The Computation of Optimal Subset Repairs[*]

Dongjing Miao
Harbin Institute of Technology
miaodongjing@hit.edu.cn

Zhipeng Cai
Georgia State University
zcai@gsu.edu

Jianzhong Li
Harbin Institute of Technology
lijzh@hit.edu.cn

Xiangyu Gao
Harbin Institute of Technology
gaoxy@hit.edu.cn

Xianmin Liu
Harbin Institute of Technology
liuxianmin@hit.edu.cn

ABSTRACT

Computing an optimal subset repair of an inconsistent database is becoming a standalone research problem and has a wide range of applications. However, it has not been well-studied yet. A tight inapproximability bound of the problem computing optimal subset repairs is still unknown, and there is still no existing algorithm with a constant approximation factor better than two. In this paper, we prove a new tighter inapproximability bound of the problem computing optimal subset repairs. We show that it is usually NP-hard to approximate it within a factor better than 17/16. An algorithm with an approximation ratio $(2 - 1/2^{\sigma-1})$ is developed, where σ is the number of functional dependencies. It is the current best algorithm in terms of approximation ratio. The ratio can be further improved if there are a large amount of *quasi*-Turán clusters in the input database. Plenty of experiments are conducted on real data to examine the performance and the effectiveness of the proposed approximation algorithms in real-world applications.

PVLDB Reference Format:
Dongjing Miao, Zhipeng Cai, Jianzhong Li, Xiangyu Gao, Xianmin Liu. The Computation of Optimal Subset Repairs. *PVLDB*, 13(11): 2061-2074, 2020.
DOI: https://doi.org/10.14778/3407790.3407809

1. INTRODUCTION

A database I is inconsistent if it violates some given integrity constraints, that is, I contains conflicts or inconsistencies. For example, two tuples conflict with each other, if they have the same city but different area code. The inconsistency has a serious influence on the quality of databases, and has been studied for a long time.

In the research on managing inconsistencies, the notion of *repair* was first introduced decades ago [4]. *Subset repair* is a popular type of repair [19, 2]. A subset repair of an inconsistent database I is a consistent subset of I, which satisfies all the given integrity constraints, obtained by performing a minimal set of operations on I. An *optimal subset repair* is the subset repair with minimum cost, where the cost can be defined in different ways depending on the requirement of applications.

Nowadays, the computation of optimal subset repairs is becoming a basic task in a wide-range of applications not only data cleaning and repairing, but also many other practical scenarios. Thus, the computation of optimal subset repairs is becoming a standalone research problem and attracted a lot of attention. In the following, we give three example scenarios, in which the computation of optimal subset repairs is a core task.

Data repairing. As discussed in [40, 20, 11], the benefit of computing optimal subset repairs to data repairing is twofold. First, computing optimal subset repairs is considered an important task in automatic data repairing [22, 45, 25]. In those methods, the cost of a subset repair is the sum of the confidence of each deleted tuple. Therefore, an optimal subset repair is considered as the best repaired data. Second, optimal subset repairs are also used as the best recommendation to enlighten the human how to repair data from scratch in semi-automatic data repairing methods [7, 9, 24, 30]. In addition, the deletion cost to obtain an optimal subset repair also can be used as a measurement to evaluate the database inconsistency degree [11].

Consistent query answering. Optimal subset repairs are the key to answer aggregation queries, such as count, sum and average, in inconsistent databases. For example, a consistent answer of a sum query is usually defined as the range between its greatest lower bound (*glb*) and its least upper bound (*lub*). Optimal subset repairs can help deriving such two bounds.

Consider the inconsistent database *GreenVotes* shown in Figure.1(a), due to functional dependency

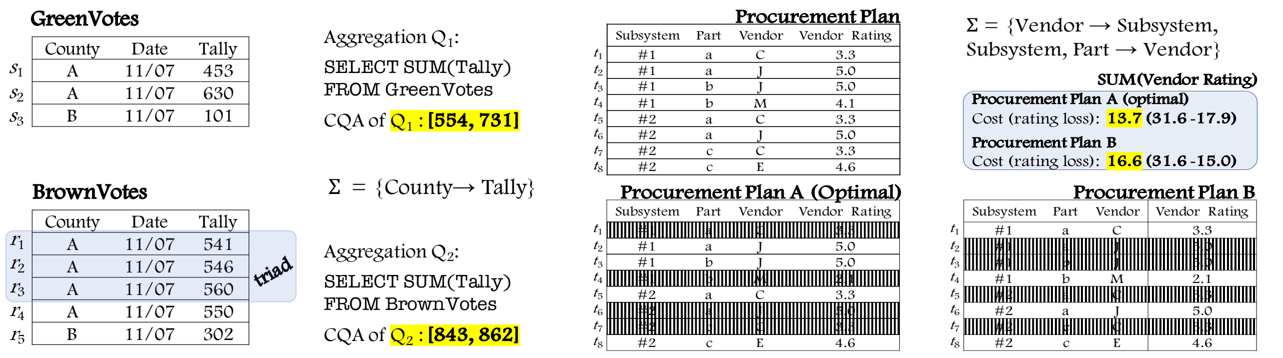
$$\mathit{County}, \mathit{Date} \rightarrow \mathit{Tally},$$

tuples s_1 and s_2 conflict with each other, hence, it has two repairs $J_1 : \{s_1, s_3\}$ and $J_2 : \{s_2, s_3\}$. Obviously, aggregation query Q_1 has different answers, such as 554 in J_1 but 731 in J_2. A smart way to answer such queries, as defined in [6, 10], is to return the *lub* and *glb* which are 554 and 731.

Computing the *lub* is equivalent to computing optimal subset repairs since the *lub* is exactly the sum of weights of

[*]Work funded by the National Natural Science Foundation of China (NSFC) Grant No. 61972110, 61832003, U1811461, and the National Key R&D Program of China Grant No. 2019YFB2101900.

Proceedings of the VLDB Endowment, Vol. 13, No. 11
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3407790.3407809



GreenVotes

	County	Date	Tally
s_1	A	11/07	453
s_2	A	11/07	630
s_3	B	11/07	101

Aggregation Q_1:

```
SELECT SUM(Tally)
FROM GreenVotes
```

CQA of Q_1 : **[554, 731]**

$\Sigma = \{\text{County} \to \text{Tally}\}$

BrownVotes

	County	Date	Tally
r_1	A	11/07	541
r_2	A	11/07	546
r_3	A	11/07	560
r_4	A	11/07	550
r_5	B	11/07	302

Aggregation Q_2:

```
SELECT SUM(Tally)
FROM BrownVotes
```

CQA of Q_2 : **[843, 862]**

(a) Consistent query answering of scalar aggregation

Procurement Plan

	Subsystem	Part	Vendor	Vendor Rating
t_1	#1	a	C	3.3
t_2	#1	a	J	5.0
t_3	#1	b	J	5.0
t_4	#1	b	M	4.1
t_5	#2	a	C	3.3
t_6	#2	a	J	5.0
t_7	#2	c	C	3.3
t_8	#2	c	E	4.6

$\Sigma = \{\text{Vendor} \to \text{Subsystem}, \text{Subsystem, Part} \to \text{Vendor}\}$

SUM(Vendor Rating)

Procurement Plan A (optimal)
Cost (rating loss): **13.7** **(31.6 -17.9)**

Procurement Plan B
Cost (rating loss): **16.6** **(31.6 -15.0)**

Procurement Plan A (Optimal)

	Subsystem	Part	Vendor	Vendor Rating
t_1	#1	a	C	3.3
t_2	#1	a	J	5.0
t_3	#1	b	J	5.0
t_4	#1	b	M	2.1
t_5	#2	a	C	3.3
t_6	#2	a	J	5.0
t_7	#2	c	C	3.3
t_8	#2	c	E	4.6

Procurement Plan B

	Subsystem	Part	Vendor	Vendor Rating
t_1	#1	a	C	3.3
t_2	#1	a	J	5.0
t_3	#1	b	J	5.0
t_4	#1	b	M	2.1
t_5	#2	a	C	3.3
t_6	#2	a	J	5.0
t_7	#2	c	C	3.3
t_8	#2	c	E	4.6

(b) Refine the procurement plan when business constraints change

Figure 1: Example applications of computing optimal subset repairs

tuples in an optimal subset repair, J_2, if the vote tallies are treated as tuple weights.

On the other side, computing the *glb* can be speed up by an optimal subset repair. Computing *glb* is equivalent to find a weighted maximum minimal vertex cover that is NP-hard and cannot be approximated within $o(n^{-1/2})$ [15] but can be solved in $O(2^k n^c)$ time [49], where k equals to the cost of an optimal subset repair, and $c > 1$. If k is clear in advance, we will be able to decide to run the exact algorithm and tolerate exponential running time, or to run an approximation algorithm and tolerate the unbounded accuracy loss.

Moreover, when the two ranges are computed, one can safely conclude that Brown won the election, since the maximum vote tally for Green is smaller than the minimum vote tally for Brown.

Optimization. Let us consider an example raised in marketing as shown in Figure.1(b). The procurement plan in Figure.1(b) lists all the current vendors for all the parts, and each vendor has a rating representing its production capacity, after-sales service, supply price and so on [29]. Note that, all the tuples in the list are correct and consistent. However, the business consideration recently changes. First, a vendor is no longer allowed to take part in two or more subsystems for protection of commercial secrets. Second, each part should be supplied from the same vendor for convenience of maintenance. For now, we need to refine the plan to satisfy these requirements but preserving high-rating vendors as possible.

The problem of refining the procurement plan can be transformed into the problem of computing optimal subset repairs. First, the business requirements can be written as the following two functional dependencies

$$\begin{aligned} \textit{Vendor} &\to \textit{Subsystem}, \\ \textit{Subsystem, Part} &\to \textit{Vendor}. \end{aligned}$$

Then, an optimal subset repair is the best refined plan if we treat *vendor rating* as tuple weight. For example, in Figure.1(b), plan A is an optimal subset repair and has a total rating 17.9, which is much better than another plan B.

In fact, there are many other applications of optimal subset repairs, but we can not detail any more here due to space limitation. As we can see from these applications, computing optimal subset repairs is becoming a standalone problem. However, it has not been well-studied yet, especially on the complexity and algorithms.

Complexity aspects. It is already shown that the problem of computing optimal subset repair with respect to functional dependencies is NP-complete [6, 10] even if given two functional dependencies. This result is too rough since it is just proved by a particular case. The most recent work in [40] improves the complexity result. The boundary between polynomial intractable and tractable cases is clearly delineated, such that the problem of computing optimal subset repairs cannot be approximated within some small constant unless P≠NP, if the given constraint set can be simplified as one of four specific cases, otherwise it is polynomial solvable. Other complexity results on computing optimal repairs exist, such as [22], [37], [12], [41], [28]. They are either restricted to repairs obtained by value modifications instead of tuple deletions or constraints stronger than functional dependencies. Complexity results on repair checking exists, such as [19], [2]. Repair checking is to decide if a given database is a repair of another database. However, it is a problem much easier than computing optimal subset repairs. Due to their restrictions, these results cannot help deriving tight complexity results for the problem of computing optimal subset repairs. Therefore, a tight lower bound remains unknown so far.

Algorithm aspects. When the database schema and the number of functional dependencies are both unbounded, computing optimal subset repairs is equivalent to the weighted minimum vertex cover problem. Algorithms for vertex cover can be applied directly to compute optimal subset repairs. However, even the best algorithm [34] can only give an approximation with ratio 2 if the data is large enough. Fortunately, as shown in the applications above, the relation schema is fixed in practice, and the number of constraints is bounded by some constant. In such context, the problem is no longer the general vertex cover problem, but it is not clear whether the problem can be better approximated.

Although a large body of research on *data repairing* concerned with the computation of an optimal repair, they do not tell us if we could do better in terms of approximation ratio for computing optimal subset repairs. Approximation algorithms [13, 42, 37] focus on computing repairs obtained by value modification rather than subset repairs. Heuristic methods [13, 22, 21] and probabilistic methods [44, 25] are unable to provide theoretical guarantee on approximation ratio. SAT-solver based methods [24] suffer an exponential time complexity. Other methods [18, 31] try to modify both data and constraints, which are not able to return an optimal subset repair that we need.

In addition, as we mentioned above, the notion *subset repair* was first proposed in the research on *consistent query answering*. However, the series of works on *consistent query answering* also cannot provide a better approximation to an optimal subset repair. Their goal is to derive a certain answers for given queries by finding the answers held in all repairs [4, 48]. There are three approaches to achieve the goal. The first is the answer set programming based approaches [5, 16]. The second is the query rewriting based approaches [46, 47]. The third is the SAT/LP-solver based approaches[38, 39, 27, 26]. All the approaches try to avoid computing exponentially many repairs in the size of database.

In summary, the following problems for computing optimal subset repairs remain open.

Firstly, a tight lower bound is still unknown so that it is impossible to decide whether an algorithm for computing optimal subset repairs is optimal in terms of approximation ratio.

Secondly, it is unknown whether we can approximate optimal subset repairs within a *constant* better than 2, which has a very important practical significance.

This paper focuses on the two open problems, and the main contributions are as follows.

Firstly, a tighter lower bound of the problem of computing optimal subset repair is obtained. Concretely, it is proved that the problem of approximating optimal subset repairs within 17/16 is NP-hard for most of the polynomial intractable cases. For the other cases, it is proved that the problem of approximating optimal subset repairs within 69246103/69246100 is also NP-hard.

Secondly, an algorithm with an approximation ratio $(2 - 1/2^{\sigma-1})$ is developed, where σ is the number of given functional dependencies. This ratio is a constant better than 2. It is the current best algorithm in terms of approximation ratio.

Thirdly, to further improve the approximation ratio, the *quasi*-Turán cluster is proposed for extending the algorithm above. Based on the *quasi*-Turán cluster, an approximation algorithm for computing the subset repairs with ratio $(2 - 1/2^{\sigma-1} - \epsilon)$ is designed, where the factor $\epsilon \geq 0$ depends on the characteristic of input database.

Fourthly, plenty of experiments are conducted on real data to examine the performance and the effectiveness of the proposed approximation algorithms in real-world applications.

The following parts of this paper is organized as follows. Necessary conceptions and definition of the problem of computing subset repairs are formally stated in Section 2. The complexity analysis of the problem is shown in Section 3. The approximation algorithms are given in Section 4. Experiment results are illustrated in Section 5. At last, Section 6 concludes the paper.

2. PROBLEM STATEMENT

Concepts used in database theory and the formal definition of the problem are given in this section.

2.1 Integrity Constraints

The consistency of databases is usually guaranteed by integrity constraints. The languages for specifying integrity constraints can get quite involved. For example, functional dependency [1] is the most popular type of integrity constraint and has a long history. A large body of work on managing data quality is built on it. An important special case of functional dependencies are key constraints [1]. Conditional functional dependencies [14] generalize functional dependencies by conditioning requirements to specific data values using pattern tableaux. In particular, a single tuple may violate a conditional functional dependency.

In the research on computing optimal subset repairs, functional dependency is commonly used integrity constraint [6, 40]. For convenience of discussion, this paper also takes functional dependency as the integrity constraint like [40]. Note that, it is quite straightforward to extend our results to the other types of integrity constraints mentioned above.

Let R be a relation schema, and I be an instance of R.

Functional dependency. Let X and Y be two sets of attributes of a relation R. A functional dependency φ is an expression of the form $\mathsf{X} \to \mathsf{Y}$, which requires that there is no pair of tuples t_1, t_2 in any instance I of R such that $t_1[\mathsf{X}] = t_2[\mathsf{X}]$ but $t_1[\mathsf{Y}] \neq t_2[\mathsf{Y}]$.

Inconsistent database. Let Σ be a set of functional dependencies for relation schema R and I be an instance of R. I does not satisfy a given functional dependency $\varphi : \mathsf{X} \to \mathsf{Y}$, denoted by $I \nvDash \varphi$, if there are two tuples $t_1, t_2 \in I$ such that $t_1[\mathsf{X}] = t_2[\mathsf{X}]$ but $t_1[\mathsf{Y}] \neq t_2[\mathsf{Y}]$.

An instance I of R is said to be *consistent* with respect to Σ, denoted by $I \models \Sigma$, if I satisfies every functional dependencies in Σ. Otherwise, I is *inconsistent*, denoted by $I \nvDash \Sigma$, *i.e.*, $\exists \varphi \in \Sigma$ such that $I \nvDash \varphi$.

2.2 Subset Repairs

Let Σ be a set of functional dependencies over R. A *consistent subset* of I with respect to Σ is a subset $J \subseteq I$ such that $J \models \Sigma$. A *subset repair* (*a.k.a*, *S-repair*) is a consistent subset that is not strictly contained in any other consistent subset. Formally, a consistent subset J of I is a *subset repair* of I with respect to Σ if $J \cap K \neq J$ for any other consistent subset K of I.

Before defining the cost of a subset repair, we need to discuss the weight of each tuple. The weight of a tuple t_i is a non-negative number, denoted by w_i, representing some important information concerned by applications. The weight is defined in different ways for different applications. The followings are some examples. In the context of data repairing, the weight of a tuple usually represents the confidence of its correctness. In the context of answering aggregation queries, the weight of a tuple usually represents its contribution to query results. In the context of optimizations, the weight of a tuple represents the benefit of a tuple.

Based on the tuple weights, the cost of a subset repair $J \subseteq I$ is defined as

$$C(J, I) = \sum_{t_i \in I \setminus J} w_i,$$

which represents the cost of obtaining a subset repair. Note that the cost of a subset repair is $|I \setminus J|$, which is the distance between J and I if every tuple has weight 1. This kind of cost is commonly used in data quality management [11].

Optimal subset repair. A subset repair $J \subseteq I$ is an *optimal subset repair* of I, *optimal S-repair* for short, if

$$C(J^*, I) = \min\{C(J, I) : J \in \mathcal{S}_{I,\Sigma}\},$$

where $\mathcal{S}_{I,\Sigma}$ is the set of all subset repairs of instance I.

For example, in Figure.1(b), both plan A and B are S-repairs of procurement plan. A is an optimal S-repair since it has a minimum cost 13.7.

2.3 Problem Definition and Remarks

The problem of computing optimal subset repairs is formally defined as follows.

DEFINITION 1. (OPTSR) *Let R be a fixed relation schema and Σ be a fixed set of functional dependencies over R. For any instance I of R, the problem of* OPTSR *is to compute an optimal subset repair J^* of I with respect to Σ.*

Complexity. In practice, compared with the data size, the number of attributes in the relation is very small, and so is the number of functional dependencies. This motivates us to adopt *data complexity* as the measure to perform the complexity analysis of OPTSR. In this way, the relation schema R and the functional dependency set Σ are fixed in advance, and thus, the number of attributes in R and the number of functional dependencies can be considered as constants. The instance I of R is considered as the *input* of OPTSR. This paper focus on the data complexity.

Approximation. OPTSR is NP-hard under most settings of R and Σ. This paper will develop approximation algorithms for solving the problem OPTSR.

For later use, the approximation of optimal subset repairs is explicitly defined here. Let J^* be an optimal subset repair of I. For a constant $k \geq 1$, a subset repair J is said to be a *k-optimal S-repair* if

$$C(J^*, I) \leq C(J, I) \leq k \cdot C(J^*, I)$$

In particular, an *optimal S-repair* of any instance I is a 1-optimal S-repair of I.

For example, in Figure.1(b), plan B is a 1.5-optimal S-repair of procurement plan since its cost $16.6 \leq 1.5 \times 13.7$.

3. COMPLEXITY OF PROBLEM OPTSR

In this section, we prove a tighter lower bound of approximation ratio for OPTSR problem. In the rest of the paper, we refer the lower bound of approximation ratio and the approximation ratio as lower bound and ratio respectively.

As shown in [40], it is possible to decide for any input FD set Σ if OPTSR is polynomially solvable using the procedure $\mathsf{OSRSucceed}$. Basically, $\mathsf{OSRSucceed}$ iteratively reduces the left hand side of each FD in Σ according to three simple rules. It terminates in polynomial time until any FD in Σ cannot be further simplified, and returns *true* if there are only FDs of the form $\emptyset \to X$ left. It is proved that OPTSR is polynomially solvable if $\mathsf{OSRSucceed}(\Sigma)$ returns *true*. Otherwise, Σ makes OPTSR polynomially unsolvable, and there is a fact-wise reduction[1] from one of the following four special FD sets to Σ,

$$\begin{aligned}
\Sigma_{A\to B\to C} &= \{A \to B, B \to C\},\\
\Sigma_{A\to B\leftarrow C} &= \{A \to B, C \to B\},\\
\Sigma_{AB\to C\to B} &= \{AB \to C, C \to B\},\\
\Sigma_{AB\leftrightarrow AC\leftrightarrow BC} &= \{AB \to C, AC \to B, BC \to A\}.
\end{aligned}$$

That is, if OPTSR is NP-hard for a given Σ, then one of the following cases must happen,

Case 1. There is a fact-wise reduction from $\Sigma_{A\to B\to C}$ or $\Sigma_{A\to B\leftarrow C}$ to Σ, and it is NP-hard to compute a $(67/66-\epsilon)$-optimal S-repair.

[1]Fact-wise reduction is a kind of strict reduction that is formally defined in [40]. For any $\epsilon > 0$, if problem B has a $(1 + \epsilon)$-approximation, then problem A has a $(1 + \epsilon)$-approximation whenever there is a fact-wise reduction from A to B.

Case 2. There is a fact-wise reduction from $\Sigma_{AB\to C\to B}$ to Σ, and it is NP-hard to compute a $(57/56 - \epsilon)$-optimal S-repair.

Case 3. There is a fact-wise reduction from $\Sigma_{AB\leftrightarrow AC\leftrightarrow BC}$ to Σ, and it is NP-hard to compute a $(1+\epsilon)$-optimal S-repair for a certain $\epsilon > 0$.

Therefore, the lower bound of problem OPTSR can be improved by finding a tighter lower bound for the four special FD sets.

In this section, we derive a tighter lower bound for case 1 and 2. Moreover, instead of the lengthy proof, we provide a much conciser proof where all the reductions are built from the problem MAX-NM-E3SAT [32]. For case 3, we carefully merge four previous reductions to derive the bound, which is not shown explicitly in previous research.

LEMMA 1. *Let FD set be one of $\Sigma_{A\to B\to C}$, $\Sigma_{A\to B\leftarrow C}$ and $\Sigma_{AB\to C\to B}$. Given an $\epsilon > 0$, it is NP-hard to compute a $(17/16 - \epsilon)$-optimal S-repair for OPTSR even if every tuple in the instance has weight 1.*

PROOF. We here show three similar reductions. All of them are built from problem MAX-NM-E3SAT [32] which cannot be approximated better than 7/8. Note that every input expression ϕ of problem MAX-NM-E3SAT is subject to the following restrictions, (i) each clause has exactly three variables, (ii) each clause is *monotone* as either "$x + y + z$" or "$\bar{x} + \bar{y} + \bar{z}$", and (iii) any variable x does not occur more than once in each clause.

Specifically, each reduction builds an instance I_ϕ of the relation schema $R(A, B, C)$, in which every tuple has weight 1, for each expression ϕ of MAX-NM-E3SAT.

Reduction for $\Sigma_{A\to B\to C}$. First, two tuples (j, x_j, x_j) and $(j, x_j, \bar{x}_j)$ are inserted into I_ϕ for each variable x_i. Second, a tuple is inserted into I_ϕ for each clause c_i and each variable x_j in it as follows.

1. If c_i contains a positive literal of variable x_j, insert (c_i, x_j, x_j) into I_ϕ,
2. If c_i contains a negative literal of variable x_j, insert $(c_i, x_j, \bar{x}_j)$ into I_ϕ.

Suppose there are n variables and m clauses in ϕ, then exactly $2n+3m$ tuples are created in total. Intuitively, $A \to B$ guarantees that exactly one of its three corresponding tuples survives in any S-repair once a clause is satisfied. $B \to C$ guarantees that the assignment of each variable should be *valid*, *i.e.*, either true or false, but not both.

Reduction for $\Sigma_{A\to B\leftarrow C}$. First, two tuples (j, x_j, x_j) and $(j, \bar{x}_j, x_j)$ are inserted into I_ϕ for each variable x_i. Second, a tuple is inserted into I_ϕ for each clause c_i and each variable x_j in it as follows.

1. If c_i contains a positive literal of variable x_j, insert (c_i, x_j, x_j) into I_ϕ,
2. If c_i contains a negative literal of variable x_j, insert $(c_i, \bar{x}_j, x_j)$ into I_ϕ.

Suppose ϕ contains n variables and m clauses, then exactly $2n + 3m$ tuples are created in total. Similarly, $A \to B$ guarantees that exactly one of its three corresponding tuples survives in any S-repair once a clause is satisfied. $C \to B$ guarantees the consistency of variable assignment.

Reduction for $\Sigma_{AB\to C\to B}$. First, two tuples $(x_i, 1, x_i)$ and $(x_i, 0, x_i)$ are inserted into I_ϕ for each variable x_i. Second, a tuple is inserted into I_ϕ for each clause c_i and each variable x_j in it as follows,

1. If c_i contains a positive literal of variable x_j, insert $(c_i, 1, x_j)$ into I_ϕ,

2. If c_i contains a negative literal of variable x_j, insert $(c_i, 0, x_j)$ into I_ϕ.

Suppose there are n variables and m clauses in ϕ, then exactly $2n + 3m$ tuples are created in total. $AB \to C$ guarantees that exactly one of the three tuples survives in any S-repair once the corresponding clause is satisfied. $C \to B$ guarantees the consistency of variable assignment.

Deriving lower bound. We here only show the process of deriving the lower bound for $\Sigma_{AB\to C\to B}$. The processes of deriving other two are similar. Let ϕ be an input expression of problem MAX-NM-E3SAT, and I_ϕ be the instance data built by our reduction.

The functional dependency $AB \to C$ guarantees that any S-repair J of I_ϕ contains at most one of the three tuples with the same value c_i on attribute A, where $1 \le i \le m$. The functional dependency $C \to B$ guarantees that any S-repair J of I_ϕ contains either $(c_i, 1, x_j)$ or $(c_i, 0, x_j)$ for $1 \le i \le m$ and $1 \le j \le n$.

Therefore, we can derive a variable assignment τ from every S-repair J,

$$s.t. \qquad \tau(x_i) = \begin{cases} 1, & \text{if } (x_i, 1, x_i) \in J, \\ 0, & \text{otherwise.} \end{cases}$$

$\tau(\cdot)$ is valid such that $\tau(x_i)$ either 0 or 1, but not both, for each $1 \le i \le n$.

Let J^* be an optimal S-repair. It can be proved that $I_\phi \setminus J^*$ does not contain both $(x_i, 1, x_i)$ and $(x_i, 0, x_i)$ for each variable x_i. Since if it is not true, there must be some i such that both $(x_i, 1, x_i)$ and $(x_i, 0, x_i)$ are in $I_\phi \setminus J^*$. We can pick one of them and insert it into J^* without resulting in inconsistency. Then, an S-repair J larger than J^* is found, *i.e.*, $C(J, I_\phi) > C(J^*, I_\phi)$. This is a contradiction.

Let τ be an arbitrary valid variable assignment, and $\mathcal{N}(\phi)$ be the number of clauses in ϕ satisfied by τ. Let $\tau_{\max}$ be an optimal variable assignment, and $\mathcal{N}_{\max}(\phi)$ be the number of clauses in ϕ satisfied by $\tau_{\max}$. Thus, we have

$$\mathcal{N}_{\max}(\phi) = |I_\phi| - |I_\phi \setminus J^*| - n$$

and for any solution J of I_ϕ,

$$\mathcal{N}(\phi) \ge |I_\phi| - |I_\phi \setminus J| - n.$$

Additionally, we have $|I_\phi| = 2n + 3m$.

Let $k > 1$ and J be a k-optimal S-repair of I_ϕ such that

$$|I_\phi \setminus J^*| \le |I_\phi \setminus J| \le k \cdot |I_\phi \setminus J^*|,$$

then we have

$$\begin{aligned} \frac{\mathcal{N}(\phi)}{\mathcal{N}_{\max}(\phi)} &\ge \frac{|I_\phi| - |I_\phi \setminus J| - n}{|I_\phi| - |I_\phi \setminus J^*| - n} \\ &\ge \frac{|I_\phi| - k \cdot |I_\phi \setminus J^*| - n}{|I_\phi| - |I_\phi \setminus J^*| - n} \\ &= 1 + (1-k) \cdot \frac{|I_\phi \setminus J^*|}{|I_\phi| - |I_\phi \setminus J^*| - n}. \end{aligned} \quad (1)$$

Since each clause has exactly three literals, we have

$$|I_\phi \setminus J^*| \ge n + 2 \cdot \frac{|I_\phi| - 2n}{3}.$$

Apply this fact in the right hand of inequality (1), the following inequality holds

$$\frac{|I_\phi \setminus J^*|}{|I_\phi| - |I_\phi \setminus J^*| - n} \ge \frac{2|I_\phi| - n}{|I_\phi| - 2n} = 2 + \frac{3}{\frac{|I_\phi|}{n} - 2}.$$

Since $|I_\phi| = 2n + 3m > 2n$, we get

$$\frac{|I_\phi \setminus J^*|}{|I_\phi| - |I_\phi \setminus J^*| - n} > 2. \qquad (2)$$

Apply inequality (2) into inequality (1), then

$$\frac{\mathcal{N}(\phi)}{\mathcal{N}_{\max}(\phi)} > 3 - 2k$$

This inequality implies the problem MAX-NM-E3SAT can be approximated with ratio $3 - 2k$ if a k-optimal S-repair can be computed in polynomial time.

Suppose $k < 17/16$, then $3 - 2k > 7/8$, which implies the MAX-NM-E3SAT problem admits an approximation with ratio better than $7/8$. This contraries to the hardness result obtained in [32]. Therefore, $k \ge 17/16$.

Finally, the same bound can be derived in the same way for $\Sigma_{A\to B\to C}$ and $\Sigma_{A\to B\leftarrow C}$. Then the lemma follows immediately. $\square$

The following lemma derives the explicit lower bound for $\Sigma_{AB\leftrightarrow AC\leftrightarrow BC}$.

Lemma 2. *For $\Sigma_{AB\leftrightarrow AC\leftrightarrow BC}$, it is NP-hard to compute a $(69246103/69246100 - \epsilon)$-optimal S-repair for any $\epsilon > 0$, even if every tuple in the instance has weight 1.*

Proof. By merging the following $\mathcal{L}_{\alpha,\beta}$-reductions given in previous literatures,

MAX B29-3SAT	$\prec_{529,1}$	3DM [33]
3DM	$\prec_{1,1}$	MAX 3SC [33]
MAX 3SC	$\prec_{55,1}$	MECT-B [3]
MECT-B	$\prec_{\frac{7}{6},1}$	OptSR(R, $\Sigma_{AB\leftrightarrow AC\leftrightarrow BC}$) [40]

we can conclude that MAX B29-3SAT can be approximated within 680/679 if a $(69246103/69246100 - \epsilon)$-optimal S-repair can be computed in polynomial time for any $\epsilon > 0$ when Σ is $\Sigma_{AB\leftrightarrow AC\leftrightarrow BC}$, which is contrary to the hardness result shown in [23]. $\square$

Based on Lemma 1 and 2, the following theorem gives a new complexity result.

Theorem 1. *Given a fixed schema R and a fixed FD set Σ, for any $\epsilon > 0$,*

1. *If* OSRSucceed(Σ) *returns* true, *then an optimal S-repair can be computed in polynomial time.*

2. *If* OSRSucceed(Σ) *returns* false, *and there is a fact-wise reduction from $\Sigma_{AB\leftrightarrow AC\leftrightarrow BC}$ to Σ, then it is NP-hard to compute a $(69246103/69246100 - \epsilon)$-optimal S-repair.*

3. *Otherwise, it is NP-hard to compute a $(17/16 - \epsilon)$-optimal S-repair.*

Proof. Case *1* has been proved by [40]. Case *2* is proved by Lemma 2. Case *3* is proved by Lemma 1. $\square$

In Figure.1(b), the two functional dependencies form the case $\Sigma_{AB\to C\to B}$. Our result implies that no polynomial algorithm could compute a solution with a constant ratio better than 17/16 for the case in Figure.1(b).

Relation with Vertex Cover. In general, OptSR is equivalent to the weighted minimum vertex cover problem (WMVC) if the schema is not fixed and the number of functional dependencies is unbounded. Therefore, approximation algorithms for WMVC could be directly applied to computing approximations of optimal S-repairs. On the other side, the current best known approximation ratio for WMVC is $(2-\Theta(1/\sqrt{\log n}))$ [34]. However, this is a ratio depending on the size of input. In fact, there is no polynomial time $(2-\epsilon)$-approximation algorithm for WMVC [36] and $\epsilon > 0$ if the unique game conjecture [35] is true. This rules out the possibility to compute a $(2-\epsilon)$-optimal S-repair for any constant $\epsilon > 0$ unless unique game conjecture is false.

4. APPROXIMATION ALGORITHMS

Three approximation algorithms are developed for OptSR in this section.

The first algorithm, named BL-LP, is a baseline algorithm. It returns a $(2-1/n)$-optimal S-repair. This approximation ratio dependents on the size of input data.

The second algorithm, named TE-LP, is an improved algorithm. It returns a $(2-1/2^{\sigma-1})$-optimal S-repair, where σ is the number of functional dependencies in Σ and it is a constant.

The third algorithm, named et-TE-LP, is an extension of algorithm TE-LP. It returns $(2-1/2^{\sigma-1}-\epsilon)$-optimal S-repair where $\epsilon \geq 0$ is still a factor independent of the data size. It outperforms TE-LP when there are a large amount of *quasi*-Turán clusters in the input database.

4.1 Algorithm BL-LP

We start from a half-integral linear programming formulated for OptSR, then show a basic rounding strategy to obtain an approximation to optimal subset repairs, and finally analyze the performance of the algorithm.

4.1.1 A Half-Integral Linear Programming

For an input I of OptSR, we define a 0-1 variable x_i for each tuple $t_i \in I$ as follows,

$$x_i = \begin{cases} 0, & \text{if } t_i \text{ is in a feasible S-repair } J, \\ 1, & \text{otherwise,} \end{cases}$$

OptSR can be formulated as the following 0-1 integer linear programming $\mathcal{M}_{\text{ILP}}$,

$$\begin{aligned} &\textit{minimize} && \textstyle\sum_{t_i\in I} w_i x_i \\ &\textit{s. t.} && x_i + x_j \geq 1, \quad \forall\ t_i, t_j \in I, \{t_i, t_j\} \not\models \Sigma, \\ & && x_i \in \{0,1\}, \quad \forall\ t_i \in I. \end{aligned}$$

We could routinely relax $\mathcal{M}_{\text{ILP}}$ to a linear program where any solution $\mathbf{x}$ takes value from $[0,1]^n$. Furthermore, every extreme point of the linear program relaxation takes value from $\{0,\ 1/2,\ 1\}^n$ as shown in [43]. This gives the following half-integer programming $\mathcal{M}_{\text{HIP}}$,

$$\begin{aligned} &\textit{minimize} && \textstyle\sum_{t_i\in I} w_i x_i \\ &\textit{s. t.} && x_i + x_j \geq 1, \quad \forall\ t_i, t_j \in I, \{t_i, t_j\} \not\models \Sigma, \\ & && x_i \in \{0,1/2,1\}, \quad \forall\ t_i \in I. \end{aligned}$$

Let $\mathbf{x}^*$ be the optimal solution of $\mathcal{M}_{\text{ILP}}$, and $\mathbf{x}'$ be the optimal solution of $\mathcal{M}_{\text{HIP}}$, then

$$\sum_{t_i\in I} w_i x'_i \leq \sum_{t_i\in I} w_i x^*_i \tag{3}$$

4.1.2 Σ-Partition

To obtain a good approximation from the $\mathcal{M}_{\text{HIP}}$, we need first to partition the input I with respect to Σ, and then do the rounding based on it. Intuitively, all the tuples of I are grouped into non-empty consistent subsets in such a way that every tuple is in exactly one consistent subset.

A Σ-partition of instance I, say $\mathcal{P}(I,\Sigma)$, is a collection of disjoint consistent subsets of I such that

- $\forall K \in \mathcal{P}(I,\Sigma), K \models \Sigma$,
- $\bigcup_{K\in\mathcal{P}(I,\Sigma)} K = I$,
- $\forall K_i, K_j \in \mathcal{P}(I,\Sigma), K_i \cap K_j = \emptyset$.

The size of a Σ-partition is the number of sets in it. The smallest Σ-partition is preferred to do the rounding. Unfortunately, the size of a Σ-partition may be as large as n. When all the tuples in I are pairwise conflicting with each other, the size of any Σ-partition is n. Therefore, the size of a Σ-partition cannot be bounded by a constant.

We use greedy search to find a Σ-partition. For each tuple $t \in I$, the greedy search works as follows.

1. Find an existing consistent set $K \in \mathcal{P}$ such that $K \cup \{t\} \models \Sigma$ and put t into K.
2. If no such K can be found, create a new set $K = \{t\}$ and $\mathcal{P}(I,\Sigma) = \mathcal{P}(I,\Sigma) \cup K$.
3. Until all the tuples are picked into a consistent set, a Σ-partition $\mathcal{P}(I,\Sigma)$ is obtained.

4.1.3 Approximate via Σ-Partition-Based Rounding

A solution of $\mathcal{M}_{\text{HIP}}$ can be obtained by a Σ-partition-based rounding. When a Σ-partition $\mathcal{P}(I,\Sigma)$ is found, the rounding of solution $\mathbf{x}'$ of $\mathcal{M}_{\text{HIP}}$ is carried out as follows.

1. Let $x_i = x'_i$ if $x'_i \neq 1/2$.
2. Let $x_i = 0$ if $x'_i = 1/2$ and $t_i \in K^*$ such that

$$K^* \leftarrow \arg\max_{K_j\in\mathcal{P}(I,\Sigma)} \left|\{x'_i |\, t_i \in K_j \wedge x'_i = 1/2\}\right|,$$

otherwise, let $x_i = 1$.

3. Output the S-repair $J = \{t_i : x_i = 0\}$.

The baseline algorithm BL-LP based on the rounding method above is presented as follows.

Algorithm 1 BL-LP

Input: Instance I of R, FD set Σ, and Σ-partition $\mathcal{P}(I,\Sigma)$

Output: Approximate optimal S-repair

```
1: Formulate the $\mathcal{M}_{\text{HIP}}$ with respect to $I$ and $\Sigma$;
2: Solve $\mathcal{M}_{\text{HIP}}$ by LP-solver to obtain a solution $\mathbf{x}'$;
3: Let $J \leftarrow \emptyset$;
4: Find $K^*$ from $\mathcal{P}(I,\Sigma)$ such that
       $K^* \leftarrow \arg\max_{K_j\in\mathcal{P}(I,\Sigma)} |\{x'_i|\, t_i \in K_j \wedge x'_i = 1/2\}|$;
5: for each $t_i \in I$ do
6:     if $x'_i = 0$ or ($x'_i = 1/2$ and $t_i \in K^*$) then
7:         $J \leftarrow J \cup \{t_i\}$;
8: return $J$
```

4.1.4 Performance Analysis of BL-LP

THEOREM 2. *Let $p \geq 2$ be the size of the Σ-partition of inconsistent instance I. Algorithm BL-LP returns a $(2-2/p)$-optimal S-repair in $O(n^2)$ time.*

PROOF. We first prove J is an S-repair. Let $\mathbf{x}'$ be the solution of $\mathcal{M}_{\text{HIP}}$. Let t_i and t_j be any two tuples such that $\{t_i, t_j\} \not\models \Sigma$. Without loss of generality, we assume $x'_i \geq x'_j$. Clearly, either $x'_i = 1$ or $x'_i = x'_j = 1/2$.

When $x'_i = 1$, $t_i \notin J$ due to the rounding.

When $x'_i = x'_j = 1/2$, the following four statements hold due to the rounding. First, if $t_i, t_j \notin K^*$, $t_i, t_j \notin J$. Second, if $t_i \in K^*, t_j \notin K^*$, $t_j \notin J$. Third, if $t_i \notin K^*, t_j \in K^*$, $t_i \notin J$. Finally, if $t_i, t_j \in K^*$, $t_i, t_j \notin J$.

Therefore, J is a consistent subset. Since $\mathbf{x}'$ is the optimal solution of $\mathcal{M}_{\text{HIP}}$, J is maximal. Hence, J is an S-repair.

Now, we analyze the performance of BL-LP. Let

$$\begin{aligned} S_1 &= \{x'_i : \ t_i \in I, x'_i = 1\}, \\ S_{1/2} &= \{x'_i : \ t_i \in I, x'_i = 1/2\} \quad \text{and} \\ S_{K^*} &= \{x'_i : \ t_i \in K^*, x'_i = 1/2\}. \end{aligned}$$

Thus, J satisfies the following inequality,

$$\begin{aligned} C(J,I) &= \sum_{x'_i \in S_1} w_i x'_i + 2\left(\sum_{x' \in S_{1/2}} w_i x'_i - \sum_{x' \in S_{K^*}} w_i x'_i\right) \\ &\leq \sum_{x'_i \in S_1} w_i x'_i + 2\left(\sum_{x'_i \in S_{1/2}} w_i x'_i - 1/p \cdot \sum_{x'_i \in S_{1/2}} w_i x'_i\right) \\ &\leq \sum_{x'_i \in S_1} w_i x'_i + (2-2/p) \cdot \sum_{x'_i \in S_{1/2}} w_i x'_i \\ &\leq (2-2/p) \sum_{x'_i \in S_1 \cup S_{1/2}} w_i x'_i \end{aligned}$$

Let $\mathbf{x}^*$ be the optimal solution of $\mathcal{M}_{\text{ILP}}$, J^* be the optimal S-repair. Due to inequality (3), we have

$$\sum_{x'_i \in S_1 \cup S_{1/2}} w_i x'_i = \sum_{t_i \in I} w_i x'_i \leq \sum_{t_i \in I} w_i x^*_i = C(J^*, I)$$

Therefore, $C(J,I) \leq (2-2/p) \cdot C(J^*, I)$.

Algorithm BL-LP takes $O(\sigma n^2)$ time to formulate the $\mathcal{M}_{\text{HIP}}$ and find a Σ-partition. Then, by calling the existing fast LP-solver, it takes $O(n)$ time to solve the $\mathcal{M}_{\text{HIP}}$ and do the rounding. Therefore, the time complexity of algorithm BL-LP is $O(n^2)$ since σ is a constant. □

Remarks. The value of p in Theorem 2 depends on the value distribution over input instance I. As mentioned above, p may be as large as n. Hence, algorithm BL-LP may returns a $(2-o(1))$-optimal S-repair. We next show algorithm TE-LP with a much better approximation ratio.

4.2 Algorithm TE-LP

In this subsection, we develop an improved algorithm TE-LP which returns a $(2-1/2^{\sigma-1})$-optimal S-repair, where σ is the constant number of functional dependencies in Σ. Formally, let J^* be an optimal S-repair, algorithm TE-LP requires $O(n^2)$ time and returns a solution J such that

$$C(J^*, I) \leq C(J, I) \leq (2-1/2^{\sigma-1}) \cdot C(J^*, I) \qquad (4)$$

4.2.1 Overview of algorithm TE-LP

The algorithm TE-LP consists of three main steps as shown in algorithm 2.

Algorithm 2 TE-LP

```
Input:  Instance $I$ of $R$, FD set $\Sigma$
Output: Approximate optimal S-repair $J$
1: $J \leftarrow \emptyset$;
2: $I' \leftarrow TrimInstance(I, \Sigma)$;
3: $\mathcal{P}(I', \Sigma) \leftarrow FindPartition(I', \Sigma)$;
4: $J \leftarrow$ BL-LP $(I', \Sigma, \mathcal{P}(I', \Sigma))$;
5: return $J$
```

Step 1 Trim instance. It first trims instance I by eliminating all the *triads*, where a triad is a set of three tuples such that they are pairwise conflicting with each other.

Step 2 Find partition. After trimming off all the triads in I, $I' \subseteq I$ is obtained whose Σ-partition has a constant size. Algorithm TE-LP turns to compute such a small Σ-*partition* of I'.

Step 3 Approximate. Algorithm TE-LP runs BL-LP on the trimmed instance I', and computes an S-repair with a better ratio based on the small Σ-partition.

We next detail each step and prove the related theoretical results.

4.2.2 Trim Instance by Triad Elimination

Triad. Any subset $T = \{t_i, t_j, t_k\} \subseteq I$ is a *triad* if

- $w_i \neq 0$, $w_j \neq 0$, $w_k \neq 0$,
- $\exists \varphi \in \Sigma, \quad \{t_i, t_j\} \not\models \varphi, \{t_i, t_k\} \not\models \varphi, \{t_j, t_k\} \not\models \varphi$.

For example, in Figure.1(a), $\{r_1, r_2, r_3\}$ is a triad, and there is no triad in Figure.1(b). Actually, any three of $\{r_1, r_2, r_3, r_4\}$ is a triad.

Due to the definition of S-repair, we have the following observation.

OBSERVATION 1. *For any triad $T \subseteq I$, any S-repair J of I contains at most one tuple of T.*

By the observation, we can trim off all the triads of input instance I without a significant loss of approximation accuracy. Then the resultant instance has a small Σ-partition.

Algorithm 3 *TrimInstance*

```
Input:  Instance $I$ of $R$ and FD set $\Sigma$
Output: Instance $I'$ with no triad in it
1: $H \leftarrow \emptyset$;
2: while there exists a triad in $I$ do
3:     find a triad $T = \{t_i, t_j, t_k\} \subseteq I$;
4:     $w(T) \leftarrow \min\{w_i, w_j, w_k\}$;
5:     for each $z \in \{i, j, k\}$ do
6:         reweight $t_z$ such that $w_z \leftarrow w_z - w(T)$;
7:         if $w_z = 0$ then $H \leftarrow H \cup \{t_z\}$;
8: $I' \leftarrow I \setminus H$;
9: return $I'$
```

Trim Instance. As shown in algorithm 3, *TrimInstance* obtains a new instance I' in the following steps.

Step 1. Find a triad $T \subseteq I$.

Step 2. Suppose $T = \{t_1, t_2, t_3\}$ and $w_1 \leq w_2 \leq w_3$. Reweight each tuple by reducing w_1, *i.e.*, $w_1 \leftarrow 0$, $w_2 \leftarrow w_2 - w_1$, $w_3 \leftarrow w_3 - w_1$.

Step 3. Remove tuples with weight 0 from instance I.

Step 4. Repeat steps 1-3 until no more triad. Instance $I' \subseteq I$ is obtained.

When the iteration terminates, a new instance I' is obtained. Obviously, some tuples in I' are reweighted while the others may not. Nevertheless, the weight of each tuple never increases.

The following lemma shows that trimming off triads does not result in a significant loss of approximation ratio.

LEMMA 3. *Let $\mathcal{T} = \{T_1, \ldots, T_m\}$ be a sequence of triads processed by TrimInstance as shown in algorithm 3. For each tuple $t_z \in I$ and each $1 \le h \le m$, suppose $w_z^{(h-1)}$ is the weight of t_z before trimming off T_h, and $w_z^{(h)}$ is the weight of t_z after trimming off T_h. For each $1 \le h \le m$, let the h-th triad $T_h = \{t_i, t_j, t_k\} \in \mathcal{T}$, and*

$$w(T_h) = \min\{w_i^{(h-1)}, w_j^{(h-1)}, w_k^{(h-1)}\},$$

then for any consistent subset J of I, we have

$$\sum_{T_h \in \mathcal{T}} 2w(T_h) \le \sum_{t_i \in I \setminus J, w_i^{(m)} < w_i} \left(w_i - w_i^{(m)}\right).$$

PROOF. If all triads of $\mathcal{T}$ are disjoint, this lemma holds immediately. In case of the triads of $\mathcal{T}$ are not necessarily disjoint, we prove the lemma by induction.

Base case 1. When $|\mathcal{T}| = 0$, there is no triad in I, and hence no tuple is reweighted. Then, we have

$$\sum_{T_h \in \mathcal{T}} 2w(T_h) = 0 \le \sum_{t_i \in I \setminus J^*, w_i^{(0)} < w_i} \left(w_i - w_i^{(0)}\right) = 0$$

Base case 2. When $|\mathcal{T}| = 1$, only one triad is processed. Let $\mathcal{T} = \{T\}$, $T = \{t_1, t_2, t_3\}$ and $w(T) = \min\{w_1, w_2, w_3\}$. Thus,

$$\sum_{T_h \in \mathcal{T}} 2w(T_h) = 2w(T).$$

At the same time, OBSERVATION 1 states that any consistent subset J of I contains at most one tuple of T. Hence, $I \setminus J$ contains at least two tuples of T. Let $t_2, t_3 \in I \setminus J$.

After trimming off T, tuples t_2 and t_3 are reweighted as $w_2^{(1)} = w_2 - w(T)$ and $w_3^{(1)} = w_3 - w(T)$, that is

$$\sum_{T_h \in \mathcal{T}} 2w(T_h) = 2w(T) = (w_2 - w_2^{(1)}) + (w_3 - w_3^{(1)})$$

Therefore,

$$\sum_{T_h \in \mathcal{T}} 2w(T_h) \le \sum_{t_i \in I \setminus J, w_i^{(1)} < w_i} \left(w_i - w_i^{(1)}\right).$$

Inductive step. Suppose the lemma holds for any $|\mathcal{T}| < m$. Consider $|\mathcal{T}| = m$. Without loss of generality, let $T_1 = \{t_1, t_2, t_3\}$ be the first triad trimmed off by TE-LP, and suppose $w_1 \le w_2 \le w_3$. Thus, $w(T_1) = w_1 = \min\{w_1, w_2, w_3\}$ and we have

$$\sum_{T_h \in \mathcal{T}} 2w(T_h) = 2w_1 + \sum_{T_h \in \mathcal{T} \setminus \{T_1\}} 2w(T_h) \tag{5}$$

After trimming off T_1 from I, a new instance $I^{(1)}$ is obtained. $I^{(1)} = I \setminus \{t_1\}$ if $w_2 > w_1, w_3 > w_1$. $I^{(1)} = I \setminus \{t_1, t_2\}$, if $w_2 = w_1, w_3 > w_1$. $I^{(1)} = I \setminus \{t_1, t_2, t_3\}$, if $w_2 = w_1, w_3 = w_1$.

Let J be an arbitrary consistent subset of I, $J \cap I^{(1)}$ is still a consistent subset of $I^{(1)}$. Now, $|\mathcal{T} \setminus \{T_1\}| < m$. Due to the assumption,

$$\sum_{T_h \in \mathcal{T} \setminus \{T_1\}} 2w(T_h) \le \sum_{t_i \in I^{(1)} \setminus J, w_i^{(m)} < w_i^{(1)}} \left(w_i^{(1)} - w_i^{(m)}\right)$$

Since only tuples of T_1 are reweighted, we have

$$w_i^{(1)} = \begin{cases} w_i - w_1, & i \in \{1, 2, 3\}, \\ w_i, & \text{otherwise}, \end{cases}$$

By inequality (5), we have

$$\begin{aligned} \sum_{T_h \in \mathcal{T}} 2w(T_h) &= 2w_1 + \sum_{T_h \in \mathcal{T} \setminus \{T_1\}} 2w(T_h) \\ &\le 2w_1 + \sum_{t_i \in I^{(1)} \setminus J, w_i^{(m)} < w_i^{(1)}} \left(w_i^{(1)} - w_i^{(m)}\right) \\ &\le 2w_1 + \sum_{t_i \in I \setminus J, w_i^{(m)} < w_i^{(1)}} \left(w_i^{(1)} - w_i^{(m)}\right) \\ &= 2w_1 + \sum_{\substack{t_i \in I \setminus J, \\ w_i^{(m)} < w_i}} \left(w_i - w_i^{(m)}\right) - 3w_1 \\ &\le \sum_{t_i \in I \setminus J, w_i^{(m)} < w_i} \left(w_i - w_i^{(m)}\right) \end{aligned}$$

The statement also holds true when $|\mathcal{T}| = m$. This establishes the inductive step. □

Intuitively, after trimming off all the triads from I, the loss of the cost of any S-repair is at most $\sum_{T_h \in \mathcal{T}} 3w(T_h)$. Lemma 3 states that the loss of the cost of an optimal S-repair is at least $\sum_{T_h \in \mathcal{T}} 2w(T_h)$. Hence, the lost of the approximation ratio does not exceed 3/2 after the trim.

4.2.3 Find A Σ-Partition with Constant Size

After trimming off all the triads in I, a new instance I' is obtained. We prove that I' has a Σ-partition with at most 2^σ consistent subsets. Note that, constant σ is the number of functional dependencies in Σ.

Let $\mathcal{P}(I, \Sigma)$ be a Σ-partition of I with respect to Σ, then it is said to be a 2^σ-Σ-partition if

$$|\mathcal{P}(I, \Sigma)| \le 2^\sigma$$

The following algorithm is developed to find a 2^σ-Σ-Partition.

Algorithm 4 *FindPartition*

```
Input:  Instance $I$ of $R$ and FD set $\Sigma$
Output: A $2^\sigma$-$\Sigma$-Partition $\mathcal{P}(I, \Sigma)$
1: Let $K_1 \leftarrow I$ and $K_p \leftarrow \emptyset, \forall 2 \le p \le 2^\sigma$;
2: $\mathcal{P}(I, \Sigma) := \{K_p : 1 \le p \le 2^\sigma\}$;
3: for each $\varphi \in \Sigma$ do
4:    for each non-empty set $K \ne \emptyset \in \mathcal{P}(I, \Sigma)$ do
5:       Find an empty set $K' = \emptyset$;
6:       while $t \in K$ do
7:          if $K' \cup \{t\} \models \varphi$ then
8:             $K' \leftarrow K' \cup \{t\}$, $K \leftarrow K \setminus \{t\}$;
9: return $\mathcal{P}(I, \Sigma)$
```

Given any instance I with no triad, *FindPartition* iteratively partitions I as follows.

Step 1. Initially, $\mathcal{P}(I,\Sigma) = \{I\}$.

Step 2. In the i-th iteration, each $K \in \mathcal{P}(I,\Sigma)$ is partitioned into at most two disjoint sets which are consistent with respect to the *i-th* functional dependency $\varphi_i \in \Sigma$.

Step 3. *FindPartition* terminates until all the functional dependencies are considered.

LEMMA 4. *Algorithm 4 returns a 2^σ-Σ-partition of the input I with no triad in $O(n^2)$ time.*

PROOF. For each $1 \le i \le \sigma$, let φ_i be the *i-th* functional dependency to be considered, and $\Sigma^{(j)} = \{\varphi_i : 1 \le i \le j\}$. Note that, $\Sigma^{(0)} = \emptyset$ and $\Sigma^{(\sigma)} = \Sigma$.

Since there are 2^σ sets in $\mathcal{P}(I,\Sigma)$ returned by algorithm 4, we only need to prove that each set $K \in \mathcal{P}(I,\Sigma)$ is consistent with respect to $\Sigma^{(i)}$ after the i-th iteration.

After the $(i-1)$-th iteration, every set $K \in \mathcal{P}(I,\Sigma)$ satisfies $K \models \Sigma^{(i-1)}$. Now, suppose after the i-th iteration, there is a set $K \in \mathcal{P}(I,\Sigma)$ such that $K \not\models \Sigma^{(i)}$. There must be at least two $t, t' \in K$ such that $\{t,t'\} \not\models \varphi_i$. At this point, there must be another tuple $t'' \in K'$ such that $\{t,t''\} \not\models \varphi_i$ and $\{t',t''\} \not\models \varphi_i$. Otherwise, either t or t' should be put into K'. Now, a triad $\{t,t',t''\}$ is found in I, it contradicts. Hence, every set $K \in \mathcal{P}(I,\Sigma)$ satisfies $K \models \Sigma^{(i)}$ after the i-th iteration. Therefore, $K \models \Sigma^{(\sigma)} = \Sigma$ for each $K \in \mathcal{P}(I,\Sigma)$ returned by algorithm 4. $\square$

4.2.4 Performance Analysis of TE-LP

THEOREM 3. *Algorithm* TE-LP *returns a $(2-1/2^{\sigma-1})$-optimal S-repair J in $O(n^2)$ time, and J satisfies*

$$C(J^*, I) \le C(J, I) \le (2 - 1/2^{\sigma-1}) \cdot C(J^*, I) \quad (6)$$

PROOF. After trimming off all the triads from I, an instance I' is obtained. The cost of solution J is

$$C(J, I) = C(J, I \backslash I') + C(J, I') \quad (7)$$

Let $\mathcal{T} = \{T_1, \ldots, T_m\}$ be the sequence of triads processed by TE-LP. For each tuple $t_i \in I'$, let $w_i^{(m)}$ be the weight of t_i after trimming all the m triads.

First, let us consider $C(J, I \setminus I')$ in equation (7). All the tuples in $I \setminus I'$ have weight 0. The weight of any tuple t_i is decreased only when trimming off some triad T_h, containing t_i, in $\mathcal{T}$. Hence,

$$C(J, I \setminus I') = \sum_{t_i \in I \backslash I'} w_i \le \sum_{T_h \in \mathcal{T}} 3w(T_h) \le 3/2 \sum_{T_h \in \mathcal{T}} 2w(T_h)$$

Let J^* be an optimal S-repair of I. By lemma 3,

$$\sum_{T_h \in \mathcal{T}} 2w(T_h) \le \sum_{t_i \in I \backslash J^*, w_i^{(m)} < w_i} \left(w_i - w_i^{(m)}\right)$$

Therefore,

$$C(J, I \setminus I') \le 3/2 \sum_{t_i \in I \backslash J^*, w_i^{(m)} < w_i} \left(w_i - w_i^{(m)}\right)$$

Note that, $\forall t_i \in (I \setminus I') \setminus J^*$, $w_i^{(m)} = 0$. Therefore, the left part can be written as

$$C(J, I \backslash I') \le 3/2 \left(\sum_{t_i \in (I \backslash I') \backslash J^*} w_i + \sum_{\substack{t_i \in I' \backslash J^*, \\ w_i^{(m)} < w_i}} \left(w_i - w_i^{(m)}\right) \right) \quad (8)$$

Second, let us consider $C(J, I')$ in equation (7). Let K^* be an optimal S-repair of I'. Thus,

$$C(K^*, I') \le C(J^*, I')$$

Let $r = 2 - 1/2^{\sigma-1}$, then by theorem 2,

$$C(J, I') \le r \cdot C(K^*, I') \le r \cdot C(J^*, I') = r \cdot \sum_{t_i \in I' \backslash J^*} w_i^{(m)}$$

Since

$$\sum_{t_i \in I' \backslash J^*} w_i^{(m)} = \sum_{\substack{t_i \in I' \backslash J^*, \\ w_i^{(m)} < w_i}} w_i^{(m)} + \sum_{\substack{t_i \in I' \backslash J^*, \\ w_i^{(m)} = w_i}} w_i,$$

we have

$$C(J, I') \le r \cdot \left(\sum_{\substack{t_i \in I' \backslash J^*, \\ w_i^{(m)} < w_i}} w_i^{(m)} + \sum_{\substack{t_i \in I' \backslash J^*, \\ w_i^{(m)} = w_i}} w_i \right) \quad (9)$$

Since $3/2 \le r$ and inequalities (8), (9) and (7), we have the bound

$$\begin{aligned} C(J, I) &\le r \cdot \left(\sum_{t_i \in (I \backslash I') \backslash J^*} w_i + \sum_{\substack{t_i \in I' \backslash J^*, \\ w_i^{(m)} < w_i}} w_i + \sum_{\substack{t_i \in I' \backslash J^*, \\ w_i^{(m)} = w_i}} w_i \right) \\ &= r \cdot \left(\sum_{t_i \in (I \backslash I') \backslash J^*} w_i + \sum_{t_i \in I' \backslash J^*} w_i \right) \\ &= r \cdot C(J^*, I) \\ &= (2 - 1/2^{\sigma-1}) \cdot C(J^*, I) \end{aligned}$$

Therefore, J is a $(2 - 1/2^{\sigma-1})$-optimal S-repair.

Algorithm TE-LP takes $O(\sigma n^2)$ time to formulate $\mathcal{M}_{\text{HIP}}$ and find a Σ-partition. Then, by calling the existing fast LP-solver, it takes $O(n)$ time to solve the $\mathcal{M}_{\text{HIP}}$ and do the rounding. Therefore, the time complexity of algorithm TE-LP is $O(n^2)$ since σ is a constant. $\square$

Remarks. The approximation ratio of TE-LP only depends on the number of functional dependencies other than the data size. For instance, it will return a 1.5-optimal S-repair for the cases $\Sigma_{A\to B\to C}$, $\Sigma_{A\to B\leftarrow C}$ and $\Sigma_{AB\to C\to B}$. And it will return a 1.75-optimal S-repair for the case $\Sigma_{AB\leftrightarrow AC\leftrightarrow BC}$, no matter how large the data is.

4.3 Algorithm et-TE-LP

TE-LP can be extended to a more efficient algorithm et-TE-LP when the input instance contains a large portion of disjoint *quasi*-Turán clusters defined as follows.

Given an integer $h > 1$, a subset $K \subseteq I$ is an *h-quasi*-Turán cluster if there exists a functional dependency of Σ, say $\varphi : \mathsf{X} \to \mathsf{Y}$, such that

1. K consists of all tuples of I sharing the same X-value, and
2. K satisfies

$$\sum_{t_i\in K} w_i \geq (h+1)\cdot \max_{y}\left\{\sum_{t_i\in K, t_i[\mathsf{Y}]=y} w_i\right\}$$

Obviously, the elmination of an *h-quasi*-Turán cluster may lead to a loss of the approximation ratio, but the loss can be bounded by a factor depending on h.

Example. Consider *BrownVotes* shown in Figure.1(a), $K = \{r_1, r_2, r_3, r_4\}$ is a 2-*quasi*-Turán cluster, becuase it consists of all tuples sharing the same value of *County*, and satisfies the constraint: $w_1 + w_2 + w_3 + w_4 \geq (2+1)\times w_3 = 3\times 560$, since $w_3 = 560$ is larger than the other three weights. Any optimal S-repair should drop the weight of $2197 - 560$, while any S-repair disjoint with K drops the weight 2197, therefore, the ratio loss is 560/2197 which is less than 1/3. □

Algorithm et-TE-LP iteratively calls TE-LP to compute the approximate S-repair. In each iteration, it first eliminate all the disjoint *h-quasi-Turán clusters* from the input instance, and then calls TE-LP to compute an approximate S-repair. At last, the best approximate S-repair among those already computed is returned. The procedure of et-TE-LP is shown in algorithm 5.

Algorithm 5 et-TE-LP

```
Input:   Instance $I$ of $R$ and FD set $\Sigma$
Output:  Approximate optimal S-repair $J$
1: Let $J \leftarrow \emptyset$ and $h = 2$;
2: for each $h < n$ do
3:     $I' \leftarrow$ EliminateQuasiTuránClusters$(I, \Sigma, h)$;
4:     $J' \leftarrow$ TE-LP$(I', \Sigma)$;
5:     if $J'$ is better than $J$ then $J \leftarrow J'$;
6: return $J$
```

4.3.1 Performance Analysis of et-TE-LP

Let $I_1^{(h)}$ be the set of all tuples eliminated by *EliminateQuasiTuránClusters* in the h-th iteration and $I_2^{(h)} = I - I_1^{(h)}$ for any integer $h > 1$. Let $J^{(h)}$ be the approximate S-repair returned by calling TE-LP in the h-th iteration of et-TE-LP. Obviously, $I_1^{(h)} \subseteq I \setminus J^{(h)}$. We define $\rho^{(h)}$ as

$$\rho^{(h)} = \frac{\sum_{t_i\in I_1^{(h)}} w_i}{\sum_{t_i\in I\setminus J^{(h)}} w_i} = \frac{C(J, I_1^{(h)})}{C(J^{(h)}, I_1^{(h)}) + C(J, I_2)}.$$

The following theorem shows the performance of et-TE-LP.

THEOREM 4. *Algorithm et-TE-LP returns a* $(2-1/2^{\sigma-1}-\epsilon)$*-optimal S-repair in* $O(n^3)$ *time, where*

$$\epsilon = \max_h\{(1 - 1/h - 1/2^{\sigma-1})\cdot \rho_h\} \geq 0$$

for $1 < h < n$ *and* $\rho_h \geq 0$.

PROOF. Let $J_1^{*(h)}$ be the optimal S-repair of $I_1^{(h)}$, then

$$\frac{C(J^{(h)}, I_1^{(h)})}{C(J_1^{*(h)}, I_1^{(h)})} \leq \frac{h+1}{h}$$

Let $J_2^{*(h)}$ be the optimal S-repair of $I_2^{(h)}$, then

$$\frac{C(J^{(h)}, I_2^{(h)})}{C(J_1^{*(h)}, I_2^{(h)})} \leq 2 - \frac{1}{2^{\sigma-1}}$$

If J^* is the optimal S-repair of I, then

$$C(J_1^{*(h)}, I_1^{(h)}) + C(J_2^{*(h)}, I_2^{(h)}) \leq C(J^*, I).$$

Now, the bound of approximation ratio is

$$\begin{aligned}
\frac{C(J^{(h)},I)}{C(J^*, I)} &\leq \frac{\frac{h+1}{h}\cdot C(J_1^{*(h)}, I_1^{(h)}) + (2-\frac{1}{2^{\sigma-1}})\cdot C(J_2^{*(h)}, I_2^{(h)})}{C(J_1^{*(h)}, I_1^{(h)}) + C(J_2^{*(h)}, I_2)} \\
&\leq 2 - \frac{1}{2^{\sigma-1}} + \frac{\left(\frac{h+1}{h} - (2-\frac{1}{2^{\sigma-1}})\right)\cdot C(J, I_1^{(h)})}{C(J^{(h)}, I_1^{(h)}) + C(J, I_2)} \\
&\leq 2 - 1/2^{\sigma-1} + \left(\frac{h+1}{h} - (2 - \frac{1}{2^{\sigma-1}})\right)\cdot \rho_h \\
&= 2 - 1/2^{\sigma-1} - (1 - 1/h - 1/2^{\sigma-1})\cdot \rho_h.
\end{aligned}$$

Since algorithm 5 finally selects the best solution,

$$\frac{C(J^{(h)}, I)}{C(J^*, I)} \leq 2 - 1/2^{\sigma-1} - \max_h\{(1 - 1/h - 1/2^{\sigma-1})\cdot \rho_h\}.$$

In each iteration, algorithm et-TE-LP takes $O(n\log n)$ time to eliminate all the *h-quasi*-Turán clusters by sorting tuples in I for each functional dependency. And et-TE-LP takes $O(n^2)$ time to run TE-LP in each iteration. Since there are n iterations in total, the time complexity of algorithm et-TE-LP is $O(n^3)$. □

Remarks. As we can see, the performance of et-TE-LP depends on the characteristics of input rather than the data size. et-TE-LP has an approximation ratio much better than TE-LP for a non-zero $\rho^{(h)}$ where $1 < h < n$.

5. EXPERIMENTS

We conducted experiments with both real-life and synthetic data to examine BL-LP, TE-LP and et-TE-LP. Given the schema and constraint set, both the efficiency and the accuracy of the three algorithms were evaluated by varying data. In addition, taking consistent query answering as an example, we discovered the effectiveness of our algorithms in the real-world applications.

5.1 Experimental Settings

Experiments are conducted on a CentOS 7 machine with eight 16-core Intel Xeon processors and 3072GB of memory.
Datasets. All the datasets used in the experiments were collected from real-world applications.
PROCP was an instance of the schema Procurement Plan shown in Figure.1(b). The data in PROCP are the online product reviews collected from AMAZON[2] and some other online sales[3]. The number of tuples was varied from 10K to 40K. The FD set Σ_1 consists of the two functional dependencies shown in Figure.1(b).
ORDER was synthesized by propagating PROCP with *zip*, *area code*[4] and *street information*[5] for US states. The number of tuples was varied from 10K to 40K. The FD set Σ_2 consists of {Area, Phone} $\rightarrow$ {Street, City, State}, {ID} $\rightarrow$ {Name, Price}, {Zip} $\rightarrow$ {City, State}, {Street, City} $\rightarrow$ {Zip}.

[2]https://data.world/datafiniti/consumer-reviews-of-amazon-products
[3]https://data.world/datafiniti/grammar-and-online-product-reviews
[4]http://www.geonames.org/
[5]http://results.openaddresses.io/

DBLP was extracted from dblp[6] of schema (title,authors, year,publisher,pages,ee,url), which has 40K tuples. The FD set Σ_3 includes {url} $\rightarrow$ {title}, {title} $\rightarrow$ {author}, {ee} $\rightarrow$ {title}, {year, publisher, pages} $\rightarrow$ {title, ee, url}.

One can easily verify that Σ_1, Σ_2 and Σ_3 cannot be simplified further by procedure OSRSucceed [40]. Hence, the problem is NP-hard in each of these cases.

Injecting Inconsistency. PROCP was used to simulate the procurement plan refinement in marketing, which is already an inconsistent database with respect to Σ_1. Inconsistent versions of ORDER and DBLP were generated by adding noises only to the attributes that are related to some functional dependencies. We refer to the initial consistent data of as I_0 and refer to the inconsistent version as I_1. The noise is controlled by the noise rate ρ. We introduced two types of noises: typos and values from active domain.

Weights. In accordance to the cost model defined in Section 2.1, we assign weights to the tuples in the following way. A tuple is called a dirty tuple that there is some attribute value of t in I_1 is different from its counterpart in I_0. For dirty tuple t of ORDER and DBLP, a random weight $w(t)$ in $[0, a]$ was assigned to t. For the other tuples t of ORDER and DBLP, a weight $w(t)$ in $[b, 1]$ was selected randomly to t. This follows the assumption commonly used in data repairing that the weight of a dirty tuple is usually slightly lower than the weight of other tuples. In the experiments, we set $a = 0.3$ and $b = 0.7$. For PROCP, in the efficiency and accuracy testing, customer rating was treated as the tuple weight, which is a number in $[0, 5]$. In the effectiveness test, the price is also treated as the tuple weight which may contribute to the results of aggregation queries.

Methods. We have implemented algorithms BL-LP, TE-LP and et-TE-LP. As a basic tool, GLPK-LP/MIP-Solver[7] was used to solve large-scale linear programming. We also have implemented L2VC as a tool to obtain an upper bound line of approximation, which is the 2-approximation algorithm for weighted minimum vertex cover given in [8]. All the algorithms were implemented in C++.

Metrics. To test the accuracy, L2VC was used as an upper bound of optimal S-repairs. To examine the efficiency, we ran each algorithms 10 times at each dataset size and plotted the average running time. To evaluate the effectiveness of our algorithms, we issued 100 pairs of aggregation queries similar to the examples shown in Figure.1(a). Each query is either `SUM` on attribute *price* of PROCP and ORDER, or `SUM` on attribute *pages* of DBLP. For each query pair (Q, Q'), either $lub\,(Q) \leq glb\,(Q')$ or $lub\,(Q) > glb\,(Q')$ holds. We used our algorithms to help deciding such relationship. Specifically, let $\tilde{J}$ be the approximate optimal S-repair, r be the approximation ratio of the algorithm used to compute $\tilde{J}$, and $W(Q)$ is the maximum value of the query result. We compute the estimation of the aggregation result of Q by

$$\widetilde{lub}\,(Q) = W(Q) - \frac{C(\tilde{J})}{r}.$$

Note that dividing r is to get rid of the false positive. To obtain the truth-ground, we employed the FPT algorithm [17] for weighted minimum vertex cover and another FPT algorithm [49] for weighted maximum minimal vertex cover to compute the exact lub and glb for the query result. Let Y_i be a 0-1 variable such that

$$Y_i = \begin{cases} 0, & \text{if } lub\,(Q_i) \leq glb\,(Q'_i) \text{ and } \widetilde{lub}\,(Q_i) \leq glb\,(Q'_i)\,, \\ 0, & \text{if } glb\,(Q_i) \geq lub\,(Q'_i) \text{ and } glb\,(Q_i) \leq \widetilde{lub}\,(Q'_i)\,, \\ 1, & \text{otherwise.} \end{cases}$$

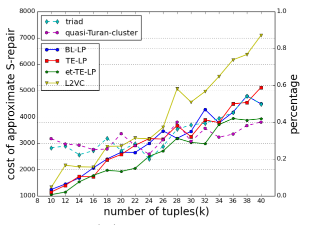
Let $Y = \sum_{i=1}^{100} Y_i$ be the number of correct decisions made by our algorithms. Due to the time and space limitation of the two FPT algorithms, $W(Q)$ was bounded by 120 by restricting the range of each aggregation in our experiments. The gap between two query results is defined as

$$gap(Q, Q') = \max\left\{\frac{lub(Q)}{glb(Q')}, \frac{glb(Q')}{lub(Q)}\right\}$$

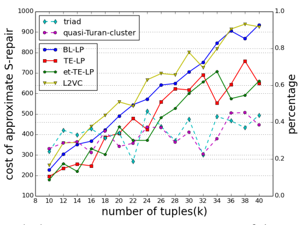
to help quantify the difficulty of query answering.

5.2 Experimental Results

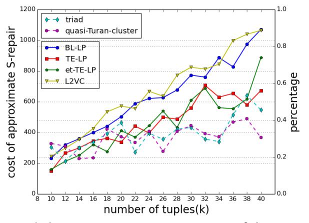
We report our findings concerning about the performance and effectiveness of our algorithms.



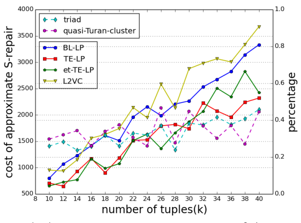
(a) PROCP



(b) ORDER and $\rho = 10\%$



(c) DBLP and $\rho = 10\%$



(d) DBLP and $\rho = 20\%$

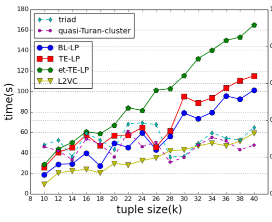
Figure 2: The accuracy of approximation algorithms

Accuracy. We first show the accuracy of BL-LP, TE-LP and et-TE-LP by varying the number of tuples and noise rate. The x-axis is the number of tuples. The left y-axis is the cost of approximate S-repairs, and the right y-axis is the percent of triads or *quasi*-Turán clusters processed during the procedure. In Figure 2, we ran them together with the upper bound L2VC on datasets consisting of 10K to 40K tuples by a step of 2K with noise rate ρ ranging from 5% to 20%. TE-LP and et-TE-LP outperformed BL-LP and L2VC on PROCP, *i.e.*, the gap in between is larger than the other cases. This is also consistent with the theoretical result, that is, their ratio bounds are less than 3/2 which are much better than 2, recall that Σ_1 contains two functional dependencies. In theory, TE-LP and et-TE-LP should not perform as well on PROCP with Σ_1, since Σ_2 and Σ_3 contain more constraints. However, the theoretical results we proved are worst-case approximation ratios. In our experiments, the accuracy was not necessarily better when the FD set Σ is smaller, because the accuracy is also affected by other factors which cannot be controlled by us, such as the ratio of the number of 1-variables to the number of 1/2-variables in the solution returned by LP solver and so on. Nevertheless, on average, TE-LP and et-TE-LP outperformed BL-LP and

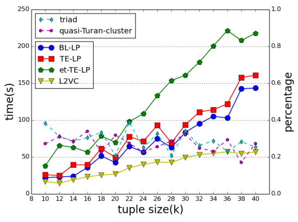
[6]https://dblp.org/xml/
[7]https://www.gnu.org/software/glpk/

L2VC in our experiments, which is evident from their average distances to BL-LP and L2VC.

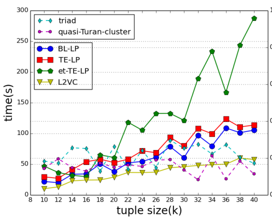
TE-LP performed much better than L2VC and BL-LP on ORDER and DBLP, as shown in Figure.2(b), 2(c) and 2(d), since a large percentage of noise was trimmed off as triads. et-TE-LP was behaving better than L2VC and BL-LP whenever the percentage of disjoint *quasi*-Turán-clusters is large. TE-LP and et-TE-LP were not doing very well on PROCP, because the size of greedy Σ-partition was always small, so that the results were not improved largely by trimming off triads or *quasi*-Turán-clusters. Moreover, et-TE-LP is even better than TE-LP when the percentage of disjoint *quasi*-Turán-clusters processed by et-TE-LP is much larger than that of triads processed by TE-LP. In addition, such two percentages always vary with the noise rate ρ. We can see that the outputs of TE-LP and et-TE-LP were getting better when ρ became large. As we can see, et-TE-LP is not significantly better than TE-LP, however, both of them are much better than the baseline algorithms in most cases.



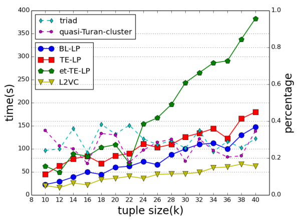
(a) PROCP



(b) ORDER and $\rho = 10\%$



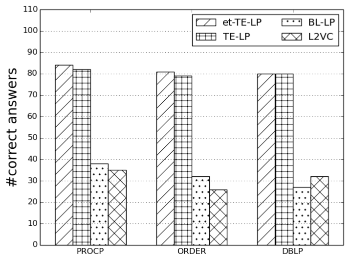
(c) DBLP and $\rho = 5\%$



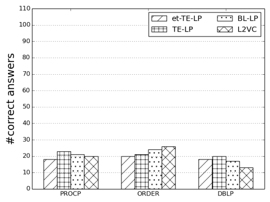
(d) DBLP and $\rho = 15\%$

Figure 3: The running time of algorithm

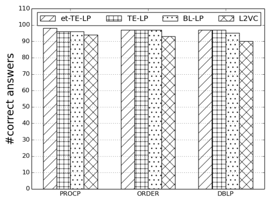
Efficiency. We evaluated the efficiency of our algorithms by varying the number of tuples and noise rate. The results for the three datasets are shown in Figure.3(a)-(d). The left x-axis is the number of tuples, the left y-axis is the running time in seconds, and the right y-axis is the percent of triads or *quasi*-Turán clusters processed during the procedure. Since L2VC does not call the LP solver, it took a running time lower than our algorithms by loosing the accuracy. In theory, et-TE-LP took the largest time cost to compute a good approximation. This is because it takes a lot of time to find h-*quasi*-Turán clusters for every possible h. In practice, after several iterations, it terminated once h is large enough, and there is almost no cluster with a large h value, therefore, et-TE-LP was almost linear with TE-LP. Observe that et-TE-LP sometimes performed very efficiently like TE-LP. This is because the disjoint *quasi*-Turán clusters covers most of the inconsistencies. TE-LP took nearly the same time cost with BL-LP to compute a good approximate S-repair since it is very fast to do the trimming and those triads covers most of the inconsistencies. In addition, as shown in Figure.3(b), TE-LP and et-TE-LP saved more time when $\rho = 10\%$ since the percentages of triads and *quasi*-Turán clusters could be processed are larger than the cases with smaller ρ.



(a) 1.2<Gap(Q,Q')<1.3



(b) Gap(Q,Q')<1.12



(c) Gap(Q,Q')>1.5

Figure 4: Number of correct answers with $\rho = 10\%$

Effectiveness. We evaluated the precision of our algorithms to help to answer aggregation queries posed on inconsistent databases. The results for dataset PROCP are given in Figure.4. In the three figures, the x-axis is for the combination of dataset and algorithms, and the y-axis is for precision. We set the noise rate at 10%. The three figures show that algorithm TE-LP and et-TE-LP are more effective in average. Specifically, Figure.4(c) and 4(a) imply that our algorithms with better ratios returned more dependable answers (*i.e.*, high precision) for those aggregation queries. When the gap is large (larger than 1.5 in Figure.4(c)), all the algorithms performed well since the it is easy to distinguish $lub(Q)$ and $glb(Q')$. When the gap is not very small (around 1.2∼1.3 in Figure.4(a)), both TE-LP and et-TE-LP outperformed BL-LP and L2VC much better. The reason is that the gap is nearly the approximation ratio of TE-LP and et-TE-LP but much lower than that of the other two. Indeed, however, all the algorithms proposed in this paper have a low precision for those query pairs with a small gap (less than 1.12 in Figure.4(b)). In such cases, the gap is nearly 0, it is hard to distinguish the values of $lub(Q)$ and $glb(Q')$ by using an approximation algorithm unless an exact algorithm. Nevertheless, when the gap is not very small, the union of TE-LP and et-TE-LP are always the current best choice for answering aggregation queries.

6. CONCLUSIONS

Optimal subset repairs are NP-hard to be approximated within 17/16 for most of the polynomial intractable cases. For the other cases, the problem approximating optimal subset repairs within 69246103/69246100 is also NP-hard. Optimal subset repairs can be approximated within a ratio $(2 - 1/2^{\sigma-1})$ if given σ functional dependencies. It is a constant better than 2. Our results give a way to compute optimal S-repairs efficiently with both theoretical and empirical guarantee. However, there is still a large gap between the upper bound and the lower bound derived in this paper. Tighter bounds on computing optimal S-repairs are suggested in the future work.

7. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases: The Logical Level.* Addison-Wesley, 1995.

[2] F. N. Afrati and P. G. Kolaitis. Repair checking in inconsistent databases: algorithms and complexity. In *ICDT*, pages 31–41, 2009.

[3] O. Amini, S. Pérennes, and I. Sau. Hardness and approximation of traffic grooming. *Theor. Comput. Sci.*, 410(38-40):3751–3760, 2009.

[4] M. Arenas, L. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *PODS*, pages 68–79, 1999.

[5] M. Arenas, L. Bertossi, and J. Chomicki. Answer sets for consistent query answering in inconsistent databases. *Theor. Pract. Log. Prog.*, 3(4):393–424, 2003.

[6] M. Arenas, L. Bertossi, J. Chomicki, X. He, V. Raghavan, and J. Spinrad. Scalar aggregation in inconsistent databases. *Theor. Comput. Sci.*, 296(3):405–434, 2003.

[7] A. Assadi, T. Milo, and S. Novgorodov. DANCE: data cleaning with constraints and experts. In *ICDE*, pages 1409–1410, 2017.

[8] R. Bar-Yehuda and S. Even. A linear-time approximation algorithm for the weighted vertex cover problem. *J. Algorithms*, 2(2):198–203, 1981.

[9] M. Bergman, T. Milo, S. Novgorodov, and W.-C. Tan. QOCO: a query oriented data cleaning system with oracles. *PVLDB*, 8(12):1900–1903, 2015.

[10] L. Bertossi. Database repairs and consistent query answering: origins and further developments. In *PODS*, pages 48–58, 2019.

[11] L. Bertossi. Repair-based degrees of database inconsistency. In *LPNMR*, pages 195–209, 2019.

[12] L. Bertossi, L. Bravo, E. Franconi, and A. Lopatenko. The complexity and approximation of fixing numerical attributes in databases under integrity constraints. *Inform. Syst.*, 33(4):407–434, 2008.

[13] P. Bohannon, W. Fan, M. Flaster, and R. Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *SIGMOD*, pages 143–154, 2005.

[14] P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for data cleaning. In *ICDE*, pages 746–755, 2007.

[15] N. Boria, F. D. Croce, and V. T. Paschos. On the max min vertex cover problem. *Discrete Appl. Math.*, 196:62–71, 2015.

[16] M. Caniupán and L. Bertossi. The consistency extractor system: answer set programs for consistent query answering in databases. *Data Knowl. Eng.*, 69(6):545–572, 2010.

[17] J. Chen, I. A. Kanj, and G. Xia. Improved upper bounds for vertex cover. *Theor. Comput. Sci.*, 411(40):3736–3756, 2010.

[18] F. Chiang and R. J. Miller. A unified model for data and constraint repair. In *ICDE*, pages 446–457, 2011.

[19] J. Chomicki and J. Marcinkowski. Minimal-change integrity maintenance using tuple deletions. *Inf. Comput.*, 197(1-2), 2005.

[20] X. Chu, I. F. Ilyas, S. Krishnan, and J. Wang. Data cleaning: overview and emerging challenges. In *SIGMOD*, pages 2201–2206, 2016.

[21] X. Chu, I. F. Ilyas, and P. Papotti. Holistic data cleaning: putting violations into context. In *ICDE*, pages 458–469, 2013.

[22] G. Cong, W. Fan, F. Geerts, X. Jia, and S. Ma. Improving data quality: consistency and accuracy. *PVLDB*, 7(6):315–325, 2007.

[23] P. Crescenzi. A short guide to approximation preserving reductions. In *CCC*, pages 262–273, 1997.

[24] M. Dallachiesa, A. Ebaid, A. Eldawy, A. Elmagarmid, I. F. Ilyas, M. Ouzzani, and N. Tang. NADEEF: a commodity data cleaning system. In *SIGMOD*, pages 541–552, 2013.

[25] C. De Sa, I. F. Ilyas, B. Kimelfeld, C. Ré, and T. Rekatsinas. A formal framework for probabilistic unclean databases. In *ICDT*, pages 26–28, 2019.

[26] A. A. Dixit. CAvSAT: a system for query answering over inconsistent databases. In *SIGMOD*, pages 1823–1825, 2019.

[27] A. A. Dixit and P. G. Kolaitis. A SAT-based system for consistent query answering. In *SAT*, pages 117–135, 2019.

[28] S. Flesca, F. Furfaro, and F. Parisi. Consistent query answers on numerical databases under aggregate constraints. In *DBPL*, pages 279–294, 2005.

[29] Gartner. *Vendor Rating Service*, accessed May 15, 2020.

[30] F. Geerts, G. Mecca, P. Papotti, and D. Santoro. The llunatic data-cleaning framework. *PVLDB*, 6(9):625–636, 2013.

[31] L. Golab, I. F. Ilyas, G. Beskales, and A. Galiullin. On the relative trust between inconsistent data and inaccurate constraints. In *ICDE*, pages 541–552, 2013.

[32] V. Guruswami and S. Khot. Hardness of Max 3SAT with no mixed clauses. In *CCC*, pages 154–162, 2005.

[33] V. Kann. Maximum bounded 3-dimensional matching is MAX SNP-complete. *Inform. Process. Lett.*, 37(1):27–35, 1991.

[34] G. Karakostas. A better approximation ratio for the vertex cover problem. *ACM Trans. Algorithms*, 5(4):41:1–41:8, 2009.

[35] S. Khot. On the unique games conjecture. In *FOCS*, page 3, 2005.

[36] S. Khot and O. Regev. Vertex cover might be hard to approximate to within 2-ϵ. *J. Comput. Syst. Sci.*, 74(3):335–349, 2008.

[37] S. Kolahi and L. V. S. Lakshmanan. On approximating optimum repairs for functional dependency violations. In *ICDT*, pages 53–62, 2009.

[38] P. G. Kolaitis, E. Pema, and W.-C. Tan. Efficient querying of inconsistent databases with binary integer programming. *PVLDB*, 6(6):397–408, 2013.

[39] P. Koutris and J. Wijsen. Consistent query answering for self-join-free conjunctive queries under primary key constraints. *ACM Trans. Database Syst.*, 42(2):1–45, 2017.

[40] E. Livshits, B. Kimelfeld, and S. Roy. Computing optimal repairs for functional dependencies. *ACM Trans. Database Syst.*, 45(1):1–46, 2020.

[41] A. Lopatenko and L. Bertossi. Complexity of consistent query answering in databases under cardinality-based and incremental repair semantics. In *ICDT*, pages 179–193, 2007.

[42] A. Lopatenko and L. Bravo. Efficient approximation algorithms for repairing inconsistent databases. In *ICDE*, pages 216–225, 2007.

[43] G. L. Nemhauser and L. E. Trotter. Vertex packings: structural properties and algorithms. *Math. Program.*, 8(4):232–248, 1975.

[44] T. Rekatsinas, X. Chu, I. F. Ilyas, and C. Ré. HoloClean: holistic data repairs with probabilistic inference. *PVLDB*, 10(11):1190–1201, 2017.

[45] B. Salimi, L. Rodriguez, B. Howe, and D. Suciu. Interventional fairness: causal database repair for algorithmic fairness. In *SIGMOD*, pages 793–810, 2019.

[46] J. Wijsen. On the consistent rewriting of conjunctive queries under primary key constraints. *Inform. Syst.*, 34(7):578–601, 2009.

[47] J. Wijsen. Certain conjunctive query answering in first-order logic. *ACM Trans. Database Syst.*, 37(2):1–35, 2012.

[48] J. Wijsen. Foundations of query answering on inconsistent databases. *SIGMOD Rec.*, 48(3):6–16, 2019.

[49] M. Zehavi. Maximum minimal vertex cover parameterized by vertex cover. *SIAM J. Discrete Math.*, 31(4):2440–2456, 2017.