Database Repairing with Soft Functional Dependencies

Nofar Carmeli, Martin Grohe, Benny Kimelfeld, Ester Livshits, Muhammad Tibi

Database Repairing with Soft Functional Dependencies

NOFAR CARMELI*, Inria, LIRMM, Univ Montpellier, CNRS, France
MARTIN GROHE, RWTH Aachen University, Germany
BENNY KIMELFELD, Technion – Israel Institute of Technology, Israel
ESTER LIVSHITS*, University of Edinburgh, UK
MUHAMMAD TIBI, Technion – Israel Institute of Technology, Israel

A common interpretation of soft constraints penalizes the database for every violation of every constraint, where the penalty is the cost (weight) of the constraint. A computational challenge is that of finding an optimal subset: a collection of database tuples that minimizes the total penalty when each tuple has a cost of being excluded. When the constraints are strict (i.e., have an infinite cost), this subset is a "cardinality repair" of an inconsistent database; in soft interpretations, this subset corresponds to a "most probable world" of a probabilistic database, a "most likely intention" of a probabilistic unclean database, and so on. Within the class of functional dependencies, the complexity of finding a cardinality repair is thoroughly understood. Yet, very little is known about the complexity of finding an optimal subset for the more general soft semantics. The work described in this manuscript makes significant progress in that direction. In addition to general insights about the hardness and approximability of the problem, we present algorithms for two special cases (and some generalizations thereof): a single functional dependency, and a bipartite matching. The latter is the problem of finding an optimal "almost matching" of a bipartite graph where a penalty is paid for every lost edge and every violation of monogamy. For these special cases, we also investigate the complexity of additional computational tasks that arise when the soft constraints are used as a means to represent a probabilistic database via a factor graph, as in the case of a probabilistic unclean database.

CCS Concepts: • **Information systems** → Data cleaning; • **Theory of computation** → Incomplete, inconsistent, and uncertain databases.

Additional Key Words and Phrases: Database inconsistency, database repairs, integrity constraints, soft constraints, functional dependencies

1 INTRODUCTION

Various challenges in data management are based on soft variants of database constraints (also referred to as *weak* or *approximate* constraints). In constraint discovery and mining, for instance, the goal is to find constraints, such as Functional Dependencies (FDs) [9, 19, 22, 30] and beyond [8, 23, 29], that generally hold in the database but not necessarily in a perfect manner. There, the reason for the violations might be rare events (e.g., agreement on the zip code but not the state) or noise (e.g., mistyping). Soft constraints also arise when reasoning about uncertain data [16, 20, 33, 34]—the database is viewed as a probabilistic space over possible worlds, and the violation of a weak constraint in a possible world is viewed as evidence that affects the world's probability.

Our investigation concerns the latter application of soft constraints. To be more precise, the semantics is that of a *parametric factor graph*: the probability of a possible world is the product of *factors* where every violation of the constraint contributes one factor; in turn, this factor is a weight that is assigned upfront to the constraint. This formalism has been adopted in various

*Part of this work was done while the author was at Technion – Israel Institute of Technology.

Authors' addresses: Nofar Carmeli, nofar.carmeli@inria.fr, Inria, LIRMM, Univ Montpellier, CNRS, Montpellier, France; Martin Grohe, grohe@informatik.rwth-aachen.de, RWTH Aachen University, Aachen, Germany; Benny Kimelfeld, bennyk@cs.technion.ac.il, Technion – Israel Institute of Technology, Haifa, Israel; Ester Livshits, ester.livshits@ed.ac.uk, University of Edinburgh, Edinburgh, UK; Muhammad Tibi, m7mdtb@cs.technion.ac.il, Technion – Israel Institute of Technology, Haifa, Israel.

database frameworks that involve uncertainty and soft interpretation of constraints [2, 13, 31, 33–35], and is highly inspired by successful concepts such as the Markov Logic Network (MLN) [32].[1] In these applications, the weights are typically learned from examples. Once we have the weights, the computational problems are the conventional in probabilistic modeling: marginal inference (compute the probability of a query answer) and maximum likelihood (find the most probable world)—the problem that we focus on here.

More specifically, we consider the case where constraints are FDs. By taking the logarithms of the factors, the problem we study can be formally defined as follows. We are given a database D and a set Δ of FDs, where every tuple and every FD has a weight (which is a nonnegative number). We wish to obtain a cleaner subset E of D by deleting tuples. The cost of the subset E includes a penalty for every deleted tuple, and a penalty for every violation of (i.e., pair of tuples that jointly violate) an FD; the penalties are the weights of the tuple and the FD, respectively. The goal is to find a subset E with a minimal cost. In what follows, we refer to such E as an *optimal subset* and to the optimization problem of finding an optimal subset as *soft repairing*. The optimal subset corresponds to the "most likely intention" in the Probabilistic Unclean Database (PUD) framework of De Sa, Ilyas, Kimelfeld, Ré and Rekatsinas [33] in a restricted case that is studied in their work; the PUD model provides the theoretical formulation of the HoloClean [31] and HoloDetect [17] systems. The optimal subset is also referred to as the "most probable world" in the probabilistic database model of Sen, Deshpande and Getoor [34]. In the special case where the FDs are hard constraints (i.e., their weight is infinite or just too large to pay), an optimal subset is simply what is known as a "cardinality repair" [26] or, equivalently [25], a "most probable database" [16].

Even though the problem has arisen and been attacked in several systems and frameworks, including successful systems such as HoloClean, the assumption has always been that this problem is intractable and, hence, heuristic solutions were deployed. The computational challenge of soft repairing is that there are exponentially many candidate subsets. And indeed, the problem can be computationally intractable for certain combinations of FDs. An easy argument shows that the problem is at least as computationally hard as in the case of hard constraints (e.g., as shown by De Sa et al. [33]), and there are cases where the latter is intractable [25]. Nevertheless, there are also many cases where the problem is tractable for hard constraints (and, as we explain below, past results show precisely how to classify a set of FDs into the tractable or intractable side). In these cases, is the soft version more difficult than the hard counterpart (i.e., finding a cardinality repair)? Not necessarily. De Sa et al. [33] showed otherwise for the case of a key constraint, where they give a polynomial-time algorithm. This is a restricted case, and they left the more general case (that we study here) open. Indeed, we found it challenging to generalize their tractability beyond a single key constraint. Nevertheless, we have made a substantial progress towards a full answer to the question, and in this article we describe our findings.

Formally, we investigate the data complexity of the problem, where the database schema and the FD set are fixed, and the input consists of the database D and all involved weights. Moreover, we assume that D consists of a single relation; this is done without loss of generality, since the problem boils down to soft repairing each relation independently (since an FD does not involve more than one relation). As aforesaid, the complexity of the problem is very well understood in the case of hard constraints (cardinality repairs). Gribkoff, Van den Broeck and Suciu [16] established complexity results for the case of unary FDs (having a single attribute on the left-hand side), and Livshits, Kimelfeld and Roy [25] completed the picture to a full (effective) dichotomy over all possible sets of FDs. For example, the problem is solvable in polynomial time for the FD sets

[1] More precisely, an MLN can be viewed as a database with weak constraints, where the set of facts includes all possible tuples over the (finite) domains of the attributes.

$\{A \rightarrow B\}$, $\{A \rightarrow B, B \rightarrow A\}$ and $\{A \rightarrow B, B \rightarrow A, B \rightarrow C\}$, but is NP-hard for $\{A \rightarrow B, B \rightarrow C\}$. Livshits et al. [25] also gave approximation upper bounds for the problem, and these results were later strengthened by Miao et al. [28] who gave inapproximability results and tighter upper bounds, in addition to empirical evidence of effectiveness via a practical implementation.

When FDs are soft (and violations are allowed), the problem seems to be fundamentally more challenging, both to solve and to reason about. As said earlier, for every Δ where it is intractable to find a cardinality repair, the soft version is also intractable. But the other direction is false (under conventional complexity assumptions). For example, soft repairing is hard for $\Delta = \{A \rightarrow B, B \rightarrow A, B \rightarrow C\}$, for the following reason. We can set the weights of $A \rightarrow B$ and $B \rightarrow C$ to be very high, making each of them a hard constraint in effect, and the weight of $B \rightarrow A$ very low, making it ignorable in effect. Hence, an optimal subset is a cardinality repair for $\{A \rightarrow B, B \rightarrow C\}$ that, as said above, is hard to compute. So, which sets of FDs have a tractable soft repairing? The only polynomial-time algorithm we are aware of is the aforementioned algorithm of De Sa et al. [33] for the special case of a single key constraint, that is, $\Delta = \{X \rightarrow Y\}$ where XY contain all of the schema attributes.

We first generalize the tractability of De Sa et al. [33] from a key constraint to an arbitrary FD. Like theirs, our algorithm employs dynamic programming, but in a more involved fashion. This is because their algorithm is based on the fact that in a key constraint $X \rightarrow Y$, any two tuples that agree on X are necessarily conflicting. We also show that our algorithm can be generalized to additional sets of FDs. For example, it turns out that the FD set {name $\rightarrow$ address , name address $\rightarrow$ email} is tractable as well. (Note that the address attribute on the left-hand side of the second FD is not redundant, as in the ordinary semantics, since the FDs are treated as soft constraints.) In Section 4 we phrase the more general condition that this FD set satisfies.

While trying, we could not find any way of generalizing the dynamic-programming algorithm beyond the above class. At that point of our research, we were conjecturing that this class captures *all* tractable cases. Yet, to our surprise, we were proven wrong. We investigated the case of a *matching constraint* that required a special treatment in the case of hard constraints [25]. These are FD sets $\Delta = \{X \rightarrow Y, X' \rightarrow Y'\}$ over a schema with the attributes $A_1, \ldots, A_k$ where $X \cup Y = X' \cup Y' = X \cup X' = \{A_1, \ldots, A_k\}$ and there are no attributes other than $A_1, \ldots, A_k$. The simplest example is $\{A \rightarrow B, B \rightarrow A\}$ over the binary schema (A, B) that represents a bipartite graph, and the problem is that of finding the best "almost matching" of a bipartite graph where a penalty is paid for every lost edge and every violation of monogamy. A more involved example is $\{AB \rightarrow C, AC \rightarrow B\}$ over the schema (A, B, C). For such constraints, we were able to employ a tool that played a key role in our polynomial-time algorithm: reduction to the *Minimum Cost Maximum Flow* (MCMF) problem [14]. In this version of the flow problem, each edge has a capacity and a cost, and the goal is to find, among the maximum flows, the one with the least total cost.

The general conclusion is that we identified two main techniques that can be used for solving the problem of repairing with soft constraints in an exact manner: the first is a nontrivial generalization of the dynamic programming that was used previously for a single key [33], and the second is a connection to a flow problem with costs (that, as far as we are aware, has not been applied yet in the context of data repairing). Whether our algorithms cover all of tractable cases remains an open problem for future investigation. (In the Conclusions we discuss the simplest FD sets where the question is left unsolved.) We do show, however, that there is a polynomial-time approximation algorithm with an approximation factor 3, that is, an algorithm that returns a subset where the penalty is at most three times the optimum.

Next, we investigate the soft constraints beyond the challenge of optimal repairing. Recall that soft repairing is the problem of finding the most probable world of a probabilistic database represented as a parametric factor graph, where the soft constraints are used as the templates for

the factors. Hence, we can view the soft constraints as representing a probabilistic database that is driven by the randomness of repairing. With that in mind, other fundamental tasks arise. One example is that of *marginal inference*, where the goal is to compute the probability of an event such as the inclusion of a tuple (or at least one tuple that satisfies some condition). Marginal inference often requires the computation of the normalization factor of the factor graph, a number known as the *partition function* [6, 32]. In turn, the partition function is also needed for other related tasks such as computing the most probable explanation and weight learning [12, 27]. Another general tool for solving probabilistic inference tasks is that of *sampling*, that is, generating a random subset of the database where the probability of each subset is the same as its probability in the representation via the soft constraints. This is a challenging tasks in discriminative models (as opposed to generative models) such as soft constraints.

Focusing on the two tractable FD sets for soft repairing, we investigate the complexity of three tasks: marginal inference, computation of the partition function, and sampling. In the case of marginal inference, we consider queries that test for the existence of a given tuple (i.e., we wish to compute the marginal probability of the tuple), and more generally, the containment of at least one tuple from a given subset (e.g., all of the tuples that satisfy some property). We show that these tasks can be solved in polynomial time for a single FD and the aforementioned generalization; in contrast, we prove that for $\{A \to B, B \to A\}$ it is intractable (#P-hard) to compute the partition function, and marginal inference is intractable already for the query that asks whether the database is nonempty.

Comparison to an earlier conference version. A preliminary version of this manuscript has been published in conference proceedings [3]. Compared to that version, this manuscript includes several additions. The main addition is the investigation of the probabilistic inference tasks that were not mentioned in the conference version. In particular, Section 6 is new. Another addition is a discussion on the FD sets that are not covered by our algorithms, and specifically the reason why the set $\{A \to B, A \to C\}$ is apparently more challenging computationally than (the logically equivalent) $\{A \to BC\}$ when FDs are interpreted as soft constraints. This discussion is given in Section 4.3. Finally, throughout the manuscript, we include additional examples, discussions, and explanations.

Organization. The rest of the manuscript is organized as follows. We give the formal setup and the problem definition in Section 2. We then discuss the complexity of the general problem and its relationship to past results in Section 3. We describe our algorithm for soft repairing in Sections 4 and 5 for a single FD and a matching constraint, respectively. We investigate the probabilistic inference tasks in Section 6, and finally conclude in Section 7.

2 FORMAL SETUP

We begin with preliminary definitions and terminology that we use throughout the manuscript.

2.1 Databases, FDs and Repairs

For the main problem that we study in this work (defined later as Problem 2.1), it suffices to consider a database with a single relation. Hence, we define schemas and databases accordingly. A *relation schema* $R(A_1, \ldots, A_k)$ consists of a relation symbol R and a set $\{A_1, \ldots, A_k\}$ of attributes. A *database* D over R is a set of facts f of the form $R(c_1, \ldots, c_k)$, where each c_i is a *constant*. We denote by $f[A_i]$ the value that the fact f associates with attribute A_i (i.e., $f[A_i] = c_i$). Similarly, if $X = B_1 \cdots B_k$ is a sequence of attributes from $\{A_1, \ldots, A_k\}$, then $f[X]$ is the tuple $(f[B_1], \ldots, f[B_k])$. We assume that every fact $f \in D$ is associated with a nonnegative weight, hereafter denoted w_f. (The weight of a fact is sometimes derived from a validity/existence probability [16, 34].)

A *Functional Dependency* (FD) over the relation schema $R(A_1, \ldots, A_k)$ is an expression φ of the form $X \rightarrow Y$ where $X, Y \subseteq \{A_1, \ldots, A_k\}$. A *violation* of an FD in a database D is a pair $\{f, g\}$ of tuples from D that agrees on the left-hand side (i.e., $f[X] = g[X]$) but disagrees on the right-hand side (i.e., $f[Y] \neq g[Y]$). An FD $X \rightarrow Y$ is *trivial* if $Y \subseteq X$. We denote by $\mathrm{vio}(D, \varphi)$ the set of all the violations of the FD φ in D. We say that D *satisfies* φ, denoted $D \models \varphi$, if it has no violations (i.e., $\mathrm{vio}(D, \varphi)$ is empty). The database D satisfies a set Δ of FDs, denoted by $D \models \Delta$, if D satisfies every FD in Δ; otherwise, D violates Δ (denoted $D \not\models \Delta$). An FD $X \rightarrow Y$ is entailed by an FD set Δ if every database D that satisfies Δ also satisfies $X \rightarrow Y$. The closure of an attribute set X w.r.t. Δ, denoted $\mathrm{Closure}_\Delta(X)$, is the set of all attributes A such that the FD $X \rightarrow \{A\}$ is entailed by Δ. We assume that every FD $\varphi \in \Delta$ has a nonnegative weight denoted by w_φ.

When there is no risk of ambiguity, we may omit the specification of the relation schema $R(A_1, \ldots, A_k)$ and simply assume that the involved databases and constraints are all over the same schema.

Let D be a database and let Δ be a set of FDs. A *repair* (*of D w.r.t. Δ*) is an inclusion-maximal consistent subset E; that is, $E \subseteq D$ and $E \models \Delta$, and moreover, $E' \not\models \Delta$ for every E' such that $E \subsetneq E' \subseteq D$. Note that the number of repairs can be exponential in the number of facts of D. A *cardinality repair* is a repair E of a maximal cardinality (i.e., $|E| \geq |E'|$ for every repair E').

2.2 Soft Constraints

We define the concept of *soft constraints* (or *weak constraints* or *weighted rules*) in the standard way of "penalizing" the database for every missing fact, on the one hand, and every violation, on the other hand. This is the concept adopted in past work such as the *parfactors* of De Sa et al. [33], the *soft keys* of Jha et al. [20], and the *PrDB* model of Sen et al. [34]. The concept can be viewed as a special case of the *Markov Logic Network* (MLN) [32].

Formally, let D be a database and Δ a set of FDs. The *cost* of a subset E of a database D is then defined as follows.

$$\mathrm{cost}(E \mid D) \stackrel{\text{def}}{=} \left(\sum_{f \in (D \setminus E)} w_f \right) + \left(\sum_{\varphi \in \Delta} w_\varphi |\mathrm{vio}(E, \varphi)| \right) \tag{1}$$

As for the computational model, we assume that every weight is a rational number r/q that is represented using the numerator and the denominator, namely (r, q), where each of the two is an integer represented in the standard binary manner.

2.3 Problem Definition: Soft Repairing

The main problem we study in this manuscript, referred to as *soft repairing*, is the optimization problem of finding a database subset with a minimal cost. Since we consider the data complexity of the problem, we associate with each relation schema and set of FDs a separate computational problem.

Problem 2.1 (Soft Repairing). *Let $R(A_1, \ldots, A_k)$ be a relation schema and Δ a set of FDs.* Soft repairing *(*for $R(A_1, \ldots, A_k)$ and Δ*) is the following optimization problem: Given a database D, find an optimal subset of D, that is, a subset E of D with a minimal* $\mathrm{cost}(E \mid D)$.

Note that a cardinality repair is an optimal subset in the special case where the weight w_φ of every FD φ is ∞ (or just higher than the cost of deleting the entire database), and the weight w_f of every fact f is 1. Livshits et al. [25] studied the complexity of finding a *weighted cardinality repair*, which is the same as a cardinality repair but the weight w_f of every fact f can be arbitrary. Hence, both types of cardinality repairs are consistent (i.e., the constraints are strictly satisfied).

Flights

Flight	Airline	Date	Origin	Destination	Airplane	
UA123	United Airlines	01/01/2021	LA	NY	N652NW	3
UA123	United Airlines	01/01/2021	NY	UT	N652NW	2
UA123	Delta	01/01/2021	LA	NY	N652NW	1
DL456	Southwest	02/01/2021	NC	MA	N713DX	2
DL456	Southwest	03/01/2021	NJ	FL	N245DX	1
DL456	Delta	03/01/2021	CA	IL	N819US	4

(a) D

Flights

Flight	Airline	Date	Origin	Destination	Airplane	
UA123	United Airlines	01/01/2021	NY	UT	N652NW	2
DL456	Southwest	02/01/2021	NC	MA	N713DX	2
DL456	Southwest	03/01/2021	NJ	FL	N245DX	1

(b) E_1

Flights

Flight	Airline	Date	Origin	Destination	Airplane	
UA123	United Airlines	01/01/2021	LA	NY	N652NW	3
DL456	Delta	03/01/2021	CA	IL	N819US	4

(c) E_2

Flights

Flight	Airline	Date	Origin	Destination	Airplane	
UA123	United Airlines	01/01/2021	LA	NY	N652NW	3
UA123	United Airlines	01/01/2021	NY	UT	N652NW	2
DL456	Delta	03/01/2021	CA	IL	N819US	4

(d) E_3

Fig. 1. For the relation Flights(Flight, Airline, Date, Origin, Destination, Airplane) and the FDs Flight $\rightarrow$ Airline (with $w_{\varphi_1} = 5$) and Flight Airline Date $\rightarrow$ Destination (with $w_{\varphi_2} = 1$), a database D, a cardinality repair E_1, a weighted cardinality repair E_2, and an optimal subset E_3.

In contrast, an optimal subset in the general case may violate one or more of the FDs. In the next section we recall the known complexity results for cardinality and weighted cardinality repairs.

Example 2.2. Our running example is based on the database of Figure 1 over the relation schema

Flights(Flight, Airline, Date, Origin, Destination, Airplane)

that contains information about domestic flights in the United States. The weight of each tuple appears on the rightmost column. The FD set Δ consists of the following FDs:

- Flight $\rightarrow$ Airline: a flight is associated with a single airline.
- Flight Airline Date $\rightarrow$ Destination: a flight on a certain date has a single destination.

We assume that the weight of the first FD is 5, and the weight of the second FD is 1.

The database E_1 of Figure 1 is a cardinality repair of D as no repair of D can be obtained by removing less then three facts. However, E_1 is not a weighted cardinality repair, since its cost is eight, while the cost of E_2 is six. The reader can easily verify that E_2 is a weighted cardinality repair of D. Finally, E_3 is not a repair of D in the traditional sense as it contains a violation of the second FD, but it is an optimal subset of D with $\text{cost}(E_3 \mid D) = 5$. □

Algorithm 1 Simplify(Δ)

Remove trivial FDs from Δ
if Δ is not empty **then**
  find a removable pair (X, Y) of attribute set
  $\Delta := \Delta - XY$
return Δ

Remark 2.3. *We end this section with a comment about the case of multiple relations, which is beyond our formal setup. When the database has multiple relations and each relation schema has associated soft FDs, soft repairing decomposes to the individual relations straightforwardly: the least-cost subset of D consists of the least-cost subset of each relation of D. Hence, it suffices to discuss the problem in the simple case of a single-relation database.*

3 PRELIMINARY COMPLEXITY ANALYSIS

We consider the data complexity of the problem of computing an optimal subset. We assume that the schema and the set of FDs are fixed, and the input consists of the database. Livshits et al. [25] studied the problems of finding a cardinality repair and a weighted cardinality repair, and established a dichotomy over the space of all the sets of functional dependencies. In particular, they introduced an algorithm that, given a set Δ of FDs, decides whether:

(1) A weighted cardinality repair can be computed in polynomial time; *or*
(2) Finding a (weighted) cardinality repair is APX-complete.[2]

No other possibility exists. The algorithm is a recursive procedure that attempts to simplify Δ at each iteration by finding a *removable* pair (X, Y) of attribute sets, and removing every attribute of X and Y from all the FDs in Δ (which we denote by $\Delta - XY$). We say that a pair (X, Y) of attribute sets is removable if it satisfies the following properties:

- $\text{Closure}_\Delta(X) = \text{Closure}_\Delta(Y)$,
- XY is nonempty,
- the left-hand side of every FD in Δ includes either X or Y.

Note that the sets X and Y may be the same, and then the condition states that every FD contains X on the left-hand side.

The simplification procedure for an FD set Δ is depicted here as Algorithm 1. If we are able to transform Δ to an empty set of FDs by repeatedly applying simplifications, then the algorithm returns true and finding an optimal consistent subset is solvable in polynomial time. Otherwise, the algorithm returns false and the problem is APX-complete. We state their result for later reference.

Theorem 3.1. [25] *Let Δ be a set of FDs. If Δ can be emptied via Simplify(Δ) steps, then a weighted cardinality repair can be computed in polynomial time; otherwise, finding a cardinality repair is APX-complete.*

The hardness side of Theorem 3.1 immediately implies the hardness of the more general soft-repairing problem. Yet, the other direction (tractability generalizes) is not necessarily true. As discussed in the Introduction, if $\Delta = \{A \to B, B \to A, B \to C\}$, then Δ, as a set of hard constraints, is classified as tractable according to Algorithm 1; however, this is not the case for soft constraints. We can generalize this example by stating that if Δ contains a *subset* that is hard according to Theorem 3.1, then soft repairing is hard. (This does not hold when considering only hard constraints,

[2]Recall that APX is the class of NP optimization problems that admit constant-ratio approximations in polynomial time. Hardness in APX is via the so called "PTAS" reductions (cf. textbooks on approximation complexity, e.g., [15]).

as the example shows that there exists an easy Δ with a hard subset.) In the following sections, we are going to discuss tractable cases of FD sets. Before that, we will show that the problem becomes tractable if one settles for an approximation.

3.1 Approximation

The following theorem shows that soft repairing admits a constant-ratio *approximation*, for the constant three, in polynomial time. This means that there is a polynomial-time algorithm for finding a subset with a cost of at most three times the minimum.

Theorem 3.2. *For all FD sets, soft repairing admits a 3-approximation in polynomial time.*

Proof. We reduce soft repairing to the problem of finding a minimum weighted set cover where every element belongs to 3 sets. A simple greedy algorithm finds a 3-approximation to this problem in linear time [18].

We set the elements to be $\{(\{f, g\}, \delta) \mid f, g \in D, \delta \in \Delta, f \text{ and } g \text{ contradict } \delta\}$. Each element $(\{f, g\}, \delta)$ belongs to three sets: f with weight w_f, g with weight w_g, and $(\{f, g\}, \delta)$ with weight w_δ. Each minimal solution to this set cover problem can be translated to a soft repair: the selected sets that correspond to tuples are removed in the repair. Indeed, a minimal set cover of such a construction has to resolve each conflict by either paying for the removal of at least one of the tuples or paying for the violation. □

In terms of formal complexity, Theorem 3.2 implies that the problem of soft repairing is in APX (for every set of FDs). From this, from Theorem 3.1 and from the discussion that follows Theorem 3.1, we conclude the following.

Corollary 3.3. *Let Δ be a set of FDs. Soft repairing for Δ is in APX. Moreover, if any subset of Δ cannot be emptied via Simplify(Δ) steps, then soft repairing is APX-complete for Δ.*

In the next two sections, we present exact algorithms for soft repairing with respect to restricted classes of FD sets.

4 ALGORITHM FOR A SINGLE FUNCTIONAL DEPENDENCY

In this section, we consider the case of a single functional dependency, and present a polynomial-time algorithm for soft repairing. Later on (in Section 4.2), we will extend the algorithm to a more general class of FD sets.

4.1 Single FD

We assume that the single FD is $\varphi \stackrel{\text{def}}{=} X \rightarrow Y$ and that our input database is D. We split D into *blocks* and *subblocks*, as we explain next. The blocks of D are the maximal subsets of D that agree on the X values. Denote these blocks by $D_1, \ldots, D_m$. Note that there are no conflicts across blocks; hence, we can solve the problem separately for each block and then an optimal subset E is simply the union of optimal subsets E_i of the blocks D_i:

$$E = \bigcup_{i=1}^{m} E_i$$

The subblocks of a block D_i are the maximal subsets of D_i that agree on the Y values (in addition to the X values). We denote these subblocks by $D_{i,1}, \ldots, D_{i,q_i}$. Note that two facts from the same subblock are consistent, while two facts from different subblocks are conflicting.

Example 4.1. Consider the database D of Figure 1 and the FD Flight $\rightarrow$ Airline. The database has two blocks:

Flight	Airline	Date	Origin	Destination	Airplane	
UA123	United Airlines	01/01/2021	LA	NY	N652NW	3
UA123	United Airlines	01/01/2021	NY	UT	N652NW	2
UA123	Delta	01/01/2021	LA	NY	N652NW	1

corresponding to the value UA123 of the attribute Flight, and

Flight	Airline	Date	Origin	Destination	Airplane	
DL456	Southwest	02/01/2021	NC	MA	N713DX	2
DL456	Southwest	03/01/2021	NJ	FL	N245DX	1
DL456	Delta	03/01/2021	CA	IL	N819US	4

corresponding to the value DL456 of the attribute Flight. We denote these blocks by D_1 and D_2, respectively.

The first block can be further split into the following two subblocks:

Flight	Airline	Date	Origin	Destination	Airplane	
UA123	United Airlines	01/01/2021	LA	NY	N652NW	3
UA123	United Airlines	01/01/2021	NY	UT	N652NW	2

Flight	Airline	Date	Origin	Destination	Airplane	
UA123	Delta	01/01/2021	LA	NY	N652NW	1

corresponding the values United Airlines and Delta of the attribute Airline and denoted $D_{1,1}$ and $D_{1,2}$, respectively. Similarly, the second block is split into the following two subblocks $D_{2,1}$ and $D_{2,2}$, corresponding to the values Southwest and Delta of the attribute Airline, respectively.

Flight	Airline	Date	Origin	Destination	Airplane	
DL456	Southwest	02/01/2021	NC	MA	N713DX	2
DL456	Southwest	03/01/2021	NJ	FL	N245DX	1

Flight	Airline	Date	Origin	Destination	Airplane	
DL456	Delta	03/01/2021	CA	IL	N819US	4

From here we continue with dynamic programming. For a number $j \in \{0, \ldots, q_i\}$, where q_i is the number of subblocks of D_i, and a number $k \in \{0, \ldots, |D_{i,1} \cup \cdots \cup D_{i,j}|\}$ of facts, we define the following values that we are going to compute:

- $C[i, j, k]$ is the cost of an optimal subset of $D_{i,1} \cup \cdots \cup D_{i,j}$ (i.e., the union of the first j subblocks) with precisely k facts.
- $F[i, j, k]$ is a subset of $D_{i,1} \cup \cdots \cup D_{i,j}$ that realizes $C[i, j, k]$, that is,

$$|F[i, j, k]| = k \quad \wedge \quad \text{cost}\left(F[i, j, k] \mid D_{i,1} \cup \cdots \cup D_{i,j}\right) = C[i, j, k]$$

Note that if multiple choices of $F[i, j, k]$ exist, we select an arbitrary one. Once we compute the $F[i, q_i, k]$, we are done since it then suffices to return the best subset over all k:

$$E_i = F[i, q_i, k] \text{ for } k = \operatorname*{argmin}_k C[i, q_i, k]$$

Example 4.2. We continue Example 4.1. We assume that the weight w_φ of the FD Flight $\rightarrow$ Airline is 5. Consider the block D_2 and its subblock $D_{2,1}$. The value $C[2, 1, k]$, for any k, is the cost of an optimal subset of $D_{2,1}$ with k facts. We have the following:

$$C[2, 1, 0] = 2 + 1 = 3 \text{ (as we have to delete both facts),}$$

$C[2,1,1] = 1$ (as deleting the second fact is less costly than deleting the first),

$C[2,1,2] = 0$ (as there are no violations of the FD in a subblock).

Hence, when considering only the subblock $D_{2,1}$, an optimal subset is the one that contains both facts. Now, we also consider the subblock $D_{2,2}$. Here, the value $C[2,2,k]$, for any k, is the cost of an optimal subset of $D_{2,1} \cup D_{2,2}$ with k facts. We have that:

$$C[2,2,0] = C[2,1,0] + 4 = 7$$

as we have to delete all three facts. This is equivalent to choosing an optimal subset of $D_{2,1}$ with 0 facts (represented by the term $C[2,1,0]$) and an optimal subset of $D_{2,2}$ with 0 facts. Moreover,

$$C[2,2,1] = C[2,1,0] = 3$$

since keeping the fact of $D_{2,2}$ is less costly than keeping any fact of $D_{2,1}$; hence, we choose an optimal subset of $D_{2,1}$ with 0 facts (represented by the term $C[2,1,0]$) and an optimal subset of $D_{2,2}$ with 1 fact. We also have that:

$$C[2,2,2] = C[2,1,2] + 4 = 4$$

as if we keep a fact from each subblock, we will have to pay the cost 5 of the FD violation. Thus, deleting the only fact of $D_{2,2}$ has the lowest cost, and this option is obtained by choosing an optimal subset of $D_{2,1}$ with 2 facts (and cost $C[2,1,2]$) and an optimal susbet of $D_{2,2}$ with 0 facts. Finally,

$$C[2,2,3] = C[2,1,2] + 5 + 5 = 10$$

because the fact of $D_{2,2}$ violates the FD with every fact of $D_{2,1}$. Here, we choose an optimal subset of $D_{2,1}$ with 2 facts (and cost $C[2,1,2]$) and an optimal subset of $D_{2,2}$ with 1 fact, and pay 5 for each violation of the FD. We conclude that an optimal subset of $D_{2,1} \cup D_{2,2}$ (hence, of the block D_2) is the one that keeps the fact of $D_{2,2}$ and removes the facts of $D_{2,1}$; the cost of this subset is 3.

We now generalize the idea of Example 4.2 and show how to compute $C[i,j,k]$ and $F[i,j,k]$. We will focus on the former, as the latter is obtained by straightforward bookkeeping. The key observation is that if we decide to delete t facts from $D_{i,j}$, then we always prefer to delete the t facts with the minimal weight. We use this observation as follows.

For a subblock $D_{i,j}$ and $t \in \{0,\ldots,|D_{i,j}|\}$, denote by $\mathrm{top}(t, D_{i,j})$ an arbitrary subset of $D_{i,j}$ with t facts of the highest weight. Hence, $\mathrm{top}(t, D_{i,j})$ is obtained by taking a prefix of size t when sorting the tuples of $D_{i,j}$ from the heaviest to the lightest. Then $C[i,j,k]$ is computed as follows.

$$C[i,j,k] = \begin{cases} 0 & j = 0 \text{ and } k = 0; \\ \infty & j = 0 \text{ and } k > 0; \\ \min\limits_{t \in \{0,\ldots,k\}} \Big(C[i,j-1,k-t] + t(k-t)w_\varphi + \sum\limits_{\substack{f \in D_{i,j} \setminus \\ \mathrm{top}(t,D_{i,j})}} w_f \Big) & \text{otherwise.} \end{cases}$$

The first two cases (where $j = 0$) refer to the case where the database is empty. If we need an empty repair, this has no cost as we remove no tuples and violate no FDs. If we need a larger repair, then this is not possible, and we treat the cost as infinite. If the database we repair is nonempty (the third case), we go over all options for the number t of facts taken from the subblock $D_{i,j}$ and choose an option with the minimum cost. This cost consists of the following components:

- The cost of repairing within $D_{i,j}$ with t tuples is $\sum_{f \in D_{i,j} \setminus \mathrm{top}(t,D_{i,j})} w_f$. This is the cost of removing every fact that is not in $\mathrm{top}(t, D_{i,j})$ from the jth subblock.
- The cost of repairing within $D_{i,1} \cup \cdots \cup D_{i,j-1}$ with $k - t$ tuples is $C[i, j-1, k-t]$.

Algorithm 2 L/C-Simplify(Δ)

```
remove trivial FDs from Δ
if Δ is not empty then
    find A such that in each FD, A is either an lhs attribute or a consensus attribute
    Δ := Δ − A
return Δ
```

- The cost of violations between $D_{i,1} \cup \cdots \cup D_{i,j-1}$ and $D_{i,j}$ is $t(k-t)w_\varphi$. This is the cost of the violations in which the jth subblock participates: any combination of a fact from $D_{i,j}$ and a fact from the other subblocks is a violation of φ.

This completes the description of the algorithm. The correctness is a straightforward conclusion from this description due to the definition of the cost in Equation (1). In conclusion, we establish the following theorem.

Theorem 4.3. *In the case of a single FD, soft repairing can be solved in polynomial time.*

In the next section, we extend our algorithm beyond a single FD and, accordingly, establish a generalization of Theorem 4.3, namely Theorem 4.5.

4.2 Generalization

We now show how the idea from the previous section can be generalized to some FD sets beyond singletons. An attribute A is an *lhs attribute* of an FD $X \rightarrow Y$ if $A \in X$, and it is a *consensus attribute* of $X \rightarrow Y$ if $X = \emptyset$ and $A \in Y$ (hence, $X \rightarrow Y$ states that all tuples should have the same A value). The simplification step of Algorithm 2 removes an attribute A if for every FD in Δ, it is either an lhs or a consensus attribute.

Clearly, if Δ consists of a single FD, then it can be emptied by repeatedly applying L/C-Simplify steps. We will show that soft repairing for Δ is solvable in polynomial time whenever it can be emptied via L/C-Simplify(Δ) steps. Next, we show an example that involves multiple FDs.

Example 4.4. Consider the database and the FD set of our running example (Example 2.2). This FD set, which we denote here by Δ_1, can be emptied via L/C-Simplify(Δ) steps, by selecting attributes in the following order:

$$\begin{aligned}
&\{\mathsf{Flight} \rightarrow \mathsf{Airline},\ \mathsf{Flight\ Airline\ Date} \rightarrow \mathsf{Destination}\}\\
\mathsf{Flight}:&\{\emptyset \rightarrow \mathsf{Airline},\ \mathsf{Airline\ Date} \rightarrow \mathsf{Destination}\}\\
\mathsf{Airline}:&\{\mathsf{Date} \rightarrow \mathsf{Destination}\}\\
\mathsf{Date}:&\{\emptyset \rightarrow \mathsf{Destination}\}\\
\mathsf{Destination}:&\{\}
\end{aligned}$$

Hence, this section shows that soft repairing can be solved in polynomial time for Δ_1.

Next, consider the FD set Δ_2 consisting of the following FDs: Flight $\rightarrow$ Airline and Flight Date $\rightarrow$ Destination. This FD set is logically equivalent to Δ_1; hence, they both entail the exact same cardinality repairs. However, these sets are no longer equivalent when considering soft repairing. In particular, two facts that agree on the values of the Flight and Date attributes, but disagree on the values of the Airline and Destination attributes, violate only one FD in Δ_1 but two FDs in Δ_2, which affects the cost of keeping these two tuples in the database. In fact, the FD set Δ_2 cannot be emptied via L/C-Simplify(Δ) steps, as after removing the Flight attribute, no other attribute is either an lhs or a consensus attribute of the remaining FDs. The complexity of soft repairing for Δ_2 remains an open problem. □

Note that whenever Δ can be emptied via L/C-Simplify(Δ) steps, it can also be emptied via Simplify(Δ) steps. Indeed, if L/C-Simplify(Δ) eliminates the attribute A, then we can take: *(a)* $X = \{A\}$ and $Y = \emptyset$ in Algorithm 1 if A is a consensus attribute of some FD, *or (b)* $X = Y = \{A\}$ if A is an lhs attribute of every FD. This is expected due to Theorem 3.1 and the observation of Section 3 that soft-repairing is hard whenever computing a cardinality repair is hard.

We now present a polynomial-time algorithm for soft repairing in the case where Δ can be emptied via L/C-Simplify(Δ) steps. Our algorithm generalizes the idea of the algorithm for a single FD, and we again use dynamic programming. The main observation is as follows. Let A be an attribute chosen by L/C-Simplify(Δ), and let $D_1, \ldots, D_m$ be the maximal subsets of D that agree on the value of A, which we refer to as blocks (w.r.t. A). Two facts from different blocks violate *all* of the FDs wherein A is a consensus attribute and *none* of the FDs wherein A is an lhs attribute. Therefore, to compute the cost of a soft repair, each pair of facts from different blocks is charged with the violation of all FDs wherein A is a consensus attribute. Then, we can remove A from all FDs and continue the computation separately for each block.

We fix an order of the attributes in which they can be removed with L/C-Simplify. Intuitively, we handle the attributes one after the other in that order. When we handle the ℓth attribute, the first $\ell - 1$ attributes already have a predefined value, stored as an assignment τ. To support the computation, as we did in the case of a single FD, we further allow to restrict the number j of values that we can consider for the currently handled (ℓth) attribute and to restrict the number k of tuples we keep.

More specifically, our recursion computes the cost of repairing a certain subset of the database according to a simplified set of FDs by keeping a predetermined number of tuples:

- The FD set we consider is the one obtained after removing the first $\ell - 1$ attributes via L/C-Simplify(Δ) steps.
- The database we consider is the original one filtered as follows: for each of the first $\ell - 1$ attributes, we only keep tuples with a specific value, given by the assignment τ; for the ℓth attribute, we only keep the tuples with the first j possible values for this attribute.
- The repair must consist of exactly k tuples.

We now set some notation. Let Δ be an FD set that can be emptied via L/C-Simplify(Δ) steps, and let $A_1, \ldots, A_n$ be the attributes in the order of such an elimination process. For each $\ell \in \{1, \ldots, n+1\}$, we denote by Δ_ℓ the FD set in line 2 of the ℓth iteration of this execution (after removing the trivial FDs). Thus, Δ_1 contains every non-trivial FD of Δ, and Δ_{n+1} is empty. We also denote by w_ℓ the total weight of the FDs in Δ_ℓ of which A_ℓ is a consensus attribute (if there are no such FDs, then $w_\ell = 0$). Given $1 \leq \ell \leq n+1$, if τ is an assignment to the attributes $A_1, \ldots, A_{\ell-1}$, then D^τ denotes the database $\sigma_\tau D$ (i.e., the database that contains all the tuples that agree with τ on the values of the attributes $A_1, \ldots, A_{\ell-1}$). We denote by $D^\tau_1, \ldots, D^\tau_{q^\tau_\ell}$ the blocks of D^τ w.r.t. A_ℓ. That is, q^τ_ℓ denotes the number of different values that A_ℓ can take in D^τ, while $D^\tau_1, \ldots, D^\tau_{q^\tau_\ell}$ denote a partition of D^τ according to the value of A_ℓ. Moreover, we denote by $\tau \wedge (A_\ell = j)$ the assignment to the attributes $A_1, \ldots, A_\ell$ given by block D^τ_j. We denote by $F[\ell, \tau, j, k]$ an optimal subset of $D^\tau_1 \cup \cdots \cup D^\tau_j$ of size k w.r.t. Δ_ℓ, and we denote its cost by $C[\ell, \tau, j, k]$. Thus, our recursion is defined by 4 parameters: ℓ is the index of the currently handled attribute, τ is the assignment to the $\ell - 1$ previously handled attributes, j is the number of values we are allowed to choose from for the currently handled attribute A_ℓ, and k is the number of tuples we need to keep.

According to the cost definition in Equation (1), our goal is to compute $F[1, \emptyset, q^\emptyset_1, k]$ for $k = \operatorname{argmin}_k C[1, \emptyset, q^\emptyset_1, k]$. We again focus on the computation of $C[\ell, \tau, j, k]$, as $F[\ell, \tau, j, k]$ can be obtained similarly with bookkeeping. The DP algorithm we propose is given by the following formula.

$$C[\ell, \tau, j, k] = \begin{cases} \sum_{f \in D^\tau \setminus \mathrm{top}(k, D^\tau)} w_f & \ell = n+1, \\ 0 & j = 0, k = 0, \\ \infty & j = 0, k > 0, \\ \min_{t \in \{0,\ldots,k\}} \Big(C[\ell, \tau, j-1, k-t] + t(k-t) w_\ell + & \text{otherwise.} \\ C[\ell+1, \tau \wedge (A_\ell = j), q_{\ell+1}^{\tau \wedge (A_\ell = j)}, t] \Big) \end{cases}$$

The first line (where $\ell = n+1$) refers to the case where the FD set is empty. Since no FDs need to be taken into account, the optimal subset of D^τ of size k consists of the k facts of the highest weight, and its cost is the sum of weights of all other facts. The next two lines (where $j = 0$) refer to the case where the database is empty. If we need an empty repair, this has no cost as we remove no tuples and violate no FDs. If we need a larger repair, this is not possible, and we treat the cost as infinite. The fourth line refers to the general case that does not fall within one of the previously-mentioned edge cases. In this case, we reduce the problem by determining how many tuples in the repair use a specific value for the currently handled attribute A_ℓ. More specifically, we determine the number of tuples t taken from the last block D_j^τ by going over all options for t and taking one with minimum cost. The cost has three components:

- The cost of repairing within D_j^τ with t tuples is $C[\ell+1, \tau \wedge (A_\ell = j), q_{\ell+1}^{\tau \wedge (A_\ell = j)}, t]$. Within D_j^τ, all tuples have $A_\ell = j$, and so it is enough to find an optimal repair for the set of FDs from which A_ℓ is removed.
- The cost of repairing within $D_1^\tau, \ldots, D_{j-1}^\tau$ with $k - t$ tuples is $C[\ell, \tau, j-1, k-t]$.
- The cost of violations between D_j^τ and $D_1^\tau \cup \ldots \cup D_{j-1}^\tau$ is $t(k-t)w_\ell$. Here we pay w_ℓ for every pair of facts from the two parts since every such pair is a violation of the FDs in which A_ℓ is a consensus attribute.

The given recursion can be computed in polynomial time via dynamic programming. Therefore, we conclude the following generalization of Theorem 4.3.

Theorem 4.5. *Let Δ be a set of FDs. If Δ can be emptied via L/C-Simplify(Δ) steps, then soft repairing for Δ is solvable in polynomial time.*

4.3 Limitations of the Algorithm

We now discuss the limitations of our dynamic programming approach. In particular, we explain the challenges in extending it to other simple FD sets, even those that are logically equivalent to a single FD. Consider the FD set $\Delta_1 = \{A \rightarrow BC\}$ over $R(A, B, C)$. In our algorithm for a single FD, we split the database into blocks (that is, partitions through grouping by A) and then, for each block, we go over its subblocks (i.e., partitions by BC) one by one. The main observation here is that when we consider a certain subblock, the facts we choose to keep from this subblock will be in conflict with *every* fact of the previous subblocks. This does not depend on the specific facts that we choose to keep from the current subblock or the previous ones. The cost of the new violations that we introduce also does not depend on the specific facts that we choose from the current subblock, but it depends only on the number of facts that we keep (and the weight of the FD).

Now, consider the FD set $\Delta_2 = \{A \rightarrow B, A \rightarrow C\}$ over $R(A, B, C)$. While Δ_1 and Δ_2 are equivalent as hard constraints, they are not equivalent as soft constraints, even if both of them have the same weight. (See Bárány et al. [4] for a more detailed discussion about the difference between the logical view of constraints and their interpretation as weighted factor graphs.) It is again the case that we can split the database into blocks of facts that agree on the value of attribute A and these blocks

A	B	C	
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1

Fig. 2. A database D over $R(A, B, C)$.

are independent. Now, let us again try to split each block into subblocks of facts that agree on the values of both attributes B and C. When we consider a certain subblock, the cost of the new violations that we introduce again does not depend on the specific facts that we choose to keep from this subblock, as they all agree on the values of both B and C and, so, they are involved in the same violations. However, the cost of these new violations does depend on the specific facts that we chose to keep from the previous subblocks, as two facts from different subblocks may violate only the FD $A \rightarrow B$, only the FD $A \rightarrow C$, or both FDs, and each one of these options entails a different cost. Hence, for the FD set Δ_2, the information regarding the number of facts from the previous subblocks (the value $k - t$ in the definition of $C[i, j, k]$) is insufficient. When we consider the jth subblock, we have to take into account every possible combination of values $\ell_1, \ldots, \ell_{j-1}$ that represent the number of facts we chose to keep from the previous $j - 1$ subblocks. In this case, the running time of the algorithm is no longer polynomial, but rather exponential.

The following example illustrates the differences between the FD set Δ_1 and Δ_2.

Example 4.6. Consider the simple database of Figure 2 (where the weight of each fact is one) and the FD set Δ_1. Assume that $w_{A \rightarrow BC} = 1$. This database consists of a single block with four subblocks: $D_1 = \{(0, 0, 0)\}$, $D_2 = \{(0, 0, 1)\}$, $D_3 = \{(0, 1, 0)\}$, and $D_4 = \{(0, 1, 1)\}$. Assume that, in the dynamic programming algorithm, we go over the subblocks in the order D_1, D_2, D_3, D_4. When we consider the last subblock D_4, if we decide to keep the only fact of this subblock, then we have to pay 1 for each fact of the previous subblocks that we decided to keep. This is because the fact $(0, 1, 1)$ violates the FD $A \rightarrow BC$ with every other fact of the database.

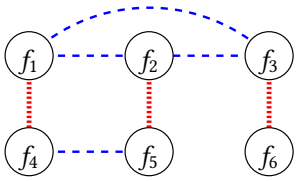
Now, consider the FD set Δ_2 and assume that $w_{A \rightarrow B} = w_{A \rightarrow C} = 1$. If we use the same approach, when we consider the subblock D_4, the additional cost that we have to pay for keeping the only fact of D_4 now depends on the specific choices we made for the previous subblocks. This holds since the cost of a violation between $(0, 1, 1)$ and $(0, 0, 1)$ is 1, as these two facts jointly violate the FD $A \rightarrow B$. The same holds for the violation between $(0, 1, 1)$ and $(0, 1, 0)$, as these two facts jointly violate the FD $A \rightarrow C$ that is associated with the same weight. But the cost of a violation between $(0, 1, 1)$ and $(0, 0, 0)$ is 2, as these facts violate both FDs. Hence, to compute the cost of the additional violations, we not only need to know the total number of facts we chose to keep from the previous subblocks, but the number of facts we chose to keep from each individual subblock; thus, we need to consider exponentially many options. □

5 ALGORITHM FOR MATCHING CONSTRAINTS

In this section, we describe a polynomial-time algorithm for the special case of bipartite matching where the schema is $R(A, B)$ and $\Delta \stackrel{\text{def}}{=} \{A \rightarrow B, B \rightarrow A\}$. Note that each of the two FDs has a separate weight. In Section 5.1, we extend the algorithm into the more general case of (what we refer to as) a *matching constraint*, where the FD set Δ states two keys that cover all of the attributes. (We give the precise definition in Section 5.1.)

	A	B
f_1	a_1	b_1
f_2	a_1	b_2
f_3	a_1	b_3
f_4	a_2	b_1
f_5	a_2	b_2
f_6	a_3	b_3

(a) Database



(b) Conflict graph

Fig. 3. A database over $R(A, B)$ and its conflict graph w.r.t. $\{A \rightarrow B, B \rightarrow A\}$.

In the remainder of this section, we assume the input D over $R(A, B)$. We begin with an observation. For $E \subseteq D$ it holds that:

$$\sum_{f \in (D \setminus E)} w_f = \sum_{f \in D} w_f - \sum_{f \in E} w_f$$

Since the value $\sum_{f \in D} w_f$ does not depend on the choice of E, minimizing the value $\left(\sum_{f \in (D \setminus E)} w_f\right) + \left(\sum_{\varphi \in \Delta} w_\varphi |\mathsf{vio}(E, \varphi)|\right)$ is the same as minimizing the value $\left(\sum_{f \in E} -w_f\right) + \left(\sum_{\varphi \in \Delta} w_\varphi |\mathsf{vio}(E, \varphi)|\right)$. We use the following notation:

$$w_D(E) = \left(\sum_{f \in E} -w_f\right) + \left(\sum_{\varphi \in \Delta} w_\varphi |\mathsf{vio}(E, \varphi)|\right)$$

To solve the problem, we construct a reduction to the *Minimum Cost Maximum Flow* (MCMF) problem. The input to MCMF is a flow network $\mathcal{N}$, that is, a directed graph (V, E) with a *source* node s having no incoming edges and a *sink* node t having no outgoing edges. Each edge $e \in E$ is associated with a *capacity* c_e and a *cost* $c(e)$. A flow f of $\mathcal{N}$ is a function $f : E \rightarrow \mathbb{R}$ such that $0 \leq f(e) \leq c_e$ for every $e \in E$, and moreover, for every node $v \in V \setminus \{s, t\}$ it holds that $\sum_{e \in I_v} f(e) = \sum_{e \in O_v} f(e)$ where I_v and O_v are the sets of incoming and outgoing edges of v, respectively. A *maximum flow* is a flow f that maximizes the value $\sum_{(s,v) \in E} f(s, v)$, and a *minimum cost maximum flow* is a maximum flow f with a minimal cost, where the cost of a flow is defined by $\sum_{e \in E} f(e) \cdot c(e)$. We say that f is *integral* if all values $f(e)$ are integers. It is known that, whenever the capacities are integral (i.e., natural numbers, as will be in our case), an integral minimum cost maximum flow exists and, moreover, can be found in polynomial time [1, Chapter 9].

From D we construct n instances $\mathcal{N}_1, \ldots, \mathcal{N}_n$ of the MCMF problem, where n is the number of facts in D, in the following way. First, we denote the FD $A \rightarrow B$ by φ_1 and the FD $B \rightarrow A$ by φ_2. We also denote by $D.A$ the set of values occurring in attribute A in D (that is, $D.A = \{\mathsf{a} \mid \exists f \in D(f[A] = \mathsf{a})\}$). We do the same for attribute B and denote by $D.B$ the set of values that occur in attribute B in D. For each value $\mathsf{a} \in D.A$ we denote by $\#_{D.A}(\mathsf{a})$ the number of appearances of the value a in attribute A (i.e., the number of facts $f \in D$ such that $f[A] = \mathsf{a}$). Similarly, we denote by $\#_{D.B}(\mathsf{b})$ the number of appearances of the value b in attribute B in D. Observe that

$$\mathsf{vio}(D, \varphi_1) = \frac{1}{2} \cdot \sum_{\mathsf{a} \in D.A} [\#_{D.A}(\mathsf{a}) \cdot (\#_{D.A}(\mathsf{a}) - 1)]$$

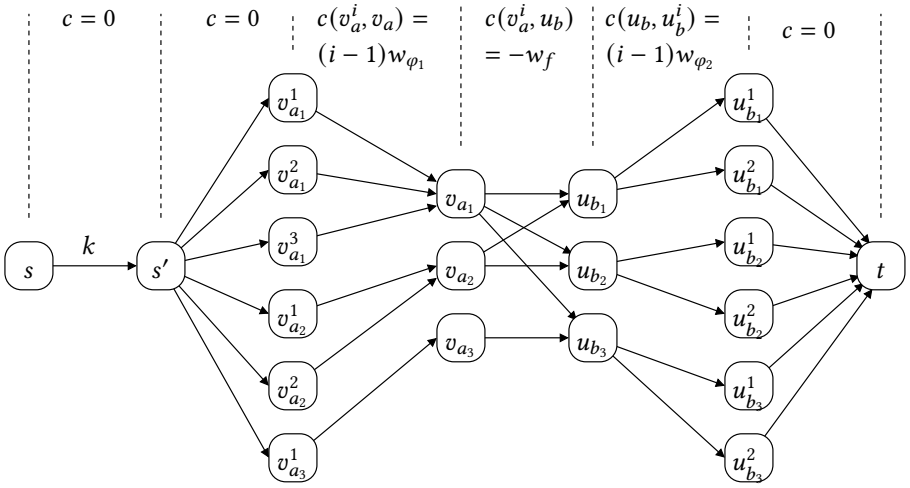


Fig. 4. The network $\mathcal{N}_k$ constructed from the database of Figure 3a. The capacity of all edges is 1, except for the edge (s, s') that has capacity k.

since every fact of the form $R(\mathsf{a}, \mathsf{b})$ violates φ_1 with every fact $R(\mathsf{a}, \mathsf{c})$ where $\mathsf{b} \neq \mathsf{c}$. Similarly, it holds that

$$\mathsf{vio}(D, \varphi_2) = \frac{1}{2} \cdot \sum_{\mathsf{b} \in D.B} [\#_{D.B}(\mathsf{b}) \cdot (\#_{D.B}(\mathsf{b}) - 1)]$$

Next, we describe the construction of the network $\mathcal{N}_k$. Our construction for the database of Figure 3a is illustrated in Figure 4. Note that Figure 3b depicts the conflict graph of the database of Figure 3a w.r.t. $\Delta = \{A \to B, B \to A\}$, which contains a vertex for each fact in the database and an edge between two vertices if the corresponding facts jointly violate an FD of Δ. The blue edges in the conflict graph are violations of the FD $A \to B$ and the red edges are violations of the FD $B \to A$.

For each $k \in \{1, \ldots, n\}$ we construct the network $\mathcal{N}_k$ that consists of the set $\{s, s', t\} \cup V \cup A \cup B \cup U$ of nodes where:

- $A = \{v_\mathsf{a} \mid \mathsf{a} \in D.A\}$
- $B = \{u_\mathsf{b} \mid \mathsf{b} \in D.B\}$
- $V = \{v^i_\mathsf{a} \mid \mathsf{a} \in D.A, 1 \leq i \leq \#_{D.A}(\mathsf{a})\}$
- $U = \{u^i_\mathsf{b} \mid \mathsf{b} \in D.B, 1 \leq i \leq \#_{D.B}(\mathsf{b})\}$

$\mathcal{N}_k$ contains the following edges:

- (s, s'), with cost $c(s, s') = 0$
- (s', v^i_a) for every $v^i_\mathsf{a} \in V$, with cost $c(s', v^i_\mathsf{a}) = 0$
- $(v^i_\mathsf{a}, v_\mathsf{a})$ for every value $\mathsf{a} \in D$, with cost $c(v^i_\mathsf{a}, v_\mathsf{a}) = (i-1) \cdot w_{\varphi_1}$
- $(v_\mathsf{a}, u_\mathsf{b})$ for every $\mathsf{a} \in D.A$ and $\mathsf{b} \in D.B$ such that $f = R(\mathsf{a}, \mathsf{b})$ occurs in D, with cost $c(v_\mathsf{a}, u_\mathsf{b}) = -w_f$
- $(u_\mathsf{b}, u^i_\mathsf{b})$ for every value $\mathsf{b} \in D$, with cost $c(u_\mathsf{b}, u^i_\mathsf{b}) = (i-1) \cdot w_{\varphi_2}$
- (u^i_b, t) for every $u^i_\mathsf{b} \in U$, with cost $c(u^i_\mathsf{b}, t) = 0$

The capacity of the edge (s, s') is k and the capacity of the other edges is 1. The intuition for the construction is as follows. A network with edges of the form $(v_\mathsf{a}, u_\mathsf{b})$ that are connected to a source on one side and a target on the other corresponds to a matching, which in turn corresponds to a traditional repair. To allow violations of $A \to B$, we add the vertices v^i_a. The cost of a violation

of this FD is defined by the cost of the edges $(v^i_{\mathsf{a}}, v_{\mathsf{a}})$. In particular, if we keep k facts of the form $R(\mathsf{a}, \cdot)$ for some $\mathsf{a} \in D.A$ we pay $\sum_{i=1}^{k}(i-1)w_{\varphi_1}$ for violations of φ_1. We include the vertices u^i_{b} to similarly allow violations of $B \rightarrow A$. The discarding of facts is discouraged by offering gain for the edges $(v_{\mathsf{a}}, u_{\mathsf{b}})$. Finally, to prevent the case where the flow always fills the entire network (which corresponds to taking all facts and paying for all violations), we introduce the edge (s, s') which limits the capacity of the network, and enables us to find the minimum cost flow of a given size k. We will show that for every k, the cost of the solution to the MCMF problem on $\mathcal{N}_k$ will be the cost of the "cheapest" subinstance of D of size k. Hence, the solution to our problem is the cost of the minimal solution among all the instances $\mathcal{N}_1, \ldots, \mathcal{N}_n$.

Given an integral flow f in $\mathcal{N}_k$, the repair $D[f]$ induced by f, is the set of facts $R(\mathsf{a}, \mathsf{b})$ corresponding to edges of the form $(v_{\mathsf{a}}, u_{\mathsf{b}})$ such that $f(v_{\mathsf{a}}, u_{\mathsf{b}}) = 1$. Moreover, given a subinstance E of D of size k, we denote by f_E the integral flow in $\mathcal{N}_k$ defined as follows.

- $f_E(s, s') = k$
- $f_E(s', v^i_{\mathsf{a}}) = 1$ for $1 \leq i \leq \#E.A(\mathsf{a})$ and $f_E(s', v^i_{\mathsf{a}}) = 0$ for $i > \#E.A(\mathsf{a})$ for every $\mathsf{a} \in E.A$
- $f_E(v^i_{\mathsf{a}}, v_{\mathsf{a}}) = 1$ for $1 \leq i \leq \#E.A(\mathsf{a})$ and $f_E(v^i_{\mathsf{a}}, v_{\mathsf{a}}) = 0$ for $i > \#E.A(\mathsf{a})$ for every $\mathsf{a} \in E.A$
- $f_E(v_{\mathsf{a}}, u_{\mathsf{b}}) = 1$ if $R(\mathsf{a}, \mathsf{b}) \in E$ and $f_E(v_{\mathsf{a}}, u_{\mathsf{b}}) = 0$ otherwise
- $f_E(u_{\mathsf{b}}, u^i_{\mathsf{b}}) = 1$ for $1 \leq i \leq \#E.B(\mathsf{b})$ and $f_E(u_{\mathsf{b}}, u^i_{\mathsf{b}}) = 0$ for $i > \#E.B(\mathsf{b})$ for every $\mathsf{b} \in E.B$
- $f_E(u^i_{\mathsf{b}}, t) = 1$ for $1 \leq i \leq \#E.B(\mathsf{b})$ and $f_E(u^i_{\mathsf{b}}, t) = 0$ for $i > \#E.B(\mathsf{b})$ for every $\mathsf{b} \in E.B$

The reader can easily verify that f_E is indeed an integral flow in $\mathcal{N}_k$. Clearly, the value of the flow is k.

We have the following lemma.

Lemma 5.1. *Every integral solution f to MCMF on $\mathcal{N}_k$ satisfies* $\mathrm{cost}(f) = w_D(f[D])$.

Proof. First, note that it cannot be the case that $f(s', v^j_{\mathsf{a}}) = 0$ while $f(s', v^i_{\mathsf{a}}) = 1$ for some $j < i$ and $i \in \{1, \ldots, \#_{D.A}(\mathsf{a})\}$. Otherwise, we can construct a different integral flow f' with $f'(s', v^j_{\mathsf{a}}) = f'(v^j_{\mathsf{a}}, v_{\mathsf{a}}) = 1$, $f'(s', v^i_{\mathsf{a}}) = f'(v^i_{\mathsf{a}}, v_{\mathsf{a}}) = 0$, and $f'(e) = f(e)$ for every other edge e. It holds that $\mathrm{cost}(f') = \mathrm{cost}(f) - c(v^i_{\mathsf{a}}, v_{\mathsf{a}}) + c(v^j_{\mathsf{a}}, v_{\mathsf{a}})$, and since $c(v^i_{\mathsf{a}}, v_{\mathsf{a}}) > c(v^j_{\mathsf{a}}, v_{\mathsf{a}})$ we will have that $\mathrm{cost}(f') < \mathrm{cost}(f)$ in contradiction to the fact that f is a solution to MCMF on $\mathcal{N}_k$. Therefore, for every $\mathsf{a} \in D.A$, if the flow entering the node v_{a} is ℓ, then $f(s', v^i_{\mathsf{a}}) = f(v^i_{\mathsf{a}}, v_{\mathsf{a}}) = 1$ if $i \leq \ell$ and $f(s', v^i_{\mathsf{a}}) = f(v^i_{\mathsf{a}}, v_{\mathsf{a}}) = 0$ otherwise. Thus, the total cost of the edges of the form $(v^i_{\mathsf{a}}, v_{\mathsf{a}})$ is $\sum_{i=1}^{\ell}\left[(i-1)w_{\varphi_1}\right] = \frac{1}{2}\ell(\ell-1)w_{\varphi_1}$. By the definition of $f[D]$, there are $\#_{f[D].A}(\mathsf{a})$ edges of the form $(v_{\mathsf{a}}, u_{\mathsf{b}})$ for which $f(v_{\mathsf{a}}, u_{\mathsf{b}}) = 1$. By the definition of a flow, this is also the flow entering the node v_{a}, and we have that $\ell = \#_{f[D].A}(\mathsf{a})$. We conclude that the total cost of the flow on edges of the form $(v^i_{\mathsf{a}}, v_{\mathsf{a}})$ is

$$\sum_{\mathsf{a} \in f[D].A}\left[\frac{1}{2} \cdot \#_{f[D].A}(\mathsf{a}) \cdot (\#_{f[D].A}(\mathsf{a}) - 1) \cdot w_{\varphi_1}\right] = \mathsf{vio}(f[D], \varphi_1) \cdot w_{\varphi_1}\,.$$

The same argument shows that the total cost of the flow on edges of the form $(u_{\mathsf{b}}, u^i_{\mathsf{b}})$ is $\mathsf{vio}(f[D], \varphi_2) \cdot w_{\varphi_2}$.

Finally, the total cost of the edges of the form $(v_{\mathsf{a}}, u_{\mathsf{b}})$ is $\sum_{g \in f[D]}(-w_g)$ by the definition of $f[D]$ and the construction of the network. We conclude that:

$$\mathrm{cost}(f) = \left(\sum_{g \in f[D]}(-w_g)\right) + \mathsf{vio}(f[D], \varphi_1) \cdot w_{\varphi_1} + \mathsf{vio}(f[D], \varphi_2) \cdot w_{\varphi_2}$$

and $\mathrm{cost}(f) = w_D(f[D])$ by definition. □

The next lemma follows straightforwardly from the construction of $\mathcal{N}_k$ and the definition of f_E.

Lemma 5.2. *Every subinstance E of D satisfies* $\text{cost}(f_E) = w_D(E)$.

Now, let E be an optimal subset of D w.r.t. Δ and assume that $|E| = k$. Let f^* be a solution with the minimum cost among all the solutions to MCMF on $\mathcal{N}_1 \ldots, \mathcal{N}_n$. Lemma 5.2 implies that there is an integral flow f_E in $\mathcal{N}_k$ such that $\text{cost}(f_E) = w_D(E)$. Hence, we have that $\text{cost}(f^*) \leq w_D(E)$. By applying Lemma 5.1 on f^*, there is another subinstance E' of D such that $w_D(E') = \text{cost}(f^*)$. Since E is an optimal subset, we have that $w_D(E) \leq w_D(E')$. Overall, we have that $\text{cost}(f^*) \leq w_D(E) \leq w_D(E') = \text{cost}(f^*)$, and we conclude that $\text{cost}(f^*) = w_D(E)$. Therefore, by taking the solution with the lowest cost among all solutions to MCMF on $\mathcal{N}_1, \ldots, \mathcal{N}_n$, we indeed find a solution to our problem, and that concludes the description of our algorithm.

Example 5.3. Consider again the database of Figure 3a. Assume that:

$$w_{\varphi_1} = w_{\varphi_2} = 2 \qquad w_{f_1} = w_{f_2} = w_{f_3} = w_{f_4} = w_{f_5} = w_{f_6} = 1$$

Since the cost of a violation is "too high" in this case (i.e., it is always cheaper to delete a fact involved in a violation than to keep the violation), an optimal subset in this case is, in fact, an optimal repair in the traditional sense (that is, when the constraints are assumed to be hard constraints). One possible optimal repair in this case is $\{f_2, f_4, f_6\}$. The flow corresponding to this repair in the network $\mathcal{N}_3$ is illustrated in Figure 5a.

Now, assume that:

$$w_{\varphi_1} = w_{\varphi_2} = 1 \qquad w_{f_1} = w_{f_2} = w_{f_3} = w_{f_4} = w_{f_5} = w_{f_6} = 3$$

In this case, the cost of deleting a fact is "too high", since each fact is involved in at most two violations, and the cost of keeping the violation is lower than the cost of removing facts involved in the violation. Therefore, the database itself is an optimal subset, and the corresponding flow in the network $\mathcal{N}_6$ is illustrated in Figure 5b.

As another example, assume that:

$$w_{\varphi_1} = w_{\varphi_2} = 1 \qquad w_{f_1} = w_{f_2} = w_{f_5} = 2, w_{f_3} = w_{f_4} = 1, w_{f_6} = 3$$

Here an optimal subset consists of the facts in $\{f_1, f_2, f_5, f_6\}$, and the corresponding flow in the network $\mathcal{N}_4$ is illustrated in Figure 5c. If we modify the weight of φ_2 and define $w_{\varphi_2} = 4$, while keeping the rest of the weight intact, it is now cheaper to delete the fact f_2 rather than keep the violations it is involved in with f_1 and f_5; hence, an optimal subset in this case is $\{f_1, f_5, f_6\}$, and the corresponding flow in the network $\mathcal{N}_3$ is illustrated in Figure 5d. □

We therefore establish the following.

Theorem 5.4. *Soft repairing is solvable in polynomial time for $R(A, B)$ and $\Delta = \{A \to B, B \to A\}$.*

Note that the FD set $\{A \to B\}$ over $R(A, B)$ is in fact a special case of the result of Theorem 5.7, as we can compute an optimal subset for this FD set using the algorithm described above by defining $w_{B \to A} = 0$. However, this algorithm works only for the case where the single FD is a key and fails to compute the correct solution when the schema contains attributes that do not appear in the FD. The algorithm described in the proof of Theorem 4.3, on the other hand, can handle this case and does not assume anything about the underlying schema.

5.1 Generalization

We now extend our algorithm beyond bipartite matching to the more general case of a matching constraint. By a "matching constraint" we refer to the case of $\hat{\Delta} = \{X \to Y, X' \to Y'\}$ over a schema $\hat{R}(A_1, \ldots, A_k)$ where $X \cup Y = X' \cup Y' = X \cup X' = \{A_1, \ldots, A_k\}$. An example follows.

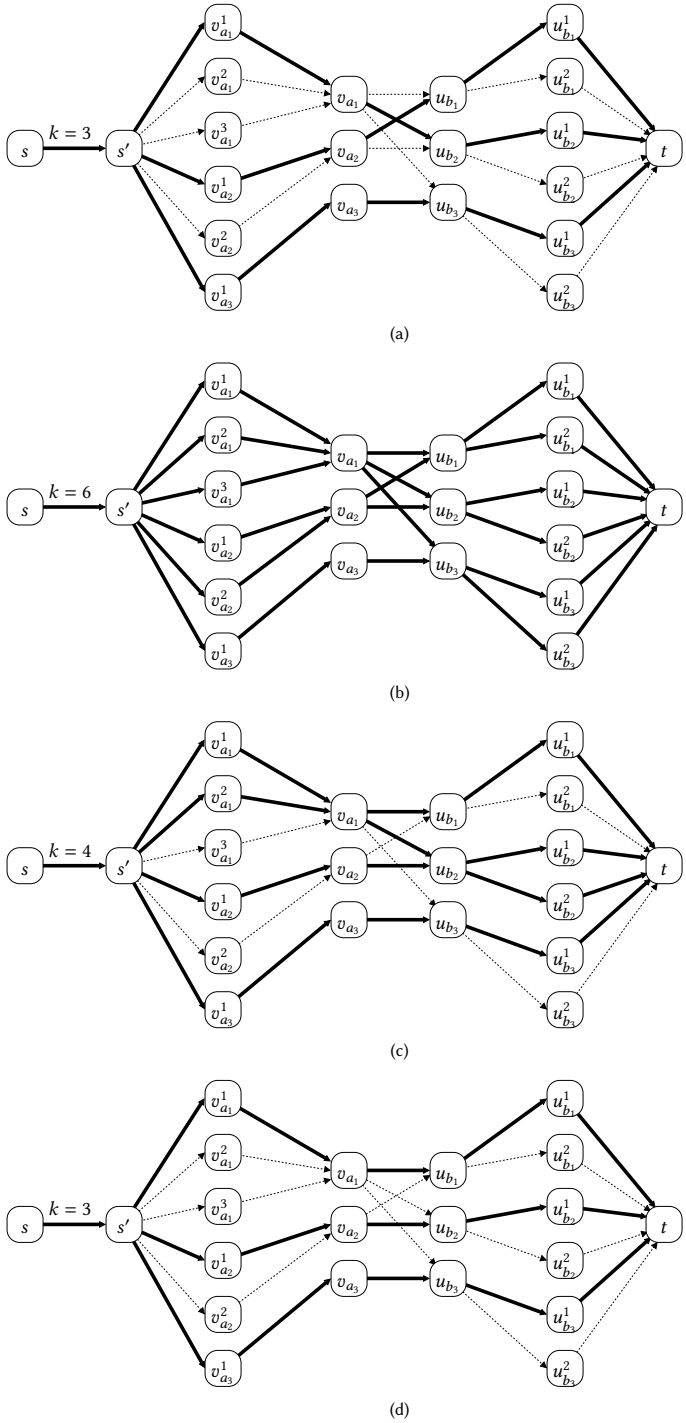


Fig. 5. The flow in the network $\mathcal{N}_k$ corresponding to an optimal subset of the database of Figure 3a for different weights.

Example 5.5. Consider the database of our running example (Figure 1), and the following FDs:

- Flight Airline Date → Origin Destination Airplane,
- Origin Destination Airplane Date → Flight Airline.

The reader can easily verify that these two FDs form a matching constraint. On the other hand, consider the set consisting of the following two FDs:

- Flight Date → Airline Origin Destination Airplane,
- Origin Destination Airplane Date → Flight Airline.

Here, we do not have a matching constraint since while it holds that

$$X \cup Y = X' \cup Y' = \{\text{Flight, Airline, Date, Origin, Destination, Airplane}\},$$

the set $X \cup X'$ misses the Airline attribute. □

The generalization of Theorem 5.4 from $\Delta = \{A \to B, B \to A\}$ over $R(A, B)$ to the general case of a matching constraint is fairly straightforward. Given an input $\hat{D}$ for soft repairing over $\hat{R}$ and $\hat{\Delta}$, we construct an input D over R and Δ by defining unique values $a(\pi_X(\hat{f}))$ and $b(\pi_{X'}(\hat{f}))$ for the projections $\pi_X(\hat{f})$ and $\pi_{X'}(\hat{f})$ over X and X', respectively, of every fact $\hat{f}$ of $\hat{D}$. Then, the database D is simply the set of all the pairs $a(\pi_X\hat{f})$ and $b(\pi_{X'}\hat{f})$ for all facts $\hat{f}$ of D:

$$D \stackrel{\text{def}}{=} \{(a(\pi_X\hat{f}), b(\pi_{X'}\hat{f})) \mid \hat{f} \in \hat{D}\}$$

In addition, we define $w_f \stackrel{\text{def}}{=} w_{\hat{f}}$ whenever $f = (a(\pi_X\hat{f}), b(\pi_{X'}\hat{f}))$ and $w_{A\to B} \stackrel{\text{def}}{=} w_{X\to Y}$ and $w_{B\to A} \stackrel{\text{def}}{=} w_{X'\to Y'}$. Note that the mapping $f \to \hat{f}$ is reversible since $X \cup X' = \{A_1, \ldots, A_k\}$. So, in order to solve soft repairing for $\hat{D}$, we solve it for D and transform every fact f of D into the corresponding fact $\hat{f}$ of $\hat{D}$.

Example 5.6. Consider the matching constraint from Example 5.5. Figure 6 demonstrates the reduction in this case.

We get the following result. The proof is by showing the correctness of the reduction.

Theorem 5.7. *Soft repairing is solvable in polynomial time whenever Δ is a pair of FDs that constitutes a matching constraint.*

Proof. We prove that D has a subset E with $\text{cost}(E \mid D) = k$ if and only if $\hat{D}$ has a subset $\hat{E}$ with $\text{cost}(\hat{E} \mid \hat{D}) = k$. Let E be a subset of D with cost k. Let $\hat{E}$ be a subset of $\hat{D}$ that includes the fact $\hat{f}$ for every $f \in E$. By definition, we have that $\sum_{f\in(D\setminus E)} w_f = \sum_{f\in(\hat{D}\setminus\hat{E})} w_{\hat{f}}$; hence, it is left to show that $\sum_{\varphi\in\Delta} w_\varphi |\text{vio}(E, \varphi)| = \sum_{\hat{\varphi}\in\hat{\Delta}} w_{\hat{\varphi}} |\text{vio}(\hat{E}, \hat{\varphi})|$. Let $f, g \in E$ such that $\{f, g\} \not\models (A \to B)$. Hence, it holds that $f[A] = g[A]$ while $f[B] \neq g[B]$. From the construction of D, we have that $\pi_X\hat{f} = \pi_X\hat{g}$, while $\pi_{X'}\hat{f} \neq \pi_{X'}\hat{g}$. Thus, there is an attribute $A_i \in X'$ such that $\hat{f}[A_i] \neq \hat{g}[A_i]$ and since $A_i \notin X$ and $X \cup Y = \{A_1, \ldots, A_k\}$, it holds that $A_i \in Y$. We conclude that $\{\hat{f}, \hat{g}\} \not\models (X \to Y)$. We can similarly prove that if $\{f, g\} \not\models (B \to A)$, then $\{\hat{f}, \hat{g}\} \not\models (X' \to Y')$. Finally, because $w_{A\to B} = w_{X\to Y}$ and $w_{B\to A} = w_{X'\to Y'}$ it holds that $\sum_{\varphi\in\Delta} w_\varphi |\text{vio}(E, \varphi)| = \sum_{\hat{\varphi}\in\hat{\Delta}} w_{\hat{\varphi}} |\text{vio}(\hat{E}, \hat{\varphi})|$.

For the other direction, let $\hat{E}$ be a subset of $\hat{D}$, and let E be the subset of D that includes the fact f for every $\hat{f} \in \hat{E}$. It is again straightforward that $\sum_{f\in(D\setminus E)} w_f = \sum_{f\in(\hat{D}\setminus\hat{E})} w_{\hat{f}}$. Now, let $\hat{f}, \hat{g} \in \hat{E}$ such that $\{\hat{f}, \hat{g}\} \not\models (X \to Y)$. We have that $\hat{f}[A_i] = \hat{g}[A_i]$ for every $A_i \in X$; thus, $\pi_X\hat{f} = \pi_X\hat{g}$ and from the construction of D, it holds that $f[A] = g[A]$. On the other hand, the fact that $\hat{f}[A_i] \neq \hat{g}[A_i]$ for some $A_i \in Y$ together with the fact that $X \cup Y = X \cup X' = \{A_1, \ldots, A_k\}$ imply that $\pi_{X'}\hat{f} \neq \pi_{X'}\hat{g}$ and $f[B] \neq g[B]$. Hence, $\{f, g\} \not\models (A \to B)$. We can similarly prove that if $\{\hat{f}, \hat{g}\} \not\models (X' \to Y')$,

Flights

Flight	Airline	Date	Origin	Destination	Airplane	
UA123	United Airlines	01/01/2021	LA	NY	N652NW	3
UA123	United Airlines	01/01/2021	NY	UT	N652NW	2
UA123	Delta	01/01/2021	LA	NY	N652NW	1
DL456	Southwest	02/01/2021	NC	MA	N713DX	2
DL456	Southwest	03/01/2021	NJ	FL	N245DX	1
DL456	Delta	03/01/2021	CA	IL	N819US	4

(a) $\hat{D}$

⇓

Flights

(Flight, Airline, Date)	(Origin, Destination, Airplane, Date)	
a(UA123, United Airlines, 01/01/2021)	b(LA, NY, N652NW, 01/01/2021)	3
a(UA123, United Airlines, 01/01/2021)	b(NY, UT, N652NW, 01/01/2021)	2
a(UA123, Delta, 01/01/2021)	b(LA, NY, N652NW, 01/01/2021)	1
a(DL456, Southwest, 02/01/2021)	b(NC, MA, N713DX, 02/01/2021)	2
a(DL456, Southwest, 03/01/2021)	b(NJ, FL, N245DX, 03/01/2021)	1
a(DL456, Delta, 03/01/2021)	b(CA, IL, N819US, 03/01/2021)	4

(b) D

Fig. 6. A reduction to a bipartite matching from the matching constraint Flight Airline Date → Origin Destination Airplane and Origin Destination Airplane Date → Flight Airline.

then $\{f, g\} \not\models (B \to A)$, which again implies that $\sum_{\varphi\in\Delta} w_\varphi |\mathrm{vio}(E,\varphi)| = \sum_{\hat{\varphi}\in\hat{\Delta}} w_{\hat{\varphi}} |\mathrm{vio}(\hat{E},\hat{\varphi})|$, and the concludes our proof. □

The extension of the algorithm from bipartite matching $\{A \to B, B \to A\}$ to general matching constraints raises the question of whether the algorithm could be naturally extended to a more general class of FD sets. This problem is left as a challenge for future research.

6 PROBABILISTIC CHALLENGES

We now discuss the reasoning about soft constraints from a view broader than the discussion thus far. Recall from the Introduction that, in the bigger context, the semantics we adopt is that of a parametric factor graph, where the probability of a possible world is the product of *factors*. In turn, we get a factor from every violation of a constraint and from every deletion of a fact. With that perspective in mind, the problem of soft repairing (Problem 2.1) is one kind of a fundamental task in probabilistic inference, and there are several others that should be studied.

To be more precise, we adopt the convention (e.g., [32–34, 37]) that the weight is converted into a multiplicative factor, typically through the exponent function. In particular, a violation of constraint φ contributes to the product the factor $\exp(-w_\varphi)$, and a deleted fact f contributes the factor $\exp(-w_f)$. In the sequel, we will denote the factors by F_φ and F_f, respectively and assume that they are strictly positive. The probability of a possible world is proportional to the product of its factors, and then normalization is required for this product to define a true probability space.

To be more precise, each subset E of D is viewed as a possible world (i.e., a deterministic sample database) with the *unnormalized probability*

$$\mathcal{U}_D[E] \stackrel{\text{def}}{=} \prod_{f\in D\setminus E} F_f \cdot \prod_{\varphi\in\Delta} F_\varphi^{|\mathrm{vio}(E,\varphi)|}. \tag{2}$$

Recall that $\mathrm{vio}(D, \varphi)$ is the set of all the violations of the FD φ in D. For example, in the typical conversion of weights into factors we have $\mathcal{U}_D[E] = e^{-\mathrm{cost}(E|D)}$ where $\mathrm{cost}(E \mid D)$ is the cost function defined in Equation (1). The probability space is then defined by straightforwardly normalizing:

$$\Pr_D[E] \stackrel{\text{def}}{=} \mathcal{U}_D[E]/Z(D)$$

where $Z(D)$ is the *partition function* (or *normalization constant*) defined naturally as

$$Z(D) \stackrel{\text{def}}{=} \sum_{E \subseteq D} \mathcal{U}_D[E]\,. \tag{3}$$

6.1 Computational Challenges

With the above probability space defined, the problem of soft repairing is simply that of finding a world E with a maximal probability — a problem known in computational statistics as *Maximum a Posteriori* (MAP) inference or *Most Probable Explanation* (MPE) inference, and in the context of database cleaning as that of computing the *most likely intention* [33], the *most probable world* [34], or the *most probable database* [16]. Some other inference problems that arise from this viewpoint follow.

Problem 6.1 (Marginal Inference). *Let $R(A_1, \ldots, A_k)$ be a relation schema, Δ a set of FDs, and Q a Boolean query over $R(A_1, \ldots, A_k)$.* Marginal inference *(*of Q for $R(A_1, \ldots, A_k)$ and Δ*) is the following computational problem: Given a database D, compute the probability*

$$\Pr_D[Q] \stackrel{def}{=} \sum_{\substack{E \subseteq D \\ E \models Q}} \Pr_D[E]\,.$$

One straightforward approach to solving marginal inference approximately, with error guarantees, is the Monte Carlo sampling: evaluate the query over samples from the distribution, and take the average over all samples. Yet, for that we need to have a sampling procedure, where the probability of each E is the same as (precisely or approximately) its probability $\Pr_D[E]$ in the true distribution. This problem is often challenging in probability spaces defined via factor graphs (known as *discriminative* rather than *generative*).

Problem 6.2 (Sampling). *Let $R(A_1, \ldots, A_k)$ be a relation schema and Δ a set of FDs.* Sampling *(*for $R(A_1, \ldots, A_k)$ and Δ*) is the following computational problem: Given a database D, generate a random subset of D, so that the probability $p(E)$ of generating every $E \subseteq D$ is equal to $\Pr_D[E]$.*

As implied by the definition of the probability in Equation (2), one may need to compute the partition function $Z(D)$ in order to conduct marginal inference and sampling. In general, this need arises in other tasks such as MPE and weight learning [12, 27]. Hence, we formally define this task as well.

Problem 6.3 (Partition-Function Computation). *Let $R(A_1, \ldots, A_k)$ be a relation schema and Δ a set of FDs.* Partition-function computation *(*for $R(A_1, \ldots, A_k)$ and Δ*) is the following computational problem: Given a database D, compute $Z(D)$.*

Remark 6.4. *We again note that extending the problems discussed here to multiple relations is straightforward. For example, sampling a possible world entails independent sampling from each relation, and the partition function of the database is the product of the partition functions of the individual relations. In the case of marginal inference, we have looked at queries over a single relation, so other relations have no impact on the outcome. Yet, a nontrivial question is regarding cross-relation queries, such as conjunctive queries, where the problem inherits the computational hardness featured by tuple-independent probabilistic databases [10, 11].*

6.1.1 Numerical Representation. For the sake of formal complexity analysis, in the remainder of this section we will assume that the factors F_f and F_φ are given directly as input rather than represented through the weights w_f and w_φ, respectively. The importance of this assumption is that the conversion from a weight w to the factor e^{-w} requires the application of the exponent function, and then we immediately encounter the problem of numerical imprecision since the cost becomes irrational even if the weight is rational. The implication would be that every upper and lower bound would require to involve either approximation or an alternative (symbolic) representation of numbers.[3] Hence, we assume that each fact f is given along with its factor $F_f > 0$, and each FD φ is given along with its factor $F_\varphi > 0$; each factor is represented as a pair of positive integers in the usual binary system, where the first and second numbers stand for the numerator and the denominator, respectively, of a rational number.

6.2 Algorithms for a Single FD

We show that for a set Δ that consists of a single FD φ, all three problem have a polynomial time solution.

6.2.1 Computing the Partition Function. We start by showing that the partition function $Z(D)$ can be computed in polynomial time, given a database D. As in the case of soft repairing for a single FD, we split D into blocks $D_1, \ldots, D_n$. As there are no conflicts across blocks, it is rather straightforward that:

$$Z(D) = \prod_{i=1}^{n} Z(D_i)$$

Hence, we focus on the computation of the partition function for a single block that we denote by D'. Let $D'_1, \ldots, D'_m$ be the subblocks of D' w.r.t. φ, according to some arbitrary order.

Consider a block D' of size ℓ. We number the facts in the block by $1, \ldots, \ell$ such that the facts of each subblock are consecutive. We denote by $T[j, k, t]$ the value of the partition function when restricted to subsets E that are contained in $\{f_1, \ldots, f_j\}$, and where the facts of E are additionally penalized for any violations they have with t additional facts in the subblock of f_j and k additional facts in subblocks that occur after the subblock of f_j. Our goal is then to compute $T[\ell, 0, 0]$.

We denote by $\mathsf{SB}(f)$ the subblock that contains the fact f. The computation in each case separates to two disjoint cases: *(1)* we remove the fact f_j and *(2)* we keep this fact.

$$T[j,k,t] = \begin{cases} F_{f_j} + (F_\varphi)^k & j = 1; \\ F_{f_j} \cdot T[j-1,k,t] + (F_\varphi)^k \cdot T[j-1,k,t+1] & j > 1 \text{ and } \mathsf{SB}(f_{j-1}) = \mathsf{SB}(f_j); \\ F_{f_j} \cdot T[j-1,k+t,0] + (F_\varphi)^k \cdot T[j-1,k+t+1,0] & j > 1 \text{ and } \mathsf{SB}(f_{j-1}) \neq \mathsf{SB}(f_j). \end{cases}$$

In the base case, $j = 1$, meaning there is only one fact. We either remove the fact f_1, which contributes the factor F_{f_1}, or we keep this fact and pay for k violations among f_1 and the facts of $D'_2, \ldots, D'_m$. We next describe the case of $j > 1$.

In case we remove f_j (corresponding to the first summand of all cases), the deleted fact contributes the factor F_{f_j}. If f_{j-1} and f_j belong to the same subblock D'_i, this factor is then multiplied by $T[j-1, k, t]$ because a subset E of $\{f_1, \ldots, f_j\}$ that does not contain f_j and where the facts of E are penalized for violations with t additional facts of D'_i and k additional facts of $D'_{i+1}, \ldots, D'_m$ is in fact a subset of $\{f_1, \ldots, f_{j-1}\}$ that satisfies the same property. If, on the other hand, f_{j-1} belongs to the subblock D'_{i-1} while f_j belongs to the subblock D'_i, then we multiply the factor F_{f_j}

[3]Note that this problem did not arise in the previous sections, since there we were interested in the sum of weights and did not need to transform them into factors.

by $T[j-1, k+t, 0]$ since a subset E of $\{f_1, \ldots, f_j\}$ that does not contain f_j and where the facts of E are penalized for violations with t additional facts of D'_i and k additional facts of $D'_{i+1}, \ldots, D'_m$ is a subset of $\{f_1, \ldots, f_{j-1}\}$ which facts are penalized for violations with $k+t$ additional facts of $D'_i, \ldots, D'_m$.

In case we keep f_j (corresponding to the second summand of all cases), we have k additional violations between f_j and the additional facts of the subblocks occuring after the subblock of f_j; these violations contribute the factor $(F_\varphi)^k$. This factor is then multiplied by $T[j-1, k, t+1]$ if f_{j-1} and f_j belong to the same subblock D'_i, because a subset E of $\{f_1, \ldots, f_j\}$ that contains f_j and where the facts of E are penalized for violations with t additional facts of D'_i and k additional facts of $D'_{i+1}, \ldots, D'_m$ consists of the fact f_j and a subset E' of $\{f_1, \ldots, f_{j-1}\}$ where the facts of E' are penalized for violations with $t+1$ additional facts of D'_i (including f_j) and k additional facts of $D'_{i+1}, \ldots, D'_m$. If f_{j-1} belongs to the subblock D'_{i-1} while f_j belongs to the subblock D'_i, then we multiply the factor $(F_\varphi)^k$ by $T[j-1, k+t+1, 0]$ since a subset E of $\{f_1, \ldots, f_j\}$ that contains f_j and where the facts of E are penalized for violations with t additional facts of D'_i and k additional facts of $D'_{i+1}, \ldots, D'_m$ consists of f_j and a subset E' of $\{f_1, \ldots, f_{j-1}\}$ whose facts are penalized for violations with $k+t+1$ additional facts of $D'_i, \ldots, D'_m$ (including the fact f_j).

This completes the description of the algorithm. Hence, we get the following.

Theorem 6.5. *In the case of a single FD, the partition function can be computed in polynomial time.*

Alternative Algorithm. We now present an alternative algorithm for computing the partition function in polynomial time in the case of a single FD. While the previous algorithm is simpler, the upcoming algorithm has the advantage that it is naturally extensible to the more general case where the set of FDs can be emptied via L/C-Simplify(Δ), as we establish later in Section 6.3.

As a first step, we compute the following value for every subblock D'_i of the block D':

$$P[D'_i, k] \stackrel{\text{def}}{=} \sum_{\substack{E \subseteq D'_i \\ |E|=k}} \mathcal{U}[E \mid D'_i]\,.$$

That is, $P[D'_i, k]$ is the contribution of all the subsets of D'_i of size k to $Z(D'_i)$.

Let $f_1, \ldots, f_r$ be an arbitrary order over the facts of D'_i. For $\ell \in \{1, \ldots, r\}$ we denote by $P[D'_i, k, \ell]$ the value of the partition function when restricted to subsets E of D'_i that such that $E \subseteq \{f_1, \ldots, f_\ell\}$ and $|E| = k$. Formally,

$$P[D'_i, k, \ell] \stackrel{\text{def}}{=} \sum_{\substack{E \subseteq \{f_1, \ldots, f_\ell\} \\ |E|=k}} \mathcal{U}[E \mid \{f_1, \ldots, f_\ell\}]\,.$$

Note that the cost is with respect to the part of the database that is considered up to this point in the dynamic programming, and so, we use $\mathcal{U}[E \mid \{f_1, \ldots, f_\ell\}]$. We then have that $P[D'_i, k] = P[D'_i, k, r]$.

We now show how to compute the value $P[D'_i, k, \ell]$ using dynamic programming.

$$P[D'_i, k, \ell] = \begin{cases} 0 & \text{if } \ell = 1 \text{ and } k \notin \{0, 1\}; \\ F_{f_1} & \text{if } \ell = 1 \text{ and } k = 0; \\ 1 & \text{if } \ell = 1 \text{ and } k = 1; \\ P[D'_i, k, \ell-1] \cdot F_{f_\ell} + P[D'_i, k-1, \ell-1] & \text{otherwise.} \end{cases}$$

For $\ell = 1$ (i.e., when we consider only the first fact), we have a single world of cardinality zero (i.e., $\emptyset$) with the factor F_{f_1}, a single world of cardinality one (i.e., $\{f_1\}$) with the factor 1, and no worlds of cardinality $k \notin \{0, 1\}$. For $\ell > 1$, each world of cardinality k either contains k facts from

$\{f_1, \ldots, f_{\ell-1}\}$, in which case we have the additional factor of removing the fact f_ℓ, or $k-1$ facts from $\{f_1, \ldots, f_{\ell-1}\}$ and the fact f_ℓ, which incurs no additional cost.

Now, we use the values $P[D'_i, k]$ to compute the value $P'[D', k]$, which is defined in the same way as $P[D'_i, k]$, except that we consider subsets of D'. Clearly, we have that:

$$Z(D') = \sum_{k=0}^{|D'|} P'[D', k].$$

As in the case of $P[D'_i, k]$, we use dynamic programming to compute the value $P'[D', k]$. Here, we denote by $P'[D', k, \ell]$ the value of the partition function when restricted to subsets E of D' such that $E \subseteq \{D'_1, \ldots, D'_\ell\}$ and $|E| = k$. Hence, $P'[D', k] = P'[D', k, m]$, and we focus on the computation of $P[D', k, \ell]$. We have that:

$$P'[D', k, \ell] = \begin{cases} P'[D'_\ell, k] & \text{if } \ell = 1; \\ \sum_{t=0}^{k} P'[D', t, \ell-1] \cdot P'[D'_\ell, k-t] \cdot (F_\varphi)^{t\cdot(k-t)} & \text{otherwise.} \end{cases}$$

That is, every subset E of cardinality k of $\{D'_1, \ldots, D'_\ell\}$ consists of a subset E_1 of cardinality t of $\{D'_1, \ldots, D'_{\ell-1}\}$ and a subset E_2 of cardinality $k-t$ of D_ℓ, for some $t \leq k$. The value $(F_\varphi)^{t\cdot(k-t)}$ is the total factor due to the violations among the facts of E_1 and the facts of E_2. This concludes our computation of the partition function for a single FD.

6.2.2 Marginal Inference. We show that for a single FD, we can compute the probability $\Pr_D[Q]$ in polynomial time for any Boolean query Q that tests for the existence of a tuple that meets a certain condition (e.g., a Boolean CQ with a single atom). More precisely, we consider queries Q that ask whether the database contains one or more facts from a given subset C of D. In particular, such a query can test the existence of an individual fact f (meaning that $C = \{f\}$ and then the goal is to compute the marginal probability of f) or be a condition about projected attributes (e.g., "grade=100"). We refer to such queries as *tuple properties*. We make the assumption that we can efficiently test whether a given fact satisfies a given tuple property. As we state in Section 7, we defer to future work the consideration of richer classes of queries.

The computation of the marginal inference is similar to that of the partition function. We again start by considering a single subblock D'_i, for which we define the following values:

$$M_Q[D'_i, k, 1] \stackrel{\text{def}}{=} \sum_{\substack{E \subseteq D'_i \\ |E|=k \\ E \models Q}} \mathcal{U}[E \mid D'_i]$$

which is the unnormalized probability of all the subsets of D'_i of size k that entail Q and

$$M_Q[D'_i, k, 0] \stackrel{\text{def}}{=} \sum_{\substack{E \subseteq D'_i \\ |E|=k \\ E \not\models Q}} \mathcal{U}[E \mid D'_i]$$

which is the unnormalized probability of all the subsets of D'_i of size k that do not entail Q. As in the case of $P[D'_i, k]$, we compute these values using dynamic programming over the facts $f_1, \ldots, f_r$ of D'_i. For that purpose, we define the values $M_Q[D'_i, k, \ell, b]$ (for $b \in \{0, 1\}$) that further restrict the subsets E that we consider to those that are contained in $\{f_1, \ldots, f_\ell\}$. Then, in the case where

$\{f_\ell\} \models Q$ (hence, every subset that contains f_ℓ entails Q), we have that:

$$M_Q[D'_i, k, \ell, 1] = \begin{cases} 0 & \text{if } \ell = 1 \text{ and } k \neq 1; \\ 1 & \text{if } \ell = 1 \text{ and } k = 1; \\ M_Q[D'_i, k, \ell-1, 1] \cdot F_{f_\ell} + M_Q[D'_i, k-1, \ell-1, 0] + & \\ M_Q[D'_i, k-1, \ell-1, 1] & \text{otherwise.} \end{cases}$$

$$M_Q[D'_i, k, \ell, 0] = \begin{cases} 0 & \text{if } \ell = 1 \text{ and } k \neq 0; \\ F_{f_1} & \text{if } \ell = 1 \text{ and } k = 0; \\ M_Q[D'_i, k, \ell-1, 0] \cdot F_{f_\ell} & \text{otherwise.} \end{cases}$$

In the case where $\ell = 1$ (i.e. when we consider subsets of $\{f_1\}$), if we keep f_1, then the query is entailed and there is no cost for removing a fact or violating an FD, while if we remove f_1, this contributes the factor F_{f_1} and the query is not entailed. In the case where $\ell \neq 1$, in order to entail the query, we either need to keep f_ℓ, and then we can choose any subset of size $k-1$ of $\{f_1, \ldots, f_{\ell-1}\}$ (which contributes the factor $M_Q[D'_i, k-1, \ell-1, 0] + M_Q[D'_i, k-1, \ell-1, 1]$), or we remove f_ℓ (with factor F_{f_ℓ}) and choose a subset that entails the query from $\{f_1, \ldots, f_{\ell-1}\}$ (with factor $M_Q[D'_i, k, \ell-1, 1]$). In order to not entail the query, we need to remove f_ℓ (contributing the factor F_{f_ℓ}) and choose a subset that does not entail the query from $\{f_1, \ldots, f_{\ell-1}\}$ (with factor $M_Q[D'_i, k, \ell-1, 0]$).

In the case where $\{f_\ell\} \not\models Q$ (where the inclusion or exclusion of f_ℓ has no impact on the entailment of Q) we have:

$$M_Q[D'_i, k, \ell, 1] = \begin{cases} 0 & \text{if } \ell = 1; \\ M_Q[D'_i, k, \ell-1, 1] \cdot F_{f_\ell} + M_Q[D'_i, k-1, \ell-1, 1] & \text{otherwise.} \end{cases}$$

$$M_Q[D'_i, k, \ell, 0] = \begin{cases} 0 & \text{if } \ell = 1 \text{ and } k \notin \{0, 1\}; \\ F_{f_1} & \text{if } \ell = 1 \text{ and } k = 0; \\ 1 & \text{if } \ell = 1 \text{ and } k = 1; \\ M_Q[D'_i, k, \ell-1, 0] \cdot F_{f_\ell} + M_Q[D'_i, k-1, \ell-1, 0] & \text{otherwise.} \end{cases}$$

When $\ell = 1$ (i.e. when we consider subsets of $\{f_1\}$), the query is not entailed regardless of our choice to keep or remove f_1, but the removal of this fact contributes the factor F_{f_1}. In the case where $\ell \neq 1$, in order to entail the query, we either need to keep f_ℓ, and then we need to choose a subset of size $k-1$ of $\{f_1, \ldots, f_{\ell-1}\}$ that entails Q (which contributes the factor $M_Q[D'_i, k-1, \ell-1, 1]$), or we remove f_ℓ (with factor F_{f_ℓ}) and choose a subset of size k that entails Q from $\{f_1, \ldots, f_{\ell-1}\}$ (with factor $M_Q[D'_i, k, \ell-1, 1]$). In order to not entail the query, we either need to keep f_ℓ, and then we need to choose a subset of size $k-1$ of $\{f_1, \ldots, f_{\ell-1}\}$ that does not entail Q (which contributes the factor $M_Q[D'_i, k-1, \ell-1, 0]$), or we remove f_ℓ (with factor F_{f_ℓ}) and choose a subset of size k that does not entail Q from $\{f_1, \ldots, f_{\ell-1}\}$ (with factor $M_Q[D'_i, k, \ell-1, 0]$).

Now, we use the values $M_Q[D'_i, k]$ to compute $M'_Q[D', k]$ that is defined in a similar way over the subsets of D'. We again use dynamic programming over the subblocks $D'_1, \ldots, D'_m$ of D', and compute the values $M'_Q[D', k, \ell]$ that are restricted to subsets contained in $D'_1 \cup \cdots \cup D'_\ell$. It holds that:

$$M'_Q[D', k, \ell, b] = \begin{cases} M_Q[D'_\ell, k, b] & \text{if } \ell = 1; \\ \sum_{b=1-[(1-b_1)\cdot(1-b_2)]}^{b_1, b_2} \sum_{t=0}^{k} M'_Q[D', t, \ell-1, b_1] \cdot & \\ M_Q[D'_\ell, k-t, b_2] \cdot (F_\varphi)^{t\cdot(k-t)} & \text{otherwise.} \end{cases}$$

Note that the only case when $1 - [(1-b_1)\cdot(1-b_2)] = 0$ is when $b_1 = b_2 = 0$.

At this point, we have that values $M'_Q[D', k, b]$ for $b \in \{0, 1\}$ for every block D' of D. Now, a subset E of D entails Q if and only if $(E \cap D') \models Q$ for at least one block D' of D. Hence, we now use dynamic programming over the blocks $D_1, \ldots, D_n$ of D to compute the values $M''_Q[D, k, b]$ (defined similarly to $M'_Q[D', k, b]$, but over the subsets of D). This computation is very similar to that of $M'_Q[D', k, b]$, with the main difference being that there are no violations among facts of different blocks (hence, we remove the factor $(F_\varphi)^{t\cdot(k-t)}$ from the second case). In particular, we have that:

$$M''_Q[D, k, \ell, b] = \begin{cases} M'_Q[D_\ell, k, b] & \text{if } \ell = 1; \\ \sum_{b=1-[(1-b_1)\cdot(1-b_2)]}^{b_1,b_2} \sum_{t=0}^{k} M''_Q[D, t, \ell-1, b_1] \cdot M'_Q[D_\ell, k-t, b_2] & \text{otherwise.} \end{cases}$$

and $M''_Q[D, k, b] = M''_Q[D, k, n, b]$.

Finally, we are only interested in the values $M''_Q[D, k, 1]$ (as we compute that probability of entailing Q) and we need to normalize them using the partition function $Z(D)$ that we computed in the previous section. Formally,

$$\Pr_D[Q] = \frac{1}{Z(D)} \sum_{k=1}^{|D|} M''_Q[D, k, 1]$$

and that concludes our computation of the marginal inference. We establish the following.

Theorem 6.6. *In the case of a single FD, marginal inference of tuple properties is solvable in polynomial time.*

6.2.3 Sampling. We continue with the case that Δ consists of a single FD φ and show that, given a database D, we can sample a random subset in polynomial time.

As before, since there are no conflicts across blocks, we can handle one block at a time, and we focus on sampling a subset of a single block D'. The general idea is to iterate over the facts of D' in some predefined (arbitrary) order and select, for each fact, whether or not to include it in the sample. Let E be a subset of D', and let f_j be the jth fact of D' in our order. We denote by $M_{E,j}$ the event that denotes the membership of f_j in a sample precisely as in E; that is, if $f_j \in E$ then $M_{E,j}$ is the event that f_j is in the sample, and otherwise, $M_{E,j}$ is the event that f_j is *not* in the sample. Then, the probability that E is obtained through the sampling procedure is as follows.

$$\Pr_{D'}[E] = \Pr_{D'}[M_{E,1}, \ldots, M_{E,|D'|}] = \prod_{j=1}^{|D'|} \Pr_{D'}[M_{E,j} \mid M_{E,1}, \ldots, M_{E,j-1}] \tag{4}$$

Therefore, we can sample one fact at a time, where at each step the probability of taking the next fact depends on the choices that were made before. If the previous choices are $B_1, \ldots, B_{j-1}$, where B_k is some decision regarding the kth fact of D' (either keeping or removing it), then we take the next fact with probability $\Pr_{D'}[C_j \mid B_1, \ldots, B_{j-1}]$, where C_j is the event of taking the jth fact of D'. Given a database D and an event X, denote by $\mathrm{W}_D(X)$ the sum of the weights of all subsets of D where X holds. That is,

$$\mathrm{W}_D(X) \stackrel{\text{def}}{=} \sum_{\substack{E \subseteq D \\ E \models X}} \mathcal{U}[E \mid D]\,.$$

Therefore, we randomly select f_j in each step, with the following probability:

$$\Pr_{D'}[C_j \mid B_1, \ldots, B_{j-1}] = \frac{\Pr_{D'}[B_1, \ldots, B_{j-1}, C_j]}{\Pr_{D'}[B_1, \ldots, B_{j-1}]} = \frac{\frac{\mathrm{W}_{D'}(B_1,\ldots,B_{j-1},C_j)}{Z(D')}}{\frac{\mathrm{W}_{D'}(B_1,\ldots,B_{j-1})}{Z(D')}} = \frac{\mathrm{W}_{D'}(B_1, \ldots, B_{j-1}, C_j)}{\mathrm{W}_{D'}(B_1, \ldots, B_{j-1})}$$

From our discussion so far we conclude that, in order to sample, it is enough to compute values of the form $\mathrm{W}_{D'}(B_i, \ldots, B_j)$ where B_k is either the event that the kth fact is taken or the event that this fact is not taken. This can easily be done using the approach from Section 6.2.1. As every summand in the formula for the partition function matches either the choice of taking a fact or not taking it, we simply use the same formula while replacing by zero the summands that correspond to the choices opposite to what we wish to compute.

The correctness follows from Equation (4). Let E be a possible sampled outcome. Since this subset is chosen through a series of decisions with the probabilities given by the equation, it will be selected with probability $\Pr_{D'}[E]$, as required. Consequently, we establish the following.

Theorem 6.7. *In the case of a single FD, sampling can be done in polynomial time.*

6.3 Generalization Beyond a Single FD

Similarly to soft repairing, the tractability results that we established in the previous section can be generalized to FD sets Δ that can be emptied by repeatedly applying L/C-Simplify(Δ) depicted in Algorithm 2. In this section, we show how the algorithms can be extended to this generalization.

We first show how to compute the partition function. Let Δ be an FD set that is emptied by L/C-Simplify. Consider the input relation D. Our goal is to compute the value $Z(D)$. For convenience, we will make the assumption that *all* attributes of D appear in Δ. (This assumption simplifies the base case of our dynamic programming, as we explain next.) Observe that we can make this assumption since, otherwise, we can always add to Δ the FD $\gamma = \emptyset \rightarrow A_1 \ldots A_n$ with the factor $F_\gamma = 1$ (hence, have no impact on the semantics), where $\{A_1, \ldots, A_n\}$ is the set of all attributes of D. Clearly, if Δ can be emptied by L/C-Simplify before the addition of γ, then it can be likewise emptied after the addition of γ.

Similarly to what we have done in Section 4.2, the idea is that we repeatedly partition the database according to each attribute A that L/C-Simplify(Δ) eliminates. In other words, we apply *group-by* A for every such A, and continue grouping each partition recursively according to the future attributes that L/C-Simplify(Δ) eliminates. We refer to each partition that we get as a *group*. Suppose that $A_1, \ldots, A_n$ are the attributes in the order of their elimination by the algorithm. Hence, each group is defined by an assignment of database values to $A_1, \ldots, A_i$ for some $i = 0, \ldots, n$. We can view the process as defining a tree over the groups, where:

- The root is the full database D;
- The children of each group D', which is defined by an assignment to $A_1, \ldots, A_i$, are the pairwise-disjoint groups that form a partition of D' by extending the assignment to A_{i+1}.

See Figure 7 for an illustration. Note that the tree may have duplicates, that is, situations where D' is equal to its single child D_1, if it happens to be the case that all tuples of D' agree on A_i. Also note that the number of vertices of this tree is polynomial in the size of D, since the depth is the number n of attributes and each level has at most $|D|$ vertices (since the level forms a partition of D).

For each group D' defined by an assignment to the attributes $A_1, \ldots, A_i$ we compute the values $Q[D', k]$ that is the contribution to $Z(D')$ from all of the subsets of D' of cardinality k. That is:

$$P[D', k] \stackrel{\text{def}}{=} \sum_{\substack{E \subseteq D' \\ |E|=k}} \mathcal{U}[E \mid D'] \,.$$

This is enough, since then we have

$$Z(D) = \sum_{k=0}^{|D|} P[D, k] \,.$$

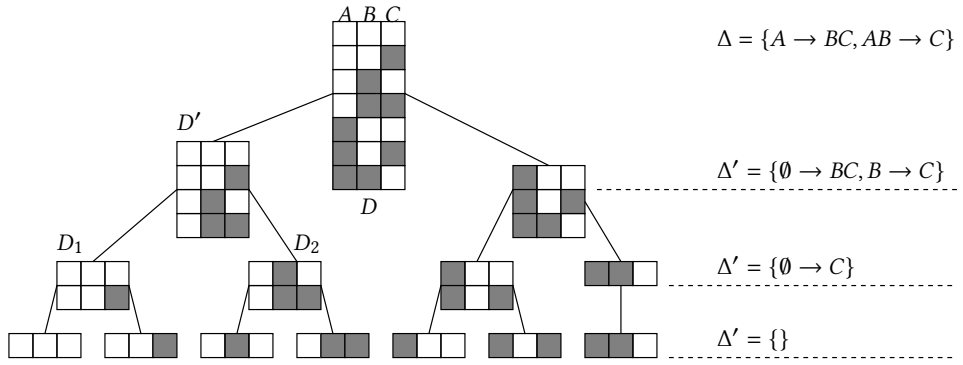


Fig. 7. An illustration of the tree implied by the elimination order A, B, C via L/C-Simplify(Δ). Here, $A_1 = A$, $A_2 = B$ and $A_3 = C$. The FD set $\Delta' = \Delta - A_1 - \cdots - A_i$ is shown for every level i.

We will compute the $P[D', k]$ in a bottom-up fashion. Let us begin with the leaves. In each leaf D' we have a single fact f (hence, the FDs cannot be violated). This is true since $A_1, \ldots, A_n$ include all attributes of the database, due to the assumption that we made at the beginning of this section. Therefore, we get that

$$P[D', k] = \begin{cases} F_f & \text{if } k = 0; \\ 1 & \text{if } k = 1; \\ 0 & \text{otherwise.} \end{cases}$$

corresponding to the two worlds $\emptyset$ (having the unnormalized probability F_f) and $\{f\}$ (having the unnormalized probability 1).

Next, we consider an internal group D' that is defined by an assignment to the attributes $A_1, \ldots, A_i$. By the definition of D', all of the tuples agree on all of the attributes $A_1, \ldots, A_i$, but not necessarily on $A_{i+1}, \ldots, A_n$. Let $D_1, \ldots, D_m$ be the children of D' that are obtained by grouping D' by A_{i+1}. (See Figure 7 for an illustration.) We break $P[D', k]$ into values $P[D', k, \ell]$ that restricts $P[D', k]$ to the union $D_1 \cup \cdots \cup D_\ell$:

$$P[D', k, \ell] \stackrel{\text{def}}{=} \sum_{\substack{E \subseteq D_1 \cup \cdots \cup D_\ell \\ |E| = k}} \mathcal{U}[E \mid D_1 \cup \cdots \cup D_\ell]$$

Hence, $P[D', k] = P[D', k, m]$.

It is then left to compute $P[D', k, \ell]$. Let $\Delta' = \Delta - A_1 - \cdots - A_i$, that is, the FD set obtained by eliminating $A_1, \ldots, A_i$ (as done by applying L/C-Simplify(Δ) repeatedly), and let $A = A_{i+1}$. Observe that the attribute A occurs in every nontrivial FD of Δ', either on the right-hand side of a consensus FD or on the left-hand side of another FD. Let F_A be product of the factors of all FDs in Δ that become nontrivial consensus FDs in Δ'. (Note that distinct FDs in Δ may become the same FD in Δ' due to the elimination of $A_1 - \cdots - A_i$. See, e.g., $\emptyset \to BC$ and $B \to C$ in Figure 7.) Then $P[D', k, \ell]$ is the sum of the contributions of unions $E \cup E'$ of subsets E of $D_1, \ldots, D_{\ell-1}$ and E' of D_ℓ such that $|E| = t$ and $|E'| = k - t$ for $t = 0, \ldots, k$. The contribution of such $E \cup E'$ is the product of three factors:

- The unnormalized probability of E, summed up to $P[D', t, \ell - 1]$ that we computed in the previous iteration;
- The unnormalized probability E', summed up to $P[D_\ell, k - t]$ that we computed in a previous step of the bottom-up evaluation;

- The factor contributed by violating the consensus FDs of Δ' (all of which contain A), that is, $(F_A)^{t\cdot(k-t)}$.

Note that no cost is incurred by the non-consensus FDs, since there are no violations between D_is as they disagree on the lhs attribute A.[4] Hence, we get the following:

$$P[D', k, \ell] = \begin{cases} P[D_1, k] & \text{if } \ell = 1; \\ \sum_{t=0}^{k} P[D', t, \ell - 1] \cdot P[D_\ell, k - t] \cdot (F_A)^{t\cdot(k-t)} & \text{otherwise.} \end{cases}$$

This concludes the computation of the partition function. For the marginal inference of a Boolean condition over tuples, we apply a similar extension to the algorithm of Section 6.2.2. Moreover, the sampling algorithm of Section 6.2.3 generalizes immediately as it only requires the computation of marginal probabilities and is otherwise unaware of the actual FDs. We conclude the following.

Theorem 6.8. *Let Δ be a set of FDs. If Δ can be emptied via L/C-Simplify(Δ) steps, then the following can be performed in polynomial time: (a) computation of the partition function, (b) computation of a marginal probability for a tuple condition, and (c) sampling.*

We conclude that, similarly to soft repairing, the probabilistic challenges are tractable for the case of a single FD and its generalization. In the next section, we show that this is *not* the case for matching constraints.

6.4 Hardness for Matching Constraints

In this section, we focus on matching constraints. We show that while soft repairing can be solved in polynomial time for such constraints, this is no longer the case when considering other probabilistic challenges. In particular, we show the following.

Theorem 6.9. *Computing the partition function is #P-hard for $R(A, B)$ and $\Delta = \{A \to B, B \to A\}$.*

Proof. Consider the FD set $\{A \to B, B \to A\}$ over $R(A, B)$. We show that computing the partition function is #P-hard via a reduction from the problem of computing the number of matchings in a bipartite graph, known to be #P-complete [36]. The idea is the following. We add to the database D a fact $R(u, v)$ for every edge (u, v) in the bipartite graph G. We associate a high weight with each of the FDs, so that the contribution to the partition function of subsets that do not correspond to matchings (i.e., subset that violate the constraints) is very low. Then, we define $F_f = 1$ for every fact f of the database, so that the contribution to the partition function of each subset that corresponds to a matching is 1. More formally, we define:

$$F_{A\to B} = F_{B\to A} = 2^{-|D|}$$

Then, the value of the partition function is:

$$Z(D) = 2^{-|D|\cdot k_1} + \cdots + 2^{-|D|\cdot k_n} + m$$

where n is the number of subsets $E_1, \ldots, E_n$ of D that violate the FDs and m is the number of subsets of D that satisfy the FDs. For each subset E_i, we denote by k_i the number of violations of the FDs in E_i (i.e., the number of fact pairs that jointly violate an FD). Note that m is the number of matchings in G; hence, our goal is to obtain this value, given $Z(D)$.

We have the following:

$$2^{-|D|\cdot k_1} + \cdots + 2^{-|D|\cdot k_n} \leq n \cdot 2^{-|D|}$$

[4]Note that the algorithm does not ignore the non-consensus FDs of Δ'—each of them becomes a consensus FD in deeper parts of the tree, as also illustrated in Figure 7.

Now, $2^{|D|}$ is the total number of subsets of D. Clearly, the empty subset satisfies the constraints; hence, we have that $n < 2^{|D|}$ and we conclude that:

$$2^{-|D|\cdot k_1} + \cdots + 2^{-|D|\cdot k_n} < 1$$

Therefore, in order to obtain m from the partition function it is sufficient to take the integer part of $Z(D)$. □

From the hardness of the partition function we can also conclude the hardness of marginal inference, even for the simplest tuple property.

Theorem 6.10. *For $R(A, B)$ and $\Delta = \{A \to B, B \to A\}$, it is #P-hard to compute the probability that the database is nonempty (i.e., marginal inference for the tautology tuple property).*

Proof. We show a reduction from the problem of computing the partition function, which is #P-hard due to Theorem 6.9. Let D be a given database over $R(A, B)$, and let Q_t be the tautology query. The non-normalized probability that D is empty is given by $\prod_{f \in D} F_f$. Hence, the probability of Q_t (i.e., that a random world is nonempty) is given by:

$$\Pr_D[Q_t] = 1 - \left(\prod_{f \in D} F_f\right) / Z(D)$$

Hence, from $\Pr_D[Q_t]$ we can compute the partition function:

$$Z(D) = \frac{\prod_{f \in D} F_f}{1 - \Pr_D[Q_t]}$$

This concludes the proof. □

The complexity of sampling is left open for future research. In particular, we do not know whether Theorem 6.9 is also true for the sampling problem.

7 CONCLUSIONS AND OPEN PROBLEMS

We studied the complexity of soft repairing for functional dependencies, where the goal is to find an optimal subset under penalties of deletion and constraint violation. The problem is harder than that of computing a cardinality repair, and we have developed two new, nontrivial algorithms solving natural special cases. A full classification of the FD sets remains an open challenge for future research; specifically, the question is what fragment of the positive side of the dichotomy of Livshits et al. [25] remains positive when softness is allowed. We have also shown that the problem becomes tractable if we settle for a 3-approximation.

Open Problems. Several directions are left open for future work. A direct open problem is to characterize the class of tractable FDs via a full dichotomy. The simplest sets of FDs where the complexity of soft repairing is open are the following:

- $\{A \to B, A \to C\}$. As discussed in Section 4.3, this problem is different from $\{A \to BC\}$ that consists of a single FD, even though the two sets are logically equivalent when viewed as in the traditional sense as hard constraints.
- $\{A \to B, B \to A\}$ in the case where the schema has attributes different from A and B, starting with $R(A, B, C)$.
- $\{\emptyset \to A, B \to C\}$.

The problem is also open for classes of constraints that are more general than FDs, including equality-generating dependencies (EGDs), denial constraints, and inclusion dependencies. Yet, the problem for these types of dependencies is open already in the case of cardinality repairs, with the

exception of some cases of EGDs [24]. Another clear direction is that of *update repairs* where we are allowed to change cell values instead of (or in addition to) deleting tuples and where complexity results are known for hard constraints [21, 25].

We also discussed some fundamental tasks that apply to every probabilistic model. In our case, we adopted the convention that the soft constraints are interpreted as the templates for the factors of a factor-graph representation of the probabilistic database. In terms of our contributed results, we only touched the tip of the iceberg and there is much more to be done, as many questions are then left open for future research:

- Handling more general FD sets and constraints beyond FDs;
- Allowing approximate solutions for the tasks, with error guarantees;
- Proving lower bounds for sampling (and for the other challenges regarding the FD sets that we have not considered);
- Evaluating cross-relation queries such as joins, conjunctive queries, and beyond.

This work can be naturally generalized to the goal of constructing, in addition to the database repair, a revised set Δ of *constraints* that better reflects the current data (since, e.g., some FDs became obsolete or outdated). This challenge has been studied in the past under the restriction that the repair is consistent with respect to the revised constraints [5, 7], whereas our optimization goal may allow for a balance between the change in the data, the revision of the constraints, and the level of violation.

Finally, we remark that all of the algorithms presented in this manuscript have been designed as proofs of tractability. We believe that there is much to be done in order to transform them into practical solvers of realistic problems, and this is also an important direction for future research.

ACKNOWLEDGMENTS

This work has been funded by the Israel Science Foundation (ISF) under grant 768/19, and the German Research Foundation (DFG) under grants GR 1492/16-1 and KI 2348/1-1.

REFERENCES

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows - theory, algorithms and applications*. Prentice Hall, 1993.
[2] A. Arasu, C. Ré, and D. Suciu. Large-scale deduplication with constraints using dedupalog. In *ICDE*, pages 952–963. IEEE Computer Society, 2009.
[3] A. authors. Anatomized title.
[4] V. Bárány, B. ten Cate, B. Kimelfeld, D. Olteanu, and Z. Vagena. Declarative probabilistic programming with datalog. *ACM Trans. Database Syst.*, 42(4):22:1–22:35, 2017.
[5] G. Beskales, I. F. Ilyas, L. Golab, and A. Galiullin. On the relative trust between inconsistent data and inaccurate constraints. In *ICDE*, pages 541–552. IEEE Computer Society, 2013.
[6] P. Blunsom and M. Osborne. Probabilistic inference for machine translation. In *EMNLP*, pages 215–223. ACL, 2008.
[7] F. Chiang and R. J. Miller. A unified model for data and constraint repair. In *ICDE*, pages 446–457. IEEE Computer Society, 2011.
[8] X. Chu, I. F. Ilyas, and P. Papotti. Discovering denial constraints. *PVLDB*, 6(13):1498–1509, 2013.
[9] C. Combi, M. Mantovani, A. Sabaini, P. Sala, F. Amaddeo, U. Moretti, and G. Pozzi. Mining approximate temporal functional dependencies with pure temporal grouping in clinical databases. *Comp. in Bio. and Med.*, 62:306–324, 2015.
[10] N. N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *VLDB J.*, 16(4):523–544, 2007.
[11] N. N. Dalvi and D. Suciu. The dichotomy of probabilistic inference for unions of conjunctive queries. *J. ACM*, 59(6):30:1–30:87, 2012.
[12] S. Ermon, C. P. Gomes, A. Sabharwal, and B. Selman. Accelerated adaptive markov chain for partition function computation. In *NIPS*, pages 2744–2752, 2011.
[13] C. Ge, S. Mohapatra, X. He, and I. F. Ilyas. Kamino: Constraint-aware differentially private data synthesis. *Proc. VLDB Endow.*, 14(10):1886–1899, 2021.
[14] A. V. Goldberg and R. E. Tarjan. Finding minimum-cost circulations by successive approximation. *Math. Oper. Res.*, 15(3):430–466, Aug. 1990.

[15] T. F. Gonzalez, editor. *Handbook of Approximation Algorithms and Metaheuristics*. Chapman and Hall/CRC, 2007.
[16] E. Gribkoff, G. V. den Broeck, and D. Suciu. The most probable database problem. In *BUDA*, 2014.
[17] A. Heidari, J. McGrath, I. F. Ilyas, and T. Rekatsinas. Holodetect: Few-shot learning for error detection. In *SIGMOD Conference*, pages 829–846. ACM, 2019.
[18] D. S. Hochbaum. Approximation algorithms for the set covering and vertex cover problems. *SIAM Journal on computing*, 11(3):555–556, 1982.
[19] Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen. TANE: an efficient algorithm for discovering functional and approximate dependencies. *Comput. J.*, 42(2):100–111, 1999.
[20] A. K. Jha, V. Rastogi, and D. Suciu. Query evaluation with soft-key constraints. In *PODS*, pages 119–128, 2008.
[21] S. Kolahi and L. V. S. Lakshmanan. On approximating optimum repairs for functional dependency violations. In *ICDT*, volume 361 of *ACM International Conference Proceeding Series*, pages 53–62. ACM, 2009.
[22] S. Kruse and F. Naumann. Efficient discovery of approximate dependencies. *Proc. VLDB Endow.*, 11(7):759–772, 2018.
[23] E. Livshits, A. Heidari, I. F. Ilyas, and B. Kimelfeld. Approximate denial constraints. *Proc. VLDB Endow.*, 13(10):1682–1695, 2020.
[24] E. Livshits, I. F. Ilyas, B. Kimelfeld, and S. Roy. Principles of progress indicators for database repairing. *CoRR*, abs/1904.06492, 2019.
[25] E. Livshits, B. Kimelfeld, and S. Roy. Computing optimal repairs for functional dependencies. *ACM Trans. Database Syst.*, 45(1):4:1–4:46, 2020.
[26] A. Lopatenko and L. E. Bertossi. Complexity of consistent query answering in databases under cardinality-based and incremental repair semantics. In *ICDT*, volume 4353 of *Lecture Notes in Computer Science*, pages 179–193. Springer, 2007.
[27] D. Lowd and P. M. Domingos. Efficient weight learning for markov logic networks. In *PKDD*, volume 4702 of *Lecture Notes in Computer Science*, pages 200–211. Springer, 2007.
[28] D. Miao, Z. Cai, J. Li, X. Gao, and X. Liu. The computation of optimal subset repairs. *Proc. VLDB Endow.*, 13(11):2061–2074, 2020.
[29] E. H. M. Pena, E. C. de Almeida, and F. Naumann. Discovery of approximate (and exact) denial constraints. *Proc. VLDB Endow.*, 13(3):266–278, 2019.
[30] F. Pennerath, P. Mandros, and J. Vreeken. Discovering approximate functional dependencies using smoothed mutual information. In *KDD*, pages 1254–1264. ACM, 2020.
[31] T. Rekatsinas, X. Chu, I. F. Ilyas, and C. Ré. Holoclean: Holistic data repairs with probabilistic inference. *Proc. VLDB Endow.*, 10(11):1190–1201, 2017.
[32] M. Richardson and P. Domingos. Markov logic networks. *Mach. Learn.*, 62(1-2):107–136, Feb. 2006.
[33] C. D. Sa, I. F. Ilyas, B. Kimelfeld, C. Ré, and T. Rekatsinas. A formal framework for probabilistic unclean databases. In *ICDT*, volume 127 of *LIPIcs*, pages 6:1–6:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.
[34] P. Sen, A. Deshpande, and L. Getoor. PrDB: managing and exploiting rich correlations in probabilistic databases. *VLDB J.*, 18(5):1065–1090, 2009.
[35] D. Suciu. Probabilistic databases for all. In *PODS*, pages 19–31. ACM, 2020.
[36] L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979.
[37] C. Zhang, C. Ré, M. J. Cafarella, J. Shin, F. Wang, and S. Wu. Deepdive: declarative knowledge base construction. *Commun. ACM*, 60(5):93–102, 2017.