2019 IEEE 35th International Conference on Data Engineering (ICDE)

Deletion Propagation for Multiple Key Preserving Conjunctive Queries: Approximations and Complexity

Zhipeng Cai* Dongjing Miao†,* Yingshu Li*

*Department of Computer Science, Georgia State University, Atlanta, Georgia
Email: {zcai,yili}@gsu.edu, dmiao1@student.gsu.edu

†School of Computer Science and Technology, Harbin Institute of Technology, Harbin, China
Email: miaodongjing@hit.edu.cn

***Abstract*—This paper studies the deletion propagation problem in terms of minimizing view side-effect. It is a problem fundamental to data lineage and quality management which could be a key step in analyzing view propagation and repairing data. The investigated problem is a variant of the standard deletion propagation problem, where given a source database D, a set of key preserving conjunctive queries $\mathcal{Q}$, and the set of views $\mathcal{V}$ obtained by the queries in $\mathcal{Q}$, we try to identify a set T of tuples from D whose elimination prevents all the tuples in a given set of deletions on views $\Delta\mathcal{V}$ while preserving any other results. The complexity of this problem has been well studied for the case with only a single query. Dichotomies, even trichotomies, for different settings are developed. However, no results on multiple queries are given which is a more realistic case. We study the complexity and approximations of optimizing the side-effect on the views, *i.e.*, find T to minimize the additional damage on $\mathcal{V}$ after removing all the tuples of $\Delta\mathcal{V}$. We focus on the class of key-preserving conjunctive queries which is a dichotomy for the single query case. It is surprising to find that except the single query case, this problem is NP-hard to approximate within any constant even for a non-trivial set of multiple *project-free* conjunctive queries in terms of view side-effect. The proposed algorithm shows that it can be approximated within a bound depending on the number of tuples of both $\mathcal{V}$ and $\Delta\mathcal{V}$. We identify a class of polynomial tractable inputs, and provide a dynamic programming algorithm to solve the problem. Besides data lineage, study on this problem could also provide important foundations for the computational issues in data repairing. Furthermore, we introduce some related applications of this problem, especially for query feedback based data cleaning.**

***Keywords*-key preserving; conjunctive query; deletion propagation; approximation**

I. Introduction

Deletion propagation is a special *view update* problem. The motivation is to enable the restricted database access via materialized views so as to translate the updates (*e.g.*, deletions) specified on views back into the source database *properly*. The major concern is that updates are usually not fully specified on multiple views defined by different queries. Therefore, the previous literature has made efforts to guarantee the correctness of translation and improved expression ability of updating interfaces by designing syntax and semantics, and deriving necessary qualifying criterion. Bancilhon and Spyratos [1] initiated a series of research by defining the *complementary* view to inherent ambiguity so as to compute the translated database. Cosmadakis and Papadimitriou [16] followed this way characterizing complementary views and extended it to functional dependencies. Keller [29] later derived translator coherence qualifying criterion. Bohannon *et al.* [4] then developed the language for updatable views with explicit update policies.

In this paper, we study another aspect of view update which is to translate updates back into source data with the minimum side-effect [6], [14], [15], [17], [18], [30]–[32] even though updates are incompletely specified. Motivative examples could be found in many real-life applications such as relational data repairing [2], [3], database debugging [32], batched view updating and so on. For example, some semi-automatic data repairing systems [2], [3] typically generate a set of queries first so as to cover the source data as many as possible or the most inconsistent part.

In these procedures, collecting feedbacks on query results is the critical step, and these feedbacks are usually obtained by (empirical) rule-based detection or user-specification (e.g., crowd or domain experts). These feedbacks usually specify errors contained in the result or missing tuples. However, the incompleteness of feedbacks may lead to the non-existence of side-effect-free updated database. Therefore, an updated database with the minimum side-effect should be a recommendation or partial suggestion of removing errors. Another example is database debugging [32], and a similar task is to find an updated database suggestion once wrong tuples in the query results are identified by users. A scenario regarding information extraction is also mentioned in [31].

The problem considered in this paper is a special case of view propagation. Given undesired tuples in the materialized views defined by conjunction queries, it is to seek a way of tuple deletion from the tables in source database to eliminate undesired tuples in the view. A solution here is the database updated by applying the deletion on the source tables. The set of tuples in the views but different from the undesired ones

2375-026X/19/$31.00 ©2019 IEEE
DOI 10.1109/ICDE.2019.00052

are called side-effect if they are eliminated by the deletion on the source database. A solution is called side-effect-free only if no tuples other than the undesired ones are eliminated. The examples mentioned above show that a side-effect-free solution does not always exist. Therefore, this series of works try to figure out the complexity and propose algorithms for the problem minimizing the side-effect [6], [15], [32], typically take the cardinality of result tuples lost as the measurement. A solution is optimal if its side-effect is of minimum cardinality. This is different from the source side-effect counterpart studied in [2], where the measurement is the cardinality of the tuples deleted in the source database in order to eliminate all the undesired view tuples.

In this paper, we focus on the more realistic extension of the cases studied in the previous research, *i.e.*, the case of multiple views. The previous works have figured out the complexity classes of most major cases taking only one view as the input, although some dichotomies and trichotomies for the computational complexity are derived in the literatures [6], [15], [30]. For example, in the case studied in [15], [30], not only the view is defined by a single query, but also the deletion on view is also a single tuple. Detailed results on the complexity of multi-tuple deletion are given in [32] where the deletion on view could be multiple tuples but view is still single. However, besides lack of approximation algorithms for polynomial intractable cases, no complexity results are known for the case of multiple views.

To conduct a non-trivial investigation on the case of multiple views, we focus on the views defined by key-preserving conjunctive queries. As shown in the previous research, on data complexity, deletion propagation with the minimum view side-effect is NP-*hard* for *sj-free* conjunctive queries in which only a single view is given. On the other hand, Miao *et al.* [36] shows that on the combined complexity aspect, minimizing view side-effect is Σ_2^{P}-*hard* for single *sj-free* conjunctive view, worst still, it is beyond NP (*i.e.*, NP(k)-*hard* for every integer k) even when the deletion could be bounded in advance based on priori knowledge. Fortunately, on the combined complexity, the case of single *project-free* conjunctive query view is shown to be polynomial tractable (*i.e.*, LOGSPACE) containing self-join and an algorithm has been developed [37]. These results imply that the projection operator in a conjunctive query produces ambiguity which increases the hardness of the reversion process to find an optimal solution. It is easy to see that a tractable condition has to be within *project-free* conjunctive fragment, otherwise NP-hard or beyond NP, because the case studied in this paper takes multiple views defined by different queries as input. Therefore, we investigate the complexity of the case taking multiple *project-free* views as input to derive polynomial tractable conditions and develop approximation for this case. Note that, other than data complexity, approximation under combined complexity is much more powerful since it could deal with views defined by more complex queries and parallel applications.

Our contributions are as follows. We first show that the previous complexity results for the single view case [31], [32] no longer hold for the multiple queries case. We derive the following new complexity results.

- Even for two *project-free* conjunctive views, the view side-effect problem cannot be approximated within $O(2^{\log^{1-\delta}\|\mathcal{V}\|})$ where $\delta = 1/\log\log^c\|\mathcal{V}\|$ for any $c < 0.5$, unless P=NP. Such lower bound can also be applied to the balanced deletion problem defined for more practical cases in Section III.
- The view side-effect problem and its balanced version can be approximated within $O(2l \cdot \|\mathcal{V}\| \cdot \log\|\Delta\mathcal{V}\|)$ for the general case, where l is the maximum *arity*(Q) among all the given queries.
- The view side-effect problem can be approximated within l and $O(2\sqrt{l \cdot \|\mathcal{V}\|})$ for the tree case.
- The view side-effect problem can be solved polynomially for the more restrictive tree case.

TABLE I
SUMMARY OF NOTATIONS

Notation	*Definition*
$\mathcal{S}$	schema
D	database instance
T	relation symbol
t	tuple symbol
$Q, Q(D), V$	database query, its result, and its materialized view
$\mathcal{Q}$	a set of queries $\{V_1, \ldots, V_n\}$
$\mathcal{V}$	a set of views $\{V_1, \ldots, V_n\}$
$\Delta\mathcal{V}$	a set of deletions $\{\Delta V_1, \ldots, \Delta V_n\}$ on the views in $\mathcal{V}$
$\lvert\cdot\rvert$	the size or the number of elements in a given set
$\lVert\cdot\rVert$	the sum of the size of the elements in a given collection

The paper is organized as follows. Section II presents the formal settings and definitions. The problems and related results are introduced in Section III. The complexity results for minimizing view side-effect are shown in Section IV. In Section V, we propose the approximations for minimizing view side-effect. A polynomial tractable case is identified with a dynamic programming algorithm in Section V.D. The related applications are elaborated in Section VI and the related works are discussed in Section VII. Section VIII concludes the paper.

II. Formal Statements and Preliminaries

In this section, we review some classical concepts and present the formal statements in our studies.

A. Schemas and Instances

Let Const be a set of potential constants, and unless otherwise stated, these constants are denoted by lower-case letters from the beginning of the alphabet such as 'a', 'b' and 'c'. A schema $\mathcal{S}$ is a finite sequence $(T_1, \ldots, T_m)$ containing distinct relations T_i, and each T_i has an arity $Dim_i > 0$. An instance D (over a relation T) is a sequence $(T_1^D, \ldots, T_m^D)$ such that each T_i^D is a finite relation of arity Dim_i over Const (*i.e.*, T_i^D is a finite subset of Const^{Dim_i}). If $t \in \mathsf{Const}^{Dim_i}$, then t is called a tuple, and it is a tuple of instance D if $t \in T_i^D$. Notationally, we regard an instance as a set of its

facts. For examples, $T(t) \in D$ means t is in T^D; $D' \subseteq D$ means D' is a sub-instance of D, that is, $T_i^{D'} \subseteq T_i^D$ for all $i = 1, \ldots, m$. For convenience, we usually ignore the superscript D in the context that instance D is given, *i.e.*, we use T_i instead of T_i^D.

B. Conjunctive Queries, Key Preserving, and Views

Let Var be a set of variables. We assume that Var and Const are disjoint sets. We denote variables by lower-case singletons from the end of the alphabet such as "x", "y" and "z".

Conjunctive Queries. We use the *datalog* style to denote a conjunctive query (CQ), that is, a CQ defined on a schema $\mathcal{S}$ could be written as

$$Q(\mathbf{y}_1, \ldots, \mathbf{y}_q) :- T_1(\mathbf{x}_1, \mathbf{y}_1, \mathbf{c}_1), \ldots, T_q(\mathbf{x}_q, \mathbf{y}_q, \mathbf{c}_q)$$

where $\mathbf{x}_i$ and $\mathbf{y}_i$ are two vectors of variables disjoint with each other. $\mathbf{c}_i$ is a tuple of constants (from Const). A CQ is actually a conjunction of atomic formulas ("*atom*" for simplicity) over $\mathcal{S}$. Here, $Q(\mathbf{y}_1, \ldots, \mathbf{y}_q)$ is the head of query Q, and the other part is called the body of Q. For any i, every x in $\mathbf{x}_i$ is called an existential variable, and every y in $\mathbf{y}_i$ is called a head variable. We use $\mathsf{Var}_\exists(Q)$ and $\mathsf{Var}_h(Q)$ to denote the sets of existential and head variables of Q. In this paper, each $\mathbf{y}_i$ should not be empty. The width or arity of query Q, denoted as *arity*(Q), is the sum of the lengths of tuples $\mathbf{y}_i$.

For example, consider the following conjunctive queries

$$Q_1(y_1, y_2, w) :- T_1(x, y_1, z), T_2(x, y_2, w)$$

$$Q_2(y, y_1, y, y_2, y, y_3) :- T_1(\underline{y, y_1}), T_2(\underline{y, y_2}), T_3(\underline{y, y_3})$$

Query Q_1 has a width *arity*(Q_1) of 3 and has two atoms $T_1(x, y_1, z)$ and $T_2(x, y_2, w)$, while query Q_2 has a width *arity*(Q_2) of 6 and has three atoms $T_1(y, y_1)$, $T_2(y, y_2)$, and $T_3(y, y_3)$. There are two existential variables in Q_1, namely x and w, and its three head variables are y_1, y_2 and y_3. There is no existential variable in Q_2, and the three head variables are the same as those of Q_1.

For convenience, every time we refer to a query Q, the underlying schema $\mathcal{S}$ will usually not be mentioned. We assume that this schema is a default one that consists of the relation symbols appearing in Q (and each symbol has the arity it takes in that Q). We mainly investigate *key preserving* conjunctive queries introduced below.

Key-preserving. In any atom T, there is at least one key attribute position, and a key specifies a set of key attribute positions of T. A key on T states that no two tuples in T have the same values in all positions in key. Intuitively, any two tuples are different in any table having a key. Any variable located at the key attribute position is called a key variable. For convenience, we underline all the key variables if necessary, like $T_1(x, \underline{y}, c)$. Formally, any CQ Q is a key preserving conjunctive query if (a) every atom of Q has a key, and (b) all of the key variables in the keys are included in the head of Q. Consider query Q_1 again.

$$Q_1(\underline{y_1, y_2}, w) :- T_1(x, \underline{y_1}, z), T_2(x, \underline{y_2}, w)$$

All the key variables (y_1 and y_2) are included in $\{y_1, y_2, w\}$ which is the head of Q_1.

Note that if a query is key preserved, then there must be no key variables contained in $\mathsf{Var}_\exists(Q)$ (the set of existential variables). Obviously, a *project-free* conjunctive query is always key preserved, like Q_2.

View. To formally define views, we begin with an assignment of a query. An assignment for Q is a mapping from $\mathsf{Var}(Q)$ to Const. For an assignment μ on Q, $\mu(\mathbf{y})$ is a tuple created by substituting every head variable $\mathbf{y}$ with constant $\mu(y)$. For an assignment on atom, tuple $\mu(T)$ is obtained by substituting every variable x with constant $\mu(x)$. Given an instance D, a match for Q of D is an assignment μ for Q where $\mu(T)$ is a tuple from D for all atoms T. Every match is also called an answer of query Q. If μ is a match for Q in D, then $\mu(y)$ is called an answer for Q. The set of all the answers for Q in D is called query result $Q(D)$ which is just the result of evaluating Q over D.

A *view* V is a materialized query result $Q(D)$. Each individual query answer t in a view V is called a view tuple of V. The width of a view $Q(D)$ is also the width of a view tuple in it, which equals to *arity*(Q).

C. Deletion Propagation

The core of deletion propagation is the side-effect analysis. Therefore, we focus on (but not restricted to) the problem of minimizing view side-effect [6], [30]–[32] when propagating the deletion of multiple answers on multiple queries back to the source relations.

In the view side-effect minimizing problem for multiple conjunctive queries, the input inludes a source database instance D, a set of queries $\mathcal{Q} = \{Q_1, \ldots, Q_m\}$, a set of views $\mathcal{V} = \{V_1, \ldots, V_m\}$ where $V_i = Q_i(D)$, and the deletion $\Delta\mathcal{V} = \{\Delta V_1, \ldots, \Delta V_m\}$ specified on the views in $\mathcal{V}$. The output is a set of tuples ΔD minimizing the *view side-effect*

$$s_{view} = \sum_{i=1}^{n} s_i$$

such that for each $Q_i \in \mathcal{Q}$,

(*a*). $Q_i(D \setminus \Delta D) \subseteq V_i \setminus \Delta V_i$, and
(*b*). $s_i = |V_i \setminus \Delta V_i| - |Q_i(D \setminus \Delta D)|$.

Consider the example in [15], where we are given two relations $T_1(\underline{\textit{AuName}, \textit{Journal}})$ and $T_2(\underline{\textit{Journal}, \textit{Topic}}, \#\textit{Papers})$. An instance D on the two relations includes seven tuples as shown in Fig.1.

Let ΔV be (John, XML), then various tuple deletions in D could be found to remove ΔV from view $Q_3(D)$. Obviously, each of tuples (John, TKDE), (John, TODS), (TKDE, XML, 30) and (TODS, XML, 30) is with matching values in ΔV. In order to delete ΔV, one can check that removing (John, TKDE) and (John, TODS) from *Author*, or removing (John,

AuName	*Journal*
Joe	TKDE
John	TKDE
Tom	TKDE
John	TODS

(a) $T_1(\underline{AuName}, \underline{Journal})$

Journal	*Topic*	*#Papers*
TKDE	XML	30
TKDE	CUBE	30
TODS	XML	30

(b) $T_2(\underline{Journal}, \underline{Topic}, \#Papers)$

AuName	*Topic*
Joe	CUBE
Joe	XML
Tom	CUBE
Tom	XML
John	CUBE
John	**XML**

(c) $Q_3(x,z) :- T_1(\underline{x,y}), T_2(\underline{y,z}, w)$

AuName	*Journal*	*Topic*
Joe	TKDE	CUBE
Joe	TKDE	XML
Tom	TKDE	CUBE
Tom	TKDE	XML
John	TKDE	CUBE
John	**TKDE**	**XML**
John	TODS	XML

(d) $Q_4(\underline{x,y,z}) :- T_1(\underline{x,y}), T_2(\underline{y,z}, w)$

Fig. 1. An example for key-preserving CQ and view propagation.

TKDE) from *Author* and (TODS, XML, 30) from *Journal* both result in the minimum view side-effect, *i.e.*, one additional view tuple is forced to be deleted.

Consider another deletion ΔV = (John, TKDE, XML) from $Q_4(D)$. Note that deleting either (John, TKDE) from *Author* or (TKDE, XML, 30) from *Journal* works due to the key preserving property of query Q_4. Checking the view side-effect can be easily performed by finding the occurrences of key values of the deleted relation tuples in the view, and this is the important property we utilize in our work.

This example only shows the single query case, while this paper studies the case of multiple queries. In many practical data cleaning or provenancing applications, the multiple query case is more general and meaningful. It is obvious that the complexity of the multiple query case is higher than or at least equal to that of the single query case. However, the exact complexity class of the multiple query case is still not known. We are going to offer a characterization based on the key-preserving property to figure out the lower and upper bounds of the multiple query case.

D. Preliminaries

We now introduce some preliminary knowledge.

The Red-Blue Set Cover Problem [8]. Given two disjoint finite sets, one as a set of red elements $R = \{r_1, \ldots, r_\rho\}$ and another one as set of blue elements $B = \{b_1, \ldots, b_\beta\}$, and a collection $\mathcal{C} \subseteq 2^{R \cup B}$ of subsets of $R \cup B$, the Red-Blue Set Cover Problem is to find a collection $\mathcal{C}' = \{C_{i_1}, \ldots, C_{i_m}\} \subseteq \mathcal{C}$ such that all the blue elements are covered, while the number of the covered red elements is minimized. Formally, let $Cost(R, B, \mathcal{C}')$ be the total number of the red elements contained in the sets in $\mathcal{C}'$. The objective is to minimize cost

$$Cost(R, B, \mathcal{C}') = \{\left|R \cap (\cup_j C_{i_j})\right| \, | B \subseteq \cup_j C_{i_j}\}.$$

Let $|\mathcal{C}|$ be the size of $\mathcal{C}$. By a reduction from MMSA$_3$ [8], Carr *et al.* shows that the Red-Blue Set Cover problem cannot be approximated within $O(2^{\log^{1-\delta}|\mathcal{C}|})$ where $\delta = 1/\log\log^c|\mathcal{C}|$ for any $c < 0.5$, unless P=NP. We show a linear reduction from it in the next section to the deletion propagation problem for multiple queries so as to obtain the lower bound.

The Positive-Negative Partial Set Cover Problem [38]. The input is similar with that of the Red-Blue Set Cover problem. Let a set of positive elements $P = \{p_1, \ldots, p_\rho\}$ and a set of negative elements $N = \{n_1, \ldots, n_\beta\}$ be two disjoint finite sets, and let $\mathcal{C} \subseteq 2^{P \cup N}$ be a collection of subsets of $P \cup N$. In Positive-Negative Partial Set Cover, instead of covering all the blue elements, the requirement is relaxed so that it aims to find the best trade-off between covering the blue elements and not covering too many red ones. In the context of such Positive-Negative Partial Set Cover, the red and blue elements are called negative and positive elements, respectively. Formally, given an input instance which is a triplet $(N, P, \mathcal{C})$, a solution is a collection $\mathcal{C}' = \{C_{i_1}, \ldots, C_{i_m}\} \subseteq \mathcal{C}$, and its cost is defined as

$$Cost(R, B, \mathcal{C}') = \left|P \setminus (\cup_j C_{i_j})\right| + \left|N \cap (\cup_j C_{i_j})\right|,$$

namely the number of uncovered positive elements plus the number of covered negative elements. The goal is to minimize $Cost(R, B, \mathcal{C}')$.

A linear reduction has been shown from the Red-Blue Set Cover to this problem by Miettinen [38], thus passing the lower bound $O(2^{\log^{1-\delta}|\mathcal{C}|})$ where $\delta = 1/\log\log^c|\mathcal{C}|$ for any $c < 0.5$.

Prime-Dual Approximation. As a widely applied approximation technique, Vazirani provided a detail introduction and talk. The primal-dual method uses a linear programming relaxation of an integer linear programming formulation of the problem being approximated and the dual linear programming of this formulation. The algorithm starts from zero solutions, which are infeasible for the primal program and feasible for the dual one. The algorithm successively changes these solutions, such that the dual remains feasible, while the primal eventually becomes feasible, and the α-relaxed primal and the β-relaxed dual complementary slackness conditions hold. Then, such a pair of feasible solutions are $\alpha\beta$-approximations of their respective linear programming ones. [44]. Sometimes, the algorithm prunes the feasible primal and dual solutions to ensure that the relaxed complementary slackness conditions hold.

III. Results on Complexities

Recalling the results for the single query case, it is polynomial-tractable for a key-preserving conjunctive query. However, for multiple queries, the problem becomes extremely hard. The first result we obtained is negative even for *project-free* conjunctive queries, *i.e.*, select-join queries. We show that it is extremely hard to approximate even for multiple key-preserving conjunctive queries.

Theorem 1. *Even for two project-free conjunctive queries, the view side-effect problem cannot be approximated within* $O(2^{\log^{1-\delta}\|\mathcal{V}\|})$ *where* $\delta = 1/\log\log^c\|\mathcal{V}\|$ *for any* $c < 0.5$*, unless P=NP.*

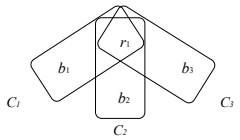
Proof: We here provide the proof sketch.

Schema. Given any instance of Red-Blue Set Cover $(R, B, \mathcal{C})$, we build a schema containing only one relation denoted by $T(\mathbf{x})$ in which $\mathbf{x}$ is an $|R \cup B|$-dimensional vector of variables.

Database instance. We add $|\mathcal{C}|$ tuples into T where each tuple $t \in T$ refers to a set $C \in \mathcal{C}$. For the value invention of each tuple corresponding to $C \in \mathcal{C}$, for each element $b_i \in B$, let the i-th value of t be b_i if $b_i \in C$. For the rest cell of tuple t, fill them by distinct values in T. Finally, a table instance with $|\mathcal{C}|$ tuples of $|R \cup B|$ dimensions is built. It is actually a bijection between T and $\mathcal{C}$.

View. We define a view for each element in $R \cup B$. In fact, all the views are defined by a set of *project-free* conjunctive queries $\{Q_{r_1}, \ldots, Q_{r_\rho}, Q_{b_1}, \ldots, Q_{b_\beta}\}$. We use each query to generate a view corresponding to each element. Concretely, each query Q_{r_i} is to build view V_{r_i} as a join path for each red element $r_i \in R$ and to build view V_{b_i} as a join path for each blue element $b_i \in B$.

View Deletion. Let view deletion $\Delta\mathcal{V}$ be the set of views $\{V_{b_i}\}$ corresponding to all the blue elements in B.



$\rightarrow$ T:

r_1	b_1
r_1	b_2
r_1	b_3

$\mathcal{V}$

$\surd$	$V_{r_1} :- Q_{r_1}(r_1, b_1, r_1, b_2, r_1, b_3)$	
$\times$	$V_{b_1} :- Q_{b_1}(r_1, b_1)$	
$\times$	$V_{b_2} :- Q_{b_2}(r_1, b_2)$	$\Delta\mathcal{V}$
$\times$	$V_{b_3} :- Q_{b_3}(r_1, b_3)$	

Fig. 2. Example $\mathcal{C} : \{C_1(r_1, b_1), C_2(r_1, b_2), C_3(r_1, b_3)\}$ is an instance of Red-Blue Set Cover. T is the corresponding table built in the reduction. The views are defined by SJ query and view deletions are the last three tuples.

One can verify that there is a solution such that covering all the blue elements while minimizing the number of covered red elements, if and only if there is a way of tuple deletion from T eliminating $\Delta\mathcal{V}$ while minimizing the damage on results of $Q_{r_1}, \ldots, Q_{r_\rho}$, *i.e.*, minimizing view side-effect. ■

It is easy to check the linearity of this reduction, so that view side-effect for multiple *project-free* conjunctive queries cannot be approximated within $O(2^{\log^{1-1/\log\log^c \|\mathcal{V}\|} \|\mathcal{V}\|})$ for any $c < 0.5$, unless P = NP.

As a further step, here we talk about a variant of the standard view side-effect problem, named *balanced deletion propagation*. For this balanced version, we consider not only removing the bad view tuples but also preserving the good view tuples. Formally, the input still includes the database instance, the views and tuple deletions on them. Let ΔD be a solution of tuple deletions in source data, then the view tuples deleted from view V_i by ΔD should be $V_i - Q_i(D \setminus \Delta D)$. The balanced version is to trade off deleting view tuples and not deleting too many good view tuples, *i.e.*, to minimize

$$\sum_{i=1}^{|\mathcal{V}|} |V_i - Q_i(D \setminus \Delta D)| + \sum_{i=1}^{|\mathcal{V}|} |V_i \setminus \Delta V_i - Q_i(D \setminus \Delta D)|$$

where each $Q_i \in \mathcal{Q}$.

By utilizing the idea of such reduction, we can show the hardness result for the *Balanced Deletion Propagation* problem.

Theorem 2. *Unless P=NP, even for two project-free conjunctive queries, the Balanced Deletion Propagation problem cannot be approximated within* $O(2^{\log^{1-\delta} \|\mathcal{V}\|})$ *where* $\delta = 1/\log\log^c \|\mathcal{V}\|$ *for any* $c < 0.5$.

Proof: Here we reduce Positive-Negative Partial Set Cover to Balanced Deletion Propagation. We still build a table corresponding to the positive and negative sets according to the way of the proof of Theorem 1. Define a view for each element in $N \cup P$. All the views are defined by a set of *project-free* conjunctive queries $\{Q_{p_1}, \ldots, Q_{p_\rho}, Q_{n_1}, \ldots, Q_{n_\beta}\}$. Each query is to generate a view corresponding to an individual element. Concretely, each query Q_{p_i} is to build view V_{p_i} as a join path for each positive element $p_i \in P$ and to build view V_{n_i} as a join path for each blue element $n_i \in N$. At last, let view deletion $\Delta\mathcal{V}$ be the set of views $\{V_{n_i}\}$ corresponding to all the blue elements of N. One can verify that there is a solution such that covering all the blue elements while minimizing the number of covered red elements, if and only if there is a way of tuple deletion from T eliminating $\Delta\mathcal{V}$ while minimizing damage on the results of $Q_{r_1}, \ldots, Q_{r_\rho}$, *i.e.*, minimizing view side-effect. ■

This result again implies balanced deletion propagation for multiple *project-free* conjunctive queries cannot be approximated within $O(2^{\log^{1-\delta} \|\mathcal{V}\|})$ where $\delta = 1/\log\log^c \|\mathcal{V}\|$ for any $c < 0.5$, unless P = NP. Concretely, we employ the second corollary in [38]. In fact, it is shown that unless NP$\subseteq$DTIME$(n^{\text{polylog}(n)})$, it is impossible to approximate Positive-Negative Partial Set Cover within $O(2^{\log^{1-\delta} |\mathcal{C}|})$, for any $\delta > 0$. The first theorem given by Miettinen [38] could be applied here to obtain a similar result on the hardness of Red-Blue Set Cover. Combining Theorem 3.1 in [8] and Theorem 1 in [38], we can derive the following result on the lower bound of the standard and balanced versions of the view side-effect problem.

Unless P = NP, it is impossible to approximate Positive-Negative Partial Set Cover within $O(2^{\log^{1-\delta} n})$ where $\delta = 1/\log\log^c n$ for any $c < 0.5$.

This statement and the second corollary in [38] imply a similar approximation ratio for the balanced version, because through the reduction from positive-negative partial set cover to balanced deletion propagation, an approximation of balanced deletion propagation within $O(2^{\log^{1-\delta} \|\mathcal{V}\|})$ follows, due to the approximation within $O(2^{\log^{1-\delta} 2|\mathcal{C}|})$ of positive-negative partial set cover.

The second corollary in [38] states the inapproximability of positive-negative partial set cover within $O(2^{\log^{1-\delta}|B|})$ for any $\delta > 0$, unless P = NP.

Therefore, the linear reduction from positive-negative partial set cover implies the inapproximability of balanced deletion propagation within $O(2^{\log^{1-\delta} \|\Delta\mathcal{V}\|})$.

IV. Approximation Algorithms

In this section, we propose the approximate algorithms for the view side-effect problem. To deal with more practical scenario, we provide an algorithm solving the weighted version, where each view tuple to be preserved has a weight representing user preference.

A. Approximation for the General Case

Our first result is shown by the following claim. Let l be the maximum width of the given key-preserving conjunctive queries (views), namely

$$l = \max_{Q \in \mathcal{Q}} \textit{arity}(Q)$$

, then we have the following result.

Claim 1. *Let $\mathcal{V}$ be the set of given views and $\Delta\mathcal{V}$ be the set of intended tuple deletions. View side-effect can be approximated within $O(2\sqrt{l \cdot \|\mathcal{V}\| \cdot \log \|\Delta\mathcal{V}\|})$.*

This can be done by reducing view side-effect to Red-Blue Set Cover. (*a*) Generate a red element for each view tuple to be preserved, (*b*) Generate a blue element for each view tuple to be deleted, and (*c*) Generate a set for each tuple t such that this set contains exactly the view tuples containing the tuple in it, *i.e.*, t is on the corresponding join paths simultaneously. Due to the *key-preserving* property, it is always feasible to find all the corresponding tuples. The weights of the view tuples are transferred as they are. It is easy to check that a solution to the obtained red-blue set cover instance is mapped to a solution for the corresponding instance of the view side-effect problem in this reverse manner. This reduction preserves the feasibility and the cost of a solution, thus passing the upper bound from red-blue set cover to view side-effect. We employ the algorithm *LowDegTwo* proposed by Peleg [41] to solve the red-blue set cover instance obtained by the reverse reduction. Then transform the solution cover back to a set of the corresponding tuples, *i.e.*, deletion on the source data. It is easy to check this mapping from view side-effect to Red-Blue Set Cover preserves the approximation ratio. Since the algorithm for Red-Blue Set Cover achieves a ratio of $2\sqrt{|\mathcal{C}| \log |B|}$, the transferred ratio should be $O(2\sqrt{l \cdot \|\mathcal{V}\| \cdot \log \|\Delta\mathcal{V}\|})$. The claim follows since each tuple involved in the views defines a set in the reduction of view side-effect to red-blue set cover.

For the balanced view side-effect problem, we show that a feasible solution of a ratio guarantee could be found as follows.

Lemma 1. *Balanced deletion propagation could be approximated within $2\sqrt{l \cdot (\|\mathcal{V}\| + \|\Delta\mathcal{V}\|) \cdot \log \|\Delta\mathcal{V}\|}$.*

For any instance of balanced deletion propagation, one can reduce it to positive-negative partial set cover. As any instance of positive-negative partial set cover can be approximated within $2\sqrt{(|\mathcal{C}| + |B|) \log |B|}$, and obtain a solution for balanced deletion propagation by transforming the approximate solution back to a set of corresponding tuples. This preserves the approximation ratio, as claimed above, yielding the approximation ratio of $2\sqrt{l \cdot (\|\mathcal{V}\| + \|\Delta\mathcal{V}\|) \cdot \log \|\Delta\mathcal{V}\|}$.
Next, we present a characterization to distinguish better cases.

B. Characterization by Dual Graph

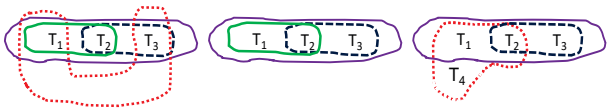
Now, we address a class of *sj-free* key preserving conjunctive queries. According to the dichotomy characterized by Kimelfeld *et al.* [30], the case of a conjunctive query without *head-domination* is polynomial intractable even for the *sj-free* ones. However, an *sj-free* conjunctive query of key-preserving property does not imply head-dominations, *e.g.*, *sj-free* key-preserving conjunctive query $Q(\underline{y_1}, \underline{y_2}) :- T_1(\underline{y_1}, x), T(x, \underline{y_2})$ is *sj-free* key-preserving but not of head-domination.

Dual Hypergraph. Given schema $\mathcal{S} = \{T_1, \ldots, T_m\}$, let $\mathcal{Q}$ be a set of *sj-free* key preserving conjunctive queries $\{Q_1, \ldots, Q_m\}$ where each individual query has form

$$Q_i :- T_{i_1}, \ldots, T_{i_{q_i}}.$$

Its dual hypergraph $H(\mathcal{Q})$ has vertex set $\{T_1, \ldots, T_m\}$. Each query Q_i determines the hyperedge consisting of all those relations in it: $e_i = \{T_{i_j} | 1 \le j \le q_i\}$.

For all the given *sj-free* key preserving conjunctive queries in $\mathcal{Q}$, a *dual graph* could be obtained in the following way. If every connected component is a hypertree [23], then the input is a *forest* case.



(a) Hypergraph for $\mathcal{Q}_1$ (b) Hypertree for $\mathcal{Q}_2$ (c) Hypertree for $\mathcal{Q}_3$

Fig. 3. A dual hypergraph and a dual hypertree.

For example, given the following five queries

$$Q_1 :- T_1, T_2, T_3, \quad Q_2 :- T_1, T_2, T_4$$
$$Q_3 :- T_1, T_2, \quad Q_4 :- T_1, T_3, \quad Q_5 :- T_2, T_3$$

and three query sets $\mathcal{Q}_1 = \{Q_1, Q_3, Q_4, Q_5\}$, $\mathcal{Q}_2 = \{Q_1, Q_3, Q_5\}$, and $\mathcal{Q}_3 = \{Q_1, Q_2, Q_5\}$ as shown in Fig.3, the dual hypergraphs of $\mathcal{Q}_2$ and $\mathcal{Q}_3$ are both hypertrees, but the dual hypergraph of $\mathcal{Q}_1$ is not.

We provide two approximations for the forest cases where each component is a hypertree.

C. l-Approximation for the Forest Cases

Having shown the hardness, we design two approximation algorithms for (weighted) view side-effect. Similar results will be shown for the balanced version. The first algorithm makes use of the primal-dual technique for view side-effect on trees. This algorithm is inspired by a related primal-dual-algorithm for multicut on trees [25].

We are given the linear relaxation of the integer linear programming formulation of the view side-effect of the tree case as follows. Let $\mathcal{R}$ be the set of tuples to be preserved $\{V_1 \setminus \Delta V_1, \ldots\}$, then we have

$$\textit{minimize} \sum_{r \in \mathcal{R}} w_r x_r \quad (1)$$

s. t.

$$\forall r \in \mathcal{R} \qquad k_r x_r - \sum_{t \in r} y_t \geq 0 \quad (2)$$

$$\forall r \in \Delta\mathcal{V} \qquad k_r x_r - \sum_{t \in r} y_t \geq 1 \quad (3)$$

$$\forall t \in D \qquad y_t \geq 0 \quad (4)$$

$$\forall r \in \mathcal{R} \qquad x_r \geq 0 \quad (5)$$

Here, y_t is the variable indicating the deletion of tuple t from source database D, and x_r is the variable indicating the accident elimination of view tuple r which is supposed to be preserved. Concretely, delete tuple t if $y_t = 1$ and view tuple r is eliminated by accident if $x_r = 1$. Formula (3) indicates that every view tuple (*i.e.*, *project-free* conjunctive query result tuple) to be deleted is removed by deleting at least one of the tuples joined by its query, and Formula (2) indicates that once a view tuple to be preserved loses any tuple joined in, it will be removed as well. Our objective is to minimize the total weight of the view tuples to be preserved but actually removed by accident.

The following LP is the dual to the LP formulated above.

$$\textit{maximize} \sum_{r \in \Delta\mathcal{V}} v_r \quad (6)$$

s. t.

$$\forall r \in \mathcal{R} \qquad k_r v_r \leq w_r \quad (7)$$

$$\forall t \in D \qquad \sum_{r \in \Delta\mathcal{V} \wedge t \in r} v_r - \sum_{s \in \mathcal{P} \wedge t \in s} v_s \leq 0 \quad (8)$$

$$\forall r \in \mathcal{R} \qquad v_r \geq 0 \quad (9)$$

$$\forall r \in \Delta\mathcal{V} \qquad v_r \geq 0 \quad (10)$$

The interpretation of an optimal integer solution to the dual LP is to maximize the total number of view tuples to be deletet such that the weight constraints hold for the view tuples to be preserved that dominate the view tuples to be deleted involving each tuple. We have the primal complementary slackness conditions in our formulation as follows.

$$\forall t \in D, \qquad y_t > 0 \Rightarrow \sum_{r \in \Delta\mathcal{V} \wedge t \in r} v_r = \sum_{s \in \mathcal{P} \wedge t \in s} v_s \quad (11)$$

$$\forall r \in \mathcal{R}, \qquad x_r > 0 \Rightarrow k_r v_r = w_r \quad (12)$$

We have the relaxed dual complementary slackness conditions as follows.

$$\forall t \in D, \qquad y_t > 0 \Rightarrow \sum_{r \in \Delta\mathcal{V} \wedge t \in r} v_r = \sum_{s \in \mathcal{P} \wedge t \in s} v_s \quad (13)$$

$$\forall r \in \mathcal{R}, \qquad x_r > 0 \Rightarrow k_r v_r = w_r \quad (14)$$

For the tree case, we define the depth of a tuple as the length of the path from this tuple to the root tuple at the beginning of the tree like join result. Define the lowest common ancestor of two tuples t_1 and t_2 as the tuple of the path from one to the other with the minimum depth. The lowest common ancestor is unique since in a tree, the path can be found by looking at the first tuple of the path from t_1 to the root tuple that is also on the path from the root tuple to t_2. Therefore, each previous tuple on the part of the path from t_1 to this tuple is less deep, while every next tuple on the following part of the path back to t_2 is deeper. Algorithm 1 is then provided based on the primal-dual technique.

Algorithm 1 PrimeDualVSE$(D, \mathcal{V}, \Delta\mathcal{V}, \mathcal{R}, \omega)$

1: Mark the infeasible primal and dual zero solutions.
2: Pick any table as the beginning such that the root of trees is tuples contained in this table.
3: **for** $\forall t \in D$ in increasing depth with respect to roots **do**
4: (a) For each view tuple r to be deleted such that r is a join result $(t, ..., t')$ with $lca(t, t') = v$: increase v_r as much as possible (not necessarily integrally), with the necessary increase of the intersecting view tuples to be preserved, to preserve the feasibility of the dual linear programming.
5: (b) All the saturated (in the sense of Constraint (8) holds with equality and for every view tuple to be preserved, Constraint (7) holds with equality) tuples are chosen to be deleted ($y_t = 1$), and those x_r are set as 1 in case to support Constraint (2).
6: **end for**
7: **for** each chosen tuple t ($y_t = 1$) in the reverse procedure of the addition, if deleting it is not necessary for fulfilling constraint (3) **do**
8: (a) $y_t \leftarrow 0$.
9: (b) $x_r \leftarrow 0$ if any x_r can be set as 0 while obeying Constraint (2).
10: **end for**

The correctness of Algorithm **PrimeDualVSE** could be guaranteed as follows.

Theorem 3. *Algorithm **PrimeDualVSE** returns a feasible l-approximation for view side-effect.*

Proof: First, we prove that the returned solution, defined by $\{y_t | t \in D, y_t = 1\}$, is feasible. Because we take all the saturated tuples in the maximum view tuples to be deleted, and this means all those view tuples of $\Delta\mathcal{V}$ have been eliminated. This is correct, since otherwise, we could have a view tuple of $\Delta\mathcal{V}$ where no tuple is saturated, contradictory to the maximality of that view tuple. The returned primal solution is feasible because those x_r's are picked to make the primal solution feasible, and the pruning step prunes only if the primal solution remains feasible. The dual solution is also feasible because the view tuples are routed in a manner that preserves feasibility. In order to assert the approximation ratio, we prove that the primal and the relaxed dual complementary slackness holds. The primal conditions hold because the taken edges are saturated. The relaxed dual condition (13) holds

because x_r is positive only if at least one of the tuple r is selected. Therefore, if the left hand side of Condition (13) is positive, then $\sum_{t \in r} y_t \geq 1$, and the required inequality follows. Condition (14) follows because of the argument given in the proof of Lemma 18.5 in [44]. Having primal and dual feasible solutions that satisfy the exact primal and the l-relaxed dual complementary slackness conditions implies that the primal solution is an l-approximation to the optimum. ■

Proposition 1. *Let n be the size of the input database instance, Algorithm 1 terminates in $O(l \cdot \|\Delta\mathcal{V}\|^2 \cdot \|\mathcal{V}\| + \|\mathcal{V}\|^4)$ time.*

D. $2\sqrt{\|\mathcal{V}\|}$-Approximation for the Forest Cases

We now propose another algorithm for view side-effect on trees that approximates within $2\sqrt{\|\mathcal{V}\|}$, which is sometimes better than factor l of the last algorithm. We refine the algorithm LowDegTwo for Red-Blue Set Cover [8], using the l-approximation to view side-effect problem. Recall that l is defined as the maximum width, namely $\max_{Q \in \mathcal{Q}}$ *arity*(Q), among all queries. We use the algorithm for standard view side-effect since the extension for the weighted version is straightforward. We first describe Algorithm **LowDegTreeVSE** inspired by **LowDegTwo** [8].

Algorithm 2 LowDegTreeVSE$(D, \mathcal{V}, \Delta\mathcal{V}, \mathcal{R}, \omega, \tau)$

1: Remove the tuples t of D joined in more than τ view tuples to be preserved (*i.e.* from $\mathcal{R}$).
2: Let the renewed instance be $(D', \mathcal{R}', \mathcal{V}', \Delta\mathcal{V}', \omega, \tau)$, where D' is the instance after tuple deletion and so are the other input parameters.
3: **if** the new instance is infeasible **then**
4: **return** D.
5: **end if**
6: For prune, find out view tuples to be preserved such that $\mathcal{R}'_> = \{r \in \mathcal{R}', \text{ where } arity(r) > \sqrt{\|\mathcal{V}\|}\}$.
7: Prune those wide view tuples such that $\mathcal{R}'' = \mathcal{R}' \setminus \mathcal{R}'_>$.
8: **return** **PrimeDualVSE**$(D', \mathcal{V}', \Delta\mathcal{V}', \mathcal{R}'', \omega, \tau)$.

The following claim follows.

Claim 2. *Let τ be the last input to Algorithm **LowDegTreeVSE**, we have $|\mathcal{R}'_>| < \sqrt{\|\mathcal{V}\|} \cdot \tau$.*

Proof: Since every tuple of D is joined in at most τ view tuples to be preserved, we have

$$|\mathcal{R}'_>| \sqrt{\|\mathcal{V}\|} \quad < \quad \sum_{r \in \mathcal{R}'_>} |\{t | t \in r\}| \quad \leq \quad \sum_{r \in \mathcal{R}'} |\{t | t \in r\}|$$

so that

$$|\mathcal{R}'_>| \cdot \sqrt{\|\mathcal{V}\|} \quad < \quad \|\mathcal{V}'\| \cdot \tau \quad \leq \quad \|\mathcal{V}\| \cdot \tau.$$

The first inequality is from the definition of $\mathcal{R}'_>$, and the equality is a reversal of the summation order. Therefore,

$$|\mathcal{R}'_>| \quad < \quad \frac{\|\mathcal{V}\| \cdot \tau}{\sqrt{\|\mathcal{V}\|}} = \sqrt{\|\mathcal{V}\|} \cdot \tau.$$

■

We can now prove the following claim.

Claim 3. *Let OPT* be an optimal solution for an input instance of view side-effect. If input $\hat{\tau}$ is $\max |\{r \in \mathcal{R} | \, r \text{ contains } t, \text{ and } t \in OPT^*\}|$ and apply Algorithm **LowDegTreeVSE**, then it returns a $2\sqrt{\|\mathcal{V}\|}$-approximation.*

Proof: Due to the value of τ, the algorithm will return a feasible solution at last. An l-approximation could be guaranteed which has been shown in advance. The input of **PrimeDualVSE** has $l \leq \sqrt{\|\mathcal{V}\|}$, thus its solution, say α, gives

$$|\mathcal{R}'(\alpha)| \leq \sqrt{\|\mathcal{V}\|} \, |\mathcal{R}'(\alpha^*)| \, .$$

Apply $|\mathcal{R}'_>| < \sqrt{\|\mathcal{V}\|}\hat{\tau}$ in the last claim so that the total number of the view tuples removed by accident is

$$|\mathcal{R}(\alpha)| < \sqrt{\|\mathcal{V}\|} \, |\mathcal{R}'(OPT^*)| + \sqrt{\|\mathcal{V}\|}\hat{\tau}.$$

Since $\hat{\tau} \leq |\mathcal{R}(OPT^*)|$, we can bound $|\mathcal{R}(\alpha)|$ by $2\sqrt{\|\mathcal{V}\|} \, |\mathcal{R}(OPT^*)|$. ■

The algorithm for approximating the actual problem can now be shown as follows where $\hat{\tau}$ is unknown in advance.

Algorithm 3 LowDegTreeVSETwo$(D, \mathcal{V}, \Delta\mathcal{V}, \mathcal{R}, \omega)$

1: Let *OPT* be D
2: **for** $\tau = 1$ to $|\mathcal{R}|$ **do**
3: Let α be LowDegTreeVSE$(D, \mathcal{V}, \Delta\mathcal{V}, \mathcal{R}, \omega, \tau)$
4: **if** $\omega(\mathcal{R}(\alpha)) < \omega(\mathcal{R}(OPT))$ **then**
5: Let OPT be α
6: **end if**
7: **end for**
8: **return** OPT

The following theorem is then derived from the last claim.

Theorem 4. *Algorithm **LowDegTreeVSETwo** approximates the solution to view side-effect problem within $2\sqrt{\|\mathcal{V}\|}$.*

This has been proven for the non-weighted case, and it is straight-forward to extend the result to the weighted case.

We next provide a dynamic programming for a more restricted tree case.

E. Forest Cases with Pivot Tuple

We design a dynamic programming for a class of forest cases. To illustrate this class, we introduce an additional structure of *data dual graph*.

Data dual graph. Let $H(\mathcal{Q})$ be the hypergraph of $\mathcal{Q}$, and it is an hypertree. Then for each tuple r in query result $Q(D)$, we take the projection $r[T]$ on any involved relation T, say t, as a vertex. In this way, take each tuple t as a path in the order of the layout of $H(\mathcal{Q})$.

Forest dual data graph with pivot tuple. In a graph obtained in this way (we call it dual data graph), for each connected component, there exist some fixed relation T and a tuple $t \in T$, called *pivot tuple*, such that every view tuple is on a path from t to all the other tuples.

This assumption allows us to solve both view side-effect and its balanced version exactly using dynamic programming. We define the recursion to be a subtree and the view tuples strictly joined through its root after possible tuple deletions outside this subtree.

Let $t \in T$ be a vertex and r be the set of tuples on the (only) path from t to the root tuple of the tree. Let $\mathcal{T}(t) := \{r \in \mathcal{V} | \ r \text{ contains } t\}$ be the set of the originally given view tuples that pass through t. Denote by $S(t)$ the subtree rooted on t. The possible subsets of view tuples that enter $\mathcal{T}(t)$ after deleting some tuple outside of $\mathcal{T}(t)$ are $\mathcal{T}(t) = \{\mathcal{T}(t) \setminus \mathcal{T}(\{t\}) | t \in r_t\}$. We do not consider deleting a subset of tuples on r, because for $\mathcal{T}(t)$ it would be equivalent to deleting the tuple of this subset closest to t. The dynamic programming solves view side-effect exactly.

Algorithm 4 DPTreeVSE$(D, \mathcal{V}, \Delta\mathcal{V}, \mathcal{R}, \omega)$

1: The algorithm maintains the status transition array indexed by $D \times \mathcal{T}(t)$.
2: **for** each tuple $t \in D \setminus t_{root}$ in a post-order traversal **do**
3: **for** each $r \in \mathcal{T}(t)$ **do**
4: Delete the tuple from t to its parent if and only if t maximizes the total objective function in $S(t)$.
5: Memoize the resulting tuple deletion and the resulting objective function for the current entry $(t, r) \in D \times \mathcal{T}(t)$.
6: **end for**
7: **end for**
8: The completed status transition array contains an optimal set of tuple deletions.

It is a polynomial algorithm since its status transition array is of a poly size. Indeed, $\mathcal{T}(t)$ contains $|r|$ elements, as defined and explained above.

V. Related Applications

Besides the connection with *why-provenance*, *where-provenance* and query reverse engineering [15], we introduce some more practical applications of deletion propagation, especially from the view side-effect aspects.

Data annotation. Recall ΔV_1 = (John, XML) in Example 1. It is already known that there is an error (John, XML) in view $Q_1(D)$ since researcher John does not do any research on XML. As stated in [15], data annotation is an important feature of new generation database. Therefore, one may want to look for tuples ΔD from the source data D in order to annotate the error such that the corresponding annotations propagate to the fields of view tuples in ΔV via Q_1. However, since errors in views are essentially produced by the errors in source data, then usually if there is an error found in one view, it is almost sure that some errors appear in other views. By making use of propagating annotations on the results of multiple queries, the candidate tuples in the source data can be found more accurately. Concretely, we can observe from Example 1 that there are usually multiple optimal solutions, some of which may not be the best choice, but the results could be shrunk by merging deletions specified on the results of multiple queries, and the candidate will be found more accurately. Ideally, if the views and view deletions are given completely, we can always find the view side-effect free solutions. Intuitively, the more queries and its views, the closer we approach the side-effect free solution. This is why we should provide a necessary tool for such a more general and practical case.

Query-oriented cleaning. Alternatively, in the context of data repairing like the query-oriented cleaning framework, one may want to delete ΔD so as to achieve a tuple-level repair of the inconsistent or inaccurate ΔV. Emerging data cleaning tools [2], [3] integrate user efforts on query result corrections. They find out inconsistent information by making use of materialized (or non-materialized) views specified by user queries. However, the latest proposed cleaning system QOCO [2] interacts with domain experts (like crowds) to identify potentially wrong answers in the query results. In their system, queries are generated to cover source tuples as many as possible. However, the feedbacks on user queries are processed one by one, not in the way of batch process due to the theoretical limit which cannot provide a guarantee on the feasibility and accuracy of the result obtaining by the batch process. Such a non-batch process introduces additional dependent factors, namely the order of processing, which potentially leads to more damage on loss on quality of data cleaning. To tackle this problem, our theoretical study provides a guarantee on the batch process so that it enables the batch process with theoretical guarantee.

Balanced version. We now introduce the motivation of the balanced version. Obviously, there are always scenarios where the unexpected damage on views caused by removing all the bad results is too huge, and such loss or cost sometimes is too expensive for users to afford. If the information on view errors is not complete (like the crowds stated above), ΔV may not be specified accurately. Therefore, in such a case, we just want to see if there is some possible solution ΔD on the views removing major part of the bad results ΔV while taking small side-effect on the other results. The balanced version is just defined for this kind of cases.

VI. Related Works

Some efforts have been spent on studying the computational complexities of *source side-effect* and *view side-effect*. The previous results regarding *source side-effect* are presented in Table II and Table III. The previous results regarding *view side-effect* are summarized in Table IV and Table V. The complexity of relational query languages was first investigated by Chandra and Merlin [9] about 40 years ago. Then, it became one of the most primary concerns in the theoretical study of the database field. Unfortunately, for query evaluation, the available complexity results of query languages are shown in a general way and seems too rough. A few years later, two new metrics are proposed in [43] to measure the complexity of query evaluation. These two new metrics are *data complexity* and *combined complexity*. By considering these two metrics,

TABLE II
POLY-TRACTABLE CASES OF THE SOURCE SIDE-EFFECT PROBLEM

Complexity	Citations	Query Class
PTime	Buneman *et al.* 2002 [6]	*project-free* & *sj-free* conjunctive queries
	Cong *et al.* 2012 [15]	*key-preserving* conjunctive queries
	Freire *et al.* 2015 [24]	*triad-free* & *sj-free* conjunctive queries
		fd-induced-triad-free & *sj-free* conjunctive queries

TABLE III
HARD CASES OF THE SOURCE SIDE-EFFECT PROBLEM

Complexity	Citations	Query Class
NP-complete	Buneman *et al.* 2002 [6]	select-free conjunctive queries
	Cong *et al.* 2012 [15]	*non-key-preserving* conjunctive queries
	Freire *et al.* 2015 [24]	queries with *triad*
		queries with *fd-induced triad*
co-W[1]-*complete*	[36]	conjunctive queries for parameter query size or #variables
		positive queries for parameter query size
co-W[SAT]-*hard*		positive queries for parameter #variables
co-W[t]-*hard*		first-order queries for parameter query size
co-W[P]-*hard*		first-order queries for parameter #variables

TABLE IV
POLYNOMIAL TRACTABLE CASES OF THE VIEW SIDE-EFFECT PROBLEM

Complexity	Citations	Query Class
PTime	Buneman *et al.* 2002 [6]	*project-free* & *sj-free* conjunctive queries
	Cong *et al.* 2012 [15]	*key-preserving* conjunctive queries
	Kimelfeld *et al.* 2012 [30]	*sj-free* conjunctive queries having *head-domination*
		sj-free conjunctive queries having *fd-head-domination*
FPT	Kimelfeld *et al.* 2013 [32]	*sj-free* conjunctive queries having *level-k head-domination*
		sj-free conjunctive queries having *head-domination*

TABLE V
HARD CASES OF THE VIEW SIDE-EFFECT PROBLEM

Complexity	Citations	Query Class
NP-complete	Buneman *et al.* 2002 [6]	*select-free* conjunctive queries
	Cong *et al.* 2012 [15]	*non-key-preserving* conjunctive queries
	Kimelfeld *et al.* 2012 [30]	*non-head-domination* conjunctive queries
		non *fd-head-domination* conjunctive queries
	Kimelfeld *et al.* 2013 [32]	non *level-k head-domination* conjunctive queries
NP(k)-complete	Miao *et al.* 2017 [36]	conjunctive queries for *bounded source deletions*
Σ_2^{P}-complete	Miao *et al.* 2016 [37]	conjunctive queries under general settings

the complexity of query evaluation can be investigated in a more reasonable way. Intuitively, *data complexity* helps with identifying the scenarios where the query size is small with respect to the database instance involving a reasonable number of join operations. Therefore, it is represented as a function of the size of the database instance and the considered query is regarded as a fixed one. Differently, *combined complexity* focuses on more general scenarios where both the considered query and database instance are not fixed. Thus, the considered query and database instance are both regarded as variables of the complexity function. The combined complexity for a specific class of queries involving join operations is usually inevitably exponential higher than the data complexity. Because of this reason, data complexity is more meaningful for practical query evaluations, thus it draws a lot of attentions.

Thanks to such two metrics, more complexity results of the view side-effect problem are obtained [6], [14], [15], [30]–[32]. For data complexity, Kimelfeld *et al.* [31] introduced a dichotomy called '*head-domination*'. It is shown that for *sj-free* conjunctive queries, view side-effect is polynomial tractable by the unidimensional algorithm for the queries with the head-domination property, and no PTAS even without the head-domination property. The functional dependency restricted version is a natural extension of view side-effect. Things change too much with the presence of functional dependency. Thus, the work in [30] introduced another extended dichotomy, called '*fd-head domination*'. Besides single deletion, they also identified a trichotomy, called '*level* k-

domination', for multi-tuple deletion in [32]. It is claimed that finding a k-factored constant-optimal solution is polynomial tractable for a query with the level k-domination property, but it is NP-hard to derive a constant optimal solution. For combined complexity, there exist many results for different fragments of monotone (*i.e.*, Select-Project-Join-Union) query [14], [15], [37]. The authors in [14], [15] presented the intractable cases and illustrated a key-preserving condition to recognize a polynomial tractable case. The authors in [37] investigated the functional dependency restricted view side-effect problem. Similarly, several polynomial tractable and intractable cases are presented for view side-effect.

Moreover, there are some results on the source side-effect problem [6], [14], [15], [24]. The work in [24] is very related to this paper. Freire *et al.* identified the '*triad*' property for dual hypergraph representation of conjunctive queries. Based on this work, we can know that the resilience decision problem is polynomial decidable if the dual hypergraph of the given query q excludes the *triad* structure. Otherwise, the problem is NP-complete. Similarly, if a set of functional dependencies are defined in advance, then the '*triad*' structure should be changed to '*fd-induced triad*' which is more general. Then, a tight dichotomy follows.

The view update problem is another related problem. It has been widely investigated for many years. The example works include [1], [4], [16], [19], [29]. The view update problem studies how to perform an update on the source data in order to remove ambiguity or achieve the expected update to a specified view while guaranteeing the semantic correctness. Most of the previous works mainly focus on recognizing the conditions under which uniqueness of update can be guaranteed if the update is carried out. It is worth mentioning that an update is not always ideally unique in practice. Therefore, such works are only effective for the restricted scenarios. A more practical study is to identify an update to database instance D which can enable the specified update to $Q(D)$ while minimizing the side-effect, *e.g.*, unintended update. This is essentially the same as view propagation.

VII. Conclusion

We conclude that the previously established complexity results for the single view case no longer hold in the multiple query case. Instead, we derive the new complexity results even for two *project-free* conjunctive views, and show that the view side-effect problem cannot be approximated within $O(2^{\log^{1-\delta}\|\mathcal{V}\|})$ where $\delta = 1/\log\log^{c}\|\mathcal{V}\|$ for any $c < 0.5$, unless P=NP. This lower bound can also be applied to the balanced deletion problem which is defined for more practical cases. The view side-effect problem and its balanced version can be approximated within $O(2\|\mathcal{V}\|\log\|\Delta\mathcal{V}\|)$ for the general case. We show that the view side-effect problem can be approximated within l and $O(2\sqrt{\|\mathcal{V}\|})$ for the forest cases where l is the maximum *arity*(Q) among all the given queries. Finally, a dynamic programming validates that the view side-effect problem can be solved polynomially for the more restrictive forest case.

Acknowledgment

This work is partly supported by the National Natural Science Foundation of China (NSFC) under grant NOs. 61832003, U1811461, 61732003 and the National Science Foundation (NSF) under grant NOs. 1252292, 1741277, 1829674 and 1704287.

References

[1] F. Bancilhon, N. Spyratos. Update semantics of relational views. *ACM Transactions on Database Systems*, 6(4):557-575, 1981.

[2] M. Bergman, T. Milo, S. Novgorodov, W. Tan. Query-oriented data cleaning with oracles. In *SIGMOD*, pages 1199-1214, 2015.

[3] M. Yakout, A. K. Elmagarmid, J. Nevillem, M. Ouzzani, I. F. Ilyas. Guided data repair. *PVLDB*, 4(5):279-289, 2011.

[4] A. Bohannon, B. C. Pierce, J. A. Vaughan. Relational lenses: a language for updatable views. In *PODS*, pages 338-347, 2006.

[5] D. Brügmann, C. Komusiewicz, H. Moser. On generating triangle-free graphs. *Electronic Notes in Discrete Mathematics*. 32:51-58, 2009.

[6] P. Buneman, S. Khanna, W.-C. Tan. On propagation of deletions and annotations through views. In *PODS*, pages 150-158, 2002.

[7] P. Buneman, A. Chapman, J. Cheney. Provenance management in curated databases. In *SIGMOD*, pages 539-550, 2006.

[8] R. D. Carr, S. Doddi, G. Konjevod, M. Marathe. On the red-blue set cover problem. In *SODA*, pages 345-353, 2002.

[9] A. K. Chandra, P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC*, pages 77-90, 1977.

[10] A. Chapman, H. V. Jagadish. Why not?. In *SIGMOD*, pages 523-534, 2009.

[11] J. Cheney, L. Chiticariu, W.-C. Tan. Provenance in databases: why, how, and where. *Foundations and Trends in Databases*, 1(4):379-474, 2009.

[12] R. Chen, S. Singh, S. Prabhakar. U-DBMS: a database system for managing constantly-evolving data. In *VLDB*, pages 1271-1274, 2005.

[13] H. Chockler, J. Y. Halpern. Responsibility and blame: a structural-model approach. *Journal of Artificial Intelligence Research*, 22:93-115, 2004.

[14] G. Cong, W. Fan, F. Geerts. Annotation propagation revisited for key preserving views. In *CIKM*, pages 632-641, 2006.

[15] G. Cong, W. Fan, F. Geerts, J. Li, J. Luo. On the complexity of view update analysis and its application to annotation propagation. *IEEE Transactions on Knowledge and Data Engineering*, 24(3):506-519, 2012.

[16] S. S. Cosmadakis, C. H. Papadimitriou. Updaies of relational views. In *PODS*, pages 742-760, 1983.

[17] Y. Cui, J. Widom. Run-time translation of view tuple deletions using data lineage. *Technique Report*, Stanford, 2001.

[18] R. Fagin, J. Ullman, M. Y. Vardi. On the semantics of updates in databases. In *PODS*, pages 352-365, 1983.

[19] U. Dayal, P. A. Bernstein. On the correct translation of update operations on relational views. *ACM Transactions on Database Systems*, 7(3):381-416, 1982.

[20] D. Deutch, Y. Moskovitch, V. Tannen. A provenance framework for data-dependent process analysis. *PVLDB*, vol.7, pp 457-468, 2014.

[21] R. G. Downey, M. R. Fellows. Parameterized complexity. Springer-Verlag, New York, 1999.

[22] T. Eiter, T. Lukasiewicz. Complexity results for structure-based causality. *Artificial Intelligence*, 142(1):53-89, 2002.

[23] R. Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *Journal of the ACM*, 30(3):514-550, 1983.

[24] C. Freire, W. Gatterbauer, N. Immerman, A. Meliou. The complexity of resilience and responsibility for self-join-free conjunctive queries. *PVLDB*, 9(3):180-191, 2015.

[25] N. Garg, V. V. Vazirani, M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18(1):3-20, 1997.

[26] M. Grohe. The parameterized complexity of database queries. In *PODS*, pages 82-92, 2002.

[27] J. Håstad. Some optimal inapproximability results. *Journal of the ACM*, 48(4):798-859, 2001.

[28] J. Huang, T. Chen, A. Doan, J. F. Naughton. On the provenance of non-answers to queries over extracted data. *PVLDB*, 1(1):736-747, 2008.

[29] A. M. Keller. Algorithms for translating view updates to database updates for views involving selections, projections, and joins. In *PODS*, pages 154-163, 1985.

[30] B. Kimelfeld. A dichotomy in the complexity of deletion propagation with functional dependencies. In *PODS*, pages 191-202, 2012.
[31] B. Kimelfeld, J. Vondrák, R. Williams. Maximizing conjunctive views in deletion propagation. *ACM Transactions on Database Systems*, 37(4):1-37, 2012.
[32] B. Kimelfeld, J. Vondrák, D. P. Woodruff. Multi-tuple deletion propagation: approximations and complexity. *PVLDB*, 6(13):1558-1569, 2013.
[33] A. Meliou, W. Gatterbauer, K. F. Moore, D. Suciu. The complexity of causality and responsibility for query answers and non-answers. *PVLDB*, 4(1):34-45, 2010.
[34] A. Meliou, W. Gatterbauer, S. Nath, D. Suciu. Tracing data errors with view-conditioned causality. In *PODS*, pages 505-516, 2011.
[35] A. Meliou, S. Roy, D. Suciu. Causality and explanations in databases. *PVLDB*, 7(13):1715-1716, 2014.
[36] D. Miao, Z. Cai, J. Li. On the complexity of bounded view propagation for conjunctive queries. *IEEE Transactions on Knowledge and Data Engineering*, 30(1):115-127, 2018.
[37] D. Miao, X. Liu, J. Li. On the complexity of sampling query feedback restricted database repair of functional dependency violations. *Theoretical Computer Science*, 609:594-605, 2016.
[38] P. Miettinen. On the positive-negative partial set cover problem. *Information Processing Letters*, 108(4):219-221, 2008.
[39] X. Niu, B.S. Arab, S. Lee, Su. Feng, *et. al*. Debugging transactions and tracking their provenance with reenactment. *PVLDB*, 10(12):1857-1860, 2017.
[40] C. H. Papadimitriou, M. Yannakakis. On the complexity of database queries (extended abstract). In *PODS*, 1997, pp. 12-19.
[41] D. Peleg. Approximation algorithms for the label-cover max and red-blue set cover problems. *Journal of Discrete Algorithms*, 5(1):55-64, 2007.
[42] S. Roy, D. Suciu. A formal approach to finding explanations for database queries. In *SIGMOD*, pages 1579-1590, 2014.
[43] M. Y. Vardi. The complexity of relational query languages (extended abstract). In *STOC*, pages 137-146, 1982.
[44] V. Vazirani. Approximation Algorithms. Springer-Verlag, Berlin, 2003.