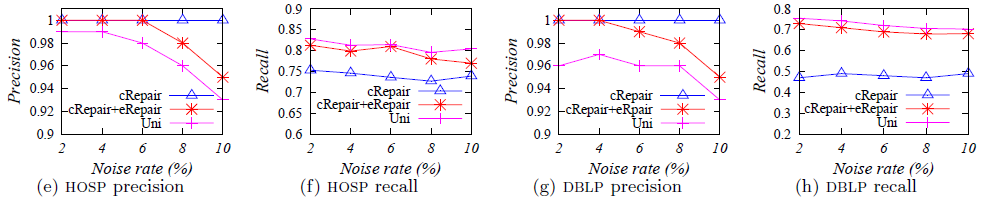
**Exp3：确定性和可靠性修复的精确度**

对比cRepair产生的确定性修复，cRepair+eRepair产生的确定性和可靠性修复，Uni产生的所有修复；

结论：

1. 确定性修复的精确度最高，但回溯率很低；因为cRepair只对asserted attribute进行修复，对noi%不敏感；
2. 由Uni得到的修复精确度最低，回溯率最高，且精确度对noi%十分敏感；因为Uniclean的最后一步是采用启发式方法，会产生possible fix；
3. cRepair+eRepair得到的精确度和回溯率在两者中间，精确度同样对noi%敏感；

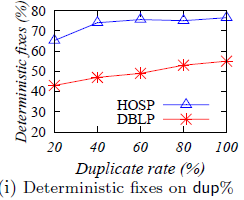
如下图所示：



**Exp4：dup%和asr%对确定性修复的影响**

看在cRepair中确定性修复占有的比例；我们将asr%=40%，dup%从20~100%变化；

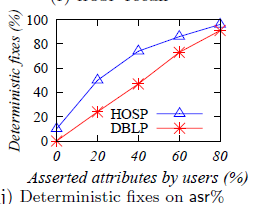
如图：



结论：当dup%升高时，找确定性修复越多；

若将dup%=40%，asr%从0%~80%变化；

如图：

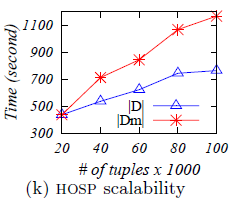


结论：cRepair中找到的确定性修复主要由asr%控制，所以清理规则只对asserted attribute使用；

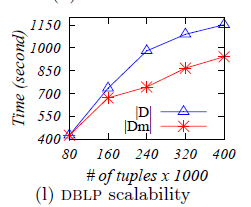
**Exp5：可扩展性**

|D|大小的脏数据和|Dm|大小的主数据，noi%=6%，dup%=40%；

如图所示：



确定|Dm|的大小为60K，|D|的大小从20K~100K



确定|D|的大小为60K，|Dm|的大小从20K~100K

结论：Uni的扩展性比quaid好，说明Uni中的索引结构和优化方法十分有效；

实验结论：

1. 通过将匹配和修复结合在一起来清理数据比分别单独使用配合和修复的方法性能提高了30%，15%；
2. 确定性修复和可靠性修复的精确度较高，等noi%<4%时，精确度可达100%；且当noi%上升时，精确度降低的很缓慢；
3. 通过UniClean得到的备选修复景区度可达96%；
4. 我们的数据清理方法可扩展性也很强，比只用CFD的quaid快将近50倍；
5. **结论：**

提出了将匹配和修复两个过程结合起来的方法。提出了一个基于CFD和MD得到的清理规则的统一的框架，使得匹配和修复过程结合起来。并且，根据确信值和熵值提出了确定性修复和可靠性修复。我们通过实验证明这个方法能够有效的提高数据质量，且它们的可扩展性也很强。